Computer Communications 34 (2011) 1283–1293

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A CAPWAP-based solution for frequency planning in large scale networks of WiFi Hot-Spots

M. Bernaschi [a], F. Cacace [b], A. Davoli [c], D. Guerri [c], M. Latini [a], L. Vollero [b],*

[a] Istituto Applicazioni del Calcolo-CNR, Rome, Italy
[b] Università Campus Bio-Medico, Rome, Italy
[c] CASPUR, Rome, Italy

ARTICLE INFO

Article history:
Received 4 November 2009
Received in revised form 30 December 2010
Accepted 5 January 2011
Available online 7 January 2011

Keywords:
CAPWAP
Wireless networks
Network management
Frequency planning

ABSTRACT

We present the results of the experimental work we carried out to test a solution for frequency planning developed for the *Provincia di Roma* network of WiFi Hot-Spots. This work is both an example of how the Control And Provisioning of Wireless Access Points (CAPWAP) protocol may help in addressing the issues that arise in the deployment and control of large scale, possibly heterogeneous, wireless networks and a real-world test of our open source implementation of the CAPWAP protocol. Simulations and experimental tests confirm that the proposed technical solution is effective in improving network performance.

1. Introduction

Large deployments of wireless Access Points (AP) pose serious problems in defining consistent strategies for their management, configuration and control. These issues forced network vendors to propose proprietary centralized solutions aimed at simplifying the administration of wireless networks. All proposed solutions share two common elements: (i) they split functionalities that APs provide and (ii) add more centralized functions for the monitoring and the remote control of the network. By splitting functionalities of APs, more flexible network infrastructures can be implemented. Indeed, such separation allows to centralize the management of critical functions like channel selection, authentication and encryption. Whereas, leaving in the APs time-critical functions, like beacon generation and frames' acknowledgment, avoids the introduction of expensive components, like high performance interconnections, making proposed solutions competitive on the market.

The Internet Engineering Task Force recently recognized the importance of the interoperability in this field and started a Working Group, named Control and Provisioning of Wireless Access Points (CAPWAP), with the goal of defining standard solutions to such issues. The CAPWAP WG focused on problems like configuration, monitoring, control and management of large scale deployments of wireless networks in general, and of IEEE 802.11 networks (O'Hara et al. [8]) in particular, by identifying a number of functions that should be provided in such scenarios. The WG has defined a protocol, the CAPWAP protocol, capable of providing interoperability among devices supporting these functions. In a recent paper (Bernaschi et al. [3]), we presented our open source implementation of the CAPWAP protocol based on Calhoun et al. [1] and Calhoun et al. [2].

Here we present a real-world application of the CAPWAP protocol, that is a solution for frequency planning in a large-scale network of wireless Access Points distributed across the city of Rome (Italy) and its *Provincia* (a surrounding area roughly equivalent to a county). The application represents both an architecture and a technical solution with very few and limited predecessors. Along with its description, we present simulations and experimental results obtained by testing the solution on real devices in a number of different configurations and conditions.

The rest of the paper is organized as follows: Section 2 describes the CAPWAP protocol, its functionalities for the management of IEEE 802.11 networks and some recent additions and enhancements to *OpenCAPWAP*, our open source implementation; Section 3 describes the main features of the *Provincia di Roma* WiFi network; Section 4 describes the solution we developed for frequency planning of the *Provincia di Roma* WiFi network along with simulations and performance results on real devices. Finally, Section 5 concludes the paper with the future perspectives of this activity.

2. The CAPWAP protocol

The Control And Provisioning of Wireless Access Points (CAPWAP, Calhoun et al. [1]) is a recent effort of IETF aiming at defining

* Corresponding author. Tel.: +39 06225419631.
E-mail address: vollero@ieee.org (L. Vollero).

0140-3664/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2011.01.002

an interoperable protocol, enabling a single point of control, called Access Controller (AC) to manage and control a collection of possibly heterogeneous Wireless Termination Points (WTPs). Although originating from IEEE 802.11 architectures, the CAPWAP specifications aim at being independent of a specific WTP radio technology. The goals explicitly stated in current specifications of CAPWAP are the following:

(i) centralize authentication and policy enforcement functions for a wireless network;
(ii) move processing away from the WTPs, leaving there only time critical functions and
(iii) provide a generic encapsulation and transport mechanism.

Currently the CAPWAP protocol defines the communication and general management functions among AC and WTPs. CAPWAP control and data messages are sent by using UDP, over separate UDP ports, and secured by using Datagram Transport Layer Security (DTLS, Dierks et al. [7]). The CAPWAP protocol transport layer introduces resiliency by using a request/response paradigm, where timeouts schedule retransmissions when a response does not follow a certain request. Two types of payload may be managed by this transport protocol: CAPWAP data messages and CAPWAP control messages. CAPWAP data messages encapsulate wireless frames forwarded by the WTP to the AC or by the AC to the WTP. CAPWAP control messages are CAPWAP management, control or monitoring messages exchanged among AC and WTPs. CAPWAP also defines a discovery protocol for the automatic association of WTPs to the AC.

The implementation of CAPWAP for a specific wireless technology is called "binding". For instance, (Calhoun et al. [2]) defines the binding for IEEE 802.11 WLANs. Specifically, (Calhoun et al. [2]) devises two operational architectures: Split MAC (SM) and Local MAC (LM). The difference between the two architectures is based on where specific functionalities are implemented: in the WTP, in the AC or in both. In both architectures, real-time IEEE 802.11 services, including beacons generation and probe responses, are implemented on the WTP (see *OpenCapwap* [9] for further details). CAPWAP control messages for IEEE 802.11 are enriched by the introduction of specific information for radio control, management and monitoring. Moreover, special control frames for the Quality of Service management using Wireless Multimedia (WMM) extensions are defined in Calhoun et al. [2].

It is important to understand that CAPWAP, albeit based on the same transport protocol (UDP) and aimed at the same goal (the management of network devices), is pretty different with respect to the Simple Network Management Protocol (SNMP). The SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. SNMP is more general than CAPWAP. Most of the professional grade network devices include an SNMP Agent that need to be enabled and configured to communicate with the Network Management System (NMS). There is a standard set of statistical and control values defined for the managed devices on a network, however the extension of these standard values with values specific to a particular device is, notoriously, a painful process. CAPWAP is, by definition, restricted to the control of wireless Access Points but for these devices it offers, independently from the wireless technology, a number of specific features and a generic encapsulation and transport mechanism that make easier to extend its functionality as we describe in Section 2.1.

2.1. Protocol implementation and support for external application

The state diagram reported in Fig. 1 represents the lifecycle of a WTP-AC session with a Finite State Machine (FSM), as defined in

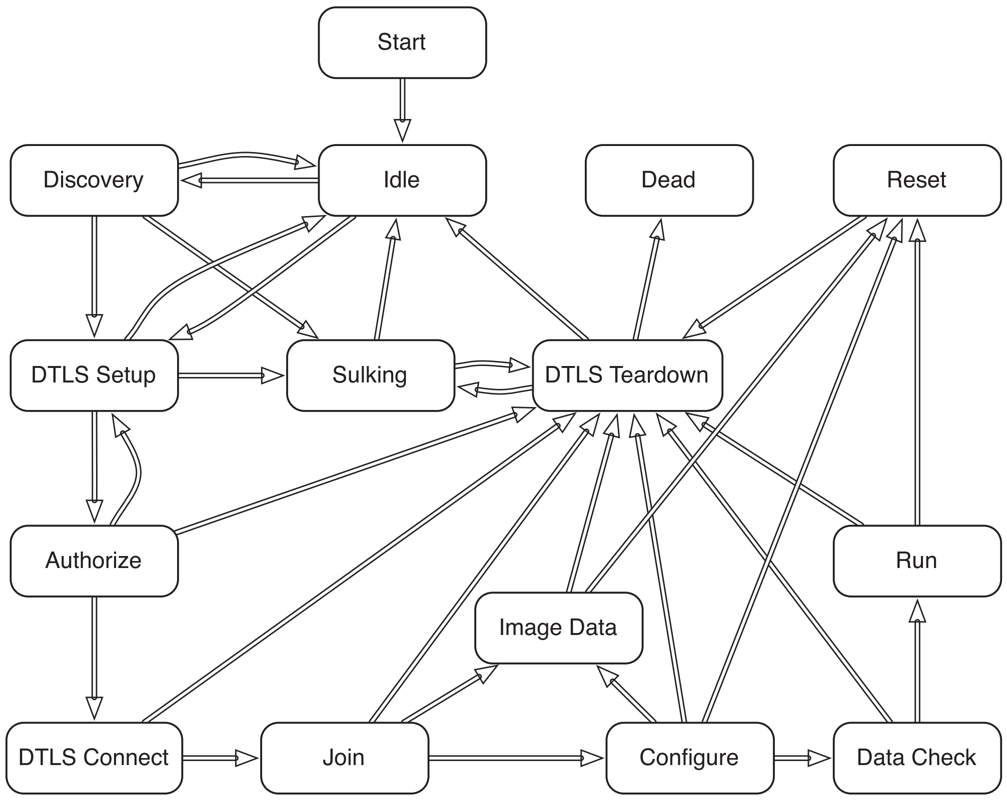

Fig. 1. CAPWAP finite state machine.

the protocol specification (Calhoun et al. [1]). Use of DTLS by the CAPWAP protocol results in the juxtaposition of two nominally separate yet tightly bound state machines. The DTLS and CAPWAP state machines are coupled through an API consisting of commands and notifications. Certain transitions in the DTLS state machine are triggered by commands from the CAPWAP state machine, while certain transitions in the CAPWAP state machine are triggered by notifications from the DTLS state machine.

Our implementation of the CAPWAP protocol consists of two multi-thread applications, one running on the ACs and one running on the WTPs. Each WTP acts as a client of an AC. Each AC manages both the communication with WTPs already registered and new requests sent by WTPs not-yet registered. A much more detailed description of our original implementation of the CAPWAP protocol can be found in [3]. In the rest of this section we describe our more recent work aimed at extending the possibilities of the basic CAPWAP protocol.

Our initial implementations of *OpenCapwap* did not offer support for generic external applications that needed CAPWAP protocol functionalities. A mechanism to address such issue has been added to the latest, *OpenCapwap 0.92* at the time of this writing, implementation. This mechanism is based on an application management server added to the *OpenCapwap* AC daemon. Hereafter we describe how such support works and how an application interacts with *OpenCapwap*'s AC daemon to send and receive data not strictly related to the CAPWAP protocol itself.

As Fig. 2 shows, in the basic architecture of *OpenCapwap*, there are four main components:

- *AC daemon*: it implements CAPWAP Access Control functionalities and manages external applications via a dedicated thread and a TCP server socket. The AC and external applications exchange data by following a simple textual protocol (described in Section 2.1.1).
- *WTP daemon*: it implements CAPWAP Wireless Termination Point functionalities and interacts with external applications by sending and receiving data after the CAPWAP protocol messages have been unpacked (by reading and removing CAPWAP specific headers). The applications hereafter presented use a UDP socket to this purpose.
- *External application (AC Side)*: by using the external applications management protocol, it can retrieve the list of WTPs associated to the AC (specified by the WTP id number and WTP name), send data to a specific WTP (addressed by means of its WTP id or WTP name) or to groups of the associated WTPs and receive data from WTPs to which data were sent.
- *External application (WTP Side)*: it exchanges data with *OpenCapwap*'s WTP daemon by using a UDP socket.

2.1.1. AC architecture

Fig. 3 shows the new architecture of the AC. With respect to the previous version of *OpenCapwap*, an additional thread, named *Application Management Thread (AMT)*, is present. The AMT listens for incoming requests on a *stream* (TCP) socket so that external applications, after having established a connection with this socket, can send commands like:

- *LIST_MSG*: on response to this message the external application receives the list of WTPs currently associated to the AC.
- *CONF_UPDATE_MSG*: a *custom* data payload can be sent by using this command. It uses the *Configuration Update Request* and the *Configuration Update Response* CAPWAP messages to send data back and forth between the AC and one or more WTPs. External applications may use the AC as a relay to reach any

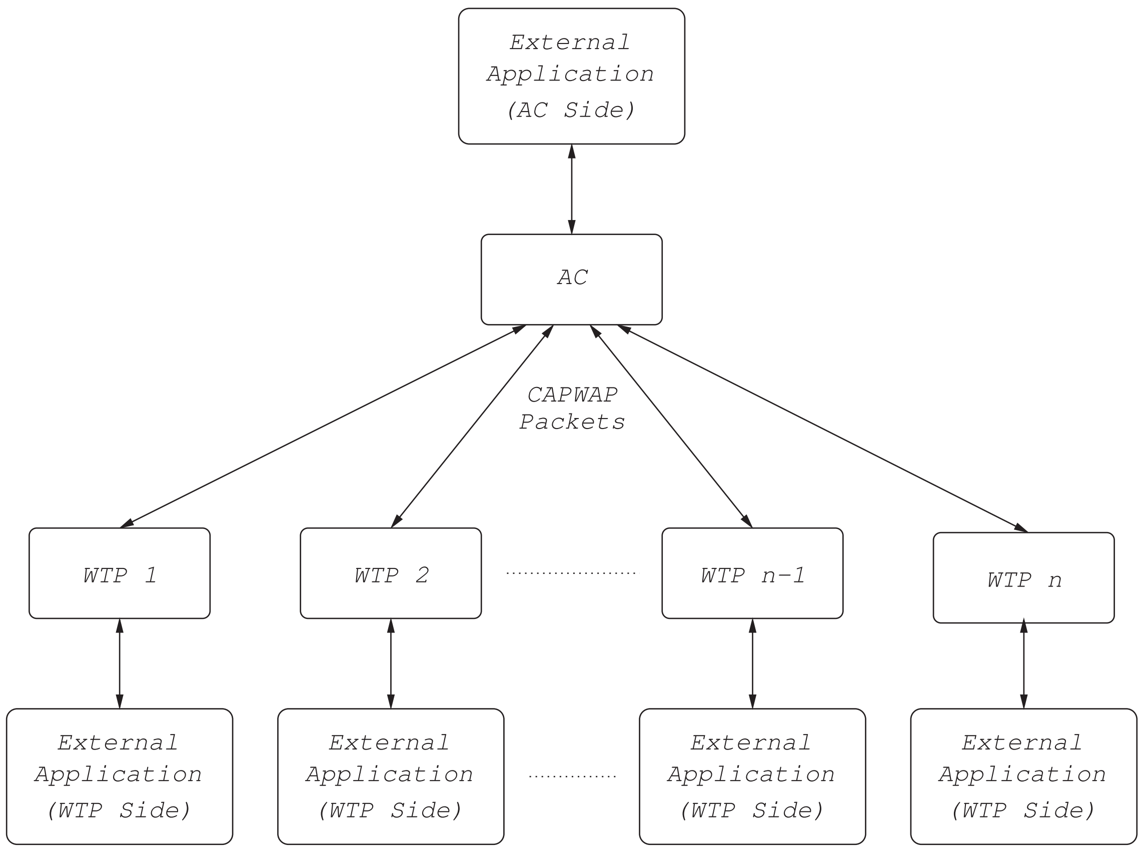
External Application (AC Side)
AC
CAPWAP Packets
WTP 1
WTP 2
WTP n-1
WTP n
External Application (WTP Side)
External Application (WTP Side)
External Application (WTP Side)
External Application (WTP Side)

Fig. 2. *OpenCapwap* 0.92 general architecture.

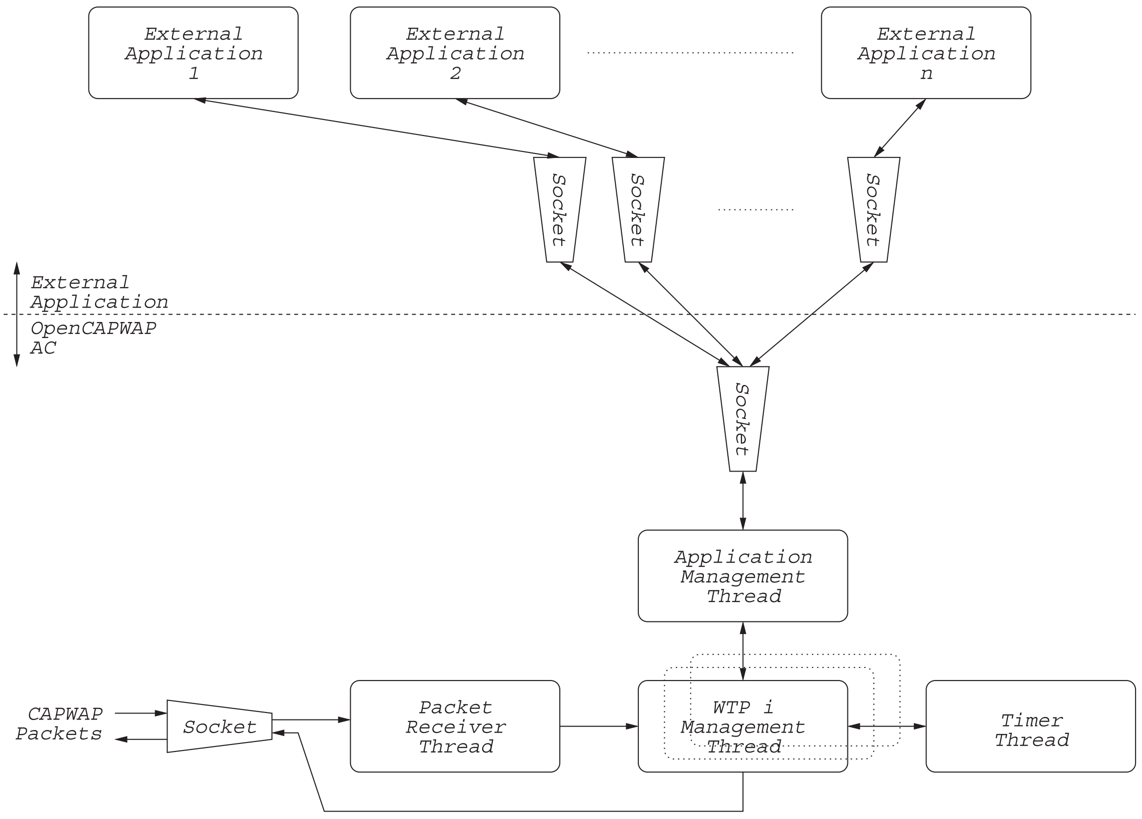

Fig. 3. *OpenCapwap* 0.92 AC architecture.

WTP by sending a message made up by: (i) the index representing the type of *Configuration Update* message to be sent; (ii) the WTP identifier to which the payload must be sent to (or −1 to address all associated WTPs) followed by (iii) the data payload itself.

2.1.2. WTP architecture

For the WTP daemon, we resorted to the simple architecture presented in Fig. 4. A datagram (UDP) socket is added to the main WTP thread. During the processing of *Configuration Update Request* messages, the data payload is handed through this socket to the external application (if any). The external application (WTP side) can then process such data payload and send a response back to the external application by relaying data through the WTP daemon (which forwards them to the AC daemon in a *Configuration Update Response* message).

2.1.3. Sending commands to groups of Hot-Spots

As an example of usage of this architecture, we describe our solution for sending commands to groups of Hot-Spots that make use of *OpenCapwap* and *OpenWrt*, a GNU/Linux distribution for low-powered, limited-hardware generic Access Points and more in general for embedded devices (OpenWRT [6]).

The solution is based on two applications, the *Uci Server* (WTP side) and the *Remote Uci* (AC side) that allow to configure WTPs by using *OpenWrt*'s *Unified Command Interface* (UCI). The goal of the

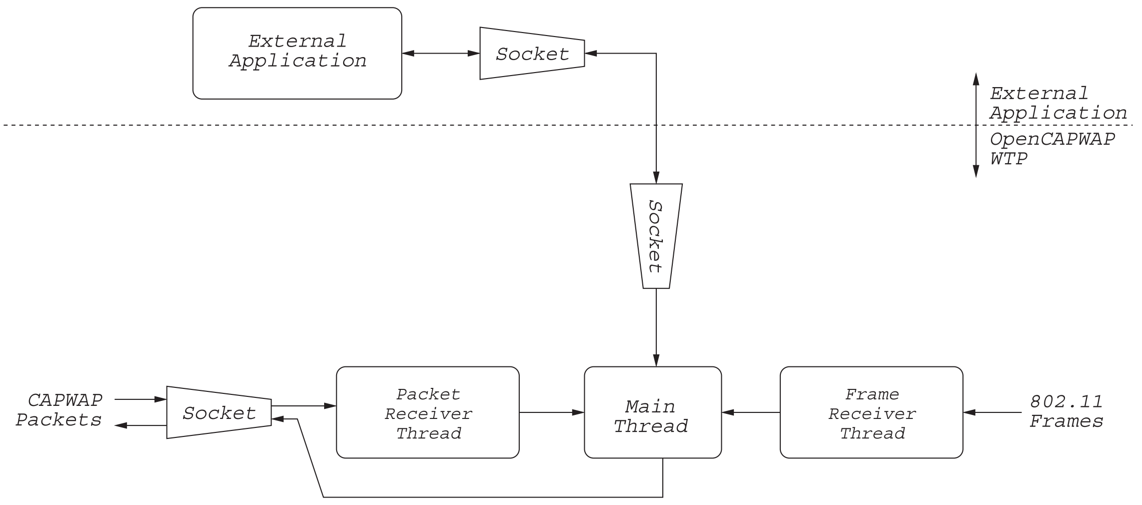

Fig. 4. *OpenCapwap* 0.92 WTP architecture.

UCI is to manage the administration of an entire *OpenWrt* system from a single command line utility while providing many advanced features such as custom configurations and configuration backup/restore. Our goal was to bring UCI's ease of use to the AC daemon.

To that purpose, the architecture of Fig. 2 becomes like that shown in Fig. 5 where *Remote Uci* is a command line utility (written in Python 2.5) interacting with *OpenCapwap*'s AC daemon via the external application management mechanism presented above. *Remote Uci* implements all the main commands provided by *OpenWrt*'s UCI plus some custom commands:

- *wtps*: lists WTPs associated with the AC to which *Remote Uci* is connected. This command is independent from UCI and it does not generate any CAPWAP message but only data traffic between *Remote Uci* and *OpenCapwap*'s AC daemon.
- *show*: shows the current UCI configuration.
- *set*: adds or modifies part of the UCI configuration.
- *delete*: deletes part of the UCI configuration.
- *commit*: permanently saves changes made to an UCI configuration. Changes to the configuration are saved through a complete restart of the services for which the configuration has been changed. The autonomous restart of *OpenWrt*'s services was not supported by UCI but is part of the extended functionalities offered by our *Uci Server*.
- *commitwithreboot*: permanently saves changes made to the UCI configuration. Changes to the configuration are saved by means of a complete *OpenWrt* reboot. This command was not supported by the original UCI and has been implemented from the ground up.
- *defaultrevert*: restores a default fail-safe configuration (to which *Uci Server* must be pointed to) and triggers a complete *OpenWrt* reboot. Also this command was not supported by the original UCI.

All of the above commands, along with those of the original UCI, are executed remotely from the AC on a single WTP or on a subset of WTPs associated with the AC.

The data payloads that make this remote configuration mechanism possible are rather simple. They are composed by an id that identifies any of the above commands, the length of the command arguments to be appended upon execution and the actual string of command arguments. This simple payload is then handed to the AC daemon through the TCP socket discussed above by encapsulating the data in a *Configuration Update Request* message.

The WTP side counterpart of *Remote Uci* is *Uci Server*, a lightweight daemon (written in C) that interacts with *OpenCapwap*'s WTP and *OpenWrt*'s UCI. When a *Configuration Update Request* containing a specific data payload reaches a WTP, the WTP daemon, after removing the CAPWAP header, hands the data payload to the local *Uci Server* through an UDP socket. The *Uci Server* then executes the requested UCI command and signals to the WTP daemon, through the same UDP socket, the exit status of the command. Additionally, *Uci Server* can also send back the output of the command executed. The WTP daemon then builds a new *Configuration*

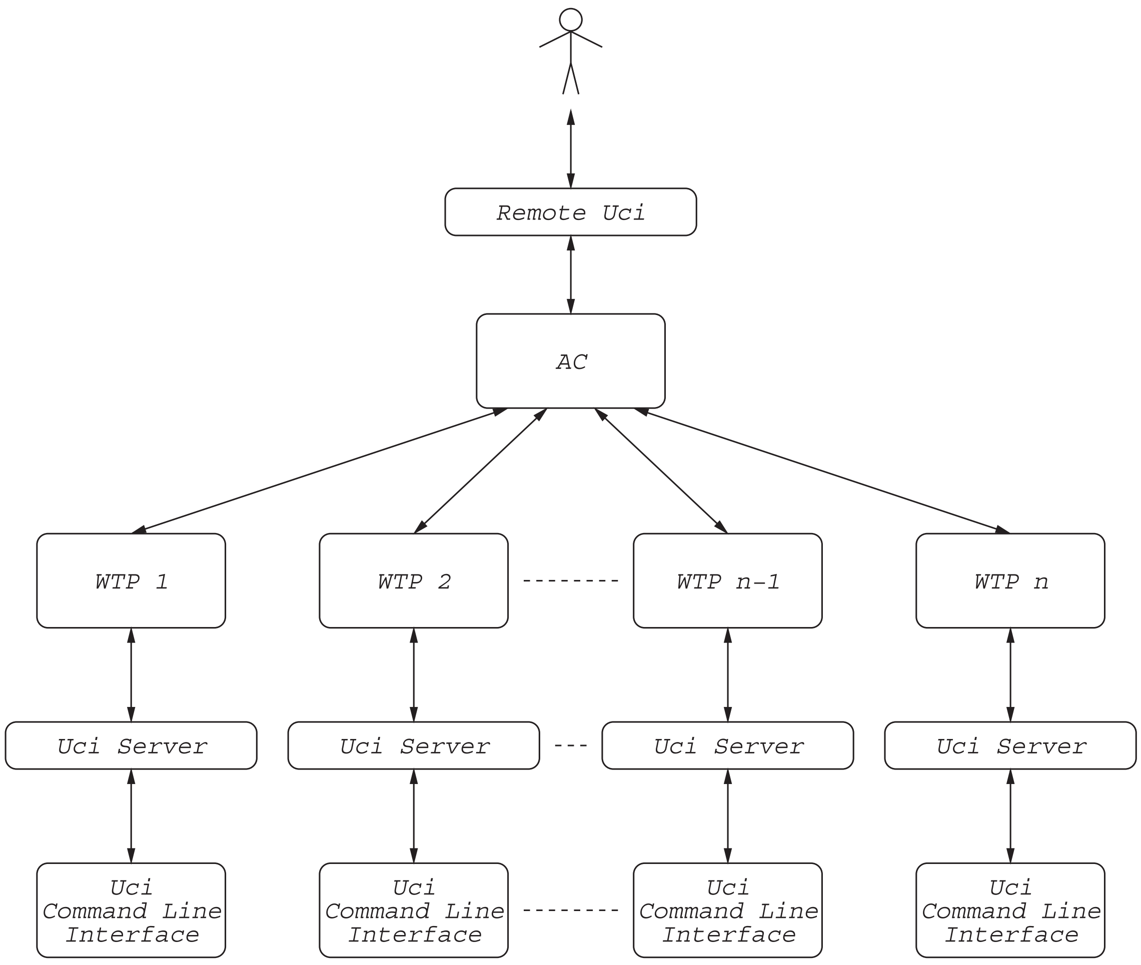
Remote Uci
AC
WTP 1
WTP 2
WTP n-1
WTP n
Uci Server
Uci Server
Uci Server
Uci Server
Uci Command Line Interface
Uci Command Line Interface
Uci Command Line Interface
Uci Command Line Interface

Fig. 5. Example application architecture.

Update Response that contains the data payload made up by the exit status and the output of the UCI command executed (if any). In such a way an administrator can operate on many WTPs by issuing various UCI commands and receiving the corresponding output from the WTPs to which those commands have been issued.

Despite being a pretty simple daemon, *Uci Server* implements a quite sophisticated mechanism that prevents an administrator from saving erroneous or corrupted UCI configurations that could result in one or more unreachable WTPs. This functionality is available with the commands **commit** and **commitwithreboot**, which are those that actually save any change to an UCI configuration. All these software components are part of the current *OpenCapwap* distribution.

3. The *Provincia di Roma* WiFi network

The *Provincia di Roma* WiFi network, developed and maintained by CASPUR InterUniversity Consortium, is a project of the *Provincia* surrounding and including the city of Rome (Italy). *Provincia di Roma* includes about 120 cities with more than 4 million of inhabitants and it has an extension of about 5000 Km^2.

In December 2009 the network consisted of more than 150 Access Points, while within the end of 2010, at least 500 Access Points will be available in public spots, like parks, libraries, government buildings. The access to the network is granted after a registration procedure (that is required only once) based on the exchange of SMS messages. The ESSID is the same for all Access Points (*provinciawifi*). The user provides the credentials received during the registration procedure through a captive portal.

Each Access Point sets up a Virtual Private Network with a concentrator, running GNU/Linux, which hosts multiple instances of pfSense (pfSense [4]) through the KVM (KVM [5]) virtualization hyper-visor. Each pfSense instance is connected to the network backbone and provides, among other services, Internet connectivity to the users.

All data required for the administration (*e.g.*, the stock list of the devices in use, users registration, *etc.*) and data collected during the operations (*e.g.*, syslog, firewall logs, *etc.*) are managed by using a MySQL DBMS. An application based on the Google Maps API is used to show, via browser, the availability status of the Hot-Spots.

4. A CAPWAP based solution for frequency planning

IEEE 802.11b defines a number (11, 13 or 14) of possible transmission channels for wireless communications, depending on national regulations, but at most three of those channels can be used simultaneously without cross-interference (namely Channel 1, Channel 6 and Channel 11). When configuring or upgrading large deployments of WTPs, the configuration of frequencies used by every WTP may be a major problem in optimizing overall network performance. Optimal or sub-optimal frequencies reuse is desirable to reduce interference among adjacent cells, but this is not always possible.

4.1. Related work on frequency planning

There are many proposals in the literature to deal with the problem of frequencies reuse in wireless networks. They can be grouped in two main categories:

- approaches that aims at minimizing the overall interference by using linear programming techniques. The problem is formulated starting from the interference level at the WTPs (Akl et al. [11]) or at the single mobile stations (Haidar et al. [12], Al-Rizzo et al. [13], Lee et al. [14]). These approaches require intensive computations, due to the complexity of the constraints, and are therefore suitable to static situations, where the frequency assignment is computed only once or at large intervals.
- approaches that model the problem as a graph coloring problem which is solved through some heuristic method (Mishra et al. [15–17]). In this case, a sub-optimal solution is generally considered acceptable. The reduced computational complexity of these methods makes them more suited to dynamical situations where the frequency assignment needs to be modified more frequently.

A method for the centralized management of channel allocation should be capable of (i) taking advantage of the centralized nature of the wireless LAN; (ii) dealing with dynamic situations, with wide variations of stations distribution and throughput generated at each WTP; (iii) being aware of the presence of external APs operating in the managed area but that are not controlled by the AC.

The third requirement adds additional constraints and it is not uncommon due to the widespread diffusion of wireless home or personal networks. In a publicly operated and geographically sparse network, as the *Provincia di Roma WiFi* network, it is mandatory to consider external sources of interference, whereas this may not be the case, for example, in a corporate WLAN. Most existing proposals, however, focus on the case in which all the WTPs are under the control of the manager of a centralized network (Levanti et al. [10], Lee et al. [14], Arunesh et al. [18], Chandra et al. [19], Liu et al. [20]).

A possible approach to the centralized management of channel allocation based on the CAPWAP protocol is presented in (Levanti et al. [10]). It implements a variant of the algorithm proposed in (Arunesh et al. [18]). The authors conclude that the exploitation of a centralized management algorithm can avoid not only the transient channel adjustments due to subsequent and independent WTP decisions, but also the so-called *blocked cell problem*, thanks to the complete knowledge of the network topology. The blocked-cell phenomenon happens when a WTP is located in proximity of a set of other WTPs that use the three orthogonal channels, a situation not unrealistic in practice. In these conditions, if the adjacent WTPs do not hear each other, it is very likely that their transmissions are operated asynchronously and, in presence of saturated traffic, the considered WTP would sense the channel busy all the time. The blocked-cell effect can reduce the throughput of the affected cell to zero, with the WTP waiting indefinitely for channel release. In the scheme proposed by (Levanti et al. [10]), each WTP periodically scans the channels in order to know all the potential interferences. Once the scanning is over, each WTP sends a report to the Access Controller indicating the identifiers of the interfering WTPs and the reception power levels. On the basis of this information, the centralized manager running on the AC assigns a channel to the WTPs, while minimizing the total reciprocal interference in the network. Since the number of orthogonal channels may not be sufficient for covering the interfering cells, the planning algorithm should try to assign orthogonal channels to the subset of interfering cells which result most loaded, while trying to avoid the blocked-cell phenomenon.

4.2. The iFP solution for frequency planning

The *iFP* algorithm that we present here is compliant with the above requirements, and in particular:

- it has low computational complexity, thus allowing to cope well with dynamic situations;

- it is well suited for large deployments of WTPs that are sparsely distributed in large areas and are affected by the interference of WTPs belonging to other networks.

iFP extends the approach of (Liu et al. [20]) in two directions: it takes into account environmental interference from outside the centralized WLAN and it introduces a heuristic technique for graph coloring based on the proposal of (Bhowmick et al. [21]) when several APs needs to be managed together (APs' cluster). However, the main goal of this section is not to highlight the originality or the performance of the *iFP* algorithm in itself. Rather, we aim at validating the centralized management approach to frequency planning in realistic scenarios, that are characterized by a noisy environment, a non trivial network topology and stations mobility. For this reason we compare the performance of *iFP* only with static or local frequency planning techniques, rather than with other centralized methods. It is however worth noting that (i) to the best of our knowledge, no other frequency planning algorithm considers the case of non managed APs operating in the managed area; (ii) for simple network topologies (segregated clusters of up to 4–6 WTPs) the allocation computed by *iFP* is the same as the best possible allocation computed by enumerating the possibilities.

In the sequel, we report results from extensive simulations on the topology of the *Provincia di Roma* WiFi Network, as well as experimental results from a laboratory test-bed. The choice of a test-bed is motivated by the need of measuring the improvement of performance (overall interference and saturation throughput) with respect to the fixed assignment and to the local approach, where each WTP chooses autonomously its own frequency based on local measurements. This would be rather difficult with the *Provincia di Roma WiFi* network in its current state of deployment since the WTPs' and APs' density is still too low to detect reliably the improvement due to the centralized method.

4.3. A centralized algorithm for frequency planning

Let $V = \{a_i\}$ be the set of WTPs of the network, with $i = 1,\ldots,n$. V includes WTPs of the managed network as well as external, non managed APs. Let k be the number of non interfering wireless channels. We say that a mobile station is *active* when, in a specified time interval, it is transmitting or receiving data packets.

Definition 4.1. A channel assignment is a mapping $C : V \to \{1,2,\ldots,k\}$.

Definition 4.2. The signal level function is a mapping $P : V \times V \to \mathbb{N}^0$, such that $P(a_i,a_j) = P(a_j,a_i)$ and $P(a_i,a_i) = 0$.

The function $P(\cdot,\cdot)$ models the spatial WTPs distribution. It corresponds to the signal strength of the WTP a_j received at the WTP a_i, which depends on the distance between the two WTPs. Our analysis does not consider physical asymmetry in communication channels. This assumption can be easily removed with minimal changes in the following algorithms.

Definition 4.3. The neighborhood $v(a_i)$ of the WTP $a_i \in V$ is $v(a_i) = \{a_j \in V\colon P(a_i,a_j) > 0\}$.

We consider $P(a_i,a_j) = 0$ when $P(a_i,a_j) \leqslant \epsilon_s$, where ϵ_s is the minimum signal power level capable of introducing interferences.

Definition 4.4. An interference graph for a channel assignment is a weighted oriented graph $G_C = (V,E)$, where $(a_i,a_j) \in E$ iff $a_j \in v(a_i)$ and the weight function $w_C(a_i,a_j)$ is

$$w_C(a_i,a_j) = s_i I(C(a_i),C(a_j))P(a_i,a_j) \tag{1}$$

where

- s_i is the number of active mobile stations associated to the WTP a_i.
- $I(c_1,c_2)$ is the interference factor between channel c_1 and c_2.

Notice that the interference graph depends on the channel assignment and that in general $w_C(a_i,a_j) \neq w_C(a_j,a_i)$, since s_i is taken into consideration. The value of $I(c_i,c_j)$ can be assigned for a given PHY layer by means of empirical or theoretical methods. It must be remarked that the assignment algorithm may only modify the channel assignment of the managed WTPs, since the assignment of the remaining APs is fixed.

Definition 4.5. The WTP interference function corresponding to a channel assignment C is

$$I_C(a_i) = \sum_{a_j \in v(a_i)}^{n} w_C(a_i,a_j) \tag{2}$$

Definition 4.6. The network interference function corresponding to a channel assignment C is

$$N_C(V) = \sum_{i=1}^{n} I_C(a_i) \tag{3}$$

The goal of the frequency planning algorithm is to minimize the network interference function N_C under the constraint of fixed channel assignments for non managed APs. The algorithm is shown in Algorithm 1 and has two phases. In the first phase, which is an extension of the method proposed in (Liu et al. [20]) and corresponds to the function *planningHeurCluster* (S_i) illustrated in Algorithm 2, the set of managed WTPs is divided into disjoint clusters according to the neighborhood relationship, and a local optimization is performed on each cluster by ordering the WTPs according to $I_C(a_i)$ and then finding for each managed WTP the channel assignment that minimizes the value $I_C(a_i)$. In the second step, which is based on the backtracking approach proposed in (Bhowmick et al. [21]) and corresponds to the function *planningValidateCluster*$(S,j,|S|,bestValue,bestAssignment)$ of Algorithm 3, an exhaustive search is performed on each cluster, starting from the WTP with the largest $I_C(a_i)$. In order to avoid an exponential growth of the search tree, the algorithm implements a pruning strategy that removes failing branches that surely exceed the interference value generated at the first step or the best temporary interference value obtained during algorithm execution.

Algorithm 1 (*Frequency planning algorithm.*).

```
{$S_1,S_2,\ldots,S_M$ are the connected clusters of the interference
  graph G.}
{Each $S_i$ is an array which contains the WTPs of one cluster.}
{$S_i[1],\ldots,S_i[L]$ are the elements of $S_i$.}
{$S_i[k]$.channel is the channel assigned to WTP k of cluster i.}
for $i = 1$ to M do
  if $S_i$ has just one element then
    $S_i[1]$.channel ← findBestChannel($S_i[1]$)
  else
    clusterInterference ← planningHeurCluster($S_i$)
    $\overline{S} \leftarrow S_i$
    planningValidateCluster($\overline{S}, 1, |S_i|$, clusterInterference, $S_i$)
  end if
end for
```

Algorithm 2 (*Heuristic frequency assignment.*).

```
planningHeurCluster($S$)
$stepIntValue \leftarrow calculateN_C(S)$
$S \leftarrow sortCluster(S)$
$\overline{S} \leftarrow S$
repeat
  $actualIntValue \leftarrow stepIntValue$
  $S \leftarrow \overline{S}$
  for $i = 1$ to length(S) do
    $\overline{S}[i].channel \leftarrow findBestChannel(S[i])$
  end for
  $stepIntValue \leftarrow calculateN_C(\overline{S})$
until $stepIntValue < actualIntValue$
return $stepIntValue$
```

Algorithm 3 (*Exhaustive search with pruning.*).

```
planningValidateCluster($S, j, |S|, bestValue, bestAssignment$)
if $j = |S|$ then
  $interference \leftarrow calculateN_C(S)$
  if $interference < bestValue$ then
    $bestAssignment \leftarrow S$
    $bestValue \leftarrow interference$
  end if
else
  for $c = 1$ to NUM_CHANNELS do
    $partialValue \leftarrow calculateN_C([S[1], S[2], \ldots, S[j-1], c])$
    if $partialValue < bestValue$ then
      $S[j].channel \leftarrow c$
      planningValidateCluster($S, j+1, |S|, bestValue, bestAssignment$)
    end if
  end for
end if
```

The function *findBestChannel*$(\cdot)$ finds the best channel for a WTP, being fixed the configuration of all the other WTPs in the Hot-Spots network. The function *calculateN*$_C(S)$ computes the contribution of cluster S to the network interference. The function *sortCluster*(S) sorts the elements of S from the biggest to the lowest according to the their interference value. Eventually $|S|$ is the cardinality (the number of elements) of cluster S.

4.4. Simulation analysis

The simulations analysis proposed in this section aims at evaluating the interference reduction that LCCS,[1] heuristic (*iFP* without optimal search) and *iFP* configurations obtain with respect to a completely random assignment of channels.

We developed a custom simulator software in C++. The simulator has a module for loading Hot-Spots network scenarios and a module for generating random interfering APs. A clustering module groups up the WTPs in interference sets based on power propagation and radio sensitivity considerations. Assuming a free space propagation model, this translates in a proximity rule: a WTP belongs to a WTPs group if another WTP in the group is closer than about 100 m or if there is a STA in its range which is also in the range (100 m) of another WTP of the group. Three configuration modules implement the configuration algorithms, while a performance evaluation module implements the Hot-Spots network interference evaluation. In the configuration and performance evaluation module we used the same propagation model that we used in the clustering module.

The reference topology, composed of 244 WTPs, is the Provincia di Roma WiFi Network (see Fig. 6). The metric we used in the comparison is the gain G defined as

$$G_s = \frac{Nc_{random}}{Nc_s} \text{ with } s \in \{lccs, heuristic, iFP\}$$

The topology is populated with a random number of interfering APs (uniform distribution in the 80 km $\times$ 80 km area of the Hot-Spots network), with each AP choosing a random transmission channel (uniform distribution in the set $\{1,2,\ldots,11\}$). The APs densities range has been chosen to respect real data collected in the network. The number of stations connected to each WTP is also a uniform random parameter. Accordingly to information collected in the real network, we considered two cases: (i) low loaded networks with an average number of connected stations ranging in the set $\{0,\ldots,4\}$ and (ii) high loaded networks with an average number of connected stations ranging in the set $\{0,\ldots,8\}$. The average number of clusters is 142, with the biggest clusters having four WTPs. The number of interfering APs per cluster ranges from 0 to 2 when the lowest APs density scenarios are considered and from 2 to 24 when we consider the highest APs density scenarios.

Fig. 7 shows the gain, average value and 90% confidence interval, for the low loaded network scenario and for various interfering APs density. All the strategies introduce an interference reduction slightly increasing with the increasing of the density of interfering APs. The gain of heuristic and *iFP* is significantly higher than that of pure LCCS, proving the needing of a centralized coordination of WTPs in the solution of frequency planning. Heuristic configuration is obviously outperformed by *iFP*, but in most cases the additional gain is negligible. The main reason of this low additional gain is in the topology where there is a high number of low populated clusters where iFP and heuristic approaches define similar configurations.

Fig. 8 shows the gain, average value and 90% confidence interval, for the high loaded network scenario and for various interfering APs density. Also in this case the gain introduced by centralized solutions is significantly higher than that obtainable by LCCS. Although the gain increasing of heuristic and *iFP* is lower, the interference absolute value is slightly higher than that measured for similar configurations in low loaded conditions. Indeed, the interference value for random configuration is a linear function of the average number of connected stations.

Eventually, to stress further algorithms performance, we considered a very dense scenario where the interfering APs density is of about 25 APs in each $100 \times 100\,\text{m}^2$. This translates in a number of 78 APs interfering in a single cluster area in the worst case. The average gains with respect to random configurations of LCCS, Heuristic and iFP are about 6.8, 8.22 and 8.25, respectively. The increased gain of LCCS can be explained with the increasing impact that local interferences have when increasing node density.

Simulations analysis proves that the optimal configuration introduces significant reduction in the network interference metric. This reduction, as it is shown in the following section, translates in a significant improvement in the performance that each cluster of the Hot-Spots network can provide to users.

[1] Least Congested Channel Search: in this approach each WTP chooses the local least congested channel without any coordination with neighbor WTPs. For each WPTs' cluster, the WTPs are randomly ordered and sequentially configured based on this approach.

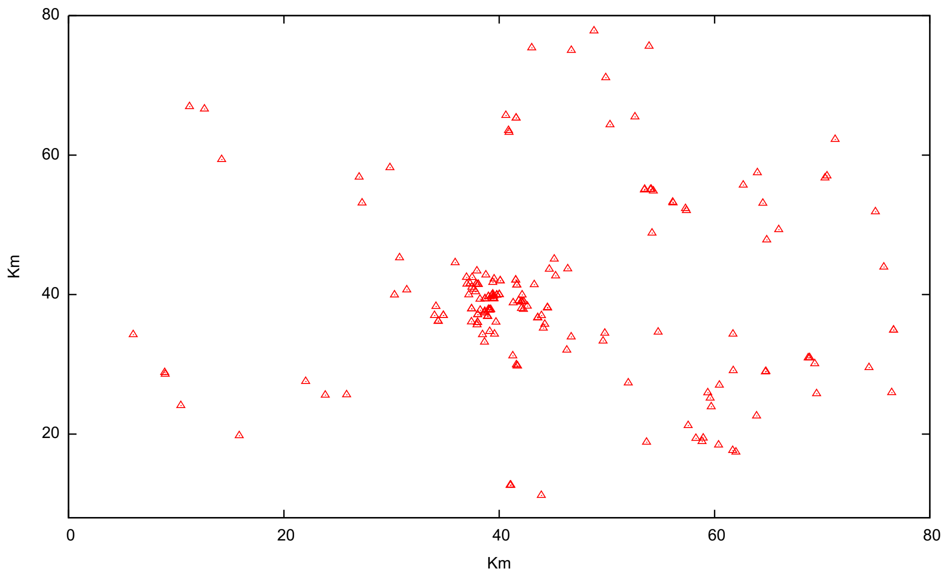

Fig. 6. *Provincia di Roma* WTPs' infrastructure topology.

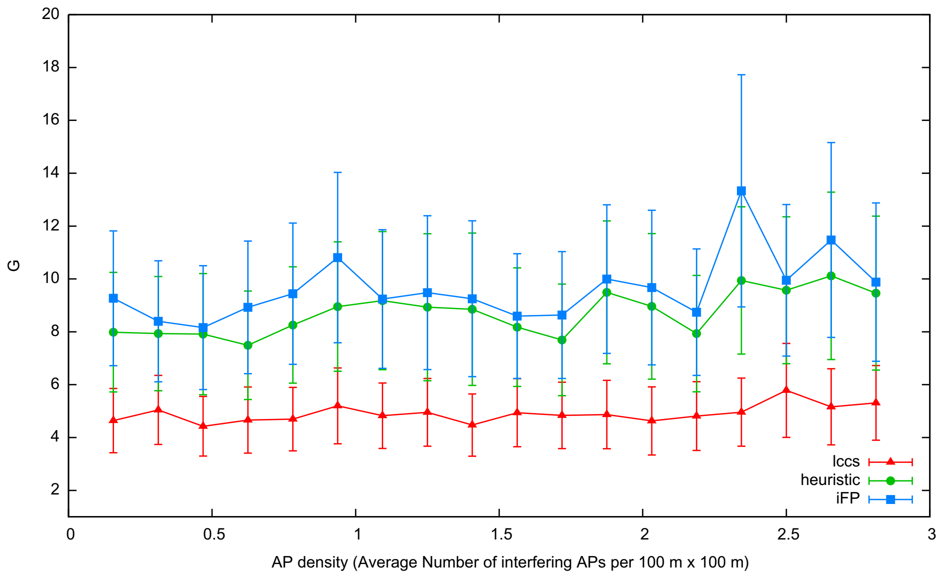

Fig. 7. Average interference gain for low loaded network.

4.5. Experimental results

In this section we report the experimental results obtained by using the frequency planning algorithm described above. The implementation of the algorithm makes use of the CAPWAP protocol for the data exchange between the WTPs and the AC and the commands for frequency assignments are issued by using the mechanism described in Section 2.1.3. Among the many experiments we carried out, we describe here those we consider more significant, one concerning the resulting interference level among WTPs and the other the effect on the throughput of the associated stations.

The first experiments included 5 and 9 WTPs of the *Provincia di Roma WiFi* network. These lightweight devices are based on the *alix31d1* board, with an AMD Geode processor at 433 MHz, 128 MB of RAM memory, and a permanent CompactFlash memory of 1 GB. These devices were equipped with the *openWRT* operating system, the *openCAPWAP* 0.92 client daemon and the frequency planning application (client side). The *Madwifi* driver used on all the WTP's wireless cards of the testbed and the *Provincia di Roma WiFi* network allows for a *bgscan* (background scan) operation as well as for the creation of virtual interfaces. Thus each WTP can discover the presence of non-managed APs in its vicinity by periodically scanning for their presence without service disruption.

In this experiment the managed network included 5 WTPs, 3 of which formed a completely connected interference subgraph, whereas the remaining two were connected by an interference edge with one of the WTPs. In this scenario there were 14 additional APs, not managed through CAPWAP, that used random channels. The plot in the left part of Fig. 9 compares the overall network

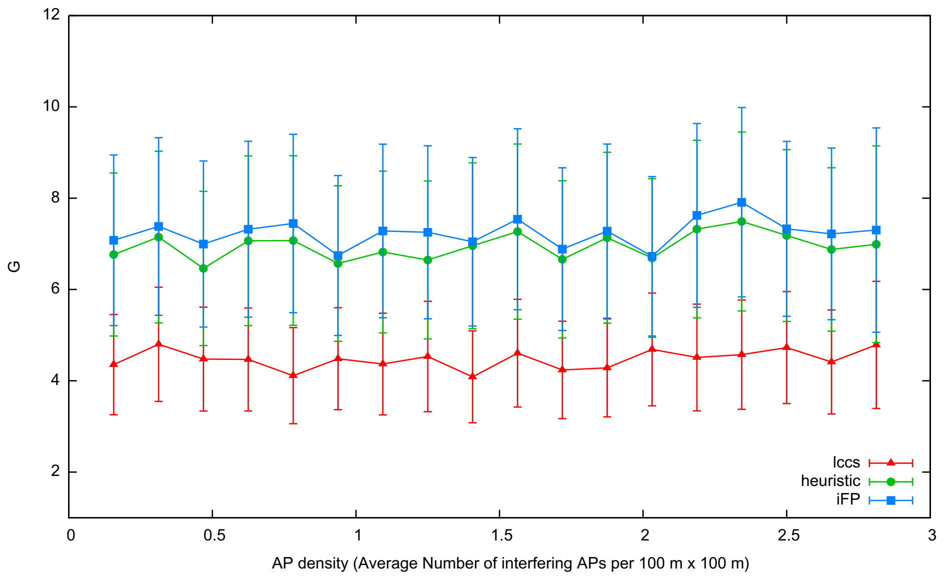

Fig. 8. Average interference gain for high loaded network.

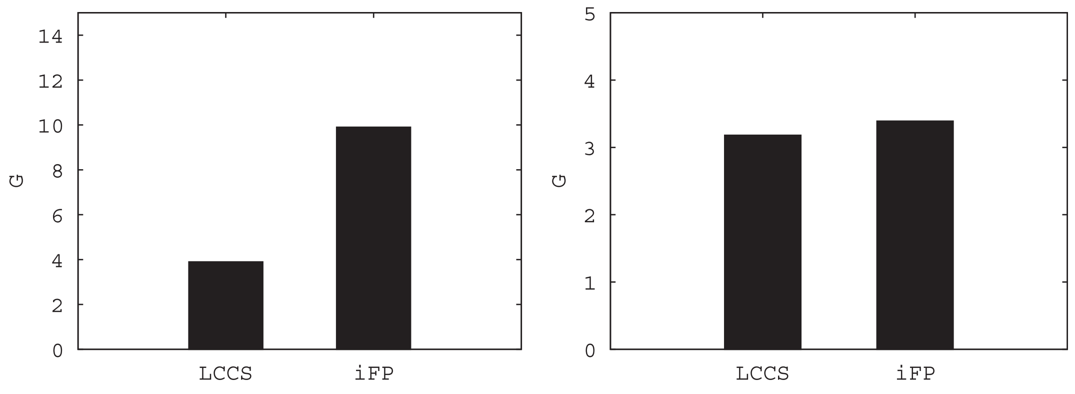

Fig. 9. Average gain value in the first scenario with 5 (left) and 9 (right) WTPs.

interference ratio for the assignment chosen by the LCCS algorithm and the assignment determined by the *iFP* algorithm with respect to an assignment where all the 5 WTPs are on channel 1. In the second scenario, a different topology with 9 WTPs in 2 clusters is used. In this case the number of external interfering WTPs was 30. As the histograms show, in the former case the algorithm improves the interference value, with a remarkable gain with respect to LCCS. In the latter scenario, the reduced gain introduced by iFP with respect to LCCS can be explained by the increased number of interfering APs that constrains the algorithms to behave almost the same.

In the second set of experiments we aimed at measuring the impact of the interference among neighboring WTPs on the data traffic and the behavior of the planning algorithm in presence of station mobility. In these experiments we used 4 WTPs and 6 mobile stations. The devices in both cases were HP tc4200 laptops running Linux Ubuntu OS with kernel 2.6.15-26-386. The wireless NICs used in the test-bed were NETGEAR WG511T PCI cards equipped with an Atheros chipset and MADWiFi 0.9.2.1 open source driver using the IEEE 802.11b PHY protocol.

In these experiments the interference graph was complete. Four WTPs had one station associated to them, whereas the remaining one had 3. Each station and WTP attempted to send and receive an UDP traffic flow of 8 Mbps generated by the *mgen* tool, thus the wireless channels were saturated. In this case there were 14 additional APs not managed through CAPWAP and with a lower signal level, that used random channels.

The experiments underwent two phases, each one lasting 180 s. In the second phase, two mobile stations migrated from one WTP to another. The experiment was repeated for the frequency allocation schemes. In the *fixed* case, all the WTPs used the same channel and this assignment was maintained in the two phases. In the *LCCS* case, the local channel assignment was computed at the beginning of each phase. In the *iFP* case, the algorithm was used to compute dynamically the frequency allocation scheme. Each case was repeated three times, and the average throughput is reported, separately for sent and received traffic, in Fig. 10. The throughput value in the histogram corresponds to an average over 180 s among all the stations. Notice also that LCCS requires to stop the network traffic more often than *iFP*. Indeed, since it does not have a global knowledge of the network, LCCS is more likely to change the frequency of WTPs involved in the communication, while, in general, *iFP* exhibits a more stable behavior. This stability lack of LCCS is particularly exacerbated in big clusters of WTPs where LCCS may generate unstable configurations and poor network performance for connected users.

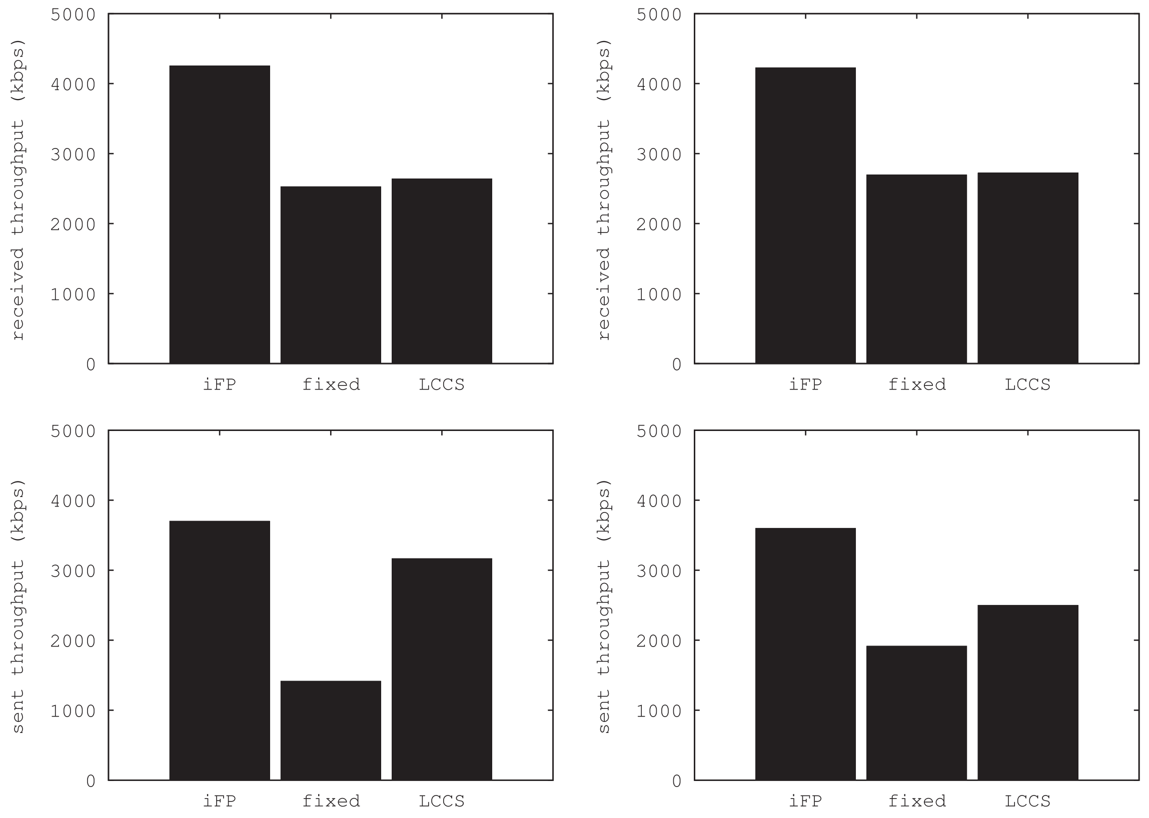

Fig. 10. Throughput comparison for the mobility scenario, average throughput sent and received at the mobile stations, phase 1 (left) and phase 2 (right).

5. Conclusions

We presented a real-world application of *OpenCAPWAP*, an open source implementation of the CAPWAP protocol. The results we obtained confirm that CAPWAP functionalities for network monitoring are a valuable input to algorithms for network management and configuration, while CAPWAP functionalities for network control are fundamental in the enforcement of policies triggered by these algorithms. Our present work aims at using our CAPWAP implementation as a building block of a comprehensive and autonomic architecture for Hot-Spots network management, monitoring and configuration.

Acknowledgment

We thank Andrea Del Moro, Federica Giovannini and Ludovico Rossi for the development of the first prototype implementation of the protocol. We thank Massimo Vellucci, Mauro Bisson and Daniele de Sanctis for their efforts in improving and updating the implementation of the protocols.

References

[1] P. Calhoun, M. Montemurro, D. Stanley, 'CAPWAP Protocol Specification', Internet RFC 5415, IETF, March 2009.
[2] P. Calhoun, M. Montemurro, D. Stanley, 'CAPWAP Protocol Binding for IEEE 802.11', Internet RFC 5416, IETF, March 2009.
[3] M. Bernaschi, F. Cacace, G. Iannello, M. Vellucci, L. Vollero, OpenCAPWAP: an open source CAPWAP implementation for the management and configuration of WiFi Hot-Spots, Computer Networks 53 (2) (2009).
[4] pfSense, <http://www.pfsense.com/>.
[5] KVM, <http://www.linux-kvm.org>.
[6] OpenWRT, <http://openwrt.org>.
[7] T. Dierks, E. Rescorla, 'The Transport Layer Security (TLS) Protocol Version 1.1', IETF, RFC 4346, April 2006.
[8] B. O'Hara, P. Calhoun, J. Kempf, Configuration and Provisioning for Wireless Access Points (CAPWAP) Problem Statement, IETF, RFC 3990 (Informational), February 2005.
[9] OpenCapwap, <http://capwap.sourceforge.net>.
[10] A. Levanti, F. Giordano, I. Tinnirello, A centralized approach for automatic frequency planning in WLAN, in: Proceedings of MediaWiN 2007.
[11] R. Akl, A. Arepally, Dynamic channel assignment in IEEE 802.11 Networks, in: IEEE Portable 2007: International Conference on Portable Informations Devices, May 2007, pp. 1–5.
[12] M. Haidar, R. Ghimire, H. Al-Rizzo, R. Akl, Y. Chan, Channel assignment in an IEEE 802.11 based on signal-to-interference ratio, in: IEEE CCECE: Canadian Conference on Electrical and Computer Engineering: Communications and Networking, May 2008, 2008, pp. 1169–1174.
[13] H. Al-Rizzo, M. Haidar, R. Akl, Y. Chan, Enhanced channel assignment and load distribution in IEEE 802.11 WLANs, in: Proceedings of IEEE International Conference on Signal Processing and Communications, November 2007, pp. 768–771.
[14] Y. Lee, K. Kim, Y. Choi, Optimization of AP placement and channel assignment in wireless LANs, in: Proceeding of the 27th IEEE Annual Conference on Local Computer Networks (LCN), November 2002, pp. 831–836.
[15] A. Mishra, S. Banerjee, W. Arbaugh, Using partially overlapped channels in wireless meshes, in: Proceedings IEEE Wimesh, 2005.
[16] A. Mishra, S. Banerjee, W. Arbaugh, Weighted coloring based channel assignment for WLANs, ACM SIGMOBILE Mobile Computing and Communications Review 9 (3) (2005) 19–31.
[17] A. Mishra, E. Rozner, S. Banerjee, W. Arbaugh, Exploiting partially overlapping channels in wireless networks: turning a peril into an advantage, in: Proceedings of the ACM SIGCOMM/Usenix Internet Measurement Conference ICM '05, 2005, pp. 311–315.
[18] M. Arunesh, V. Shrivastava, S. Banerjee, W. Arbaugh, Partially overlapped channels not considered harmful, in: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, 2006, pp. 63–74.
[19] R. Chandra, P. Bahl, P. Bahl, Multinet: connecting to multiple IEEE 802.11 networks using a single wireless card, in: Proceedings IEEE INFOCOM 2004, 23rd Annual Joint Conference of IEEE Computer and Communications Societies, vol. 2, 2004, pp. 882–893.
[20] H. Liu, H. Yu, X. Liu, C. Chuah, P. Mohapatra, Scheduling multiple partially overlapped channels in wireless mesh networks, in: Proceedings of the IEEE International Conference on Communications, ICC '07, 2007, pp. 3817–3822.
[21] S. Bhowmick, P. Hovland, A backtracking correction heuristic for improving performance of graph coloring algorithms, in: Proceedings of the Second International Workshop on Combinatorial Scientific Computing, 2005.