Making a Case for Flexible 802.11 Architectures

Pablo Salvador*†, Francesco Gringoli‡, Pablo Serrano†, Nicolò Facchi‡, Stefano Paris§
*Institute IMDEA Networks, Madrid, Spain, Email: josepablo.salvador@imdea.org
†University Carlos III of Madrid, Spain, Email: pablo@it.uc3m.es
‡CNIT / University of Brescia, Brescia, Italy
Email: francesco.gringoli,nicolo.facchi@ing.unibs.it
§Mathematical and Algorithmic Sciences Lab, France Research Center, Huawei Technologies Co. Ltd.
Email: stefano.paris@huawei.com

***Abstract*—In the past years, researchers have been advocating for flexible 802.11 devices that dynamically adapt to the varying network conditions, looking for efficient alternatives to the 802.11 standard MAC. In this work we demonstrate that this flexibility is readily available at the MAC level, and its operation can be tuned by re-programming the firmware inside the wireless chipsets that are built on relatively generic hardware modules. We show this by implementing the new amendment IEEE 802.11aa in legacy equipments by simply coding the frame exchange schemes at the firmware level. Nevertheless, we claim that the lack of flexibility in the way these modules interact results in a bottleneck that severely degrades performance. In our work, we prove this inefficiency of the 802.11 hardware architecture that hinders high throughput features, as in our case study of 802.11aa reliable multicast. To solve this problem, we provide new directions for the revision of the current hardware architecture and propose a new vision for the future design of wireless chipsets.**

***Keywords*-Standard; 802.11; WLAN; 802.11aa; Multicast; Implementation; NIC; Commodity hardware**

I. Introduction

According to recent works [1]–[4], the *monolithic* model for network interfaces successfully adopted in the wired domain (i.e., the one-standard-many-manufacturers paradigm), will not fulfill the future requirements of the unlicensed wireless universe. Spectrum resources are, in fact, finite, and accommodating in the same set of frequencies many wireless technologies, each with its own channel access algorithm, and new devices running increasingly bandwidth-hungry applications, results a very tough task. While in the wired case the current switching technology supports almost infinite-bandwidth and sharply-isolated communication channels, this is not the case in the wireless domain: indeed, the operating conditions of current wireless networks (like the number of overlapping data-link segments, or the presence of competing technologies) cannot be predicted by any standard. For these reasons, researchers are advocating for a greater degree of flexibility in the architecture of wireless interfaces.

In the last decade, however, manufacturers kept following the historical design model, releasing new Wi-Fi[1] radio features as they were amended, or only drafted.[2] No attempts have been made to introduce chipset openness or flexibility, apart from tuning standard-specified parameters, e.g., the values of the Contention Windows (CW).[3] On the contrary, the few attempts to improve the MAC resulted in incompatible features, like e.g. the Atheros *Turbo* mode.

Given the above, claiming that Wi-Fi Network Interface Cards (NICs) are already somehow flexible could result shocking. Nonetheless, as Wi-Fi NICs have to support the relatively complex operations required by the CSMA/CA channel access, they tend to embed more flexible and reconfigurable logic than their wired counterparts, which focus on increasing the dequeuing speed by removing communication bottlenecks with the main host [5], [6]. In fact, the prevalent channel access of 802.11 (the Distributed Coordination Function (DCF)) or its variants are run in the firmware of these chipsets, to support actions like: managing each frame type in a specific way, adjusting the data-rate on the fly or handling virtual internal collisions. However, no details on this flexibility have been reported: firmwares are closed-source, and the few research groups that have used non-standard functionality are typically provided with custom vendors' firmwares [7], [8], which illustrates how manufactures do not disclose this information.

To the best of our knowledge, the sole exception to the above "secrecy" is the academic OpenFWWF project,[4] which released the first open-source firmware compatible with a few ERP-OFDM-PHY NICs from Broadcom. The firmware can be used to implement, e.g., MAC enhancements [9], [10] or reactive jamming [11], demonstrating the actual NIC flexibility. Recently, we added to the firmware a set of mechanisms from the novel 802.11aa amendment [12], which enhances the robustness of multimedia traffic by means of different MAC solutions. Our implementation further illustrates the ability of existing NICs to support types of frame exchanges introduced by new standard documents. Consequently, in this work we show that the true limitation for deploying (these) new features in existing NICs is not the lack of flexibility at the layer two, but a bottleneck imposed by the Direct Memory Access (DMA) subsystem when transferring frames from the host CPU to the NIC.

[1]Note that we use the terms Wi-Fi and IEEE 802.11 interchangeably.

[2]As in the case of 802.11 HT-PHY, many pre-N wireless chipsets were shipped with laptops that we still use today, like the Atheros AR9285 or AR5B93 and the Broadcom 43224 and 43225.

[3]See Intel: http://www.intel.com/support/wireless/wlan/sb/CS-034398.htm; and Atheros: http://wireless.kernel.org/en/users/Drivers/Atheros.

[4]http://www.ing.unibs.it/openfwwf/

More specifically, differently from previous works [1], [2], [13], [14], in this paper we show that having a large flexibility at layer two is a *necessary but not sufficient* condition to make a NIC ready for future protocol implementations. We claim that the entire chain between the main host running the Operating System (OS) and the NIC must be redesigned, to support not only "channel access algorithms" but also complex "data elaboration algorithms" including, among others, policing outgoing frames, forwarding received frames (independently of the host kernel) and so on.

The rest of the paper is organized as follows. We first provide a Related Work section on how the 802.11 world has approached flexibility so far. Then, we give in Section III an overview of the main MAC changes introduced by subsequent 802.11 amendments focusing on the time constraints. We show how current chipsets support the evolved mechanisms without major alterations at the MAC CPU level (although we focus on a specific Broadcom chipset, most of our conclusions apply to other manufacturers – Atheros, Intel or Ralink – as they share the same architecture). We then describe in Section IV the implementation of the set of MAC mechanisms recently standardized in 802.11aa amendment known as Group Addressed Transmission Services (GATS) [15]. Building upon the knowledge from this implementation, we give some insights on the lessons learned and identify the main factors that limit the performance of current platforms (Section V). Leveraging on these results, we propose and discuss in Section VI an evolved architectural paradigm that allows for a higher degree of flexibility and enhance the performance of the standards that will likely come. Finally, Section VII closes the paper with some final remarks.

II. Related work

The seminal work of [16] was among the first to underline how commercial NICs could be turned into flexible research platforms, by replacing the stock firmware with a different one offering custom transmission/reception paths. However, even if this work revealed the flexibility of the PRISM 11b chipset at the MAC layer with its MAC CPU, it did not disclose how to modify the firmware, which remained closed-source. Since then, and with the end of the PRISM project,[5] many works selected Atheros-based NICs as the next research platform of choice, thanks to the active collaboration of the manufacturer, excellent open-source drivers (e.g.: Mad-Wifi) and a lot of documentation. Apart from works targeting specific features, [17] and [18] provided software frameworks for the easy customization of some functionalities of the Atheros AR5212 chipset, such as disabling packet acknowledgment, fast channel switching and modifying the number of retries for failed frames.

None of the above supported ad-hoc frame exchanges with the required timing, e.g., receiving a frame and forging the corresponding reply within a few microseconds. Because of this, a number of projects designed 802.11-compliant implementations from scratch, many of them building around an FPGA: e.g., SMiLE [19], used to test novel localization schemes; or Sora [13] and WARP [20], the former being a very powerful Software Defined Radio (SDR) device that connects to the computer that handles frame-symbol transformation through a mini-PCI interface, and the latter being a standalone device that executes upper and lower MAC functions in two different soft-cores. A platform really similar to a real Wi-Fi NIC, i.e., with a MAC CPU synthesised on a FPGA executing the MAC firmware and a mini-PCI interface for connecting to the main host, was introduced in [21]: unfortunately, the project did not evolve further from the 802.11b stage.

Differently than all previous approaches, the architecture proposed in the European project FLAVIA[6] decoupled the logic that drives an access protocol from the hardware implementation. Authors demonstrated this vision with the Wireless MAC Processor on top of inexpensive consumer NICs from Broadcom, showing the possibility to re-program the MAC behaviour in real-time with a graphical oriented programming language [3], [14].

To the best of our knowledge, no previous work has identified the need to bring flexibility also at the *frame interface* between the host and the wireless NIC. This could look rather surprising, given that this sort of flexibility was introduced in the last few years to enable a faster bottleneck-free communication between the host and some wired interfaces like with the Receive-Side Scaling technique used by Intel on 10Gb Ethernet NIC [5], [6].

III. The Operation of Existing 802.11 Platforms

Here we describe how despite the MAC enhancements introduced by subsequent 802.11 amendments, the operation of 802.11 has not radically changed over the years. We conjecture one of the key reasons for this is backwards compatibility, to ensure that legacy and novel devices can coexist in the same WLAN. But with the introduction of the recent 802.11aa standard the situation is notably different, which challenges the use of existing 802.11 architectures as we detail in Section IV.

A. The evolution of the 802.11 MAC over the amendments

The widely used (and now, legacy) DCF involves a CSMA/CA access scheme that retransmits MAC Protocol Data Units (MPDUs) through a stop-and-wait mechanism. More specifically,[7] when an MPDU is ready for transmission, the hardware sets a Backoff Counter (BC) to a uniformly distributed value limited by the Contention Window to $CW-1$. Then, it waits for the channel to be idle for a Distributed Inter Frame Space time ($DIFS$), to start decrementing the BC at every *time slot*, eventually transmitting the MPDU when BC reaches zero. If the destination successfully acknowledges the MPDU by sending an ACK frame after a Short IFS

[5]The maintenance of the PRISM project ended up in 2007.

[6]FP7 ICT FLAVIA project,"FLexible Architecture for Virtualizable future wireless Internet Access" http://www.ict-flavia.eu

[7]For space reasons and to improve readability we omit some particularities of the operation of the MAC protocol

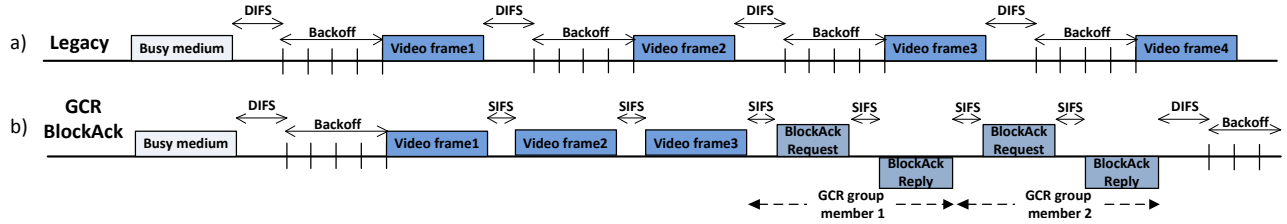

Fig. 1: Comparison of legacy and GCR Block ACK multicast transmission in an 802.11aa WLAN.

($SIFS$), then the CW is reset to the minimum CW_{min}, otherwise it is doubled (until a maximum CW_{max} is reached). Interoperability between NICs from different vendors requires full respect of time constants such as DIFS, SIFS and slot time durations, which depend on the particular PHY specification.

The 802.11e amendment introduces more complexity with the Enhanced Distributed Channel Access (EDCA) mechanism. EDCA provides concurrent channel access via four independent queues of different access priorities, each one with a specific BC and transmission parameters $DIFS$, CW_{min} and CW_{max}, which can be considered as four instantiations of the DCF procedure. In addition, 802.11e introduces the Transmission Opportunity ($TXOP$) parameter (also per-queue), which supports that two stations can exchange a number of Data-ACK pairs with just a $SIFS$ time between each frame, for up to $TXOP$ units of time. Along the same lines, the amendment also introduces an optional unicast channel access policy, namely, the Block Acknowledgment (BA) for reducing the channel access time overheads. With BA, instead of continuous Data-ACK exchanges, a transmitter sends a burst of data frames, separated by a $SIFS$ interval, addressed to the same destination. Then, the transmitter polls the receiver with a BA Request (BAREQ) to trigger the transmission of the BA Reply (BAREP) frame, which contains the indication of the success/failure frames embedded into a bitmap. The 802.11n amendment does not only make this functionality mandatory for High Throughput NICs (HT-PHY), but it also introduces the HT-ACK feature for the acknowledgment of Aggregated MPDU (A-MPDU):[8] the destination, upon receiving the A-MPDU, may transmit the BA right after a SIFS, and no BAREQ is required from the sender.

Finally, the recent 802.11aa amendment presents some fresh novelties to the *multicast* delivery service, which has not undergone any change since its very first definition in 1997. A novel set of mechanisms, called Group Addressed Transmission Service (GATS), provides different trade-offs between reliability and complexity for video streaming [22], building on legacy access primitives. We overview GATS in the next subsection and describe current hardware implementations in Section III-C.

B. The 802.11aa Group Addressed Transmission Service

A key novelty of 802.11aa is the specification of the Group Addressed Transmission Service (GATS). With the default multicast scheme, which we illustrate in Fig. 1a (*legacy*), video frames are transmitted only once, using a rate from the Basic Rate Set (BRS)[9], and never acknowledged. Because rates from the BRS are normally low, the *performance anomaly* [23] reduces the overall throughput of the whole WLAN, while the lack of feedback harms reliability. With GATS, frames are addressed to a group of stations listening to the same (non-multicast) address, namely, the groupcast concealment address, using one of the following mechanisms: (i) Directed Multicast (DMS); (ii) GroupCast with Retries (GCR) Unsolicited Retry (UR), and (iii) GCR Block Acknowledgment (BA). We focus on the latter, as the first two mechanisms do not expose the problem we address in this paper. Interested reader is referred to [12] for further details.

BA mechanism extends the standard Block Acknowledgment operation to support multiple destinations. The access point (AP) sends up to GCR-buffer-size consecutive frames in a burst, and then polls each intended destination subscribed to the groupcast address to receive their acknowledgments. Based on the feedback, the transmitter drops the frames successfully received by all stations, and retransmits the others. The operation of BA is illustrated in Fig. 1b.

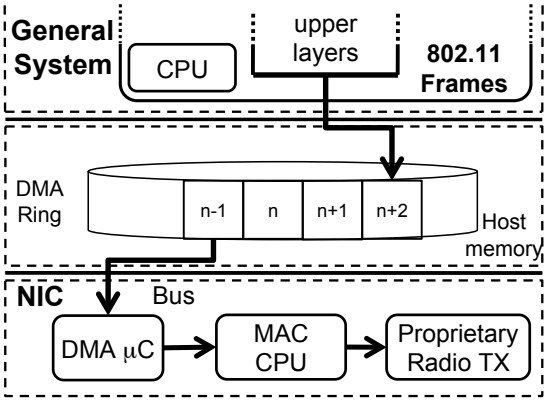

Fig. 2: NIC high level architecture: transmission path.

C. Current hardware designs

Wi-Fi NICs are typically based on the hardware architecture depicted in Fig. 2, which details the transmission path (the process on the reception path is basically the reciprocal case). In contrast to old-styled *Programmed Input/Output* (PIO) designs, modern approaches use *Direct Memory Access* (DMA) for transferring data between the memory of the main host running the OS and the NIC circuitry. A kernel thread first writes frames into a consecutive set of memory pages, which

[8]This mechanism enables the transmission of many MPDUs with a single physical layer preamble and without spacing in-between

[9]BRS for 802.11g = 6, 12 and 24 (Mb/s)

are arranged in a circular ring of DMA slots, postponing the operation if there are no free slots; then it programs the DMA controller on the NIC to start retrieving and transmitting frames.

On the NIC side, the MAC CPU focuses on re/transmission operations, with the host communication being offloaded by the DMA subsystem that caches the first few frames from every queue in the NIC internal memory. When the BC of a queue reaches zero, the MAC CPU schedules the transmission of the Head-of-Line frame and when the frame is acknowledged or exceeds the maximum retransmission attempts, the MAC CPU reports a *frame done* to the kernel via an interruption request (IRQ). Then, the corresponding DMA microcontroller (DMA μC) starts fetching the next frame, and the kernel reclaims the DMA slot in the ring for the new frames that may arrive from upper layers.

Two are the main benefits of this DMA-based approach: (i) there are always frames available to the NIC, unlike the PIO case; and (ii) the main host CPU is relieved while the NIC transmits frames. Both the host CPU and the NIC CPU focus on their very specific tasks and frames can be transmitted at the maximum throughput allowed by the standard timings.

In nuce, this scheme emulates a simple FIFO queue for pushing frames to the NIC, and therefore it results adequate as long as there is a strict order in handling the frames: when the Head-of-Line (HOL) frame is served the NIC may access a new one.[10] However, for the case of Block Acknowledgment the operation is slightly different: not all the frames in a burst or an A-MPDU need to be retransmitted, as they might experience different channel conditions. Unfortunately, once the burst is handled, none of the frames are available at the NIC for retransmission, thereby this calls for the collaboration of the host kernel. After processing the BAREP, the kernel resends to the NIC the requested frames, possibly interleaved with new ones.[11] However, for 802.11aa this will slow down the video streaming rate or mix old frames, likely to have expired, with new ones. We describe in the following how we overcome this issue in our 802.11aa implementation, underlining how this version of BA (in contrast to the 802.11n) is hard to deploy with the current DMA-based operation.

IV. Implementing block acknowledgements over current architectures

Here we provide a very short description of our implementation of the GCR BA over a commercial off-the-shelf (COTS) platform [12]. We develop our implementation over Alix 2d2 boxes, which are equipped with a 500 MHz x86-like CPU that is able to process more frames per second than the maximum rate supported by the standard (54 Mb/s), and an 802.11 b/g card based on the Broadcom chipset BCM94318MPG. We install Linux on the machines, and perform our modifications over the default *b43* driver (and *mac80211* stack) and the OpenFWWF firmware installed in these wireless chipsets.

[10]For the case of the multiple queues supported in EDCA, it can be extended with multiple DMA controllers, each programmed to present different memory blocks to the NIC.

[11]Note that this operation holds for current 802.11n chipsets, e.g., Broadcom 43224 and 43225 ones.

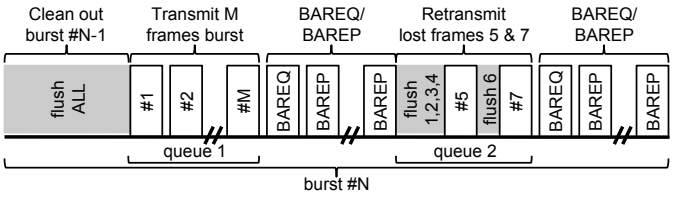

Fig. 3: Basic Operation of GCR BA from the transmitter side.

A. Implementing GCR BA

The GCR BA mechanism requires to send burst of M frames, collect and keep delivery information, and retransmit the required frames. To this end, the transmitter kernel waits for M frames from upper layers or a timeout expires. Then it initiates the DMA operation: as the NIC memory (16-KByte size) is not big enough to store a burst of arbitrary length for retransmitting lost frames, we use the four additional DMA queues available in the NIC for feeding it with copies of the same burst for handling retransmission.

At the end of each burst, the firmware forges the transmission of the BAREQ. The AP polls each station of the groupcast in a round-robin fashion: if no BAREP is received within a timeout the AP retransmits the BAREQ, up to seven times. The firmware then flushes the frames received by all destinations from the queues, and retransmit the other frames. When all the GCR frames of a burst are correctly received, or their lifetime expires, the firmware flushes all the queues and start over. In this way, the operations of GCR BA from the transmitter side for each burst can be decomposed in the three phases illustrated in Fig. 3: removing unused frames of the previous burst from the retransmission queues; transmitting the burst of M frames; sending and processing the BAREQ and BAREP pairs; selectively retransmitting lost frame by flushing those already received. The last two operations may be repeated multiple times.

At the receiver side, it is required that all stations in the groupcast transmit the BAREP upon the reception of the BAREQ. Each receiver stores the sequence number of the first frame in the burst and maps correctly received frames into a bitmap. This information is then copied into the BAREP. The management of duplicated received frames is performed by the *mac80211* kernel module.[12]

B. Performance impairments

As said above, the detailed validation and assessment of our implementation is available in our previous work [12]. Here our interest is to quantify how efficient is our implementation, i.e., the difference between experimental results and the maximum achievable performance. To this aim, we use a backlogged desktop machine as AP and measure the maximum

[12]Our implementation, as well as a detailed technical report, are available at http://www.ing.unibs.it/~openfwwf/GATS.php.

TABLE I: Efficiency of the implementation.

M	Theoretical R_i (Mb/s)	Experimental R_e (Mb/s)	η
8	30.81	24.25	78.7%
16	36.31	27.08	74.6%
32	39.86	28.61	71.8%

achievable bandwidth using GCR BA, with a packet length of $L = 1400$ B, to a group of 10 stations. Given that the 2.4 GHz band is quite populated in our testbed, we perform experiments during night on channel 14 where activity from external sources is negligible: for this reason links are very good and even if we fix data-rate to 54 Mb/s, with power set to 20 dBm we have practically no losses.

We define the efficiency as $\eta = R_e/R_i$, with R_e being the experimental throughput (average of five 30-s experiments), and R_i the maximum theoretical one. This is computed as:

$$R_i = \frac{M \times L}{T_{\text{GCR BA Exchange}}} \quad (1)$$

where M is the size of the burst, L is the payload in bytes and $T_{\text{GCR BA Exchange}}$ is the total time for the exchange illustrated in Fig. 3, assuming that the time to flush the queues and fetch new data is zero. We report the obtained values of R_e, R_i and η in Table I, for different values of the burst length M (the higher the M the larger the throughput, as the relative amount of time spent in the BAREQ-BAREP pairs is smaller).

According to the results, it is clear that there is a notable efficiency loss, with all η values being below 80%. Furthermore, the higher the M the lower the η, which suggests that the reason for this performance impairment is that the chipset is not very efficient when handling long batches of frames. To understand this, we modify the firmware to produce verbose log files, identifying that the flushing and fetching operations take relatively long periods of time. The reason for this is that these operations require the DMA system to operate with frames, which does not result very efficient in a number of circumstances, as we analyze next.

V. Anatomy of the Bottleneck

To understand if the DMA operations are the main reason for the identified inefficiency, we set up the same scenario as in the previous experiments. However, in this case we tweaked the firmware in the AP to assume that all BAREPs report a successful reception of frames, which corresponds to the most demanding case in terms of DMA operations. Indeed, under this assumption, after the reception of the last BAREP the AP has to empty all the retransmission queues and prepare for the next transmission burst, while in case some of the BAREP reported unsuccessful reception, the AP has to periodically switch between retransmission and flushing.

We also tweaked the firmware to measure the time t_f when the first flush operation is made (right after the arrival of the last BAREP), and the time t_b when the first frame of the next burst is available. These times are measured inside the NIC, thus avoiding the latency that appears when measuring inside the kernel due to the communication bus. We perform the experiment for three different burst sizes, $M = \{8, 16, 32\}$, and packet lengths ranging from 100 B to 1400 B in steps of 100 B. For each of these configurations, we perform five 30 s experiments, collecting approximately 37500 time delays (i.e., $t_b - t_f$). Then, in order to take into account the time required by flushing and fetching operations, we compute R_i adding the *median* of the time delays[13] to the denominator of (1). We reported the obtained values in Fig. 4 as "Experimental", while the values without this extra time are labeled as "Theoretical".

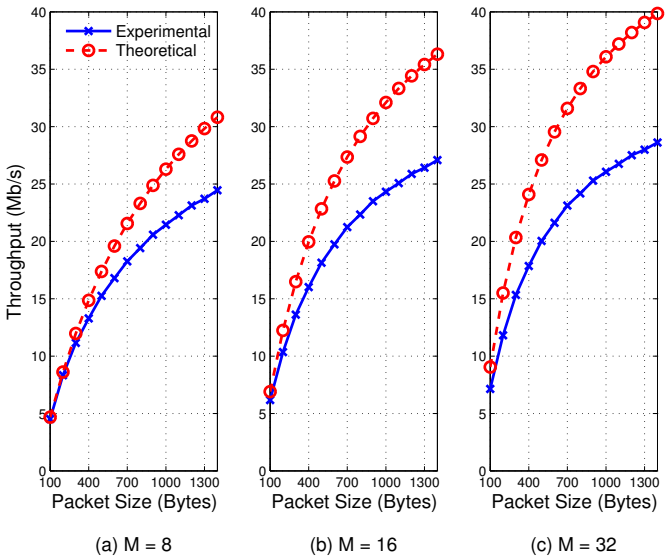

Fig. 4: Ideal and experimental throughput (MCS = 54 Mb/s) of 802.11aa GCR BA for different burst sizes and different packet lengths.

According to the results, it is evident that the flushing and fetching of new data leads to substantial performance impairments, which become more drastic with larger values of M or L. Also, we note that the values for $L = 1400$ B correspond to the ones reported in Table I, which validates to some extent our implementation of GCR BA, as the performance drop (for this value of L) can be attributed exclusively to the DMA operations, thus excluding any implementation issue.

We next analyzed in more detail the impact of M and L on the time to perform the flush and fetch operations. To this aim, we wrote an ad-hoc firmware that basically switches between two states: a *waiting state*, in which it does nothing while the application in user-space fills the DMA queue, and a *flushing state*, in which it records a timestamp, flushes the first M packets from the queue, and records another timestamp, so the difference between timestamps corresponds to the "flushing time". We plot the resulting flushing times for representative values of M and L in Fig. 5.

The figure illustrates that flushing times do not depend on the packet length until a certain threshold value L_{th}, and then grow linearly with L. This threshold depends on the burst size: the higher the M, the lower the L_{th}. Taking into account that each packet piggybacks a firmware header of

[13] We use the median of the collected samples as this removes "spurious" values due to clock wrap arounds.

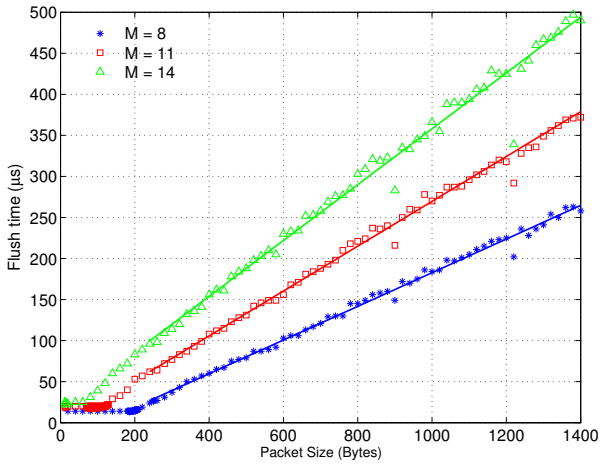

Fig. 5: Flushing times.

110 B, for all M cases with $M \times (L_{th} + 110) \approx 3072$ B, we have that the flushing times are constant (and very similar) for those configurations in which the packet bursts fits in the internal memory of the NIC. Actually, according to our measurements when $L < L_{th}$ the flushing time amounts to a couple of us per packet, which is approximately the time required to execute the drop loop code. When $L > L_{th}$, new packets must be fetched from the main system memory, thus resulting in a linear growth of the flushing time with the quantity of transferred data. Given that the transfer rate we measured under these conditions (i.e., 44 Mb/s) is well below the theoretical capabilities advertised for mini-PCI devices (266 Mb/s), we conclude that the slow transfer rate not only depends on the overhead of the PCI signaling, but also on the DMA interface of the NIC which is not optimized. Still, it should be noted that the measured transfer rate of 44 Mb/s is well above the maximum achievable throughput with 802.11a/g, which is around 30 Mb/s.

VI. Evolved Architecture for 802.11 Platforms

Based on our experience reported in Section V, we argue that apart from improving the mechanisms for transferring data between the NIC and the host system, the development of future wireless NICs should be (even more) inspired by current computer architectures. In this approach, *software developers* write both user programs and the OS, which interfaces to the hardware through a set of specific drivers released by *manufacturers*. This maps into NIC internals to an architecture where MAC algorithms are developed by *protocol implementers* using APIs created by *manufacturers* for accessing the NIC hardware. We next describe in detail our technical vision and its advantages.

A. Technical description

We present in Fig. 6 the evolved NIC architecture, thereby we build on Figure 2, and focus again on the transmission path. First, we introduce a *Real Time OS* executed by the NIC CPU that isolates the *MAC threads* from the mechanisms that exchange data with the host, and from the low level transmission primitives implemented by *proprietary PHY drivers*.

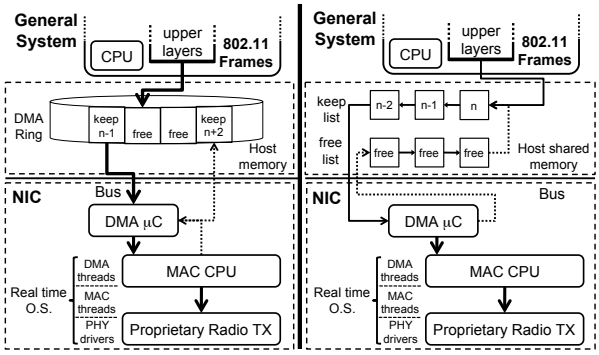

Fig. 6: Visionary architectures for 802.11 NICs.

At the radio, we still discard the SDR approach even if *ultimate* choice for targeting PHY flexibility: it either requires a really powerful internal CPU[14] or fails to provide decent MAC support if the host CPU is involved because of latencies. Also the FPGA solution is unfeasible: apart from energy issues, OFDM compliant PHY requires a "fat" FPGA with too many gates which makes it too expensive for consumer electronics. For these reasons passband-PHY circuitry should still be wired to inexpensive and energy efficient designs. This limits the flexibility of the radio, but manufacturers could easily strive PHYs to avoid strictly fixed MCS: e.g., they can allow reconfiguration of number of OFDM sub-carriers, ratio of the convolutional en/decoder for bit error protection, mapping between user-bits and symbols, and other parameters like the frequency and transmission power.

With regards to the *MAC CPU*, given that passband operations are offloaded by the hardwired PHY and considering the long MAC timings, even a slow clock of 44 MHz like in current chipsets[15] is enough to schedule frame transmission within sub-microsecond granularity. To this end the integration of an energy efficient ARM architecture, currently spreading as co-processors for many tasks (e.g., H.264 encoders), could be even more convenient than acquiring a third party design.

As for the communication mechanism between the host and the NIC, which we demonstrated in Section IV to be the real bottleneck for the new GCR BA procedure, and showed to be the major concern for the actual implementation of many recent proposals, we propose three solutions. The first two build on improvements of the DMA system: here the main problem is the looping over a ring of slots using a producer/consumer paradigm, with the MAC CPU allowed to only *release* a slot after usage.

The MAC program could instead report to the host that the slot is either *released* or *kept*: only in the first case, the slot will be replaced with a new frame. Then, the MAC program loops again over *kept* frames, directly programming the DMA controller for locating specific slots (Fig. 6 left). A second solution is to enable complete memory sharing between host and MAC CPU, as in inexpensive video chipsets, by means of a data structure (double linked list). The first list, populated

[14] With USRP and GnuRadio, decoding 11g frames at highest MCS pushes full load on a modern x86 CPU

[15] Atheros chipsets are clocked at 44 MHz, Broadcom at 88 MHz

by the host with new frames, is drained by the MAC CPU that moves released frames into the second list, and the host CPU will recycle these frames (Fig. 6 right). This option requires simpler mechanisms at both host and NIC CPUs for atomic read and assignment, which are already present on inexpensive ARM architectures. These two solutions permit manufacturers to keep low NIC memory and costs. A third solution, not reported in the figure for simplicity, consists of increasing the internal memory, so that the DMA system is only used for copying frames for later internal use by the NIC.

B. Advantages

Similarly to what happened in the computer world, abandoning the monolithic approach in favor of our more structured proposal, which integrates the operation of components provided by different players, would lead to a number of advantages. First, *hardware manufacturers* can focus on what really differentiates NICs, i.e.: the enhancement of key radio components, such as, the front-end to recover the clock and decode signals in noisy environments; the design of energy-efficient techniques, and the introduction of new mechanisms for de/encoding spatially multiplexed signal streams. Second, *protocol implementers* shall not have to deal with hardware specific details, hidden by primitives for transmitting and receiving frames, scheduling timers or configuring PHY specific parameters. Instead, they could focus on the actual protocol implementation, that would benefit from continuous software upgrades lined up with the latest version of a standard. Third, *developers of the Real time OS* specialize in improving algorithms for the allocation of the NIC internal resources to either MAC programs or driver threads; or on enhancing data communication mechanisms with the host by tuning the management of the shared memory. Fourth, *developers of the Host OS* do not have to implement different drivers, as they would solely interface with a *generic* device, leading to easier porting of the NIC itself to many OS. Finally, *consumers* will benefit from a better network experience, with resource-efficient devices that are constantly updated with new OS and protocol releases.

VII. Conclusions

In this paper, we have illustrated that existing 802.11 interfaces are based on generic hardware architectures, which are flexible enough to implement new mechanisms introduced by recent standards, but present certain bottlenecks that preclude an efficient operation. Building on our experiences from implementing the 802.11aa GATS mechanisms, we have proposed an evolved hardware architecture for these interfaces, inspired by the evolution of computer architectures, which would support timely updates of the platforms while providing the adequate motivations for all involved players. As a future work we aim to implement these proposals as a proof-of-concept in the WARP platform.

Acknowledgements

This work has been partially supported by the European Community through the CROWD project, FP7-ICT-318115, as well as through the WISHFUL project, H2020-645274.

References

[1] X. Zhang, J. Ansari, G. Yang, and P. Mähönen, "TRUMP: Supporting Efficient Realization of Protocols for Cognitive Radio Networks," in *Proc. IEEE DYSPAN*, 2011, pp. 476–487.

[2] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvind, and H. Balakrishnan, "Airblue: A System for Cross-Layer Wireless Protocol Development," in *Proc. ACM/IEEE ANCS*, 2010, pp. 4:1–4:11.

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

[4] G. Bianchi and I. Tinnirello, "One Size Hardly Fits All: Towards Context-Specific Wireless MAC Protocol Deployment," in *Proc. ACM WiNTECH*, 2013, pp. 1–8.

[5] Y. Dong, D. Xu, Y. Zhang, and G. Liao, "Optimizing Network I/O Virtualization with Efficient Interrupt Coalescing and Virtual Receive Side Scaling," in *Proc. IEEE CLUSTER*, 2011, pp. 26–34.

[6] H. Rauchfuss, T. Wild, and A. Herkersdorf, "A Network Interface Card Architecture for I/O Virtualization in Embedded Systems," in *Proc. USENIX WIOV*, 2010, pp. 1–7.

[7] Y. Grunenberger, M. Heusse, F. Rousseau, and A. Duda, "Experience with an implementation of the idle sense wireless access method," in *Proc. ACM CoNEXT*, 2007, pp. 24:1–24:12.

[8] D. Halpering, W. Hu, A. Sheth, and D. Wetherall, "Tool Release: Gathering 802.11n Traces with Channel State Information," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 53–53, 2011.

[9] 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 *Proc. USENIX NSDI*, 2010, pp. 205–218.

[10] P. Salvador, V. Mancuso, P. Serrano, F. Gringoli, and A. Banchs, "VoIP-iggy: Analysis and Implementation of a Mechanism to Boost Capacity in IEEE 802.11 WLANs Carrying VoIP traffic," *IEEE Transactions on Mobile Computing*, vol. 99, p. 14, 2013.

[11] D. Berger, F. Gringoli, N. Facchi, I. Martinovic, and J. Schmitt, "Gaining Insight on Friendly Jamming in a Real-world IEEE 802.11 Network," in *Proc. ACM WiSec*, 2014, pp. 105–116.

[12] P. Salvador, L. Cominardi, F. Gringoli, and P. Serrano, "A First Implementation and Evaluation of the IEEE 802.11aa Group Addressed Transmission Service," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 1, pp. 13–18, 2014.

[13] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. M. Voelker, "Sora: High Performance Software Radio Using General Purpose Multi-core Processors," in *Proc. USENIX NSDI*, 2009, pp. 75–90.

[14] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, and I. Tinnirello, "MAClets: Active MAC Protocols over Hard-Coded Devices," in *Proc. ACM CoNext*, 2012, pp. 229–240.

[15] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 3: MAC Enhancements for Robust Audio Video Streaming*, IEEE Amendment 802.11aa, 2012.

[16] A. Ganz, A. Savvides, and Z. Ganz, "Media Access Control Development Platform for Wireless LANs," in *Proc. IEEE ICECS*, 1999.

[17] A. Sharma and E. Belding, "FreeMAC: Framework for Multi-Channel MAC Development on 802.11 Hardware," in *Proc. ACM PRESTO*, 2008.

[18] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald, "SoftMAC: A Flexible Wireless Research Platform," in *Proc. ACM HotNets*, 2005.

[19] L. Wisniewski, H. Trsek, I. Dominguez-Jaimes, A. Nagy, R. Exel, and N. Kerö, "Location-based Handover in Cellular IEEE 802.11 Networks for Factory Automation," in *Proc. IEEE EFTA*, 2010.

[20] A. Khattab, J. Camp, C. Hunter, P. Murphy, A. Sabharwal, and E. W. Knightly, "WARP: A Flexible Platform for Clean-slate Wireless Medium Access Protocol Design," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 12, no. 1, pp. 56–58, 2008.

[21] D. Armstrong and M. Pearson, "A Rapid Prototyping Platform for Wireless Medium Access Control Protocols," in *Proc. IEEE ASAP*, 2007, pp. 403–408.

[22] A. de la Oliva, P. Serrano, P. Salvador, and A. Banchs, "Performance Evaluation of the IEEE 802.11aa Multicast Mechanisms for Video Streaming," in *Proc. IEEE WoWMoM*, 2013, pp. 1–9.

[23] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance Anomaly of 802.11b," in *Proc. IEEE INFOCOM*, 2003, pp. 1–8.