UNIVERSITÀ DEGLI STUDI DI ROMA "TOR VERGATA"

FACOLTÀ DI INGEGNERIA

DOTTORATO DI RICERCA IN

Ingegneria delle Telecomunicazioni e Microelettronica

CICLO DEL CORSO DI DOTTORATO: XXV

Improving Service Support in Wireless Community Networks

by

Claudio Pisa

Supervisor

prof. Giuseppe Bianchi

Coordinators

prof. Andrea Detti

prof. Pierpaolo Loreti

A.A. 2012/2013

Abstract

Wireless Community Networks (WCNs) are an emerging phenomenon of grassroots spontaneous network building. Inside a WCN networking devices are deployed, owned and managed by different individuals or organizations, with minimal coordination, without a global planning or a common budget. The technical information is openly published, allowing for any skilled individual to independently join the network.

Wireless technologies are the natural choice for building these networks. IEEE 802.11 devices get progressively more popular, performing and cheap, and it is easy to install them on rooftops, whithout the need for the expensive construction work associated to copper or fiber optics deployments.

Moreover, WCNs are usually open to experimentation, which makes them rare, if not unique, real-World laboratories, with the additional bonus of several skilled users willing to try researcher's solutions, especially if these are published as open source software.

In this scenario, whose underlying technologies are depicted in Chapter 1, the problems on which this thesis focuses are:

- **Packet-droppers detection.** In a WCN misconfigurations that create black-holes in which user traffic gets dropped may easily occur. This problem is tackled in Chapter 2;

- **Distributed service discovery.** Rather than in a single central location, services are deployed around the WCN by its users. In Chapter 3 a mechanism to announce services in a decentralized fashion is devised and implemented;

- **Multicast multimedia streaming.** There is no native support for efficient

multicast stream delivery in WCNs. A solution is proposed and implemented in Chapter 4;

- **Scalable video streaming in WLANs.** Wireless technologies are used, in WCNs, not only for the transport network but also on the access networks, usually as open hot spots. In Chapter 5 a cross-layer mechanism is designed and implemented, which exploits scalable video coding to gracefully adapt to WLAN conditions and maximize the overall quality of video streaming;

- **Flexible and modular MAC layer services.** The cross-layer approach depicted in Chapter 5 demands for services provided at the MAC layer in order to be able to support complex and dynamic scenarios, which are common in the real World. In Chapter 6 a flexible, virtualizable and modular wireless MAC is is devised, and how this can be exploited in a scalable video streaming scenario.

Finally, the considerations in the Conclusions close this thesis, while the list of publications derived from my work are summarized in Appendix A.

Acknowledgements

I would like to thank all the people who made my Ph.D. possible: my girlfriend Simona and all my family, for supporting me in my choices; my coordinators: professors Giuseppe Bianchi, Andrea Detti and Pierpaolo Loreti; professors Douglas Leith, David Malone, Cristina Cano, Paul Patras and all the people at the Hamilton Institute, National University of Ireland, Maynooth, where I have been a visiting student: the work done there is still in post-processing, so regretfully it didn't make it into this thesis; and my collegues at the netgroup lab, especially my paper co-authors: Saverio, Michele, Riccardo.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis Overview

Wireless Community Networks (WCNs) are computer networks built by volunteers and characterized by the absence of a central management and by a distributed ownership of the infrastructure. The employed technologies are prevalently wireless, as devices are easier and cheaper to deploy on user rooftops. All the technical information is openly published on the Web, allowing for anybody to join independently, with just minimal coordination with the rest of the network members, but this also means that node updates are not easy, as many subjects have to agree and organize. The unreliability of the wireless medium and the distributed and unplanned character of WCNs pose new challenges to researchers. Some of these have been addressed during my Ph.D. and summarized in this thesis. Section 1.2 will introduce to the motivations and characteristics of this rising social phenomenon.

Sharing or reaching Internet access is the major driver to the building and growth of WCNs, but simultaneously users deploy services for other users inside their houses or offices, scattered over the network topology. Having an up to date central services directory is both difficult and artificial, because of the decentralized and chaotic nature of these networks. Chapter 3 illustrates our solution for distributed service discovery, which is now used in real WCNs.

Streaming of multimedia content is an increasingly popular service, which is also very resource demanding. Supporting multicast on the network can be an

effective way to mitigate the impact of these services, but the technologies employed in WCNs lack native support. In Chapter 4 we show how we can effectively build overlay multicast distribution trees, without the need for an update of all the devices of the network.

WCN members, which often come from a FLOSS background, have contributed to the evolution of routing protocols by extending existing open source implementations with new features, or devising new algorithms from scratch (§ Section 1.3). This makes the routing plane the natural choice for implementing network support for distributed service discovery and multicast streaming.

Also, in a decentralized self-managed environment, as the number of users grow, new problems may arise, such as misconfigurations or malicious behaviors. To foster the growth of WCNs, detecting intentionally or unintentionally misbehaving nodes and diverting user traffic to avoid them should also be supported by the routing plane. Chapter 2 deals with this problem.

Furthermore, at the edge of the network, users deploy wireless hot spots to ease access to the WCNs. If multicast support can save resources on the backbone, video streaming at the access network can be optimized by using scalable video and packet scheduling techniques. Chapter 5 tackles this problem. We show how we can achieve an already good performance with a simple application-layer scheduling approach and then how this can be improved through cross-layer packet scheduling. Nevertheless, during this work we clashed with the limits of the current architecture of Linux wireless drivers. Thus, in Chapter 6 we propose an alternative architecture, which is modular, flexible and virtualizable, and we show how this can be exploited in the video streaming scenario.

1.2 Wireless Community Networks

Wireless Community Networks are a new phenomenon, started around year 2000, and enabled by the availability of widespread, cheap and inter-operable wireless technologies, most notably IEEE 802.11 (Wi-Fi) [1]. Their initial socio-economic motivation was simply the reduction of the Internet access cost. The sharing of one person's broadband connection across a group of persons was made possible by

allowing group members to reach the network gateway through eventually multiple wireless relay links, typically implemented through the wireless bridging features native in the 802.11 standard (Wireless Distribution Service). Early wireless communities were hence collateral to the network, and with the explicitly stated goal of free-riding on them.

The number of wireless communities that emerged in the last years is impressive [2, 3, 4, 5, 6], and Europe seems to have taken the lead in such an expansion. As of today, their drivers are changing, and there is an emerging awareness of the high potentiality of the community network approach. Even if the motivation to have free of charge broadband access still remains, this is now only one among many others[1]. Wireless communities are in fact facing a much more constructive challenge: building bottom-up broadband wireless metropolitan area networks out of the cooperation of individuals. The network is available to the community members free of charge, and in addition to Internet access, it is devised to support services to meet the specific interests and social needs for the community.

Wireless community technology has also dramatically evolved. At the very beginning, wireless communities were built over standard protocol facilities (e.g. IEEE 802.11 WDS), but nowadays copper and fiber optics links can be easily found. To the extent that the word "wireless" is often omitted, and only "Community Networks" is used, even if the prevailing technologies remain wireless. Community members (typically practitioners in the areas of Packet Radio or Free Software) then started to add "intelligence" and capabilities to their networks, by extending and adapting operating systems (e.g. special Linux distributions such as OpenWRT), devising open-software wireless card drivers (e.g. the Atheros card driver MADWiFi), and implementing brand new protocols and solutions (e.g. the routing protocols OLSR or B.A.T.M.A.N., described in Section 1.3). The design and public domain distribution of network tools, mainly based on software, to be used on wireless devices has become an important part of community life thanks to groups of technically skilled volunteers. These software tools are typically customized to operate on extremely low cost devices, supporting cheap and easy to manage wireless

[1] Additional drivers worth mentioning include the willingness to narrow the digital divide areas that are poorly served by broadband, or bypassing the traditional ISPs for socio-political reasons, such as avoiding censorship, logging and tracking.

technologies. The marginal per-node costs, and the fact that each node cost is up to a single community member, has dramatically reduced the barriers for the wireless communities emergence, whose deployment basically only depends on the ability to find a suitable group of skilled people willing to start the community.

In order to regulate data transit between nodes, some agreements, analogous to FLOSS licenses, such as the Pico Peering [7], Freenetworks.org [8] and XOLN [9] have been devised by community network members.

In parallel, and independent of the wireless community efforts, but rather stimulated by the need of providers and the scientific efforts from the academy, significant improvements also occurred in the technical field of Wireless Mesh Networks (WMN). Wireless Mesh are multi-hop networks where an all-wireless backbone infrastructure is deployed among (typically fixed and wire-powered) wireless interconnected nodes, frequently called Mesh Routers and/or Mesh Access Points, to which ordinary clients connect as in a normal wireless access network. One or more Gateways provide the connectivity to the backhaul, while the other Mesh nodes extend the access coverage area by allowing remote nodes to connect to the gateway by relaying through multiple mesh nodes. WMNs have received the attention of standardization Task Groups in various technology work groups (e.g., 802.11s [10]).

The basic idea behind community networks is that the user itself becomes part of the network, eventually providing the access infrastructure to other users. This feature makes community networks highly scalable, both in terms of installation costs, and in terms of numbers of supported users. To reach this ultimate goal, the technology must be extremely simple and easily adaptable to changing network conditions. Moreover, community networks shift away from the classical provider-user paradigm, featuring a service provisioning model mainly based on cooperation. *Thus, complexity on context and unpredictability on service discovery, provision and management are both expected concerns, which have to be resolved. To this end, advanced service discovery and advertising techniques and methods enabling context awareness have been designed and implemented in this Ph.D. thesis.*

Multimedia group communications are among the most promising services for wireless communities and multicast is the most suitable approach to network resources optimization. Taking advantage of their broadcasting capability, WMNs are inherently well-suited for multicasting. However, most of the proposed multicast

routing protocols for ad hoc networks do not really address network resources optimizations of ad hoc communication. They often use broadcast mechanisms and build their own multicast trees, without taking into account MAC/PHY information or topology. This has severe impact on network resources consumption. *Therefore, in this thesis a novel approach to multicast protocols is proposed, taking advantage of an hybrid overlay cross-layer protocol, in order to introduce features missing to current solutions such as fast adaptability to topology dynamics, and routing protocol optimizations.*

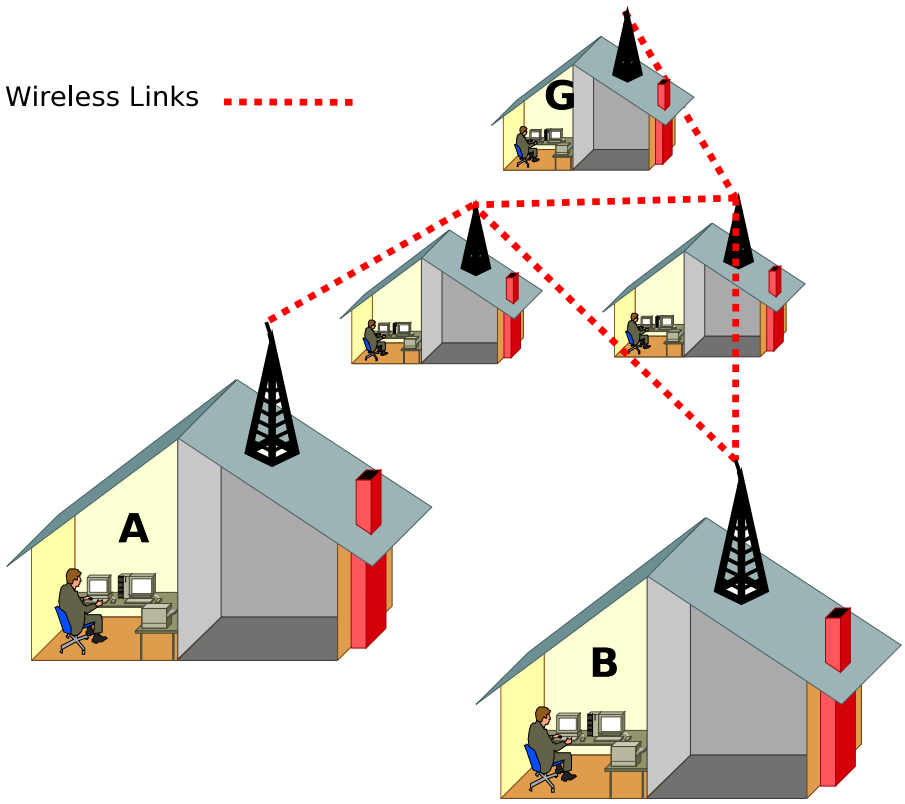


Figure 1.1: Sketch of a Wireless Community Network

In wireless community networks users can communicate with each other directly without accessing the public Internet. However, the typical use of these networks is to provide Internet access, with a subset of users sharing their broadband connectivity acting as gateways. Usually, community network members provide, along with Internet access, local services, such as voice over IP, DNS, gaming, file sharing,

Web pages, weather stations, to other community members. These services not exploited by the majority of the users. *We believe this happens because essential tools like DNS and service discovery are hard to deploy in highly distributed and anarchic networks.*

1.3 Routing Protocols Employed in Wireless Community Networks

Being prevalently wireless, the routing protocols used in community networks are tailored to this technology. These effectively take into account the unreliability of the wireless medium and make use of radio-aware metrics and flooding techniques. Among the most widespread routing protocols employed in Wireless Community Networks (WCNs) we can find the Optimized Link State Routing protocol (OLSR) [11], Babel [12] and Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.) [13], in its variants B.A.T.M.A.N. Advanced [14] and B.A.T.M.A.N. Experimental [15].

These protocols have been object of studies both by academics [16] [17] and by community driven experts [18]. A brief summary of the key aspect of each protocol follows, with emphasis on OLSR, as part of the work of this thesis focuses on extending this protocol with new applications.

1.3.1 OLSR

The Optimized Link State Routing (OLSR) protocol [11], originally devised for Mobile Ad-hoc Networks (MANETs), is arguably today the most widely used protocol in Wireless Community Networks. The reason behind its popularity can be probably tracked to the fact that it was among the first protocols to be extended with the Expected Transmission Count (ETX) metric [19], which aims at detecting high-throughput paths in multi-hop wireless networks. The ETX of a path is the expected total number of packet transmissions (including retransmissions) required to successfully deliver a packet along that path. For practical networks, paths with the minimum ETX have the highest throughput. The ETX metric incorporates the

effects of link loss ratios, asymmetry in the loss ratios between the two directions of each link, and interference among the successive links of a path. Busy networks that use the ETX route metric will also maximize total network throughput.

The OLSR protocol is an optimization of the classical link state algorithm tailored to the requirements of a mobile wireless LAN. The key concept used in the protocol is that of multipoint relays (MPRs). MPRs are selected nodes which forward broadcast messages during the flooding process. This technique substantially reduces the message overhead as compared to a classical flooding mechanism, where every node retransmits each message when it receives the first copy of the message. In OLSR, link state information is generated only by nodes elected as MPRs. Thus, a second optimization is achieved by minimizing the number of control messages flooded in the network. As a third optimization, an MPR node may chose to report only links between itself and its MPR selectors. Hence, as contrary to the classic link state algorithm, partial link state information is distributed in the network. This information is then used for route calculation. OLSR provides optimal routes (in terms of number of hops, if the ETX metric is not employed). The protocol is particularly suitable for large and dense networks as the technique of MPRs works well in this context.

Figure 1.2 shows the node in the center, with neighbors and 2-hop neighbors, broadcasting a message. In (a) all nodes retransmit the broadcast, while in (b) only the MPRs of the central node retransmit the broadcast.

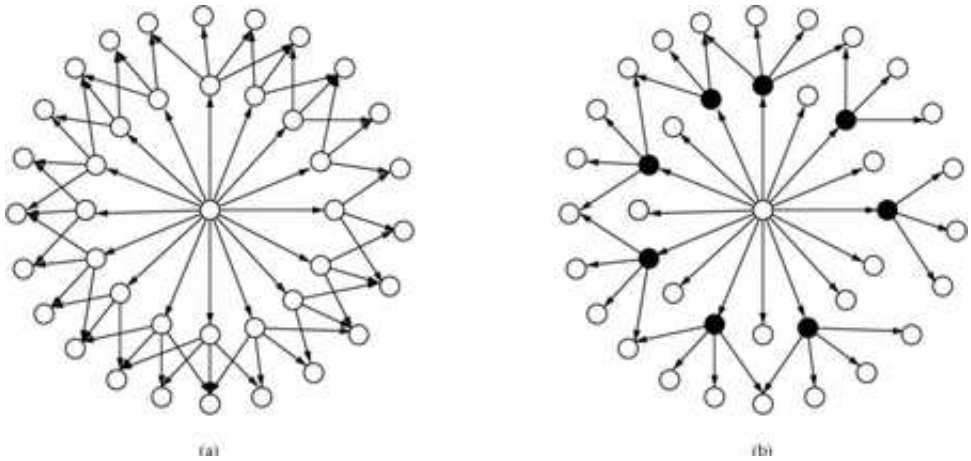


Figure 1.2: Pure flooding and MPR flooding

OLSR packets (Figure 1.3) contain one or more OLSR messages. It is possible to define new OLSR messages to add new features to the basic protocol, and this has been indeed done by the author as illustrated in the following chapters of this thesis.

To infer the network topology, OLSR sends over wireless links broadcast packets that any other node in radio proximity can receive. Two are the fundamental message types of the OLSR protocol: HELLO, and TC. HELLO messages are used for neighbor discovery and link sensing; these packets expire after one hop and are never forwarded. TC messages are used for network topology information diffusion; these packets are forwarded away from the originator to deliver topology information encapsulated into new OLSR packets at each hop.

Wireless community mesh networks are characterized by fixed nodes mounted on the roofs of buildings and houses. OLSR was originally devised for MANETs, with mobile nodes, but in the case of wireless community networks most of the nodes have fixed positions. These nodes act as OLSR routers, where one or more radio interfaces are connected to the mesh backbone and send and receive OLSR protocol routing packets. The other interfaces, typically wired, are attached to IP subnets announced into the mesh networks with HNA (Host and Network Association) messages. The end-user terminals do not have to run the routing daemon as their IP addresses are advertised by the nearest OLSR node. It is common that end-users get their IP address via DHCP from the nearest OLSR node.

The UniK OLSR[2] [20] is the implementation employed in practice by Wireless Community Networks, and thus this thesis work focuses on it.

A more detailed presentation of the protocol described in the RFC follows.

1.3.1.1 RFC 3626

The Optimized Link State Routing protocol (OLSR), described in IETF RFC 3626 [11], is a popular networking protocol developed for mobile ad hoc networks. It is a *proactive* protocol and, as the name suggests, is based on *Link State* routing. OLSR does not rely on any central entity nor makes any assumption on the underlying link-layer protocol, is aware of asymmetric links but does not exploit them, and uses an

[2]also known at olsrd or olsr.org implementation

optimization technique called *Multipoint Relaying (MPR)* to diffuse messages in the network.

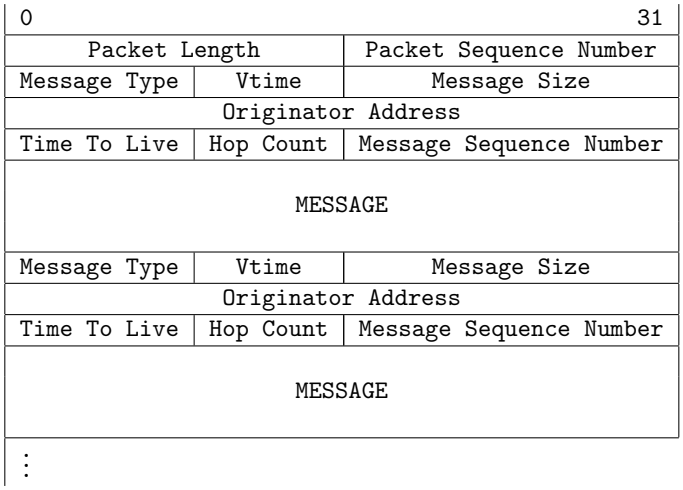


Figure 1.3: Basic OLSR Packet Format

1.3.1.1.1 Basic Packet Format OLSR packets are contained in UDP datagrams using IANA assigned port 698, and have the general format described in figure 1.3, where:

- **Packet Length** expresses the length of the packet, in bytes;
- **Packet Sequence Number** is incremented by one for each transmitted packet and is maintained separately for each OLSR interface[3].

OLSR messages, which may be of various types, are contained in the rest of the packet. All message types share a common header, with the following fields:

- **Message Type** specifies the type of the message;
- **Vtime** indicates the length of the validity time regarding the information contained in the message;
- **Message Size** specifies the size of the message;

[3] An *OLSR interface* is a network interface participating in an OLSR MANET.

- **Originator Address** is the *Main Address*[4] of the node that generated the message;
- **Time To Live** contains the maximum number of times that a message must be forwarded over the network, and is decremented at every hop;
- **Hop Count** is incremented at every hop;
- **Message Sequence Number** is a number that is unique for each message and is assigned by the originator node.

1.3.1.1.2 Multiple Interface Declaration (MID) Messages When a node has multiple *OLSR interfaces* participating in an OLSR MANET, the association between its *Main Address* and the addresses of all its OLSR interfaces must be announced to the other nodes in the network. This is accomplished through Multiple Interface Declaration (MID) messages, whose format is shown in figure 1.4.

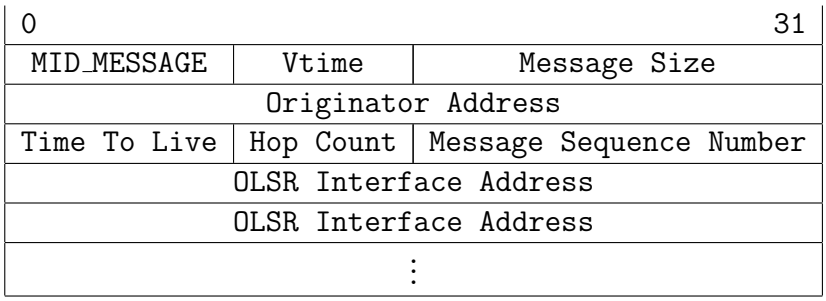


Figure 1.4: MID Message Format

The **OLSR Interface Address** field contains the address of an OLSR interface associated to the *Main Address* indicated in the **Originator Address** field.

1.3.1.1.3 Multipoint Relays (MPR) Classical flooding (i.e. forwarding of received messages to all neighbors) is very expensive in terms of bandwidth. That's why OLSR uses *Multipoint Relaying*: an optimization obtained by avoiding redundant retransmissions.

[4]The *Main Address* is an IP address that a node must choose as its *node id* among all the available addresses on all OLSR interfaces.

To achieve this, each node selects among its symmetric 1-hop neighborhood[5], considering their willingness[6], some nodes as *Multipoint Relays (MPRs)*, so that each 2-hop neighbor[7] can be reached through an MPR. The set of MPRs selected by node X is called the *MPR set* of X. Nodes that are in the MPR set of other nodes have the responsibility of forwarding their messages.

1.3.1.1.4 HELLO Messages *HELLO* messages are used for the purpose of link sensing, neighborhood detection and MPR selection. Emitted at fixed time intervals, contain information about the status of the node's links and neighbors. Their format is shown in figure 1.5.

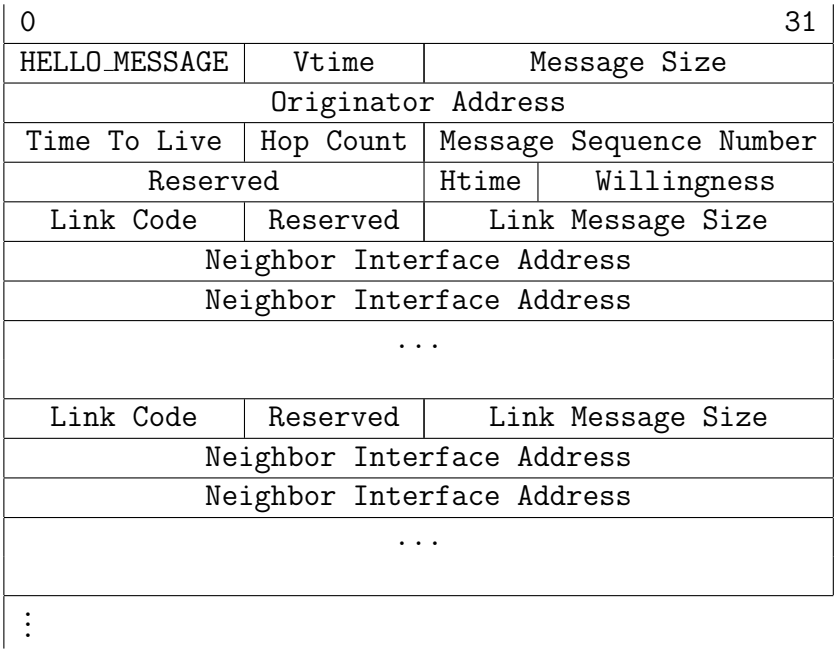


Figure 1.5: HELLO Message Format

- **HTime** specifies the size of the time interval between emissions of subsequent HELLO messages;

[5]The *symmetric 1-hop neighborhood* of a node is the set of nodes which have at least one symmetric link with the node itself.

[6]Willingness is explained at page 12.

[7]A *2-hop neighbor* is a node heard by a neighbor.

- **Willingness** indicates, with a number between 0 (`WILL_NEVER`) and 7 (`WILL_ALWAYS`), the willingness of the node to carry and forward traffic for other nodes;

- **Link Code** specifies information about the link (asymmetric, symmetric, lost, ...) and neighbor (symmetric neighbor, MPR, ...) whose interface is indicated in the associated **Neighbor Interface Address** fields;

- **Link Message Size** is the distance, in bytes, measured between two subsequent **Link Code** fields (or the end of the message).

1.3.1.1.5 Topology Control (TC) Messages In *Link State* routing, each node spreads information about its neighbors over the whole network. In OLSR this task is achieved by *Topology Control (TC)* messages, whose format is shown in figure 1.6.

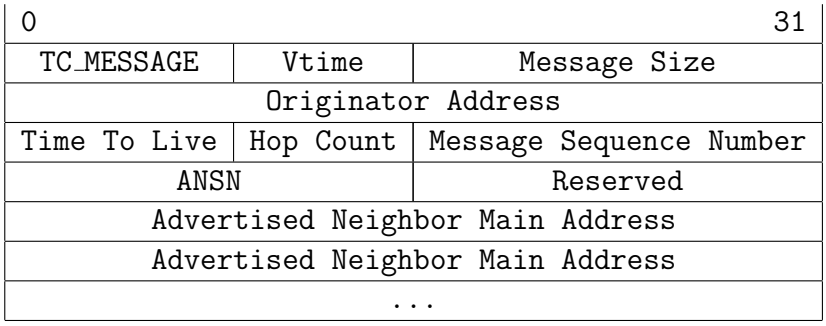


Figure 1.6: TC Message Format

- **Advertised Neighbor Sequence Number (ANSN)** is incremented every time the node's neighbor set changes;

- **Advertised Neighbor Main Address** contains the main address of a neighbor node.

If TC Redundancy (see subsection 1.3.1.1.8, page 13) is not used, TC messages are emitted by MPR nodes only.

1.3.1.1.6 Host and Network Association (HNA) Messages When a node has some network interfaces participating in an OLSR MANET and other interfaces which do not, it may be desirable to inject routing information in OLSR. *Host and Network Association (HNA)* messages serve for this task. Their format is displayed in figure 1.7.

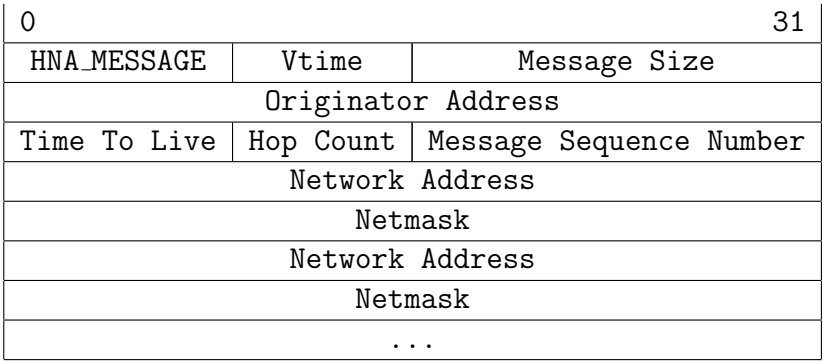


Figure 1.7: HNA Message Format

The **Network Address** and **Netmask** pair of fields specify the non-OLSR networks' data to be injected inside an OLSR MANET.

1.3.1.1.7 Link Hysteresis To prevent unstable links (not rare over the wireless medium) from having consequences on the stability of the information maintained by OLSR nodes or even affect the routing process, the *Link Hysteresis* mechanism may be applied.

It uses an upper and a lower threshold on the link quality. Asymmetric links may be considered by the protocol as symmetric only when their link quality is greater than the upper threshold, while symmetric links may be considered asymmetric only when their link quality is less than the lower threshold. In this way a delay is introduced in the link sensing process, but greater stability is achieved.

1.3.1.1.8 TC Redundancy By using the *TC Redundancy* mechanism, the robustness of the network topology information is increased. When an OLSR node parameter, called `TC_REDUNDANCY`:

- is 0, then the node includes in TC messages only the links with its *MPR*

selector set[8];

- is 1, then the node includes in TC messages the links with its *MPR selector set* and with its *MPR set*;

- is 2, then the node includes in TC messages all of its symmetric neighbors.

1.3.1.1.9 MPR Redundancy Optimization achieved by Multipoint Relaying may be traded off with more robustness with respect to topology changes by increasing the number of selected MPRs per node. This can be useful, for example, in mobile environments, where may be desirable to have the reachability of a node advertised by more nodes. A parameter called MPR_COVERAGE affects the MPR selection process by specifying through how many MPRs every two-hop neighbor should, if possible, be reached.

1.3.2 B.A.T.M.A.N.

To cope with problems arisen during the deployment of routing protocols in real World networks, members of the Freifunk wireless community network [3] designed and implemented the B.A.T.M.A.N (Better Approach To Mobile Ad-hoc Networking) routing protocol [13].

The UniK OLSR implementation has undergone a number of community-driven changes from its original specification in order to deal with the challenges imposed by city-wide wireless mesh networks. While some of its components proved to be unsuitable in practice (like MPR and Hysteresis) new mechanisms have been added by community network developers (e.g. Fish-eye and ETX). However, due to the constant growth of existing community mesh networks and because of the inherent requirement of a link-state algorithm to recalculate the whole topology-graph (a particularly challenging task for the limited capabilities of embedded router hardware), the limits of this algorithm have become a challenge. Recalculating the whole topology graph once in an actual mesh with 450 nodes takes several seconds on a small embedded CPU.

[8]The *MPR selector set* of a node is constituted by the nodes that selected it as MPR.

The approach of the B.A.T.M.A.N. algorithm is to divide the knowledge about the best end-to-end paths between nodes in the mesh to all participating nodes. Each node perceives and maintains only the information about the best next hop towards all other nodes. Thereby the need for a global knowledge about local topology changes becomes unnecessary. Additionally, an event-based but timeless flooding mechanism prevents the accruement of contradicting topology information (the usual reason for the existence of routing loops) and limits the amount of topology messages flooding the mesh (thus avoiding overly overhead of control-traffic). The algorithm is designed to deal with networks that are based on unreliable links.

The original B.A.T.M.A.N. has evolved into other community-driven projects, most notably B.A.T.M.A.N. Advanced [14], which is a layer 2 implementation and extension of the original protocol, now included in the vanilla Linux kernel distribution, and BMX6, or B.A.T.M.A.N. Experimental, which optimizes protocol overhead by being IPv6 oriented and aware, and exploiting distinctions between local and global addresses.

1.3.3 Babel

Babel [12] is a loop-avoiding distance-vector routing protocol that is robust and efficient both in ordinary wired networks and in wireless mesh networks.

Babel is a proactive routing protocol based on the distance-vector algorithm. It uses the Expected Transmission count (ETX) metric to select routes more intelligently than using a simple hop-count approach. BABEL has two distinctive characteristics that optimize relay performance. First, it uses history-sensitive route selection to minimize the impact of route flaps: the situation where a node continuously changes its preferred route between source and destination pair and leads to route instability. Thus, when there is more than one route of similar link quality, the route selection favors the previously established path rather than alternating between two routes. Second, BABEL executes a reactive update and forces a request for routing information when it detects a link failure from one of its preferred neighbors. Given the link quality measurements were previously completed at initialization stage, BABEL claims to have almost immediate route convergence time when triggering an explicit update.

Moreover Babel, in it's "Babel-Z" variant, is channel-aware, which makes it suitable for networks that include point-to-point links or networks that are built using multi-radio devices.

Chapter 2

Packet-droppers Detection in Wireless Community Networks

The focus of this chapter is the integration of trust in the routing plane of Wireless Community Networks. A modular framework is proposed for the detection and avoidance of misbehaving nodes.

Wireless community networks (§ Section 1.2) are prone to either unintentional (e.g. misconfiguration) or intentional node misbehavior. We introduce a fully distributed trust-based routing framework, tightly integrated with OLSR. The framework, designed to be modular for easy upgrade, relies on active probes, hidden in the normal data traffic through adaptation of steganographic techniques. The combination of path-wide measurements into a distributed trust framework, preliminarily based upon the well known EigenTrust [21] mechanism, permit to infer whether, and which, packet-droppers (i.e. nodes misbehaving at the data plane) affect the network forwarding operation. The resulting per-node trust values are then transformed into suitable "weights" provided as input to the OLSR protocol for mitigation through re-routing. A simulation-based performance evaluation shows that the proposed framework appears already effective in detecting and circumventing packet-droppers, despite the relative simplicity of the preliminarily considered algorithms.

2.1 Overview

The distinguishing characteristic of a wireless community network is the impossibility to rely on a centralized network management and monitoring framework, as almost each node composing the mesh is owned by a distinct entity, frequently an individual person.

Nowadays, there exist wireless community networks of more than 20 thousand nodes [5]. Even if relatively small with respect to commercially operated networks, these scales have become extremely challenging especially for management purposes. As the responsibility of controlling each node is ultimately up to its owner, misconfiguration impairing the network behavior may frequently emerge. Now, if routing problems are already non trivial to detect (e.g. taking advantage of the network-wise state each node maintains using information broadcasted by the OLSR protocol), data plane misbehavior is even more challenging, especially when it does not reflect into a routing anomaly. This is for instance the case of a node firewall misconfiguration: the node may still properly forward one-hop traffic, such as the OLSR control traffic, but may prevent the forwarding of multi-hop data traffic. In this case the routing is safe but the data delivery is unintentionally harmed.

On top of this, we should further consider that *intentional* node misbehavior cannot be ruled out in a relatively large community network, where it has become practically impossible for any participant to know (and trust) every other node. Intentional attacks to the OLSR protocol have been extensively addressed in both literature and in community deployments, and consensus has been reached on means for securing the OLSR routing plane [22, 23, 24]. Conversely, a consensus solution appears still lagging for what concerns disruption of data forwarding, where even in the case of a safe routing plane, a misbehaving node might agree to forward data packets but might fail to do so (we name such a node as "packet-dropper").

In the rest of this chapter we are going to show the following:

- the proposal of a modular framework that interworks with the OLSR routing protocol and ensures a reliable data delivery against the presence of packet-droppers;
- the design of a mechanism that uses ordinary packets as "hidden" probes, and

enables nodes to evaluate the reputation of other nodes only by exploiting its own data traffic and without the need of "overhearing" other node's traffic [25];

- the design of a link weighting function that integrates trust in the OLSR metric and practically allows to enforce routing on the path with the best path-trustworthiness, where path-trustworthiness is the trustworthiness of the *worst* node of a path;
- the preliminary assessment of the performance of the proposed approaches via NS2 simulation in the presence of UDP traffic (the extension to TCP traffic requires to further address more targeted attacks such as malicious SYN packets dropping, and is left to future work).

2.2 Design Requirements for Trusted Routing

We believe that approaches devised to detect and mitigate data delivery misbehavior in wireless community networks should be designed with in mind the following requirements.

First, the widespread adoption of OLSR calls for mechanisms that are readily integrated in the OLSR operation, i.e. the level of node trustworthiness computed by the trust-mechanism should drive the weighting of the links of the OLSR topology. Since OLSR forwarding occurs hop-by-hop on the basis of the routing table that each node has autonomously computed, it is necessary that all nodes practically operate on the same weighted topology to avoid routing loops in the network. This implies that the trust mechanism must provide *a global vision of the level of nodes' trustworthiness*.

Second, since the IEEE 802.11 WLAN is the widespread radio technology used in the deployment of wireless community networks, the trust mechanism must be *fully compliant with IEEE 802.11*.

Third, in order to limit radio interference and improve the network throughput, there is an increasing adoption of nodes with multiple radio cards and directional antennas, therefore the trust mechanism must be *compliant with multiple radio configurations*.

Fourth, the approaches should be the least invasive as possible and yield *minimal computational load* to be easily supported by cheap devices.

2.3 Proposed OLSR-based Framework

The proposed framework (Fig. 2.1) is composed by OLSR and three modules, named: reputation-module, trust-module and weighting-module. The reputation-module and trust-module form the trust-mechanism that assesses the nodes' trustworthiness, while the weighting module properly maps the level of nodes' trustworthiness to the weight of links of the OLSR topology.

The reputation-module of a node S computes the level of reputation that node S has of all other nodes. A node S evaluates the reputation of a node D by monitoring the correctness of data forwarding by node D. Reputation values computed by S are collected in a reputation-vector that is flooded in the network and locally sent to the trust-module.

The trust-module collects local and remote reputation-vectors and by combining them it computes a trust-vector. The i-th element of the trust-vector represents the level of trustworthiness of the i-th network node. The trust-vector is sent to the weighting-module.

Finally, the weighting-module makes use of the values of the trust-vector elements to derive and to enforce the weights of the links of the OLSR topology.

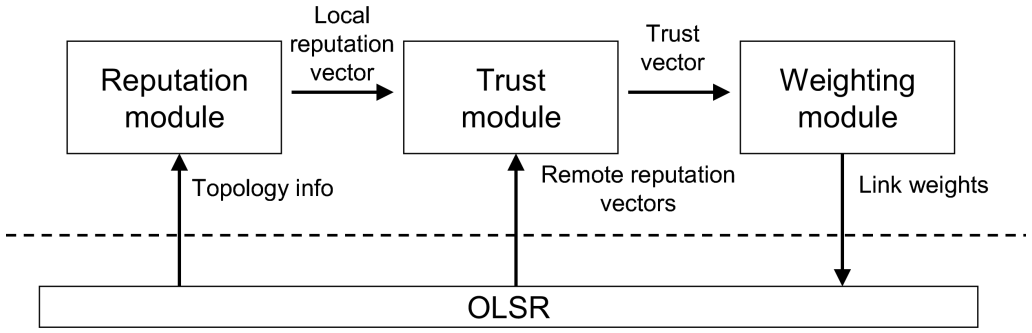


Figure 2.1: Trust-based routing framework

2.4 Reputation-module

The reputation-module assesses the reputation that a node S has of other nodes. A widespread solution to this issue consists in overhearing the traffic directed to neighboring nodes in order to evaluate the correctness of their forwarding operation [25, 26, 27, 28, 29]. Although effective in several scenarios, overhearing may be difficult in the presence of multiple directional antennas or when multi-rate transmissions are allowed, and these cases are common in a wireless community network scenario.

To support cases when overhearing is difficult we devise a reputation-module by adopting a *path-wide probe-based* approach that best fits connectionless UDP traffic. The underlying idea is the following: a node S selects another node D and sends to D probe packets. When the destination node D receives a probe packet it sends back a probe response [1].

An important issue consists in how to deploy probe packets. We promote an approach based on "implicit" probes, hidden in normal traffic, and whose details are presented in Section 2.7. This prevents a malicious attacker to operate by correctly forwarding probe packets, while dropping data packets without losing reputation. Note that probe responses do not require protection (i.e., they can be made explicit) as an attacker dropping responses would reduce its own reputation level. Moreover, the use of normal traffic at the endpoints does not require overhearing and limits computational complexity, since each node performs computation only on its own data traffic rather than on all forwarded traffic. It is worth to observe that in the presence of deployments where the goal is not to defend against malicious attackers, but only monitor the emergence of unintentional forwarding misbehavior, explicit probes may be further deployed to increase the number of paths controlled, and hence the effectiveness of the reputation framework.

When the probe response is received, node S increases the reputation of *all* nodes of the path S-to-D; otherwise, if the probe response is not received within a time-out (e.g., 500ms), the reputation of all nodes of the path S-to-D is decreased[2]. Which

[1]Note that in case overhearing was feasible, our path-wide probe-based implementation would work with or would be replaced by an overhearing approach

[2]In case of non-symmetric paths (e.g., in presence of OLSR ETX extension) also the reputation of the nodes of the reverse path has to be increased or decreased

are the nodes of the S-to-D path is inferred by the OLSR routing module .

In practice, the reputation r_{si} that a node S has of another node I is equal to the probability to get a response for a probing packet sent by S and that is deemed to pass through node I. We evaluate this probability through a moving average, based for instance on the last 10 samples. If a probe succeeds, then the value of the sample is 1; otherwise sample value is 0. In addition, a soft-state mechanism resets r_{si} to the default value, i.e. 1, if no probing (i.e., no traffic) interesting the node I has been done since a valuable amount of time, for instance 60 seconds.

The reputation values r_{si} are collected in a *reputation-vector*. This vector is flooded into the OLSR network by using *reputation-messages*, a new type of OLSR message defined by us, which contains the reputation-vector in its payload and is signed by the originator S. Flooding is achieved by using the OLSR default forwarding algorithm [11].

We observe that it is fair to increase the reputation of all the nodes of the path in case of a successful probing, because indeed all nodes succeed in data forwarding. Conversely, it may be unfair to decrease the reputation of all the nodes of the path in case of failure of the probing, because indeed only a single node may misbehave. This means that some node may obtain a reputation that is worse that the deserved one. Such an unfair decrease of reputation is clearly an adverse side effect of path-wide probing. However, the reputation-worsening is limited by combining locally and globally the results of different probes. Local mitigation occurs when node S exchanges probes with different destinations; in doing so a "good" node obtains a decrease of its reputation when the probe is on a path that includes both the good and a misbehaving node, but the good node obtains an increase in its reputation every time the misbehaving node does not belong to the probe path. Furthermore, reputation-worsening is globally limited because, as we explain below, the trust-module combines the reputation values estimated by other nodes.

2.5 Trust-module

The aim of the trust-module is to combine the reputation-vectors provided by the local and remote reputation-modules in order to compute a global level of trustworthiness for each network node. We tested in our implementation of the trust-module two well known trust frameworks, the well-know EigenTrust framework [21] and the ITRM [30] framework.

2.5.1 Eigentrust Framework

The EigenTrust framework fits well to our scenario, as departing from a formalization of the (natural) concept of transitive trust, leads to values of node trustworthiness that are global over the network, i.e. each network node computes the same values[3].

For sake of completeness we only report the formulas of the basic EigenTrust algorithm we used. First of all normalized reputation values c_{si} are defined as follows:

$$c_{si} = \frac{\max(r_{si},0)}{\sum_i \max(r_{si},0)} \tag{2.1}$$

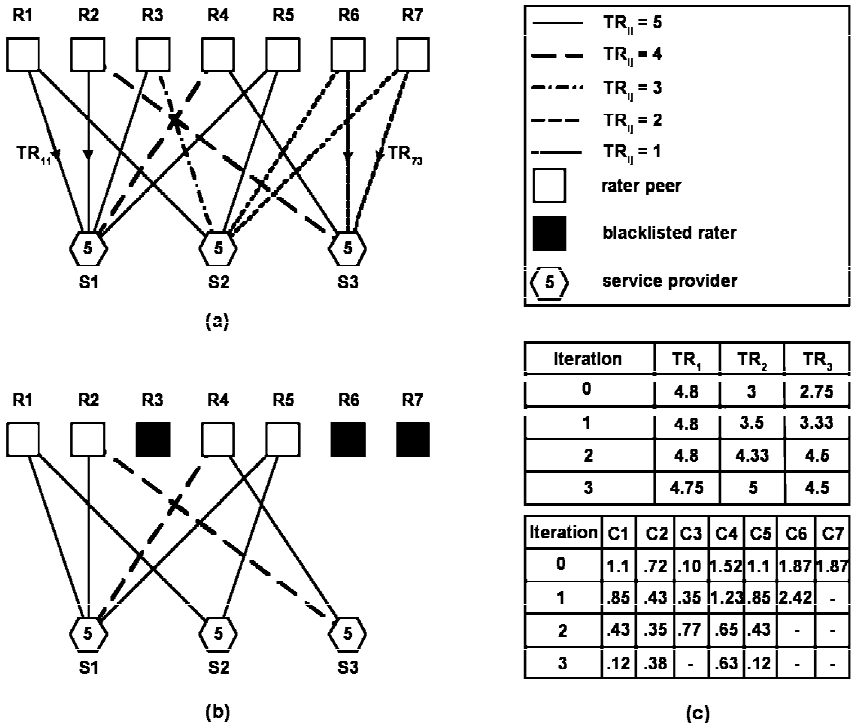
By defining t_i as the trustworthiness of node i, $\vec{t} = [t_i]$ as the *trust-vector*, $\vec{e}$ as a vector representing uniform probability distribution over all nodes, $C = [c_{ji}]$ as the matrix containing all the normalized values of the reputation-vectors and δ as a small value (e.g., 0.001), then the EigenTrust algorithm computes $\vec{t}$ through the iteration reported in algorithm 1.

2.5.2 ITRM Framework

We shortly describe the iterative method for trust and reputation management referred as ITRM. The algorithm can be applied to centralized schemes, in which a central authority collects the reports and forms the reputations of the service providers as well as report/rating trustworthiness of the (service) consumers. The proposed iterative algorithm is inspired by the iterative decoding of low-density

[3]It is worth to observe that in [21] the authors specify also a reputation system to compute r_{si} that unfortunately does not fit well in case of packet-droppers

parity-check codes over bipartite graphs. The scheme is robust in filtering out the peers who provide unreliable ratings. To use ITRM in our decentralized scenario, because every node collects all the reputation information collected by the other nodes with the OLSR flooding feature. At that point every node is able to individually run the ITRM algorithm to determine the trust vector. We integrated ITRM into our NS2 implementation, achieving better results than expected, as we show in Section 2.8.



Iteration	TR_1	TR_2	TR_3
0	4.8	3	2.75
1	4.8	3.5	3.33
2	4.8	4.33	4.6
3	4.75	5	4.5

Iteration	C1	C2	C3	C4	C5	C6	C7
0	1.1	.72	.10	1.52	1.1	1.87	1.87
1	.85	.43	.35	1.23	.85	2.42	-
2	.43	.35	.77	.65	.43	-	-
3	.12	.38	-	.63	.12	-	-

(c)

Figure 2.2: Illustrative example of ITRM [30]

2.6 Weighting-module

The weighting-module aims at weighting the links of the OLSR topology on the basis of the values contained in the trust-vector $\vec{t}$. We devise an implementation of the weighting-module with the following goal: *a longer path without malicious*

Algorithm 1 Basic EigenTrust Algorithm

$\vec{t_0} = \vec{e}$
repeat
  $\vec{t_{k+1}} = C^T \vec{t_k}$
until $abs(\vec{t_{k+1}} - \vec{t_k} < \delta)$
$\vec{t} = \vec{t_{k+1}}$

nodes is always preferred to a shorter one with a malicious node. We point out that such a goal requires a careful design of a trust-to-weight mapping function, since the (OLSR) shortest-path algorithm evaluates the cost of a path by *summing* the level of trustworthiness (i.e., link weights) of belonging nodes and we wish this cost to resemble the level of trustworthiness of the *worst* nodes.

We propose a trust-to-weight mapping composed by two steps: in the first step the trust values t_i are quantized in three symbolic values of trustworthiness, named tq_{high}, tq_{medium}, tq_{low}; in the second step link weights are derived by these symbolic levels of trustworthiness.

The symbolic quantized value tq_i of a trust value t_i is reported hereafter:

$$tq_i = \begin{cases} tq_{high} & \text{if } t_i \geq \frac{1}{N} \\ tq_{medium} & \text{if } \frac{1}{2N} \leq t_i < \frac{1}{N} \\ tq_{low} & \text{if } t_i < \frac{1}{2N} \end{cases} \tag{2.2}$$

where N is the total number of nodes in the network. The reason behind the selection of the threshold values of eq. 2.2 is that if all the N nodes of the network are well-behaving, then their normalized trust is $\frac{1}{N}$. By increasing the number of misbehaving nodes, the trust of good nodes increases beyond $\frac{1}{N}$ and the trust of misbehaving nodes tends to zero. We use three quantization levels to address the case of partial packet droppers, that will fall in the tq_{medium} level.

Now we insert the quantized values of node trustworthiness in the OLSR metric. We define the quantized trustworthiness tq_{si} of a unidirectional link between node S and node I as the lower quantized trustworthiness among tq_s and tq_i. The weight w_{si} of the unidirectional link S-to-I is a function of tq_{si} and therefore we have three possible numerical weights, named w_{low}, w_{medium}, w_{high}, that are respectively assigned in case of tq_{si} is equal to tq_{low}, tq_{medium} or tq_{high}. The values of w_{low}, w_{medium}, w_{high} should ensure that, whatever is the path stretch in terms of network hops,

shortest-path routing selects the most trusted path, where the path-trustworthiness is equal to the (quantized) trustworthiness of the worst node of the path. To achieve this goal it is enough that the values w_{low}, w_{medium}, w_{high} comply with the following two conditions:

$$\begin{cases} ML \cdot w_{high} < w_{medium} \\ ML \cdot w_{medium} < w_{low} \end{cases} \tag{2.3}$$

where ML is the maximum path length in the network. For instance, by choosing ML=10 the possible values would be: w_{low}=100, w_{medium}=1, w_{high}=0.01.

Finally, we observe that albeit we have considered only three levels of quantization (low, medium, and high) the reasoning that we followed can be repeated for more quantization levels.

2.7 Hiding Probes

To hide probe packets we leverage steganographic techniques for creating *implicit* probes "hidden" within the plain data traffic. To achieve this task, it is required that each sender and destination pair shares a dedicated secret. Secret sharing may exploit an eventual PKI deployed for securing the routing plane [22] or may be done through dedicated asymmetric cryptographic schemes[4].

Let us assume that S sends UDP traffic to a destination D, and that S and D share a dedicated secret k_{SD}. When an UDP/IP packet P is sent from S to D, it is recognized as probe by both source and destination nodes (and only by themselves, because of the secrecy of k_{SD}) if the condition $HMAC_{k_{SD}}(P) \leq threshold$ holds. Here, HMAC is a keyed hash construction based on a secure one-way hash function, keyed with the node pair secret, and threshold is a configuration value that permits to configure the percentage of probes (1/32 in our specific case, by setting threshold equal to 1/32 of the maximum HMAC value). Furthermore, we recall that the HMAC must be applied to the content of the packet P which is not mutable during its forwarding in the network (for instance, its TTL which must be hence excluded

[4]For instance, using Diffie-Hellmann (but the same holds for many other schemes, e.g. identity-based cryptography, bilinear pairings, etc), each node can preliminary notify to all the remaining nodes (e.g. through broadcast delivery) its public Diffie-Hellman parameter, so that each remaining node can asynchronously compute a per-node-pair shared secret.

from the hash computation).

Since a malicious packet-dropper does not know k_{SD}, it cannot compute the HMAC and hence detect whether the packet is a probe or not. The only possible weakness occurs when packets P which satisfy the condition of being probes happen to appear duplicated in the data stream, as in this case a node keeping track of all the delivered packets and recognizing them as probes based on the occurrence of a probe response would be able to identify the ones subsequent to the first one. Note that the first packet remains protected, as this is disclosed to be a probe only too late from the point of view of a malicious node. This issue could be in principle avoided by adding freshness to *every* packet payload - e.g. in the form of a random trailer. However this would need to remove such extra data at destination and complicate the implementation. In practice, we think that the presence of application layer headers which contain changing information (such as sequence numbers and timestamps in RTP/UDP streams) is sufficient to avoid this concern.

The probe response is a special packet constructed by the destination. It does not need to be hidden, as the lack of probe response delivery to the source would imply node misbehavior (and hence a node dropping probe responses would harm its own reputation). Therefore, it is constructed as an explicit UDP message that contains a (different) keyed message authentication code for the original packet P. As such, the source is able to link the probe response to the packet P initially send, whereas a malicious node is not able, missing the secret, to forge fake probes. Any duplicated or invalid probe response is discarded by S.

Finally, we observe that an implicit probing mechanism does not fully address the scenario of TCP traffic. Indeed, a malicious packet-dropper might immediately discard packets that establish a connection thus preventing the exchange of traffic and probes. Moreover, also considering initial TCP SYN packets as probes would be ineffective because a malicious node could forward these packets but drop the remaining traffic, thus blocking further probing. We argue that a possible approach for devising a reputation-module for TCP traffic could be based on the analysis of connection level phenomena such as TCP timeout.

2.8 Performance Evaluation

We test our algorithm in a mesh network formed by 25 fixed nodes disposed as in figure 2.3. We use Network Simulator 2 (v2.34), enriched by the OLSR modules of [31]. A node can directly communicate only with its horizontal or vertical neighbors. MAC layer is IEEE 802.11 operating at 11 Mbps. A simulation lasts 300 sec; beginning at 20 sec and every 10 sec each node sets up a CBR/UDP session with a random destination; UDP packets have a length of 1452 bytes and the UDP bit-rate is 220 kbps. Regarding the probing performed by the reputation-module, we consider a threshold equal to 1/32 of the HMAC maximum value, i.e. in average a probe every 32 delivered packets. Therefore during an UDP session of 10 seconds we have on average 6 probes; the reset of a local reputation value occurs after 60 seconds of probing inactivity. Regarding the model of the malicious node we assume that it drops all UDP packets and advertises a bad reputation for all other nodes.

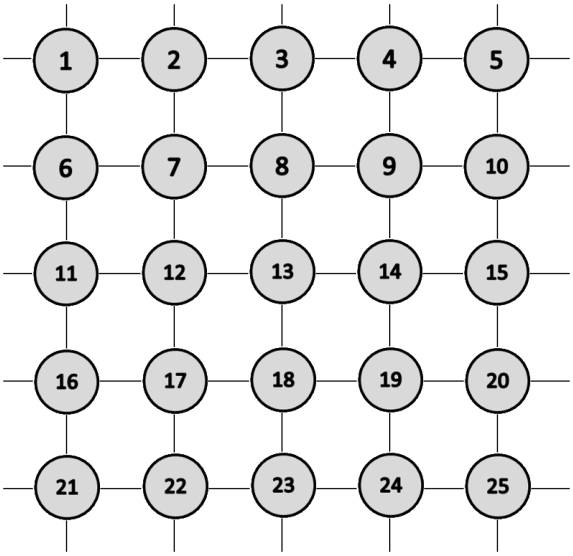


Figure 2.3: Simulation scenario

Figure 2.4 reports the time evolution of the quantized trust value tq_i of nodes 12, 13 and 14, in case node 13 drops all the UDP packets to be forwarded while the other network nodes correctly forward packets. Since the beginning of UDP traffic, i.e. 20 sec, the malicious node 13 gets a low level of trust, while neighbor nodes 12 and 14 get a high trust level. We also observe that good nodes 12 and 14 get

during a very small period of time a wrong reputation and this is an evidence of the worsening effects coming out from the path-wide probing approach that we followed (see section 2.4); conversely sometimes nodes 13 gets a high level of trust due to the temporary reset of the soft-state mechanism.

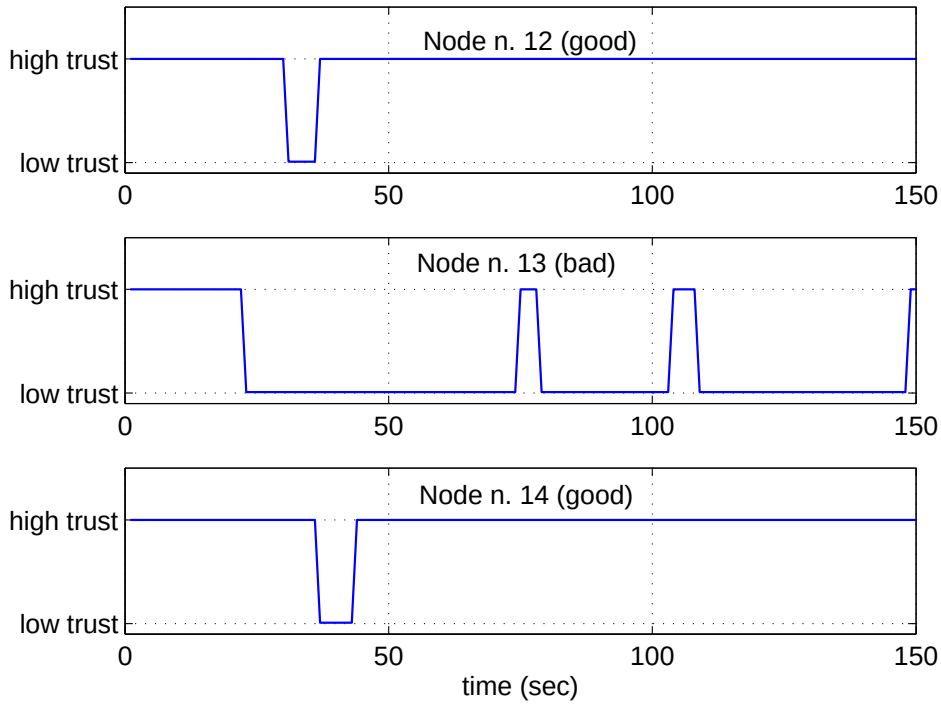


Figure 2.4: Time evolution of quantized trust values in case node 13 is a packet-dropper

Figure 2.5 shows the number of packets delivered to every node for forwarding. When trust-based-routing is disabled (upper plot) the three dropper nodes intercept a significant number of packets and the total amount of forwarded packets in the network is lower. Enabling trust-based-routing the droppers are identified and skipped, leading to a higher number of forwarded packets for the other nodes of the network.

Figure 2.6 shows the effectiveness of the trust-based-routing in coping with many independent packet-droppers[5]. We measure the effectiveness E as the reduction

[5]The ids of the dropping nodes are: 1-dropper: {13}, 2-droppers: {12,14}, 3-droppers:

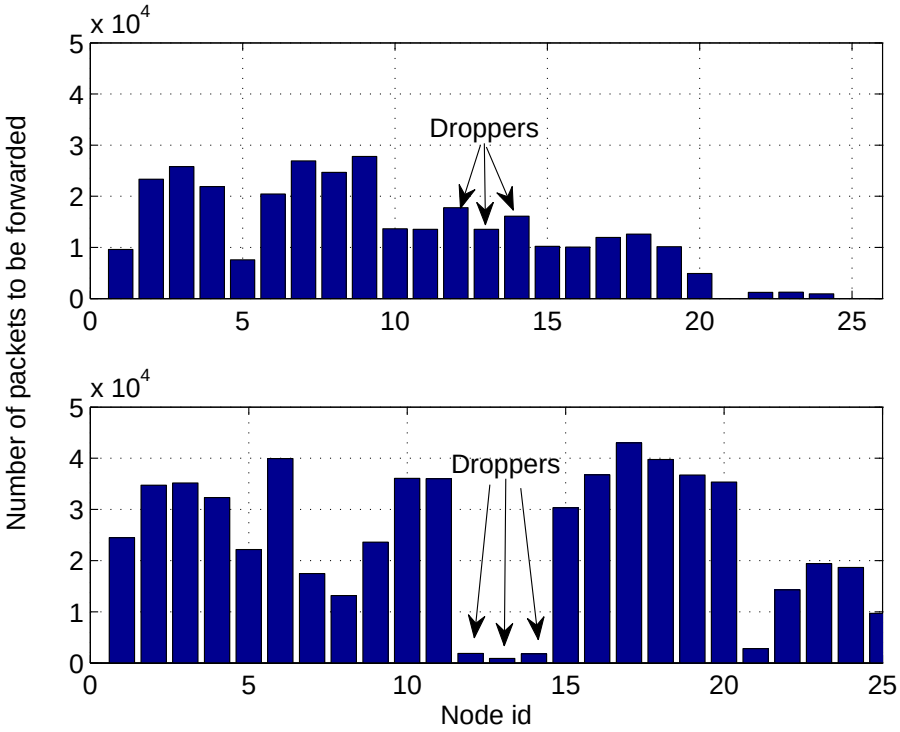


Figure 2.5: Number of packets to be forwarded by network nodes when nodes 12, 13 and 14 are packet-droppers, in cases of absence (upper plot) and presence (lower plot) of trust-based-routing

of packets intercepted by droppers when the trust-based-routing is activated, in formula:

$$E = 1 - \frac{P_{trust}}{P_{plain}}$$

where P_{trust} and P_{plain} are the number of packets intercepted by packet-droppers in case of trust-based-routing and in case of plain OLSR routing respectively. In case of a single packet-dropper, we observe a reduction of the number of intercepted packets that is about 97%. We do not obtain a reduction of the 100% since the reset timeout of reputation-modules allows traffic to be temporary intercepted by the droppers. Obviously, by increasing the number of droppers we have a decrease of E that however remains beyond 90% up to 4 packet-droppers.

{12,13,14}, 4-droppers: {7,9,17,19}, 5-droppers: {7,9,13,17,19}, 6-droppers: {7,8,9,17,18,19}, 7-droppers: {3,7,8,9,17,18,19}

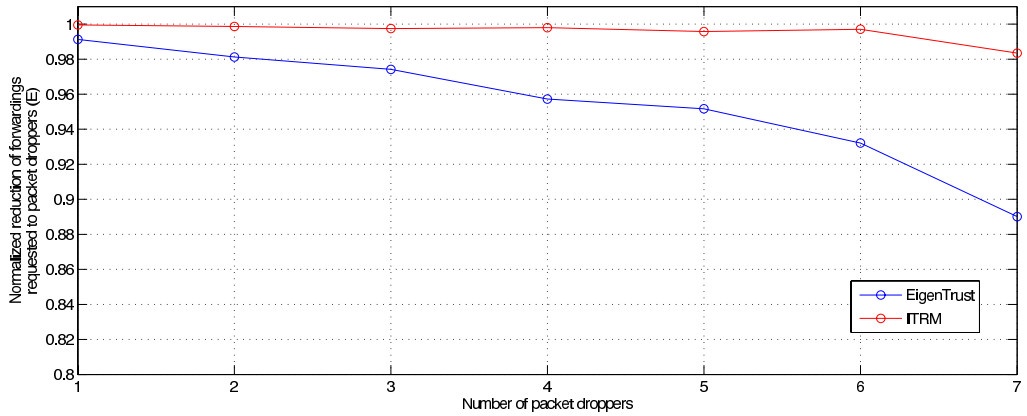


Figure 2.6: Normalized reduction (E) of number of packets to be forwarded by packet-droppers in case of trust-based-routing with respect to the case of plain OLSR, versus the number of packet-droppers

Finally, figure 2.7 shows the impact of the reputation reset timeout and of the number of probes per second in case of 3 packet-droppers. We observe that both parameters have a marginal impact on system performance on the condition that a too short timeout (e.g., lower than 20 sec) and too few probes per second (e.g., lower than 3 probes) are avoided.

The last fact we want to show, is that our mechanism does not introduce losses when there is not any attacker or any faulty router. While it is clear that in case of an attack the mechanism reduces packet loss because the network paths do not traverse the attacker node anymore, it is not straightforward that the trust framework do not introduce packet loss in case that no attack is in place. To demonstrate this we simulated a congested network with a packet loss around 30% and no attackers in place. When we turn on the framework, either with the EigenTrust or the ITRM trust modules, we note that we have just a small increase of packet loss (see figure 2.8. This small increase of packet loss is due the loss of probes because of network congestion. However the additional loss is so little that is possible to neglect it.

2.9 Related Work

This section surveys design aspects of relevant and recent work in literature aimed at mitigating attacks on the data plane in wireless multi-hop networks.

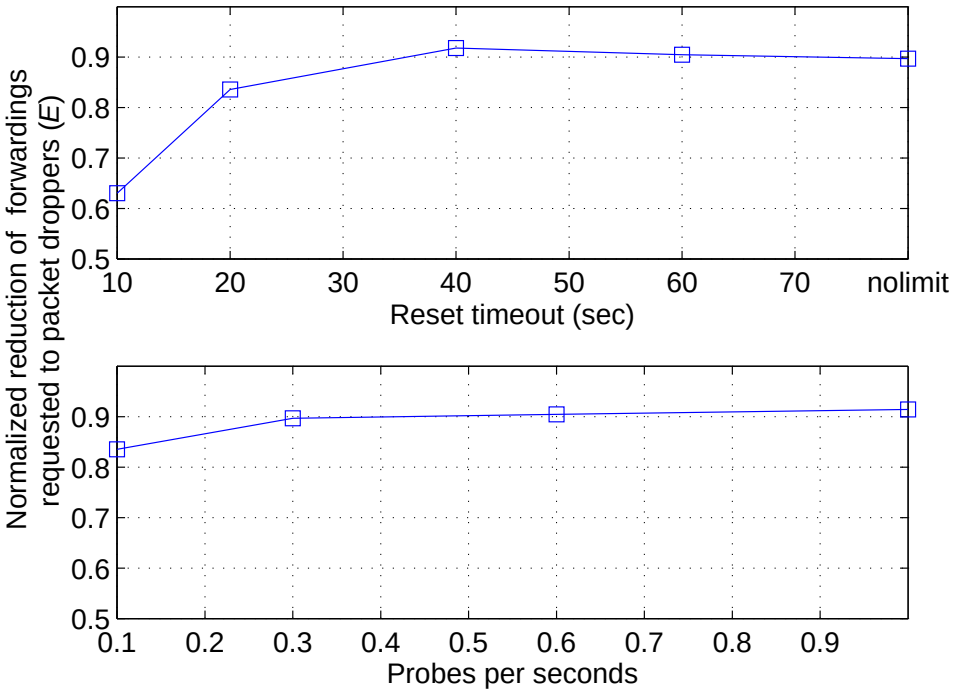


Figure 2.7: Normalized reduction (E) of number of packets to be forwarded by 3 packet-droppers in case of trust-based-routing with respect to the case of plain OLSR, versus the duration of the reset time-out and the frequency of probe packets

In [25] the authors introduce the idea of two modules called *watchdog* to identify misbehaving nodes, and *pathrater* to help the routing protocol (DSR), in choosing routes that avoid these nodes. To identify misbehaviors, the watchdog "overhears" neighbors, by putting its network interface in promiscuous mode, to check if data packets are actually forwarded by them, and then this information is used by the path rater as a metric to compute the best routes. In other works similar neighbor monitoring mechanisms have been exploited, together with protocols devised to spread reputation values over the network: CONFIDANT [26] makes use of alarm messages, while CORE [27] propagates this information through a *provider and requestor* scheme. Moreover, in [28], the monitoring is enhanced by using unkeyed hashes in the DSR route discovery phase.

In [32] the proposed scheme copes with attackers that independently fail to forward data packets or are not honest in propagating trustworthiness information to

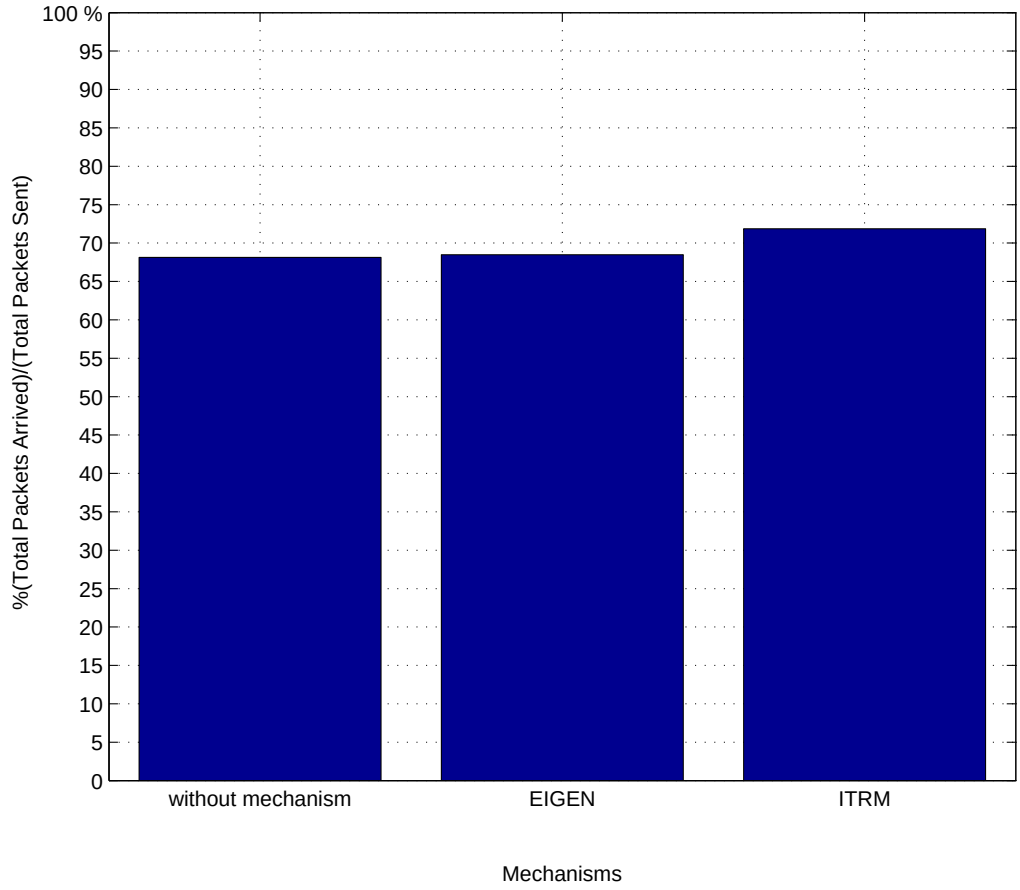


Figure 2.8: We show that when there is no attacker in the network, the activation of our framework does not increase packet loss because of lost probes

the other nodes. A hop by hop acknowledgment scheme, based on overhearing, is used to gather first-hand trust information in the assumption that the underlying routing protocol is based on source routing. When insufficient information is available the node can query a set of recommender nodes to obtain second-hand information. The amount of second-hand information used can be tuned to achieve either the goal of accuracy or speed of detection.

The usage of the watchdog mechanism is instead limited, in [33], to the monitoring of broadcast packets, while for unicast packets two-hop cryptographic acknowledgments are employed. The proposed scenario is a DSR MANET, where a PKI is assumed to be in place, and both attacks to the data and control planes are considered. The gathered reputation information is then used as the basis for an

accusation-based collaborative node isolation mechanism.

In [34] authors introduce the concept of witnesses, i.e. nodes that do not belong to the data forwarding path but that are able to monitor (by overhearing) nodes belonging to it. Witnesses send reports to the packet source on the forwarding behavior of the monitored nodes that belong to the data path. The authors provide an analytical model and study its performance. In the special case of a network topology that does not allow for witnesses, the mechanism falls back to the watchdog approach.

In [35], the authors devise a reputation-based mechanism for AODV MANETs with very mobile and sparse nodes based on EigenTrust. Each node collects information on other nodes checking, with a mechanism similar to the previously mentioned watchdog, if data is actually forwarded. This reputation information, spread over the network, together with information on the centrality of the nodes, i.e. how wide is their view of the network, is then used to classify nodes into zones, after applying a variant of the EigenTrust algorithm, to compute trust values.

The key difference between our work and the current state of the art, is that the proposed framework enables nodes to evaluate the reputation of other nodes without the need of overhearing, this makes possible to use the framework in case of multiple directional antennas setups or when multi-rate transmissions are used. Moreover, the proposed path-wide probe-based approach requires the use of a link state routing protocol.

2.10 Chapter Conclusions

This chapter illustrated a modular trust-based routing framework for OLSR based Wireless Community Networks. The proposed approach does not require overhearing, and limits computational load since a node only performs packet-level trust computation upon packets of its traffic. Such virtues are achieved by assessing the nodes' reputation (in terms of proper forwarding of data packets) via a path-wide approach leveraging "hidden" active probes deployed into the normal data stream. A preliminary performance evaluation shows that the proposed approach is effective, even if it is based on a very simple reputation algorithm (Section 2.4) and it

is targeted to detect "packet droppers". As future work, we plan to improve the extent and effectiveness of the framework in correctly and rapidly identify misbehaving nodes, by devising more sophisticated inference algorithms capable to better assess the forwarding operation of an intermediate network node using only end-to-end measurements (e.g. using methodologies similar to those exploited by network tomography research [36]).

The work illustrated in this chapter has been presented at the conference IEEE ICC 2011 with an article entitled "A framework for Packet-Droppers Mitigation in OLSR Wireless Community Networks", co-authored by Francesco Saverio Proto, Andrea Detti and Giuseppe Bianchi.

Chapter 3

Multicast DNS and Service Discovery in Wireless Community Networks

In this chapter we propose an extension to the OLSR protocol to support the delivery of mDNS traffic. This protocol enhancement makes possible to perform distributed name resolution and service discovery in community networks, using standard tools already installed on most users' computers. The designed protocol extension has been implemented and released open source. The solution has been tested both on standard PCs and on embedded devices running OpenWRT, and included in the olsr.org standard distribution [37].

3.1 Overview

OLSR is one of the most widespread routing protocol in wireless community networks [3] [38] [6] [2]. These open infrastructures are ran by volunteers to offer Internet connectivity to neighborhoods or villages. Even if the users are connected directly one to each other on high speed wireless networks, few services other than Internet connectivity are popular among the members of the communities. To the best of our knowledge community services are not exploited, because wireless community networks (WCNs) usually lack reliable internal DNS servers and service

directories. WCNs are highly distributed and unstable, and this makes very hard to deploy centralized services. A DNS or service directory server could be most of the time unreachable to the majority of the users due to failures or network splits. Moreover, the standard DNS protocol [39] requires a centralized server as source of authority for the domain, but in the anarchic scenario of communities there is not a single entity responsible for the network's deployment, management and maintenance. In other words, WCNs are not distributed only in terms of topological position of the nodes, but also in terms of administrative domains. Setting up a DNS service for the domain of a WCN presupposes that an administrative domain covers the whole network, but this is not necessarily true.

When a central server is not available, name service protocols installed today on most users' terminals [40] [41] send broadcast queries. However, in a network running the OLSR routing protocol, the broadcast domain is limited to the 1-hop neighborhood. This makes it impossible to use a name service protocol that requires a broadcast medium.

The goal of the work described in this chapter is to provide distributed name resolution and service discovery in WCNs running the OLSR protocol. We devise a transport mechanism for mDNS [42] packets in a OLSR network, to perform name resolution where it was not possible with limited broadcast-multicast domain. Moreover, once the network is able to transport mDNS packets, it is possible to exploit DNS based Service Discovery to run decentralized applications with other users in the community network. The solution we present is completely distributed, it is backward compatible with the already deployed nodes in the network and it is transparent to the end-user. Standard user applications that are off-the-shelf in recent operating systems, like browsers or chat clients, are already able to exploit services announced via DNS based service discovery.

Multicast DNS (mDNS) [42] is a protocol that uses APIs that are similar to the ones of the normal unicast Domain Name System, but implemented differently. The details of the mDNS protocols can be found in [42]; in the following we give a short explanation functional to understand our work. Each host on the multicast domain stores its own list of DNS resource records (e.g. A, MX, SRV, etc) and acts as a DNS server. When an mDNS client wants to resolve a name, it sends a DNS query in multicast, and the host with the corresponding A record replies with its

IP address. There are no central hosts responsible for the functioning of the whole system, and in case of failure or network split, the service still works between the users connected. Because of the lack of a central repository, names are assigned on a first-come basis. If a host discovers that its hostname is already taken from another host in the network then it chooses a new hostname.

DNS based Service Discovery (DNS-SD) is built on top of the Domain Name System. It uses DNS SRV, TXT, and PTR records to advertise Service Instance Names. The hosts offering the different services publish details of available services like instance, service type, domain name and optional configuration parameters. Service types are given informally on a first-come basis.

In figure 1.1 we can see a sketch of a typical wireless community deployment, where the node G shares its Internet connectivity acting as a gateway for all the other nodes. Obviously if the user attached to node A wants to communicate with the user attached to node B, it is not necessary to pass through the public Internet. However, if a user is offering a service on his host (e.g. a Web server) another user willing to connect to this service must know a priori the IP address and the port where the service is located. This is not feasible because i) the IP address may change over time, ii) users may not know each other a priori and iii) most of them are not skilled enough to go into the details of IP addresses and TCP/UDP ports.

To solve the problems listed above, we take advantage of the fact that most users' computers will generate mDNS traffic to announce the active network services. Moreover, hardware produced in the past few years like network printers and network attached storage boxes (NAS), generate mDNS traffic to announce their presence in the network and the available network services.

The mDNS traffic scope is inside the subnet of the user, and will not pass beyond the OLSR router. To extend the multicast domain for mDNS traffic, an OLSR node equipped with our mDNS plugin can passively capture mDNS packets and forward them into the mesh network as OLSR signaling (as we explain in detail in section 3.2). The captured traffic is then received by the other OLSR nodes that can reproduce it on their attached subnets.

Network addressing configuration (or autoconfiguration) for the OLSR nodes and their attached subnets is not in the goal of this work. We assume that the network has a consistent addressing, and that all the nodes are configured correctly. Every

host must have a unique IP address and must be routable: IP unicast connectivity between any pair of nodes must be possible. If the network is not fully routable, unreachable services may be announced, as a route to reach the host running the service may not exist. Because all the users must be able to offer services from their hosts, NAT must be avoided. It is useless to announce a service on a host behind a NAT: the service will not be available if does not exist a route to the IP address where the service is announced.

3.2 The OLSR Protocol Extension for mDNS transport

The OLSR protocol is extensible to provide additional functionality if desired. The OLSR packet (Figure 1.3) is a transport container for different messages.

As illustrated in Section 1.3.1 there are two fundamental message types in the OLSR protocol: HELLO, and TC. HELLO messages are used for neighbor discovery and link sensing; these packets expire after one hop and are never forwarded. TC messages are used for network topology information diffusion; these packets are forwarded away from the originator to deliver topology information encapsulated into new OLSR packets at each hop.

The protocol can be extended with other message types to support new OLSR applications. A new OLSR application can deliver information to all the other OLSR nodes minimizing the traffic, even if only a subset of nodes are equipped with the new OLSR application. This is possible because unknown OLSR messages are processed according to the default forwarding algorithm. For example an OLSR node may want to advertise a subnet attached to one of its interfaces. The node floods an HNA (Host and Network Association) message to all the other nodes. A node capable of understanding the HNA application will add a route for the announced network in its routing table, while a node not aware of the HNA application will forward the HNA messages according to the default forwarding algorithm.

To extend the OLSR protocol we define the *mDNS message type* as in Figure 3.1.

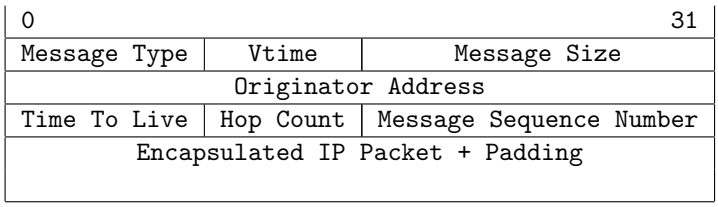


Figure 3.1: mDNS OLSR message

The key idea is to capture the IP packets containing the mDNS traffic[1] and encapsulate them in the payload of the mDNS OLSR messages. Because our payload is limited by the MTU of the wireless interface, we introduced the smallest possible overhead in the message header. We have 44 bytes and 76 bytes of total overhead when working respectively in IPv4 and in IPv6. This means that with a typical MTU of 1500 bytes our protocol can deliver IPv4 packets up to 1456 bytes and IPv6 packets up to 1424 bytes. However, mDNS packets are typically smaller and different mDNS OLSR messages may be aggregated in a single OLSR packet.

The transport protocol is defined as follows: when mDNS traffic is captured, it is encapsulated into a mDNS OLSR message and flooded into the network according to the default forwarding algorithm. The flooding is natively optimized by OLSR to avoid useless retransmissions. An OLSR mDNS-aware node receiving a mDNS OLSR message, decapsulates the IP packet and sends it on its non-OLSR attached interfaces.

3.3 Implementation of the OLSR mDNS Plugin

The mDNS OLSR extension is developed as a plugin for the UniK OLSR Implementation, also known as olsrd [20]. The source code is distributed under the BSD License, and is freely available in the official olsrd distribution [37].

Implementing the extension as a plugin, that is loaded upon configuration, is a natural choice, as it is not required for every node to support the mDNS application, but only a subset. For example, backbone-only nodes, with no end users directly connected, may avoid using the plug-in.

[1]It is easy to filter mDNS traffic because it is sent over the well known UDP port 5353

To enable the mDNS plugin the following block must be added to the olsrd configuration file:

```
LoadPlugin "olsrd_mdns.so.1.0.0"
{
PlParam     "NonOlsrIf"  "ethX"
}
```

Where one or more interfaces not participating in the OLSR network are specified.

The plugin has two main functions that are registered into the olsrd scheduler. After creating raw socket descriptors to sniff IP packets on the desired interfaces, these socket descriptors are passed under the control of the OLSR main scheduler. When a mDNS packet is captured, a mDNS OLSR message is generated on all the OLSR interfaces of the node.

The plugin also registers itself to the scheduler to receive incoming mDNS OLSR messages: upon reception the encapsulated IP packet is sent over all the non OLSR interfaces of the node, and the message is processed according to the default forwarding algorithm.

The implementation is IPv6 ready. The olsrd daemon works with both IPv4 or IPv6 (but does not support the two protocols simultaneously). The mDNS plugin is able to work with IPv4 or IPv6 packets depending on the current configuration. We proved the functionality of our implementation testing it both in virtual environment with Netkit [43] and in a real community network [2]. In future work we plan to test the plugin on other community networks with a higher number of users.

3.4 Chapter Conclusions

In this chapter we presented an extension to the OLSR protocol to deliver Multicast DNS traffic in a wireless mesh network. We make use of the optimized OLSR flooding mechanism to solve the problem of the limited broadcast and multicast domain. This new feature lets wireless community networks users take advantage of mDNS based service discovery tools already installed on their computers and supports the

deployment of services in a decentralized fashion. The protocol extension has been implemented and disseminated in the wireless network communities.

To reduce bandwidth consumption and for higher scalability, a future version of the protocol will compress the payload of the mDNS OLSR messages. This is mainly composed of ASCII text, therefore even with a lightweight compression it is possible to significantly reduce its size.

The work illustrated in this chapter has been presented at the IEEE SECON Workshops 2009 with an article entitled "The OLSR mDNS Extension for Service Discovery", co-authored by Francesco Saverio Proto.

Chapter 4

Multicast Multimedia Streaming in Wireless Mesh Networks

In Wireless Community Networks users can access both the Internet and internal network services. However the routers lack multicast support and users cannot exploit the high capacity of wireless links to stream and receive multimedia services. We designed and implemented an extension to the OLSR protocol to support the delivery of multicast traffic using the Overlay Borůvka-based Ad-hoc Multicast Protocol (OBAMP). This protocol enhancement makes possible to create a multicast distribution tree among a subset of nodes, providing the mesh users with multicast community services.

4.1 Overview

We described in Section 1.3 the state of the art of routing protocols that exists for wireless mesh networks, focusing especially on OLSR, widely employed in Wireless Community Networks.

We devise an hybrid approach where we mix an overlay protocol, to run the multicast protocol only on the subset of nodes willing to participate to the creation of the spanning tree, and the underlay protocol to share information about the quality of the links and the underlay network topology. In this way we get the benefit of the overlay approach without duplicating features as node discovery, link

sensing and metric calculations.

The goal of the work described in this chapter is to provide a simple tool to make possible for the users to run multicast streaming services over their WCN. We integrated the OBAMP overlay protocol [44] for multicast delivery in the OLSR routing protocol. The OLSR network is enhanced with the creation of an overlay multicast distribution tree for delivery of multimedia streaming.

We opted for an overlay solution because in a real WCN is not feasible to install a multicast protocol daemon on every router. WCNs are characterized by a highly decentralized administration and, consequently, network nodes have very different hardware capabilities. For this reason we choose to use an overlay protocol: to avoid the installation of dedicated software on all the routers of the network.

The work we present in this chapter, is not only a proposed solution for multicast support in mesh decentralized networks, but is also designed for use in real networks, as it is backward compatible with the already deployed nodes in the network and it is transparent to the end-users.

4.2 Related Work

Before our contribution the only way to have delivered broadcast or multicast traffic on a OLSR network, was to use the Basic Multicast Forwarding (BMF) plugin.

The Basic Multicast Forwarding Plugin floods IP-multicast and IP-local-broadcast traffic over an OLSRD network. It uses the Multi-Point Relays (MPRs) as identified by the OLSR protocol to optimize the flooding of multicast and local broadcast packets to all the hosts in the network. To prevent broadcast storms, a history of packets is kept; only packets that have not been seen in the past 3-6 seconds are forwarded.

The plugin has two main drawbacks. The first is that to properly work all routers on the network have to be upgraded to support the plugin. The second is that the plugin makes use of UDP tunnels with tun/tap devices and it is coded with the use of the libpthread library. This specific implementation details make the BMF plugin not very stable with some versions of OpenWRT because libpthread has known problems with the uclibs.

4.3 The OBAMP Protocol

OBAMP is an overlay protocol: it runs only on the hosts willing to participate to a multicast group. User data is distributed over a shared distribution tree formed by a set of non-cyclic UDP tunnels.

The use of an overlay protocol makes possible to add new features without re-deploying the routing software on all network nodes at once.

The characteristic of OBAMP is that the protocol first makes a mesh network with overlay links (UDP tunnels) between all the OBAMP nodes, and then it creates a distribution spanning tree over these mesh links. To limit signaling and improve the system scalability, OBAMP nodes do not build a full mesh among them, but create only the necessary links to keep the OBAMP overlay network connected.

Previous research [44] shows that the OBAMP protocol exhibits better scalability performance in a many to many communications scenario than two state-of-the-art protocols such as ALMA [45] and ODMRP [46]. In WCNs many to many communication is the default scenario, because network traffic is generated by the users.

4.4 Integrating OBAMP and OLSR

In this section we extend the OLSR routing protocol devising and implementing an OBAMP extension. The OBAMP protocol has been already implemented [47] as a standalone application. However, the integration with the underlay routing protocol leads to optimization in terms of signaling and efficiency. Functions like neighbor discovery and path-cost calculations are done at once for both the overlay and the underlay protocol. This leads to greater accuracy and lower CPU consumption, that is critical on embedded devices like wireless routers.

This version of the OBAMP protocol, implemented as an OLSR plugin, is simplified for use in WCNs, where we assume the nodes to be in fixed positions on the roof of the houses. Thus all protocol features regarding mobility have not been implemented.

A WCN is usually composed by an OLSR backbone and subnets, attached to the OLSR routers, where the users are connected. Implementing OBAMP on the

OLSR routers, and thus making the multicast delivery service transparent to the user, avoids the deployment problems that would be introduced by the installation of dedicated software on end user terminals.

To extend the OLSR protocol we define the *OBAMP alive* message as in Figure 4.1.

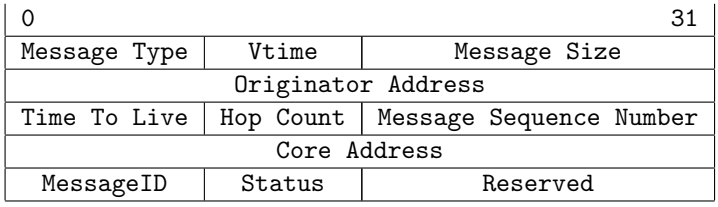


Figure 4.1: OBAMP alive OLSR message

OBAMP-enabled OLSR nodes (*OBAMP nodes* from now on) generate periodically *OBAMP alive* messages. These are forwarded in the whole network (also by the nodes that are not OBAMP-enabled thanks to the OLSR default forwarding algorithm). Because of the flooding mechanism every OBAMP node has a complete list of all the other OBAMP nodes in the mesh network. The OBAMP nodes discover each other exploiting the OLSR signaling, to later start their own unicast signaling for the OBAMP protocol to create the overlay distribution tree.

The OBAMP network must have a node called *Core*, that starts the TREE_CREATE procedure for the creation of the multicast tree. The node with the smallest IP address is designated as the Core node. After the discovery phase the rest of the signaling between OBAMP nodes happens in unicast, on a dedicated signaling port without interfering with the OLSR protocol.

4.5 Implementation

The OBAMP OLSR extension is developed as a plugin for *olsrd* [20]. The source code is distributed open source, and is freely available in the official olsrd distribution [37].

The natural choice for the OBAMP OLSR extension was to implement it as a plugin as it is not required for every node to support the overlay OBAMP protocol,

but only a subset[1].

4.6 Operation Description

Each OBAMP node should have at least one interface participating in the OLSR network and at least one interface not participating in the OLSR network (*non-OLSR interface*), to which user hosts are supposed to be attached.

The key idea is to capture the IP packets containing the multicast traffic and encapsulate them in UDP tunnels for delivery on the overlay distribution tree. These messages, in order to avoid duplicate packets, are identified by a sequence number and by the IP address of the OBAMP source node where the traffic has been generated and hence encapsulated.

More specifically, the transport protocol is defined as follows: when a multicast packet is captured on a non-OLSR interface of an OBAMP node, it is encapsulated in the payload of an *OBAMP data* message and forwarded into the overlay multicast distribution tree. The other OBAMP nodes receiving the *OBAMP data* message will forward the data on the non-cyclic overlay spanning tree and will also decapsulate the contained IP packet and then send it through all their non-OLSR attached interfaces.

4.7 Implementation Details

The plugin uses the olsrd scheduler to control the raw sockets descriptors to sniff multicast IP packets on the non-OLSR interfaces.

The OLSR scheduler has a default polling rate that is too slow for OBAMP to function correctly. However this is a tunable parameter (through the configuration file), and we found an optimal setting at 1 ms, that leads to an acceptable CPU consumption, even on embedded devices.

The plugin also registers itself to the scheduler to receive incoming *OBAMP alive* OLSR messages to discover the other OBAMP nodes. The rest of the signaling is

[1]For example, backbone-only nodes, with no end users directly connected, may avoid using the plugin.

received in unicast on the UDP port 6226.

The following timers are defined:

- OBAMP_alive_timer: every OBAMP node sends alive messages to advertise its presence to the other OBAMP nodes in the network. In the alive message every nodes states its IP address, and if it has already a tree link or not (we will see later this information is important for the outer tree create procedure). Note that if the list of known OBAMP nodes changes, the Core Election procedure is called to check if the CoreNode has changed.

- mesh_create_timer: every OBAMP node every OBAMP_MESH_CREATE_IVAL seconds evaluates the distance from the other OBAMP nodes and selects a subset of nodes to keep mesh links with. To select its overlay neighbors, an OBAMP node first calculates the ETX distance from the nearest OBAMP nodes, and then creates overlay mesh links with every node whose distance is in the range ($minETX$,$minETX + 1$). Note that to reduce signaling and to increase scalability, the overlay mesh links are setup only with a subset of the nearest OBAMP nodes.

- tree_create_timer: the Core node of the network every OBAMP_TREE_CREATE_IVAL seconds sends a message called TREE_CREATE on its mesh links. The creation of the spanning tree is very similar to what happens in the spanning tree protocol [48]. When a TREE_CREATE message is received, an OBAMP node enables a tree link with its parent and forwards the TREE_CREATE message on the other mesh links. TREE_CREATE messages are generated only by the Core and are numbered, so TREE_CREATE messages received over loops can be discarded.

- outer_tree_create_timer: the mesh_create algorithm may create a cluster of OBAMP nodes within the network that are disconnected between each other (see mesh_create_timer above). This happens if there are groups OBAMP nodes that are far from each other. In this case only the cluster where the Core is present will receive the TREE_CREATE and will receive traffic from the distribution tree. To overcome this problem, if in a cluster there are not TREE_CREATE messages in OBAMP_TREE_CREATE_IVAL seconds, the

node with the smallest IP in the cluster will make a long mesh link with the nearest node that has at least a tree link. All the necessary information to perform this procedure is diffused in the OBAMP_ALIVE messages.

- purge_nodes_timer: checks expire time of various variables, and deletes nodes or tree links in a soft state fashion

4.8 Plugin Configuration

To enable the OBAMP plugin the following block must be added to the olsrd configuration file:

```
LoadPlugin "olsrd_obamp.so.1.0.0"
{
PlParam     "NonOlsrIf"  "eth1"
}
```

The only parameter the user has to specify are the interfaces, not participating to the OLSR network, where she wants to capture and decapsulate multicast traffic.

4.9 Chapter Conclusions

In this chapter we presented an extension to the OLSR protocol and its implementation, to deliver multicast traffic in Wireless Community Networks. We make use of the OLSR flooding mechanism to integrate the underlay and the overlay protocols. As a result we obtain improved efficiency and reduced signaling. We capture the multicast traffic at the wireless routers to avoid the installation of dedicated software on the end user terminals. The protocol extension has been implemented and then tested and disseminated in the Wireless Community Networks.

The work illustrated in this chapter has been presented at IEEE WoWMoM 2010 with an article entitled “Implementation of the OBAMP overlay protocol for

multicast delivery in OLSR wireless community networks", co-authored by Francesco Saverio Proto.

Chapter 5

Cross-Layer Scalable Video streaming in WLANs

Wireless Local Area Network (WLANs), usually in the form of "hot spots" are becoming increasingly common in public spaces around the World, and Wireless Community Networks are no exception, as the access to end-users is usually provided by means of wireless Access Points. Moreover, as video streaming services popularity and availability increases, the need arises for resource optimization aimed at the maximization of the delivered video quality. For this purpose, H.264 Scalable Video Coding (SVC) emerges as a promising encoding technique that allows to adapt to network conditions.

In this chapter we will outline the scalable video streaming over WLANs scenario, design a simple application-layer packet scheduler and present the related results, achieved by using SVEF, a framework, developed and released by us, for the evaluation of scalable video streams, on a real testbed. Then we will extend this scenario to the presence of uplink non-video streams, devise an analytical model and address the associated optimization problem, in order to build a practical cross-layer packet scheduler.

5.1 Overview

Streaming high quality videos on a lossy wireless network is a challenging task. Because network losses are arbitrary, the decoder should be resilient to data loss, to produce a video output when some of the original information is missing.

In this scenario, the features and the design of H.264 Scalable Video Coding (SVC) [49] make it especially appealing for the implementation of cross-layer and application-aware access points. With attention to network schedulers, the key idea is to implement QoS classifiers, able to easily inspect packet payloads and queues (and eventually drop) packets using a policy that takes into account the properties of the video stream, simplifying the work of concealing at the decoding stage.

An SVC stream is composed of multiple "layers" which carry incremental video enhancement information. As such, it can be adapted to a capacity throttling or fluctuation simply by dropping (in part or in full) one or more enhancement layers, thus obviating the need for costly transcoding operations or the provision of multiple streams at different quality levels. H.264 SVC provides a very smooth adaptation process at a level of granularity down to individual dropping decisions for single video layer protocol data units (also called Network Abstraction Layer Units or NALUs in H.264 notation). An Overview of H.264 SVC can be found below, in Section 5.2.

In Section 5.3, we perform an experimental assessment on the use of H.264 Scalable Video Coding with an application-aware packet scheduler. This assessment is among one of the first works to document experimental results for application-aware H.264 Scalable Video Coding (SVC) support over IEEE 802.11 Wireless LANs. This is achieved by introducing a bandwidth throttling device, called Virtual BottleNeck (VBN), before the WLAN Access Point. Throttling is set to a bandwidth slightly smaller than the actual WLAN capacity (either known or estimated), so that all packet/frame losses occur inside the VBN. Here, loss events are controlled by a scheduling mechanism devised to operate with information taken from the H.264 Network Abstraction Layer Units (NALUs). Despite its relative simplicity, the implemented scheduler exhibits effective video adaptation performance and close to optimal bandwidth efficiency.

Setting up the trial was not trivial due to the lack of suitable publicly available

tools. Section 5.4 illustrates the evaluation framework called SVEF (Scalable Video Evaluation Framework) that we have developed for this purpose.

SVEF is the first open-source framework for experimental assessment of H.264 Scalable Video Coding (SVC) delivery over real networks. In this research area very little experimental work has been performed due to the unavailability of real-time H.264 SVC players, the limitations of existing decoding software libraries when challenged with network-impaired received SVC streams (e.g., affected by random loss of NALUs), and the lack of solutions for SVC streaming support. SVEF overcomes these issues by developing missing components and by integrating them in a hybrid online/offline experimental framework. With SVEF, SVC video traffic is delivered over the chosen networks using a custom simplified RTP *Streaming Server*, and can be further processed by local or in-network adaptation modules. Received video traffic is buffered in raw form until the completion of the experiment, and then post-processed offline through a set of custom software scripts. As main output, they deliver the same uncompressed video that would have been displayed by an online video player. The scripts include an offline *Filter* to check for NALU decoding dependencies and playout delay, a specially devised approach to extract and decode the resulting video stream, and an offline *Filler* providing a basic form of concealment of missing video frames.

Finally, Section 5.5 generalizes the application-aware approach to address the issue of delivering scalable video over Wireless LANs in presence of uplink traffic. A major contribution consists in the formalization of the problem, taking into account the unique characteristics of the WLAN MAC operation.

5.2 H.264 Scalable Video Coding

Scalable Video Coding (SVC) is a very promising encoding technique that allows to adapt to variable network conditions [50]. Its basic concepts have been investigated by the research community for almost two decades, and its exploitation sped up by the finalization, in 2007, of an SVC specification in the framework of the ITU H.264 advanced video coding standards family [49].

An H.264 SVC stream is defined as a sequence of NALUs. A NALU is composed

of a header and a payload carrying, partially or entirely, an encoded video frame. The NALU header [49, 51] contains information about the type of data and its relevance in the decoding process. From the information reported in the NALU header, we are specifically interested in three parameters: dependency_id (DID), temporal_id (TID), and quality_id (QID). Each parameter determines a specific scalability feature. DID allows *Coarse Grain Scalability*, TID allows *Temporal Scalability*, and QID allows *Medium Grain Scalability*.

Coarse Grain Scalability (CGS) provides the ability to *coarsely* adapt video properties, e.g., the video's spatial resolution from CIF to 4CIF. The video should be encoded with a suitable set of coarse enhancement sub-streams, called *dependency-layers*. The DID parameter identifies the dependency-layer to which a NALU belongs. The decoding of a NALU having $did > 0$ depends on NALUs belonging to the dependency-layer $did - 1$, and having the same value for the TID and QID parameters. Following this dependency rule, we can coarsely reduce video quality by removing NALUs with a DID greater than a specific value. For simplicity's sake, we do not consider Coarse Grain Scalability in the remainder of this work. However, extending our work to CGS is straightforward.

Temporal Scalability allows to adapt the video frame-rate. The TID specifies the *temporal-layer* of the NALU, i.e., the "frame-rate sub-stream". A NALU belonging to the temporal-layer $tid > 0$ and with $qid = 0$ depends on NALUs of temporal layer $tid - 1$, with the same DID and QID parameters. Following this rule, a frame-rate scaling may be accomplished by removing NALUs with a TID greater than a specific value.

Medium Grain Scalability (also known as *progressive refinement*) allows the adaptation of video quality. The video should be encoded with a set of quality enhancement sub-streams, called *quality-layers*. Adding a quality layer reduces the encoding quantization error, and thus improves the PSNR. The QID parameter identifies the quality layer to which a NALU belongs. A NALU belonging to quality layer $qid > 0$ depends on NALUs belonging to quality layer $qid - 1$, having the same DID and TID parameters. Following this dependency rule, quality scaling may be achieved by removing NALUs with a QID greater than a specific value.

For our purposes, it is essential to drop excess NALUs so that decoding dependencies are respected. Restricting our attention to the temporal and medium

grain scalability only (for simplicity, in our experimental results we have not considered spatial scalability), the following decoding dependencies hold (the arrow means "depends on"):

$$\begin{array}{lllll} (tid > 0, & qid = 0) & \rightarrow & (tid - 1, & qid = 0) \\ (tid \geq 0, & qid > 0) & \rightarrow & (tid, & qid - 1) \end{array}$$

The first rule states that temporal dependencies are enforced only on the quality layer 0, i.e., that a NALU belonging to the temporal-layer $tid > 0$ and with $qid = 0$ depends on NALUs of temporal-layer $tid - 1$, again with $qid = 0$. The second rule states that quality improvements are progressively applied to a considered temporal layer, i.e., a NALU belonging to the quality-layer $qid > 0$ depends on NALUs of quality-layer $qid - 1$, with the same DID and TID. These rules are graphically highlighted in the left drawing in Figure 5.2.

5.3 H.264 Scalable Video Streaming over WLANs

Adaptation of the video stream to the available network capacity should be done in an *application-aware* fashion. Dropping NALUs of a lower layer may make corresponding higher layer NALUs useless because of missing decoding dependencies. Many SVC adaptation solutions have been proposed [51, 52, 53, 54, 55, 56]. These are either implemented at the remote video server, or deployed within the network (e.g. at middleboxes such as proxies or at wireless base stations/access points). The adaptation mechanism requires feedback about the available capacity, which can be provided with lower overhead and in a more timely manner, the closer to the capacity bottleneck the adaptation occurs. For example, it is possible to leverage locally available detailed channel state information for adaptation at base stations or access points.

Wireless networks are usually characterized by a network capacity that may significantly vary over time. Such bandwidth fluctuations may be caused by the arrival and departure of traffic sessions competing for access to the shared medium, as in the case of a Wireless LAN [57]. Moreover, channel quality variations may trigger physical rate adaptation mechanisms [58, 59], which cause frequent and stepwise abrupt changes of the available capacity. Most importantly, random loss of

packets is frequently due to network congestion and wireless channel impairments [60], and further affects the available rate.

Despite the large interest in H.264 SVC adaptation and numerous proposed approaches, at the time of this work little experimental work had been carried out on real network testbeds, especially when random NALU losses may be encountered. We believe the reasons for this are twofold.

The first relates to the fact that JSVM [61], the existing reference open source software for H.264 SVC coding/decoding released and maintained by the MPEG/ITU Joint Video Team, in the version available at the time of this work, i.e. version (9.15) is not able to decode video streams affected by out of order, corrupted, or missing NALUs. However, these issues frequently occur when transmitting SVC streams over unreliable wireless channels.

The second reason is the lack of freely available H.264 SVC streaming servers as well as clients (solutions such as those provided in [62] are restricted). For the case of servers, it is worth to remark that, at the time of this work, the support of H.264 SVC over the widely used Real-time Transmission Protocol (RTP) was still work in progress [63]. This may be a reason why, to the best of our knowledge, no public domain software appears available.

5.3.1 WLAN Scenario

The approach proposed for cross-layer support of H.264 SVC delivery over WLANs is sketched in Figure 5.1. In this testbed we restrict our investigation to the case of "downlink video streaming". This is representative of a video-on-demand scenario, where end users connected to a WLAN hot-spot independently access one or more video servers placed in the wired network.

5.3.2 Virtual BottleNeck

The idea behind the Virtual BottleNeck (VBN) illustrated in Figure 5.1 is very simple, but practical and effective. It emerges from the observation that MAC-layer frame losses only rarely occur in a Wireless LAN because of channel quality impairments. In fact, starting from Auto Rate Fallback [58], several rate adaptation

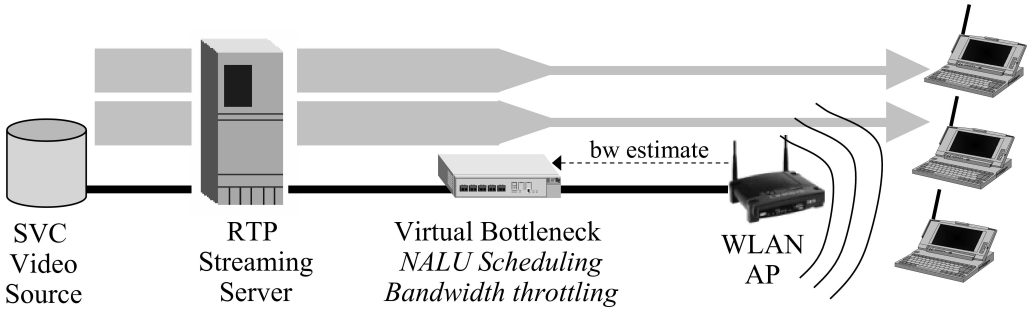


Figure 5.1: Network scenario with video server, VBN, WLAN AP, etc.

mechanisms [64, 65, 59] have been proposed to improve frame delivery, by estimating the channel quality and/or measuring the experienced frame loss ratio, and then switching to a suitable modulation scheme. The 802.11 MAC function retransmits MAC frames corrupted because of channel errors or channel access collisions. As a result, a MAC frame is lost completely only if it reaches a maximum number of retransmissions. In the 802.11 standard, this is a relatively large value (the default settings being 4 – Short Retry Limit – and 7 – Long Retry Limit – retransmissions, depending on the length of the MAC frame [57]). Therefore, in normal conditions, the MAC frame loss ratio seen by higher layers is typically low. It only becomes significant if severe channel degradation occurs, so harshly that even rate adaptation to the minimal available transmission rate is not sufficient.

We can thus assume that the majority of all MAC frame losses occur at the AP buffer. Loss events clearly occur when the load offered to the AP is greater than the maximum throughput available at the AP. In general, the time-varying capacity $C_{AP}(t)$ depends on i) the number of stations competing with the AP for channel access and ii) the individual transmission rates of *all* competing stations [66].

The Virtual BottleNeck is a traffic control box placed in the wired network before the AP. It intercepts all the traffic offered to the AP itself. Its goal is to enforce a traffic throttling function devised to prevent the traffic offered to the AP from overflowing the available capacity $C_{AP}(t)$. Provided that the throttling function is able to follow the variations in time of the AP capacity, and provided that a sufficient AP buffering capability is available and a sufficient bandwidth margin is deployed between the traffic offered by the VBN and the actual AP capacity, the ultimate result is that the AP buffer will never saturate, and hence no frame loss will occur

at the AP itself. Rather, all the losses will occur inside the VBN box.

Several mechanisms exist for the run-time estimation of the available AP capacity and the consequent dynamic control of the throttling function, e.g., [67, 68, 69, 70]. However, the details of this estimation are outside the scope of this thesis. Here, we are interested in taking full advantage of the VBN in cross-layer scheduling traffic; i.e., exploiting application layer information.

We remark that since the VBN is a separate control entity, it can easily be deployed in any pre-existing WLAN infrastructure with legacy Access Points. If the WLAN supports 802.11e Quality of Service enhancements (as is the case in our experimental set-up), these can be exploited by configuring the VBN to set the IP Type Of Service (TOS) field to the value 160 (for WMM - Wireless Multimedia - compliant APs) so that MAC frame transmission occurs with EDCA *video access category*.

5.3.3 Cross-Layer VBN Scheduling for H.264 SVC Traffic

The proposed cross-layer scheduling algorithm operates at the network layer. Its service policy is based on the control information contained in the header of the H.264 SVC Network Abstraction Layer Units (NALUs).

The design target of our proposed cross-layer scheduler is to exploit the control information contained in the NALUs to

1. accomplish an efficient usage of the wireless resource by avoiding to transfer NALUs that will not be decoded by the receiver because of missing dependencies;

2. provide a smooth adaptation of the video quality versus changes in the available capacity $C_{AP}(t)$ or the offered load of the video traffic.

These two goals can be accomplished through a priority queuing discipline, dedicating a *separate* queue to each possible TID-QID combination. Considering that the default range for TID values is from 0 to 4, and considering two additional enhancement quality-layers (i.e., QID values in the range from 0 to 2), we deploy $5 \times 3 = 15$ limited-size queues, with queue #0 having the highest priority and queue #14 having the lowest one, as shown in Figure 5.2. An incoming NALU is delivered

to a queue #n according to the following classification rule, schematized also in the figure:

$$n = 5qid + tid$$

This ensures that a NALU x will have a lower service priority than the NALUs x depends upon (first goal). Anyway, it may exceptionally happen that at the receiver side, a delivered NALU lacks other NALUs it depends on, since the dropping decision is taken locally at each queue. For instance, a NALU with a given tid and $qid = x$ may be dropped from its queue due to a peak load fluctuation, while the lower priority queue associated to the same tid but $qid = x + 1$ may not experience the same fluctuation.

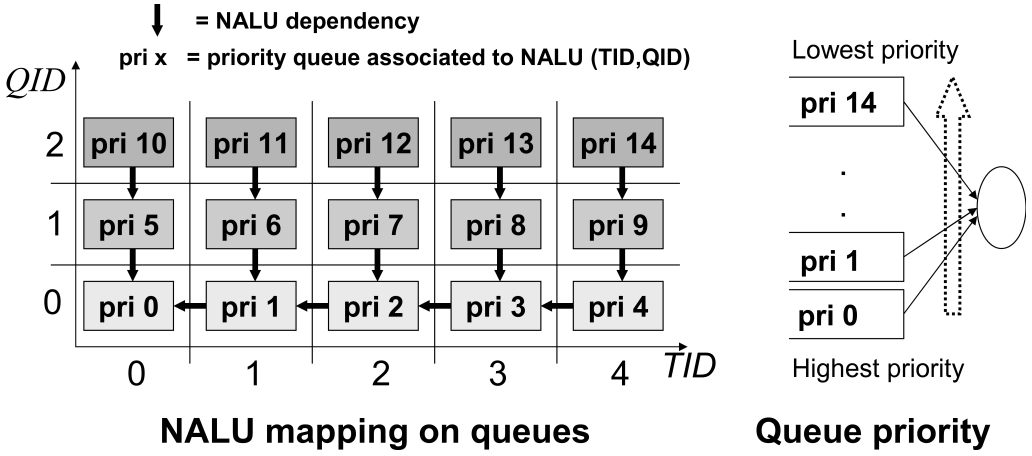


Figure 5.2: Mapping of SVC substreams to priority queues

Finally, we observe that in the presence of congestion, the NALUs of the higher quality layers will be discarded first, and only later the NALUs of the base layer (second goal).

5.3.4 Experimental Results

To cope with the unavailability of essential software components, we make use of an hybrid online/offline testbed that makes use of the SVEF framework. While the NALUs are delivered in real-time, several pre-processing and post-processing mechanisms can only be applied off-line. The result is a testbed, which is functionally equivalent to a purely online video delivery process. The SVEF Framework is

illustrated in detail in Section 5.4

Results are provided for the following setup. We use a 10 second clip of a publicly available 4CIF YUV video (soccer game) at 30 fps. A 50 second video sequence is generated by concatenating 5 repetitions of the video. Through JSVM, we encode the 50 seconds video with three different approaches that are detailed in Table 5.1. The SVC (A) and SVC (B) are encoded with Medium Grain Scalability and have three quality-layers (base-layer BL and two enhancement layers MG1 and MG2). SVC (B) has a larger base layer than (A), and the enhancement layers reduce in bitrate as the QID increases. The opposite behavior is chosen for (A). Finally, the AVC video only uses a base layer. All the encoded videos have approximately the same average bitrate, similar bitrate fluctuations (about $\pm$ 30%) and the same client-side playout buffer of 5 seconds. For the experiments, we assume that a first user starts retrieving the video stream at time 0. A new user arrives every 8 seconds (240 frames) and begins the downloading of the same video stream. Video performances are measured for the first user.

	BL (kbps)	MG1 (kbps)	MG2 (kbps)	Full (kbps)	PSNR (dB)
SVC(A)	648	907.3	1304.7	2860	36.64
SVC(B)	1295	815	637	2748	36.50
AVC	2693	-	-	2693	36.49

Table 5.1: Video test-sequence parameters

The experiments are based on an indoor WLAN deployment with 5 stations associated to an AP. All stations experience good average channel conditions (the distance to the AP is less than 2 meters in LOS conditions) and support the maximum 11 Mbps 802.11b physical layer rate with no losses. The VBN has been throttled to 6.0 Mbps, a value just below the measured MAC throughput at the AP of about 6.3 Mbps. This guarantees, as we confirmed in subsequent measurements, that no MAC frames are lost at the AP buffer. Moreover, we also perform tests with the WLAN physical layer rate reduced to 2 Mbps; in this cases the VBN is throttled to 1.5 Mbps.

For the evaluation, results are reported in terms of a video quality metric (PSNR)

as well as a delivery efficiency metric. The video streams (note: 600 MB per video sequence) are also made available for download[1].

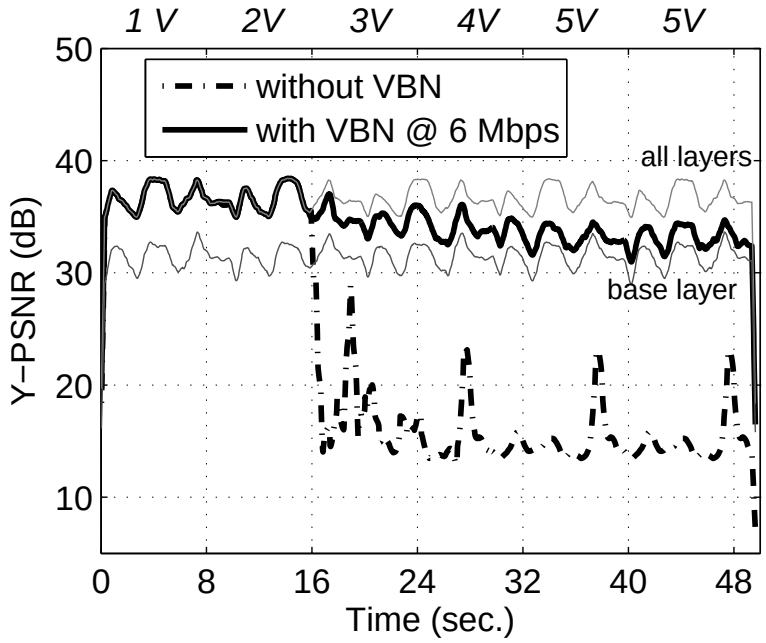


Figure 5.3: SVC (A) with/without scheduler and WLAN @ 11 Mbps

5.3.4.0.10 Impact of the VBN Figure 5.3 shows the Y-PSNR (luminance) over time, measured for the video stream SVC (A) delivered to the first user with and without VBN scheduling, with respect to the original, pre-encoding raw video. The PSNR is compared to two reference curves: i) the ideal PSNR (top curve labeled "all layers") of the stream for the case of no NALU loss, where the resulting PSNR depends only on the degradation due to the encoding process, and ii) the PSNR provided by the base layer only (labeled "base layer"), assuming that all base layer NALUs are received and all NALUs of other layers are dropped.

Figure 5.3 confirms that *the delivery performance of H.264 SVC is poor without cross-layer scheduling enforced by the VBN, i.e., when MAC frames, and as a consequence NALUs, are dropped randomly.* A sudden severe PSNR degradation occurs under overload conditions. The resulting video frequently "freezes" (meaning that several video frames were lost), and the overall video quality is unacceptable. With

[1]URL: `http://netgroup.uniroma2.it/iwcld09`

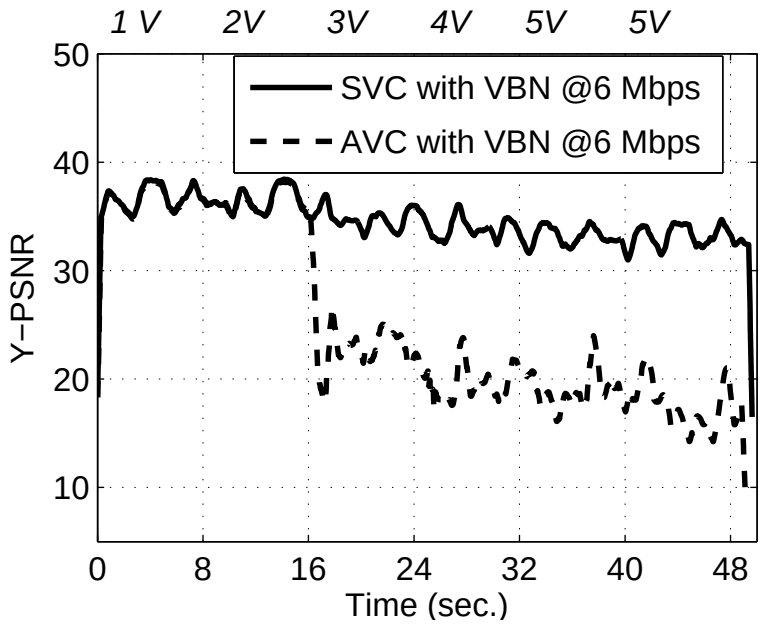


Figure 5.4: SVC (A) versus AVC with scheduler and WLAN @ 11 Mbps

an average total video rate of 2.86 Mbps, this happens when three streams are delivered. The PSNR does not degrade further when additional streams are admitted. This is due to the fact that the PSNR given by the comparison of two random frames from the same test sequence is around 15 dB, as confirmed by further experiments (not shown here). Thus, this is the lowest PSNR value we can expect.

Conversely, the cross-layer scheduler allows for a smooth degradation of the H.264 SVC stream. When all 5 users share the channel, they achieve an average rate of 700 kbps per user. The PSNR approaches that of the base layer alone, which is the expected behavior, given that the base layer uses on average 650 kbps.

5.3.4.0.11 H.264 SVC versus AVC Figure 5.4 shows the performance advantages of SVC (A) compared to AVC. In both cases, frame losses are controlled through the VBN scheduling. Again, the results confirm that *SVC is a much more suitable coding mechanism in a scenario where large variations in the available capacity occur*. When three or more stations compete and an overload emerges, SVC reduces the quality of the video, which results in a significantly smoother PSNR degradation compared to AVC's temporal scalability adaptation.

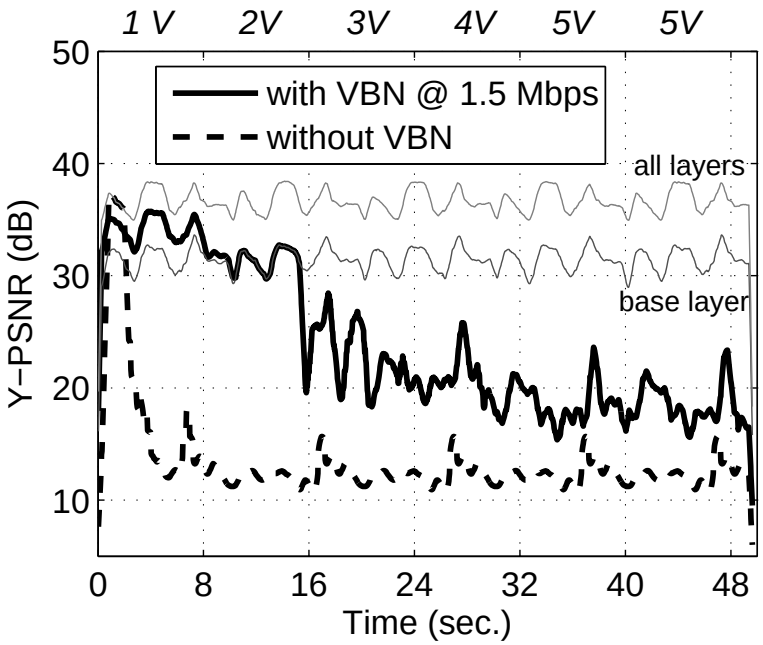


Figure 5.5: SVC (A) with/without scheduler and WLAN @ 2 Mbps

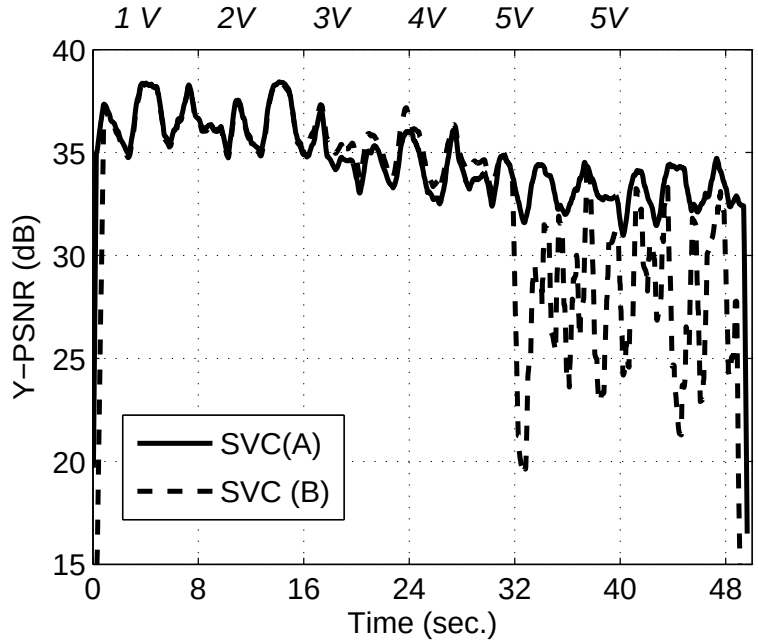


Figure 5.6: SVC (A) and SVC (B) with scheduler and WLAN @ 11 Mbps

While somewhat obvious, this consideration has important practical implications on how to encode H.264 SVC streams when they are delivered over a WLAN.

Recalling that the H.264 SVC base layer is AVC encoded, we expect that under severe overload conditions where it is impossible to transmit the base layer without NALU losses, the quality degradation will become significant. This can be seen from Figure 5.5. The setup is the same as that of Figure 5.3, with the fundamental difference that the WLAN is configured to provide only a 2 Mbps PHY rate and the VBN rate is set to 1.5 Mbps. The key difference between Figure 5.5 and Figure 5.3 is that video stream scaling occurs immediately. The available bandwidth is lower than the total bandwidth requirement for all the layers and, much more importantly, when three or more streams compete, adaptation is required also for the base layer. As a consequence, performance drops as in the AVC case (although not nearly as dramatically as in the case of no cross-layer scheduling).

This insight is especially significant for the following reason. In Figure 5.6, we run the experiments for two different encoding choices: SVC (A) and SVC (B). The figure clearly highlights that it is preferable to reduce the size of the base-layer, especially if this produces only a marginal degradation of the overall video encoding efficiency (as in our experiments). While for SVC (A), degradation affects only the enhancement layer NALUs, SVC (B) experiences base-layer NALU losses and the resulting AVC temporal scalability reduces PSNR much more severely.

5.3.4.0.12 Scheduler Impact and Performance In Table 5.2, we provide some summarizing results on delivery efficiency for the previous experiments. In addition, the last row shows the performance experienced by an AVC stream for the scenario of Figure 5.5 with the VBN throttled to 1.5 Mbps.

The table reports three performance metrics: *transmission efficiency*, defined as the percentage of NALUs received by the client which can be used for decoding (i.e., for which encoding dependencies and playout delay conditions are satisfied), the total number of video frames that could not be decoded (out of the 1490 frames transmitted), and the average PSNR over the whole 50 seconds of the experiments.

The most interesting result provided in the table is the transmission efficiency of the considered scheduler. Though the scheduler does not guarantee that all NALU dependencies will ultimately be satisfied, the performance for both AVC and SVC cases with VBN is close to 100% efficiency. Without the scheduler, transfer efficiency is always very poor.

Video Type	Scenario (VBN, WLAN rate)	TX Efficiency	# Missing Frames	Average PSNR
SVC (A)	6Mbps, 11Mbps	100.00 %	0	34.67
SVC (A)	no VBN, 11Mbps	54.64 %	941	22.52
AVC	6Mbps, 11Mbps	99.92 %	845	25.25
SVC (B)	6Mbps, 11Mbps	98.39 %	121	32.75
SVC (A)	1.5Mbps, 2Mbps	100.00 %	799	24.02
SVC (A)	no VBN, 2Mbps	21.33 %	1424	13.51
AVC	1.5Mbps, 2Mbps (not shown in the fig.)	99.39 %	1391	16.37

Table 5.2: Performance Summary

The column reporting the number of missing video frames gives an impression of the perceived quality of the final video stream (again, please refer to the web site at `http://netgroup.uniroma2.it/iwcld09` for visual comparison of the actual streams). In the 6 Mbps scenario, SVC yields 0 missing frames, compared to the 845 misses of the AVC. In the low rate 1.5 Mbps scenario, almost all frames are missing for AVC and SVC without scheduler, while a reasonable quality is achieved in the SVC case with scheduler. Looking at the actual video stream, we see that frames are missing periodically and the SVC has scaled to operate at about the half of the frame rate.

5.4 The SVEF Evaluation Framework

The shortcomings encountered during the work described in Section 5.3 encouraged us to develop, and publicly release an open-source software framework SVEF, described in this section, aimed at the performance evaluation of H.264 scalable video streaming. SVEF uses JSVM [61] for H.264 SVC coding/decoding, but includes additional software to i) support H.264 SVC video streaming over IP, through encapsulation of NALUs in a simplified RTP structure, and ii) support receiver side decoding and reproduction of an SVC stream affected by arbitrary NALU losses and playout delay constraints.

The latter is done in an offline fashion, over a raw NALU trace received at the

client. The same uncompressed video that would have been displayed by a real video player client at the user side is, in SVEF, obtained through post-processing of the raw NALU trace, and subsequent application of i) a *Filter* for NALU decoding dependency and playout delay checking, ii) a custom method to extract and decode the resulting video stream, and iii) an offline *Filler* devised to provide a simple form of error concealment.

SVEF is meant to reproduce a distribution chain formed by three actors: streaming server, middlebox and receiver. All actors are connected by an IP network.

Figure 5.7 shows the structure of SVEF with interactions between single tools and data flows depicted as arrows. The software modules inherited from the JSVM package [61] are represented in gray. The whole process, from the encoding of the original video source to the evaluation after the streaming over a network can be summarized in four steps, better detailed in the following sub-sections:

1. A YUV (video is encoded in H.264 SVC format through the JSVM Encoder. The encoded video and its NALU-trace are transferred to the Streamer.

2. The encoded video is transmitted over the IP network by the Streamer, at a fixed frame-rate.

3. In presence of a middlebox, the video NALUs first enter a cross-layer scheduler and then the NALUs are forwarded to the receiver (for example through a wireless link).

4. The Receiver generates in real time a trace of the received NALUs. At the end of the streaming process, the received NALU trace is processed to produce a YUV file (filtered-YUV video) characterized by missing frames due to transmission losses, unsatisfied decoding dependencies or excessive delay. The filtered-YUV video is processed to achieve a simple error concealment, obtaining a final-YUV video with the same number of frames as the original video.

SVEF has been published as open source software at:
http://svef.netgroup.uniroma2.it.

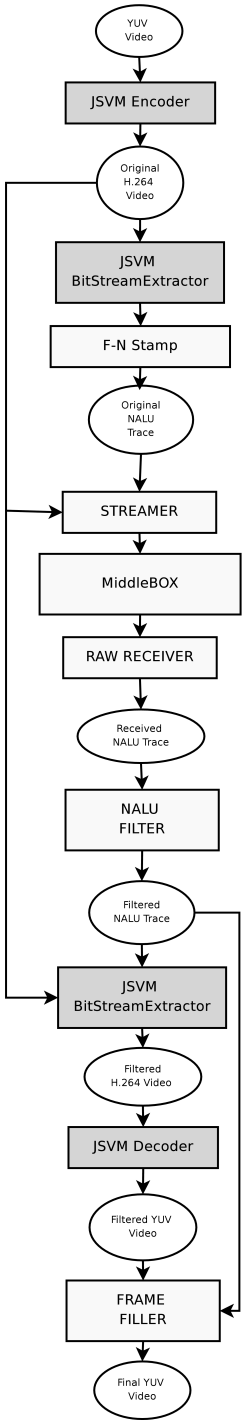


Figure 5.7: SVEF Software Chain

5.4.1 Video Encoding

We use raw video files stored in the standard YUV format. Video sources are encoded in the H.264 SVC format through the JSVM Encoder. Currently, the framework supports SVC with a single dependency layer and an arbitrary number of quality enhancement layers. From the resulting H.264 encoded video file, we generate an original NALU trace file through the JSVM BitStreamExtractor tool. This trace contains for each NALU the entry shown in Figure 5.8, where `mem-offset` represents the memory offset from the beginning of the encoded video file up to the current NALU, `NALU-size` represents the length of the current NALU and `Frame-Number` is the number of the video frame to which the NALU belongs. This latter parameter is not provided by JSVM BitStreamExtractor. For this purpose we developed the F-N Stamp module.

mem-offset	NALU-size	DID	TID	QID	Frame-Number

Figure 5.8: NALU-trace entry

5.4.2 The Streamer

The Streamer transfers the NALUs over an IP network by parsing the NALU trace and loading data from the H.264 file. For each entry in the NALU trace file, the streamer seeks and loads the corresponding NALU from the H.264 file. This NALU constitutes the payload of a custom layer-5 packet, whose header (see Figure 5.9) resembles the RTP [63] header, as both have the same size (including payload header). Hence the Streamer feeds the network with packets that have the same length as if a real RTP stack was used. Moreover, with respect to RTP, the custom layer-5 header does not add information that may improve the cross-layer scheduling process, but simply exploits the DID, TID, and QID parameters that are also contained in the RTP payload header. We choose to use a custom header, because we make use of the `mem-offset` information in the generation of the "filtered H.264 video file" described in section 5.4.4.

A layer-5 packet is entirely encapsulated in a UDP packet, which in turn is encapsulated in a set of IP packets. For the Streamer design, we had to choose where

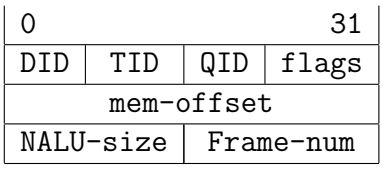


Figure 5.9: Layer-5 header

to fragment large NALUs to fit into network packets. During the video encoding process frames may be split in slices. Working with slices yields smaller NALUs and reduces the problem of fragmentation. However, we discarded this application-layer solution, since we want our evaluation framework to be independent of codec parameters.

The two remaining options were fragmentation at the IP layer or at the RTP layer. Fragmentation of IP or RTP packets is similar, with the additional limitation that UDP imposes a maximum payload size of 64 kbytes. Moreover, the NALU SVC header is considered as RTP payload and is thus carried only by the first IP fragment. This means that to perform a cross-layer NALU-based scheduling, a complete reconstruction of the NALU is necessary, both in case of IP or RTP fragmentation.

To limit the programming effort, we choose to fragment at the IP layer, as IP fragmentation/reassembly is a native feature of the Linux kernel. Thus, the Streamer cannot transfer NALUs with a length exceeding 64 kbytes. When some NALUs exceed this threshold, it becomes necessary to re-encode the video enabling the generation of two or more slices per frame.

Finally, we observe that the streamer transmits only NALUs of type “SliceData”, while the H.264 “ParameterSet” and “StreamHeader” NALUs are provided to the receiver off-line.

5.4.3 The Middlebox

The SVEF Middlebox is deployed upon a Linux system to perform cross-layer scheduling. Figure 5.10 shows the middlebox architecture that we have designed.

When the IP fragments reach the ingress interface, they are reassembled into whole IP packets. For this purpose, the IP_conntrack module of the Linux Kernel is

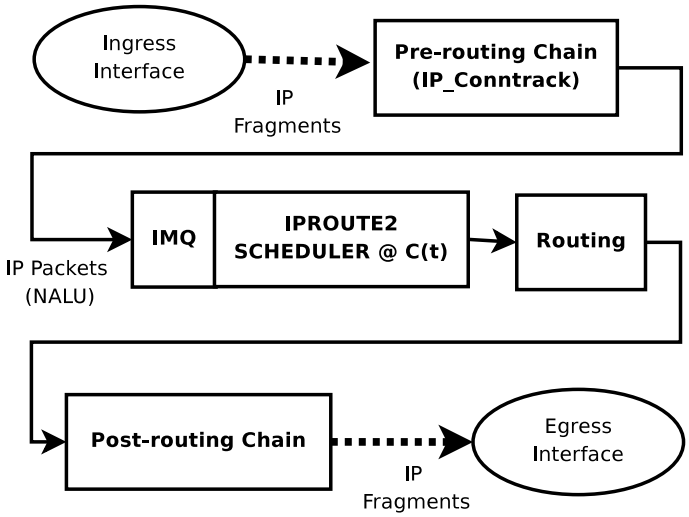


Figure 5.10: The Middlebox

used. When an IP packet (i.e., a NALU) is fully reconstructed, it is transferred to an Intermediate Queue device (IMQ) by means of an Iptables jump. On the IMQ egress we can enforce a custom scheduling policy, taking care that the overall output bandwidth C is equal (or smaller) to the one available on the following network path. Obviously, if this capacity varies over time, the scheduler has to be timely informed by an additional module. The scheduler can operate in a "cross-layer" fashion, since it is able to classify traffic according to the (DID, TID, QID) information contained in the RTP payload header, or in the custom layer-5 header. After exiting the scheduler, the IP packets return to the routing decision module and are fragmented again.

5.4.4 Receiver-side Tools

This section describes the tools used at the client side to reproduce a YUV video equivalent to the one that would have been displayed by an H.264 SVC video player. We expect the client to receive the NALUs and dejitter and order them in a play-out buffer, which synchronously provides NALUs to the decoder. The decoder is expected to appropriately discard the NALUs with unsatisfied dependencies (see

section 5.2) and also to conceal missing frames. We now describe how our framework reproduces such a chain of client-side operations based on three tools: NALU-Receiver, NALU-Filter and Frame-Filler.

The NALU-Receiver represents the network end-point. It decodes and writes in real time the layer-5 headers of the received packets, thus building a client-side (received) NALU trace file. Moreover, the NALU reception times are recorded through time-stamps.

The NALU trace file is then passed to the NALU-Filter tool that: i) reorders the NALUs according to the sending order, ii) removes NALUs received after the play-out buffer deadline and, iii) removes NALUs with unfulfilled decoding dependencies. The latter cannot be decoded and, moreover, are not handled properly by the current version (v 9.15) of the JSVM Decoder.

The NALU reordering is performed by the NALU Filter on the `mem-offset` field's basis.

To discard NALUs because of excessive delay, the NALU Filter computes a NALU's expected reception time from first NALU's reception time t_0 and frame-rate f as follows: $t_0 + f \times$`frame-number`. When the difference between the reception time and the expected time exceeds a specific play-out delay, the NALU is deleted from the NALU trace file.

After the removal of NALUs with excessive delay, the NALU Filter discards those NALUs for which the decoding dependencies described in section 5.2 are not satisfied (i.e., if a NALU y depends on NALU x and NALU x is not available in the NALU trace-file, then NALU y is deleted).

The resulting (filtered) NALU trace-file is used as a "map" pointing out which NALUs of the original H.264 video file are effectively decoded at the receiving side. We use this map and the original H.264 video file as input to the JSVM BitStreamExtractor tool to obtain a (filtered) H.264 video file. In essence, this is a NALU-subsampled version of the original video file, corresponding to what a hypothetical H.264 SVC client would have decoded and displayed in real time. Then this filtered H.264 video is handed to the JSVM Decoder, which generates a video in uncompressed YUV format.

The filtered YUV video has, in general, fewer frames than the original YUV video because of missing base-layer NALUs. In order to properly compute the

PSNR, the Frame-Filler has to conceal the missing frames by copying the previous frame. Missing frames are identified through the `frame-number` field of the filtered NALU trace.

5.4.5 Performance Parameters

Currently, SVEF measures the following performance parameters:

1. number of lost frames,
2. frame-by-frame PSNR, and
3. transmission-efficiency.

The number of lost frames is computed by the Frame-Filler, and the frame-by-frame PSNR is obtained using the JSVM PSNR tool, fed with the original YUV video and the error-concealed YUV video. We report below the definition of PSNR for frame number n:

$$PSNR(n)_{dB} = 20log_{10}\frac{V_{peak}}{\sqrt{\frac{1}{N_{col}N_{row}}\sum_{i=0}^{N_{col}}\sum_{j=0}^{N_{row}}[Y_S(n,i,j) - Y_D(n,i,j)]^2}}$$

If k is the number of bits per pixel (considering only the luminance component) we have $V_{peak} := 2^k - 1$. The part under the fraction line is the mean square error (MSE) computed from the luminance components Y_S and Y_D of the source image S and of the destination image D.

The *transmission efficiency* (TE) is defined as the ratio of the number of NALUs received by the client which can be usefully decoded (NAL_{useful}) to the total number of received NALUs ($NAL_{received}$).

$$TE = \frac{NAL_{useful}}{NAL_{received}}$$

This measure represents the efficiency of the overall streaming process in exploiting the communication resources. A low transmission efficiency means that most of the received NALUs are useless, and thus their transmission only wasted communication resources. Without a scheduler, NALUs are lost at random and this leads to a considerable number of unsatisfied dependencies at the receiving side and a low transmission efficiency. Conversely, a well-devised cross-layer scheduler should substantially improve the transmission efficiency.

5.4.6 SVEF Experimental Flow

The typical SVEF experiment goes through the following steps:

1. We first encode a raw YUV video file into SVC using the *JSVM H264Encoder*.

2. From the resulting H.264 encoded video file we generate a packet trace file with the *JSVM BitStreamExtractor* tool. This file is used as an hint track by the video streamer (described below) for timing and packetization purposes.

3. We feed the H.264 video and the packet trace to a custom *video streamer* module, which constructs a simplified RTP header for each NALU. We map one NALU to each RTP unit and send RTP packets according to the video frame-rate. Large NALUs are thus split through IP layer fragmentation.[2]

4. IP fragments received at the VBN are reassembled using the standard IP_conntrack Linux Kernel facility. Then entire IP packets (i.e., NALUs) are sent to an internal virtual interface (an Intermediate Queuing Module of Linux) where we implement the cross-layer scheduling using the IPROUTE 2 tools. At the exit of this virtual-interface, IP packets are again fragmented and transferred to the output interface.

5. IP fragments are received by the WLAN AP, which transmits them to the end devices. At the receiver side, all the received NALUs are collected in an H.264 JSVM compatible trace file. We post-process the trace with a custom software module (*NALU-Filter*) devised to i) discard NALUs received after a pre-set maximum playout delay (5 seconds in our experiments) and ii) check NALU dependencies and discard the NALUs for which dependencies are missing.[3] Besides the filtered H.264 trace file, the NALU-Filter also returns the number

[2]The alternative would have been to use RTP fragmentation. This allows to deal with NALU sizes greater than 64 kbytes (the upper limit for UDP datagrams), but would require to i) implement RTP fragmentation/reassembly from scratch, and ii) to reassemble NALUs at the VBN before their scheduling. Unlike AVC, the RTP header field does not contain SVC information and the tuple (DID,TID,QID) is carried as RTP payload. Hence, it is available only on the first RTP fragment. In our experiments, the 64 kbytes upper limit for the NALU size was never reached with SVC, while for AVC we solved this issue by dividing each video frame in two slices.

[3]This latter check is strictly necessary since the JSVM H264Decoder (v9.15) hangs if a NALU dependency fails.

of received NALUs and the number of filtered NALUs. These are two basic metrics to evaluate if wireless resources are used efficiently. If no NALUs are filtered, no resources are wasted.

6. The filtered H.264 trace file is used to reconstruct an H.264 video, which is in turn decoded with the JSVM H264Decoder, thus obtaining an uncompressed YUV file.

7. This YUV file may contain missing frames, since relevant NALUs may have been lost. To maintain the original temporal sequence and to simplify PSNR measurements, we develop a simple *Frame-Filler* tool. It outputs a final YUV video that has the same number of frames as the original one. When a frame is missing, the Frame-Filler inserts the last available received frame instead, which is a very basic form of error concealment.

8. Finally, the resulting YUV file and the original one are compared with the *JSVM PSNR* evaluation tool to assess overall video quality.

5.5 H.264 Scalable Video Streaming over WLANs in Presence of Uplink Traffic

In this section we provide a theoretical formulation of a QoE utility-optimal cross-layer scheduling problem for H.264 SVC downlink delivery over WLANs. We show that, because of the unique characteristics of the WLAN MAC operation, this problem significantly differs from related approaches proposed for scheduled wireless technologies, especially when the WLAN carries background traffic in the uplink direction. From these theoretical insights, we derive, design, implement and experimentally assess a simple practical scheduling algorithm, whose performance is very close to the optimum.

5.5.1 Scenario

Wireless Local Area Networks [57] are characterized by capacity constraints which vary over time. Arrival and departure of traffic sessions competing for access to the

shared medium may cause huge fluctuations in the capacity available to the access point or to stations. Channel condition changes may trigger physical (PHY) rate adaptation mechanisms [58, 59] which yield frequent and abrupt step-wise changes of the available rate. Furthermore, because of the well known "performance anomaly" of the WLAN Medium Access Control operation [66], the actual throughput achieved by a station experiencing a good channel quality is severely affected by the PHY rate changes due to variations in the channel quality experienced by *other* stations.

In this section we extend our previous work to the case of the presence of background uplink traffic.

We address the problem of scheduling downlink video traffic in Wireless LAN hotspots in order to maximize the utility provided to the customers in terms of QoE. The section provides two main contributions:
1) to the best of our knowledge, ours appears to be the first work which recognizes that such a problem requires a novel formulation in the context of WLAN systems.
2) from the theoretical insights gained from the formulation of the model, we design, analyze, implement in a Linux Access Point, and experimentally assess, a *practical* (sub-optimal) scheduler whose performance is marginally lower than the optimal solution.

5.5.1.1 Problem Statement

An issue largely addressed in literature concerns the allocation of wireless channel resources so that the video quality is optimized with respect to some chosen performance metric. For adaptive video streams, the common approach is to rely on (Quality of Experience, QoE) utility curves, expressing the user-perceived quality which model the application utility (e.g. expressed in terms of Mean Opinion Score (MOS), or Peak Signal to Noise Ratio (PSNR)) versus the network resources committed to that stream. When considering a fixed capacity network, the resulting optimization problem becomes straightforward. It suffices to distribute a known and constant pool of resources, namely the overall capacity, to the different streams so that the resulting overall utility is, for instance, maximized. Many papers have generalized this problem to the multi-rate case, most notably in the context of scheduled technologies such as 802.16, 3G/LTE, etc. In this case, there is no notion of

"total" wireless network capacity, as the PHY rate of the wireless terminals depends on the channel conditions and the specific modulation and coding techniques employed. However, the optimization problem can be reduced to that of allocating a known and constant pool of resources, by considering the available channel time, or equivalently the PHY symbol rate, instead of capacity.

At a first glance, the WLAN hot-spot scenario comprising downlink video streaming appears very similar. In fact, the problem of allocating resources to the different video streams becomes a centralized scheduling problem, as the AP may take appropriate decisions on how to allocate its transmission capacity to the connected users. A closer look reveals, however, that the case differs substantially from the previous ones, especially when the AP is not the only active station in the network. When two or more stations are accessing the network, channel resources are managed in terms of transmission opportunities, rather than in terms of capacity or channel time. Thus the utility optimization problem cannot be expressed as the sharing of a given amount of capacity. Instead, it is necessary to take into account that, in a multi-rate scenario, the channel time dedicated to the video streams transmitted by the AP depends upon the scheduling decisions taken by the AP itself. Although the consequences of the MAC layer's sharing of transmission opportunities rather than time is well known and understood, to the best of our knowledge no prior work appears to relate these insights to the utility optimization problem discussed above.

5.5.1.2 Related Work

A substantial amount of prior work applies SVC coding to video transmission over wireless networks. An high level framework and the general challenges of adaptive (scalable) video streaming in a wireless context are presented in [71]. Three techniques for video content delivery in such scenarios are identified: scalable video representation, an end-system capable of performing network aware adaptation (end-to-end approach), and adaptive QoS support from the network.

The representation of scalable video concerns the encoding of the video into different substreams with different quality levels as discussed in the standard H.264/SVC codec [72]. The network aware adaptation of end systems is often used to avoid congestion in the network, e.g., in conjunction with a transport protocol like TFRC

[73]. A framework that uses TFRC for efficient congestion aware SVC video delivery has been proposed in [74]. Rate smoothness and real-time requirements for video streaming are addressed in [75]. Further proposals provide QoS support from the network, for example through layer-based in-network packet dropping [76], use of priority queuing taking into account the layers' importance [77], and rate distortion models for link adaptation [78].

None of the aforementioned solutions for SVC transmission over wireless networks consider multiple video streams, where the rate distortion properties of the videos depend on the specific video content of the individual streams. The fairness-throughput trade off when streaming multiple videos to different users has been analyzed in [79] based on a gradient-based scheduling, and in [80] based on remaining video playback time for each client. In [81], Ji et al. extend the gradient-based scheduling to optimize the resource allocation so as to meet delay constraints.

At the time of this work, most of the existing research is based on simulation and theoretical analysis, and few experimental results for SVC in WLANs are available [82]. Moreover, to the best of our knowledge, existing work does not consider the dependency of the MAC layer operation, even if in a centralized downstream setting, on the allocation of capacity to video transmissions. Furthermore, the impact of other applications as well as uplink traffic on the downlink scheduling problem has not been pointed out.

5.5.2 Utility-Maximizing Cross-Layer Downlink Scheduling Problem

In this section we formalize the problem of maximizing the utility of cross-layer scheduling for downlink video streaming over a WLAN network. The following is not restricted to video traffic, but addresses the general problem of maximizing the utility of generic downlink flows whose utility curve is known.

We consider an 802.11e WLAN formed by one AP delivering video streams to M associated stations. The video delivery occurs through the AP EDCA video access category. In addition, we consider further traffic (in a separate EDCA Access Category) generated at the stations and destined to the AP, that we call "non-video downstream" (nvd) traffic.

5.5.2.1 Assumptions and Notation

For simplicity, we use the following assumptions.

Assumption A1: the non-video-traffic is generated by a *constant* number of stations in saturation conditions [83], i.e., they always have a frame available for transmission.

Assumption A2: 802.11 frame transmissions are error-free, and collision of the video traffic generated by the AP with non video traffic generated by other stations is assumed negligible.

Assumption A3: the quality of each video stream i is described by a utility curve $U_i(b_i)$, where b_i is the average bit-rate assigned to stream i. The utility curves are assumed to be known at the AP for each considered stream.

Of these assumptions, A1 appears necessary to prevent the background traffic to further depend on the AP operation. A2 is realistic in the presence of a relatively small number of competing stations and considering the higher priority of the video access category. It is an assumption, which permits to neglect the complications that an analysis devised to further take into account channel collisions would raise. As such, it permits us to focus our contribution on the core aspects of the problem tackled in this work. Finally, A3 restricts our treatment to utility curves based on *average* values. We have specifically used average Peak Signal to Noise Ratio (PSNR) versus average application layer bit rate as utility metric, but any other utility metric based on average rates would fit our framework.

We use the following notation. Station i is the station which receives video stream i and we use index i to either refer to the receiving station or the delivered stream unless ambiguity occurs. We consider a multi-rate scenario where each station is connected to the AP using a specific PHY rate. Let C_i be the *maximum application-layer* delivery rate that is available to stream i, in the assumption that the AP might continuously transmit MAC frames to station i *without* any backoff time between consecutive frames. Clearly, C_i will be lower than the actual PHY rate assigned to the station, because of overhead, and is higher than the maximum throughput achievable by the stream. Simple computation allows to derive the value C_i from the PHY rates for a specific 802.11 PHY layer. For instance, for a station exploiting an 11 Mbps PHY rate, $C_i = 7.21$ Mbps, whereas $C_i = 1.76$ Mbps in the 2 Mbps

case.

5.5.2.2 Problem Formulation

Our ultimate goal consists in determining and enforcing the combination of application-layer bit rates b_1,..,b_M that the AP should grant to the M video streams so that the total utility $\sum_{i=1}^{M} U_i(b_i)$ is maximized under the constraint that the WLAN provides sufficient "channel resources" to deliver the resulting MAC layer traffic.

To formalize the problem we introduce variables x_i defined as the percentage of the *whole* WLAN channel time assigned to video stream i. The average application-layer bit rate b_i assigned to the i-th stream is $b_i = x_i C_i$. We conveniently classify the channel time into two categories:

- x_{vd}: percentage of time assigned to the AP for the transmission of MAC frames carrying video traffic;
- x_{nvd}: the remaining percentage of channel time.

The latter includes the time spent by the AP for independently delivering non video traffic through other EDCA AC queues, the time spent by *background* stations for their non video transmission, and the supplementary channel time wasted by the MAC protocol operation (specifically, unused channel time - empty channel slots - because of backoff counters count-down).

If x_{vd} were a known constant, the utility-maximizing allocation would be the solution of the straightforward constrained maximization problem:

$$\max_{\{x_1,..x_M\}} \sum_{i=1}^{M} U_i(x_i C_i) \qquad (5.1)$$
$$s.t. \sum_{i=1}^{M} x_i = x_{vd}$$

We now proceed by deriving x_{vd}. Let a *round* be defined as the time interval between two consecutive AP transmissions of video packets. Figure 5.11 depicts the WLAN channel time as a sequence of consecutive rounds. The average duration of round T_r can be expressed as the sum of i) the average time T_{vd} consumed by the AP on the wireless interface to transfer a MAC frame containing video traffic, and ii) an

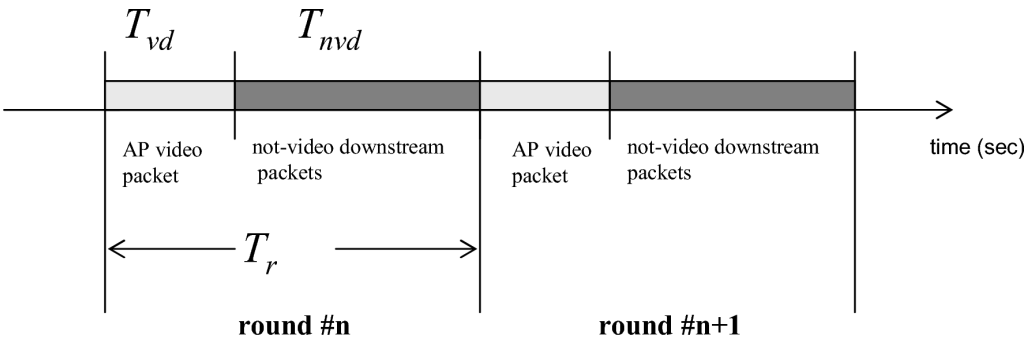


Figure 5.11: Evolution of time as sequence of *rounds*

average remaining time T_{nvd} which includes *both* the channel time wasted because of the WLAN MAC operation, as well as the time consumed for transmitting uplink or downlink non-video MAC frames. The percentage of time assigned to the AP for video traffic transmissions is then

$$x_{vd} = \frac{T_{vd}}{T_{vd} + T_{nvd}} \tag{5.2}$$

Now, two fundamental observations hold:

1. Under the saturation assumption A1, T_{nvd} is a constant independent of the AP scheduling decision;

2. Conversely, the average time T_{vd} is directly affected by the specific scheduling decision, namely the choice of the tuple $\{x_1,..x_M\}$.

As a consequence, x_{vd} is not constant but depends on the scheduling strategy. As an example, consider a single non-video station and let us assume that this station uses the same contention window of the AP's video access category. Given the same MAC parameters, in average a non-video transmission will occur for each frame transmitted by the AP (due to the long term fairness property of the 802.11 Distributed Coordination Function). Since i) the time to transmit the non-video frame depends only on the PHY rate of the non-video station, and since ii) the average number of channel slots that elapses between two consecutive channel transmissions is a constant which depends only on the MAC layer parameters ($CW_{\min}/4$, if we neglect collisions), also the average time T_{nvd} is constant and independent of the AP operation. In contrast, in a multi-rate scenario T_{vd} significantly depends on the

choices made by the AP. If the AP transmits all frames to a station with a large PHY rate, this time will be shorter than in the case where the AP sends frames to a low PHY rate station.

Going back to the general case, let us redefine the scheduling rule as follows. Let α_i be the fraction of frames transmitted by the AP to station i with respect to the total number of frames transmitted by the AP, with the obvious constraint

$$\sum_{i=1}^{M} \alpha_i = 1 \tag{5.3}$$

Then, the average time spent by the AP to transmit a video frame is computed as the weighted average

$$T_{vd} = \sum_{i=1}^{M} \alpha_i T_{x,i} \tag{5.4}$$

where $T_{x,i}$ is the average time needed to transmit a frame to station i, depending on the rate of station i. Substituting equation (5.4) into (5.2),

$$x_{vd} = \frac{T_{vd}}{T_{vd} + T_{nvd}} = \frac{\sum_{i=1}^{M} \alpha_i T_{x,i}}{\sum_{j=1}^{M} \alpha_j T_{x,j} + T_{nvd}} \tag{5.5}$$

where we recognize that each addendum of this sum is indeed x_i, i.e.,

$$\frac{\alpha_i T_{x,i}}{\sum_{j=1}^{M} \alpha_j T_{x,j} + T_{nvd}} = x_i \tag{5.6}$$

The above considerations permit us to provide two equivalent formulations of the utility maximization problem in WLANs.

1) Formulation based on percentage of transmission opportunities α_i provided to MAC frames addressed to station i:

$$\max_{\{\alpha_1,\cdots,\alpha_M\}} \sum_{i=1}^{M} U_i(x_i C_i) \tag{5.7}$$

$$x_i = \frac{\alpha_i T_{x,i}}{\sum_{j=1}^{M} \alpha_j T_{x,j} + T_{nvd}}$$

$$s.t. \sum_{i=1}^{M} \alpha_i = 1$$

2) Formulation based on the percentage of channel time x_i allocated to the video stream i:

$$\max_{\{x_1,..x_M\}} \sum_{i=1}^{M} U_i(x_i C_i) \tag{5.8}$$
$$s.t. \ \sum_{j=1}^{M} x_j + T_{nvd} \sum_{i=1}^{M} \frac{x_i}{T_{x,i}} = 1$$

where the constraint yields from straightforward algebra. For an intuitive explanation of the latter constraint, consider a time interval of one second. Within a one second time period, x_i can be alternatively interpreted as the amount of time dedicated to stream i. Hence, $\sum_{i=1}^{M} x_i$ is the time consumed to transmit video downstream traffic. The remaining time is spent by the MAC backoff operation and by the transmission of non-video stations. This accounts, in average, to one time interval T_{nvd} per each frame transmitted by the AP. Since, in a second, the number of MAC frames delivered to station i is given by the ratio $x_i/T_{x,i}$, and hence the total part of the one second spent for MAC backoff operation and transmission of non-video frames is $\sum_{i=1}^{M} x_i/T_{x,i}$ multiplied by T_{nvd}.

From the utility maximization problem (equation 5.8), we can identify the cross-layer information required to optimally schedule video traffic. The MAC layer information includes i) C_i for each $i = 1,..M$, ii) the transmission time of each MAC frame $T_{x,i}$, and iii) T_{nvd}. The required application layer information is $U_i(b_i) = U_i(x_i C_i)$ for every delivered stream i.

5.5.3 Practical scheduler

In this section we design a practical scheduler, tailored to H.264 SVC, which leverages the insights emerged during the problem formalization, and conveniently exploits them in a form suitable for fast practical operation.

We remark that the optimal solution to the constrained maximization problem can be obtained for example using linear programming techniques. However, the problem solution requires the run-time estimation of T_{nvd} (that can be also complicated) and the consequent change of scheduler policies. To overcome this problem we propose in this section a practical scheduler, that is T_{nvd} independent, but that

can lead the performance close to the optimum.

5.5.3.1 Practical Scheduler Overview

The practical scheduler that we propose runs *above* the MAC layer, and it is devised to properly arrange the order of NALUs delivered to the MAC layer. This permits to retain a fully standard and application-layer unaware MAC operation. As discussed below, the priority order depends on cross-layer information, and specifically it depends on both utility information provided by the application layer, as well as per-station channel rate information provided by the MAC layer (gathered by the NIC driver). A fundamental feature of our proposed approach is that, unlike the optimal approach presented in the previous section, it does not require the explicit *run-time* knowledge of the T_{nvd} time, indeed an information not easily gathered from the NIC driver. As a result, the proposed approach is seamlessly adaptive to any available AP delivery capacity and related fluctuations. Of course, this simplicity is paid with a sub-optimal operation; however, numerical results will later on prove that, with average PSNR versus rate utility curves, the performance degradation is almost negligible.

A convenient way to order NALUs as well as drop excess ones is to rely on a bank of small-size (we used 10 NALUs each) priority queues . The assignment of NALUs to queues, as well as priority values to queues, must take into account both i) the requirement that NALU decoding dependencies must be respected in the delivery order, as well as ii) the goal of maximizing the utility brought by the delivered NALUs. As illustrated in Figure 5.12, we propose to accomplish this goal by:

1. deploying a number of dedicated queues per each video stream, so that each queue carries the stream NALUs belonging to a specific video layer for that stream, and arrange them in an *intra-stream priority order* (Section 5.5.3.2);

2. sorting the deployed queues on the basis of cross-layer information (utility and rate per each video stream), so that they are arranged in an *inter-stream* priority order (Section 5.5.3.3);

3. deploying a *flat service priority discipline* for orderly draining NALUs from the sorted bank of queues.

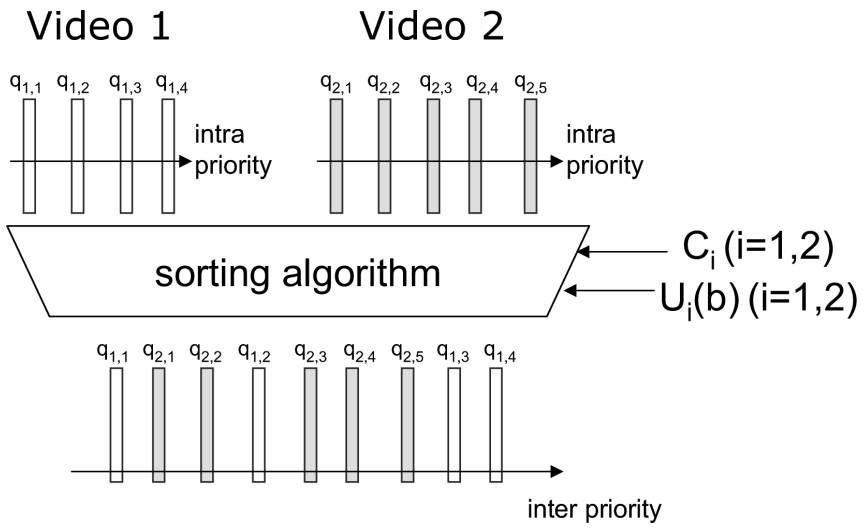


Figure 5.12: Conceptual sketch of queue merging

5.5.3.2 Intra-stream Priority

The traffic generated by a same video stream is conveyed to a bank of queues, where each queue accommodates NALUs belonging to a given scalability layer, i.e. a (TID,QID) pair. Considering that the default range for TID values is from 0 to 4, and considering two additional enhancement quality-layers (i.e., QID values in the range from 0 to 2), we deploy $5 \times 3 = 15$ limited-size queues, numbered from 0 to 14.

Intra-stream queues are sorted by setting a priority order among the video layers. As discussed in Section 5.3, a natural approach is to use the video-layer-to-priority mapping illustrated in Figure 5.2, which gives higher priority (0 being the highest priority) to the temporal scalability layers (in their TID order), and then decreasing priority to the corresponding quality enhancement. In formulae, a NALU with $QID = q$ and $TID = t$ is delivered to the queue with priority index $s = 5q + t$.

We remark that this intra-stream priority assignment does respect the decoding dependencies enforced by H.264 SVC and permits to reach a transfer efficiency (one minus the percentage of NALUs received at the destination, but discarded because of missing dependencies) very close to 100% [82].

5.5.3.3 Inter-stream Priority

Let us denote with $q_{i,s}$ the queue of the video stream i that handles NALUs for the video substream $s = 5q + t \in (0,14)$. Let $b_{i,s}$ be the average application-layer bit rate rate needed to deliver all the i-th video substreams until level s. Let $\{U_i(b_{i,s})\}$ be a vector of utility values computed in correspondence of the rates $b_{i,s}$, i.e. the utility brought by the transmission of all the video layers until level s.

The inter-stream queue sorting is based on a greedy algorithm. Rather than optimizing the scheduling for a given T_{nvd} value, the algorithm scans a set of pre-established quantized T_{nvd} values (or decision points). In our algorithm, T_{nvd} values are scanned in decreasing order, since a lower T_{nvd} value yields an higher AP capacity for delivering video traffic. For each T_{nvd} value, we determine the next queue to be served, namely the one which provides the best possible utility improvement. The sub-optimal nature of the proposed algorithm stems from the fact that the queue ordering committed for a prior T_{nvd} value cannot be changed, but only extended with the selection of a new queue to be served.

The following three parameters ($step$, N_{steps}, U_{th}) are used to perform the greedy algorithm:

- $step$. This is the time step according to which the T_{nvd} values are quantized. We used a small step (1 ms, roughly equivalent to the transmission time of a MAC frame at the maximum 802.11b PHY rate), so that at each step, at most one new queue is accommodated in the sorted list;
- N_{steps}. A large value (we used 200), so that the starting value $T_{nvd} = N_{steps} \cdot step$ does not permit to accommodate any stream;
- U_{th}. A minimum utility increment in order to take a commit decision (we used 0.5 dB for PSNR utility curves).

In brief, at each decision point, the greedy algorithm works as follows:

1. At the decision point h, we compute the total application-layer bit rate $b_i(h)$ that the i-th video would receive if all the queues already inserted in the inter-stream priority list were completely served. From $b_i(h)$ we derive the corresponding time percentage $x_i(h) = b_i(h)/C_i$.

2. We then check whether the next decision point $h-1$, corresponding to a smaller T_{nvd} value, would permit to accommodate a new queue. This is accomplished by computing the stability constraint condition in the formalization (5.8) as:

$$sc(h-1) = \sum_{i=1}^{M} x_i(h) + \left(\sum_{i=1}^{M} \frac{x_i(h)}{Tx_i} \right) T_{nvd}(h-1) \tag{5.9}$$

If the value $sc(h-1)$ is lower than 1, extra space is available for accommodating an additional queue.

3. Provided that this is the case, we compute, for each stream, the utility improvement that can be achieved by picking the next queue to be served among the ones that have not yet been inserted in the list, respecting the inter-stream priority order illustrated in Figure 5.12. This is provided by computing, for all streams i, the quantities

$$\tilde{x}_i(h-1) = \frac{1 - \left(\sum_{j=1, j\neq i}^{M} x_j(h) + \sum_{j=1, j\neq i}^{M} \frac{x_j(h) T_{nvd}(h-1)}{Tx_j} \right)}{1 + T_{nvd}(h-1)/Tx_i}$$
$$I(h-1|i) = U_i(\tilde{x}_i(h-1)C_i) - U_i(x_i(h)C_i) \tag{5.10}$$

where $\tilde{x}_i(h-1)$ is the percentage of time assigned to stream i if all the new capacity available is provided to such stream, and $I(h-1|i)$ is the utility improvement, with respect to the overall utility achieved in the previous iteration, that we would obtain if all the extra-time were given to the i-th stream.

4. Finally, we select, as the next queue to be inserted in the list, the one which yields the maximum utility improvement $I(h-1|i)$, provided that such an improvement is greater than a predetermined threshold U_{th}.

The procedure of the algorithm is elaborated in Algorithm 2.

5.5.3.4 Analytical Utility Performance of the Proposed Algorithm

Given a T_{nvd} value, is possible to analytically compute the utility performance provided by the proposed algorithm. It suffices to compute the rate b_i granted by the scheduling algorithm's operation to each i-th video stream and sum the corresponding utilities.

For this purpose, let us enumerate the queues according to their assigned priority order, from 1 (higher priority queue) to W. For a given T_{nvd} value, the priority scheduler will serve the set of queues $1..lsq$, where $lsq \leq W$ is the index of the last

Algorithm 2 inter-stream queue sorting

Input: Utility function U, number of video M, increase of step size $step$, minimum expected utility change U_{th}, number of decision points N_{step}.
Output: Optimal sequence of priority queues $\tilde{q}_{opt}$;
Initialization: index of last queue of the video i-th inserted: $last_i = 0$, index of decision point: $h = N_{step}$, Iteration index, $I = 0$.
while there are queues not yet inserted **do**
  $T_{nvd}(h-1) = (h-1) * step$
  for $i = 1$ to M **do**
    $b_i(h)$ = cumulative bitrate of the i-th video up to substream $last_i$
    $x_i(h) = b_i(h)/C_i$
  end for
  $sc(h-1)$ = see eq. 5.9
  if $sc > 1$ **then**
    continue while loop
  end if
  $max\_index = 0$
  $max\_value = 0$
  for $i = 1$ to M **do**
    if $last_i$ == number of i-th video substreams **then**
      continue for
    end if
    $\tilde{x}_i(h-1)$ = ... see eq. 5.10
    $I(h-1|i)$ = ... see eq. 5.10
    if $I(h-1|i) > max\_value$ **then**
      $max\_value = I(h-1|i)$
      $max\_index = i$
    end if
  end for
  if $max\_value > U_{th}$ **then**
    $i = max\_index$
    $k = last_i + 1$
    $last_i = last_i + 1$
    $flat_q \leftarrow q_{i,k}$ #insert the new queue related to video i-th and substream k-th
  end if
  $h = h - 1$
end while
output: $\tilde{q}_{opt}$

queue served by the scheduler. The bitrate b_i granted by the scheduler to the i-th video is the cumulative bitrate of the streams of the i-th video that feed the $1..lsq$ queues. This latter evaluation of b_i is straightforward, since the association queue-stream is known. Thus, the only remaining problem is the determination of lsq for a given T_{nvd} value. The iterative approach detailed in algorithm 3 is designed for this purpose. We start considering the first priority queue and iteratively add a queue, following the priority order. At each iteration, we evaluate the cumulative bitrate b_i associated to each video, given that all the considered queues are fully drained. From b_i we evaluate the time percentage x_i and verify the stability constraint. Whenever the stability constraint first exceeds one, it means that the last added queue is precisely the lsq one. The bit rate assigned to the lst queue is finally determined by computing, through the stability constraint, the percentage of time x_{lsv} assigned to this queue and consequently derive b_{lsv}.

Algorithm 3 Calculate $U_{tot-sub}$

for $j = 1$ to W **do**
  for $i = 1$ to M **do**
    b_i = cumulative bitrate of the i-th video feeding the queues $1..j$
    $x_i = b_i / C_i$
  end for
  $sc = \sum_{i=1}^{M} x_i + \left(\sum_{i=1}^{M} \frac{x_i}{Tx_i} \right) T_{nvd}$
  if $sc > 1$ **then**
    $lsq = j$
    lsv = index of video associated with lsq break the FOR
  end if
end for
$x_{lsv} = \frac{1 - \left(\sum_{j=1, j \neq i}^{M} x_j + \sum_{j=1, j \neq ldv}^{M} \frac{x_j T_{nvd}}{Tx_j} \right)}{1 + T_{nvd}/Tx_l sv}$
$b_{lsv} = x_{lsv} C_{lsv}$ and $U_{tot-sub} = \sum_{j=1}^{M} U_i b_j$

5.5.3.5 Linux Implementation

We conclude the section by briefly discussing the implementation of the practical scheduler in Linux. We employed an AP equipped with an Atheros card. We modified the MadWiFi driver in order to run-time retrieve both the number of empty spaces available in the EDCA AC video queue, and the actual PHY rates employed by the AP to transmit MAC frames to each station i. The scheduler

has been implemented in the Linux OS through the Linux traffic control tool at the IP layer. To prevent from losses emerging because of NIC buffer overflow, we have developed a simple flow control mechanism for delivering frames from the scheduler to the driver. It is implemented by periodically (250 ms) sampling the NIC buffer availability, and setting the IP queues draining rate for the next period to the maximum rate which guarantees that no MAC buffer overflow will occur even if the AP will not transmit at all in the next period. With the same periodicity, we query the driver for retrieving information about changes in the PHY rate of the connected stations. If this occurs, we rerun the algorithm and change the inter-stream queue priorities accordingly.

5.5.4 Performance Evaluation

We have evaluated the performance of the practical scheduler over an 802.11e WLAN test-bed. The experimental assessment of H.264 SVC in-network adaptation approaches is not straightforward, and it was made possible only by employing the tools and the methodology introduced by our SVEF evaluation framework (§ Section 5.4).

5.5.4.1 Experimental Scenario and Utility Function Evaluation

We used the soccer video with 4CIF resolution and 30 frames/s as a reference. We encoded it as H.264 SVC stream using the Joint Scalable Video Model reference software (JSVM, [61]). The resulting encoded video comprises a base layer, including 5 temporal scalability layers, plus two Medium Grained Scalability (quality enhancement) layers per each temporal layer, for a total of 15 video substreams. For the encoded video, we off-line computed an utility curve described in terms of average PSNR versus average bit rate. The computation of the PSNR versus rate curve was performed by stripping out the layers, measuring the average bit rate, decoding the resulting video, and computing the average PSNR. The resulting PSNR curve is reported in Figure 5.13.

The considered network scenario comprises of a number of downlink video streams addressed to different stations, and generated from the same video sequence translated in time. In addition to the video downstreams, we generate non video traffic

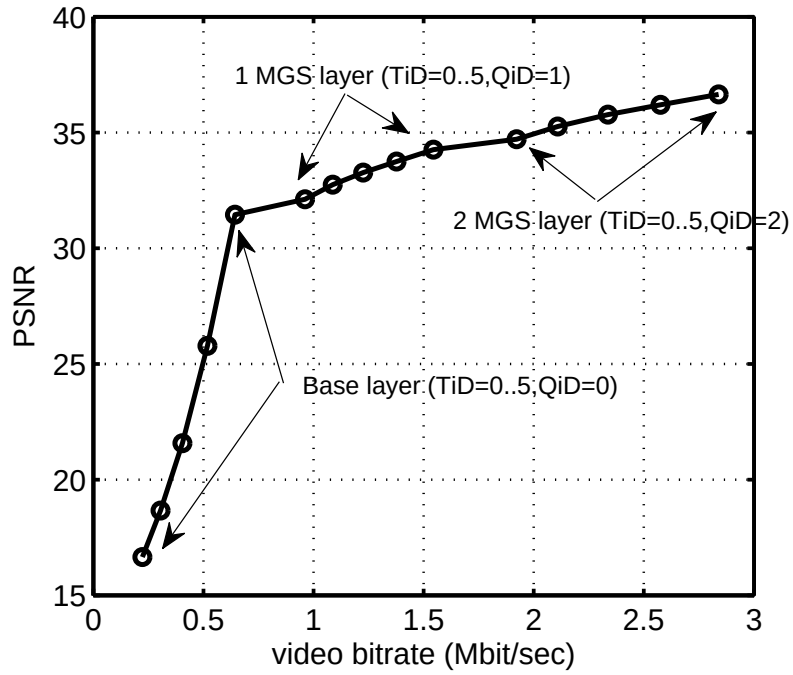


Figure 5.13: PSNR versus bitrate of the encoded video used in the analysis

from a variable (from 0 to 5) number of stations transmitting at 2 Mbit/sec, which uploads UDP greedy traffic (saturation conditions) generated through iperf. The remaining stations are Linux laptops with Ralink WiFi chipsets. The uplink traffic is best effort, thus the Ralink driver is set with CWmin=31 and AIFS=2. On the contrary, the video traffic accesses the channel with CWmin=15 and AIFS=1.

5.5.4.2 Experimental and Theoretical Performance versus the Number of Uplink Stations

In these experiments we consider two video streams delivered by the AP to two different stations, one connected with an 11 Mbps PHY rate, and the other with a driver-enforced 2 Mbps PHY rate. To prove the practical scheduler effectiveness, we introduced in the wireless network other N_{up} greedy stations that transmit packets at 2Mbps in the uplink direction. Changing the number N_{up} we induced in the wireless network different values for T_{nvd}. The downlink video traffic is delivered using the WME/EDCA video queue and the uplink traffic transmitted by the other

stations uses the best-effort queue.

Performances are measured in terms of cumulative utility (PSNR) delivered to the two video receivers and are plotted versus the number of uplink greedy stations, N_{up} that is varied from 0 to 5. Note that the actual value of T_{nvd} for each experiment is independent on the scheduler and is only a function of the PHY bit rate and of the number of greedy stations, N_{up}. Thus the optimal solution has to be computed for each N_{up} value. On the contrary, the practical scheduler works independently of the T_{nvd} value.

A fundamental problem we needed to face is the need to derive the average T_{nvd} value that resulted for each specific value of N_{up} to compare the measured results with the analytical. We were not able to derive such an average value from driver-level information. Moreover, the analytical derivation of such a parameter, in principle relatively easy (e.g., using EDCA extensions of the model [83]) from the knowledge of the MAC layer parameters employed by the competing stations, was discouraged by the fact that, as shown in [84], the operation of the two considered cards slightly, but noticeably for our specific purposes, differs from the theoretically expected performance.

Therefore, we resorted to an hybrid experimental/analytic off-line estimation of the T_{nvd} parameter. At first, we experimentally derived the actual AP throughput ρ_{AP} achieved when competing with k background best-effort Ralink stations using the 2 Mbps PHY rate (as in our scenario); All stations were loaded with saturated UDP traffic with 1500 bytes IP packets. Then, we derived T_{nvd} by recognizing that $T_{nvd} = \frac{1500 \cdot 8}{\rho_{AP}} - T_{x,AP}$, being $T_{x,AP}$ the computed transmission time for a 1500 bytes MAC frame by the AP. Using this approach, the obtained values of T_{nvd} for the different values of N_{up} are reported in Table 5.3.

Figure 5.14 reports the cumulative PSNR as resulting from i) the solution of the optimal scheduling rule determined by solving equation 5.8 (marked as "optimum"); ii) the analytical results obtained by analytically computing, through the procedure

N_{up}	1	2	3	4	5
T_{nvd}	0.004439	0.005874	0.007725	0.0102	0.0116

Table 5.3: Values of T_{nvd} for N_{up} greedy stations at 2 Mbps PHY rate

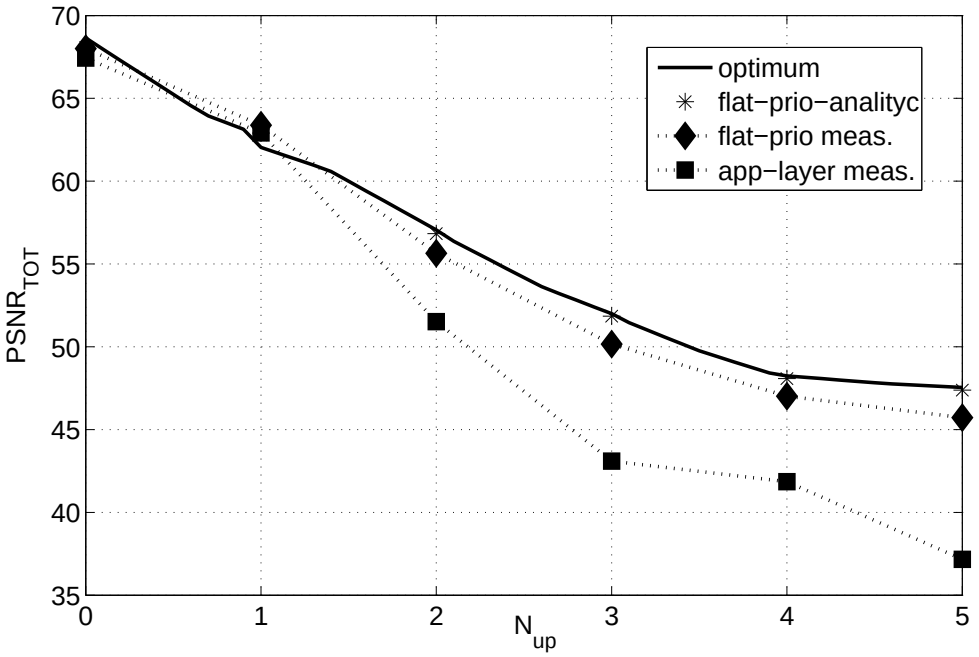


Figure 5.14: Cumulative PSNR versus the number of Uplink Stations

3, the utility achieved by the proposed sub-optimal flat priority scheme detailed in alg. 2 (marked as "flat prio analytic"); iii) the experimental results obtained by running the Linux AP implementation of the proposed flat priority approach (marked as "flat prio meas."); iv) the measurements obtained implementing a scheduler which does not rely on the knowledge of the PHY rate at which stations are connected, but uses only application layer information (marked as "meas. app. layer sched"). This scheduler deploys 15 queues, one per each video layer. However, with no supplementary insights on the available PHY rate at which streams are connected, and in the considered scenario where video layers bring the same utility for different streams (we recall that we used the same video sequence for all streams), its best strategy is simply to share the queues among the different video streams and drain traffic from the queues according to their intra-stream priority order. We have also measured the performance without any scheduler. The results have not been shown because the maximum of the curves was $29.13dB$ of PSNR, corresponding to the case without uplink traffic ($N_{up} = 0$), and thus the readability of the figure would have been compromised.

From Figure 5.14, we see that the performance advantage of the optimal scheduler is negligible with respect to the analytical solution of the practical scheduler, and it is marginal also when compared with the actual experimental results.

It could be argued that the suboptimality of the practical scheduler may depend on the considered utility curve, and that it is possible to find cases where the performance difference becomes notable. An in depth analysis of the impact of different utility shapes over the performance is left to further work. However, we believe that major performance differences are deemed to emerge only with very particular, and unrealistic, utility shapes. As expected, comparison with the application-aware-only scheduler shows that the usage of cross-layer information yields a significant performance improvement, especially in the case of significant capacity restriction (large values of T_{nvd}).

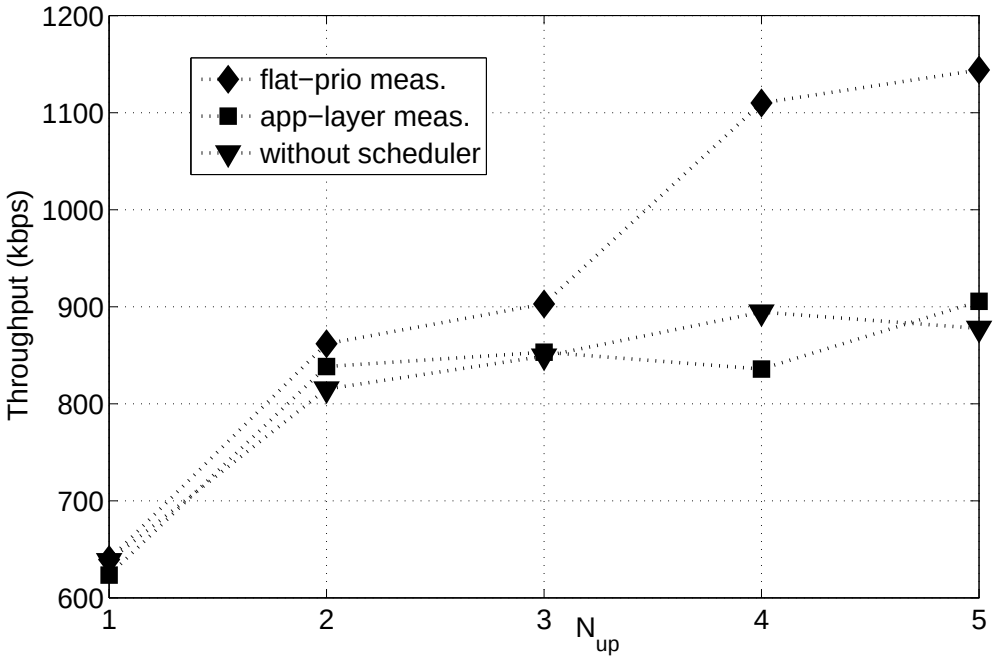


Figure 5.15: Aggregated throughput of uplink stations

Finally, in Figure 5.15 we report the aggregated non-video throughput for the greedy N_{up} uplink stations. A positive side effect of our proposed approach is that not only it does not penalize uplink stations, but it may even improve the uplink traffic throughput, as comparison with the application-layer-only scheduler and the

absence of scheduler shows. This apparently counter-intuitive fact can be explained by considering that, in order to maximize the Utility, the cross-layer scheduler tends to select packets directed to the 11 Mbps stations, hence increasing the overall network throughput of the wireless network (in other words, the scheduler tends to limit the WLAN performance anomaly).

5.5.4.3 Experimental and Theoretical Performance versus Number of Video Downstreams

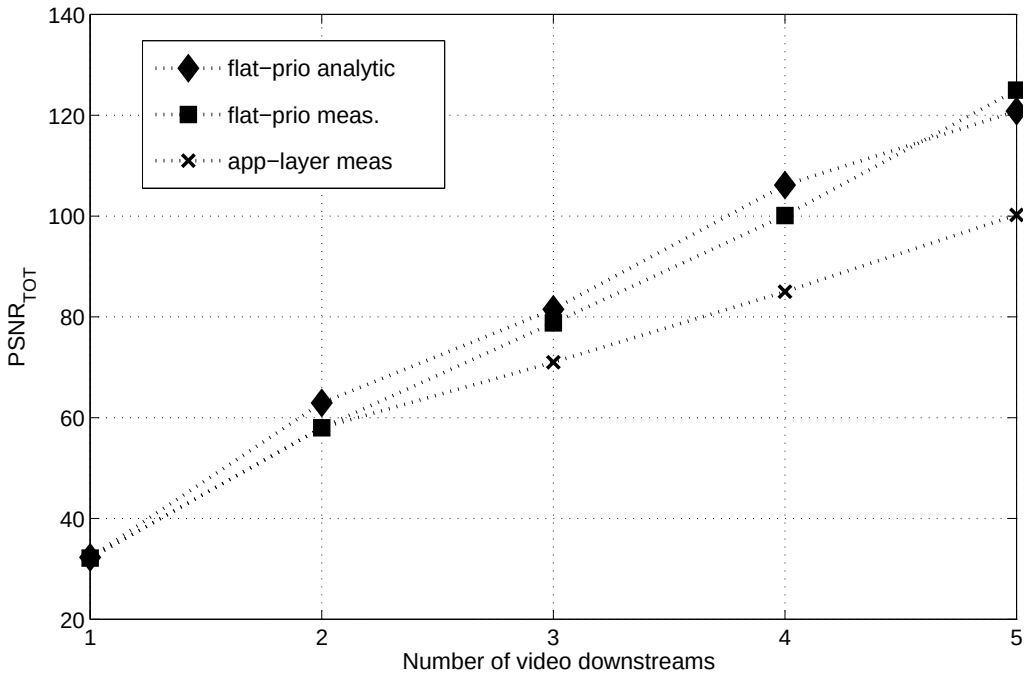


Figure 5.16: Cumulative PSNR versus number of video downstreams

A second set of measurements was performed to determine the scheduler operation for a varying number of video streams. Figure 5.16 shows results obtained in a scenario characterized by one background best-effort station transmitting at 2 Mbps, and a varying number of video streams, of which one is delivered at 2 Mbps, and the remaining at 11 Mbps.

The figure compares the experimental and analytic results obtained by the "flat prio" scheduler, with that of the application-layer-only approach. The results for the

optimal scheduler are perfectly coincident, for such scenario, with that of the proposed practical "flat prio" approach. As expected, the figure confirms the superiority of the cross-layer approach with respect to an application-aware-only approach.

The growth of the total utility with an increased number of video streams may not be considered intuitive (indeed, a comparable shared capacity is provided by the AP), but it is readily explained by considering that, as shown in Figure 5.13, the relative utility gain is large at low bit rates, and gets progressively smoother as more bit rate is provided to a given stream (as in the case of a small number of video streams).

5.6 Chapter Conclusions

In this chapter we have presented experimental results dealing with application-aware H.264 Scalable Video Coding (SVC) delivery over Wireless LANs. To the best of our knowledge, these results are among the first that are based on experimentation involving H.264 SVC in a real wireless testbed. To accomplish this goal, we had to develop several software components, presented in Section 5.4, to provide functions such as streaming, NALU dependency filtering, concealment of missing frames, etc., which are not readily supported by off-the-shelf publicly available H.264 SVC-related software.

Our preliminary results, illustrated in Section 5.3, prove the viability of the Virtual BottleNeck (VBN) mechanism and the application-aware scheduling. The VBN performs bandwidth throttling before the traffic is delivered to the WLAN Access Point, so that all packet loss occurs inside the VBN. The scheduler prioritizes the packets according to their importance for the video quality and also takes the dependency of the packets into account. This allows to discard packets at the VBN in a manner that has the least negative impact on the overall video quality. The scheduler maintains separate queues for the different priority levels.

Some conclusions can be drawn from this preliminary results. First, significant performance improvements can be achieved even with very simple scheduling approaches (we developed an approach "just" based on priority queuing). Second, results show that SVC should be encoded with a base layer that is as small as it can

be without significantly affecting overall encoding efficiency. This allows to cope well with the specific bandwidth characteristics and fluctuations expected in a WLAN. We believe that such a WLAN-aware SVC encoding should be further explored to better understand to what extent a base layer reduction is possible without impairing SVC encoding efficiency, and to understand the impact of different encoding choices for the remaining enhancement layers.

In Section 5.5 we extended the WLAN downlink video streaming scenario to the case of the presence of uplink background traffic. The analysis of this scenario and considerations on the operation of IEEE 802.11 networks produced a formalization of the problem. Although we are able to find the optimal analytical solution, estimating the time used by non-video traffic (i.e. the T_{nvd} parameter) is non trivial in the real World. For this reason we have devised and implemented a practical sub-optimal scheduler, that is able to achieve a performance close to the optimum. Section 6.3 in the next chapter will deal with the issue of non-video time estimation and how this can fit in an in-driver cross-layer scheduler architecture.

The work illustrated in this chapter has been presented in international conferences. Partially, at the IEEE IWCLD 2009 conference, with an article entitled "Application-aware H.264 Scalable Video Coding delivery over Wireless LAN: Experimental assessment", coauthored by Giuseppe Bianchi, Andrea Detti, Pierpaolo Loreti, Francesco Saverio Proto, Wolfgang Kellerer, Srisakul Thakolsri and Joerg Widmer. Partially, at IEEE ISCC 2009, with an article entitled "SVEF: an Open-Source Experimental Evaluation Framework for H.264 Scalable Video Streaming", coauthored by Andrea Detti, Giuseppe Bianchi, Francesco Saverio Proto, Pierpaolo Loreti, Wolfgang Kellerer, Srisakul Thakolsri and Joerg Widmer. And partially at IEEE WoWMoM 2010 with an article entitled "Cross-layer H.264 Scalable Video Downstream Delivery Over WLANs", coauthored by Giuseppe Bianchi, Andrea Detti, Pierpaolo Loreti, Srisakul Thakolsri, Wolfgang Kellerer and Joerg Widmer.

Chapter 6

Flexible, Modular and Virtualizable MAC Layer

In the previous chapter we have seen that scalable video streaming techniques can be very effective in WLAN hotspots, but to deploy it in real World dynamic scenarios, a support at the lower layers is needed, in order to monitor the status of low-level queues and estimate the non-video time T_{nvd}, i.e. the missing parameter that we need to be able to find the optimal solution to the problem stated in section 5.5.1.1.

But MAC layer flexibility has implications that go beyond the video delivery scenario. In this chapter we devise a general flexible MAC framework, show its wide range of applications, and finally use it to devise a dynamic scalable video scheduler.

6.1 Overview

Wireless networks are extensively deployed due to their low cost and configuration easiness. However, they are not adapted to the new services and applications that are increasingly demanded by users. Current implementation of the IEEE 802.11 specification is supported in hardware devices and software developments, but they do not provide the adaptability that would enhance user experience in next generation networks. In this work, we present a new wireless framework, based on the one currently supported by the Linux stack: *mac80211*. This new framework, named

mac80211++, has been tailored to improve MAC features in terms of: (*i*) **modularity**, by defining different 802.11 MAC services; (*ii*) **flexibility**, by enabling dynamic configurability of the 802.11 MAC; (*iii*) **virtualization**, by managing parallel independent 802.11 MACs accessing the same system resources.

Many effective approaches are proposed for adapting the wireless devices' operation to specific contexts and services, but only a few of them can be really implemented in the actual wireless interfaces. The road to accommodate necessary and advocated improvement of wireless technologies goes through the rethinking of the current protocol stacks, pushing the programmability of the system towards the physical interface. New architectures proposed for wireless interfaces provide great flexibility obtained through the rationalization and modularization of existing solutions. However the effective exploitation of this flexibility is closely related with the implementation of a suitable infrastructure able to support the dynamic composition of elementary functions.

6.2 The mac80211++ Framework

Wireless networks are a popular technology for Internet access. The evolution of new services and applications require WLANs to rapidly adapt to these modifications. However, such evolution require new amendments in IEEE 802.11 standard [1] with the consequently increase of time to approve them. In addition, these new changes must be adopted by manufacturers by exploiting a new set of devices. Consequently, this process is slow and time-consuming, making standardization much slower than real user demands.

Most of WLAN card manufacturers follow the *SoftMAC* approach, much more flexible compared to the old *FullMAC* solution. *FullMAC* leaves all the control of the MAC layer functions to the card hardware/firmware, whereas *SoftMAC* implements a new set of control MAC primitives at the software level. The framework *mac80211* [85], which is part of the Linux 802.11 stack and depicted in Fig. 6.1a, provides *SoftMAC* capabilities. Nevertheless, the *mac80211* lacks of modularity and flexibility since it is a monolithic block composed of many sub-modules highly interconnected. Following the FLAVIA [86] paradigm, we propose and develop a

preliminary implementation of a new framework, namely *mac80211++*, aimed to specify a solution whose advantages are three-fold: modularity, flexibility and virtualization.

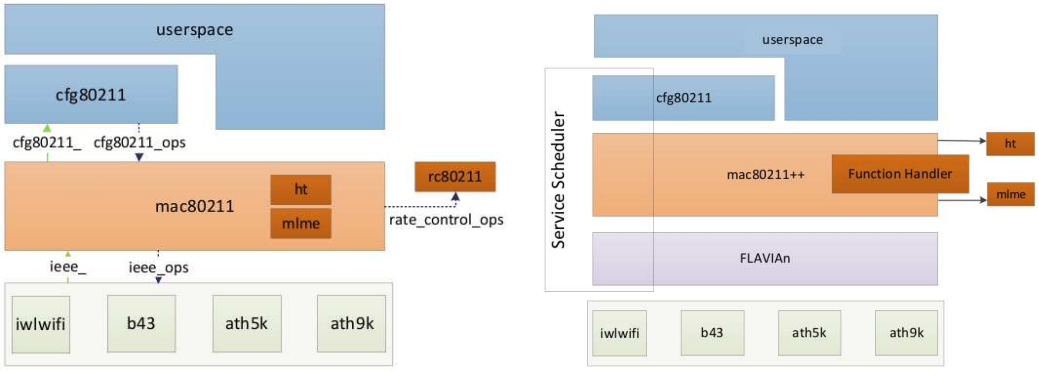
Then, the goals of *mac80211++* are to leverage on the current implementation *mac80211*, widely adopted in 802.11 networks, and to reduce the complexity and shorten the time when introducing changes to the standard. This will boost the implementation of new improved services and reduce the amount of time for these modifications to be commercialized. To this aim, we design a service scheduler and split the *mac80211* framework components. The service scheduler adds and loads new services, flexibilizing the implementation of new modules compared to the standard procedure. Untangling the highly interdependent relations of the *mac80211* components, we pursue to reduce the complexity of the current framework and to foster its modularization, by allowing sub-components being loaded independently.

6.2.1 Existing Framework

Fig. 6.1a depicts an overview of the existing Linux 802.11 stack. This stack specifies the framework *mac80211* that enables *SoftMAC*-capable device drivers used for operating with 802.11 hardware. While some of the MAC functionalities are implemented at the hardware level, *mac80211* implements features such as handling several higher-layer components of the MAC, including support for HW/SW cryptography, power saving, .11n style aggregation or LED management. The *mac80211* module plays two key roles: (*i*) Wrap the packet incoming from the upper layers and translate them into the 802.11 frame format; (*ii*) Control management operations related to the IEEE 802.11 standard.

A standard wireless driver with Linux wireless capabilities includes some kernel modules and provides interfaces used by user level tools to configure the device behavior, as depicted in Fig. 6.1a. The main modules defined in the framework are the *mac80211* and the *cfg80211*; these modules are loaded and used by the drivers (e.g., ath5k, ath9k, b43) that are implemented in separate Linux kernel modules.

Bidirectional interfaces are defined among modules as represented in Fig. 6.1a by the arrows. The exported functions provide a direct interface shown with the solid arrows. The usage of an exported function introduces a dependency in the



(a) Overview of the mac80211 framework (b) Overview of the proposed *mac80211++*

Figure 6.1: Overview of the existing (left-sided) and proposed (right-sided) frameworks.

direction of the arrow (e.g., the driver depends on mac80211). The interface in the other direction is implemented through the registration of callbacks (i.e., function pointers). In Fig. 6.1a this dependency is represented by dashed arrows and the labels represent the structure containing the function pointers.

The rigidness of the mac80211 is a caveat for developing new services. This framework is a rigid block formed by a set of sub-modules highly interconnected, e.g.: the MAC layer management entity (*mlme*), the high throughput (*ht*) or the MPDU aggregation (*agg*), as specified in the IEEE 802.11n standard [87]. These parts are defined in dedicated files but not implemented as separated modules, thus preventing any kind of modularization.

6.2.2 A New Framework: *mac80211++*

Motivated by the rigidness of the *mac80211* framework, we develop a new solution, named *mac80211++*, which aims to overcome it. Fig. 6.1b depicts the new framework, *mac80211++*, showing the new blocks introduced with respect to the existing framework depicted in Fig. 6.1a. First, we provide a proof of concept of the modularity of *mac80211++*. Second, we develop and implement a Function Handler and a Service Scheduler that manage the loading and creation of new functions and services respectively, proving flexibility. Third, we describe as well the virtualization support, by adding an overlay layer, FLAVIAn, between the device drivers and the

mac80211++ framework.

6.2.2.1 Modularity

The development of *mac80211++* can be considered as a first and relevant step towards the modularization of the wireless component inside the Linux kernel. Some *mac80211* functionalities might be conveniently separated in order to provide the developers with a novel degree of flexibility. Thus, we propose to "break" the monolithic *mac80211* framework and evolve to a new extended and more modular framework. The basic idea is to rely on the definition of well-defined interfaces for those functionalities. To this aim, we follow the rate control module approach replicating its interaction modality. This module depends on *mac80211*, but can be built as an independent module, within the framework or at the driver level, being able to load it at run-time.

Basic mgmt operations	**Event handling functions**	**Power mgmt and saving**
*mgmt_assoc	*sta_rx_queued_mgmt	*dynamic_ps_disable_work
*mgmt_auth	*mlme_notify_scan_complete	*dynamic_ps_enable_work
*mgmt_deauth	*sta_rx_notify	*dynamic_ps_timer
*mgmt_disassoc	*sta_tx_notify	*send_pspoll
*sta_setup_sdata		*recalc_ps
*sta_work		*sta_to_sleep
		*send_nullfunc
		*sta_reset_beacon_monitor
		*sta_reset_conn_monitor
		*sta_restart

Table 6.1: APIs for *mlme* support (* = *ieee*80211_)

Then, by using the same rationale we separate the management (*mlme*) algorithm (STA operation) as well as the support for high throughput (*ht*) from the rest of mac80211 modules. We describe the interfaces for the *mlme* module that we have added through a *mac80211_ops* structure to the *ieee80211_local* structure. Table 6.1 illustrates the three main categories: (*i*) the specific management part and its setup, (*ii*) the event handling part that includes several notifications, timers and management frames reception coming from the wireless network and (*iii*) functions related to power saving and power management. Fig. 6.1b illustrates the extensions carried out in the *mac80211* framework, turning it into a more modular framework.

6.2.2.2 Flexibility

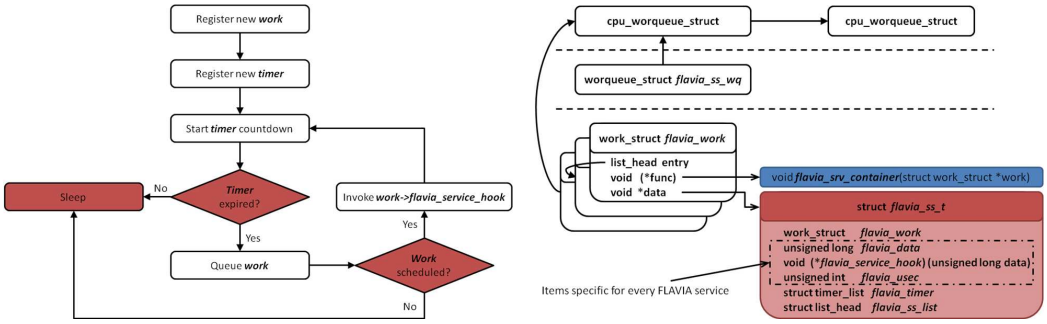
The flexibility provided by *mac80211++* fosters the extension of the basic functionalities defined by the IEEE 802.11 protocol. In particular, our framework permits to implement innovative services and enhanced functions, providing a general yet flexible mechanism to extend the mac80211 framework. To this end, we design and develop two auxiliary kernel modules, namely the *Service Scheduler* and the *Function Handler*, which are liable, respectively, for managing the scheduling of a new service and the registration of the enhanced functions, which are executed at the occurrence of specific events handled by *mac80211++* (e.g., packet reception, packet transmission or channel switching).

Service Scheduler. The Service Scheduler has been designed to provide a simple and standardized mechanism to schedule new services. Through this system, developers can focus only on the implementation of the main service functions, using the Service Scheduler as a mean to schedule periodically its execution. The Service Scheduler will run the functions registered by the service during its initialization phase. In addition, to simplify the implementation of a new service, the Service Scheduler architecture improves its maintenance, since the implementation of its internal functions can be improved to support enhanced services, as long as its APIs are not modified. Indeed, it can be easily updated with more sophisticated functionalities to meet the requirements of real-time systems.

Fig. 6.2(a) illustrates the main steps to register and execute a new service. When a new service is registered, the Service Scheduler creates a new Linux kernel *work* representing the task implementing the deferred service function, and adds it to a dedicated *work-queue* specifically designed to handle all services. When the timer expires, the Linux kernel *work* implementing the service is queued on the *work-queue*, which contains all the tasks that must be executed immediately. The Service Scheduler defines only one Linux kernel thread to extract and activate the *works* implementing the services on the *work-queue*, in order to serialize the management of all the events occurring in a distributed scenario like the channel access.

Once the *work* can be scheduled, the Linux kernel thread, which handles the *work-queue*, invokes an outer function, namely *flavia_srv_container*, which, in turn, executes the function implementing the service (pointed by *flavia_service_hook*), and

reschedules the timer to execute the service later.



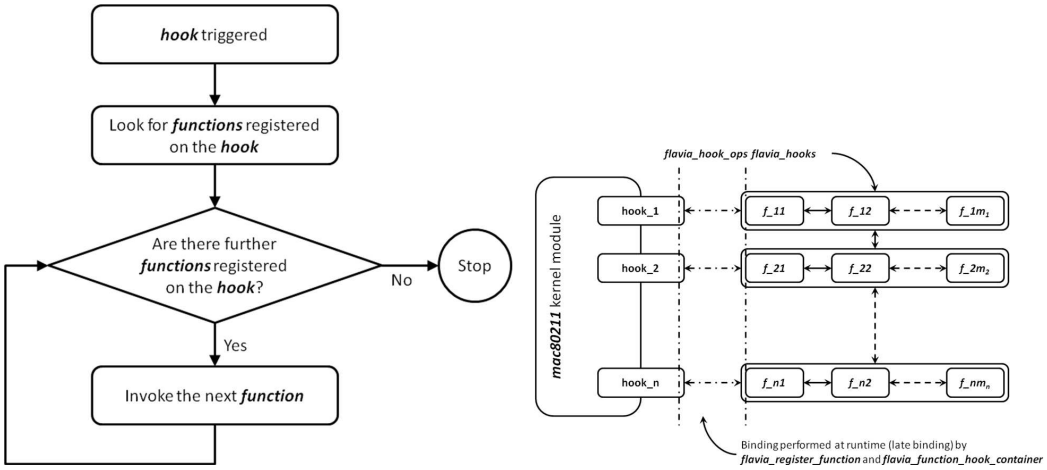
(a) Work Flow (b) Work-Queue

Figure 6.2: Service Scheduler: Flow chart and work-queue.

Function Handler. The Function Handler (FH) is designed to provide a standardized mechanism to hook the *mac80211++* code (i.e., to add piece of code that acts as glue between any function and the *mac80211* procedures). More specifically, the FH permits to register a function to any hook added to the *mac80211++* code; thus improving its functionalities with new functions. As depicted in Fig. 6.3, at the occurrence of a specific event, the Function Handler will call the functions previously registered on that hook. For example, when a new frame is received, the control flow of the *mac80211* code reaches a hook that transfers the control to the FH, which, in turn, invokes the execution of all functions registered on that hook. Note that the function invoked by the Function Handler can register a service or create a new task executed by an independent kernel thread. Therefore, the FH mechanism provides a high level of flexibility to the developers of new functionalities and services.

6.2.2.3 Virtualization

While *mac80211++* extends the capabilities of the original *mac80211*, the design of the *mac80211* framework is intrinsically bound to the physical capabilities of the HW advertised by the different device drivers. Past research work [88, 89, 90, 91] has demonstrated that single radio hardware could be virtualized in a way very similar to the virtualization of computational resource in the hardware resources of computer. *mac80211*, and by extension *mac80211++*, brings a logical view to the different wireless interfaces present in the system, but does not offer the proper



(a) Work Flow (b) Hooks Double Linked List

Figure 6.3: Function Handler: Flow-chart describing the main operations performed by the Function Handler and structure used to fulfill the management task.

abstraction necessary for implementing proper virtualization without breaking the existing code base. To overcome this limitation, we design FLAVIAn, an overlay layer that offers virtualization capabilities to *mac80211* and *mac80211++*, while preserving the existing hooks and API.

Therefore, FLAVIAn abstraction is twofold: first, FLAVIAn presents the usual drivers hooks that the traditional *mac80211* is expecting. As well, FLAVIAn proposes the counterpart drop-in replacements to be used in each hardware driver

To ensure appropriate interaction of the *mac80211* stack and the device drivers through the FLAVIAn overlay, a small modification in the driver code is also required in order to reroute the *mac80211* callbacks to the equivalent *ieee80211_flavian_ops* structure specified by the FLAVIAn overlay.

The key functionality covered by these handlers cover frame transmission, enabling/ disabling the hardware, configuring Rx filtering or notifying about status of the scanning procedure (start/complete). Similar modifications will be required to support other *mac80211* drivers (e.g., ath9k, b43) with the FLAVIAn overlay, but as we explained above, such changes will involve limited programming effort.

6.2.3 Use Cases

This section specifies a representative set of use cases to illustrate the functionality introduced by the *mac80211++* framework.

6.2.3.1 Advanced Monitoring

The *Advanced Monitoring Service* (AMS) module provides a passive monitoring service able to measure several parameters related to radio channel conditions, capabilities of neighboring nodes and MAC 802.11 parameters estimation. Each node performs PHY/MAC layer measurements within the time-scale of microseconds. The wireless cards are set to promiscuous mode to ensure a comprehensive view of the current wireless channel conditions. Then, all the measurements are performed within the normal activity of the wireless card and reported periodically. The AMS module supports multiple network interfaces per node. It works on a frame level, meaning that all the frames sent and received by each network interface must be examined by the AMS functions. This imposes high requirements on the AMS module on the effectiveness of the frame analysis (i.e., limited computational power available at the nodes).

The AMS module hooks in the mac80211 module of the Linux kernel are placed in the *ieee80211_rx()* function for the downlink frame path and in the *ieee80211_tx()* function for the uplink frame path. The measurement functions called by the hooks require access to each frame header and frame timing information to discover and calculate a set of parameters per each neighboring station interface, such as: supported rates, SNR, F/BER, RTS_Threshold or number of retransmissions.

For communication with user space the netlink mechanism is used. The application that requires monitoring data from the AMS module sends the command to the receiving function of the AMS module. This command defines the parameters the application requests and the time interval at which results are to be sent to the application.

6.2.3.2 SuperSense (SPS)

The virtualization and flexibility features of our proposed framework foster the development of *SuperSense* (SPS), an innovative monitoring service that dynamically analyses the available wireless spectrum using both passive and active techniques to estimate the best network configuration. SPS analyses continuously the available wireless channels to select the set of parameters that provides the best network performance.

The monitoring activity is performed concurrently to the data TX using two virtual interfaces operating over a single physical interface. The virtualization module is liable for scheduling the activities of the different virtual interfaces. In particular, the time spent for data transmission and active monitoring tasks is scheduled according to a time division mechanism implemented using a preemptive weighted round robin policy.

This module sets and manages the total duration of a SPS period and the specific length of the operation modes by introducing a new data structure, the *super-frame*. The duty-cycle of the super-frame, representing the alternation of transmission and monitoring phases along with the time assigned to each activity, is broadcast by the Access Point using a new Information Element (IE) contained in the beacon. The IE contains two main variables indicating the overall duration of the *super-frame* and the time spent to perform the active monitoring. Every *super-frame* always starts with an active monitoring period followed by a transmission period, in which all nodes that belong to the same BSS operate using the same medium access mechanisms (either CSMA/CA or TDMA) to transmit their data traffic. During an active monitoring frame, only one node is allowed to send probes on the wireless channel in order to estimate actively the quality of the wireless links established with nearby nodes and the interference which might be generated by external sources.

6.2.3.3 Power Saving (PS)

The goal of the Power Saving (PS) service module is to enable various power saving algorithms, such as NoA/ASPP [92], to be easily implemented by specifying helpful functional blocks and their interactions with other services or functions.

Thanks to the modularity and flexibility exposed by *mac80211++*, this PS service is easily implemented as a loadable module. For that, we define the following two functions: (*i*) *ps_policy()*, which is registered into the Function Handler. It incorporates the logic of the developed algorithms and stores information of the mechanism(s) in operation and its(their) state; (*ii*) *ps_management()*, which provides management logic to support the PS mechanisms being implemented.

A generic PS scheme might require the ability of triggering sleep/awake events. This action is ultimately performed in HW by setting the proper HW registers accordingly. We then specify a primitive to communicate to the immediately lower layer the notification to execute the chosen event: *drv_ps_notify()*: This is a notification primitive and requires drivers to provide its proper handling. Thus, we push all the "intelligence" to the upper layer, designing this way a hardware-agnostic PS framework. In order to support sleep/awake transitions typically required by power saving algorithms, it is still needed that drivers and firmware support sleep/awake events (issued by the previously mentioned primitives of the PS service).

6.2.3.4 Rate Adaptation

Most of the current *mac80211* drivers rely on rate control algorithms provided by the framework. These algorithms are encapsulated in independent kernel modules that are linked to the specific driver once a new device is being loaded. The naming convention of these modules is based on an *rc80211_* prefix, followed by the name of the algorithm. *mac80211* implements two rate adaptation schemes, Minstrel and PID, but also permits the drivers to implement specific rate adaptation mechanisms and register them upon device initialization to notify the *mac80211* framework that rate selection will be handled by the driver itself. One example of such drivers is ath9k for Atheros cards.

Despite the differences of these two approaches, they both use a common mechanism to interface with the *mac80211* framework. Specifically, the *rate_control_ops* callbacks are registered by the rate adaptation module to the framework. These design principles are illustrated in Fig. 6.4.

Given the platform's modularity and flexibility in allowing the integration of new rate adaptation schemes, we investigate how a collision aware rate control algorithm,

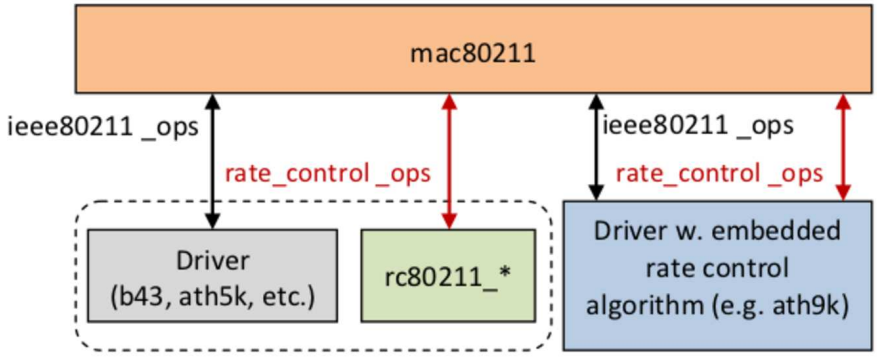


Figure 6.4: Interfacing Rate Control with mac80211.

H-RCA [93], could be implemented. This is motivated by the fact that current state-of-the-art algorithms do not distinguish losses due to packet collisions from losses that occur due to noise. The driver incorporating H-RCA takes an approach similar to the rate control algorithms of ath9k. To register the H-RCA algorithm to the *mac80211* framework, the driver is required to invoke *ieee80211_rate_control_register* function passing a reference to a *rate_control_ops* structure, which contains the handlers implemented by the algorithm.

6.3 Inter module Data Sharing for Flexible Wireless MAC

In this section we present an information management framework that greatly simplifies the data sharing among the components of a wireless stack designed according to a modular and flexible architecture. The designed information management framework enforces a data model that accounts for synchronous and asynchronous interactions, protection of values and includes a consistent memory management approach. A prototype implementation has been integrated in the wireless stack of the Linux kernel, as an extension of the mac80211 framework.

6.3.1 Section Overview

Wireless technologies are usually implemented according to standards, such as IEEE 802.11 [1]. Although the specifications include different configurable operation modes, existing implementations are designed to be comprehensive, one-size-fits-all and may fail to be effective in specific niche contexts or particular network situations. Actually, the real scenarios in which wireless technologies are used are very heterogeneous and the achieved performance is far from the "standard case". Thus, many researchers have worked to change the behavior of standard technologies in order to improve performance in terms of network capacity, transmission delay, etc. To increase the exploitation probability, many efforts have been devoted to design solutions with a limited impact on the existing standard implementation.

However the actual capability of implementing the proposed improvements depends on the availability of solutions based on flexible architectures and modular frameworks, where the wireless interface is conceived as a group of components that work together to provide the required functionalities. The resulting protocol stack is a (more or less complex) system in which many actors cooperate to the overall functioning.

The building up of a flexible and modular wireless interface requires the definition of a suitable hardware and software infrastructure, which enables the developers to exploit the new architecture and the new interfaces.

At the PHY level, a paradigm toward in this direction is represented by software defined radio (SDR). SDR proposes the implementation of the signal processing modules (filters, modulators, codecs, etc.) via software running on generic hardware. In this field, the popular JTRS Software Communication Architecture (SCA) framework [94] makes a clear separation between the "programmable" environment, the platform, and the protocol stack (i.e. the waveform), allowing for the portability of the communication stack. GNURadio [95] is also a software radio platform, which allows the combination of signal processing blocks for the generation of custom waveforms, using a customized, yet completely open, architecture.

At the MAC level, WLAN card vendors are switching from a "full-MAC" approach, in which all the MAC layer functions are left to the card's hardware/firmware, to a "soft-MAC" [96] design. Some low level primitives such as frame generation and

management are delegated to the driver, but in a vendor and platform dependent way, making modular functionality replacement an outstanding effort. Other approaches to extend the MAC functionalities make use of overlay software modules, such as MultiMAC [97] or the Overlay MAC Project [98], to switch between predefined MAC layer solutions, but the approach, inside the MAC level itself, cannot be deemed as modular.

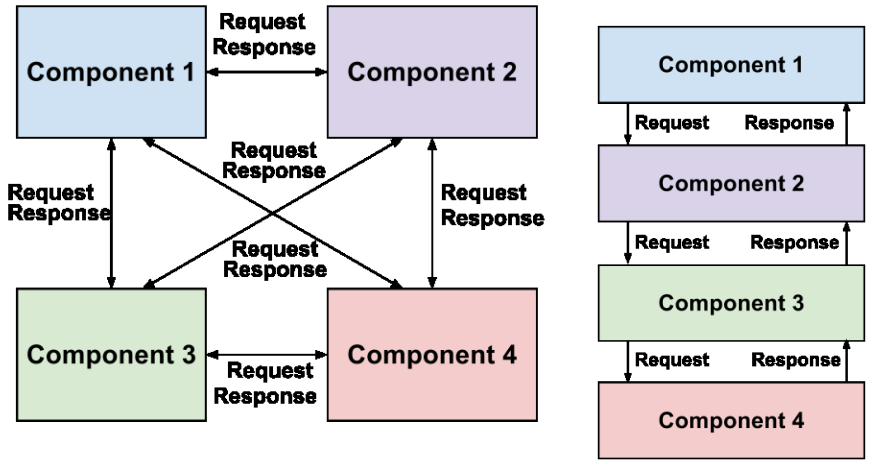
To extend the range of available primitives at the software level, some researchers have devised mixed FPGA and software radio based implementations of IEEE 802.11 MACs, such as CalRadio [99] and WARP [100], or FPGAs have been combined with old 802.11 cards [84]. Or have implemented open firmwares [101], still with the limitations imposed by the wireless card vendors design.

A critical issue of modular systems is clearly the memory management and communication among the various components, which normally is a limiting aspect of real complex software infrastructures. In fact, operating systems make available to developers several communication facilities, such as pipes, which associate interprocess communication channels to file structures [102], or M-BUS systems, i.e. middlewares that provide buses for the exchange of messages. The most notable example in this field is the D-BUS system [103].

But developing at the driver level, in kernel space, programmers don't have access to these facilities, nor to standard operating system interfaces such as file descriptors and sockets. Usually the different components of the kernel interact by accessing the same global memory areas, but often, e.g. in Linux, these areas are not made directly available to external kernel modules, which then need, in order to communicate with other components or modules, to call functions for the retrieval of pointers to global structures.

This section presents a solution for the management of information in a wireless protocol stack, designed according to a flexible and modular architecture. The information management task is committed to a dedicated component, the Data Gateway, which exposes an interface that allows to store and to retrieve data identified by unique keys. Moreover, the Data Gateway interface allows the components to protect the stored values: they can acquire an exclusive right on them, inhibiting write operations from the other components or partially limiting the value modification through the registration of specific callbacks. This section also describes

a Data Gateway module prototype implementation for the WLAN Linux protocol stack mac80211, evaluating its performance in terms of data access time performance and its exploitation in a video streaming scenario.



(a) Direct Connections (b) Stack configuration

Figure 6.5: Simple and Stack Architecture Solutions

6.3.2 Information Management Architecture

To handle the information sharing among components, it is possible to relay on different architectural patterns that offers various degrees of decoupling.

As represented in Figure 6.5, the simplest solution is to create direct links between the various components using direct communication. However, this solution causes a strong coupling among components and reduces the system flexibility making any possible change/addition in the system complicated.

Another strategy, that is widely used in protocol stacks, is to organize the components in levels, creating a hierarchy of connections. This solution reduces the degree of coupling between components leading to the classical requests and responses pattern.

To obtain the highest temporal and spatial decoupling degree is possible to use use the data-oriented architecture represented in Figure 6.6, in which the modules communicate via a "proxy" component: the Data Gateway.

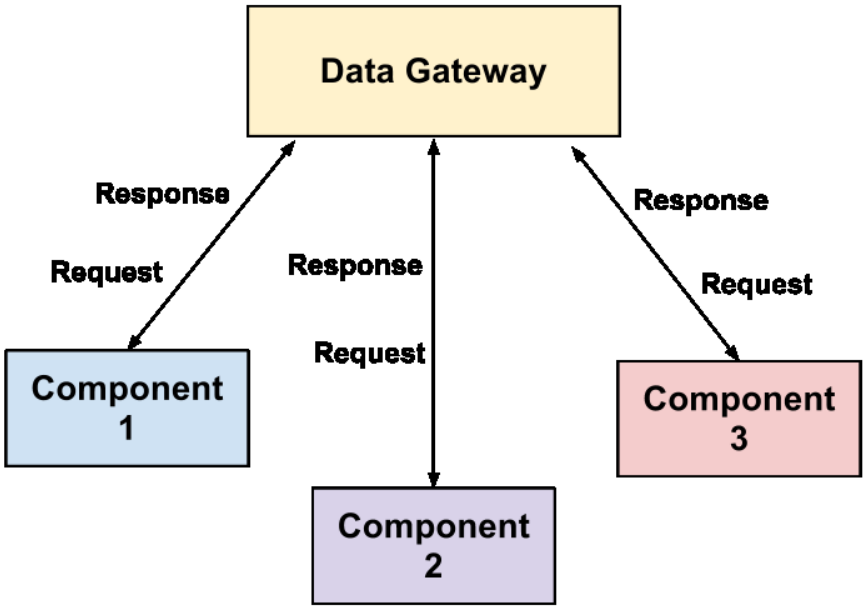


Figure 6.6: Gateway Architecture Solutions

The Data Gateway (DG) allows for information sharing among modules and it is responsible for solving conflicts which may arise when the components operate on the same set of data. The data sharing model, presented in this section, assumes that the data structures representing the system information are globally defined and that each has an assigned Unique Identifier (UID), which is notified to all the components.

The Data Gateway module offers three modes of information management, represented in Figure 6.7:

- Single writer multiple readers
- Multiple writers multiple readers
- Subscribe and notify

In the the Single writer multiple readers mode (SW), as represented in Figure 6.7, a single component is allowed to execute create/read/write/update (CRUD)

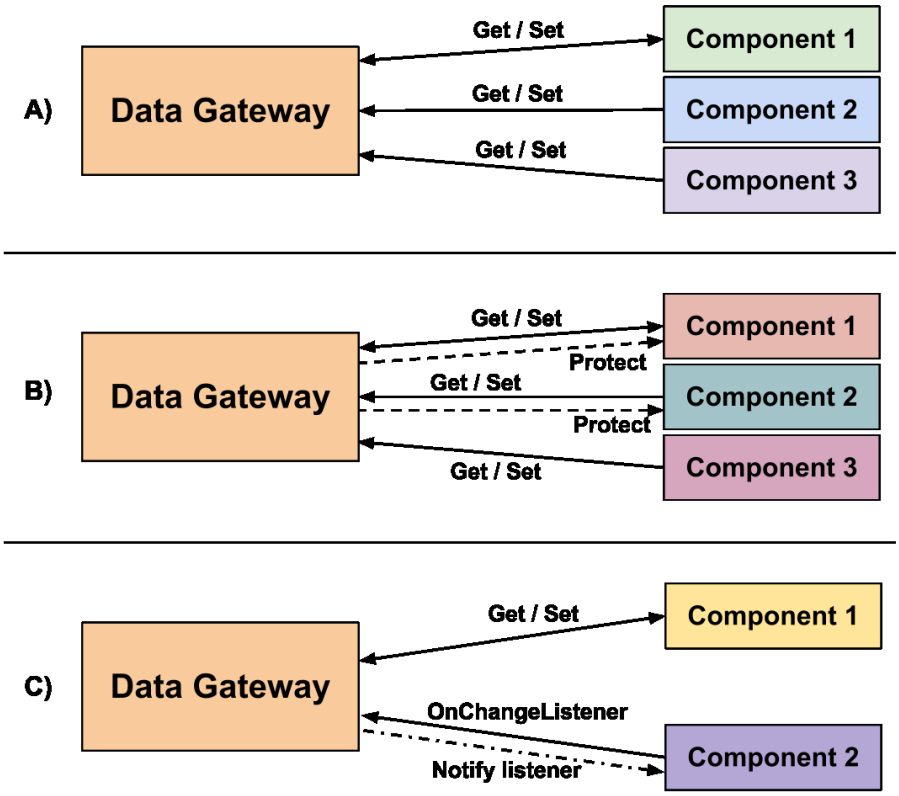


Figure 6.7: Data Gateway Interactions

operations while the others can only read the published information. The SW interaction model is typically used by components to manage information publication. Let's consider for example a measuring component of a wireless interface that continuously produces values that have to be made available to other components.

The Multiple Writers Multiple Readers (MW) interaction model allows the sharing of memory areas allowing to perform the CRUD operations on the data from every component, as reported in Figure 6.7. In order to enforce a simple form of control over the values, there is also the possibility for components to protect the stored information using callbacks that can temporarily inhibit the change or can only allow operations in accordance with predetermined logic. Consider the following example: two components need to control the value of the transmission power of a wireless card. A component may impose a limit on the maximum power that

can be set, e.g. according to the regulation of the specific country. To operate such a limitation, the component registers a protection callback which is responsible for checking and authorizing the change.

The Data Gateway also provides a push modality for information retrieval. As described in Figure 6.7, the component can register its interest in specific data. Then, when this data changes, the DG notifies the new value to all the registered components. This subscribe/notify interaction model is provided for both SW and MW data sharing modes.

6.3.3 CRUD operations

6.3.3.1 Single writer

In the Single Writer model of operation a component sets a value and receives a Modification Identifier (MID) by the Data Gateway. The MID can be used both for updating or deleting the stored data. The main operations can be schematized as follows:

- *createData(UID, value): MID;* operation that allows data initialization using a public UID. The operation produces the MID that can be later used for write and update operations.
- *changeData(MID, value): outcome;* This operation allows to set a new value for the data field identified by the MID, proving that the calling module is authorized for such operation.
- *clearData(MID): outcome;* This operation allows the initializing module to remove a data field from the DG.

6.3.3.2 Multiple writers

The multiple writers with challenge protection scheme is used when data has to be shared among multiple modules. In this case, all the system modules can read and write data. However, optionally, a module can register one or more consistencyVerification operations, to be verified before any value updates. The Data Gateway calls sequentially all the consistencyVerification operations registered at each attempt of

changing the data value. If all the consistency tests are passed, the old value is replaced by the new value.

The main operations for supporting the above behavior are defined as:

- *setValue(UID, data value): outcome;* This operation enables all the system modules to set a data field overriding the single writer protection.
- *deleteData(UID): outcome;* This method enables all the system modules to remove a data field from the DG by using the UID.
- *protectData(UID, consistencyVerification): outcome;* This method allows to register a consistency verification operation for a given data field.

6.3.3.3 Reading the Data

The Data Gateway offers both push and pull models for retrieving data through the following operations:

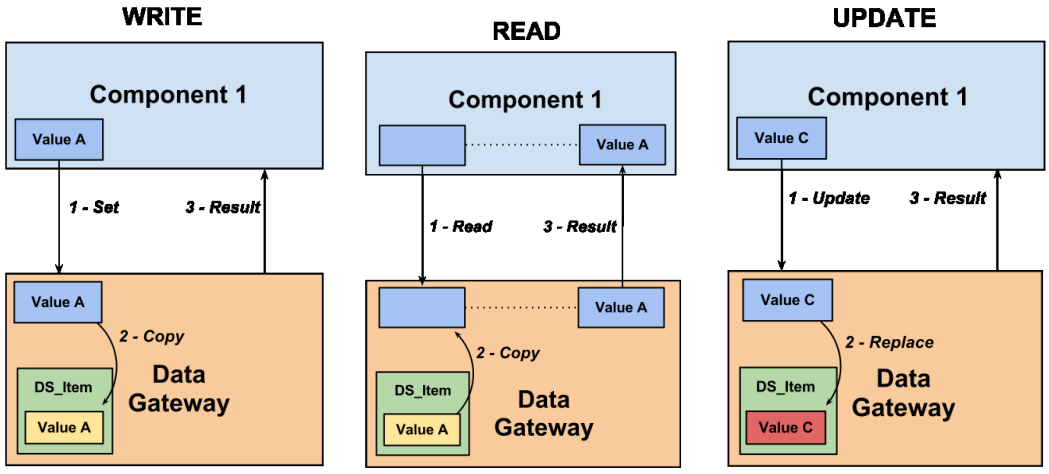
- *getData(UID)* operation allows a component to retrieve the value corresponding to the given unique identifier (UID).
- *onChangeListener(UID, changeListener)* operation is used by components to register a change listener function associated to the data field identified by the UID.

6.3.4 Memory Management Model

An important aspect of the Information Management Architecture is memory management. This section specifies how the DG deals with the stored values and how other modules can interact with the DG.

Three operations are described to illustrate how this issue has been addressed: the write, read and update operations. These define a clear distinction between the memory of the DG and the memory of the module. As a result, the deletion process gets simplified and potential issues, such as memory leaks or concurrency problems, are avoided.

Figure 6.8a represents the write operation. Component 1 passes the Value A to the DG using the set operation. The DG allocates memory to store value A and the related metadata. If the operation is completed successfully a positive response is returned.



(a) Write (b) Read (c) Update

Figure 6.8: Read, write and update interactions

The read operation is depicted in Figure 6.8b. The component 1 needs to access a previously stored value, thus it allocates enough memory space to store it and queries the DG using the key associated to the value. The DG copies the requested value, if available, in the allocated space and returns it to the requesting module.

Figure 6.8c describes the update operation. Now, in this case, a module changes a stored value by providing it to the DG. If the memory required for the new value is equal to the amount used to store the old value, then the value is simply substituted. On the contrary, if the new value exceeds the memory size, the memory cell is released and a new allocation for the new value occurs.

6.3.5 Linux Kernel Implementation

To prove the effectiveness of the proposed information management architecture we implemented the Data Gateway module in the Linux wireless mac80211 framework [85] that is used by all the modern IEEE 802.11 drivers. We first briefly describe the

mac80211 data management architecture and then the Data Gateway extension.

6.3.5.1 mac80211 Integration

mac80211 is a framework designed to provide soft-MAC functionality to the underlying drivers and cards. The actual implementation includes a "global" data structure *ieee80211_local* that is a general-purpose collection of information embedding other structures. For example, it includes the *ieee80211_hw* structure that contains the hardware specific information of the wireless card. Another contained structure is *ieee80211_ops* (the ieee80211 operations) that is used to access the driver operation from the mac80211 module. In addition, *ieee80211_local* contains the states of the wireless interface (suspended, resuming, started, etc.) and other management and configuration information. The mac80211 framework integrates the *ieee80211_local* structure in the Linux "net device" that is used by the Linux kernel to differentiate among the wireless interfaces. At run time each physical card is controlled by a driver that allocates a separate *ieee80211_local*, and the operations can occur independently on the different interfaces.

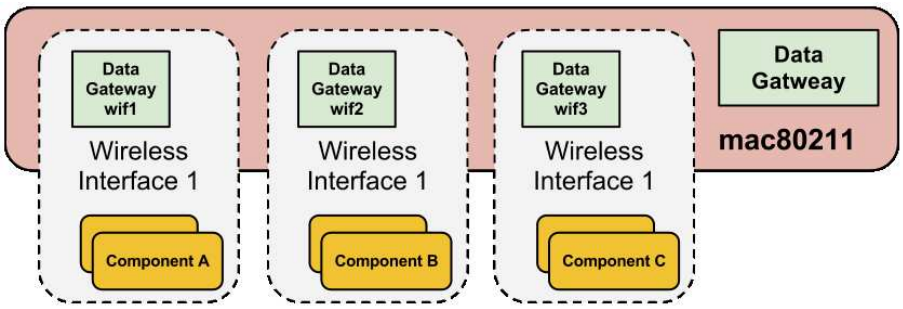


Figure 6.9: Multi Interface Repository

To extend the functionality of the IEEE 802.11 interfaces, developers have to change the *ieee80211_local* structure in order to save custom information. Moreover, since the *ieee80211_local* is specific to the single IEEE 802.11 interface, components dealing with multiple interfaces have to relay on different Linux kernel communication strategies.

Hence, to integrate the designed information management architecture in mac80211,

we have devised a practical way of extending the *ieee80211_local* structure: the implemented DG interface includes *ieee80211_local* as a parameter. When *ieee80211_local* is provided, the DG stores the data within that structure, creating separate data storage for each interface. Otherwise, if the actual interface "repository" is not provided, the DG module uses a global repository that can be accessed by all components.

6.3.5.2 Linux Module

The Data Gateway has been implemented, according to the interface presented in 6.3.3, as a Linux module that depends on mac80211 and exports a kernel wide interface, reported in Table 6.2.

All the operations receive the structure *ieee80211_local* as first parameter to select the repository and, as second parameter, an integer to identify the specific data that can be, respectively for MW or SW, the Unique Identifier (UID) or the Modification Identifier (MID). In addition, the function *ds_set* requires as input the data value and size, while the function *ds_protect* receives also the protection callback that accept as input parameters the *ieee80211_local* structure, the old value, the new value and the data UID.

The SW operations are implemented in the same way of the MW operations. The *dg_create_sw* function is used to gain the control over an UID and a unique integer MID is used to control the update and remove operations.

All the operations return an int value that allows to check if errors occur and are intrinsically designed to be thread safe.

Two operations are defined for of the push interface: *dg_on_change_listener_add* and *dg_on_change_listener_rem* that can be used respectively to register or unregister a notify operation. Both functions require as parameter the UID to select the observed data. The *dg_on_change_listener_add* operation returns a Listener Identifier (LID) associated to the registered listener. Later on this identifier can be used as parameter in *dg_on_change_listener_rem* to unregister that specific listener. Note that the notify operation is executed asynchronously in a Linux work.

Operation	Parameters
dg_set	struct ieee80211_local *local, int UID, void *data, unsigned int size
dg_remove	struct ieee80211_local *local, int UID
dg_protect	struct ieee80211_local *local, int UID, int (*protection)(void *)
dg_create_sw	struct ieee80211_local *local, int UID, void *data, unsigned int size
dg_update_sw	struct ieee80211_local *local, int MID, void *data, unsigned int size
dg_remove_sw	struct ieee80211_local *local, int MID
ds_get	struct ieee80211_local *local, int UID, void *data
dg_on_change_listener_add	struct ieee80211_local *local, int UID, void (*notify)(void *)
dg_on_change_listener_rem	struct ieee80211_local *local, int UID, int LID

Table 6.2: Implemented Data Gateway Operations

6.3.6 Data Access Performance

We provide the performance figures of the Data Gateway module in terms of access time to a stored value. The Data Gateway architecture is compared with the two other architectures discussed in Section 6.3.2: the simple architecture in which two modules are directly connected and a stack architecture with three layers (Figure 6.5).

All the modules are implemented as Linux kernel modules. The simple architecture is composed by two modules connected by a kernel exported function. The stack architecture comprises three modules connected hierarchically by means of exported functions. The Data Gateway based architecture includes the Data Gateway, a publishing module and a reading module. All the configurations perform a data

copy in response to the read call according to the memory management strategy described in Section 6.3.4.

The experiments are executed in a Linux PC with Vanilla kernel 3.4.6 (Ubuntu distribution), equipped with an Asus Motherboard P8Z68-V, a Processor Intel i7-2600@3.4GHz, 8GB of Kingston DDR3 RAM and a Corsair Force GT (V5 128GB) hard disk mounting an ext 4 file system.

Each architecture is tested with 380 sequential readings of 1 kbyte of data, allocated as a char* by the vmalloc Linux kernel function.

Results are reported in Figure 6.10 and show that the Direct connection and the Data Gateway architectures present almost the same access time. Hence the introduction of the key to access the data is almost negligible. Clearly the stack architecture has a longer access time due to the double redirection of the reading function.

We also note that the delay is almost constant for successive reading operations for all the three architectures and that the major contribution to the access time came from the memcpy function. This has been verified by testing the vmalloc and memcpy functions independently.

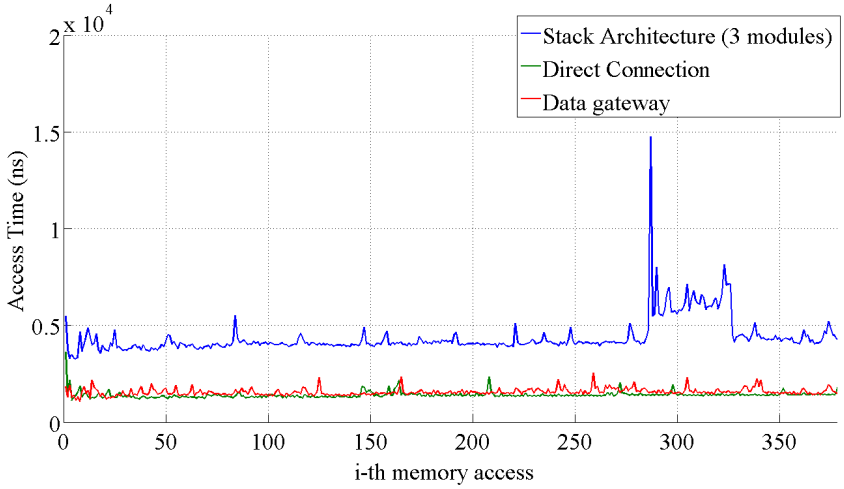


Figure 6.10: Access Time of the different architectures

We performed also a stress test of the Data Gateway architecture calculating the number of data retrievals supported by the module. We introduced a group of reading modules that access data stored in the Data Gateway continuously for

progressively increasing time intervals, up to 21 minutes. The results per second are reported in Table 6.3.

Number of Simultaneously Reading Modules	**Average number of access per module (access per second)**	**Cumulative number of access (access per second)**
1	721,098	721,098
2	521,838	1,043,676
3	382,435	1,147,305
5	191,846	959,232

Table 6.3: Stress test of the Data Gateway Module

The test uses the same machine described in Section 6.3.6. We found that the Data Gateway module supports about 721,098 requests per second (with maximum of 763,976 requests per second and a minimum of 710,270 requests per second).

6.3.7 Module Exploitation in the SVC video streaming scenario

To illustrate an use case of the Data Gateway module intercommunication facility we use the cross-layer architecture devised for the support of Scalable Video Streaming over WLANs presented in Chapter 5.

The architecture that can be deployed using the Data Gateway and [104] to support the video streaming is schematized in Figure 6.11, and comprises various MAC-level modules that seamlessly communicate: some monitor or estimate the WLAN parameters of interest and publish their values by using the Data Gateway facility, then the optimization logic module retrieves these values, computes the NALU queue configuration that maximizes the overall received video quality and stores the result of the computation in the Data Gateway. The NALU queue management module is, finally, responsible for retrieving the optimal configuration and installing the NALU queues accordingly.

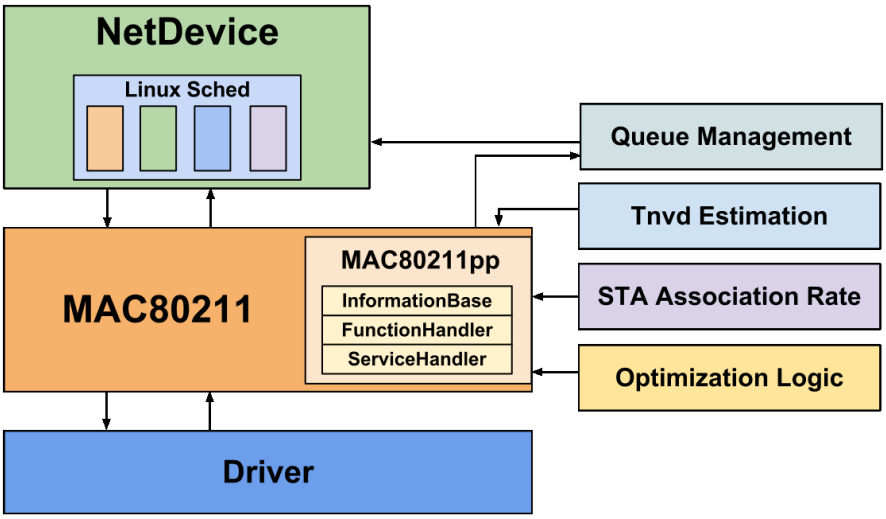


Figure 6.11: Data Gateway exploitation in a scalable video over WLAN scenario

6.4 Chapter Conclusions

The first part of this chapter has provided the specifications of a high level software architecture that proves the modularity, flexibility and virtualization to enhance user experience in wireless networks. Our specification has started from an existing framework in Linux, *mac80211*, which has been substantially extended in order to support the aforementioned features. This new framework, namely *mac80211++*, becomes our development platform. Specifically, the extension of the mac80211++ is twofold: (*i*) we intend to create a modular framework by untangling the existing mac80211, at the present at early stage of development. (*ii*) we have developed and implemented a Service Handling module that allows loading new services in real-time, based on the mac80211++ framework.

We have identified as candidate modules to be implemented key representative blocks such as SPS and Power Saving. In order to implement the virtualization, a new layer, called FLAVIAn, has been specified between the framework and the wireless drivers, exceeding the bounded capabilities of the driver on which the *mac80211* framework relies.

In the second part of this chapter, an information management mechanism for modular wireless drivers architectures has been presented. The proposed solution

allows for a loose coupling among MAC components by introducing a Data Gateway module that allows for information sharing. This new component exposes a flexible interface that enables symmetric or asymmetric information protection and push and pull data retrieval.

The Data Gateway has been implemented in the Linux kernel, and the implemented module has been exploited in an SVC video streaming over IEEE 802.11 networks scenario.

The work illustrated in Section 6.2 of this chapter has been presented at the Future Network & Mobile Summit 2012 with an article entitled "A modular, flexible and virtualizable framework for IEEE 802.11", co-authored by Pablo Salvador, Stefano Paris, Paul Patras, Yan Grunenberger, Xavier Pérez-Costa and Janusz Gozdecki.

Conclusions

We have seen how Wireless Community Networks (WCNs) have opened new study areas to networking researchers. In Chapter 1 we have illustrated the peculiarities of this phenomenon and the underlying technologies, focusing on the Optimized Link State Routing protocol (OLSR), very popular among WCNs. The development of this phenomenon might be limited by the lack of service support from the networking plane, which has been addressed by this thesis at different levels.

Chapter 2 has shown our approach to packet-droppers, i.e. misbehaving nodes, mitigation in WCNs. Compared to previous work on the subject, packet forwarding overhearing is not required, but instead steganographic techniques are employed to hide path-wide probes in normal user traffic. The information gathered from these probes is propagated on the whole network by a new OLSR protocol extension. Then, using a global reputation algorithm, packet-droppers are detected and avoided by adjusting routing metrics accordingly. Simulations show the effectiveness of this approach.

In Chapter 3 we add support at the routing level for distributed service discovery. We exploit OLSR's default message forwarding algorithm for the transport of multicast DNS packets, generated by end hosts and encapsulated in a new OLSR message type. Our implementation has been tested on the field and is now popular among WCN users.

In Chapter 4 we also devise an OLSR extension, targeted to support multimedia streaming services. Departing from the OBAMP overlay protocol for multicast delivery, we integrated its capabilities in the OLSR protocol mechanisms, reducing the overall signaling overhead. We believe that building a distribution tree that approximates the minimum spanning tree as much as possible and limiting the

protocol signaling provide a solid basis for making our solution scalable with the multicast group size. Our implementation has been tested and disseminated in WCNs.

In Chapter 5 we remain on the topic of multimedia streaming services and in particular in the context of the optimization of video streaming over IEEE 802.11 wireless local area networks. We exploit the adaptation properties of the H.264 scalable video (SVC) encoding to devise a packet scheduler that gives priority to the most important information of video streams. The lack of publicly available tools for the assessment of SVC video quality encouraged us to develop and publicly release our own set of tools (the SVEF evaluation framework). Our results, from real testbeds, show that an application-layer packet scheduler can already yield good performance in terms of overall quality of the delivered video streams. However, a generalization of the scenario to the case of the presence of background uplink traffic and a formalization of the problem lead us to find the optimal solution. This depends on a parameter, the non-video time (T_{nvd}), which is in practice hard to estimate. For this reason we developed a sub-optimal cross-layer scheduler whose performances we show to be close to our theoretical findings.

Chapter 6 is aimed at service support at the MAC layer. Researchers who wish to extend or integrate their solutions in wireless drivers have to patch the source code with inefficient or however inelegant hacks, which rarely find their way out of the scope of the specific research work. Introducing modularity, flexibility and support for virtualization in Linux wireless drivers are the targets of the mac80211++ framework. The existing Linux mac80211 layer has been analyzed and decomposed into smaller blocks, and support for the scheduling of MAC-layer services has been introduced, as well as a virtualization layer. Some services that have been effectively implemented using this framework.

In the same context, a component that allows for data exchange among modules is devised, with support for different modalities of operation, including data protection and a notification mechanism. This component is employed to devise a cross-layer scalable video streaming architecture.

Bibliography

[1] "IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements - Part 11: Wireless LAN MAC and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*.

[2] "Ninux.org wireless community network." `http://ninux.org/`.

[3] "Freifunk: non commercial open initiative to support free radio networks in the German region." `http://www.freifunk.net/`.

[4] "Athens wireless metropolitan network," September 2008.

[5] "Guifi.net." `http://guifi.net/en`.

[6] "Funkfeuer free net." `http://www.funkfeuer.at/`.

[7] "Pico peering agreement v1.0." `http://www.picopeer.net/PPA-en.html`.

[8] "Freenetworks.org peering agreement v1.1." `http://freenetworks.org/peering.html`.

[9] "Commons for open free & neutral network (OFNN)." `https://guifi.net/en/CommonsXOLN`.

[10] J. Camp and E. Knightly, "The ieee 802.11 s extended service set mesh networking standard," *Communications Magazine, IEEE*, vol. 46, no. 8, pp. 120–126, 2008.

[11] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)." IETF RFC 3626, October 2003.

[12] J. Chroboczek, "The Babel routing protocol." RFC 6126 (Experimental), 2011.

[13] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich, "Better Approach

To Mobile Ad-hoc Networking (B.A.T.M.A.N.).” IETF draft-openmesh-b-a-t-m-a-n-00, March 2008.

[14] “B.A.T.M.A.N. advanced.”

[15] A. Neumann, E. López, and L. Navarro, “An evaluation of bmx6 for community wireless networks,” in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, pp. 651–658, IEEE, 2012.

[16] M. Abolhasan, B. Hagelstein, and J. C.-P. Wang, “Real-world performance of current proactive multi-hop mesh protocols,” in *Proceedings of the 15th Asia-Pacific conference on Communications*, APCC’09, (Piscataway, NJ, USA), pp. 42–45, IEEE Press, 2009.

[17] D. Murray, M. Dixon, and T. Koziniec, “An experimental comparison of routing protocols in multi hop ad hoc networks,” in *Telecommunication Networks and Applications Conference (ATNAC), 2010 Australasian*, pp. 159 - 164, November 2010.

[18] “Wireless battle of the mesh.” `http://battlemesh.org/`.

[19] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, “A high-throughput path metric for multi-hop wireless routing,” in *MobiCom ’03: Proceedings of the 9th annual international conference on Mobile computing and networking*, (New York, NY, USA), pp. 134–146, ACM, 2003.

[20] A. Tønnesen, “Implementing and extending the optimized link state routing protocol,” Master’s thesis, UniK University Graduate Center - University of Oslo, 2004.

[21] S. D. Kamvar, M. T. Schlosser, and H. G. Molina, “The EigenTrust algorithm for reputation management in P2P networks,” in *WWW ’03: Proceedings of the 12th international conference on World Wide Web*, (New York, NY, USA), pp. 640–651, ACM, 2003.

[22] C. Adjih, D. Raffo, and P. Mühlethaler, “Attacks against OLSR: Distributed key management for security,” in *2005 OLSR Interop and Workshop*, (Ecole Polytechnique, Palaiseau, France), July 28–29 2005.

[23] C. Adjih, T. Clausen, P. Jacquet, A. Laouiti, P. Mühlethaler, and D. Raffo, “Securing the OLSR protocol,” in *Proceedings of the 2nd IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2003)*, (Mahdia,

Tunisia), June 25–27 2003.

[24] D. Raffo, C. Adjih, T. Clausen, and P. Mühlethaler, "An advanced signature system for OLSR," in *Proceedings of the 2004 ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '04)*, (Washington, DC, USA), pp. 10–16, ACM Press, October 25 2004.

[25] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, (New York, NY, USA), pp. 255–265, ACM, 2000.

[26] S. Buchegger and J.-Y. Le Boudec, "Performance analysis of the CONFIDANT protocol," in *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, (New York, NY, USA), pp. 226–236, ACM, 2002.

[27] P. Michiardi and R. Molva, "CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks," in *Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security*, (Deventer, The Netherlands, The Netherlands), pp. 107–121, Kluwer, B.V., 2002.

[28] K. Paul and D. Westhoff, "Context aware detection of selfish nodes in DSR based ad-hoc networks," in *proceedings of IEEE GLOBECOM*, pp. 178–182, 2002.

[29] S. Buchegger, C. Tissieres, and J. Le Boudec, "A test-bed for misbehavior detection in mobile ad-hoc networks - how much can watchdogs really do?," in *Proceedings of IEEE WMCSA 2004*, (English Lake District, UK), December 2004.

[30] E. Ayday, H. Lee, and F. Fekri, "An iterative algorithm for trust and reputation management," in *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pp. 2051 -2055, july 2009.

[31] W. Cordeiro, "OLSR modules for ns2 (OLSRMD, OLSRETX, OLSRML, OLSR)."

[32] C. Zouridaki, B. L. Mark, M. Hejmo, and R. K. Thomas, "E-hermes: A robust cooperative trust establishment scheme for mobile ad hoc networks," *Ad Hoc Networks*, vol. 7, pp. 1156–1168, August 2009.

[33] D. Djenouri and N. Badache, "On eliminating packet droppers in MANET: A modular solution," *Ad Hoc Networks*, vol. 7, pp. 1243–1258, August 2009.

[34] S. Yang, S. Vasudevan, and J. Kurose, "Witness-based detection of forwarding misbehaviors in wireless networks," in *Wireless Mesh Networks (WIMESH 2010), 2010 Fifth IEEE Workshop on*, pp. 1–6, IEEE, June 2010.

[35] S. Zakhary and M. Radenkovic, "Reputation-based security protocol for MANETs in highly mobile disconnection-prone environments," in *Proceedings of WONS 2010, Kranjska Gora, Slovenia*, February 2010.

[36] N. Duffield, F. L. Presti, V. Paxson, and D. Towsley, "Inferring link loss using striped unicast probes," in *IEEE INFOCOM 2001*, pp. 915–923, IEEE, April 2001.

[37] "Olsrd official website." `http://www.olsr.org/`.

[38] "Comunidade portuguesa de aficionados da tecnologia wireless." `http://unimos.net/`.

[39] J. Postel, "Domain name system structure and delegation." IETF RFC 1591, March 1994.

[40] N. W. Group, "Protocol standard for a NetBIOS service on a TCP/UDP transport: detailed specifications." IETF RFC 1002, March 1987.

[41] B. A. D. Thaler and L. Esibov, "Link-local multicast name resolution (LLMNR)." RFC 4795 (Informational), January 2007.

[42] Cheshire and Krochmal, "Multicast DNS." draft-cheshire-dnsext-multicastdns-07, September 2008.

[43] M. Rimondini, "Emulation of computer networks with netkit," Tech. Rep. RT-DIA-113-2007, Dipartimento di Informatica e Automazione, Roma Tre University, January 2007.

[44] A. Detti and N. Blefari-Melazzi, "Overlay, Borůvka-based, ad-hoc multicast protocol: description and performance analysis," *Wireless Communications and Mobile Computing*, 2007.

[45] M. Ge, S. V. Krishnamurthy, and M. Faloutsos, "Application versus network layer multicasting in ad hoc networks: the ALMA routing protocol," *Elsevier Ad Hoc Networks Journal*, vol. 4, no. 2, pp. 283–300, 2006.

[46] S. J. Lee, W. Su, and M. Gerla, "On-demand multicast routing protocol in

multihop wireless mobile networks," *ACM/Baltzer Mobile Networks and Applications*, vol. 7, no. 6, pp. 441–453, 2002.

[47] A. Detti, C. Loreti, and R. Pomposini, "Overlay Borůvka-based ad hoc multicast protocol - demonstration," in *IFIP Med-Hoc-Net 2006 - demo session*, 2006.

[48] R. Perlman, "An algorithm for distributed computation of a spanning tree in an extended LAN," *ACM SIGCOMM Computer Communication Review 15*, vol. 15, pp. 44–53, Septempber 1985.

[49] I. T. Union, "ITU-T recommendation H.264: Advanced video coding for generic audiovisual services," November 2007.

[50] M. van der Schaar, S. Krishnamachari, S. Choi, and X. Xu, "Adaptive cross-layer protection strategies for robust scalable video transmission over 802.11 WLANs," *Selected Areas in Communications, IEEE Journal on*, vol. 21, pp. 1752 - 1763, dec. 2003.

[51] R. Kuschnig, I. Kofler, M. Ransburg, and H. Hellwagner, "Design options and comparison of in-network h.264/svc adaptation," *Journal of Visual Communication and Image Representation*, vol. 19, no. 8, pp. 529 – 542, 2008. Special issue: Resource-Aware Adaptive Video Streaming.

[52] L. Han, D. Raychaudhuri, H. Liu, and K. Ramaswamy, "Cross layer optimization for scalable video multicast over 802.11 WLANs," in *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, vol. 2, pp. 838 - 843, jan. 2006.

[53] Y. P. Fallah, P. Nasiopoulos, and H. Alnuweiri, "Efficient transmission of H.264 video over multirate IEEE 802.11e WLANs," *EURASIP J. Wirel. Commun. Netw.*, vol. 2008, pp. 11:1–11:14, January 2008.

[54] I. Kofler, M. Prangl, R. Kuschnig, and H. Hellwagner, "An H.264/SVC-based adaptation proxy on a WiFi router," in *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '08, (New York, NY, USA), pp. 63–68, ACM, 2008.

[55] R. Kuschnig, I. Kofler, M. Ransburg, and H. Hellwagner, "Design options and comparison of in-network H.264/SVC adaptation," *Journal of Visual Communication and Image Representation*, vol. 19, no. 8, pp. 529 – 542, 2008. Special issue: Resource-Aware Adaptive Video Streaming.

[56] M. Eberhard, L. Celetto, C. Timmerer, E. Quacchio, H. Hellwagner, and F. S. Rovati, "An interoperable streaming framework for scalable video coding based on mpeg-21," in *Visual Information Engineering, 2008. VIE 2008. 5th International Conference on*, pp. 723 -728, 2008.

[57] "IEEE standard 802.11-2007, IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks-specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications;," June 2007.

[58] A. Kamerman and L. Monteban, "WaveLAN-II: A high-performance wireless LAN for the unlicensed band," *Bell Labs Technical Journal*, vol. 2, pp. 118–133, Aug. 1997.

[59] K. Ramachandran, H. Kremo, M. Gruteser, and P. Spasojević, "Scalability analysis of rate adaptation techniques in congested ieee 802.11 networks: An orbit testbed comparative study," in *in Proc. of IEEE WoWMoM*, 2007.

[60] C. H. Foh, Y. Zhang, Z. Ni, J. Cai, and K. N. Ngan, "Optimized cross-layer design for scalable video transmission over the IEEE 802.11e networks," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, pp. 1665 -1678, december 2007.

[61] "Joint scalable video model - reference software." `http://ip.hhi.de/imagecom\_G1/savce/downloads/SVC-Reference-Software.htm`.

[62] T. Zahariadis, "Astrals project presentation," September 2007.

[63] S. W. Y. W. T. Schierl and A. Eleftheriadis, "Rtp payload format for SVC video." draft-ietf-avt-rtp-svc-17, February 2009.

[64] J. Kim, S. Kim, S. Choi, and D. Qiao, "CARA: Collision-Aware Rate Adaptation for IEEE 802.11 WLANs," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–11, Apr. 2007.

[65] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan, "Robust rate adaptation for 802.11 wireless networks," in *Proceedings of the 12th annual international conference on Mobile computing and networking*, MobiCom '06, (New York, NY, USA), pp. 146–157, ACM, 2006.

[66] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance

anomaly of 802.11b," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 2, pp. 836 - 843 vol.2, march-3 april 2003.

[67] H. K. Lee, V. Hall, K. H. Yum, K. I. Kim, and E. J. Kim, "Bandwidth estimation in wireless lans for multimedia streaming services," *Adv. MultiMedia*, vol. 2007, pp. 9–9, January 2007.

[68] C. Sarr, C. Chaudet, G. Chelius, and I. G. Lassous., "A node-based available bandwidth evaluation in IEEE 802.11 ad hoc networks," in *Proceedings of the 11th International Conference on Parallel and Distributed Systems - Workshops - Volume 02*, ICPADS '05, (Washington, DC, USA), pp. 68–72, IEEE Computer Society, 2005.

[69] S. H. Shah, K. Chen, K. Nahrstedt, and I. Introduction, "Available bandwidth estimation in IEEE 802.11-based wireless networks," 2003.

[70] M. N. Nielsen, K. Ovsthus, and L. Landmark, "Field trials of two 802.11 residual bandwidth estimation methods," in *Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference on*, pp. 702 -708, oct. 2006.

[71] D. Wu, Y. T. Hou, and Y.-Q. Zhang, "Scalable video coding and transport over broadband wireless networks," *Proceedings of the IEEE*, vol. 89, no. 1, pp. 6-20, 2001.

[72] H. Schwarz and M. Wien, "The scalable video coding extension of the H.264/AVC standard [standards in a nutshell]," *Signal Processing Magazine, IEEE*, vol. 25, no. 2, pp. 135-141, 2008.

[73] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP friendly rate control (TFRC): protocol specification)." IETF RFC 3448, January 2003.

[74] O. I. Hillestad, A. Perkis, V. Genc, S. Murphy, and J. Murphy, "Adaptive H.264/MPEG-4 SVC video over IEEE 802.16 broadband wireless networks," in *Packet Video 2007*, pp. 26-35, 2007.

[75] J. Viéron and C. Guillemot, "Real-time constrained TCP-compatible rate control for video over the internet," *Multimedia, IEEE Transactions on*, vol. 6, no. 4, pp. 634-646, 2004.

[76] T. Schierl, T. Stockhammer, and T. Wiegand, "Mobile video transmission using scalable video coding," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 9, pp. 1204-1217, 2007.

[77] H.-L. Chen, P.-C. Lee, and S.-H. Hu, "Improving scalable video transmission over IEEE 802.11e through a cross-layer architecture," in *Wireless and Mobile Communications, 2008. ICWMC '08. The Fourth International Conference on*, pp. 241-246, 2008.

[78] Y. Fallah, H. Mansour, S. Khan, P. Nasiopoulos, and H. Alnuweiri, "A link adaptation scheme for efficient transmission of h.264 scalable video over multi-rate WLANs," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 7, pp. 875-887, 2008.

[79] R. Agrawal and V. Subramanian, "Optimality of certain channel aware scheduling policies," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 40, pp. 1533–1542, The University; 1998, 2002.

[80] T. Ozcelebi, M. Sunay, M. Civanlar, and A. Tekalp, "Application-layer QoS fairness in wireless video scheduling," in *Image Processing, 2006 IEEE International Conference on*, pp. 1673-1676, 2006.

[81] X. Ji, J. Huang, M. Chiang, G. Lafruit, and F. Catthoor, "Scheduling and resource allocation for SVC streaming over OFDM downlink systems," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 10, pp. 1549-1555, 2009.

[82] G. Bianchi, A. Detti, P. Loreti, C. Pisa, F. Proto, W. Kellerer, S. Thakolsri, and J. Widmer, "Application-aware h.264 scalable video coding delivery over wireless lan: Experimental assessment," 2009.

[83] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 3, pp. 535-547, 2000.

[84] G. Bianchi, A. Di Stefano, C. Giaconia, L. Scalia, G. Terrazzino, and I. Tinnirello, "Experimental assessment of the backoff behavior of commercial IEEE 802.11b network cards," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 1181-1189, 2007.

[85] Linux kernel mac80211 framework for wireless device driver. http://linuxwireless.org/en/developers/Documentation/mac80211.

[86] FLAVIA Project (Flexible Architecture for Internet Access). http://www.ict-flavia.eu/.

[87] "IEEE Standard for Information technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN MAC and PHY Specifications Amendment 5: Enhancements for Higher Throughput," *IEEE Std 802.11n-2009*.

[88] G. Bhanage, D. Vete, I. Seskar, and D. Raychaudhuri, "SplitAP: Leveraging Wireless Network Virtualization for Flexible Sharing of WLANs," in *IEEE GLOBECOM*, pp. 1 -6, dec. 2010.

[89] T. Hamaguchi, T. Komata, T. Nagai, and H. Shigeno, "A Framework of Better Deployment for WLAN Access Point Using Virtualization Technique," in *IEEE WAINA*, pp. 968 -973, Apr. 2010.

[90] L. Xia, S. Kumar, X. Yang, P. Gopalakrishnan, Y. Liu, S. Schoenberg, and X. Guo, "Virtual WiFi: bring virtualization from wired to wireless," in *Proc. of the 7th ACM SIGPLAN/SIGOPS*, VEE '11, (Newport Beach, California, USA), pp. 181–192, 2011.

[91] G. Aljabari and E. Eren, "Virtualization of wireless LAN infrastructures," in *IEEE IDAACS*, vol. 2, pp. 837 -841, Sept. 2011.

[92] D. Camps-Mur, X. Pérez-Costa, and S. Sallent-Ribes, "Designing energy efficient access points with Wi-Fi Direct," *Elsevier Comput. Netw.*, vol. 55, pp. 2838–2855, Sept. 2011.

[93] K. Huang, K. R. Duffy, and D. Malone, "H-RCA: 802.11 Collision-aware Rate Control," *Technical report, Hamilton Institute*, 2011.

[94] C. Aguayo Gonzalez, C. Dietrich, and J. Reed, "Understanding the software communications architecture," *Communications Magazine, IEEE*, vol. 47, pp. 50 -57, september 2009.

[95] E. Blossom, "Gnu radio: tools for exploring the radio frequency spectrum," *Linux journal*, vol. 2004, no. 122, p. 4, 2004.

[96] M. Neufeld, J. Fifield, C. Doerr, A. Sheth, and D. Grunwald, "Softmac-flexible wireless research platform," in *Proc. HotNets-IV*, 2005.

[97] C. Doerr, M. Neufeld, J. Fifield, T. Weingart, D. C. Sicker, and D. Grunwald, "Multimac-an adaptive mac framework for dynamic radio networking," in *IEEE DySPAN*, 2005.

[98] A. Rao and I. Stoica, "An overlay mac layer for 802.11 networks," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pp. 135–148, ACM, 2005.

[99] A. Jow, C. Schurgers, and D. Palmer, "Calradio: a portable, flexible 802. 11 wireless research platform," in *International Conference On Mobile Systems, Applications And Services: Proceedings of the 1 st international workshop on System evaluation for mobile platforms*, vol. 11, pp. 49–54, 2007.

[100] A. Khattab, J. Camp, C. Hunter, P. Murphy, A. Sabharwal, and E. W. Knightly, "Warp: a flexible platform for clean-slate wireless medium access protocol design," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 12, no. 1, pp. 56–58, 2008.

[101] P. Gallo, F. Gringoli, and I. Tinnirello, "On the flexibility of the IEEE 802.11 technology: Challenges and directions," in *Future Network Mobile Summit (FutureNetw), 2011*, pp. 1 -10, june 2011.

[102] M. Khambatti-Mujtaba, "Named pipes, sockets and other ipc,"

[103] R. Love, "Get on the d-bus," *Linux Journal*, vol. 2005, no. 130, p. 3, 2005.

[104] P. Salvador, S. Paris, C. Pisa, P. Patras, Y. Grunenberger, X. Perez-Costa, and J. Gozdecki, "A modular, flexible and virtualizable framework for IEEE 802.11," in *Future Network Mobile Summit (FutureNetw), 2012*, pp. 1 -8, july 2012.

Appendix A: Publications

- "Application-aware H.264 Scalable Video Coding delivery over Wireless LAN: Experimental assessment" - Bianchi, G.; Detti, A.; Loreti, P.; Pisa, C.; Proto, F.S.; Kellerer, W.; Thakolsri, S.; Widmer, J.; Cross Layer Design, 2009. IWCLD '09. Second International Workshop on 11-12 June 2009 Page(s):1 - 6 Digital Object Identifier 10.1109/IWCLD.2009.5156512

- "The OLSR mDNS Extension for Service Discovery" - Proto, F.S.; Pisa, C.; Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops '09. 6th Annual IEEE Communications Society Conference on 22-26 June 2009 Page(s):1 - 3 Digital Object Identifier 10.1109/SAHCNW.2009.5172965

- "SVEF: an Open-Source Experimental Evaluation Framework for H.264 Scalable Video Streaming" - Detti, A.; Bianchi, G.; Pisa, C.; Proto, F.S.; Loreti, P.; Kellerer, W.; Thakolsri, S.; Widmer, J.; Computers and Communications, 2009. ISCC 2009. IEEE Symposium on 5-8 July 2009 Page(s):36 - 41 Digital Object Identifier 10.1109/ISCC.2009.5202390

- "Cross-layer H.264 Scalable Video Downstream Delivery Over WLANs" - Bianchi, G; Detti, A; Loreti P; Pisa, C; Thakolsri, S.; Kellerer, W; Widmer, J.C.; World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a , vol., no., pp.1-9, 14-17 June 2010 Digital Object Identifier 10.1109/WOWMOM.2010.5534890

- "Implementation of the OBAMP overlay protocol for multicast delivery in OLSR wireless community networks" - Proto, F.S.; Pisa, C.; World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International

Symposium on a , vol., no., pp.1-3, 14-17 June 2010 Digital Object Identifier 10.1109/WOWMOM.2010.5534941

- "A framework for Packet-Droppers Mitigation in OLSR Wireless Community Networks" - Proto, F.S.; Detti, A.; Pisa, C.; Bianchi, G.; IEEE ICC 2011 5-9 June, Kyoto.

- "A modular, flexible and virtualizable framework for IEEE 802.11" - Salvador, P.; Paris, S.; Pisa, C.; Patras, P.; Grunenberger, Y.; Perez-Costa, X.; Gozdecki, J., Future Network & Mobile Summit (FutureNetw), 2012 Publication Year: 2012 , Page(s): 1 - 8