# WilmaGate: a New Open Access Gateway for Hotspot Management*

Mauro Brunato and Danilo Severina
Department of Computer Science and Telecommunications
University of Trento
Via Sommarive, 14 — I-38050 Trento, Italy
brunato|severina@dit.unitn.it

## ABSTRACT

Wireless access has already become a ubiquitous way to connect to the Internet, but the mushrooming of wireless access infrastructures throughout the world has given rise to a wide range of user authentication, authorization and accounting (AAA) mechanisms, with lots of incompatible "standards", each having its unique features and responding to specific problems.

The WilmaGate system has been developed in order to provide a viable alternative to such a scenario. The assumptions that led to this system are very simple. First, wireless users are often already registered to some traditional access provider, or to an institution: rather than requiring a different subscription to each wireless access system, we just require the subscriber's service provider or institution to collaborate with the access system for user authentication. Second, users should not be forced to install specialized clients into their computers for two reasons: their systems would grow unstable and some types of computers (e.g., PDAs) would not be able to access the network. Third, security for e-mail and passwords is mainly provided by end-to-end protocols such as SSL and TLS, so introducing low-level authentication and encryption can be regarded as unnecessary, and causes often unnecessary overhead.

In this work we oversee the system architecture, which has already been deployed, and describe its main component, the WilmaGate. A short discussion on the current system implementation and its performance is also provided.

## Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*security and protection*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*wireless communication*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*public networks*; D.4.6 [**Operating Systems**]: Security and Protection—*access controls, authentication*

## General Terms

Design, Security

## Keywords

Open Access Networks, Wireless Networks, Access Gateways, Authentication, Authorization

## 1. INTRODUCTION

WilmaGate is a wireless LAN hotspot management system based on the Open Access Network paradigm [3, 6, 4, 7, 5, 8, 1] where a multiplicity of access providers interact with many authentication operators in order to grant access to users registered to different service providers through the widest possible choice of hotspot services. To attain this goal, the WilmaGate system provides functionalities for distributed authentication and authorization; moreover, security, firewalling and other capabilities are provided as standard components of an access gateway system.

The WilmaGate's modular design is intended for easy implementation of novel features such as QoS control, context-dependent services and traffic shaping and modelling.

The general framework of WilmaGate is provided by the WILMA (Wireless Internet and Location Management Architecture) Project [2, 12], whose goal is, among others, to build an open wireless testbed where innovative algorithms and procedures can be implemented and tested safely within an open source architecture.

Within the WILMA project, the WilmaGate is serving as a common framework for the management of different WLANs deployed at the University and in the city and province of Trento. Because of the WilmaGate, these wireless networks form an integrated testbed where university students and employees, customers of a local telecommunications provider and members of other local institutions are free to login and navigate.

The rest of this paper is organized as follows. Section 2 describes previous work in the field and motivates our choice of writing a new access platform. Section 3 presents an overall description of the system, with details about its main components. Section 4 describes the "captive portal" approach

*This work was supported by the WILMA Project [12] sponsored by the Autonomous Province of Trento (PAT), and by the TWELVE PRIN Project [11], sponsored by the Italian Ministry of Education and Research (MIUR).


*WMASH'05,* September 2, 2005, Cologne, Germany.

that has been implemented in the system (although other approaches are possible) as an authentication mechanism. Section 5 shows some examples on how the WilmaGate system can be deployed in different types of network. In Section 6 we finally describe the WILMA testbed where our gateway is already deployed.

## 2. MOTIVATION AND RELATED WORK

Open Access Networks (OANs) [3, 6, 7, 5] are a possible remedy to the current vertical model of wireless access provisioning. In the traditional model, many hotspots, managed by different service providers, coexist in the same geographical area, leading to a waste of resources, mainly frequency channels, suboptimal communication capabilities because of interference, and possibly EM exposure beyond the legal limits. Overcrowded hotspots are a necessary consequence of the lack of coordination among telecommunications operators, who only grant access to their customers and seldom have roaming agreements.

On the other hand, this model prevents small operators and institutions from installing publicly accessible hotspots. In fact, people would need to subscribe to a lot of operators in order to get reasonably wide-area access grants.

A possible solution is decoupling the access and the authentication layer, having different providers for them. In particular, authentication and authorization could be managed by the traditional operators, who already have a large customer database and are interested in providing the widest possible access to them, and also by small institutions who have employee or student databases. Access operators can manage any size of hotspot, from a small one-access-point cafe to a large shopping mall to a city infrastructure. The underlying business model envisions some form of commercial agreement between each access operator and each authentication server; of course, for this model to be scalable it is necessary to have some intermediate brokers and a common agreement form. Authentication servers would manage user billing, while access operators would more naturally receive a share which is proportional to their supported traffic.

Some systems already support some versions of this model. A largely cited example is SwedenOpen.net, pioneered by the StockholmOpen Project [10, 6], a city-wide OAN infrastructure in Stockholm. The StockholmOpen model, however, requires every access provider to host a server from every authentication provider in order to support its policies.

A rather successful system is the NoCat authentication system [9]. While not really an OAN, its open-source nature and its authentication facilities are notable, and many small systems have been set up with its open-source code.

The WilmaGate system shares aspects with both aforementioned projects. In particular, it tries to maintain the Open Access nature of StockholmOpen while allowing less stringent policy coordination requirements; moreover, the initial authentication system, based on the "captive portal" approach, mimics and extends the NoCat architecture.

Development of our system has always followed some basic assumptions.

- The OAN approach: wireless users are often already registered to some traditional access provider, or to an institution: rather than requiring a new registration to each wireless access system, we just require the service provider or institution to collaborate with the access system for user authentication.
- Users should not be forced to install specialized clients into their computers for two reasons: their systems would grow unstable and some types of computers (e.g., PDAs) would not be able to access the network.
- Security for e-mail and passwords is already provided by end-to-end protocols such as SSL and TLS, so introducing low-level authentication and encryption can be regarded as unnecessary, and causes often unnecessary overhead.
- Different users may request different forms of authentication. The WilmaGate system should be built in a modular fashion, in order to install new authentication procedures as plugin modules without disrupting previous functionalities.

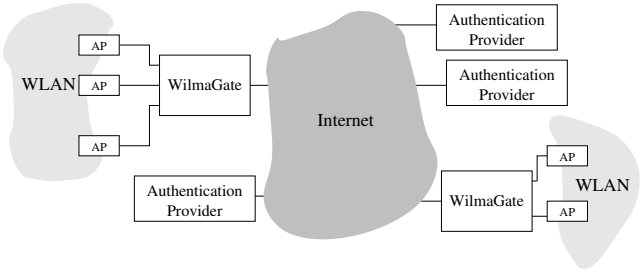


**Figure 1: An overall view of the WilmaGate system.**

## 3. SYSTEM OVERVIEW

The WilmaGate system is fully implemented in C++ as a collection of user-space applications. Linux was chosen as operating system, but portability to other operating systems, in particular Windows, is one of the main criteria in development, and the code can be executed on a Win32/ Cygwin machine. On the other hand, Linux-specific optimizations, such as the implementation as kernel-space modules, are being considered for high performance systems.

The goal of the WilmaGate is the feasibility demonstration of flexible and secure management within WLANs and public hotspots, without the need to resort to proprietary implementation, exceedingly complex procedures or mandatory installation on mobile clients of ad-hoc software.

The WilmaGate system implements a wireless open access network gateway. Its main features are:

- Support for multiple external authentication providers;
- Extensible support for several authentication and authorization techniques;
- Firewalling;
- Accounting.

Figure 1 shows an overall view of the proposed network components: WilmaGate operates as interface between wireless hotspots (light grey areas with access points in them) and the Internet or a wired LAN. A number of authentication providers is available in the Internet, and can be contacted by wireless clients to get full access to the Internet.

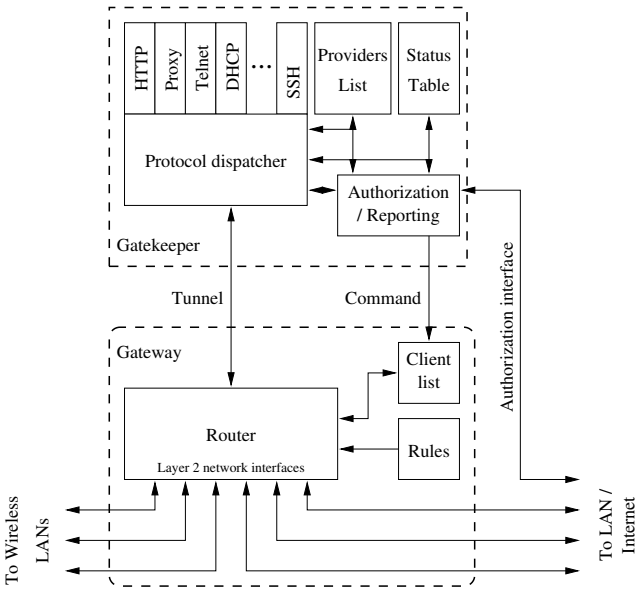


**Figure 2: Block diagram of the WilmaGate system**

The internal structure of a WilmaGate node is shown in Figure 2. The application is divided into two components in order to separate functionalities. Performance considerations can help deciding whether the two components must be embedded in the same machine or must be executed to two different computers.

The *Gateway* component is responsible for all network-intensive functionalities: it is basically a layer-3 switch with some additional ad-hoc functionalities (however, a configurable router or a firewall with enough flexibility could be applicable): it checks packets that are directed from and to authorized clients and puts them into the right interface. The Gateway component bridges the wireless LAN and the other side, which may be a corporate LAN, a WAN, a point-to-point line with several network components in it. Non-interference with other network components is an important requirement.

The *Gatekeeper* component is responsible of all CPU-intensive functionalities (table look-ups, authentication protocols etc.). It receives unauthorized packets from the Gateway: packets are subjected to processing and are used to trigger events such as an authentication procedure. The authentication procedure will involve the client, the Gateway (for packet forwarding), the Gatekeeper and an authentication provider, possibly chosen among many. The role of every component and the type of interaction depend on the authentication mechanism, and shall be discussed later.

## 3.1 The Gateway component

The Gateway machine contains various network interfaces (at least two), some of which are directed to the wireless LAN, and some to the wired LAN. Its main purpose is to serve as a configurable Layer 3 switch.

As shown in Figure 2, the *router* module (a C++ object) manages interchange of Ethernet frames among network interfaces and the Gatekeeper. The router module identifies packet sources and destinations according to their Ethernet MAC address and their IP address. Correspondence between MAC and IP addresses for authorized clients is stored in the client list, accessed by the router via a read-only interface.

| Variable | Meaning |
|---|---|
| *authWLAN* | List if authorized (MAC,IP) pairs |
| *srvLAN* | List of freely accessible IPs |
| *e* | Ethernet frame |
| *c* | Authorization command |
| *sense* | Direction of an ethernet frame |

```
1.  function initialize
2.      authWLAN ← ∅
3.      srvLAN ← predefined list of IP addresses
4.  upon receipt of e along WLAN
5.      if (e.senderMAC,e.senderIP) ∈ authWLAN
6.              or e.receiverIP ∈ srvLAN
7.          send e along LAN
8.      else switch ( action (e.protocol) )
9.          case forward:
10.             send e along LAN
11.         case submit:
12.             send e to Gatekeeper
13.         case drop:
14.             nop
15. upon receipt of e along LAN
16.     send e along WLAN
17. upon receipt of (e, sense) from Gatekeeper
18.     send e along sense
19. upon receipt of c along Command
20.     if c.command=authorize
21.         authWLAN ← authWLAN ∪ (c.MAC, c.IP)
22.     else
23.         authWLAN ← authWLAN \ (*, c.IP)
```

**Figure 3: The Gateway algorithm**

The router takes decisions on packets based on the client authorization status, stored in the *Client list*, and on the packet protocol at network and transport level; rules governing the router's behavior are stored in the *Rule table*, created on startup and accessed by the Router via a read-only interface.

In principle, all packets coming from wireless clients whose status is unauthorized are sent to the Gatekeeper for further processing, while all packets received from the Gatekeeper are forwarded to the appropriate interface, with no interference from the Gateway rules.

Rules and authorization tables are kept as simple as possible in order to avoid any bottleneck and to simplify implementation on diskless dedicated hardware with a limited amount of computational power, if required. The client list is updated via a *Command* interface, which receives simple text commands from the Gatekeeper.

## 3.2 The Gatekeeper component

The Gatekeeper component performs all tasks that require more processing than just looking at frame and packet headers. It receives all unauthorized packets from the Gateway through the "tunnel" shown in Figure 2. This tunnel is im-

plemented as a pair of UDP sockets. Packets are processed with the goal of authenticating the user, and eventually sent back to the Gateway to be forwarded.

The main functions of the Gatekeeper are:

- Client status maintenance; the Gatekeeper's *status table* is essentially a superset of the Gateway's client list;
- DHCP management;
- Client authentication and authorization.

The Gatekeeper acts according to the packets received from the Gateway component, updates its status table and issues commands back to the Gateway component. Commands are of the following form:

- Send a packet through a specified interface;
- Authorize client identified by MAC/IP pair;
- Revoke authorization to a given client.

In addition to the Gateway tunnel, Gatekeeper interacts with remote entities via two TCP sockets:

- the **Authorization** interface is used to receive information by a trusted authentication server about authorization of users;
- the **Report** interface is used to provide data about the current status of the clients and is used to inform the Gateway about the changes of the status of users.

### 3.2.1 The Status Table

The Status table represents the current status of the network and it is the core of the Gatekeeper system. Its structure is a table, where every table row is associated to one IP address of the wireless subnet, and contains status information such as the Ethernet MAC address of the currently associated wireless client, traffic information, if required also the identity of the person who got the authorization.

Moreover, each entry contains a status variable representing the DHCP association status and the authorization status of the corresponding IP address, which can assume the values:

`FORBIDDEN` The IP address is reserved for system use;

`FREE` The IP address is free and can be associated to a new client upon a DHCP request;

`FIRST_SEEN` The IP address has been offered to a new client during the discovery process;

`TEMPORARY_LEASE` The IP address is leased to the client with a short expiration time (non-authenticated client);

`AUTHORIZED` The client has been identified and gateway services are granted. Subsequent DHCP leases will confirm the current IP address with a longer expiration time.

### 3.2.2 The Providers List

The Providers List contains a list of trusted authentication providers. Each provider is associated to a name, an IP address, and some data about the identification process. This list is used to recognize the IP address that can be reached by an unauthorized user, who must be allowed to contact the authentication provider to initiate an authentication procedure.

| Variable | Meaning |
|---|---|
| *plugins* | List of packet handlers |
| *e* | Ethernet frame |
| *sense* | Direction of an ethernet frame |

```
1. upon receipt of e from Gateway
2.     for each p ∈ plugins
3.         l ← p.manage (e)
4.         if l ≠ dontcare
5.             for each (e', sense) ∈ l
6.                 send (e', sense) to Gateway
7.             exit
```

**Figure 4: The Dispatcher algorithm**

### 3.2.3 The Dispatcher

The main component in the Gatekeeper architecture is the Dispatcher. It represents the Gatekeeper's tunnel endpoint. As shown in Figure 4, when a packet is received through the tunnel (meaning that the Gateway requires some processing), the Dispatcher polls protocol-specialized modules, called "plugins" (see Section 3.2.4), until it finds one that can handle the packet. Every plugin $p$ is a C++ object which exposes to the Dispatcher the method $p$.manage $(e)$ which returns either a list of forged packets that must be sent back to the Gateway for routing, together with direction information to express the interface the packet should be sent through, or a special `dontcare` value, meaning that the next plugin object in the Dispatcher's list must be queried.

### 3.2.4 The Plugins

The plugin modules are used by the Dispatcher module in order to take decisions about specific protocols. As said in Sec. 3.2.3, all plugins expose a public method which is invoked by the Dispatcher in order to submit a packet; this function returns a list of response packets, if needed, together with some indication about the subsequent action (e.g., send back the original packet to the Gateway for forwarding, send a new packet). If necessary, the plugin modules can access and modify the status table.

**The DHCP plugin** — The DHCP plugin module manages the DHCP service in the wireless LAN. When the Dispatcher receives a DHCP packet, it submits it to the DHCP module. To decide the appropriate action, the DHCP module can access the authorization list, where the status of every IP address is stored.

**The HTTP and Proxy plugins** — The HTTP and Proxy modules handle direct and proxy-based HTTP connection requests. To implement the "captive portal" mechanism (see Section 4.2), the HTTP module forges an entire connection session (setup handshake, HTTP GET/response and shutdown handshake) by spoofing the requested web server. Its fixed response to any client request is a redirection toward a predetermined page, which can either contain instructions to correctly set the proxy server in the client system, or a login form. On the other hand, the proxy module does not substitute itself to the actual proxy server, but reads all packets and parses HTTP requests. Only when it intercepts a request that is not directed to an authentication server, it generates an HTTP 302 response that redirects the client to an authentication page. Moreover, the

| Variable | Meaning |
|---|---|
| *authTable* | Associative array $IP \Rightarrow$status |
| $m$ | Message from authentication server |
| $c$ | Command to Gateway |

```
1.  function initialize
2.      for each IP ∈ WLAN subnet
3.          authTable[IP].status ← FREE
4.  upon receipt of m from authenticator
5.      if m.command = authenticate
6.          authTable[m.IP].status ← AUTHORIZED
7.          authTable[m.IP].who ← m.ID
8.          c.command ← authorize
9.          c.IP ← m.IP
10.         c.MAC ← authTable[m.IP].MAC
11.     else if m.command = revoke
12.         authTable[m.IP].status ← FIRST_SEEN
13.         c.command ← unauthorize
14.         c.IP ← m.IP
15.     send c to Gateway's Command interface
```

**Figure 5: The authorization module algorithm**

necessary FIN packets are forged in order to shut down the proxy communication and force the client to set it up again (otherwise sequence numbers would not match anymore).

#### 3.2.5 The Authorization module

The authorization module is built upon the regular TCP/IP stack. It listens to a TCP port (default is 54273) on the wired interface. It can be contacted by a trusted authentication server, and accepts commands for user authorization. Figure 5 shows how a command issued by a remote authentication server causes the Authorization module to modify the corresponding entry of the status table and to issue the necessary commands to the Gateway in order to let the client's packets pass through without further processing.

The module has the ultimate responsibility of moving a client to the `AUTHORIZED` status, granting to it full access to the network, and to revoke it upon explicit logout or when the authorization period expires with no renewal.

It obeys two textual commands:
`authorize` *IP name email token*
`revoke` *IP*

The first command is issued by an authentication provider willing to authorize a user, the second to revoke such authorization. Note that the authorization command provides additional information for logging, i.e., the name and email of the authorized user and a token that has been issued during the DHCP process and sent to the client in the early authentication phase.

#### 3.2.6 The Reporting module

It sits on a TCP port on the wired interface (default is 54272) and is used to provide data about the current status (connected clients, trusted providers).

The same module is also invoked by the status table when a client's status changes. In this case, if needed, the Report module is responsible of sending the appropriate client authorization or revocation command to the Gateway.

# 4. CLIENT AUTHENTICATION

In a general architecture with a WilmaGate system, a client must have a valid IP address and an authorization to use Internet services. To obtain the authorization, the client must contact his own authentication server and send the requested information about his own identity. If the authentication server recognizes the user's rights, it contacts the WilmaGate to inform it that the client can access the Internet services.

When the client sends information to the authentication server it is not yet authorized for general web browsing by the WilmaGate, so the WilmaGate needs proper policy to allow the forwarding of authentication-related packets.

Figure 6 describes a slightly simplified timing diagram for the "captive portal" authentication procedure, described in the following sections.

## 4.1 DHCP discovery and request

When the user turns on the networking interface, the first packets that the client sends are related to dynamic configuration (DHCP) to request an IP address.

All DHCP-related packets are tunneled to the Gatekeeper, even if the requesting client is already authorized (e.g., an IP renewal is taking place). The Gatekeeper, operating as a DHCP server, decides the IP address to assign, whether to renew it or not, the duration of the IP lease.

To increase the system flexibility, a new DHCP relay plugin module is under test. The advantages of referring to an external DHCP server are manifold: a unified policy with other network segments, better usage of option fields, more detailed status maintenance. However, some problems had to be solved. In particular, the WilmaGate system uses different DHCP lease time for unauthorized and authorized clients, while the IP address pool is not necessarily different. Common DHCP servers do not offer such functionality, which has to be simulated by the DHCP plugin module while keeping the IP assignment tables synchronized.

## 4.2 The "captive portal" capture method

When the client obtains an IP address, it still cannot browse the web. When the browser is pointed to some URL, it will first perform a DNS query, to which the Gateway is transparent (second shadowed strip of Figure 6). The DNS query will concern either the domain name of the requested website or, if the Proxy server has been correctly set, the proxy IP.

After getting the correct DNS response, the browser opens a connection to the proxy server; after the connection is open (not all the handshake is drawn in the diagram), an HTTP GET request is issued towards the proxy server (time label A). The packet containing the GET request is intercepted by the Gatekeeper at time label B, which forges a response packet by pretending to be the proxy server. The response contains an HTTP 302 code which redirects the client to an authentication page. Moreover, the packet has the FIN flag set, so that the client browser closes its connection to the proxy server. A FIN packet is also forged and is sent to the proxy server on behalf of the client. As a result, at time label C the connection is closed and the client is redirected to the authentication provider's page. Closing the connection is necessary, because after forging the redirect response sequence numbers at the client and at the proxy endpoints would not correspond.

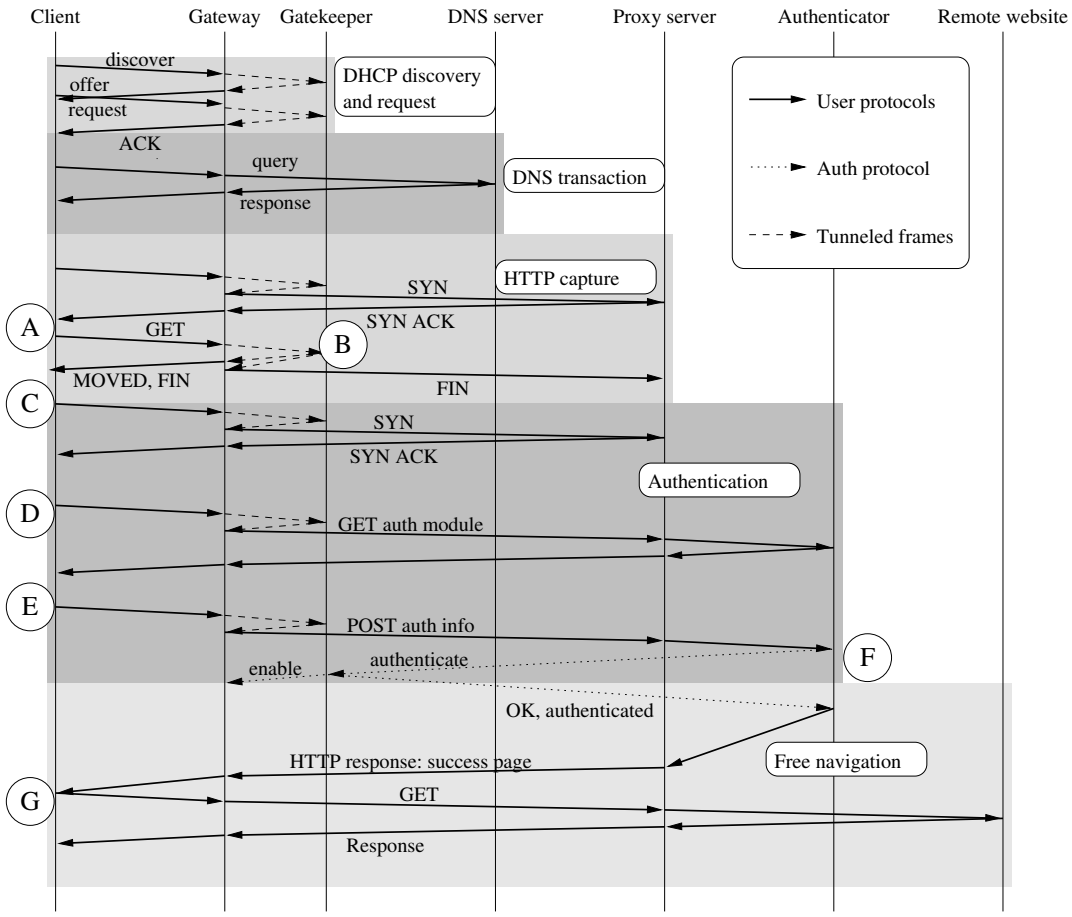


**Figure 6: Simplified timing diagram from client appearance to full authorization**

Thus, the authentication process starts with the client issuing a new web proxy connection. The next GET request (actually a CONNECT request to a secure server, time label D) is directed to the authentication server's login page, so it is admitted by the Gatekeeper. After getting the page, the user fills it in and (time label E) posts the login data to a PHP authentication script in the authentication server, which (at time label F) checks the user's login credentials (password or certificate) and, upon approval, contacts the Gatekeeper's authorization module and sends an authorization command. The Gatekeeper updates its own status table, sends an authorization command to the Gateway, unblocking the user's packets, and a response to the authentication server. The authentication server, in turn, sends a success page to the client, and the client is free to navigate the web (its connection to the net is no longer tunneled to the Gatekeeper).

It must be noted that all sensitive information are directly tunneled between the user and the authentication provider via HTTPS, so the authentication procedure cannot actually be decrypted by the access operator, who will never be able to steal passwords by eavesdropping. Only the final result of the authentication procedure, the authorization command, is sent by the authentication provider to the WilmaGate; however, this command does not contain user sensitive data. The Gatekeeper only needs to know that a client has been positively authenticated by a trusted provider.

In order to uniquely identify a client during the authentication process, a 16-byte random word called *token* is generated by the Gatekeeper during the redirection process (time label B of Figure 6) and is inserted in all subsequent authentication web pages. The same token shall be reported in the authentication message (time label F).

## 4.3 Authorization refresh

The client's authorization must be periodically renewed in order to avoid IP/MAC spoofing by unauthorized users. If the authorization is not refreshed within a given period of time, the client shall be considered gone, and authorization shall be revoked.

The authentication server's "success" page contains Javascript commands that open a pop-up browser window displaying a secure page that is periodically reloaded, each time passing a new shared random alphanumeric string, called "token".

The first token is created by the Gatekeeper upon DHCP request, and passed to the client when redirected to the authentication server. When the authentication server confirms the user's authorization, it sends in the user's token, which is checked against the stored value. A new value is generated and sent to the client's pop-up page, and will be requested at the first renewal. A new token is generated at every renewal.

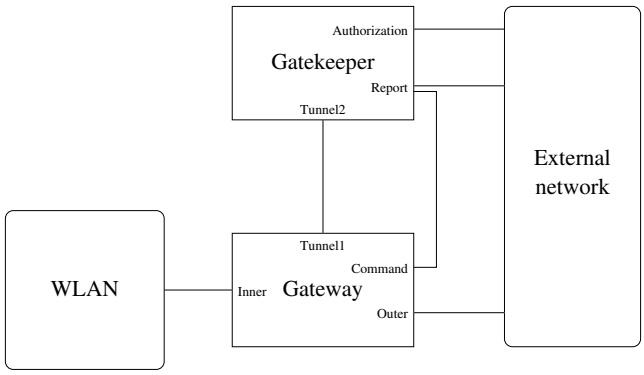


**Figure 7: Reference for configuration**

## 4.4 Issues with small devices

Small clients, such as PDAs, are not fully compatible with the authentication system described above; in particular, they may lack the ability of opening pop-up windows. Most operating systems for PDAs only support one window per program, so no automated renewal is possible and the client is accepted into the system for few minutes.

The problem can be solved in different ways:

- bypass the authentication procedure by statically inserting MAC/IP address pair in the WilmaGate configuration parameters;
- detect the client's operating system and in case allow a longer authorization time;
- create an ad-hoc authentication procedure based on some resident software that takes care of renewals.

The first solution clearly reduces the system's security, because it is open to a simple MAC spoofing attack. The two latter alternatives are being tested in order to identify the simplest and less invasive solution: while the use of a longer authorization time for PDAs can affect security, installing a client program in order to keep the authorization alive is contrary to the project philosophy as stated in Section 2, so pros and cons must be carefully weighed.

# 5. CONFIGURATION

System parameters are written in two configuration files, one for the Gateway, the other for the Gatekeeper component.

Figure 7 contains reference names of all interfaces that must be defined and configured in order to have the system up and running.

The configuration file for the Gateway component contains information about:

- the description of physical interface and its IP address toward the the network that is managed by the WilmaGate (**Inner** interface): this interface has its own physical address and a "fake" address that emulates the gateway because it must answer to ARP request from managed wireless network;
- the description of physical interfaces and their IP addresses toward the the external network (**Outer** interface): this interface has its own physical address and a "fake" address because it must respond to ARP queries on behalf of wireless clients in order to work as gateway;
- the description of the external network gateway;
- the description of physical interface and its IP address and port toward the Gatekeeper (**Tunnel1** interface);
- the description of Gatekeeper network interface parameters (**Tunnel2** interface, endpoint of **Tunnel1**);
- the external network parameters: for instance, IP addresses of proxy, DNS servers and HTTP servers for authentication purposes;
- the runtime mode: the system can run as daemon and you can specify the name and position of log file;

The configuration file for the **Gatekeeper** component contains information about:

- the description of physical interfaces and their IP addresses toward external network (**Authorization** and **Report**) and Gateway (**Tunnel2**);
- the description of Gateway network interface parameters (**Tunnel1** interface, endpoint of **Tunnel2**);
- the DHCP server: the pool of IP addresses you want to use and the lease times (short lease time for unauthorized user and long lease time for authorized users);
- the DNS server of the network;
- the captive portal method: redirection URL, redirection URL when no proxy is set, proxy configuration file URL, and authorization renewal time;
- the URL of authentication providers;
- the runtime mode: the system can run as daemon and you can specify the name and position of log file;
- static IP assignments: static IP addresses can be set for some devices (recognized by their MAC addresses).

## 5.1 Some configuration examples

The WilmaGate system can be configured in different topologies, according to pre-existing LAN topologies and desired security.

The easiest way to explain possible configuration is to refer to the following figures.

Figure 8 refers to a **All in one** configuration: both of the software (Gateway and Gatekeeper) are in the same machine. It can be used in small hotspots, SOHO or smart APs.

Figure 9 refers to a **Tandem** configuration: the Gateway and Gatekeeper softwares are in different machines and both of them have access to external network. It can be used when the target are large traffic volume, more CPU power or higher throughput.

Figure 10 refers to a **Front-end** configuration: the Gateway and Gatekeeper are in different machines but only the server with Gatekeeper software has access to external network. It can be used when existing LANs has only one available public IP address.

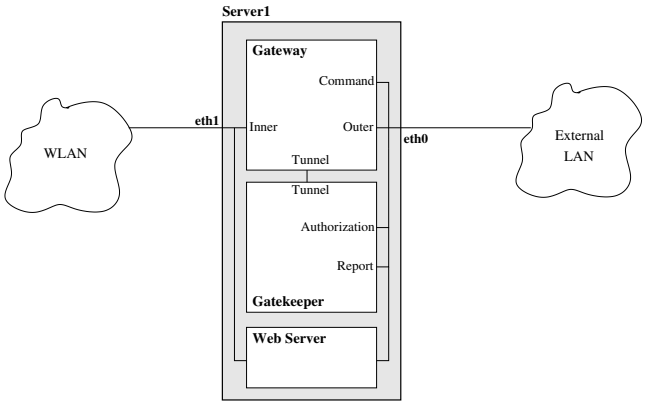


**Figure 8: Reference for "All in one" configuration**

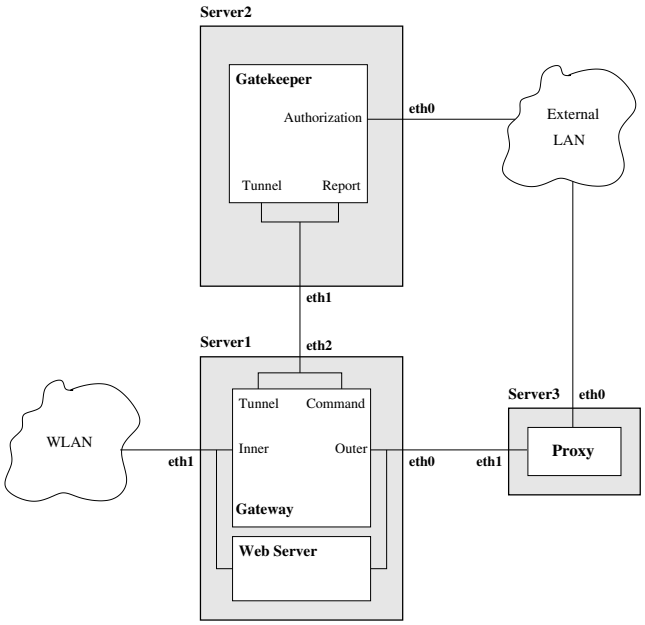


**Figure 9: Reference for "Tandem" configuration**

## 6. IMPLEMENTATION

As part of the WILMA Project [12], a number of hotspots have been deployed in the town of Trento by project participants. In particular, the University of Trento has fully covered its Science Faculty building with 20 access points, while Alpikom (a regional telecommunications operator) has connected an equal number of access points to a network of streetlight poles in one of the most populated areas of the town. While the two hotspots are managed by two different entities, both are managed by a WilmaGate. The University WilmaGate is configured in the "All in one" fashion, while the Alpikom gateway is in the "Front-end" configuration.

Access to both hotspots is granted to all university employees and students by means of a simple PHP website using the University's LDAP database for user authentication, complemented by a local MySQL database for visiting guests. Alpikom subscribers are authenticated via a similar PHP website acting as a front end to the company's RADIUS authentication server.

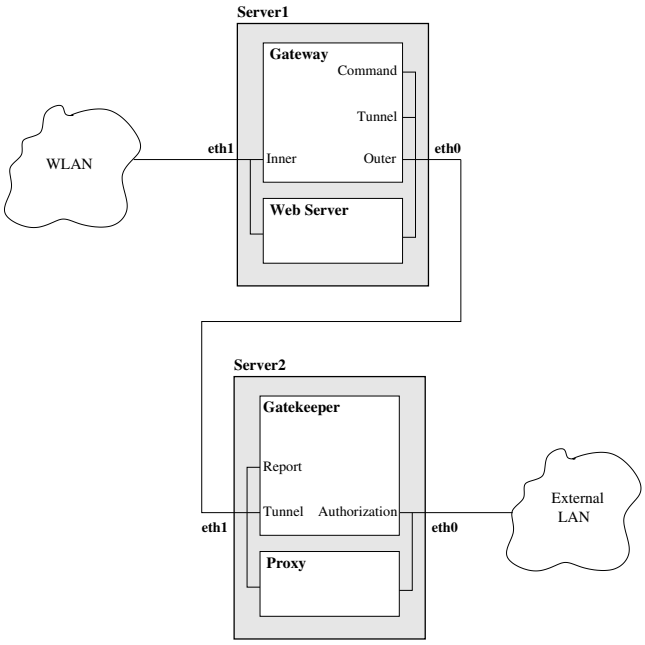


**Figure 10: Reference for "Front-end" configuration**

The university system is managing 60 to 100 different users per day with more than 300 logins per day (see Figure 11), with traffic peaks above 800 kilobytes per second.

Currently, the system has not been tested in stress conditions. However, considering that all packet routing is done by packet capture (via the libpcap mechanism) in the user space, we believe that once the current system reaches its limit there can be a great deal of optimization, such as kernel-level packet management and/or netfilter integration.

## 7. CONCLUSIONS

In this paper we have presented a gateway for a wireless open access system where every access operator can take advantage of multiple authentication servers to grant access to its users.

The system is motivated by the growing complexity and

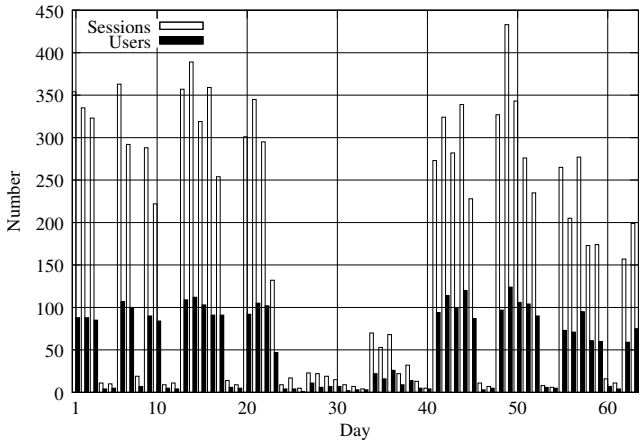


**Figure 11: Daily usage**

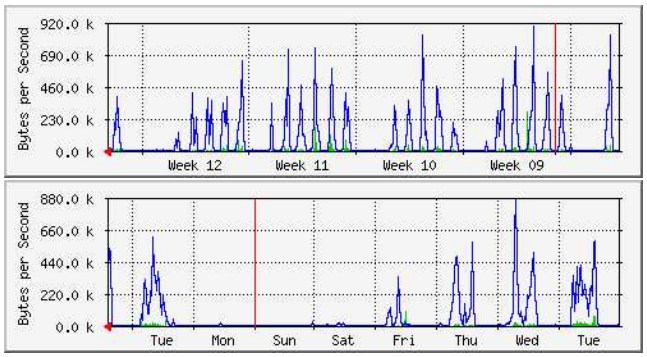


**Figure 12: Monthly (top) and weekly (bottom) throughput**

differentiation of standards concerning user authentication, and aims at making access as straightforward as possible, while maintaining the necessary reliability for network operators. A description of the main building blocks of the system has been presented, together with an analysis of an authentication mechanism.

The actual deployment of the system on a city-wide testbed in the city of Trento enables us to check the system's functionalities against real-world usage, to fine-tune its performances and to improve its functionalities guided by continuous user feedback.

## Acknowledgments

WilmaGate has been developed at the Computer Science and Telecommunications Department of the University of Trento by the Computer Networks and Mobility Research Group.

We thank all people who work in the NetMob group; in particular prof. Roberto Battiti and prof. Renato Lo Cigno for their continuing advice and support, and Alessandro Villani for putting his expertise at our disposal.

The WilmaGate system is available for research and other non-profit purposes [13].

## 8. REFERENCES

[1] M. Altmann, H. Daanen, H. Oliver, and A.-B. Suarez. How to market-manage a QoS network. *Proceedings of Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 1, 2002.

[2] R. Battiti, M. Brunato, R. Lo Cigno, A. Villani, G. Lazzari, and R. Flor. WILMA: an open lab for 802.11 hotspots. In M. Conti, S. Giordano, E. Gregori, and S. Olariu, editors, *Proceedings of PWC2003*, 2003.

[3] R. Battiti, R. Lo Cigno, M. Sabel, F. Orava, and B. Pehrson. Wireless LANs: from warchalking to open access networks. *Mobile Networks and Applications*, Vol. 10:275–287, 2005.

[4] A. Escudero, B. Pehrson, E. Pelletta, J. O. Vatn, and P. Wiatr. Wireless access in the flyinglinux.NET infrastructure: Mobile IPv4 integration in a IEEE 802.11b. *11-th IEEE Workshop on Local and Metropolitan Area Networks*, 2001.

[5] M. Hedenfalk. Access control in an operator neutral public access network. *MSc thesis, KTH/IMIT*, 2002.

[6] B. Pehrson, K. Lundgren, and L. Ramfelt. Open.Net - open operator neutral access network. In *12-th IEEE workshop on Local and Metropolitan Area Networks*, Stockholm, SE, 2002.

[7] E. Pelletta, F. Lilieblad, M. Hedenfalk, and B. Pehrson. The design and implementation of an operator neutral open wireless access network at the kista it-university. *12-th IEEE Workshop on Local and Metropolitan Area Networks*, 2002.

[8] S. Vaughan-Nichols. The challenge of Wi-Fi roaming. *ACM Computer*, 2003.

[9] NoCat Network. `http://nocat.net/`

[10] StockholmOpen Project. `http://www.stockholmopen.net/`

[11] TWELVE Project. `http://twelve.unitn.it/`

[12] WILMA Project. `http://www.wilmaproject.org/`

[13] WilmaGate download page. `http://netmob.unitn.it/wilmagate.html`