Abstract:

The main objective of 6HOP is to design and develop optimisations for heterogeneous multihop wireless networks. The optimisation machinery is the core of the so-called Wireless Adaptation Framework (WAF), which is proposed as a remedy to the performance degradation problem Internet protocols (and in particular IPv6-based) face when these operate over a multihop environment realized with heterogeneous wireless technologies. This document mainly describes the architecture of the WAF and provides the specification of its different parts, their structure and the interfaces between them. Moreover, the behaviour of a WAF system operation is described, along with the interactions it has with other standard networking functions as the routing function.

Keyword list: Multihop wireless networks, heterogeneous wireless technologies, protocol boosting, specification.
Executive Summary

The main objective of the 6HOP project is to provide enabling communication schemes for Internet protocols, which compensate for the non-negligible packet loss rate and/or increased packet delivery delay in small-scale multihop configurations over heterogeneous wireless links. In order to accomplish this goal, the 6HOP project introduced the Wireless Adaptation Framework (WAF) as the enabling technology, which enhances the performance of the Internet protocols (IPv4/IPv6) over such links. This document is based on the requirements for WAF, as these were identified and presented in the 6HOP D2.1 deliverable [1], and provides the detailed specification of the WAF architecture along with the functional characteristics of its different parts, in order to serve as a reference document for the design and implementation of WAF.

The WAF is decomposed into three major architectural parts, namely the Local Control & Management part, the Adaptation Management part and the Packet Processing part. The main purpose of the Local Control & Management part is to provide a uniform access for management and control of the underlying mechanisms with respect to the routing, boosting and link-level control of the whole framework. It serves as both an operating system and a network interface abstraction layer by hiding certain technology peculiarities, in order to achieve portable and technology independent implementations of the higher level adaptation mechanisms (i.e. adaptation algorithms and protocols). This is achieved by the definition of the Logical Link Control Translator (LLCT) module, which manages any operating system and wireless technology peculiarities of the specific link-level management and control driver interfaces, and the Wireless API module, which provides in a uniform way to the higher level entities the services needed to monitor and control the functionality of any underlying mechanism, including the LLCTs, the Packet Processing Part and the routing function.

The actual coordination of all WAF enhancements and lower level mechanisms is achieved at the Adaptation Management part, from a number of application entities classified into managers, protocols and agents, and through the Wireless API. In the context of the 6HOP project, two managers are presented, which leverage the adaptation mechanisms offered, namely the Wireless Adaptation Manager (WAM), which mainly perform optimisations by manipulating link-level and protocol boosting parameters, and the Connection Manager (CM), which controls connections over a specific wireless device or handoffs from one wireless interface to another according to link-level information received through the Wireless API.

The compensation for wireless link impairments in the 6HOP project is mainly achieved by the usage of protocol boosting techniques, which reside in the Packet Processing part of the WAF architecture. This part is ruled by the Protocol Boosters’ Framework, which provides the necessary management support for the overall packet processing functionality, the traffic classification, the boosting policies and their parameters, while the rest of the part consists of a number of actual Protocol Booster modules, with indicative representatives for the 6HOP project, the Forward Error Correction (FEC) and the Robust Header Compression (ROHC) for TCP.
References & Standards


[2] IEEE 802.x Series of Standards for Metropolitan and Local Area Networks


http://www.bluetooth.com


# List of Contents

1 INTRODUCTION TO WIRELESS ADAPTATION FRAMEWORK (WAF) ..........9

1.1 GENERAL INTRODUCTION ................................................................. 9

1.2 COMMUNICATION REQUIREMENTS ................................................. 9

1.2.1 Optimisation Requirements for Multihop Environment .................. 10

2 WAF HIGH-LEVEL ARCHITECTURE DEFINITION ..............................12

2.1 ARCHITECTURAL CONSIDERATIONS ........................................... 12

2.2 ARCHITECTURE DECOMPOSITION ................................................ 12

2.2.1 Local Control and Management Part ........................................... 13

2.2.2 Adaptation Management Part ..................................................... 13

2.2.3 Packet Processing Part ............................................................... 14

2.3 CONTROL/DATA FLOW ACROSS THE WAF ................................. 14

2.3.1 WAF Control / Data Plan ............................................................ 14

2.3.2 Internal Signalling ................................................................. 15

3 WAF LOCAL CONTROL AND MANAGEMENT ..................................16

3.1 INTERNAL STRUCTURE ................................................................. 16

3.2 WIRELESS API .............................................................................. 18

3.2.1 Model Definition ................................................................. 18

3.2.2 Service Specification .............................................................. 19

3.3 BLUETOOTH LLCT .................................................................... 29

3.3.1 Interfaces definition ............................................................... 29

3.3.2 Sequence diagram ............................................................... 32

3.4 WiFi LLCT .................................................................................. 33

3.4.1 Event detecting functions ....................................................... 33

3.4.2 Link–level statistics retrieval .................................................. 33

3.4.3 Managing link–level parameters ............................................ 33

3.5 INTERFACE WITH ROUTING INFORMATION / PROTOCOLS ..........34

4 WAF ADAPTATION MANAGEMENT .................................................36

4.1 WIRELESS ADAPTATION MANAGER (WAM) ...............................37

4.1.1 Network Functionality ............................................................. 37

4.1.2 Internal Signalling ............................................................... 37

4.1.3 External Signalling ............................................................... 38

4.1.4 Adaptation Mechanisms ....................................................... 39

4.2 CONNECTION MANAGER (CM) .................................................. 41

4.2.1 Handoffs in heterogeneous networks ...................................... 41

4.2.2 Handoff from Bluetooth to WLan network ............................. 42

4.2.3 Handoff from WLan to Bluetooth networks ............................ 43

4.2.4 Handover Optimisation ....................................................... 43

5 WAF PACKET PROCESSING ............................................................45

5.1 PROTOCOL BOOSTERS’ FRAMEWORK ........................................ 45

5.1.1 Architecture ........................................................................ 45

5.1.2 Interface with the Wireless API ............................................. 46
5.1.3 Interface with the Protocol Boosters ................................................................. 47
5.1.4 Interface with the Network Layer ................................................................. 47
5.1.5 WAF Data Frame Format .............................................................................. 49

5.2 FORWARD ERROR CORRECTION .................................................................. 49
5.2.1 Overview ........................................................................................................ 49
5.2.2 FEC protocol booster .................................................................................. 50

5.3 ROBUST HEADER COMPRESSION FOR TCP ............................................. 51
5.3.1 Overview ........................................................................................................ 51
5.3.2 ROHC-TCP protocol booster operation ...................................................... 51
5.3.3 TCP/IP field behavior .................................................................................. 53
5.3.4 ROHC Packets and packet types ................................................................. 55
5.3.5 Initialisation of TCP/IPv6 headers ............................................................... 60

6 ILLUSTRATION OF WAF OPERATION ....................................................... 63
6.1 WAF AND MULTIHOP ROUTING CO-OPERATION ..................................... 63
6.2 EXAMPLE 1: NODE WITH A SINGLE WIFI NETWORK INTERFACE .............. 64
6.3 EXAMPLE 2: MULTIHOMED NODE WITH BLUETOOTH AND WIFI NETWORK INTERFACES ...... 66

7 CONCLUSION .................................................................................................. 69

8 APPENDIX ......................................................................................................... 70
8.1 NETFILTER USE FOR PROTOCOL BOOSTERS FRAMEWORK DESIGN ............ 70
8.1.1 Netfilter overview ....................................................................................... 70
8.1.2 Netfilter within the Bridge driver .............................................................. 71
8.1.3 Netfilter hooks used by the Protocol Boosters Framework ......................... 72
List of Figures

Figure 1. WAF high-level architecture ................................................................................................ 13
Figure 2. Internal architecture of Local Control & Management Part .................................................. 17
Figure 3. Wireless API Data Model ..................................................................................................... 18
Figure 4: Bluetooth LLCT class diagram ............................................................................................ 30
Figure 5: Bluetooth sequence diagram ............................................................................................... 32
Figure 6. WAF Adaptation Management Part ....................................................................................... 36
Figure 7 WAF Boosting Capacities Discovery ..................................................................................... 38
Figure 8 WAF header structure ........................................................................................................... 39
Figure 9 Structure for neighbour capability discovery primitives ........................................................ 39
Figure 10 Structure for remote node capability discovery primitives ................................................... 39
Figure 11. WLAN-Bluetooth Interconnection Architecture .................................................................... 42
Figure 12 PB framework and its interfaces .......................................................................................... 46
Figure 13 Packets traversals across PBF ............................................................................................. 48
Figure 14 WAF Data PDU Format ....................................................................................................... 49
Figure 15. FEC specific header ............................................................................................................. 50
Figure 16: General Compression Scheme ............................................................................................. 52
Figure 17: Compression of IPv4,6 header ............................................................................................. 52
Figure 18: Compressor States ................................................................................................................. 52
Figure 19: Decompressor States ............................................................................................................. 53
Figure 20: ROHC general packet format ............................................................................................... 55
Figure 21: Padding octet ......................................................................................................................... 56
Figure 22: ADD-CID octet ....................................................................................................................... 56
Figure 23: IR packet format .................................................................................................................... 57
Figure 24: IR-DYN packet format .......................................................................................................... 57
Figure 25: IR-REPLICATE packet format .............................................................................................. 58
Figure 26: General feedback packet format .......................................................................................... 59
Figure 27: Feedback data format .......................................................................................................... 59
Figure 28: Feedback types .................................................................................................................... 59
Figure 29: IPv6 header format .............................................................................................................. 60
Figure 30: TCP header format .............................................................................................................. 61
Figure 31: Multihop wireless network arrangement............................................................................ 65
Figure 32: Bridge defined netfilter hooks ............................................................................................. 71
Figure 33: Netfilter hooks used to call protocol boosters ................................................................. 72
List of Tables

Table 1. Primitive Set of the Application Service Interface (Wireless API) ........................................ 20
Table 2. Primitive Types and parameters of the WAF_ServicesAvailable service ............................... 20
Table 3. Primitive Types and parameters of the WAF_Service_Start service .................................... 21
Table 4. Primitive Types and parameters of the WAF_Service_Stop service ...................................... 22
Table 5. Primitive Types and parameters of the WAF_Service_DiscoverNeighbours service ............. 22
Table 6. Primitive Types and parameters of the WAF_Neighbour_Connection service ...................... 23
Table 7. Primitive Types and parameters of the WAF_Neighbour_Disconnection service ................. 23
Table 8. Primitive Types and parameters of the WAF_Node_Detection service .................................. 24
Table 9. Primitive Types and parameters of the WAF_Node_Dissappearance service ....................... 24
Table 10. Primitive Types and parameters of the WAF_Connection_AddTrafficPolicy service ........... 25
Table 11. Primitive Types and parameters of the WAF_Connection_RemoveTrafficPolicy service .... 25
Table 12. Primitive Types and parameters of the WAF_LLCConnection_ChannelRateChanged service .......................................................................................................................... 26
Table 13. Primitive Types and parameters of the WAF_LLCConnection_ChannelQualityChanged service .......................................................................................................................... 26
Table 14. Primitive Types and parameters of the WAF_AddTrafficClass service .............................. 27
Table 15. Primitive Types and parameters of the WAF_RemoveTrafficClass service ........................... 28
Table 16. Primitive Types and parameters of the WAF_Attribute_Get service ................................... 28
Table 17. Primitive Types and parameters of the WAF_Attribute_Set service ................................... 29
Table 18: Classification of IPv6 header fields ...................................................................................... 53
Table 19: Classification of TCP header fields ...................................................................................... 54
Table 20: Commonly used TCP options .............................................................................................. 55
1 INTRODUCTION TO WIRELESS ADAPTATION FRAMEWORK (WAF)

1.1 GENERAL INTRODUCTION

The main objective of the 6HOP project is to provide enabling communication schemes for Internet protocols, which compensate for the non-negligible packet loss rate and/or increased packet delivery delay in small-scale multihop configurations over heterogeneous wireless links. In order to accomplish this goal, the 6HOP project introduced the Wireless Adaptation Framework (WAF) as the enabling technology, which enhances the performance of the Internet protocols (IPv4/IPv6) over such links. The requirements for WAF with respect to its application to heterogeneous multihop wireless communication networks have been identified and presented in the D2.1 6HOP deliverable [1]. Based on these requirements, the role of the present document is to provide the detailed specification of the WAF architecture along with the functional characteristics of its different parts, and to serve as the guide document for the reference design and implementation of WAF in the context of the 6HOP project.

The document is structured as follows: the rest of this introductory section reviews the communication and optimisation requirements of the Wireless Adaptation Framework (WAF). Section 2 describes the high-level architecture of the framework and introduces its decomposition into three major parts, namely the Local Management & Control part, the Adaptation Management part and the Packet Processing part. Moreover, it outlines the basic characteristics of the different data & control (signalling) flows inside WAF, both internal – i.e. between the architectural parts – and external – i.e. between WAF capable nodes. Sections 3, 4 and 5 describe in detail the internal structure of each one of the above mentioned parts, providing the specification of their operation and of their interfaces. The dynamic operation of a complete WAF system is outlined in section 6, along with the interactions between WAF and different routing mechanisms. Finally section 7 concludes the document, while in the appendix (section 8) a number of design considerations are presented, which refer to the Linux OS support of the Protocol Boosters’ Framework (PBF) appearing inside the WAF Packet Processing part.

1.2 COMMUNICATION REQUIREMENTS

The requirements for an adaptive framework in a wireless communication environment were identified and thoroughly discussed in [1], while they are also summarized below:

- **Compensation for Wireless Link Impairments:** Wireless links in a wireless network may exhibit packet error rate, high latency, or throughput constraints. These characteristics are due to the impairments of the wireless channel. It is supposed that those are compensated by the MAC and/or link layer protocols of each one wireless interface. Existing wireless technologies support different technologies for channel coding, medium access, and error control. If these bearer services are implemented without considering IP traffic requirements, then, whenever IP traffic is layered over these wireless technologies, the obvious result is performance degradation (throughput decrease and/or not latency increase). Thus, complementary machinery is needed to compensate for wireless link impairments, targeting various network functions such as:
  
  o Channel coding,
• Error control,
• Spectral efficiency,
• Routing management functions,
• Mobility management functions.

- **Uniform Wireless Interface:** There exist wireless interfaces with different features, e.g., physical layer, MAC protocol, error control, connection management, etc. Furthermore, these may be coupled with even different link layer protocols. This compilation of wireless drivers and link layer protocols should be accessed from upper layer protocols and applications for control purposes (the same way upper layer protocols and applications access the protocol stack through the socket interface for data purposes). Thus, a common interface is required for wireless drivers and lower layer protocols to be uniformly accessed by upper layer protocols. Such a common interface should support the necessary service primitives for (1) configuration of wireless drivers and link layer protocols, (2) retrieval of statistics, and (3) event handling. Moreover, this uniform wireless interface should facilitate the operation of wireless-aware applications and routing/mobility management protocols.

- **Power Consumption Mitigation:** Wireless Terminals (WTs) such as laptops, palmtops, and PDAs exhibit an upper bound to their uptime operation due to power (battery) constraints. In fact, the power consumption is directly related to the processing tasks the device runs at any one time, as well as, the radiating power. Whenever, a WT is connected to a network, its processing tasks are mainly devoted to communication. As a result, the greater the communication effort is the greater the power consumption is, which in turn, leads to battery exhaustion. Mitigations techniques should be enforced to compact power consumption up to a certain level.

- **IPv6 Support:** IPv6 protocol adoption in a wireless network presents a number of advantages against the traditional IPv4, as huge address space (highly important if we assume a huge number of wireless terminals in future deployments), better support for routing and mobility management protocols due to its inherent header structure etc. The support of IPv6 in a Multihop wireless network implies in turn a number of requirements by itself, as
  - Addressing (that is, how wireless nodes are assigned IPv6 addresses in a wireless network), and
  - Transition mechanisms (that is, how an IPv6 wireless network is seamlessly integrated with legacy IPv4-based networks).

### 1.2.1 Optimisation Requirements for Multihop Environment

Considering the multihop variant of wireless networks, further specific requirements are imposed:

- **Packet Forwarding:** In a multihop wireless network some nodes should relay packets in order to assist the communication needs of neighbouring nodes, which, for example, are located at coverage holes, or cannot directly communicate with peer wireless nodes, leading actually to multihop routing. In fact, multihop routing is a core function in a multihop wireless network, where both a cellular structure exists and instant communication is highly required. In order to apply efficient optimisations targeting to this particular networking function, certain assumptions have been made in the context of the 6HOP project, that is
The number of wireless hops should be kept relatively low (e.g. in the order of 10) so as to avoid potential risks to overall system performance due to complex protocol design.

Host mobility should be also considered small, i.e., rapid changes in network topology due to nodes’ mobility are not considered in 6HOP.

- **End-to-End Signalling:** It may be desirable for some optimisation protocols (e.g., header compression) to operate on an end-to-end basis whenever communication is realized over multiple wireless hops. This imposes additional requirements for
  
  - Signalling between the sender and receiver running the particular protocol. Signalling needed for the coordination of protocol behaviour on wireless nodes should be as lightweight as possible, yet efficient, so as to avoid protocol overheads, which in turn, may jeopardize the overall system performance
  
  - Forwarding intelligence on intermediate nodes.
2 WAF HIGH-LEVEL ARCHITECTURE DEFINITION

2.1 ARCHITECTURAL CONSIDERATIONS

6HOP introduces the Wireless Adaptation Framework (WAF), being an efficient architecture designed specifically for use in heterogeneous wireless multihop networks. The goal of this design is for WAF to serve as an enabling technology for IPv4/IPv6 communications over existing and emerging WLAN/ WPAN technologies in both single-hop and multihop configurations (this implies either typical access to remote servers as well as peer-to-peer communications). WAF is targeted to address a subset of the requirements mentioned in [1], as well as, to enable other (complementary) technologies to be coupled. This will result to the overall fulfilment of the imposed requirements.

In short, the following considerations apply to WAF:

- WAF is an enabling technology. That is, it controls the operation of various wireless devices, while it facilitates the operation of network and application layer protocols.
- WAF does not perform any routing decisions. Routing is a network function provided by some external machinery. On the other hand, WAF does facilitate routing decisions by exporting hints for link conditions or link layer protocol status to the appropriate protocols.
- WAF controls the operation of link layer protocols targeted to shield wireless transmissions. These protocols, namely protocol boosters, are fully controllable by WAF through the appropriate interfaces.
- WAF controls the wireless drivers through appropriate interfaces.
- WAF supports both WAF-capable and non-WAF capable nodes.
- WAF finds out whether or not a neighbour node is WAF-capable.

2.2 ARCHITECTURE DECOMPOSITION

Figure 1 illustrates the WAF high-level architecture. As it can be seen, a WAF-enabled station will have to incorporate three major parts, namely the

- WAF Local Control & Management Part
- WAF Adaptation Management Part
- WAF Packet Processing Part

described in the following sub-sections.
2.2.1 Local Control and Management Part

This is a core component of WAF. Its role is to provide a uniform interface (Wireless API: IF1 in Figure 1) to WAF-enabled network applications (daemons) and protocols, so as to drive their operation. It provides configuration, statistics, and event handling services, by masking the interfaces to underlying wireless drivers (IF5), protocols boosters (IF6), and routing data and parameters (IF2). It does not contain any adaptation intelligence (i.e. algorithms and policies). Instead, it provides access to certain lower layer characteristics which can be used as feedback / control points to such adaptation mechanisms implemented in higher-level entities. The importance of the uniform application interface (Wireless API) must be highlighted since this actually enables the WAF as an open and extensible framework to future incorporation of adaptation mechanisms other than those provided as pilot implementations in the context of the 6HOP project. Furthermore, the internal design of this component and its interfaces with the lower level entities it controls have to be in a way which also enables its extensibility to future protocol enhancing and wireless communication technologies. It is described in more detail in Section 3.

2.2.2 Adaptation Management Part

The coordination of all WAF enhancements and lower level mechanisms is achieved at the application layer, through a number of application entities (e.g. networking daemons). These entities may be classified into Managers, Protocols and Agents with respect to the kind of functionality and service they add to the system / network, as follows:
• **Manager(s):** These components use the Wireless API so as to perform particular control functions and adaptations on other existing components or protocols. Such functions are configuration, statistics retrieval, event handling, neighbour capability discovery, connection management (e.g. vertical handover), power management, external routing protocol adaptations etc. Core managers for the WAF architecture in the context of the 6HOP project are the Wireless Adaptation Manager (WAM) and the Connection Manager (CM), as these are described in detail in Section 4.

• **Protocol(s):** These are complete implementations of networking protocols, as routing and mobility management protocols, which use the WAF extensions in order to adapt their operation according to the local or network status at any moment.

• **Agent(s):** These are mainly network management agents, with main purpose to extend the accessibility of existing network management protocols (e.g., SNMP) to the new parameters and statistics “namespace” introduced by WAF.

### 2.2.3 Packet Processing Part

This part of the architecture is targeted to packet processing during transmission/reception to/from the wireless driver, so as to enhance performance. Thus, it intercepts into the ordinary IP / Link-level interface, through IF3 and IF4 according to Figure 1, in order to apply the appropriate Protocol Boosters on sender and/or on receiver according to the application traffic requirements and the characteristics of the particular wireless device in use. Forward Error Correction (FEC), and Robust Header compression (ROHC) for TCP are indicative examples of protocol boosters, and they are described in Section 5.

### 2.3 CONTROL/DATA FLOW ACROSS THE WAF

In the whole WAF architecture, three different communication flows have to be considered, namely:

- **Data traffic**, generated from a user application that needs to send something to another terminal.
- **Signalling traffic**, used between application-level WAF entities and exchanged on a peer to peer manner, and
- **Internal Signalling**, used by WAF entities inside the same terminal to establish, update and manage operation parameters.

### 2.3.1 WAF Control / Data Planes

In [7], a protocol operation has been defined, targeted to the enhancement of TCP-UDP/IP protocols when used over a wireless infrastructure, but only considering a single hop. In that case, the overhead derived from the large number of signalling packets and procedures was affordable, but this is not the case for 6HOP, in which a higher number of wireless hops is assumed. One of the main requirements that have to be covered in WAF is the reduction of the overhead introduced by any signalling procedures and protocol operation used between WAF entities. In order to reduce WAF signalling overhead, WAF-enabled communications can be seen as a distributed communication scheme, in which no entity acts as a master, and each node decides, independently, which protocol boosting
modules are to process each data packet. In order to assure that the receiving node will be able to invert the procedure, a new Ethernet type (i.e. a WAF Type) frame has to be defined, to facilitate the (de)multiplexing of standard IP (non boosted) and WAF (boosted) traffic, while for the latter, the applied booster sequence and parameters decided at the sender have to be included inside the WAF frame header. The WAF data frame format is described in more detail in section 5.1.5. As can be seen, using this scheme, similar to the extension headers approach used by IPv6, neither further encapsulation nor any kind of connection establishment are needed. In contrast, each node decides the boosting modules the data packet is to traverse, taking into account traffic characteristics and its neighbours’ capabilities (in the downstream direction) or the boosters related information in the WAF header of the incoming frame (in the upstream direction).

Furthermore, in order to ease the implementation and operation of the WAF, the signalling procedures to be performed between different higher WAF entities (i.e. with reference to Figure 1, between the WAF Adaptation Management parts of different nodes) are going to use traditional BSD sockets (UDP, so as to avoid TCP overhead). This signalling actually is reduced only to a capability discovery / change signalling protocol, and described in more detail in section 4. Moreover, no boosting modules are going to be applied to this type of traffic, hence, the Ethernet type field of these packets is set to the traditional 0x0800 (IP traffic) and therefore, in the reception procedure, they will not traverse the WAF Packet Processing part, but they will be processed as IP datagrams addressed to an application running on that terminal.

### 2.3.2 Internal Signalling

Figure 1 shows a number of different local communication interfaces amongst WAF entities, while they are all “masked” at the application interface by the Wireless API (IF1), through which, the user can switch the requirements for a particular type of traffic “on the fly”, as the WAF Control Part interfaces directly with the internal structures of the Packet Processing part used to manage the WAF protocol boosting operations (IF6). Moreover, similar changes may be directed through the Wireless API towards routing tables/parameters (IF2) and link-level parameters (IF5).
3 WAF LOCAL CONTROL AND MANAGEMENT

The main purpose of this part of the WAF architecture is to provide a uniform access for management and control of the underlying mechanisms with respect to the routing, boosting and link-level control of the whole framework. It serves as both an operating system and a network interface abstraction layer by hiding certain technology peculiarities, in order to achieve portable and technology independent implementations of the higher level adaptation intelligence (i.e. adaptation algorithms and protocols).

As it can be seen, and from the point of view of a Network Interface abstraction layer, the WAF at its user service interface will not have to intervene with the data stream coming from or to the higher applications or other higher WAF protocol machinery (WAF Adaptation Management part). Instead, higher level WAF entities have to use directly the standard BSD socket interfaces to the selected network interface, while the services offered by this part only deal with the control / management / monitoring of the underlying infrastructure. The latter may be achieved by mirroring or wrapping either WAF related parameters and statistics or wireless network relative information which is usually modelled within Layer Management Entities of the underlying (wireless) networking technologies, according to the standard IEEE 802.x terminology [2]. Nevertheless, main data transport operations are affected through the services offered, as this happens by the exposure of certain boosting and routing parameters.

From the point of view of an OS abstraction layer, the specification of the interface exposed by this part has to be in such an abstraction level which allows its porting to dominant different operating systems (i.e. abstract enough) while ensuring a possibly needed portability of applications which may use it (i.e. not too abstract). The latter imposes that some form of implementation guidelines should be embedded to the definition of this interface along with the abstract definition of the services offered. This is the main reason of coming up with the need of a specific definition of an API and not only an abstract service definition of an interface. Of course, the implementation of the underlying functionality exposed by this API and the internal details of interfacing with the other architectural components have to be explicitly engineered for each OS separately, utilizing the most efficient (in any metrics) internal mechanisms of a specific OS.

3.1 INTERNAL STRUCTURE

In Figure 2, the internal structure and the interfaces of the local management & control part are depicted. In order for the WAF architecture to be extendible with future wireless communication technologies as well as with boosting algorithms other than those presented in the context of the 6HOP project, a separation of functionality in distinct modules with accurately defined interfaces is needed. Thus, this part consists of:

- The **Wireless API Module**, which is actually an aggregator of the functionality of the lower level, OS or network technology dependent interfaces. It provides access:
  - to routing related information (through IF2), either to standard routing structures (i.e. routing tables) or to other adaptive parameters of IP-level routing protocol implementations (i.e. MANET routing protocols). The description of IF2 is presented in section 3.5.
to packet processing parameters (through IF6), either on boosting policies (i.e. booster processing chains and traffic classification) or on boosting algorithm specific parameters. The description of IF6 is presented in section 5.1.2.

to link-level parameters. Because different communication and link-level networking technologies may have completely different characteristics and ways of discovering neighbour nodes, connecting, getting link quality etc, this information is accessed through an intermediate translator module, namely

- The **Logical Link Control Translator (LLCT)**. Each LLCT is aware of one technology (e.g. 802.11, Bluetooth), exporting one or more services (e.g. PAN or LAN for Bluetooth, ESS or Ad-Hoc for 802.11) and masking the OS and some wireless technology peculiarities of the specific driver interfaces (IF5). Proper design should provide a uniform way to manipulate the functions exposed from the technology-specific LLCTs (interface IF7) to the Wireless API module. Descriptions of the interfaces exposed by a Bluetooth LLCT and a WiFi LLCT are presented in sections 3.3 and 3.4 respectively.

![Figure 2. Internal architecture of Local Control & Management Part](image-url)
3.2 **WIRELESS API**

### 3.2.1 Model Definition

In Figure 3, a model of the information related to the Wireless API module and exchanged at the API interface is presented. The presented model mainly aims to identify and relate information and to provide a context for the definition of the parameters of the abstract service primitives offered by the Wireless API.

**Figure 3. Wireless API Data Model**

- **Service**: It abstracts the different services which become available from the wireless technologies under the control of the LLCTs. Encoded on its Type attribute, is the kind of the service offered, e.g. SS, DSS (for WiFi), PANU, GN, NAP (for Bluetooth) while the State attribute encodes the status of the service and may be UP or DOWN.

- **LLCTSpecificServiceParams**: The technology specific parameters (i.e. the specific configuration of the servicing LLCT).

- **Neighbour / LLCConnection**: Abstract the status and the characteristics applied to the link-level (first hop) connection between the local node and a Neighbour node for a specific
Service. It is assumed that there may be one and only one such LLCConnection between the local node and one Neighbour node, therefore, the LLCConnection ID attribute is the Neighbour Node ID (which may be the MAC address actually).

- **RemoteNode / MultihopConnection:** Abstract a multi-hop connection between the local node and a RemoteNode. It is assumed that, in this context, there may be one and only one such MultihopConnection between the local node and one RemoteNode, for all possible transport streams between them (at least, for each IP address in the RemoteNode, i.e. the MultihopConnection abstracts the route to the RemoteNode) therefore, the Connection ID attribute is the RemoteNode Node ID (which may be its IP address actually).

- **LLCTSpecificConnectionParams:** The technology specific parameters and statistics related to a link layer connection.

- **Channel:** Abstracts the information (parameters and statistics) which is related to the medium characteristics over a specific LLCConnection.

- **TrafficPolicy / TrafficClass:** Abstract the information related to the different booster chains applied over the different classes of data streams which are multiplexed either over a link-level LLCConnection with a Neighbour node or over a MultihopConnection with a RemoteNode.

- **Booster:** Abstracts the information which is related to the defined booster types in the 6-HOP project. The Type attribute is one of FEC, ROHC.

- **BoosterCustomParams:** Abstracts the algorithm specific parameters and statistics of a Booster which apply to a certain usage of the Booster from a TrafficPolicy.

### 3.2.2 Service Specification

This section presents the abstract definition of the services offered at the Wireless API, as a specification of their primitives and their parameters. The set of services supported is presented in Table 1, where the encoding is as follows: the absence of any symbol means that the specific type of primitive is not defined for that service, an ‘M’ symbol means that the primitive is mandatory in any implementation of the Wireless API and a ‘C’ symbol means that the parameter is conditional, that is, it may or may not be supported in a certain implementation of the library or a certain configuration of a system (i.e. functionality not supported from the available drivers).

For each service, a table is provided in which the types of the defined primitives (i.e. request, confirmation or indication) and their parameters are encoded as follows: absence of any symbol means that the specific parameter is not defined (does not exist) in that primitive, an ‘M’ symbol means that the existence of the parameter is mandatory in any activation of that primitive and a ‘C’ symbol means that the parameter is conditional, that is, it may or may not exist in a certain implementation or in a usage context of that primitive.

Since the defined services are local and there is not any external protocol between two Wireless API modules, the ‘response’ primitive is absent in all service definitions.
Table 1. Primitive Set of the Application Service Interface (Wireless API)

<table>
<thead>
<tr>
<th>Service</th>
<th>Request</th>
<th>Confirm</th>
<th>Indicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_ServicesAvailable</td>
<td>M</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td>WAF_Service_Start</td>
<td>M</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td>WAF_Service_Stop</td>
<td>M</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td>WAF_Service_DiscoverNeighbours</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WAF_Neighbour_Connection</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>WAF_Neighbour_Disconnection</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>WAF_Node_Detection</td>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAF_Node_Dissappearance</td>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAF_Connection_AddTrafficPolicy</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WAF_Connection_RemoveTrafficPolicy</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>WAF_LLCConnection_ChannelRateChanged</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAF_LLCConnection_ChannelQualityChanged</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAF_AddTrafficClass</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WAF_RemoveTrafficClass</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WAF_Attribute_Set</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WAF_Attribute_Get</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

3.2.2.1 WAF_ServicesAvailable

- **Function**

By the usage of this service, higher layer entities (applications, WAF daemons etc.) are informed about the different networking services available locally, through which they can reach neighbour nodes. Available wireless networking services are considered these installed and loaded in the current system configuration, but not necessarily activated (e.g. Service::State may be DOWN. In contrast, and as another example, the removal of a PCMCIA card triggers the Wireless API module to “destroy” the availability of the related Services). The user of the service may get the availability information either on request (by using the confirmed request primitive) or in an unsolicited way, by an indication reception on a detection of a configuration changing condition (as in the PCMCIA insertion/removal example above).

- **Types of primitives and parameters**

  Table 2. Primitive Types and parameters of the WAF_ServicesAvailable service

<table>
<thead>
<tr>
<th>WAF_ServicesAvailable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Name</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>WAF_Services</td>
</tr>
<tr>
<td>WAF_Result</td>
</tr>
<tr>
<td>WAF_Reason</td>
</tr>
</tbody>
</table>

- **Definition of Parameters**
o WAF_Services: The set of Service entities (as currently described in the model presented in Figure 3) which identify the different networking services currently available from the installed hardware and its drivers (through the corresponding LLCT).

o WAF_Result: Encodes the result of the previously issued request primitive and may be WAF_OK or WAF_ERROR_<#Num>.

o WAF_Reason: Encodes the event that caused the activation of the indication primitive and may be WAF_SERVICE_ADDED, WAF_SERVICE_LOST, or WAF_ERROR_<#Num>.

3.2.2.2 WAF_Service_Start

• Function

This service provides to the higher layer entities the capability to activate (or to be informed about an automatic activation of) a currently available networking service.

• Types of primitives and parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
<th>Indicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF-Service</td>
<td>M</td>
<td>C</td>
<td>M</td>
</tr>
<tr>
<td>WAF_Result</td>
<td>M</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

• Definition of Parameters

o WAF_Service: A Service entity, either as currently described in the model presented in Figure 3 or as a unique identifier of such an entity. The conditionality of the presence of this parameter in the confirmation primitive, as indicated in the table above, depends on the implementation, and especially on whether the Wireless API module can handle concurrently pending requests, or each request must be confirmed before the application can issue a subsequent request. In the first case the parameter presence should be considered mandatory, in the latter case it should be considered as an option (it may or may not exist).

o WAF_Result: The result of the previously issued request primitive or the result of an automatic attempt (from the WAF sub-system) to activate a Service. May be WAF_OK or WAF_ERROR_<#Num>.
3.2.2.3 WAF_Service_Stop

- **Function**
  This service provides to the higher layer entities the capability to deactivate (or to be informed about an automatic deactivation of) a currently available wireless networking Service.

- **Types of primitives and parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
<th>Indicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF-Service</td>
<td>M</td>
<td>C</td>
<td>M</td>
</tr>
<tr>
<td>WAF_Result</td>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAF_Reason</td>
<td></td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>

- **Definition of Parameters**
  - WAF_Service: The Service entity in question (as in WAF_Service_Start service description).
  - WAF_Result: Encodes the result of the previously issued request primitive and may be WAF_OK or WAF_ERROR_<#Num>.
  - WAF_Reason: Encodes the reason that caused the automatic deactivation of the networking Service and may be WAF_ERROR_<#Num>.

3.2.2.4 WAF_Service_DiscoverNeighbours

- **Function**
  By the usage of this service, higher layer entities are informed about the set of the neighbour nodes which are directly accessible at the link layer (in one hop).

- **Types of primitives and parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_Service</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td>WAF_Neighbours</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WAF_Result</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

- **Definition of Parameters**
  - WAF_Service: The Service entity (or its unique ID) in question.
  - WAF_Neighbours: The current set of neighbour nodes which belong to the same Service.
3.2.2.5 WAF_Neighbour_Connection

- **Function**
  
  This service provides to the higher layer entities the capability to create a link-level connection with a neighbour node or be informed about an automatic link-level connection establishment, on request of this or the neighbour node.

- **Types of primitives and parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
<th>Indicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_Neighbour</td>
<td>M</td>
<td>C</td>
<td>M</td>
</tr>
<tr>
<td>WAF_Result</td>
<td></td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>

- **Definition of Parameters**
  
  o WAF_Neighbour: The neighbour description (or its unique ID) of the connected neighbour (or the neighbour that is to be connected).
  
  o WAF_Result: The result of the request primitive, encoded as WAF_OK, WAF_TIMEOUT or WAF_ERROR_<#NUM>.

3.2.2.6 WAF_Neighbour_Disconnection

- **Function**
  
  This service provides to the higher layer entities the capability to destroy a link-level connection with a neighbour node or be informed about an automatic link-level disconnection, on request of this or the neighbour node.

- **Types of primitives and parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
<th>Indicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_Neighbour</td>
<td>M</td>
<td>C</td>
<td>M</td>
</tr>
<tr>
<td>WAF_Result</td>
<td></td>
<td></td>
<td>M</td>
</tr>
<tr>
<td>WAF_Reason</td>
<td></td>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>

- **Definition of Parameters**
- WAF_Neighbour: The neighbour description (or its unique ID) of the disconnected neighbour (or the neighbour that is to be disconnected).
- WAF_Result: The result of the request primitive, encoded as WAF_OK or WAF_ERROR_<#NUM>.
- WAF_Reason: A possible reason of the neighbour disconnection which caused the activation of the indication primitive.

### 3.2.2.7 WAF_Node_Detection

- **Function**
  
  By the usage of this service, higher-level entities may be informed about a new gateway (Neighbour) or a new final destination (RemoteNode) detected by the routing function.

- **Types of primitives and parameters**

  **Table 8. Primitive Types and parameters of the WAF_Node_Detection service**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_Node</td>
<td>M</td>
</tr>
</tbody>
</table>

- **Definition of Parameters**
  
  - WAF_Node: A Node description (Neighbour or RemoteNode) of the detected node.

### 3.2.2.8 WAF_Node_Dissappearance

- **Function**
  
  By the usage of this service, higher-level entities may be informed from the routing machinery that all routes to a host, or a gateway were deleted.

- **Types of primitives and parameters**

  **Table 9. Primitive Types and parameters of the WAF_Node_Dissappearance service**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_Node</td>
<td>M</td>
</tr>
</tbody>
</table>

- **Definition of Parameters**
  
  - WAF_Node: A Node description (Neighbour or RemoteNode) of the lost node.
3.2.2.9 **WAF_Connection_AddTrafficPolicy**

- **Function**
  
  Provides to the higher layer entities the capability to add a new TrafficPolicy over a Connection with a Node (either over a MultipathConnection with a RemoteNode, or over a LLCConnection with a Neighbour Node).

- **Types of primitives and parameters**

  **Table 10. Primitive Types and parameters of the WAF_Connection_AddTrafficPolicy service**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_Connection</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td>WAF_TrafficClasses</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WAF_BoosterParameters</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>WAF_Result</td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>

- **Definition of Parameters**
  
  - **WAF_Connection**: The identification of the Connection over which the new TrafficPolicy is to be added.

  - **WAF_TrafficClasses**: The identification of the traffic classes for the TrafficPolicy to be created.

  - **WAF_BoosterParameters**: The parameter sets of the boosters assigned to the WAF_TrafficClasses.

  - **WAF_Result**: The result of the request primitive, encoded as WAF_OK or WAF_ERROR_<#NUM>.

3.2.2.10 **WAF_Connection_RemoveTrafficPolicy**

- **Function**
  
  Provides to the higher layer entities the capability to remove a TrafficPolicy over a Connection with a Node (either over a MultipathConnection with a RemoteNode, or over a LLCConnection with a Neighbour Node).

- **Types of primitives and parameters**

  **Table 11. Primitive Types and parameters of the WAF_Connection_RemoveTrafficPolicy service**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_Connection</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td>WAF_TrafficPolicy</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>WAF_Result</td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>
- Definition of Parameters
  - WAF_Connection: The identification of the connection over which the TrafficPolicy is to be removed.
  - WAF_Result: The result of the request primitive, encoded as WAF_OK or WAF_ERROR_<#NUM>.

3.2.2.11 WAF_LLConnection_ChannelRateChanged

- Function
  Provides to the higher layer entities an indication of an automatic bit-rate change.

- Types of primitives and parameters

Table 12. Primitive Types and parameters of the WAF_LLConnection_ChannelRateChanged service

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_LLConnection</td>
<td>M</td>
</tr>
<tr>
<td>WAF_ChannelRate</td>
<td>M</td>
</tr>
<tr>
<td>WAF_Reason</td>
<td>C</td>
</tr>
</tbody>
</table>

- Definition of Parameters
  - WAF_LLConnection: The identification of the neighbour connection in which the channel rate change took place.
  - WAF_ChannelRate: The current (new) bit-rate for the connection.
  - WAF_Reason: A possible reason of the automatic channel rate change.

3.2.2.12 WAF_LLConnection_ChannelQualityChanged

- Function
  Provides to the higher layer entities an indication that channel quality crossed a predefined high or low quality threshold.

- Types of primitives and parameters

Table 13. Primitive Types and parameters of the WAF_LLConnection_ChannelQualityChanged service

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_Connection</td>
<td>M</td>
</tr>
</tbody>
</table>
• Definition of Parameters
  o WAF_LLCCConnection: The identification of the neighbour connection in which the channel quality change took place.
  o WAF_ChannelQuality: The current (new) channel quality for the connection.
  o WAF_Reason: The reason of activation of the indication primitive, encoded as WAF_CQ_LOW, WAF_CQ_IN-LIMITS, WAF_CQ_HIGH.

3.2.2.13 WAF_AddTrafficClass

• Function
  Provides to the higher layer entities the capability to define a new Traffic Class.

• Types of primitives and parameters

  Table 14. Primitive Types and parameters of the WAF_AddTrafficClass service

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_TrafficClass</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td>WAF_Boosters</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>WAF_BoosterParameters</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>WAF_Result</td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>

• Definition of Parameters
  o WAF_TrafficClass: The identification and characterization of the new traffic class as this is depicted in Figure 3 (i.e. its ID and filtering criteria).
  o WAF_Boosters: The ordered list of boosters which should apply for the new traffic class.
  o WAF_BoosterParameters: The parameter sets of the boosters defined in WAF_Boosters which may be applied in a traffic class-wide manner.
  o WAF_Result: The result of the request primitive, encoded as WAF_OK or WAF_ERROR_<#NUM>.

3.2.2.14 WAF_RemoveTrafficClass

• Function
  Provides to the higher layer entities the capability to remove a traffic class from the currently available set of such classes existing inside WAF.
• Types of primitives and parameters

Table 15. Primitive Types and parameters of the WAF RemoveTrafficClass service

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAF_TrafficClass</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td>WAF_Result</td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>

• Definition of Parameters
  o WAF_TrafficClass: The identification of the traffic class which is to be removed.
  o WAF_Result: The result of the request primitive, encoded as WAF_OK or WAF_ERROR_<#NUM>.

3.2.2.15 WAF_Attribute_Get

• Function
The service by which a higher layer entity may retrieve the current value of an attribute (parameter or statistics).

• Types of primitives and parameters

Table 16. Primitive Types and parameters of the WAF_Attribute_Get service

<table>
<thead>
<tr>
<th>WAF_Attribute_Get</th>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WAF_Attribute_ID</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>WAF_Attribute_VAL</td>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>WAF_Result</td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>

• Definition of Parameters
  o WAF_Attribute_ID: The unique identification of a WAF attribute (parameter or statistic).
  o WAF_Attribute_VAL: The returned value of the previously requested attribute.
  o WAF_Result: The result of the request primitive, encoded as WAF_OK, WAF_UNDEFINED_ATTRIBUTE, WAF_ERROR_<#NUM>.

3.2.2.16 WAF_Attribute_Set

• Function
The service by which a higher layer entity may change the current value of a parameter attribute.

• Types of primitives and parameters
Table 17. Primitive Types and parameters of the WAF_Attribute_Set service

<table>
<thead>
<tr>
<th>WAF_Attribute_Set</th>
<th>Parameter Name</th>
<th>Request</th>
<th>Confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WAF_Attribute_ID</td>
<td>M</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>WAF_Attribute_VAL</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WAF_Result</td>
<td></td>
<td>M</td>
</tr>
</tbody>
</table>

- **Definition of Parameters**
  - **WAF_Attribute_ID**: The unique identification of a WAF attribute (parameter or statistic).
  - **WAF_Attribute_VAL**: The new value for the requested attribute.
  - **WAF_Result**: The result of the request primitive, encoded as WAF_OK, WAF_READ_ONLY, WAF_UNDEFINED_ATTRIBUTE, WAF_BAD_VALUE, WAF_ERROR_&lt;#NUM&gt;.

### 3.3 Bluetooth LLCT

Figure 4 shows the Bluetooth LLCT:
- A super-class (BT LLCT) which contains common methods,
- And two specialized classes that are inherited for the server (Bluetooth node providing NAP or GN service) and client sides (Bluetooth node providing PANU service), and a class who manages Bluetooth connections (BT CONNECTION SET).

### 3.3.1 Interfaces definition

#### 3.3.1.1 BT_LLCT

The class BT_LLCT contains all the common methods of Bluetooth LLCT:

**Public:**
- `setAttribute / getAttribute`: those methods allow to get and to update attributes value of BT LLCT.

Those attributes are:
Figure 4: Bluetooth LLCT class diagram

- **MTU**: the MTU of a Bluetooth packet
- **bdaddr**: the physical address of the Bluetooth card (ex: 00:05:4E:01:97:AB)
- **deviceName**: the network interface name of the BNEP device (ex: “bnep%d”)
- **IPAdress**: the IP address of the network interface
- **waitTimeOut**: duration between two link quality and data rate test
- **BTName**: the network name of the Bluetooth peripheral
- **linkQuality**: the last measure of the link quality (get only)
- **linkRate**: the last measure of data rate on the link (get only)
- **linkQualityThresholds**: the last link quality thresholds
- **BTlinkSupervisionTimeout**: to detect the loss of a Bluetooth connection
- **statistics**: Bluetooth driver statistics (get only)
- `setEventHandler/removeEventHandler`: this method allows users to give and remove an event handler for an event, an event could be:
  - new connection
  - link disconnected
  - quality link changed

**Private:**
- `addBNEPConnection`: sends a request to BNEP driver to create a new network interface
- `checkForSDPService`: checks if a service is available on a Bluetooth device

### 3.3.1.2 BT_SERVER_LLCT

The class `BT_SERVER_LLCT` contains all the methods for the server side of Bluetooth LLCT, to listen to incoming connections and manage the creation of a new BNEP interface for each new connection:

**Public:**

- `serviceStart`: brings up the Bluetooth interface, configures it, registers the NAP or GN service near the SDP daemon, and creates a socket to listen to incoming connections.

**Private:**

- `waitForConnection`: waits for incoming connections and checks for loss of active connections.
- `BNEPHandShake`: proceeds to BNEP hand checking and configures the associated network interface.
- `addSDPService`: registers a new service near SDP server.

### 3.3.1.3 BT_CLIENT_LLCT

The class `BT_CLIENT_LLCT` contains the methods for the client side of Bluetooth LLCT, searches for servers, asks for service, and manages the creation of the BNEP interface on the client side:

**Public:**

- `serviceStart`: brings up the Bluetooth interface, and configures it.
- `discoverNeighbor`: makes an HCI inquiry to search other Bluetooth devices and checks for services available on these devices.
- `connectNeighbor`: establishes a BNEP connection with a remote Bluetooth device.

**Private:**
- BNEPHandShake: Proceeds to BNEP hand checking and configures the associated network interface.
- waitForEvent: checks for loss of connection with the server

### 3.3.1.4 BT CONNECTION SET

The class BT CONNECTION SET manages all active connections created by Bluetooth LLCT.

**Public:**
- addNewConnection: add a new Bluetooth connection in the connection set.
- removeConnection: remove a connection from the connection set.
- getActiveConnectionsSockets: return an array with active sockets.

### 3.3.2 Sequence diagram

The diagram depicted in Figure 5 illustrates the method call sequence within the WAF.

![Bluetooth sequence diagram](image)

**Figure 5: Bluetooth sequence diagram**
Server side

On server the WAF configures Bluetooth attributes and events, starts the NAP or GN service which waits for incoming connection from a client and for events. When it receive an incoming connection, it manages the BNEP handshake and registers a new BNEP network interface.

Client side

On client the WAF configures Bluetooth attributes and events, starts the PANU service. Ask for neighbour with a specific service, LLCT returns a list of available neighbours, WAF chooses to connect a neighbour so LLCT ask for a connection, manage a BNEP handshake, register a new BNEP network interface, and waits for events.

3.4 WiFi LLCT

The WiFi LLCT provides a uniform set of manipulation functions to hide the IEEE 802.11x wireless driver singularities to the WAF. At the same time it exports link–level statistics to optimise the overall WAF operation.

3.4.1 Event detecting functions

These functions monitor some parameters so as to detect occurring events, acting accordingly. A typical example of such situations is the change of link quality, upon which the boosting capacity could be modified correspondingly.

3.4.2 Link–level statistics retrieval

WiFi LLCT gathers information concerning operating parameters of the wireless interface, so as the WAF can use it for future use. In this sense, WiFi LLCT acts as a data supplier for the different entities that form the WAF.

Foreseen statistics to be collected are:

- Link quality, in terms of the measured SNR (get only).
- Both packet and bit error rates, if available.

3.4.3 Managing link-level parameters

Wireless interfaces following IEEE 802.11b specification allow the users to modify a large number of their attributes, so as to adapt their operation. It depends on the particular hardware devices and their corresponding drivers whether these parameters are readable and/or writeable. Some of the most relevant ones are the following:

- Operation mode, infrastructure or ad hoc.
- Frequency, channel.
- Bit rate.
- Transmission power.
- RTS/CTS procedure.
- Fragmentation.
- Power mode.

There are also other parameters, that are more dependent on the relation between the proper device and the Operating System. Some of them are also relevant and, therefore, are briefly described below.

- The MTU of the IEEE 802.11b packet (get/set).
- MAC address (get only).
- Network interface identifier of the WiFi device (get only).

3.5 INTERFACE WITH ROUTING INFORMATION / PROTOCOLS

The routing machinery triggers the Wireless API on certain changes in the routing tables. In turn, the Wireless API forwards these events to the WAF daemons (for instance, to the WAM). The following events need to be signalled:

- Detection of a new neighbour: a new route added to the tables contains a next hop, which doesn't appear in any existing route.

- Detection of a new final destination where the end-to-end path may be boostable: a new route added to the tables has a new final destination, which is located more than one hop away and belongs to the same autonomous routing system (e.g. DSR can check this last condition on the "Last Hop External" flag). This event should be triggered only if traffic has started between the local node and this remote destination, because a node may store a route on which it acts only as forwarder.

- Removal of a neighbour: the routing machinery is notified that the link with a neighbour is no longer valid and removes all routes having it as next hop.

- Removal of a final destination: a route is removed from the tables and there is no other route to this final multi-hop destination, which belongs to the same autonomous routing system.

Besides its role of triggering events to the WAM, IF 2 could also be used to improve existing routing mechanisms. Current MANET specifications deal with networks up to a hundred of nodes, characterized by rapid topological changes. Within this scenario, the only metric that is used to route packets from a source to a destination is the number of hops. The use of more than one metric could derive in a non-convergent problem and, therefore, up to now it has been out of the scope of MANET researching issues. However, the scenarios that are targeted within the 6HOP project are less challenging in the sense that nodes can be considered as “quasi-static” and, moreover, network sizes are relatively small. Taking into account these characteristics, a multi-parametrical approach could be followed adding link-level statistics to the route election function. The WAF could be seen as an
appropriate framework to leverage that approach, as it allows information sharing between multiple machineries. For instance, if DSR is used, the information concerning network topology can be stored on a graph fashion, in which all the nodes have a unified vision of the whole network, comprising some of the metrics that are provided by the WAF; although being more complex, as a graph search algorithm (such as Dijsktra) has to be used whenever a new route is needed, it is indeed a more powerful approach, because the nodes are able to use all available information of the network to find routes towards the destination. The more the metrics you use, the more difficult the cost function becomes; this problem could be quite difficult to solve for large networks, but if the number of hops is kept at a reasonable level, as it happens with the 6HOP case, it could be reasonable to include such a strategy.
4 WAF ADAPTATION MANAGEMENT

As already described in 2.2.2, the coordination of all WAF enhancements and lower level mechanisms is achieved at the application layer, through a number of application entities classified into managers, protocols and agents. In the context of the 6HOP project, two managers are presented, which leverage the adaptation mechanisms offered at the Wireless API (IF1), as depicted in Figure 6.

- **Wireless Adaptation Manager (WAM):** WAM mainly perform optimisations for multihop. In particular, WAM manipulates wireless drivers and protocol boosters through the Wireless API. Principal operations of WAM are: (1) configuration, (2) statistics retrieval, (3) event handling, and (4) discovery whether or not a neighbour node (either a single- or multiple-hops away) is WAM-capable.

- **Connection Manager (CM):** CM controls connections over a specific wireless device or handoffs from one wireless interface to another according to link information received through the Wireless API.
4.1 **Wireless Adaptation Manager (WAM)**

4.1.1 Network Functionality

The WAM (Wireless Adaptation Manager) is responsible for the management of the boosting modules which are used to conceal channel impairments to the upper layers. In this sense, depending on a number of different parameters, such as channel characteristics, type of traffic, boosting capacity of the neighbour/destination node, battery level, etc., it will activate and configure the corresponding modules.

One major difference with the approach followed in [7] is that in 6HOP a completely distributed architecture has been adopted, instead of a hard signalling mechanism, so that it is appropriate for a multi-hop environment. Boosting capabilities do not need to be negotiated each time a new flow of traffic is to be delivered. On the contrary, a single node, in an standalone and distributed fashion, decides which modules to apply, and the data PDU will contain enough information itself so that the receiver can process the PDU accordingly, as described in section 2.3.1.

In order for this to be accomplished all nodes belonging to an ad-hoc network need to know which are their neighbours (i.e. the ones that are physically connected with them through a wireless link) and their boosting capacities, so a signalling procedure must be defined between WAM entities. In addition, a node needs to be aware of some internal information, regarding link conditions, battery level and so on, to modify its operation. This implies a traversal of different pieces of information amongst components of the WAF architecture, supported by processes of internal signalling which, even more, will be used by the WAM to interact with the rest of components (such as boosting modules) and control their operation.

The WAM should be able to access this information whenever it needs it, using a set of predefined functions, exported by the Wireless API. Moreover, it is foreseen that boosting capacity will be adaptive to changing conditions, so the WAM needs to react upon the reception of alarms, which will be triggered by other modules. As an example, if a UDP flow starts over a bad quality link, it is probable that the FEC will be used to boost it; but if afterwards, due to the movement of some of the nodes, the Signal to Noise Ratio (SNR) increases, the FEC is not needed anymore. The WAM has to be notified of this new situation, and will change the configuration of the FEC accordingly. It is also desirable that the user itself is able to both access and modify boosting configuration.

4.1.2 Internal Signalling

As described before, the WAM needs to access to a number of pieces of information in order to fulfil with the identified requirements. These will be obtained and then exported by some of the modules that constitute the WAF. In addition, it will need to exchange some information with the boosting machinery (in concrete with the tables inside the protocol booster “box”), so that to manage its operation. Therefore, a complete set of internal signalling messages need to be defined:

- LLCT: the LLCT will provide the SNR associated to the link with a specific neighbour whenever the WAM asks for it. On the other hand, it will trigger some alarm each time a new link condition is detected. In this sense, the WAM will be able to modify the SNR thresholds and the policy used to deal with them.
Routing machinery: whenever a node has reached the coverage area of another one, routing mechanisms are able to detect it. Afterwards, the WAM will start the process of discovering its boosting capacity, so routing modules will notify this situation to the WAM. In addition, the WAM will need to be aware of the next hop to reach a particular destination, so that anytime a new flow is started, it is able to activate boosting modules, taking into account the capabilities of the destination node for this particular hop.

Battery level sensor: in order to provide a power aware operation, the WAM should be able to access information regarding battery remaining level. An specific module will be devoted to obtain this information and some signalling between this and the WAM will support information exchange.

4.1.3 External Signalling

4.1.3.1 WAF capabilities discovery process

It has been already said that a WAM will need to be aware of the boosting capacities for all its neighbours. This will be possible through the exchange of some messages between WAM entities.

Two different possibilities can be distinguished: (1) upon start-up, a node will perform some kind of discovery procedure, by which it will learn the modules that its neighbours are able to use, so that it is able to activate some of them in case the type of traffic requires so. This is shown in Figure 7, where a new node, A, starts up and sends a message to announce all its neighbours the modules it is able to use. When another node, B, listens to this message, it answers, specifying its boosting capacity. As can be seen, this is a proactive mechanism (as node A does not know whether it will need the information or not) which derives on a multicast transmission.

(2) if some node (already active) moves around going inside the coverage area of another one, some of the routes belonging to the latter will change, and this will be detected by the routing machinery once it may want to start a new communication. Afterwards, the routing modules will send and event to the WAM, advertising this new situation. The WAM would like to know which boosting capabilities are available on it and for that purpose will send a message to its new neighbour. The messages exchanged are the same as in Figure 7, but in this case all transmissions are unicast (i.e. addressed to a specific node).

In order to ease the implementation, and to use a lightweight approach, all signalling traffic is transmitted over UDP, using the traditional socket interface. In this sense, the WAM of all nodes
belonging to a WAF scenario should be in charge of listening to incoming UDP datagrams through the **port 2006** which is the chosen one for this kind of signalling packets.

### 4.1.3.2 WAF services primitives

The first aspect is to define a common WAF header for all signalling PDUs. This header, depicted on Figure 8, contains the WAF version and the PDU type, which identifies the signalling primitive.

![Figure 8 WAF header structure](image)

When a node wants to discover the WAF capabilities of one or several neighbour(s), it sends a **WAF_NEIGH_CAPABILITY_REQUEST** primitive to the neighbour(s), indicating its own capabilities. If the neighbour is WAF capable, it answers with a **WAF_NEIGH_CAPABILITY_RESPONSE** primitive, containing its capabilities. Both primitives have a similar structure, which is shown on Figure 9. The Sender MAC address field identifies the originator of the primitive and the Boosters list field indicates the list of protocol boosters supported by the originator node.

![Figure 9 Structure for neighbour capability discovery primitives](image)

As the signalling traffic transits over UDP, some retransmission mechanisms should be implemented. For instance, in a unicast discovery, the initiator sends the **WAF_NEIGH_CAPABILITY_REQUEST** to the neighbour, waits for the response during a certain time delay and, if the timeout expires, it sends the request message again.

When a node wants to discover the WAF capabilities of a remote node located some hops away, it sends a **WAF_DEST_CAPABILITY_REQUEST** primitive to this node, indicating its own capabilities. If the remote node is WAF capable, it answers with a **WAF_DEST_CAPABILITY_RESPONSE** primitive, containing its capabilities. The structure of these primitives is depicted on Figure 10. In this case, the remote node is identified by its IP address, which can be obtained from the BSD socket API.

![Figure 10 Structure for remote node capability discovery primitives](image)

### 4.1.4 Adaptation Mechanisms

The WAM daemon manages the protocol boosters policy; it communicates with the protocol boosters framework (PBF, see § 5.1) to configure and activate the boosting policy on a link or on a multi-hop path. On the other hand, PBF intercepts all the outgoing packets sent to the wireless network interface and searches the remote MAC or IP address in the configuration tables set by WAM in order to get
the booster(s) to be called for this packet. PBF also intercepts all the incoming packets and, based on the boosters related information in the WAF header (see § 5.1.5), it applies the required boosters. In this later case, the boosters applied and the associated parameters rely on the information previously inserted by other boosters on a remote node.

For instance, the FEC booster needs to be applied on both sides of a link. When FEC is called for an outgoing packet, it calculates a code value based on the code length parameter and on the packet payload; then it adds its own header in the WAF header with its booster identifier, the code length and the code value. On the opposite side of the link, PBF finds the FEC identifier in the WAF header so it calls this booster, which reads its own header from the WAF header in order to deboost the packet.

### 4.1.4.1 Boosters policy

The WAM daemon has a pre-configured list of traffic classes, each associating a particular kind of traffic with a chain of protocol boosters to apply downstream. For instance, the traffic could be characterized by the DiffServ field and the protocol type. Two categories of traffic classes are defined: the single-hop classes are intended for single-hop boosting and the multi-hop classes for end-to-end boosting. Some single-hop traffic classes could be applicable to all network types while others could be suitable only for a particular network (e.g. a class could contain a booster which is beneficial only on a Bluetooth link).

The WAM determines the **single-hop boosting policy** to apply to a particular link according to the wireless network type, the neighbour WAF capabilities (in particular the list of supported boosters) and the current link quality: it selects the traffic classes which need to be applied and, for each booster being part of these classes, it fixes the parameters values. To avoid heavy signalling, this procedure doesn’t take into account the current transport streams that are transiting on the link but it rather considers all the possible types of transport stream. E.g., WAM may configure a class for FTP traffic and another for HTTP traffic, ignoring if such traffic will ever be sent over the link.

The WAM determines the **multi-hop boosting policy** to apply between the local node and a remote node some hops away according to this node’s WAF capabilities. The policy is simply a list of traffic classes, as no parameters are tuneable in this case.

### 4.1.4.2 WAM – PBF interactions

In single-hop boosting, when the WAM receives an event indicating that a new neighbour has been detected, once it has acquired the WAF capabilities of this neighbour, it determines the boosting policy on the link and tells the PBF to start boosting the link, passing in argument the neighbour MAC address, the selected list of traffic classes and the boosters parameters. Then, whenever the WAM is notified of changes in the link quality or in the link rate, it recalculates the list of classes and the boosters parameters and passes the new configuration to PBF. Finally, when an event indicates to the WAM that the communication with the neighbour has been lost, it tells the PBF to stop boosting this link, passing in argument the neighbour MAC address.

In multi-hop boosting, once WAM has acquired the remote node WAF capabilities, it selects the traffic classes and tells the PBF to start boosting the traffic with this node, passing in argument the
node IP address and the selected list of traffic classes. When WAM is notified that there is no more traffic with this remote node, it tells the PBF to stop boosting this traffic.

4.2 CONNECTION MANAGER (CM)

The main function of the Connection Manager (CM) which exists in mobile nodes as an independent, optional module is to allow seamless and fast handover between Bluetooth, GPRS and WLAN networks when multiple network interface are available, and to ensure that a wireless terminal be connected to the best available network. Several parameters can be used to determine the best available network including signal-to-noise ratio, cost, bandwidth, and services available in a network. The functionalities of the CM can be summarized as follows:

- To make sure the best possible network connection is currently in use.
- Performing vertical handoffs when a mobile node has multiple interfaces and there might be a need to perform vertical handoff.
- Changing network interface when the current interface is no longer available.

The CM collects related wireless network information through the wireless API which provides access to low level details of wireless interfaces and drivers such as network type, SNR, link quality, and services available in wireless networks. The wireless API notifies CM of the occurrence of certain types of events such as interface goes down. The CM maintains a list of currently available network interfaces and with constant monitoring of link statistics, the CM will have a predictive capability about when to conduct a handoff or change network interface.

By having information about currently available wireless interfaces, a mobile node can choose the network that fulfils its communication needs. If there are multiple interfaces and another interface is currently more suitable, a vertical handoff could be performed to another interface (for instance choosing WLAN instead of Bluetooth interface).

4.2.1 Handoffs in heterogeneous networks

The CM allows seamless and fast handover between Bluetooth, GPRS, and WLAN networks when multiple network interfaces are available, and ensures that a wireless terminal be connected to the best available network. Figure 11 shows an integrated Bluetooth and WLAN network in which a mobile node has multiple access network options. There are two roaming possibilities: from WLAN to Bluetooth or from Bluetooth to WLAN network. The main issue with the handover process is the decision about when to conduct a handover (disconnection detection) and to which network to connect to (connection establishment).

Handoff can be classified (depending on where the handoff decision is located) into: network controlled handoff, mobile controlled handoff, and mobile assisted handoff. Before a mobile node starts the handover process, some criteria or handover policy should be satisfied. The most common criteria that is used in handover initiation is the received signal strength RSS. As no standard threshold values have been defined for wireless interfaces, the threshold values are determined by analysing the wireless interfaces hardware. Two steps are involved in the handover process:
disconnection detection and connection establishment. The following sections describe handoff from WLAN to Bluetooth and from Bluetooth to WLAN network.

![Figure 11. WLAN-Bluetooth Interconnection Architecture](image)

4.2.2 Handoff from Bluetooth to WLAN network

4.2.2.1 Disconnection Detection

As the mobile node moves out of the coverage area of Bluetooth access network, the CM needs to detect the loss of connection in order to initiate the handover process. There are two approaches that can be used to detect the loss of connection:

1. Received signal strength (RSS): According to Bluetooth specifications, this parameter is an optional feature. However, it is believed that the RSS is at least supported by all chips based on the CSR chipset. By using RSS, it would possible to detect the loss of connection before a Bluetooth link goes down.

2. Bluetooth link supervision timer: By using this parameter, we can only detect the loss of connection after a Bluetooth link goes down, which will result in delay and packet losses during the cut-off period.

4.2.2.2 Connection Establishment

As a mobile node leaves the Bluetooth coverage area, the signal strength received from the Bluetooth access point weakens. The mobile node starts hearing the beacons from the WLAN access point. When the signal strength from the WLAN AP is strong enough, the CM decides to switch from the Bluetooth to the WLAN network. The CM should establish connection with the WLAN before the handover takes place to avoid delay and packet losses.
4.2.3 Handoff from WLAN to Bluetooth networks

4.2.3.1 Disconnection Detection

The handover initiation from WLAN network to Bluetooth is mainly based on the received signal strength (RSS). As the mobile node leaves the WLAN cell, the signal strength received from the AP weakens. When the RSS falls below a predefined threshold, and persist in that situation for a certain period the CM triggers the handoff process to Bluetooth network.

4.2.3.2 Connection Establishment

As a mobile node is moving out of the coverage area of a WLAN network, The CM is responsible for establishing connection with a new Bluetooth access point. The main issue with the Bluetooth is the time required to establish connection through the inquiry and paging procedures. Since the mobile node performing the handover has no previous knowledge about the available Bluetooth access point’s device addresses and clocks, it has to start the time-consuming inquiry procedure to retrieve device address and clock setting of the access point in its vicinity. Bluetooth connection establishment involves inquiry and paging procedures:

- Inquiry Procedure: The inquiry procedure enables a unit to discover which Bluetooth units are in range, and what their device addresses and clocks are. It involves a Bluetooth unit sending out inquiry packets and then receiving the inquiry reply. After the inquiry procedure has completed, a connection can be established using the paging procedure.

- Paging Process: With the paging procedure, an actual connection can be established. A unit that establishes a connection will carry out a page procedure and will automatically become the master of the connection. The paging procedure typically follows the inquiry procedure. Only the Bluetooth device address is required to set up a connection. Knowledge about the clock will accelerate the setup procedure.

Another open issue with connection establishment is the service discovery. The service discovery protocol (SDP) provides a means for applications to discover which services are available (e.g. printing, teleconferencing, and so on) and to determine the characteristics of these available services. When performing handoff, a mobile node may need to connect to the right access point that provides the required services.

4.2.4 Handover Optimisation

Efficient utilization of the scarce radio resources is certainly one of the major challenges in wireless networks. Since the Bluetooth, GPRS, and WLAN support different data rates (up to 720 Kb/s with Bluetooth, 54 Mb/s with 802.11a and 11 Mb/s with 802.11b). When the handoff is from a Bluetooth to a WLAN network (i.e. from low bandwidth to high bandwidth), WLAN has sufficient resource to support communications. However, when the handoff is from WLAN to a Bluetooth, the low bandwidth of the Bluetooth could cause congestions. To optimally maximize radio resources, it would be beneficial for the mobile node to use WLAN networks as long as it can.

If the handover initiation is only based on the RSS, there will be a ping-pong effect. To reduce the ping-pong effect when the mobile node moves at the border of the wireless network, it would be
beneficial to adopt the dwell-timer scheme described in [33], in which the mobile node will take samples of the RSS and compare it with a predefined threshold. If consecutive samples during predefined dwell time are below the threshold, the mobile node initiates the handoff procedure.

A fundamental goal is to minimize the handoff delay in the transition region. When the mobile node is roaming from WLAN or GPRS network to Bluetooth, the Bluetooth unit should acquire the device address and clock setting of the next Bluetooth access point in advance. This means that the inquiry procedure should be completed while the mobile node is still connected to the WLAN or GPRS network. Therefore, the fast handoff to the Bluetooth access point will be completed as soon as the mobile node leaves the WLAN coverage area. When the handoff is from Bluetooth to WLAN or GPRS network, the disconnection detection should be determined before Bluetooth link goes down. This can be achieved by setting the link supervision timeout to the minimum allowed value.
5 WAF PACKET PROCESSING

The protocol booster [8] is an adaptation technique that dynamically improves (boosts) the performance of the Internet protocols in a specific network environment. A protocol booster applies selectively to a portion of the network path, without altering the robustness of the end-to-end communication. A protocol booster can reside on a particular node in the path and operate independently or it can be composed of several booster elements installed on different nodes and operating in cooperation.

The 6HOP project intends to use protocol boosters in order to compensate for wireless link impairments. As an example, two boosters are considered: Forward Error Correction (FEC) and Robust Header Compression (ROHC) for TCP.

As previously explained in 4.1.4, boosters (e.g. FEC) may add some piece of information between the IP header and the Ethernet header. So, these boosters need to reside between the IP layer and the wireless driver. Furthermore, a framework is needed in order to invoke the protocol boosters in a centralized way and to provide interfaces with the rest of the system.

The WAF packet processing part is composed of the protocol boosters framework (PBF) and the list of boosters, all being presented hereafter.

5.1 PROTOCOL BOOSTERS’ FRAMEWORK

5.1.1 Architecture

The Figure 12 shows the internal architecture of the protocol boosters framework, organized in three design modules:

- The Boosters Support module acts as the controller / manager of the overall packet processing functionality and its parameters, providing an interface to the higher level parts (i.e. the Wireless API) through which configuration of the boosters policy and the boosters themselves is made possible.

- The Boosters Policy module maintains tables of the traffic classes and of the current links boosting policies; it also provides a function to classify the outgoing packets according to these policies.

- The Packets Filter module provides the interface with the network layers stack; it processes all the incoming and outgoing packets, calling the required protocol boosters on it if needed.

Interacting with PBF, the Protocol Boosters modules provide the actual packet processing functions for the upstream / downstream data traffic, which implement the boosting algorithms described in sections 5.2 and 5.3.
5.1.2 Interface with the Wireless API

The Boosters Support module provides the following services to the Wireless API:

- Addition or removal of a traffic class; each class is characterized by:
  - the class identifier,
  - the filtering criteria (transport protocol, DiffServ field, source port, destination port),
  - the ordered list of boosters to apply on outgoing packets matching the criteria.

- Addition, modification or removal of the boosting policy concerning a particular link; each link boosting policy contains the following information:
  - the neighbour identifier (e.g. its MAC address),
  - the list of traffic classes (only the classes identifiers) to apply,
  - the boosters parameters for this link.

- Addition or removal of a multi-hop boosting policy with these arguments:
  - the remote node IP address,
  - the list of traffic classes (only the classes identifiers) to apply.

- Configuration of the boosters: this concerns global parameters, independent of the link characteristics (e.g., for the Snoop protocol booster, values of the timers between retransmissions).

- Statistics retrieval from PBF concerning PBF itself or any booster.
5.1.3 Interface with the Protocol Boosters

When a Protocol Booster module is loaded, it registers itself near the PBF, indicating:

- its identifier,
- the function which handles outgoing packets,
- the function which handles incoming packets,
- the function for configuration.

Inversely, when this Protocol Booster module is unloaded, it deregisters itself from the PBF.

5.1.4 Interface with the Network Layer

To get a chance to boost some kinds of outgoing IP traffic, the PBF needs to intercept packets when they leave the IP layer, i.e. just before their entrance at the link layer. To be more precise, the PBF needs to catch all outgoing packets sent to each wireless network interface on which the boosting mechanisms are supported.

On the other hand, protocol boosters will be applied on an incoming packet only if it contains a WAF header (see § 5.1.5), i.e. if the type field of the Ethernet header equals to the WAF type. The PBF needs to intercept such packets when they leave the link layer, i.e. just before their entrance at the IP layer.

The PBF implements two functions, namely the output and input functions, to respectively handle the outgoing and incoming packets. Depending on the targeted operating system, the PBF will use an appropriate kernel method to register these functions near the network layers stack.

When a protocol booster is applied to a given packet, it may alter the packet; e.g. ROHC for TCP will compress the TCP/IP header and add its own header at the beginning of the resulting frame. So, the following network code should not attempt to access the IP header. Moreover, the booster module may decide to drop the packet or to queue it in its internal buffers for further processing. So, there must be a way to notify the network layers stack of this decision. Furthermore, the booster module may generate new packets that need to be sent over the network.

The Figure 13 shows the packets traversals across the PBF, which are described in detail below.

Output traffic path:

1. The IP layer passes the packet to the PBF output function.
2. The PBF output function calls the classifying function of the Boosters Policy module to get the list of boosters to apply to the current packet and the associated parameters. The multi-hop boosting policies are considered only if the node is the originator of the packet.
3. The PBF output function calls the output function of the first booster in the list, with this booster current parameters. If the booster returns a code indicating that the packet can continue its traversal, the PBF output function calls the next booster in the list.
4. For intermediate nodes, the PBF will check whether the outgoing packet has been boosted in an end-to-end fashion. If so, the specific (end-to-end) booster header will have been saved and the header will be added to the outgoing packet.
5. At the end of the boosters list, the packet is passed to the wireless driver transmit function.

![Diagram](image)

**Figure 13** Packets traversals across PBF

**Input traffic path:**

1. The packet is passed from the wireless driver to the PBF input function.
2. If the packet contains a WAF header, the list of boosters to apply and the associated parameters are extracted from it; if the node is not the recipient of the packet, the end-to-end boosters are ignored (in the header, a flag per booster indicates its scope, as described in § 5.1.5). The PBF input function calls the input function of the first booster in the list, with this booster current parameters. If the booster returns a code indicating that the packet can continue its traversal, the PBF input function calls the next booster in the list.
3. If an intermediate node intercepts an end-to-end boosted packet, it will extract and cache the end-to-end booster specific header(s) and mark the packet in order to re-introduce it to the same packet when it came down as an outgoing traffic once the routing protocols forwards it towards the final destination. This way of proceeding makes end-to-end boosters not to act through the IP header.
4. At the end of the boosters list, the packet is passed to the IP layer.

**Signalling traffic path:**

As already mentioned, the WAF signalling packets should not be processed by the protocol boosters. Either the stack makes it possible to filter these packets in order to pass them directly to the wireless driver or a specific mechanism is implemented inside the PBF to ignore all packets having the corresponding UDP ports (e.g. 2006 for boosting capacities discovery).
5.1.5 WAF Data Frame Format

As described in section 2.3.1 a new PDU format has to be defined for WAF. The contents of such a frame are depicted in Figure 14.

![Figure 14 WAF Data PDU Format](image)

Each boosting module inside the Packet Processing part adds its header, with its own predefined format, which has to include, at least, information concerning:

- The module itself, by a defined and unique identification.
- The length of the overhead introduced.
- Information concerning its operation if needed.

In order to differentiate between hop-by-hop boosters and end-to-end ones, a bit-flag within the Module ID field of each module specific header will be included. This way, the PBF will know how this booster must be treated depending on which kind of node (destination or intermediate) it is operating (i.e. for end-to-end boosters only the destination node can access to its information while hop-by-hop boosters, are accessed in every node).

As it is well known, wireless links are prone to cause errors on frames that are transmitted over them and an error located at the signalling part of a data PDU has a strong impact over its normal operation. In this sense, it is necessary to protect modules’ specific information with a CRC check. A length field which indicates the size of the variable part of the WAF header (the one that comprises all the boosting module specific headers) is also added in order to have a pointer to the CRC checksum field.

5.2 FORWARD ERROR CORRECTION

5.2.1 Overview

Forward Error Correction (FEC) is a widely studied reliable communication technique which has been used to implement error control at different levels, from the physical layers to the upper layers. FEC improves data reliability by introducing some amount of redundant information in the transmitted data, which allows the reconstruction of missing or erroneous data at the receiver without further interactions (opposed to ARQ). There are two possible ways of implementing FEC:

- Bit or byte-level FEC: the error correcting code is computed at a per-packet basis and a field containing this code is added to the frame. This aims to correct erroneous bits in the packets.
- Packet-level FEC: a redundant packet is added over a block of packets in order to recover of lost packets at the receiver. This technique is also known as FZC (Forward eraZure Correction).
Typical FEC schemes are stationary, i.e. they apply a fixed coding scheme independently of the current channel state and the current traffic requirements. In a wireless environment, an effective error correcting technique should be capable of adapting to the channel conditions (e.g. bit error rate, burstiness). When these conditions are good, less error protection is needed, so that the redundant overhead is minimized. And when these conditions are poor, more protection must be added, reducing the effective channel bandwidth.

Furthermore, a FEC booster should adapt its behaviour to the type of data streams. For example, audio or videoconferencing tools require low delay, even if some errors occur while file transfers need a low residual error rate, even at the expense of increased delay.

### 5.2.2 FEC protocol booster

The 6HOP project will implement at the link layer a byte-level FEC module, which applies over the IP datagrams. Under control of the WAM daemon, PBF will invoke FEC only on certain types of traffic. Furthermore, the FEC code length will be tuneable per link, allowing to adapt the corrective strength according to the current link quality.

The 6HOP project will implement, at the link layer, a FEC module, which applies to IP datagrams. Under control of the WAM daemon, PBF will invoke FEC only on certain types of traffic. Furthermore, the FEC code length will be tuneable based on the link state, allowing to adapt the corrective strength according to the current link quality.

As stated above, when FEC module is loaded, it registers itself near the PBF indicating its interfaces both for data handling and configuration, allowing the PBF to call it whenever a suitable datagram arrives, after performing the necessary filtering and policy checks.

The FEC module to be implemented within the 6HOP framework will add the corresponding redundancy to each data packet, following the WAF Data PDU format shown in Figure 14. Figure 15 depicts FEC specific header.

**Figure 15. FEC specific header**

- **FEC ID**: Protocol Booster unique identifier.
- **FEC Length**: The length in bytes of the whole FEC specific header. This field will also give information about the correcting capacity of the FEC code used to protect a particular packet.
- **FEC redundancy**: The error correcting code computed at a per-packet basis.

To allow FEC module to adapt its behaviour to both the type of data and channel state, the Boosters Support Module receives configuration requests either from the Boosters Policy Module and the Wireless API. These configuration parameters tune the FEC operation so as it applies a different correcting capacity to each of the different packets.

On transmission, FEC module will receive from the Boosters Support Module both the IP datagram and the boosting policy to be applied to this packet. FEC will compute the parity symbols, generate
the FEC specific header and afterwards send it back to the Boosters Support Module for its further insertion inside the whole WAF Data PDU.

On the other hand, on reception, after receiving the IP datagram and its specific header, the FEC module will perform the decoding operations, sending the IP datagram back if it was able to correct it, discarding it if not.

5.3 **ROBUST HEADER COMPRESSION FOR TCP**

5.3.1 **Overview**

The overhead induced by Internet protocols headers reduces the link efficiency, specially for applications using small packets sizes, e.g. real-time or interactive applications. The consequences are more sensible on lossy links with limited bandwidth, such as wireless networks. In this context, the compression of the protocols headers may improve the performance significantly.

Some studies have shown that the original header compression schemes (RFCs 1144, 2507, 2508) don’t work well enough on very lossy links. For these cases, RFC 3095 defines the Robust Header Compression scheme (ROHC) for RTP/UDP/IP, UDP/IP and ESP/IP (Encapsulated Security Payload) headers. It minimizes the probability of context invalidation on the decompressor and improves the repair mechanisms of the context. This scheme introduces the compression profile, a specification of how to compress the headers of a certain kind of packet stream over a certain kind of links. Its complex framework defines compressor and decompressor states coupled with modes of operation, which control the logic of state transitions and indicate what to perform in each state. It also describes new encoding methods, e.g. the window-based Least Significant Bits encoding.

A ROHC profile for TCP/IP [12] is under development. Interesting innovations are that it handles the TCP options such as timestamps and selective acknowledgements, and that it can compress the headers of handshaking packets (SYNs and FINs). The second point is intended to compress headers over multiple short-lived TCP connections, e.g. web browsing connections.

5.3.2 **ROHC-TCP protocol booster operation**

5.3.2.1 **Header Compression Scheme**

The goal of ROHC is to develop a header compression scheme that performs well over links with high error rates and long roundtrip times [11].

Header compression can be applied due to the fact that there is much redundancy between header field values within packets, especially between consecutive packets. The ROHC-TCP compression scheme provides improved compression efficiency and enhanced capabilities for compression of various header fields including TCP options.
Compression with ROHC can be characterized as an interaction between two states machines. One for compression and one for decompression.

5.3.2.2 ROHC States

For ROHC-TCP there are two compressor states. The Initialisation and Refresh state (IR) and the Compression state (CO) [12]. The compressor starts in the Initialisation state. When the compressor is confident that the decompressor has the required information to decompress a packet then it will switch to the Compression state. Normally the compressor operated in the Compression state.

Figure 18 presents the compressor state machine. In ROHC-TCP, the compressor starts in the IR state. Transition from the IR to CO state is based on a feedback (if a feedback channel is available) and the optimistic approach principle.

For ROHC-TCP there are three decompressor states. The No Context (NC), the Static Context (SC), and the Full Context (FC). The decompressor starts in the lowest compression state the NC state. When successfully decompressed a packet it will transit to the FC state. Once in the FC state the decompressor will stay in this state until repeated decompression failures force him to switch to the SC state.

Figure 19 shows the state machine for the decompressor.
While in the NC state, the decompressor has not yet successfully decompressed a packet. When receiving an IR, IR-DYN packet the decompressor will verify the correctness of the packet using the CRC check. Then it will update the context and use the decompressed packet as the reference packet. If a feedback channel is available and a packet is correctly decompressed the decompressor may send an ACK carrying the Master Sequence Number. If the verification of the packet fails then the decompressor should send an NACK, discard all packets until an IR packet has been successfully decompressed.

Once a packet has been successfully decompressed the decompressor can transit to the FC state. Only upon repeated failures the decompressor will switch back to the SC state. Only IR, IR-DYN packets may be decompressed in the NC state.

### 5.3.3 TCP/IP field behavior

At a general level, the header fields are classified into 5 classes:

- **INFERRED:** These fields contain values that can be inferred from other values and thus do not have to be handled at all by the compression scheme.
- **STATIC:** These fields are expected to be constant throughout the lifetime of the packet stream. Static information must be communicated once.
- **STATIC-DEF:** These fields are STATIC fields whose values define a packet stream. They are handled as STATIC.
- **STATIC-KNOWN:** These STATIC fields have well-known values and do not need to be communicated at all.
- **CHANGING:** These fields are expected to vary in some way.

#### 5.3.3.1 IPv6 header fields

Table 18 shows the classification of the IPv6 header fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Size (bits)</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
<td>STATIC</td>
</tr>
</tbody>
</table>
## 5.3.3.2 TCP header fields and options

Table 19 shows the classification of TCP header fields,

<table>
<thead>
<tr>
<th>Field</th>
<th>Size (bits)</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source port</td>
<td>16</td>
<td>STATIC-DEF</td>
</tr>
<tr>
<td>Destination port</td>
<td>16</td>
<td>STATIC-DEF</td>
</tr>
<tr>
<td>Sequence number</td>
<td>32</td>
<td>CHANGING</td>
</tr>
<tr>
<td>ACK number</td>
<td>32</td>
<td>CHANGING</td>
</tr>
<tr>
<td>Data offset</td>
<td>4</td>
<td>INFERRED</td>
</tr>
<tr>
<td>Reserved</td>
<td>4</td>
<td>CHANGING</td>
</tr>
<tr>
<td>CWR flag</td>
<td>1</td>
<td>CHANGING</td>
</tr>
<tr>
<td>ECE flag</td>
<td>1</td>
<td>CHANGING</td>
</tr>
<tr>
<td>URG flag</td>
<td>1</td>
<td>CHANGING</td>
</tr>
<tr>
<td>ACK flag</td>
<td>1</td>
<td>CHANGING</td>
</tr>
<tr>
<td>PSH flag</td>
<td>1</td>
<td>CHANGING</td>
</tr>
<tr>
<td>RST flag</td>
<td>1</td>
<td>CHANGING</td>
</tr>
<tr>
<td>SYN flag</td>
<td>1</td>
<td>CHANGING</td>
</tr>
<tr>
<td>FIN flag</td>
<td>1</td>
<td>CHANGING</td>
</tr>
<tr>
<td>Window</td>
<td>16</td>
<td>CHANGING</td>
</tr>
</tbody>
</table>
### Field Size (bits) Class

<table>
<thead>
<tr>
<th>Field</th>
<th>Size (bits)</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checksum</td>
<td>16</td>
<td>CHANGING</td>
</tr>
<tr>
<td>Urgent pointer</td>
<td>16</td>
<td>CHANGING</td>
</tr>
<tr>
<td>Options</td>
<td>0 (-352)</td>
<td>CHANGING</td>
</tr>
</tbody>
</table>

All options have been classified as CHANGING. Table 20 shows the most common used options and whether they appear only in a SYN packet or not.

<table>
<thead>
<tr>
<th>Option</th>
<th>SYN-only</th>
</tr>
</thead>
<tbody>
<tr>
<td>End of Option list</td>
<td>No</td>
</tr>
<tr>
<td>No-Operation</td>
<td>No</td>
</tr>
<tr>
<td>Maximum Segment Size</td>
<td>Yes</td>
</tr>
<tr>
<td>Window Scale</td>
<td>Yes</td>
</tr>
<tr>
<td>SACK-Permitted</td>
<td>Yes</td>
</tr>
<tr>
<td>SACK</td>
<td>No</td>
</tr>
<tr>
<td>Timestamp</td>
<td>No</td>
</tr>
</tbody>
</table>

#### 5.3.4 ROHC Packets and packet types

The general ROHC packet format is shown in Figure 20 as defined in [11]. A ROHC packet may include zero or more Padding octets. Figure 21 shows the Padding octet and Figure 22 presents the ADD-CID octet. Notice that the Padding octet is identical with the ADD-CID with CID=0.

![Figure 20: ROHC general packet format](image)

Padding is any number (zero or more) of padding octets. Either of Feedback or Header must be present. Feedback elements always start with a packet type indication. Feedback elements carry internal CID information. Header field is either a profile-specific header, or an IR, IR-DYN header or IR-REPLICATE packet. The Header field either

1. does not carry any CID information (indicating CID zero) or
2. includes one ADD-CID octet, or
3. contains embedded CID information of length one or two octets.

Alternative 1) and 2) apply only to compressed headers in channels where the CID space is small. Alternative 3) applies only to compressed headers in channels where the CID is large.

```
1 1 1 0 0 0 0 0
```

**Figure 21: Padding octet**

```
1 1 1 0 0
```

CID=0x01 – 0xF (1-15)

Add-CID=0xE1 – 0xEF

**Figure 22: ADD-CID octet**

**Large CIDs**

In order to optimise the transfer of the CID values (since the CID values changes from 1-16383) a variable number of octets is used to encode them. CID values are encoded based on the “Self-describing variable-length values” encoding method [[11]]. Up to two octets are used to transfer CIS information. The first few bits of the first octet determine the number of octets used:

First bit is 0: 1 octet (7 bits transferred)

(0x00 - 0x7F) ⇒ (127)

First bits are 10: 2 octets (14 bits transferred)

(0x8000 – 0xBFFF) ⇒ (16383)

**5.3.4.1 IR packet**

The IR header associates a CID with a profile, and typically also initialises the context. It can typically also refresh (parts of) the context. It has the following general format.
Add-CID octet

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>D</td>
</tr>
</tbody>
</table>

If for small CIDs and CID !=0

IR type octet (if D=1 dynamic chain present)

0-2 octets of CID

1-2 octets if for large CIDs

Profile

1 octet

CRC

1 octet (8-bit CRC)

Static chain

variable length

Dynamic chain

Present if D=1, variable length

Payload

variable length

Figure 23: IR packet format

Profile: The profile to be associated with the CID. In the IR packet, the profile identifier is abbreviated to the 8 LSBs.

CRC: 8-bit CRC computed using the polynomial of section 5.9.1 of RFC 3095.

5.3.4.2 IR-DYN packet

The IR-DYN packet type communicates the dynamic part of the context.

Add-CID octet

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

If for small CIDs an (CID !=0)

IR type octet

0-2 octets of CID

1-2 octets if for large CIDs

Profile

1 octet

CRC

1 octet

Dynamic chain

variable length

Payload

variable length

Figure 24: IR-DYN packet format

5.3.4.3 IR-REPLICATE Packet

Some of the information needed to initialise the context for compressing the headers of a new flow may already be present at the decompressor due to the redundancy between header fields of different flows passing through the same compressor-decompressor pair. It may be desirable to reuse this information
and remove some of the overhead normally required for the initialisation of a new header compression context. The IR-REPLICATE packet format is presented in Figure 25 as defined in [14].

![Figure 25: IR-REPLICATE packet format](image)

**5.3.4.4 CO Packet**

ROHC defines three packet types to indicating Compressed Header for the RTP profile. There is some ongoing discussion for the definition of the CO packet formats but no exact formats are designated yet. Furthermore, [15] has been published that proposes a set of packet formats for the TCP profile but has not been adopted yet from the WG.

**5.3.4.5 ROHC feedback**

Feedback carries information from decompressor to compressor. The following principal kinds of feedback are supported. In addition to the kind of feedback, other information may be included in profile-specific feedback information.

**ACK:** Acknowledges successful decompression of a packet, which means that the context is up-to-date with a high probability.

**NACK:** Indicates that the dynamic context of the decompressor is out of sync. Generated when several successive packets have failed to be decompressed correctly.

**STATIC-NACK:** Indicates that the static context of the decompressor is not valid or has not been established.
Feedback octet type

<table>
<thead>
<tr>
<th>Feedback octet type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF Code=0</td>
<td>1</td>
</tr>
<tr>
<td>Variable length</td>
<td></td>
</tr>
</tbody>
</table>

Figure 26: General feedback packet format

Feedback data

<table>
<thead>
<tr>
<th>Feedback data</th>
<th>0 1 2 3 4 5 6 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add-CID octet</td>
<td>0 1</td>
</tr>
<tr>
<td>Large CID</td>
<td>2 3 4</td>
</tr>
<tr>
<td>Feedback</td>
<td>5 6 7</td>
</tr>
<tr>
<td>Variable length</td>
<td></td>
</tr>
</tbody>
</table>

Figure 27: Feedback data format

FEEDBACK_1

<table>
<thead>
<tr>
<th>FEEDBACK_1</th>
<th>MSN</th>
</tr>
</thead>
</table>

FEEDBACK_2

<table>
<thead>
<tr>
<th>FEEDBACK_2</th>
<th>At least 2 octets</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSN</td>
<td></td>
</tr>
<tr>
<td>Feedback Options</td>
<td>Var length</td>
</tr>
</tbody>
</table>

ACK-type

<table>
<thead>
<tr>
<th>ACK-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ACK</td>
</tr>
<tr>
<td>1 NACK</td>
</tr>
<tr>
<td>2 Static ACK</td>
</tr>
<tr>
<td>3 Reserved</td>
</tr>
</tbody>
</table>

Figure 28: Feedback types

ROHC-TCP uses the same feedback options as the options defined in [RFC-3095, section 5.7.6], with the following exceptions:

- The MSN replaces RTP SN in the feedback information.
- The CLOCK option is not used
- The JITTER option in not used
5.3.5 Initialisation of TCP/IPv6 headers

5.3.5.1 Initialisation of IPv6 header

Static and Dynamic chains of IR or IR-DYN packets for profile 0x0006 MUST end with the static and dynamic parts of a TCP header.

![IPv6 header format](Figure 29: IPv6 header format)

**Static Part:**

```
<table>
<thead>
<tr>
<th>Version=6</th>
<th>Flow Label (MSB)</th>
<th>1 octet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flow Label (LSB)</td>
<td>2 octets</td>
</tr>
<tr>
<td></td>
<td>Next Header</td>
<td>1 octet (type of the following header in the static chain see section 5.7.7.8 RFC 3095)</td>
</tr>
<tr>
<td></td>
<td>Source Address</td>
<td>16 octets</td>
</tr>
<tr>
<td></td>
<td>Destination Address</td>
<td>16 octets</td>
</tr>
</tbody>
</table>
```

**Dynamic Part:**

```
<table>
<thead>
<tr>
<th>Traffic Class</th>
<th>1 octet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hop Limit</td>
<td>1 octet</td>
</tr>
<tr>
<td>Generic extension header</td>
<td>Variable length (section 5.8.6.1 RFC 3095)</td>
</tr>
</tbody>
</table>
```

**Eliminated:**

Payload Length
5.3.5.2 Initialisation of TCP header

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number</td>
<td></td>
</tr>
<tr>
<td>Acknowledge Number</td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td>Reserved</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent Pointer</td>
</tr>
</tbody>
</table>

**Static Part:**

- Source Port: 2 octets
- Destination Port: 2 octets

**Dynamic Part:**

- Master Sequence Number: 2 octets
- Sequence Number: 4 octets
- ACK Number: 4 octets
- Data Offset: 1 octet
- Reserved: 1 octet
- Master Sequence Number: 1 octet
- Window: 2 octets
- Checksum: 2 octets
- Urgent Pointer: 2 octets
- Options: Variable Length

**Eliminated:**

- Nothing

**Extra:**

- Master Sequence Number

Figure 30: TCP header format

5.3.5.3 Compression of TCP options

Header information from the packet stream to be compressed can be structured as an ordered list, which is largely constant between packets. TCP options are compressed using the “List Compression” encoding method.

The basic principles of list-based compression are the following:
1. While the list is constant, no information about the list is sent in compressed headers.

2. Small changes in the list are represented as additions (Insertion scheme), or deletions (Removal scheme), or both (Remove Then Insert scheme).

3. The list can also be sent in its entirety (Generic scheme).
6 ILLUSTRATION OF WAF OPERATION

This section illustrates the WAF operation with two examples. It aims at showing how the WAF entities, presented in the previous sections, interwork, so as to optimise communications in a multihop wireless network.

Optimisations are achieved by enforcing protocols boosters either over the local link or on end-to-end basis, as well as, by enabling an adaptive protocol stack on every wireless node. The latter is realized by exporting link layer information, while it might be useful, for (adaptive) routing protocols, in particular.

Particular emphasis is given to the co-operation of WAF with the multihop routing function. Focus is on the WAM service discovery and boosting procedures. Nevertheless, an example to illustrate the interplay between WAF and an adaptive routing protocol will be given in a future release of this document.

Discussion covers WAF boosting over the link between a source node and its neighbour (i.e., a node the particular source node can communicate directly), as well, WAF end-to-end boosting for communications between a source node and a destination node, which in turn, is located some hops away.

The first example assumes a node that has one network interface (WiFi), while the second example assumes a multihomed node (Bluetooth and WiFi). The examples provided in this section, are based on particular assumptions, and possibly don’t cover all possible cases that can be found in a multihop environment. However, they are used as indicative ones to illustrate the various facets of the WAF operation.

6.1 WAF AND MULTIHOP ROUTING CO-OPERATION

WAF is not a routing mechanism. It is a protocol framework, that allows for certain optimisations in a multihop environment. As such, it requires multihop routing. Multihop routing discovers routes in the multihop network, even between nodes that cannot directly communicate.

Furthermore, routing can be static or dynamic. The latter is realized by either a “proactive” or “reactive” routing protocol. The focus is mainly on dynamic routing. In general, proactive routing protocols discover routes in advance, while reactive routing protocols discover routes on-demand. Specifically, both types of protocols discover destinations, being either neighbours or far-destined nodes (located some hops away), manipulate accordingly a node’s routing table, and avoid routing loops. The main criterion for a route decision is the shortest path from a source to a destination node.

Typically, existing implementations of routing protocols run over 802.11 links, which in turn, operate in 802.11 ad hoc mode. It’s also typical (however, not a rule) for the routing protocol to discover neighbour(s) using “Hello” messages over 802.11 links. For example, existing AODV implementations do use “Hello” messages, which in turn, are encapsulated in UDP datagrams and broadcasted in certain time intervals. Broadcasting is an inherent characteristic for an 802.11 link. In addition, the connectionless nature of 802.11 allows the smooth layering of a routing protocol over such a link.
As opposed to 802.11 links, Bluetooth links are not broadcast by nature and a connection establishment procedure is always required. The broadcasting issue is addressed in the PAN profile using the BNEP protocol on both the PANU and NAP/GN nodes, and bridging support on the NAP/GN node. According to the PAN profile, a PANU should connect to a NAP/GN, and a user application should be present to cope with the connection establishment procedure. That is, in case of a Bluetooth link, the neighbour is always known, being the NAP/GN node.

The rationale for the co-operation between WAF and the routing machinery is twofold:

1. The routing machinery informs WAF for known routes to particular destinations. Specifically, once a route to a destination is added to a node’s routing table, this event is signalled to the wireless API, which in turn, signals the WAF user applications (e.g., WAM).

2. Link layer information exported by the wireless API is provided to the routing machinery, so as for the latter to eventually optimise its routing decisions.

In the subsequent examples, the focus is on point (1), i.e., how the routing machinery could interwork with WAF, so as for the WAM to initiate a discovery capability procedure.

### 6.2 Example 1: Node With a Single WiFi Network Interface

The first example illustrates the interactions between the different WAF entities in case of a node that has one network interface (WiFi). In particular, the example assumes communication between two wireless nodes in a multihop network. For the sake of discussion, the multihop wireless network arrangement shown in Figure 31 is assumed. Further assumptions are the following ones:

- Nodes B and E are assumed to be the source and destination, respectively.
- Communication between B and E is realized by a TCP connection.
- Multihop routing is governed by the AODV protocol, which in turn, runs on every node.
- Every node in the network is WAF capable.
- WAF supports three boosters, being FEC, ROHC TCP, and TCP booster X. Precisely, the FEC and ROHC TCP boosters are applied on a hop-by-hop basis, while TCP booster X is applied either on a hop-by-hop or an end-to-end basis (this is a theoretical one and used to ease the discussion in the particular example).
- Wireless links operate in 802.11 ad hoc mode.
- As shown in Figure 31, node B (the source) is in range with nodes A and C. That is, nodes A and C are neighbours for node B. Furthermore, node C is in range with node D, which in turn, is in range with node E (the destination).
When node B starts up, the following sequence of operations take place:

- AODV discovers neighbours A and C (through the exchange of “Hello” messages). AODV adds routes for A and C to the B’s routing table.

- At the same time, WAM on node B listens for incoming requests.

- To initiate the WAM neighbour capability discovery procedure, WAM on node B has two options: (1) to broadcast a message, or (2) to be informed for the available neighbour(s) by the wireless API, which in turn, is signalled by the corresponding alternation(s) in this node’s routing table.

- In case that the first option is selected in implementation, WAM broadcasts the WAM neighbour capability discovery request (WNCDRQ) message over UDP transport. A WAF-capable neighbour responds with a WAM neighbour capability discovery response (WNCDRS) message over UDP transport (unicast transmission). In our example, both A and C respond to B’s WNCDRQ message. This scheme allows WAM on node B to find out the neighbour(s) capabilities in-advance and apply the corresponding boosting policies over the local link(s). In addition, this scheme is advantageous in case that the routing protocol in use discovers next hop(s), only when a data transmission is requested (e.g., DSR).

- In case that the second option is selected in implementation, WAM is informed (through the wireless API) by the routing machinery for every neighbour that has been added to the node’s routing table. Upon reception of these events, WAM sends a WNCDRQ message (unicast) over UDP to every single neighbour. This scheme is advantageous in case that the routing protocol in use discover neighbour(s) prior to any data transmission request (e.g., proactive protocols, AODV).

- One way or the other, WAM on B discovers the capabilities of A and C. This leads to the enforcement of a particular boosting policy per neighbour at the PBF level. The boosting policy is associated with the MAC address of the node at the other end of the local link.
In our example, we assume that the link between B and C operates at a rather low speed, and would be beneficial to boost the link with the ROHC TCP booster. In addition, the FEC booster shields the link between B and A.

A TCP application on node B initiates a connection with node E (that is, B is the TCP source, and E is the TCP destination).

Since AODV governs the routing function in our example, upon reception of the TCP SYN segment, AODV on node B starts a route discovery to node E. The route to E is found, that is E is added as a destination to the routing table of node B, while the hop count equals to 3 and the next hop is node C.

WAM is informed for the new destination added in the routing table (through an event sent by the wireless API) and starts a WAM destination discovery capability procedure for node E. In the meantime, the packet is buffered by either the routing protocol or the PBF. The WAM destination discovery capability request (WDCDRQ) reaches node E, which in turn, responds with a WAM destination discovery capability response (WDCDRS) message. Upon reception of this message, WAM on node B instructs PBF for the corresponding boosting policy to be applied end-to-end (that is, TCP booster X boosts the communication between B and E).

PBF on node B, starts boosting TCP traffic destined to node E with TCP booster X, while it applies the ROHC TCP booster over the local link (between B and C).

6.3 EXAMPLE 2: MULTIHOMED NODE WITH BLUETOOTH AND WIFI NETWORK INTERFACES

Example 2 illustrates the interactions between the different WAF entities in case of a multihomed node (that is, the particular node has two network interfaces, being, Bluetooth and WiFi). It is assumed that Bluetooth is the (first) last hop link and the PAN profile is applied.

**Bluetooth node start-up:**

- The Bluetooth drivers (core driver, L2CAP and BNEP) and the 802.11b drivers are installed.
- The WAM and CM applications are launched.
- The following tasks are initiated by CM: (1) activation of the PANU service, (2) discovery of the BT neighbour devices providing the NAP or GN service, (3) BNEP connection establishment with the selected NAP/GN node, (4) activation of the 802.11b services.
- The WAM starts listening on incoming requests.
- The Bluetooth LLCT triggers the new neighbour event to the WAM.
- The WAM performs the WAF capabilities discovery with the neighbour node.
- If the neighbour node is WAF capable, the WAM gets the link quality from the Bluetooth LLCT; then, it defines the link boosting policy and instructs PBF to start boosting the link.

Traffic starts between the local node and a remote node located multiple hops away:
A user application sends IP traffic to a remote node located some hops away. The IP layer has no route to the destination, so it passes the packet(s) to the routing machinery.

The routing machinery buffers the packet(s) and performs the route discovery. As no WLAN network is available, the route will use the Bluetooth link as first (last) hop. Then, it triggers two events to the WAM, new neighbour and new final destination.

The WAM performs in parallel the WAF capabilities discovery with both the neighbour and the destination node.

If the destination node is WAF capable, the WAM defines the multi-hop boosting policy and instructs PBF to start boosting traffic with this node.

The WAM notifies the routing machinery that the boosting configuration is finished (the reason is that some protocol boosters need to process all the packets of a stream, for instance ROHC TCP; with this event, we could ensure that boosting is applied on every packet).

The routing machinery dequeues the packets and sends them over the network.

The PBF intercepts each packet and applies the required boosters on it.

**Link quality change on the Bluetooth link:**

- The Bluetooth LLCT notifies the WAM that the quality of the Bluetooth link has weakened under a given threshold.
- The WAM redefines the link boosting policy and passes the new configuration to PBF.

**Bluetooth node handoff:**

This example uses approach 1 of section 3.7.2.1: the Bluetooth LLCT detects that the Received Signal Strength has reached a given low threshold and it notifies the WAF applications.

The CM application looks for the best possible network:

- It first tries to join the WLAN network:
  - It starts hearing the beacons from the WLAN access point.
  - If the signal is strong enough, it switches to the WLAN network.
- If it fails, it tries to join another Bluetooth piconet:
  - It initiates a new neighbour discovery and, if it finds a new Bluetooth device with the NAP/GN service, it closes the current BNEP connection and establishes another connection with this new node.

The routing machinery also detects the link break and removes from its routing tables all the entries that used this link as first hop. If traffic flows over the broken link and there is no other route to the destination, the procedure described above is applied.

**Note:** when WAM is notified of a final destination removal, it should wait a certain delay before removing its WAF capabilities; if the route was deleted because of a link break while there is still
traffic with it, a new route may be found and it would be wasteful to discover again its WAF capabilities.

*Bluetooth node termination:*

- The user terminates the WAM and CM applications.
- The CM application terminates the PANU service, while the Bluetooth LLCT terminates the BNEP connection. It also terminates the 802.11b services.
7 CONCLUSION

The WAF architecture presented in this document fulfils the wireless multihop network requirements, as these were reviewed in section 1.2, in the following way:

1. **Compensation for wireless link impairments**: Wireless link impairments are compensated by tuning wireless driver parameters and/or applying a specific protocol booster. The Wireless Adaptation Manager (WAM) controls and configures the Protocol Boosters’ Framework (PBF) and checks the protocol boosters for proper operation. In addition, WAM monitors the link conditions through raw access (i.e., wireless API) to the wireless driver and/or protocol booster statistics and issues adjustment commands towards the wireless driver and/or protocol booster.

2. **Uniform Wireless Interface**: WAF provides a Wireless API for accessing all of the underlying mechanisms, including link layer protocols and driver parameters, routing tables and routing protocol parameters, protocol boosting configuration, monitoring and adaptation / control. Future wireless-aware applications, routing/mobility protocol daemons, connection/power managers can be built based on this interface.

3. **Power consumption mitigation**: In general, power consumption is mainly mitigated by efficient design and implementation, taking into account both CPU consumption (i.e. efficient design of computationally intensive functions) and transmission consumption (i.e. efficient protocol design and/or power-aware adaptation decisions). The WAF architecture, although does not provide a concrete solution to reduce power in a specific system or networking function, it provides the necessary framework to apply such adaptations by the specification of the mechanisms through which these functions may have their parameters altered, in order to follow alternate, power-aware strategies (e.g. power aware routing).

4. **IPv6 support**: WAF supports IPv6 transport, while interoperability with IPv4 networks is supported by existing transition mechanisms.

5. **Optimisations for multihop environment**: WAF architecture provides support for reduced, yet efficient, signalling, intelligent forwarding, and routing enhancements. The distributed solution provided by the PBF architecture and the WAF frame format in order to support the protocol boosting function eliminates the necessary WAF signalling only to a neighbours’ capability discovery exchange between peer WAM entities. Moreover, protocol boosting may be applied equivalently on both a link-level connection (single-hop) and an end-to-end remote connection (multihop), while routing protocols may leverage alternate route decisions by exposing certain parameters which control their operation through a well-defined common interface (wireless API).
8 APPENDIX

8.1 NETFILTER USE FOR PROTOCOL BOOSTERS FRAMEWORK DESIGN

The section 5.1.4 has explained how the Protocol Boosters Framework (PBF) needs to interact with the network layers stack. The implementation of such interactions is tightly dependent on the operating system in use. In the context of the 6HOP project, the Linux OS has been chosen as the enabling platform for the prototype WAF implementation in the testbed environments. So, this appendix describes how PBF will be interfaced with the Linux network layers stack.

Since the 2.4 kernel versions, Linux provides the Netfilter mechanisms [17], which aim to create a flexible framework to handle and modify packets (also referred to as packet mangling) outside of the Berkeley socket interface. After a Netfilter overview, the mechanisms are described in the context of the Bridge driver. The last subsection explains how Netfilter is used as an interface between PBF and the network layers stack.

8.1.1 Netfilter overview

Netfilter was originally created to modify and/or insert new functions within the network subsystem implementation in order to overcome some deficiencies that were detected in previous kernel versions regarding some of the functionalities that they offered. Specifically, it was originally foreseen to replace the Linux firewallsing code (ipchains), but it has evolved, and due to the fact that offers a large set of possibilities for the developers, it has become a framework which is supporting a large number of ongoing projects covering a broad research scope (ad hoc routing protocols, firewalls, Network Address Translators, existing protocols tuning, implementation of new communication components…).

It is mainly composed by three different parts. Firstly, some “hooks” are defined within each protocol (for instance, IPv4 and IPv6 define five each), which can be seen as well-defined points in a packet's journey through that protocol stack. At each of these points, the protocol itself will call the Netfilter framework with the packet and the hook number.

Secondly, some parts (mainly specific modules) of the kernel can register to listen to the different hooks for each protocol. In this sense, when a packet is passed to the Netfilter framework, it checks to see whether anyone has registered any entity for that particular pair of protocol and hook; if so, the programmer gets a chance to examine (and possibly alter) packet’s contents. Once the processing to the packet is finished, it can either been: (1) discarded (by the kernel) (NF_DROP); (2) accepted, meaning that it will continue its transversal of the protocol stack (NF_ACCEPT); (3) forgotten, meaning that it is the hook function itself which has to deal with the packet as the kernel will just skip it (NF_STOLEN); and (4) queued for user space (NF_QUEUE).

The last part concerns the way by which packets that have been queued are collected (using the ip_queue driver) in order to later send them to the user space. These packets are handled in an asynchronous fashion.

In the 6HOP project, the protocol boosters framework has to handle packets, which may have a WAF header preceding the IP header. So, it is not possible to use the Netfilter hooks at IP layer which can only handle IP datagrams. Another issue is to use the Netfilter hooks managed by the Ethernet Bridge driver.
8.1.2 Netfilter within the Bridge driver

The Figure 33 shows the possible traversals of an Ethernet frame within the bridge stack. These traversals are described in detail below, in the context of upstream and downstream traffic.

![Figure 32 Bridge defined netfilter hooks](image)

**Upstream traffic**

When a frame enters the bridge stack from a network interface inside the bridge, it is first passed to the BR_PRE_ROUTING netfilter hook, where any kernel part registered on this hook is called and can apply particular processing to the frame. Then, the frame is handled by the bridging algorithm, which is briefly described below:

- If the destination address is unicast and belongs to the forwarding database: if it is destined to a network interface inside the bridge, forward the frame on this interface; otherwise, pass it to upper layers.
- Otherwise: flood the frame on all the bridge network interfaces and pass it to the upper layers. These layers in turn decide whether to deliver this frame to the local host or to forward it according to the routing table.

Whenever a frame is forwarded to another interface inside the bridge, it crosses the BR_FORWARD and BR_POST_ROUTING netfilter hooks. On the other hand, when a frame is delivered to upper layers, it traverses the BR_LOCAL_IN hook.

**Downstream traffic**

When a frame enters the bridge stack from upper layers (originated in the same host or issued from another network interface outside the bridge), it arrives at the BR_LOCAL_OUT hook and is then processed by the bridging algorithm described above. At this point, the flow reaches the BR_POST_ROUTING hook, like incoming frames which are forwarded to another host.
8.1.3 Netfilter hooks used by the Protocol Boosters Framework

The choice of the solution exposed below was driven by the specificities of the wireless technologies involved in the project. For Bluetooth networks, the 6HOP project will consider the profiles that are best adapted to the project requirements; in particular the PAN profile [4], which defines access to a remote network and ad-hoc networking for Bluetooth devices of a single piconet. In this profile, the NAP or GN services perform Ethernet Bridge functions to forward packets from one PANU to another PANU, using a subset of the IEEE 802.1D standard [16].

So, in the PAN profile context, to apply protocol boosters on a Bluetooth link (e.g. between a NAP node and a PANU node), the best way is to rely on the netfilter hooks provided by the Linux Ethernet Bridge driver. PBF registers its output function near the BR_POST_ROUTING hook, so that it can handle all outgoing frames forwarded to any network interface inside the bridge. Similarly, it registers its input function near the BR_PRE_ROUTING hook, so that it can apply the required protocol boosters on all incoming frames issued from a network interface inside the bridge.

The Figure 33 illustrates that on an example: a node has two network interfaces, Bluetooth and IEEE 802.11b. In the Bluetooth network, this node offers the NAP (or GN) service, so it forwards frames from a PANU to another one using a bridging algorithm. When the frame enters the bridge stack from the BNEP0 network interface, the boosters input function is called from the BR_PRE_ROUTING hook; it can apply the protocol boosters based on the WAF header contained in the frame. On the other hand, when the bridge forwards the frame to its BNEP1 network interface, the boosters output function is called from the BR_POST_ROUTING hook; it can classify the frame according to the boosters policy and apply the corresponding protocol boosters.

The Figure 33 also shows traffic forwarded from a Bluetooth node to a 802.11b node using a routing mechanism at IP layer. In order to generalize the use of netfilter hooks as an interface with the protocol boosters framework, a dummy bridge network interface is installed over the 802.11b driver. Its role is only to allow the PBF to process the incoming and outgoing packets respectively at the BR_PRE_ROUTING and BR_POST_ROUTING hooks. As, in this case, all optional bridging functionalities can be deactivated (e.g. the STP protocol), the frame processing inside the bridge doesn’t add much time consuming.