Control Theoretic Optimization of 802.11 WLANs: Implementation and Experimental Evaluation

Pablo Serrano[1], Paul Patras[2], Andrea Mannocci[3,1], Vincenzo Mancuso[3,1], and Albert Banchs[1,3]

[1]University Carlos III de Madrid, Spain
[2]Hamilton Institute, NUI Maynooth, Ireland
[3]Institute IMDEA Networks, Spain

Abstract

In 802.11 WLANs, adapting the contention parameters to network conditions results in substantial performance improvements. Even though the ability to change these parameters has been available in standard devices for years, so far no adaptive mechanism using this functionality has been validated in a realistic deployment. In this paper we report our experiences with implementing and evaluating two adaptive algorithms based on control theory, one centralized and one distributed, in a large-scale testbed consisting of 18 commercial off-the-shelf devices. We conduct extensive measurements, considering different network conditions in terms of number of active nodes, link qualities and traffic generated. We show that both algorithms significantly outperform the standard configuration in terms of total throughput. We also identify the limitations inherent in distributed schemes, and demonstrate that the centralized approach substantially improves performance under a large variety of scenarios, which confirms its suitability for real deployments.

1 Introduction

The IEEE 802.11 standard for Wireless LANs [1] has become one of the most commonly used technologies to provide broadband connectivity to the Internet. The default channel access mechanism employed in IEEE 802.11 networks is based on a CSMA/CA scheme, regulated by a set of parameters that determines the aggressiveness of the stations when trying to access the channel. In particular, the contention window (CW) parameter controls the probability that a station defers or transmits a frame once the medium has become idle, and therefore has a key impact on the WLAN performance.

Commercial devices implement a fixed CW configuration, which is known to yield suboptimal performance. Indeed, for a fixed CW, if too many stations contend the collision rate will be very high, while if few stations are backlogged the channel will be underutilized most of the time. This behavior has been analyzed by several works in the literature, e.g. [2], which have shown that adapting the CW to the number of backlogged stations significantly improves performance.

Following the above result, an overwhelming number of solutions have proposed to adapt the 802.11 MAC behavior to the observed network conditions with the goal of maximizing the WLAN performance [3–13]. However, as we detail in the related work section, these previous works suffer from at least one of these two limitations: (i) their performance has not been assessed with real deployments, and therefore lack experimental evidences gathered from scenarios with non-ideal channel effects and implementation constraints [3–10]; or (ii) they rely on non-standard capabilities, or functionality that is not supported by existing wireless devices, and therefore would require arduous workarounds to be implemented [3,5,9–13]. Furthermore, most of them are based on heuristics and lack the mathematical foundations to guarantee optimal performance [3–5, 11, 13].

In this paper, we present our experiences with the implementation of two adaptive algorithms, namely the *Centralized Adaptive Control* (CAC) [14] and the *Distributed Adaptive Control* (DAC) [15], both based on a Proportional Integrator (PI) controller that dynamically tunes the CW configuration to opti-

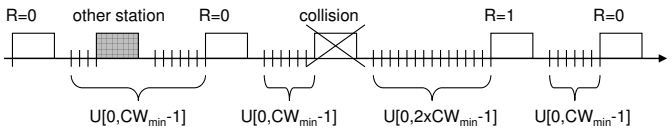

Figure 1: Retry flag marking upon collisions.

mize performance. In contrast to previous proposals, both algorithms are supported by solid theoretical foundations from control theory and can be easily implemented with unmodified existing devices.

We first provide a detailed description of the implementation of our adaptive mechanisms with commodity hardware and open-source drivers. The algorithms run as user space applications and rely on standardized system calls to estimate the contention level in the WLAN and adjust the CW configuration of 802.11 stations. We also provide insights into the differences between the theoretical design and the practical implementation of the algorithms, which arose with the inherent limitations of the real devices. By conducting exhaustive experiments in a large-scale testbed consisting of 18 devices, we evaluate the performance of our proposals under non-ideal channel effects and different traffic conditions. Additionally, we compare the performance of our algorithms against the default IEEE 802.11 configuration, and identify those scenarios where a network deployment can benefit from using such adaptive mechanisms.

Our results confirm that both approaches outperform the standard's default scheme, improving the performance by up to 50%. Our experiments also reveal that the distributed algorithm suffers from a number of problems with heterogeneous radio links, which are inherent in its distributed nature and the limitations of the wireless interfaces. In contrast, the centralized scheme exhibits remarkable performance under a wide variety of network conditions. The conclusions drawn from our analysis prove the feasibility of using adaptive MAC mechanisms in realistic scenarios and provide valuable insights for their design.

The remainder of the paper is organized as follows. Section 2 summarizes the IEEE 802.11 EDCA protocol and the underlying principles of CAC and DAC. Section 3 details the implementation of the functionality comprised by the proposed schemes. Section 4 describes our testbed and the validation of the implementation of the algorithms. Section 5 presents a thorough experimental study of the algorithms in a wide set of network conditions. Finally, Section 6 summarizes the related work and Section 7 concludes the paper.

2 Background

This section summarizes the behavior of IEEE 802.11 EDCA and the two adaptive protocols implemented in this paper.

2.1 IEEE 802.11 EDCA

The IEEE 802.11 Enhanced Distributed Channel Access (EDCA) mechanism is a CSMA/CA-based protocol that operates as follows. If a station with a new frame to transmit senses the channel idle for a period of time equal to the arbitration interframe space parameter ($AIFS$), the station transmits. Otherwise, if the channel is busy (either immediately or during the $AIFS$ period), the station continues to monitor the channel until it is sensed idle for an $AIFS$ interval, and then executes a backoff process.

Upon starting the backoff process, stations compute a random integer uniformly distributed in the range $[0, CW - 1]$, and initialize their backoff time counter with this value. The CW value is called the contention window, and depends on the number of failed transmission attempts. For the first transmission attempt the minimum contention window (CW_{min}) is used. In case of a collision, its value doubles, up to a maximum value CW_{max}. The backoff time counter is decremented once every time slot if the channel is sensed idle, frozen when a transmission is detected on the channel, and reactivated when the channel is sensed idle again for an $AIFS$ time. When the backoff time counter reaches zero, the station transmits its frame in the next time slot.

When two or more stations start transmitting simultaneously, a collision occurs. Acknowledgment (ACK) frames are used to notify a transmitting station of successfully received frames. In the case of a

failed transmission, the station doubles its CW and reenters the backoff process. Once a frame has been successfully transmitted or the retry limit has been exceeded, the CW value is set again to CW_{min}. To prevent duplicates, the standard uses a retry flag R to mark those frames that are being retransmitted, i.e., the flag is set to 0 on the first transmission attempt, and set to 1 on every retransmission (see Fig. 1). As we discuss later, our algorithms exploit this functionality to infer the network conditions and adapt the CW of the stations accordingly.

To support service differentiation, EDCA implements different access categories (ACs) at every station, each having a different backoff configuration. The parameters of each AC are announced by the Access Point using the Beacon Frames. In the rest of the paper we do not consider service differentiation and assume that all stations only execute the Best Effort AC.

2.2 Optimal Point of Operation of the WLAN

Both CAC and DAC share the goal of adjusting the CW to drive the WLAN to the optimal point of operation that maximizes the total throughput given the observed network conditions. Let p denote the probability that a transmission attempt collides. Following [2], we have shown in [14, 15] that the optimal collision probability in the WLAN p_{opt} can be approximated by

$$p_{opt} \approx 1 - e^{-\sqrt{\frac{2T_e}{T_c}}}, \tag{1}$$

where T_e is the duration of an idle slot (a PHY layer constant) and T_c is the average duration of a collision. Therefore, p_{opt} does not depend on the number of stations, but only on the average duration of a collision T_c. Given the average length $E[L]$ of the longest packet involved in a collisions, T_c can be computed using

$$T_c = T_{PLCP} + \frac{E[L]}{C} + EIFS.$$

where T_{PLCP} is the duration of the Physical Layer Convergence Protocol (PLCP) preamble and header, C is the modulation rate and $EIFS$ is a PHY layer constant.

2.3 Centralized Adaptive Control Algorithm

The *Centralized Adaptive Control* (CAC) algorithm [14], illustrated in Fig. 2, is based on a PI controller located at the Access Point (AP). This controller computes the configuration of the CW_{min} parameter as an integer ranging between the default minimum and maximum values defined by the standard specification, while CW_{max} is set as $CW_{max} = 2^m CW_{min}$, following the standard binary exponential backoff procedure.[1]

Following the above, the controller performs two tasks every beacon interval (approx. 100 ms): (i) it estimates the current point of operation of the WLAN as given by the observed collision probability p_{obs}, and (ii) based on this estimation and p_{opt}, it computes the CW configuration to be used during the next beacon interval and sends it to the stations in a beacon frame.

The computation of p_{obs} is based on the observation of the retry flag of successful frames. Let us denote by R_1 (R_0) the number of observed frames with the retry bit set (unset) during a beacon interval. Assuming that no frames exceed the retry limit given by the `MAX_RETRY` parameter,[2] and that transmissions attempts collide with a constant and independent probability,[3] the observed probability of a collision in the WLAN can be estimated with (see [14]):

$$p_{obs} = \frac{R_1}{R_0 + R_1}. \tag{2}$$

The error signal e fed into the PI controller to calculate the new CW_{min} is computed as the difference between the observed collision probability p_{obs} and the target value p_{opt}:

$$e = p_{obs} - p_{opt}. \tag{3}$$

[1] In our experiments we use the PHY layer parameters of IEEE 802.11a, hence $m = 6$ is chosen.
[2] Note that this assumption is accurate as in an optimally configured WLAN the collision probability is very low.
[3] This assumption has been widely used and shown to be accurate, see e.g. [2].

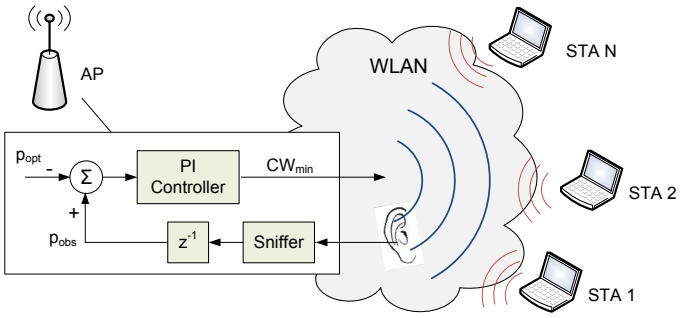

Figure 2: CAC algorithm.

In this way, when the observed collision probability is above the target value, the error signal will be positive and trigger an increase of the CW_{min}, and consequently a decrease of the collision rate in the next beacon interval. Similarly, when the collision probability is below the target value, CW_{min} is decreased in order to increase the activity on the channel.

The $\{K_P, K_I\}$ parameters of the PI controller are obtained using the Ziegler-Nichols rules, to achieve a proper trade-off between stability and speed of reaction to changes, and are given by (see [14] for the details):

$$\begin{aligned} K_P &= \frac{0.8}{p_{opt}^2(1 + p_{opt}\sum_{k=0}^{m-1}(2p_{opt})^k)}; \\ K_I &= \frac{0.4}{0.85 \cdot p_{opt}^2(1 + p_{opt}\sum_{k=0}^{m-1}(2p_{opt})^k)}. \end{aligned} \tag{4}$$

The operation of CAC is summarized in Algorithm 1.

2.4 Distributed Adaptive Control Algorithm

The *Distributed Adaptive Control* (DAC) algorithm [15] employs an independent PI controller at each station to compute its CW configuration, to drive the overall collision probability to the target value p_{opt}. As illustrated in Fig. 3, each controller computes the CW_{min} value employed by its Network Interface Card (NIC), based on the locally observed network conditions. Similarly to CAC, CW_{max} is set as $CW_{max} = 2^m CW_{min}$.

Algorithm 1 Centralized Adaptive Control algorithm.

```
1:  while true do
2:      repeat
3:          if new frame sniffed then
4:              retrieve retry flag
5:              if retry flag is set then
6:                  Increment R_1
7:              else
8:                  Increment R_0
9:              end if
10:         end if
11:     until new beacon interval
12:     compute p_obs[t] using (2)
13:     e[t] = p_obs[t] − p_opt
14:     CW_min[t] = CW_min[t − 1] + K_P · e[t]+
15:                 +(K_I − K_P) · e[t − 1]
16:     send beacon with new CW configuration
17: end while
```

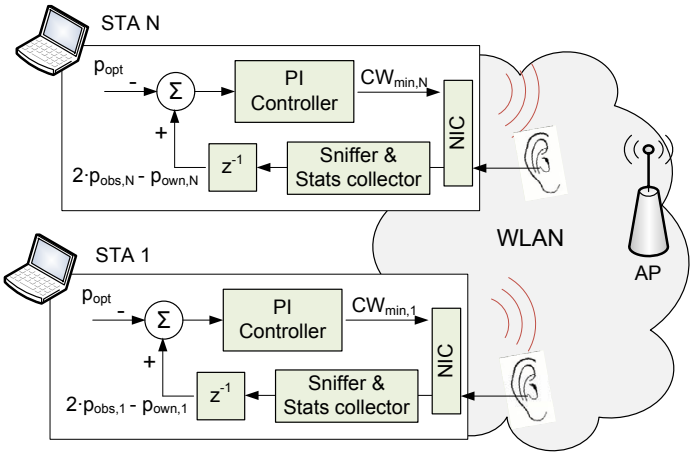

Figure 3: DAC algorithm.

While with centralized approaches all stations use the same configuration provided by a single entity, and therefore fairly share the channel, with distributed approaches this is not necessarily the case. To guarantee a fair throughput distribution, the error signal utilized in DAC consists of two terms: one to drive the WLAN to the desired point of operation, and another one to achieve fairness between stations. More specifically, the error signal at station i is given by

$$e_i = e_{collision,i} + e_{fairness,i}. \tag{5}$$

The first term of (5) ensures that the collision probability in the network is driven to the target value:

$$e_{collision,i} = p_{obs,i} - p_{opt}, \tag{6}$$

where $p_{obs,i}$ denotes the collision probability as measured by station i. When the collision probability observed by station i is larger than the target value, the above term yields a positive error that increases the CW of station i, thereby reducing the collision probability.

The second term of (5) is computed as

$$e_{fairness,i} = p_{obs,i} - p_{own,i}, \tag{7}$$

where $p_{own,i}$ is the collision probability experienced by station i. The purpose of this second component of e_i is to drive the CW of all stations to the same value. Indeed, the higher the CW_{min}, the lower the number of collisions caused, and thereby, the lower the observed collision probability $p_{obs,i}$ is. Therefore, a station will increase its CW_{min} if it experiences less collisions than the others.

To compute the error signal, each station needs to measure $p_{obs,i}$ and $p_{own,i}$. The former is computed as p_{obs} in CAC. For the computation of $p_{own,i}$, we rely on the following statistics which are readily available from wireless cards: the number of successful transmission attempts T and the number of failed attempts F. With these statistics, $p_{own,i}$ is computed as:

$$p_{own,i} = \frac{F}{F+T}. \tag{8}$$

Each station will estimate $p_{obs,i}$ and $p_{own,i}$ and compute the error signal e_i, which is provided to the PI controller for the computation of the new $CW_{min,i}$. Like in CAC, we choose to trigger an update of the $CW_{min,i}$ every beacon interval, as this is compatible with existing 802.11 hardware, which is able to update the EDCA configuration at the beacon frequency.

Although the analysis of DAC, based on multivariable control theory, significantly differs from the analysis of CAC, based on standard control theory, the $\{K_P, K_I\}$ parameters that each station uses are the same ones of (4), as proved in [15]. The DAC operation is summarized in Algorithm 2.

Algorithm 2 Distributed Adaptive Control algorithm.

while true **do**
  repeat
    if new frame sniffed **then**
      retrieve retry flag and
      increment R_0 or R_1 accordingly
    end if
  until beacon received
  compute $p_{obs,i}$ using (2)
  fetch T and F from driver stats
  compute $p_{own,i}$ using (8)
  $e[t] = 2 \cdot p_{obs,i}[t] - p_{own,i}[t] - p_{opt}$
  $CW_{min}[t] = CW_{min}[t-1] + K_P \cdot e[t] +$
    $+(K_I - K_P) \cdot e[t-1]$
  update the local CW configuration
end while

3 Implementation Details

A major advantage of CAC and DAC is that they are based on functionalities already available in IEEE 802.11 devices, and therefore can be implemented with COTS hardware. In this section we describe the hardware used in our deployment and the implementation of the functionality required by CAC and DAC.

3.1 Implementation Overview

We have implemented our algorithms using Soekris net4826-48 devices.[4] These are low-power, low-costs computers equipped with 233MHz AMD Geode SC1100 CPUs, 2 Mini-PCI sockets, 128 Mbyte SDRAM and 256 Mbyte compact flash circuits for data storage. To accommodate the installation of current Linux distributions, we have extended the storage capacity of the boards with 2-GB USB drives. As wireless interfaces, we used Atheros AR5414-based 802.11a/b/g devices.

As software platform we installed Gentoo Linux OS (kernel 2.6.24) and the popular MadWifi open-source WLAN driver[5] (version v0.9.4), which we modified as follows: (*i*) we enabled the dynamic setting of the EDCA parameters for the best effort access category, which is in line with the standard specifications but disabled by default in the driver, (*ii*) we overwrote the drivers' EDCA values for the best-effort traffic with the standard recommended ones [1], and (*iii*) for the case of DAC we modified the driver to enable the stations to employ the locally computed EDCA configuration using standardized system calls (as described in Section 3.4). The source code of the modified drivers and our implemented prototypes is available online.[6]

Fig. 4 illustrates the main modules of our implementation of CAC and DAC. The algorithms do not require introducing modifications to the hardware/firmware nor have tight timing constraints, and therefore they can run as user-space applications that communicate with the driver by means of `IOCTL` calls. We also take advantage of the ability of the MadWifi driver to support multiple virtual devices using different operation modes (master/managed/monitor) with a single physical interface. In the following we detail the implementation of the different modules.

3.2 Estimation of $\mathbf{p}_{obs}$

Both algorithms require to estimate the collision probability observed in the WLAN. For the case of CAC this is performed only at the AP and results in p_{obs}, while for the case of DAC this is performed independently at each station i and results in $p_{obs,i}$. The estimators are computed with (2), which relies

[4] `http://www.soekris.com/`
[5] `http://madwifi-project.org/`
[6] `http://www.hamilton.ie/ppatras/#code`

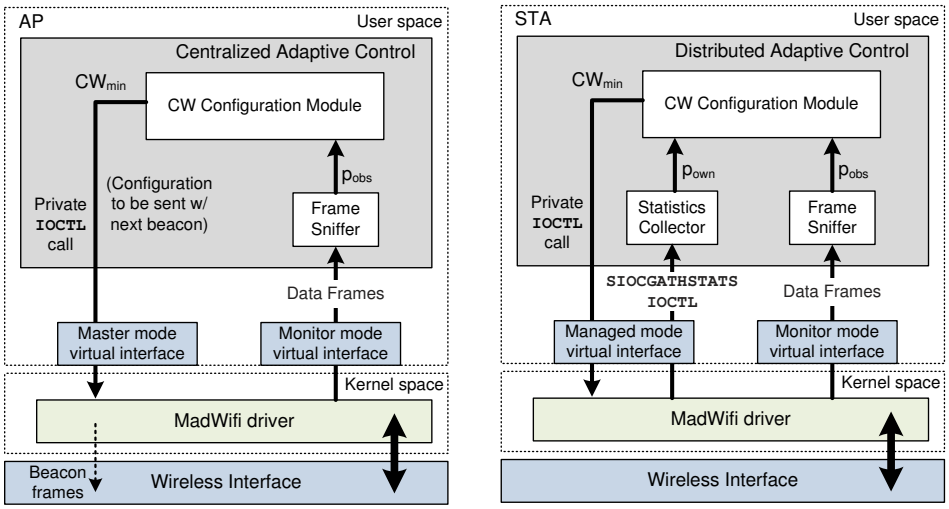

Figure 4: CAC and DAC implementations.

on observing the retry flag of the overheard frames. We next explain how these values are obtained from a practical perspective.

To overhear frames, we utilize a virtual device operating in the so called `monitor` mode with promiscuous configuration. With this configuration, the device passes all traffic to user-space applications, including frames not addressed to the station. We also configure the device to pass the received frames with full IEEE 802.11 link layer headers, such that the Frame Control field of the frames (where the retry flag resides) can be examined.

With this set-up, the algorithms open a `raw` socket to the driver, which enables the reception of Layer 2 frames. Through this socket the algorithms listen for transmitted frames and process their headers in an independent thread (the "Frame Sniffer" module of Fig. 4). For every observed frame, one of the counters used in the estimation of the collision probability is incremented: R_0 if the retry flag was unset, R_1 if the retry flag was set. Every beacon interval the computation of p_{obs} or $p_{obs,i}$ using (2) is triggered, and then the counters are reset to zero.

3.3 Estimation of p_{own}

In addition to the observed collision probability $p_{obs,i}$, the DAC algorithm requires to estimate the experienced collision probability $p_{own,i}$. We perform this computation in the "Statistics Collector" module of Fig. 4 using information recorded by the wireless driver. More specifically, at the end of a beacon interval we open a communication channel with the driver instance, configured in `managed` mode, and perform a `SIOCGATHSTATS IOCTL` request. Upon this request, the driver populates an `ath_stats` data structure, which contains detailed information about the transmitted and received frames since the Linux kernel has loaded the driver module. Out of the statistics retrieved, the records that are of particular interest for our implementation are:

- `ast_tx_packets`: number of unique frames sent to the transmission interface.
- `ast_tx_noack`: number of transmitted frames that do not require ACK.
- `ast_tx_longretry`: number of transmission retries of frames larger than the RTS threshold. As we do not use the RTS/CTS mechanisms, this is the total number of retransmissions.
- `ast_tx_xretries`: number of frames not transmitted due to exceeding the retry limit, which is set by the `MAX_RETRY` parameter.

To compute $p_{own,i}$ we need to count the number of successful transmissions and the number of failed attempts. To compute the former, we subtract from the number of unique frames those that are not acknowledged (e.g., management frames) and those that were not delivered,

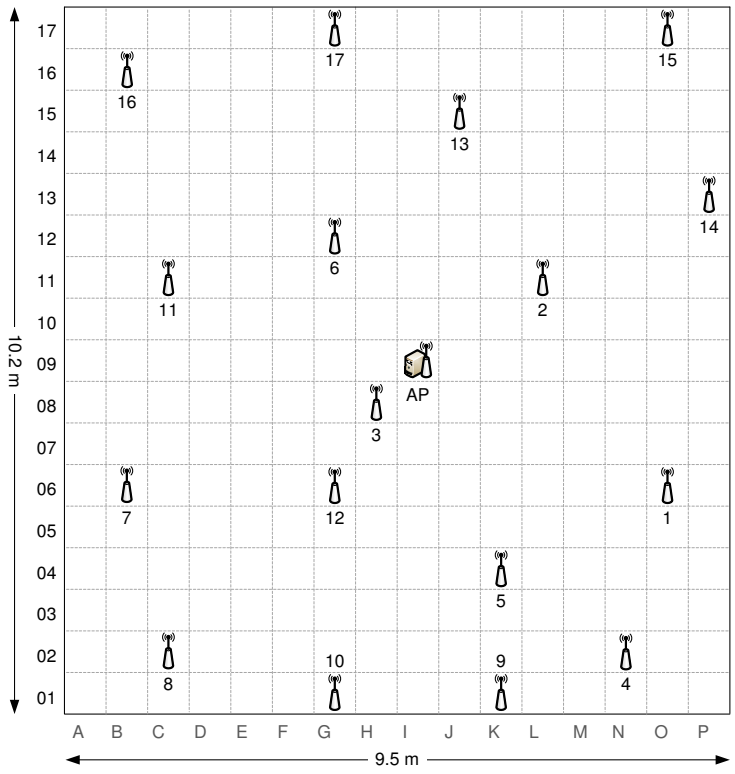

Figure 5: Deployed testbed.

$$\textit{Successes} = \texttt{ast\_tx\_packets} - \texttt{ast\_tx\_xretries} - \texttt{ast\_tx\_noack}.$$

Similarly, to compute the number of failed attempts, out of the total number of retransmissions we do not count those retransmissions caused by frames that were eventually discarded because the `MAX_RETRY` limit was reached, therefore,

$$\textit{Failures} = \texttt{ast\_tx\_longretry} - \texttt{ast\_tx\_xretries} \cdot \texttt{MAX\_RETRY}.$$

With the above, the terms F and T of (8) used to estimate $p_{own,i}$ are computed as

$$\begin{aligned} \textit{F[t] = Failures[t] - Failures[t-1],} \\ \textit{T[t] = Successes[t]- Successes[t-1],} \end{aligned}$$

where t denotes the time of the current beacon interval and $t-1$ the previous one.

3.4 Contention Window Update

With the estimated collision probabilities, CAC and DAC compute the error signal at the end of a beacon interval according to (3) and (5), respectively. Depending on this value, the PI controller triggers an update of the CW_{min} to be used in the next beacon interval t, according to the following expression:

$$CW_{min}[t] = CW_{min}[t-1] + K_P \cdot e[t] + (K_I - K_P) \cdot e[t-1].$$

To ensure a safeguard against too large and too small CW_{min} values we impose lower and upper bounds for the CW_{min}. We set these bounds to the default CW_{min}^{DCF} and CW_{max}^{DCF} values specified by the standard, which are 16 and 1024, respectively, for IEEE 802.11a [16].

The algorithms assume that the CW_{min} can take any integer value in the $[16, 1024]$ range. However, with our devices only integer powers of 2 are supported (i.e., $CW_{min} \in \{16, 32, \ldots, 1024\}$). Therefore, the value actually used is obtained as:

$$\texttt{CW[t] = 2**rint(log}_2\texttt{(}CW_{min}[t]\texttt{))}.$$

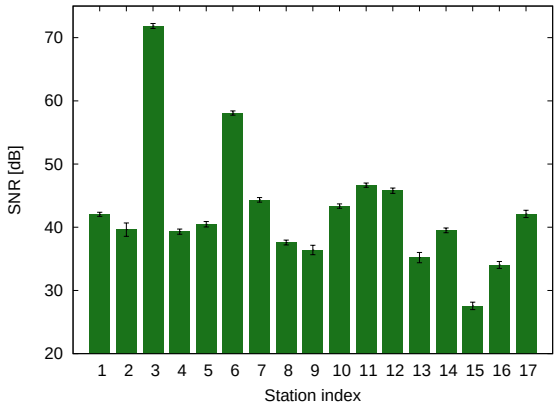

Figure 6: SNR of the links between each node and the Access Point.

where `rint(x)` is a function that returns the integer value nearest to `x` and ** represents the exponentiation operation.

To commit the computed `CW` configuration, first we retrieve the list of private `IOCTL`s supported by the device to search for the call that sets the CW_{min}. Once this call has been identified, we prepare an `iwreq` data structure with the following information: the interface name, the base-2 exponent of the `CW` computed as above, the access category index as defined by the standard (`0` for Best Effort) and an additional parameter that identifies if the value is intended to be used locally or propagated. For the case of DAC this value is set to 0, as the `CW` is only intended to the local card, while for the case of CAC is set to 1, thereby requesting the driver to broadcast the new `CW` within the EDCA Parameter Set element of the next scheduled beacon frame.

4 Testbed Description and Validation of the Implementation

In this section we first describe our testbed. Then we analyze the link qualities between each node and the AP and show that our set-up is able to mimic a realistic deployment with significant differences in terms of SNR. Finally, we confirm that, despite the constraints imposed by the devices and the realistic radio conditions, both CAC and DAC are able to drive the WLAN to a stable point of operation.

4.1 Testbed Description

Our testbed is located in the Torres Quevedo building at University Carlos III de Madrid. It consists of 18 devices deployed under the raised floor, a placement that provides physical protection as well as radio shielding to some extent (see [17]).

Fig. 5 illustrates the location of the nodes. We placed one node (denoted as AP) towards the center of the testbed, thus following the placement of an Access Point in a realistic deployment, while the other stations (numbered from 1 to 17 in no particular order) are distributed at different distances from this node. All nodes are equipped with 5 dBi omnidirectional antennas and are configured to operate on channel 64 (5.32 GHz) of IEEE 802.11a standard [16], where no other WLANs were detected. All nodes use the 16-QAM modulation and coding scheme, which provides 24 Mbps channel bit rate, as calibration measurements showed that this was the highest rate achievable by the node with the worst link to the AP (node 15). Additionally, we disabled the RTS/CTS, rate adaptation, turbo, fast frame, bursting and unscheduled automatic power save delivery functionality, as well as the antenna diversity scheme for transmission/reception.

Unless otherwise specified, all nodes use the same transmission power level of 17 dBm. Given the node placement of Fig. 5, this setting results in very dissimilar link qualities between each station and the AP (e.g., node 3 is extremely close). To confirm this link heterogeneity, we designed the following experiment. For a given node, we ran a 10-second `ping` test between the station and the AP, recording

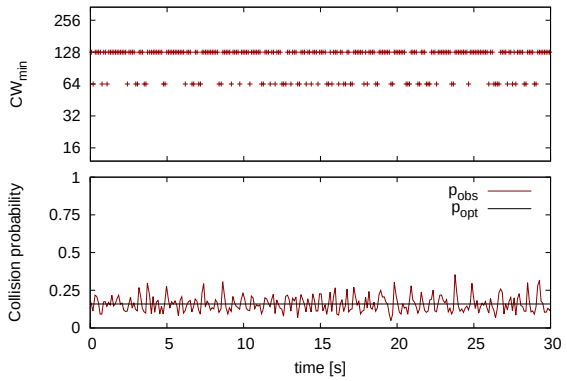

Figure 7: Announced CW_{min} and observed collision probability with CAC.

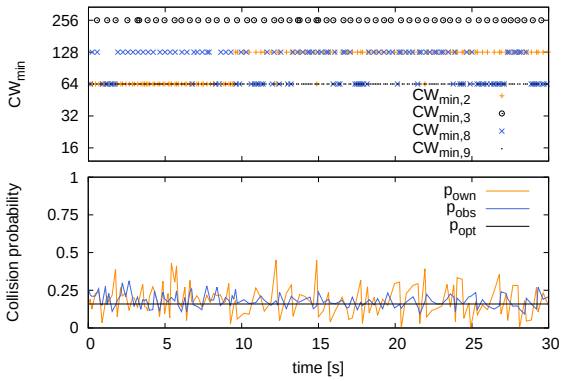

Figure 8: CW_{min} used by four selected nodes and the estimated p_{obs} and p_{own} with DAC.

the SNR values of the received frames as obtained by the `wireshark` packet analyzer[7] from the `radiotap` header.[8] This test was performed on a node-by-node basis, and repeated for 18 hours. The average and standard deviation of the SNR for each link are shown in Fig. 6.

From the figure we confirm that the use of a fixed power setting in the considered deployment results in very diverse radio links between the AP and the stations. Note that, throughout the reported experiments, we periodically repeated the above measurement in order to confirm that the radio conditions did not change.

4.2 Validation of the Algorithms

Our first set of experiments aims at confirming that the good operation properties of CAC and DAC, obtained analytically and via simulations in [14, 15], are also achieved in a real testbed. Specifically, we want to confirm that the use of the algorithms results in stable behavior despite the described hardware/software limitations and the impairments introduced by the channel conditions, and also assess their resource consumption in terms of CPU and memory usage.

Point of operation. We consider a scenario with $N = 10$ stations, constantly backlogged with 1500-Byte UDP frames, which send data to the AP utilizing `iperf`.[9] For the case of the centralized algorithm (CAC) we log its key variables, namely, the `CW` announced in beacon frames and the observed collision probability p_{obs}. Both are obtained every 100 ms and depicted in Fig. 7.

As the figure shows, CAC drives the WLAN to the desired point of operation. Indeed, the announced `CW` oscillates between the two allowed values closest to the optimal CW_{min}, while p_{obs} fluctuates stably around the desired p_{opt} given by (1). We conclude that, despite the hardware limitations imposed on the values of `CW` and the channel impairments, CAC is able to drive the WLAN to the desired point of operation.

Next we validate the operation of the distributed algorithm (DAC). We consider the same scenario as before, logging the key parameters of the algorithm at each station, namely $CW_{min,i}$, $p_{own,i}$ and $p_{obs,i}$. In Fig. 8 we depict in the upper subplot the evolution of the CW_{min} used by four representative nodes (namely 2, 3, 8 and 9), while in the lower subplot we show the collision probabilities estimated by node 2 ($p_{obs,2}$ and $p_{own,2}$).

From the two subplots we see that DAC also drives the average collision probability in the WLAN to the desired value. However, there is a key difference as compared to the previous case: while with CAC all stations use the same CW_{min} value, with DAC they operate at different average CW_{min}. Indeed, the four stations considered in the experiment use average CW_{min} values of 92, 300, 92 and 64, respectively. As we will explain in Section 5.2, this behavior is caused by the relative differences in link qualities,

[7] `http://wireshark.org`

[8] With the `radiotap` option, the driver provides additional information about received frames to user-space applications, including the signal-to-noise ratio.

[9] `http://sourceforge.net/projects/iperf/`

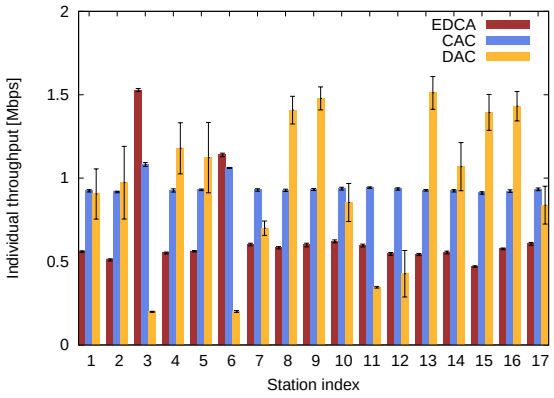

Figure 9: Throughput per station w/ UDP traffic.

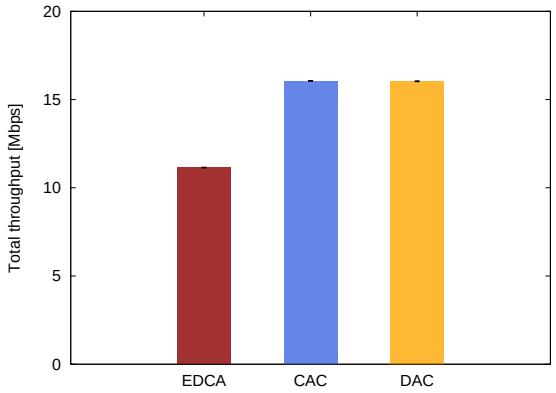

Figure 10: Total throughput w/ UDP traffic.

already identified above, combined with the inability of the wireless interface to identify the reasons for a packet loss.

Resource consumption. In addition to analyzing the performance of CAC and DAC, it is also important to assess their resource consumption. For this purpose, we analyzed the CPU and memory usage of the algorithms utilizing the `top` Linux application, which provides a dynamic real-time view of a running system. With this tool, we recorded the used shares of the CPU time and available physical memory with a frequency of 1 sample per second and computed the average usage. CAC, which runs exclusively at the AP, demands on average 39% of the CPU time and only 1.6% of the physical memory. For the case of DAC, which runs at every station, the average CPU time consumption is 28%, while the physical memory consumption is 4.3%. Given the low speed of the nodes' CPU (233 MHz) and their reduced physical memory (128 MB), these results show that both CAC and DAC are suitable for commercial deployments.

5 Performance Evaluation

We next assess the performance of the algorithms under a large number of different scenarios and evaluate their improvements over the default EDCA configuration, which we use as a benchmark. Each considered experiment runs for 2 minutes and is repeated 10 times to obtain average values of the measured metrics with good statistical significance.

5.1 UDP Throughput

We first measure the achievable throughput between the nodes and the AP when all the stations are transmitting UDP traffic at the same time. Fig. 10 plots the average and standard deviation of the total throughput obtained with each mechanism. We observe that the EDCA default configuration achieves around 11 Mbps, while the use of DAC and CAC results in a performance gain of approximately 45%. Therefore, we confirm that both approaches, by properly adapting the CW configuration to the number of contending stations, achieve a much higher efficiency.

To further examine the performance of the algorithms we plot the per-station throughput in Fig. 9. According to the figure, the use of the EDCA recommended values not only provides the lowest overall throughput figures, but also fails to provide a fair sharing of the available bandwidth. Indeed, it can be seen that, e.g., the node with the best link quality to the AP (node 3) achieves more than three times the throughput obtained by the station with the poorest link (node 15).

While DAC provides a larger total throughput than EDCA, it does not improve the level of fairness. Actually, it results in a somehow *opposite* performance as the one obtained with EDCA: stations that obtained a relatively large bandwidth with EDCA (e.g., nodes 3, 6) now obtain a relatively small bandwidth with DAC. The use of CAC, on the other hand, provides the best performance both in terms of total throughput and fairness, as it provides all stations with very similar throughput values.

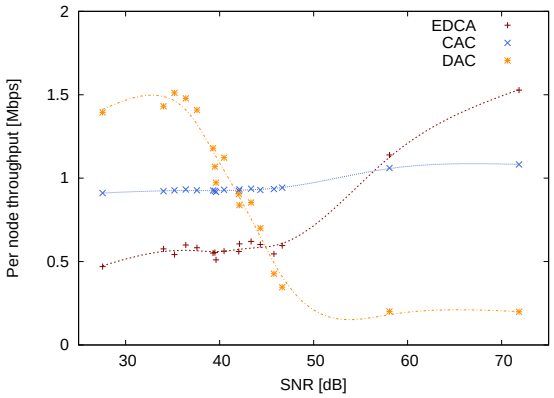

Figure 11: Throughput obtained vs. SNR.

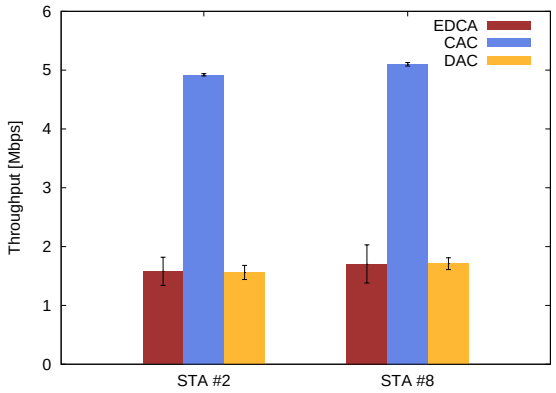

Figure 12: Performance with hidden nodes.

To quantify the throughput fairness achieved by the considered mechanisms we compute the Jain's fairness index (JFI) [18]. The resulting JFI values are 0.865, 0.997 and 0.817 for the case of EDCA, CAC and DAC, respectively. These figures confirm the good fairness properties of CAC, and shows that DAC and EDCA suffer from a higher level of unfairness, a result that we analyze next.

5.2 Impact of SNR on Throughput

We have seen that link quality affects throughput distribution, in particular for EDCA and DAC. To analyze this impact, we plot in Fig. 11 the average UDP throughput per station vs. the SNR of the link between the station and the AP. Note that for ease of visualization we also plot *natural smoothing splines* over the data points.

From the figure we observe that: (*i*) for EDCA there is a noticeable and positive correlation between SNR and throughput; (*ii*) for CAC, performance is not much affected by SNR dissimilarities, as significantly better link qualities result in very small throughput improvements; (*iii*) for DAC there is a large and negative correlation between SNR and throughput, with small differences in terms of SNR causing large differences in terms of throughput.

For the case of EDCA, the positive correlation is caused by the *capture effect*. With this effect, in case of a collision the receiver can decode the packet with the higher SNR. As a result, stations with better link quality obtain higher throughput. In contrast, the use of CAC reduces the number of collisions in the WLAN, and therefore the impact of the capture effect is significantly reduced.

For the case of DAC, the negative correlation is also driven by the capture effect as follows. Nodes with high capture probability will experience smaller collision rates than the others, and therefore will have $p_{own,i}$ smaller than $p_{obs,i}$. This will cause a positive error signal according to the $e_{fairness,i}$ term in (7), which will result in large CW_{min} values. Conversely, nodes with low capture probability will experience larger $p_{own,i}$ values and smaller $p_{obs,i}$ ones, and therefore will have smaller CW_{min} configurations. In this way, capturing nodes will transmit less often and therefore will obtain low throughput figures, while the other nodes will transmit more often and experience a higher throughput. Additional experiments with different transmission power settings, not reported due to space constraints, confirmed that a careful *equalization* of the link qualities is able to restore fairness to some extent.

5.3 Hidden Nodes Scenario

Our adaptive algorithms have been designed for scenarios where all stations are in radio range of each other and coordinate their transmissions by means of carrier sensing. However, in real deployments hidden nodes may be present, and therefore we want to investigate their behavior under such circumstances.

To this aim, we ran extensive measurements, selecting different topologies and different transmission power settings, to determine the most pathological scenario. This is obtained when node 3 acts as AP, and nodes 2 and 8 act as stations, using a transmission power level of 5 dBm. With this setting, each EDCA station transmitting in isolation (i.e., with the other station silent) obtains about 16.3 Mbps of

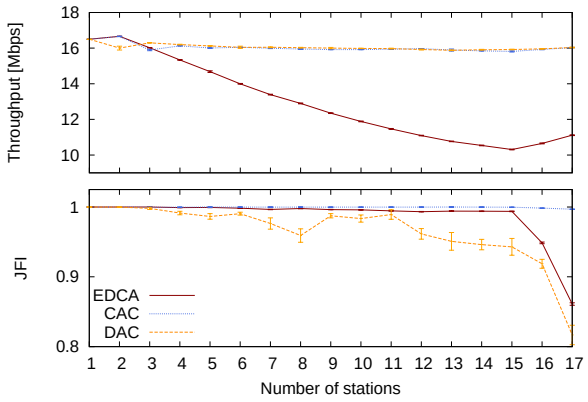

Figure 13: Total throughput and fairness for different number of stations.

UDP throughput, while if both stations transmit simultaneously the throughput of each one drops to 1.6 Mbps. Thereby we managed to reproduce a hidden node scenario.

We then repeated the experiment with CAC and DAC, and obtained the results depicted in Fig. 12. We observe that the use of DAC does not improve performance over EDCA. In contrast, CAC provides a dramatic throughput increase, i.e., more than three times the throughput attained with the other mechanisms. We conclude that CAC detects the large collision rate and commands hidden nodes to be less aggressive by announcing a higher CW_{min}, which lessens (but does not eliminate) the hidden node problem. On the other hand, a station running DAC is not able to overhear MAC (re-)transmissions from hidden nodes, and hence cannot correctly estimate the collision probability in the network.

5.4 Impact Network Size

We next evaluate the performance of the algorithms as a function of the number of stations. To this aim, we measure the total throughput and JFI for an increasing number of contending nodes, adding new stations in ascending order of their link qualitiy. We plot the obtained results in Fig. 13.

We observe that for both DAC and CAC the total throughput performance is practically flat, regardless of the number of stations. This result confirms that both approaches are able to adapt the CW to the number of stations present in the WLAN.

For the case of EDCA, performance degrades with the number of stations, which is the expected result from the use of a fixed set of (relatively small) contention parameters. However, for $N > 15$ the total throughput performance slightly grows again, a behavior caused by the capture effect as the last nodes to be added in our experiments are the ones experiencing better link qualities (nodes 3 and 6). This is confirmed by the fairness values, as for $N > 15$ there is a drop in the JFI for the case of EDCA. JFI values also confirm that DAC is more sensitive to heterogeneous link conditions, as its performance noticeably degrades with N. In contrast, with CAC the fairness index is practically constant for all N values.

5.5 TCP Throughput

We next evaluate performance in scenarios in which stations use TCP. We start by evaluating the throughput and fairness performance when all stations are constantly backlogged sending TCP traffic to the AP, replicating *bulky* FTP transfers. Note that this scenario is substantially different from the ones considered in the previous subsections, as TCP congestion control[10] introduces a "closed loop" that can lead to extreme unfairness conditions and even starvation [20].

We plot in Fig. 14 the total throughput values for the three mechanisms. According to the results, both CAC and DAC significantly outperform EDCA, improving throughput by 50% and 40%, respectively.

[10]The Linux distribution used in our deployment executes the TCP CUBIC variant [19].

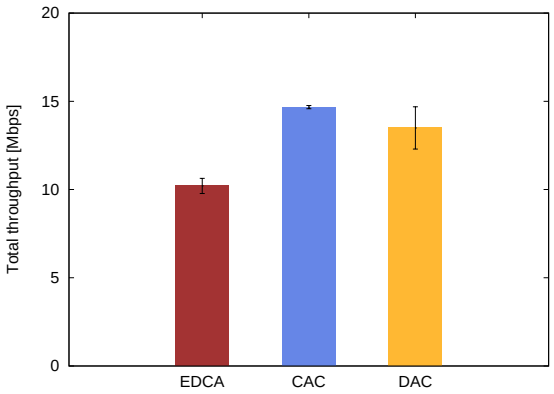

Figure 14: Total throughput w/ FTP-like traffic.

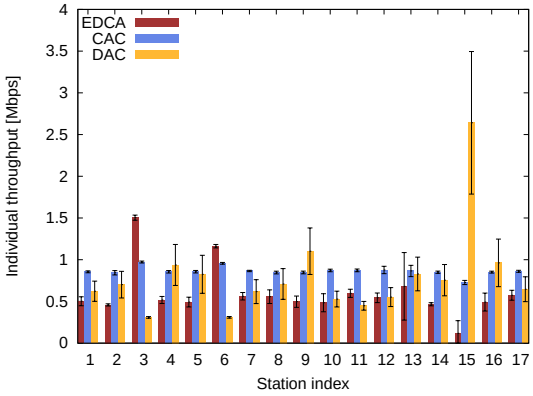

Figure 15: Throughput per station w/ FTP-like traffic.

The per-station throughput distribution is depicted in Fig. 15. With EDCA, the node with the poorest link quality (node 15) suffers from a large performance degradation, this being worse than in the UDP case (see Fig. 10). The use of DAC with TCP traffic also exacerbates the unevenness in the traffic distribution, with node 15 clearly outstanding among the other nodes. DAC results also present a large deviation, caused by relatively frequent TCP timeouts from nodes with weak radio link (e.g., node 15). Conversely, CAC yields a remarkably fair and stable throughput distribution.

Like in the UDP case, we compute the JFI values for the resulting throughput distributions. In this case, the values for EDCA, CAC and DAC are 0.787, 0.996 and 0.692, respectively. We conclude that, as expected, the performance of EDCA and DAC worsens with TCP, while CAC preserves its good properties in this scenario.

5.6 TCP Transfer Delay

We finally consider a scenario involving finite-size TCP connections. More specifically, all stations alternate periods of activity—during which a transmission of 10 MB occurs—with silent periods exponentially distributed with mean λ^{-1} [21]. We consider three different values for λ, corresponding to three different levels of activity, namely high, moderate and low. For each case we ran 1-hour experiments, logging all transfer durations and computing the per-station average delay. We use a *box-and-whisker* diagram to illustrate the distribution of the average delay among nodes: we provide the median, first and third quartiles of the average delay, as well as its maximum and minimum values.

Results are depicted in Fig. 16. With $\lambda^{-1} = 30\ s$, which corresponds to high activity, we see that CAC provides the smallest and most uniform distribution of transfer delay among nodes, with practically no difference between the best and worst performing node. In case of EDCA, the delay shows a larger median and higher variability. However, the small distance between the first and third quartiles shows that most of the stations experience similar performance. Finally, for the case of DAC, despite the median is similar to the one of CAC, results show a much larger dispersion.

When the traffic activity is moderate ($\lambda^{-1} = 60\ s$), the absolute values decrease, but the relative results are similar, i.e., CAC provides again the smallest and most uniform delays among nodes. Finally, when the activity of the nodes is low ($\lambda^{-1} = 90\ s$), medians are very similar but still CAC provides the most fair distribution of the transfer delays. From these experiments, we conclude that CAC also provides the best performance under dynamic traffic scenarios.

6 Related Work

The scientific literature offers many examples of MAC optimization approaches. Many of them are based on a centralized entity, responsible for monitoring system performance and adapting the system parameters to current conditions. Other works focus on distributed approaches to adapt MAC parameters. Very

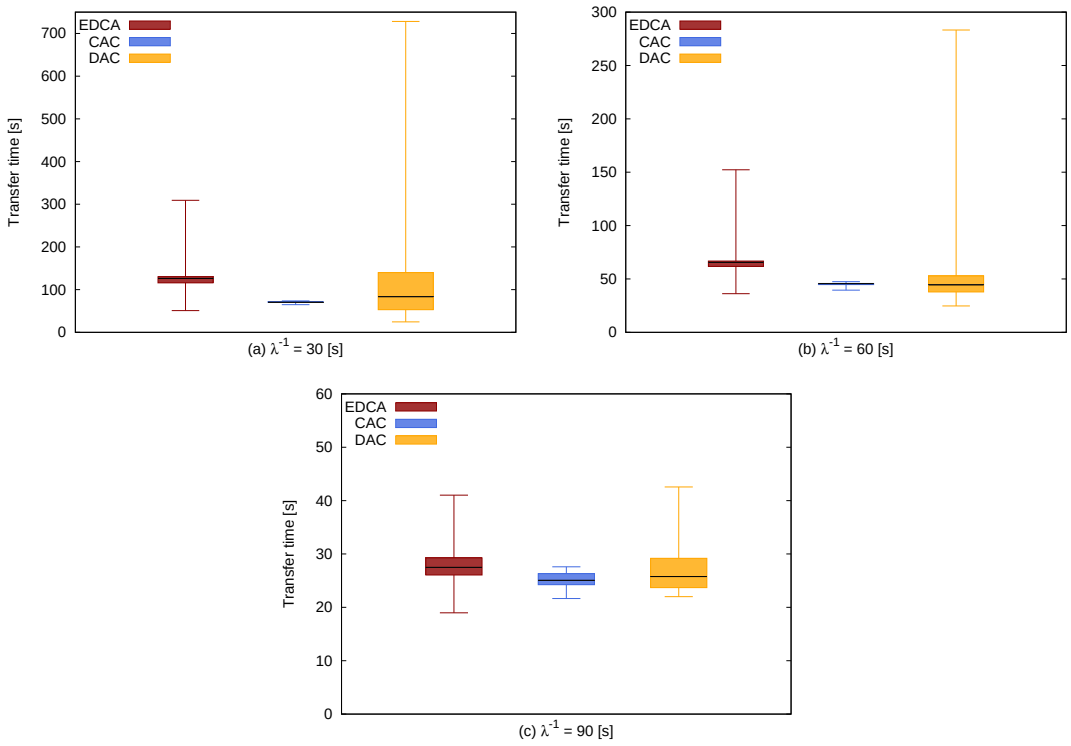

Figure 16: TCP delay performance.

little experimental work is available, and it is based on complex algorithms, non-standard functionality and small-sized networks. In the following we review the most significant contributions in each of these areas and describe the novelty of our work.

Centralized approaches. A significant number of approaches exists in the literature [3, 5, 12, 14] that use a single node to compute the set of MAC parameters to be used in the WLAN. With the exception of our CAC algorithm [14], the main drawbacks of these approaches are that they are either based on heuristics, thereby lacking analytical support for providing performance guarantees [3, 5], or they do not consider the dynamics of the WLAN under realistic scenarios [12].

Distributed approaches. Several works [6, 8, 11, 22, 23] have proposed mechanisms that independently adjust the backoff operation of each stations in the WLAN. The main disadvantages of these approaches are that they change the rules of the IEEE 802.11 standard and therefore require introducing significant hardware or firm-ware modifications.

Implementation experiences. Very few schemes to optimize WLAN performance have been developed in practice [12,13,24]. While the idea behind Idle Sense [11] is fairly simple, its implementation [24] entails a significant level of complexity, introducing tight timing constraints that require programming at the firmware level. The same limitation holds for the approach of [13], which introduces changes to the MAC protocol that require redesigning of the whole NIC implementation. Finally, the work of [12] does not propose or evaluate any adaptive algorithm to adapt the CW but just evaluates the performance of static configurations. Additionally, all of these works rely on testbeds substantially smaller than ours.

7 Conclusions

We have prototyped with standard 802.11 devices two adaptive mechanisms that tune the contention window based on the observed network conditions. In contrast to other proposals that require complex modifications, these mechanisms rely on functionalities already supported by COTS hardware/firmware, and do not introduce any extensions to the standard 802.11 MAC. We have extensively evaluated the performance of the mechanisms in an 18-nodes testbed, considering a large variety of network conditions.

With our experimental study we have identified the key limitations of the distributed scheme, inherent in realistic scenarios, and we have confirmed that the centralized mechanism significantly improves network throughput, transfer delay and fairness among stations in a broad range of circumstances, including the pathological case of hidden nodes. A major conclusion from our work is that, by simply adding a few lines of code at the AP to exploit the functionality readily available, we can achieve performance improvements of up to 50%. We believe that the results presented herein pave the way for a widespread deployment of the centralized mechanism.

References

[1] IEEE 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. Revision of IEEE Std 802.11-1999, 2007.

[2] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 535–547, Mar 2000.

[3] J. Freitag, N. L. S. da Fonseca, and J. F. de Rezende, "Tuning of 802.11e Network Parameters," *IEEE Communications Letters*, vol. 10, no. 8, pp. 611–613, 2006.

[4] L. Scalia, I. Tinnirello, J. Tantra, and C. H. Foh, "Dynamic MAC Parameters Configuration for Performance Optimization in 802.11e Networks," in *Proc. GLOBECOM*, (San Francisco, CA, USA), pp. 1–6, Dec 2006.

[5] A. Nafaa, A. Ksentini, A. A. Mehaoua, B. Ishibashi, Y. Iraqi, and R. Boutaba, "Sliding Contention Window (SCW): Towards Backoff Range-Based Service Differentiation over IEEE 802.11 Wireless LAN Networks," *IEEE Network*, vol. 19, pp. 45–51, Jul 2005.

[6] Y. Yang, J. J. Wang, and R. Kravets, "Distributed Optimal Contention Window Control for Elastic Traffic in Single-Cell Wireless LANs ," *IEEE Transactions on Networking*, vol. 15, pp. 1373–1386, Dec 2007.

[7] Q. Xia and M. Hamdi, "Contention Window Adjustment for IEEE 802.11 WLANs: A Control-Theoretic Approach," in *Proc. International Conference on Computer Communications (ICC)*, (Istanbul, Turkey), Jun 2006.

[8] Q. Ni, I. Aad, C. Barakat, and T. Turletti, "Modeling and Analysis of Slow CW Decrease for IEEE 802.11 WLAN," in *Proc. IEEE Personal, Indoor, and Mobile Radio Communications Conference (PIMRC)*, (Beijing), 2003.

[9] L. Chen, S. H. Low, and J. C. Doyle, "Joint congestion control and media access control design for wireless ad hoc networks," in *Proc. IEEE INFOCOM*, (Miami, Florida), march 2005.

[10] L. Chen, S. H. Low, and J. C. Doyle, "Random access game and medium access control desig," *IEEE Transactions on Networking*, vol. 18, pp. 1063–6692, dec. 2010.

[11] M. Heusse, F. Rousseau, R. Guillier, and A. Duda, "Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless LANs," in *Proc. ACM SIGCOMM*, (Philadelphia, PA, USA), pp. 121–132, 2005.

[12] V. A. Siris and G. Stamatakis, "Optimal CWmin selection for achieving proportional fairness in multi-rate 802.11e WLANs: test-bed implementation and evaluation," in *Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization*, (Los Angeles, CA, USA), pp. 41–48, 2006.

[13] R. Bernasconi, S. Giordano, A. Puiatti, R. Bruno, and E. Gregori, "Design and Implementation of an Enhanced 802.11 MAC Architecture for Single-Hop Wireless Networks," *EURASIP Journal on Wireless Communications and Networking*, 2007.

[14] P. Patras, A. Banchs, and P. Serrano, "A Control Theoretic Approach for Throughput Optimization in IEEE 802.11e EDCA WLANs," *Mobile Networks and Applications (MONET)*, vol. 14, pp. 697–708, Dec 2009.

[15] P. Patras, A. Banchs, P. Serrano, and A. Azcorra, "A Control Theoretic Approach to Distributed Optimal Configuration of 802.11 WLANs," *IEEE Transactions on Mobile Computing*, vol. 10, pp. 897–910, Jun 2011.

[16] IEEE 802.11, *Supplement to Wireless LAN Medium Access Control and Physical Layer specifications: high-speed physical layer in the 5 GHz band*. IEEE Std 802.11a, 1999.

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

[18] R. Jain, Chiu, D.M., and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems," *DEC Research Report TR-301*, 1984.

[19] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, pp. 64–74, July 2008.

[20] O. Gurewitz, V. Mancuso, J. Shi, and E. Knightly, "Measurement and modeling of the origins of starvation of congestion-controlled flows in wireless mesh networks," *IEEE Transactions on Networking*, vol. 17, dec. 2009.

[21] P. Barford and M. E. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proceedings of ACM SIGMETRICS*, pp. 151–169, 1998.

[22] L. Bononi, M. Conti, and E. Gregori, "Runtime optimization of ieee 802.11 wireless lans performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, pp. 66–80, Jan. 2004.

[23] F. Cali, M. Conti, and E. Gregori, "IEEE 802.11 protocol: design and performance evaluation of an adaptive backoff mechanism," *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 1774–1786, Sept. 2000.

[24] Y. Grunenberger, M. Heusse, F. Rousseau, and A. Duda, "Experience with an implementation of the Idle Sense wireless access method," in *Proceedings of the ACM CoNEXT conference*, (New York, New York), pp. 1–12, 2007.