

UNIVERSITÀ DEGLI STUDI DI TRENTO

DIPARTIMENTO DI INFORMATICA E TELECOMUNICAZIONI



THE WILMA PROJECT<sup>1</sup>

## WilmaGate: An Overview

Roberto Battiti, Mauro Brunato  
Renato Lo Cigno, Danilo Severina  
Alessandro Villani

Version 1.0

June 2005

---

<sup>1</sup>The Wilma Project is funded by the Autonomous Province of Trento

## **Summary**

This document is a technical overview of WilmaGate, the captive-portal based WLAN management system developed under the Wilma project.

WilmaGate is an open framework for the management of WLAN and Hot-Spots, also in public environments. It is inspired to the architecture of Open Access Networks, where the access infrastructure is neatly separated from the service provider infrastructure, and users are free to connect to their preferred service provider through the access WLAN.

The main technical features are discussed and introduced, referring the reader to the more detailed technical documentation for all details.



# Contents

|   |           |
|---|-----------|
| <b>1 WilmaGate at a Glance</b>                      | <b>1</b>  |
| 1.1 Project Objectives . . . . .                    | 1         |
| 1.2 Use and Misuse . . . . .                        | 1         |
| 1.3 Companion Documents . . . . .                   | 2         |
| <b>2 Features and Characteristics</b>               | <b>3</b>  |
| 2.1 Network Configuration under WilmaGate . . . . . | 3         |
| 2.2 Functional Architecture . . . . .               | 4         |
| 2.3 The Providers List . . . . .                    | 5         |
| 2.4 Plug-ins . . . . .                              | 6         |
| 2.5 The “captive portal” technique . . . . .        | 6         |
| 2.6 Security and Privacy . . . . .                  | 7         |
| 2.7 Authentication . . . . .                        | 7         |
| <b>3 Joining the WilmaGate Test-Bed</b>             | <b>9</b>  |
| 3.1 Getting WilmaGate . . . . .                     | 9         |
| 3.2 Download and Installation . . . . .             | 9         |
| <b>4 Compiling the system</b>                       | <b>10</b> |
| <b>5 Configuration</b>                              | <b>11</b> |
| 5.1 Gateway configuration . . . . .                 | 11        |
| 5.2 Gatekeeper configuration . . . . .              | 12        |
| 5.3 Web server configuration . . . . .              | 12        |
| <b>6 Running the programs</b>                       | <b>13</b> |
| <b>7 WilmaGate Evolution</b>                        | <b>13</b> |



# 1 WilmaGate at a Glance

WilmaGate is developed at the Department of Computer Science and Telecommunications

(<http://dit.unitn.it>)

of the University of Trento

by the Computer Networks and Mobility Research Project

(<http://netmob.unitn.it>)

within the Wilma Project

(<http://www.wilmaproject.org>)

framework (funded by the Autonomous Province of Trento).

WilmaGate is a Wireless LAN and HotSpot management system based on the general idea of captive portal, with distributed authentication, security, firewalling, and similar capabilities. It smoothly interfaces with DHCP management and NAT/firewalling. External VPN access can be supported if needed and multiple parallel authentication data bases are natively supported. The high-level networking “philosophy” followed by WilmaGate is the Open Access Network view, as described, for instance, in [3, 4].

Indeed, WilmaGate use is not limited to WLAN management, but can be useful for the management of any access infrastructure (also cabled), where user authentication and accounting is of any importance, and users change dynamically over time, specially if they use their own terminal.

## 1.1 Project Objectives

WilmaGate was born within the Wilma project and is now developed and maintained under the same project as well as the under the MIUR TWELVE PRIN project

(<http://twelve.unitn.it>)

and the MIUR QUASAR (Qualità e Controllabilità dei Servizi di Comunicazione su Reti Eterogenee) PRIN project.

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

The general framework of WilmaGate is that of an open test-bed, where innovative algorithms and procedures can be implemented and tested safely within an open source architecture.

Within the WILMA project, WilmaGate serves as a common framework for the management of different WLANs realized either at the University or around the city and province of Trento.

## 1.2 Use and Misuse

WilmaGate is an authentication and management tool conceived for open use and to make the use of a WLAN/HotSpot as easy and open as possible. It has not been conceived for the protection (cryptography) of users’ information, nor for access to extremely sensitive data.

Its use is recommended in scenarios where “openness” is the main issue, in scenarios where multiple users categories can be present, including “guests,” who the WLAN manager wants to grant limited access even without strong authentication. It can be safely used in public places and where public access to the Internet is granted.

It is extremely suited for the management of infrastructures used by different organizations in different times (e.g., conference halls, offices with high turn-around), since it natively interfaces with different authentication data-bases, that can be enabled at different times. It can also be efficiently used to monitor the usage of WLAN resources and identify (simple) attacks on the infrastructure.

It can be misused in several different ways. First of all it does not provide user-level data protection. This is conveniently obtained using standard SSH protocols and/or application level cryptography, and its replica within the WLAN management infrastructure is redundant. The second possible misuse is for strong protection in the access to sensitive data. The security offered by WilmaGate is the same offered by the authentication data-bases (LDAP, MYSQL, or whatever else) that are contacted by WilmaGate to grant access. For this reason the level of security and protection cannot be stronger than what provided by the underlying authentication data-bases. Indeed, one might argue that, since an additional intermediation is present, the security level is at most equal, but possibly slightly lower.

### 1.3 Companion Documents

This document is intended as a high level, not-too-technical description of the WilmaGate system and philosophy. Additional and more technical information can be found in the following documents, that are intended more for the developer that wish to include his own plug-ins, algorithms, procedures, etc. or for a system administrator that wish to install the WilmaGate to manage the access to a HotSpot or WLAN in general.

#### **WilmaGate Code Overview and Product Manual —**

This is the high-level detailed documentation of the WilmaGate system, with all its blocks, detailed installation instructions, some troubleshooting, sample configurations, etc.

It is updated often as the project develops. You can download it directly from the WilmaGate software page

(<http://netmob.unitn.it/wilmagate.html>)

or starting from the Wilma Project Home page

(<http://www.wilmaproject.org/>).

This document refers to code version 1.0; the relative documentation is freezed in [1].

#### **WilmaGate Reference Manual —**

This programmers' reference guide is dedicated to developers that want to improve, integrate, or amend WilmaGate. It is automatically generated from the code documentation through the Doxygen

(<http://www.stack.nl/~dimitri/doxygen/>)

documentation system, and it is always up-to-date with the code.

It can be downloaded together with the code from the WilmaGate software page

(<http://netmob.unitn.it/wilmagate.html>)

either downloading the most recent stable version or retrieving directly the “work in progress” from the CVS repository.

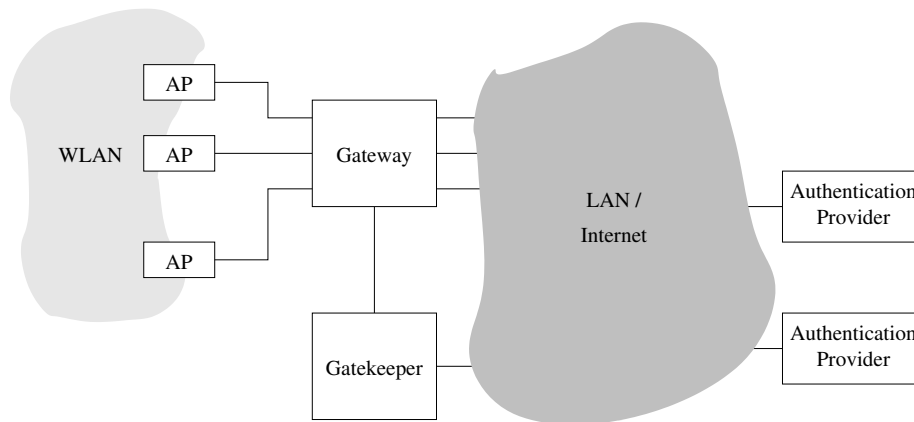


Figure 1: An overall view of the WilmaGate system.

## 2 Features and Characteristics

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 techniques;
- Firewalling;
- Accounting.

Its modular design is intended for easy implementation of novel features such as QoS control, context-dependent services and traffic shaping and modeling.

The 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; however, linux-specific optimizations, such as the implementation as kernel-space modules, are being considered.

### 2.1 Network Configuration under WilmaGate

Figure 1 shows an overall view of the proposed network components.

- The blocks “**AP**” denote wireless access points.
- The **Gateway** component is basically a layer-3 switch with some additional ad-hoc functionalities (however, a configurable router or a firewall with enough flexibility should be applicable): it checks packets that are directed from and to authorized clients and puts them into the right interface.
- The **Gatekeeper** component receives unauthorized packets from the Gateway: packets are subjected to preprocessing 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,

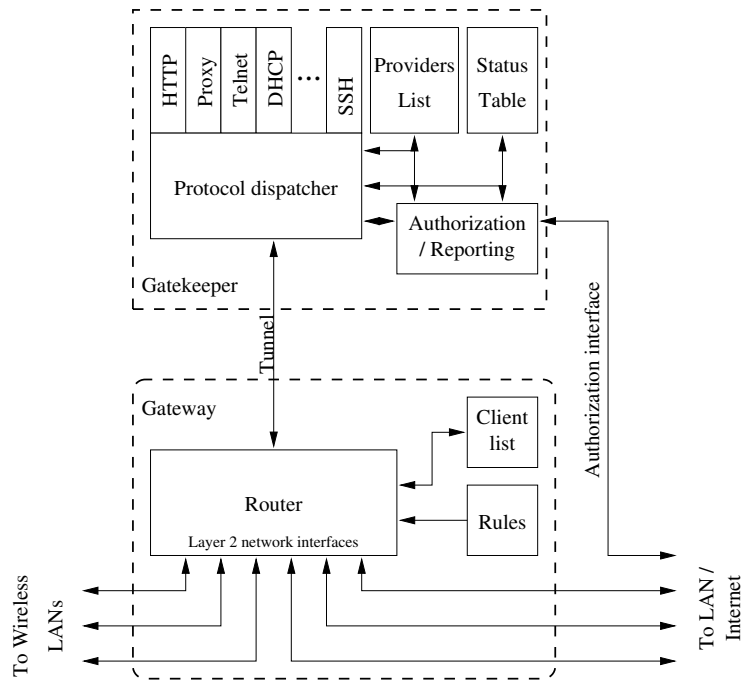


Figure 2: Block diagram of the WilmaGate system

possibly chosen among many. The role of every component and the type of interaction depend on the authentication mechanism, and shall be discussed later.

For sake of clarity, connections among components are shown as separate wires in Figure 1. However, many logical connections can be shared by separate virtual LANs on the same wire; the Gateway and Gatekeeper components can even reside in the same machine. If the AP supports an evolute operating system, the whole solution could be collapsed into a single piece of hardware: however, high performance requirements may force physical separation of the components.

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.

## 2.2 Functional Architecture

Figure 2 shows the basic block structure of the WilmaGate system. It consists of two main blocks, the *Gateway* and the *Gatekeeper*; their detailed description follows.

The Gateway machine contains various network interfaces, some of which are directed to the wireless LAN, some to the wired LAN. Packets to and from the interfaces are managed by the *router* module.

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.

The router takes decisions on packets based on the client authorization status, as

explained above, and on the packet protocol (at network and transport level); rules governing the router's behavior are stored in the *Rule Table*, created at startup and accessed by the Router via a read-only interface.

In addition to the physical LAN interfaces, the router can also send and receive packets via a UDP port connected to the Gatekeeper. In principle, all unauthorized packets 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.

The client list is updated via a "command" interface, which receives simple textual commands from the Gatekeeper.

The Gatekeeper component performs all tasks that require more processing than just looking at frame and packet headers. It basically receives all non-authorized packets from the Gateway through the "tunnel" shown in Figure 2. This tunnel is possibly implemented as a pair of UDP sockets. In case of WilmaGate implementation within a single machine, a bidirectional IPC channel could be considered. Packets are processed with the aim of authenticating the user, and eventually sent back to the Gateway to be forwarded, or bounced.

The main functions of the Gatekeeper are:

- Client status maintenance; the Gatekeeper's status table is essentially a superset of the Gateway 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 functionalities can be controlled via two TCP sockets, one devoted to client authorization functions, the other to status reporting.

### **2.3 The Providers List**

The Providers List contains a list of authorized 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 not-authenticated user: the users have not an authorization must be allowed to contact the authentication provider to receive permissions.

## 2.4 Plug-ins

The plug-in modules are used by the Dispatcher module in order to take decisions about specific protocols. They all expose a public method which is invoked by the Dispatcher in order to submit a packet; this function returns a response packet, if needed, and 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 plug-in modules can access and modify the status table.

**The DHCP plug-in** The DHCP plug-in 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 plug-in** The HTTP handles direct HTTP connection requests that bypass the Proxy server. Direct HTTP connections might be issued in case the client has disabled the proxy service.

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, usually containing instructions to correctly set the proxy server in the client system.

**The Proxy plug-in** The proxy module monitors communications between the unauthorized wireless clients and the proxy server of the fixed network. It is usually transparent, and it only acts when a client that is not authorized requests a web page which does not belong to an authentication provider. In this case, the Proxy module generates an HTTP 302 response that redirects the client to an authentication page. Moreover, the 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).

## 2.5 The “captive portal” technique

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 3). 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.

This technique is generally known as captive portal, and NoCat [2] is probably the best known open source implementation.

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 toward 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 set 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.

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 authentication 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).

The client authorization must be periodically refreshed in order to avoid IP/MAC spoofing by unauthorized users.

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, he 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.

## 2.6 Security and Privacy

No cryptography is foreseen by WilmaGate on the air channel, since this would imply that any user must have some kind or pre-set knowledge about the network before accessing it.

Privacy is normally provided by the use of secure shell level.

The integration of 802.1x techniques is left for future study, when this standard will be readily available.

## 2.7 Authentication

The authentication module is built upon the regular TCP/IP stack. It listens to the TCP port 54273 on the wired interface. It is contacted by an authorized authentication server, and accepts authentication of users.

It 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:

`authenticate 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 authentication command

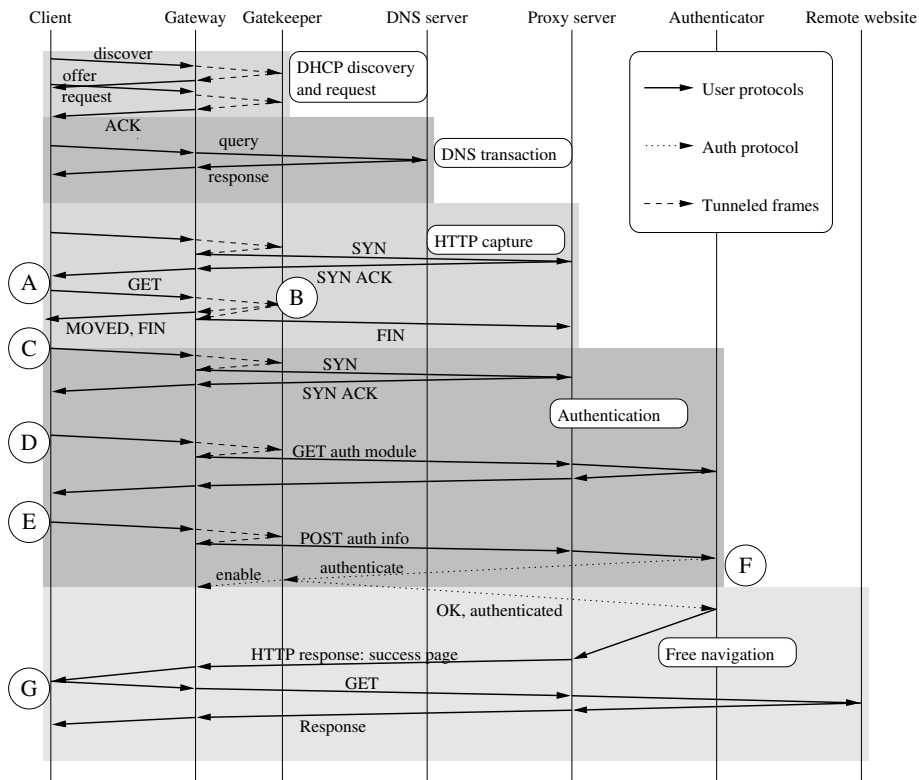


Figure 3: Simplified timing diagram from client appearance to full authentication

provides additional information for logging, i.e., the name and email of the authenticated user and a token that has been during the DHCP process and sent to the client in the early authentication phase.

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 3 describes a somewhat simplified timing diagram for the "captive portal" authentication procedure.

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 authenticated (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.

## 3 Joining the WilmaGate Test-Bed

WilmaGate is an open project, and welcomes everybody who wants to join the experimentation. WilmaGate can be joined in two different ways:

- Using it ‘as is’ for safe and secure management of WLANs and HotSpots, without the need for cumbersome client configuration and avoiding false
- Improving it and including additional feature for research purposes.

Its nature of open test bed for the experimentation of novel procedures and algorithms, makes it an ideal trial tool for many purposes, e.g., open implementations of 802.1X for WLANs with per-port/connection wireless channel cryptography.

WilmaGate software is distributed under a Free Software licence to be accepted before downloading the code.

### 3.1 Getting WilmaGate

WilmaGate is free and Open Source; however, we ask users to fulfill some administrative formalities, and to report us any anomaly, bug found, modifications and additions to the code.

Please notice that this is a research project and not a commercial operation, thus assistance should be sought only when any other possibility has been explored (in particular all available documentation read and studied) and will be issued on a best-effort policy.

### 3.2 Download and Installation

The source code is available via CVS from a server located at University of Trento. Please contact

`wilmagate@dit.unitn.it`

to get all download parameters.

After performing the source checkout, all necessary code is contained in the `wilmaproject/gateway` subfolder, which shall be referred to as the “root” directory of the program. The root directory contains the following items:

- `CVS/` is the service directory created by the CVS system to keep track of project-related items.
- `Common/` contains source code which is used both by the Gateway and by the Gatekeeper programs.
- `Gateway/` contains the Gatewayspecific source code.
- `GateKeeper/` contains the Gatekeeperspecific source code.
- `parameters/` contains the runtime configuration files for the two programs.
- `web.TEMPLATE/` contains web page templates and code for provider choice, for authentication and for system interrogation.
- `specs/` contains system documentation and specifications (and this document itself).

- `Makefile` is the overall makefile for the whole project.
- `Doxyfile` is the Doxygen configuration file for automatic API documentation.
- Some `restart` scripts to automate the execution of the programs.

## 4 Compiling the system

The system has been tested on Linux (Redhat 9 and Debian distributions) and Cygwin machines. The following packages must be installed:

- GNU C++ compiler (others have not been tested);
- Standard libraries.  
On Linux systems:
  - `glibc` (no particular requirements about version, probably also `libc5` is good, please send us feedback);
  - `libpthread` (for POSIX threads, it should be present by default);
  - `libpcap` version 0.7 or greater (for layer 2 packet capture, also present by default on many systems). Note that versions 0.6 and below miss some key functionalities.

On Cygwin systems:

- the cygwin DDLs
- `libpcap` for Windows, retrievable from

<http://winpcap.polito.it/>

The system has been tested with version 3.1beta3. Please be sure to download both the installer and the developer's pack. After installing it, please edit `Gateway/Makefile` and set the correct path in `WPCAP_PATH` for `winpcap` includes and libraries.

If all requirements are satisfied, compilation is simply achieved by typing `make` at the root directory. No errors or warnings should appear: please notify us of any compiler warning at the contact address provided above.

Compilation of the full documentation (this manual and API description) requires LaTeX and the Doxygen documentation system. The latter can be found at

<http://www.stack.nl/~dimitri/doxygen/>

RPMs and other distributions are probably included in OS installation CDs. The Cygwin version can be selected in the standard Setup program. To generate documentation, type

```
make doc
```

from the home directory. The `doc` directory shall be generated with both HTML and LaTeX contents, while the present document will be created in the `specs` directory.

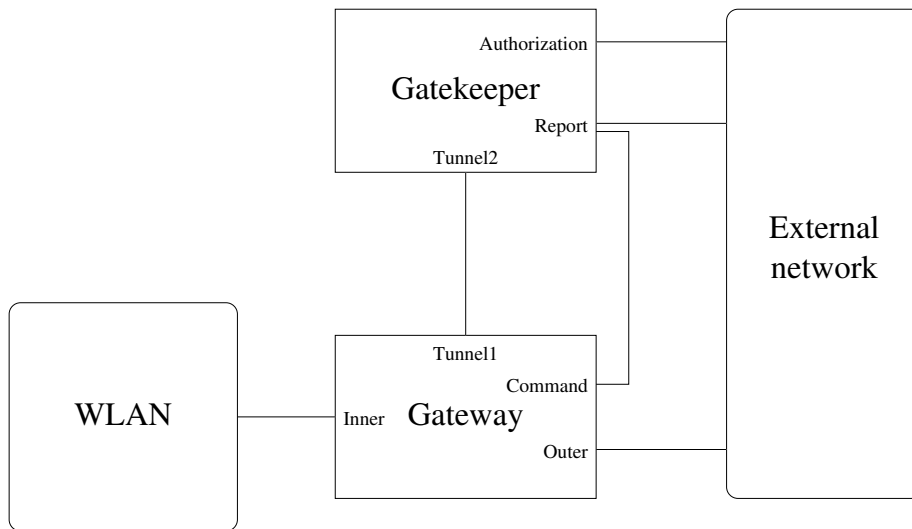


Figure 4: Reference for configuration

## 5 Configuration

Next, the system must be configured. Here you can find a brief description how to set the parameters for the system, for an in depth description you can refer to [1].

The Figure 4 shows a configuration with two different machines for Gatekeeper and Gateway, and each interface has a different name. In most systems, some of these interface correspond to the same physical device, when for instance the two components reside on the same machine.

The configuration of the system is composed by two file in the `parameters` directory. Usually you have two files with the same name and different extension: one `.gw` and the other `.gk`. Obviously the first is the configuration file for the Gateway, while the second is the configuration file for the Gatekeeper.

### 5.1 Gateway configuration

The file with `.gw` extension is the configuration file for the Gateway and contains information about:

- the description of physical interfaces and their IP addresses toward the inner network (the network that is managed by the WilmaGate);
- the description of physical interfaces and their IP addresses toward the outer network (the external network);
- the description of physical interfaces and their IP addresses toward the Gatekeeper;
- the external network parameters: for instance, IP addresses of proxy, HTTP and DNS servers;
- the runtime mode: the system can run as daemon and you can specify the name and position of log file.

## 5.2 Gatekeeper configuration

The file with `.gk` extension is the configuration file for the Gatekeeper and contains information about:

- the description of physical interfaces and their IP addresses toward external network and Gateway;
- the DHCP server: the pool of IP addresses you want to use and the lease times;
- 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: you can set static IP address for some devices (recognized by their MAC addresses).

## 5.3 Web server configuration

The “captive portal” authentication method requires the presence of one or more web servers, that must be accessible via the proxy server.

The web server must be configured so that an alias is hooked to the `web` subdirectory of the system. Three directories are contained in `web`:

- `local` contains all pages related to the first access of an unauthorized user: instructions about setting the proxy, choice of the authentication provider.
- `auth` contains all pages related to authentication and authorization renewal.
- `status` contains all pages that can be used to show the system status.

The **access server** can be located in the Gateway or in the Gatekeeper machine and it must expose the `local` directory.

If the automatic proxy configuration feature is activated, it is important to expose the `local` directory via the wireless interface of the gateway. In this case, the file with proxy parameters must be configured correctly.

The **authentication provider** must expose the subdirectory `web/auth` via a secure connection (https).

An authentication provider must contain some user authentication data. These can be provided by a RADIUS or LDAP service, by a MySQL database or in an internal table, or by any combination of such services. For instance, a centralized LDAP service can provide authentication for all regular users, while some temporary users can be managed by a MySQL table in a local machine.

Description of available services is provided in the `config/config.inc.php`, that must be configured according to the chosen authentication system.

## 6 Running the programs

From the base directory, the programs can be launched with the following commands:

```
GateKeeper/gatekeeper parameters/[gatekeeper config file].gk
Gateway/gateway parameters/[gateway config file].gw
```

It is important to start the Gatekeeper program *before* the Gateway. Both programs output a log, either to the standard output or to the specified log file, according to the specified run mode. Events such as DHCP offers and acknowledgments, authorizations and revocations are recorded in the log by the Gatekeeper. When authorization to a client is revoked, the Gatekeeper logs the traffic generated by that client during that session.

To ease program restart, the bash shell script “`restart.sh`” is provided in the main directory to be configured (`CONFIGBASENAME` variable). When launched, it will stop any gateway or gatekeeper program instance and launch the programs in the correct order. If the previous program termination left any dangling TCP socket, the script will repeatedly attempt to restart the program until socket timeouts free the required resources.

## 7 WilmaGate Evolution

WilmaGate is now a reasonably mature management environment for WLANs. It has been in use in our faculty for several months, managing an ESS of roughly 15 APs. Regular users are in the hundreds and connections per-day are on a stable rise peaking to several hundreds. It has also been used for the management of WLANs for conferences and events, as well as in the AlpiKom test-bed in Trento (see [5]) for additional details.

At the same time, it is still in its very infancy, since the potentialities of its use and its possible expansions have been barely explored.

WilmaGate development will be pursued by the Computer Networks and Mobility group at the University of Trento for several years to come, since it has been conceived as a fool to enable and foster research. Within this main scope it will remain an open source, free software, although commercial spin-offs cannot be excluded a-priori.

## References

- [1] WilmaGate: Code Overview and Product Manual (Ver. 1.0)  
<http://netmob.unitn.it/files/wilmagate-specifications-20050517.pdf>
- [2] The NoCatNet — <http://nocat.net>
- [3] M. Hedenfalk, *Access Control in an Operator Neutral Public Access Network*, MSc thesis, KTH/IMIT, Stockholm, May 2002,
- [4] R. Battiti, R. Lo Cigno, M. Sabel, F. Orava, B. Pehrson, “Wireless LANs: From WarChalking to Open Access Networks,” *Mobile Networks and Applications*, Vol. 10, 2005, pp. 275–287, Springer Science
- [5] A snapshot of the WILMA Open Lab  
[http://www.wilmaproject.org/poster\\_TRIDENTCOM\\_OpenLab.pdf](http://www.wilmaproject.org/poster_TRIDENTCOM_OpenLab.pdf)