13

Wireless Software and Hardware Platforms for Flexible and Unified Radio and Network Control (WiSHFUL)

Nicholas Kaminski[1], Spilios Giannoulis[2], Ilenia Tinnirello[3], Peter Ruckebusch[2], Piotr Gawlowicz[4], Domenico Garlisi[3], Jan Bauwens[2], Anatolij Zubow[4], Robin Leblon[5], Pierluigi Gallo[3], Ivan Seskar[6], Luiz, A. DaSilva[1], Sunghyun Choi[7], Jose de Rezende[8] and Ingrid Moerman[2]

[1]Trinity College Dublin
[2]iMinds-Ghent University
[3]CNIT
[4]Technische Universtät Berlin
[5]nCentric Europe
[6]Rutgers University
[7]Seoul National University
[8]Universidade Federal do Rio Janerio

13.1 Introduction

In the last years, we have assisted to an impressive evolution of wireless technologies for short distance communication (like IEEE 802.11, IEEE 802.15.4. Bluetooth Low Energy, etc.) due to the need of coping with the heterogeneous requirements of emerging applications, such as Internet of things, the Industry 4.0, the Tactile Internet, the ambient assistant living, and so on. Indeed, for optimizing the technology performance in these scenarios, it is often required to support some forms of *protocol adaptation*, by allowing the dynamic reconfiguration of protocol parameters and the dynamic activation of optional mechanisms, or some targeted *protocol extensions*. In both cases, prototyping, testing and experimentally validating potential solutions is a complex task, which generally requires significant time and resource investment. On one side, off-the-shelf wireless interfaces are based on radio chips which implement only the obligatory parts of the standards and arbitrarily selected optional parts, with only partially documented interfaces and with drivers being either closed or limited in functionality. On the other side, many powerful Software Defined Radio (SDR) platforms, while offering excellent flexibility at the physical layer, typically have limited performance and lack high-level specifications and programming tools as well as standard APIs for developing protocols.

Consequently, testing of new solutions often proves problematic, as experimenters can only rely on the limited optimization space enabled by the drivers, or on *open* software architectures where many functionalities have to be written from scratch and are tightly dependent on the specific hardware platform. In many cases, different experimentation platforms have to be considered for working on specific optimizations, because each platform supports a different level of complexity and controllability. This heterogeneity further slows down the innovation process, because experimenters have to be familiar with platform-specific architectures and programming tools before prototyping their solutions.

To overcome the aforementioned shortcomings and reduce the threshold for experimentation, we propose a novel approach within the European project WiSHFUL [1]. The project main goal is the design and development of a software architecture enabling a flexible radio and network control of heterogeneous experimentation platforms, based on standardized wireless technologies and SDRs, through unified programming interfaces. More specifically, the architecture is devised to allow:

- *Maximal exploitation of radio functionalities* available in current radio chips, as opposed to today's radio drivers that restrict radio functionality. For example today's radio drivers for IEEE 802.11 do not support TDMA (Time Division Multiple Access) operation, while the hardware perfectly supports it.
- *Clean separation between radio control and protocol logic*, as opposed to today's monolithic implementations, which do not allow to work separately on the logic for enabling specific protocol features and the definition of these features.

To frame this effort, several driving scenarios were identified to capture the challenges associated with the *increasing density* and *heterogeneity* of wireless devices in a concrete and tangible manner. These scenarios directly present a set of relevant and significant requirements for developing the functionalities required by the WiSHFUL control framework in order to investigate the challenges of future wireless systems experimentation. Each showcase focuses on a different source for inter-device and inter-technology interference and displays a scenario, which requires novel experimentation functionalities.

Following the definition of this set of motivating scenarios, an architecture is presented to support future wireless experimentation. This architecture is constructed to address the requirements of the tangible scenarios, capturing key challenges of future systems while allowing for extensions to support investigation of as yet unforeseen challenges.

13.2 Background

The need for fine-grained control of communication networks is well demonstrated by the interest of the scientific community in solutions that enable *software defined networking*, (SDN). *OpenFlow* [2], for instance, is a good example of an SDN-enabler because it allows researchers to control routers, without knowing the internals of vendor-specific implementations. OpenFlow focuses on controlling the forwarding rules between devices (e.g. switches, routers and wireless access points) connected by means of pre-installed links (usually wired). However, it does not explicitly deals with wireless links, where conditions change over time and strongly depend on interference and propagation conditions. Indeed, for wireless links the use of forwarding functionalities, which have inspired the match/action abstraction used for wired link, cannot be adequate for capturing the inter-link and inter-network dependencies, despite the fact that some extensions have been proposed, e.g. OpenRadio, for classifying the traffic flows on the basis of PHY-related fields and configuring the transmission power of the links. Actually, a closer look reveals that the wireless community has arguably anticipated, if not even inspired, the wired networking shift towards centralized controllers, for example with the CAPWAP protocol (Control And Provisioning of Wireless Access Points) [3] for the remote control of wireless access points. However, the CAPWAP control model was based on parametric control of technology-specific configuration parameters. WiSHFUL goal is more forward-looking, and resides in i) devising a generic programming model for wireless devices and wireless links, based on technology-independent programming abstractions and ii) showing that they can be handled with a network control framework which include global and local controllers.

To accomplish this goal, WiSHFUL pushes towards the identification of viable abstractions for radio behavior, by integrating four different platforms exposing high-level programming models for heterogeneous wireless technologies while taking into account the emerging solutions and standardization work concerning *reconfigurable radio systems* (ETSI-RRS) [4]. The four supported platforms are: *Wireless MAC Processor* (WMP) for IEEE-802.11 radios [5], *Time-Annotated Instruction Set Computer* (TAISC) for IEEE-802.15.4 radios [6], the *Implementing Radio in Software* (IRIS) for SDRs [7] and finally the popular Atheros chip based cheap-of-the-self wireless cards running the ATH9k driver [8]. Moreover, the WiSHFUL control framework complements OpenFlow, by enabling the coexistence of local and global controllers devised to react to the network events at different time scales. In the next phase WiSHFUL also plans to extend to support cross-layer control from the network layer and above as well, providing SDN like characteristics regarding the management and fine-tuning of control knobs ranging from routing protocols parameters and realization of flow control to transport layer parameters like TCP window for example. GITAR [9] supports the cross layer parameter control, especially in the context of WSNs, but can be used in all platforms that are supported within WiSHFUL as a cross layer parameter management component.

13.3 Motivating Scenarios

The emerging wireless ecosystem is characterized by a heterogeneous mix of technologies, operators, and service providers attempting to coexist in a single environment, and featuring a high-density deployment of wireless devices. High heterogeneity in device capabilities (in terms of spectral bands, coverage, management functionalities, networking models, etc.) combined with limited open, vendor-independent configuration interfaces complicate achieving the often conflicting goals of independent providers and integration of technologies to provide coherent service. Indeed, wireless devices often employ multiple radio interfaces, spanning over several standards (such as LTE, Wi-Fi, ZigBee and Bluetooth) or offering more esoteric capabilities in the form of programmable interfaces, based on software defined radio (SDR) techniques.

Experimental-driven research is essential for analyzing the performance of this eco-system, because of the difficulty in simulating or modelling the interactions between heterogeneous technologies, protocol configurations, environments and network operators. We consider some exemplary scenarios in order to identify the *functional requirements* and *control models* required for testing optimization and coexisting strategies dealing with the complexity of the wireless eco-system. From the analysis of these scenarios, we identified two main groups of functional requirements: i) configuring the radio of each wireless node, in terms of set-up of physical transmission parameters, bandwidth allocation, medium access schemes and prioritization mechanisms for different transmission queues, ii) configuring the network-wide policies for dealing with different traffic flows, by defining logical links and paths between nodes, mapping of traffic flows into transmission queues, performing flow control among multiple links and interfaces, etc. Moreover, it is required to introduce monitoring functionalities at different levels for collecting statistics about the radio performance and the local channel views.

13.3.1 Interference Management among Overlapping Cells

In dense wireless networks, co-channel interference is a fundamental problem, especially in the case of WiFi technologies working on the unlicensed ISM bands characterized by the availability of a few orthogonal channels and by the coexistence of multiple independent networks. Ultimately, this scenario examines questions related to the dynamic control of multiple Access Points in a coordinated manner. A possible solution for controlling co-channel interference is working on the adaptation of contention parameters and transmission opportunities used by co-located APs. Some research work has suggested the use of airtime as a metric to quantify the channel resources that are granted to each AP. The airtime is the sum of the channel holding times used by a given cell during a reference time interval. To enforce any decision about the network configuration, it is also required to represent a network global view, by considering the interference relationships among the APs, which depend on the specific location of the stations. In particular, it is required to detect hidden nodes, which may experience severe collision rates.

Consider the example network given in Figure 13.1. This scenario assumes four active flows in the following QoS classes – the first three are best effort (BE) while the last one is voice. Each flow is assigned to one of the two APs. Furthermore, let us assume that AP1 and AP2, are operating on the same radio channel. In such a case a cell-edge user like node STA2 may suffer from interference due to hidden node, i.e. the downlink traffic from AP1 to STA2 will collide with traffic originated from AP2. By solving the hidden node problem, the performance of all nodes in neighboring wireless networks can be improved.

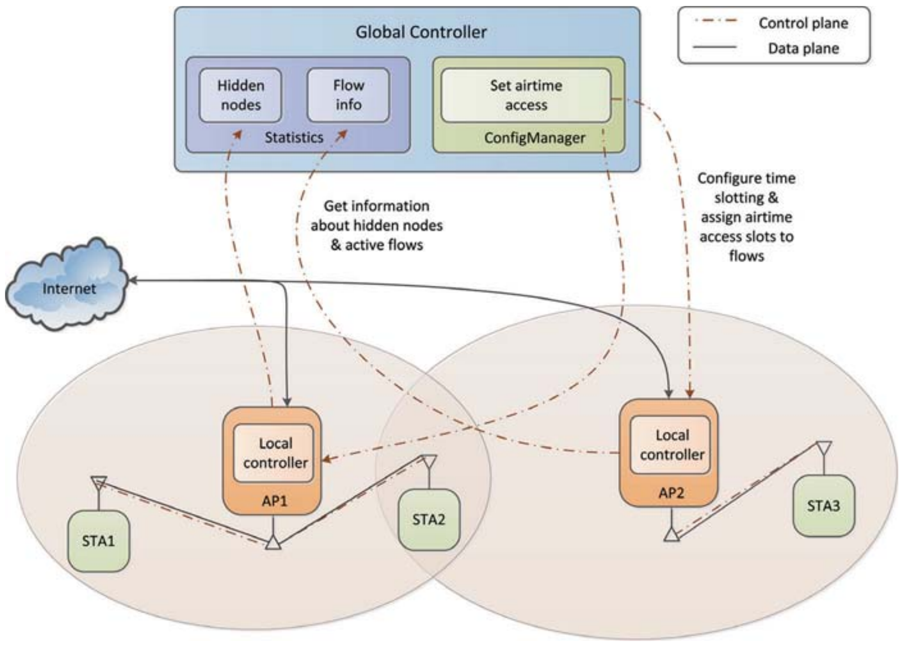


Figure 13.1 Traffic-aware 802.11 airtime management scenario.

The challenges of this scenario may be addressed by monitoring the performance of each AP. Such monitoring would make degradation associated with inefficient management clear, thus allowing rescheduling of flows to avoid interference. To accomplish this goal, global monitoring of network performance would be required. Specifically, some control entity would need the ability to monitor the active flows for detecting hidden nodes and to define appropriate channel access patterns and assign airtimes for solving the hidden node problem by dividing the competing flows in the time plane. Furthermore, tight time synchronization between APs is required for time-slotting airtime. This may be achieved by usage of PTP running over backbone interfaces.

13.3.2 Co-existence of Heterogeneous Technologies

In dense wireless networks, the co-existence of heterogeneous technologies using the same wireless resources is challenging. Indeed, although technologies working on ISM bands intrinsically deal with mechanisms for managing interference, such as carrier sense, adaptive modulations, spreading solutions, etc., it has been demonstrated that they can experience severe throughput degradation in case of coexistence of heterogeneous links, because of asymmetries in recognizing other technologies and reacting to their presence. A central controller could overcome these problems, by supporting a harmonized spectrum allocation across separate wireless technologies. This will enhance the performance in both networks and make the quality of service (QoS) characteristics (such as throughput, latency and reliability) more predictable.

As a reference example, we consider the coexistence between IEEE-802.11 (Wi-Fi in 2.4 GHz band) and IEEE-802.15.4e (time-slotted channel hoping, TSCH) illustrated in Figure 13.2. The simultaneous operation of both networks in close proximity will inevitably lead to performance degradation due to cross-technology interference. This is because of contention-free explicit scheduling of radio resources in TSCH and the unreliability of carrier-sensing (listen-before-talk) mechanism used in Wi-Fi as far as detecting IEEE 802.15.4 transmissions is concerned, rendering Wi-Fi unable to sense any wireless transmission of the other technology. The QoS in both networks can be increased by making them aware of each other.

One can imagine multiple co-existence schemes for Wi-Fi and TSCH. Some basic schemes can be implemented by only modifying the sensor network. More advanced, and also promising, schemes require cooperation between the networks. This scenario examines a traffic-aware interference

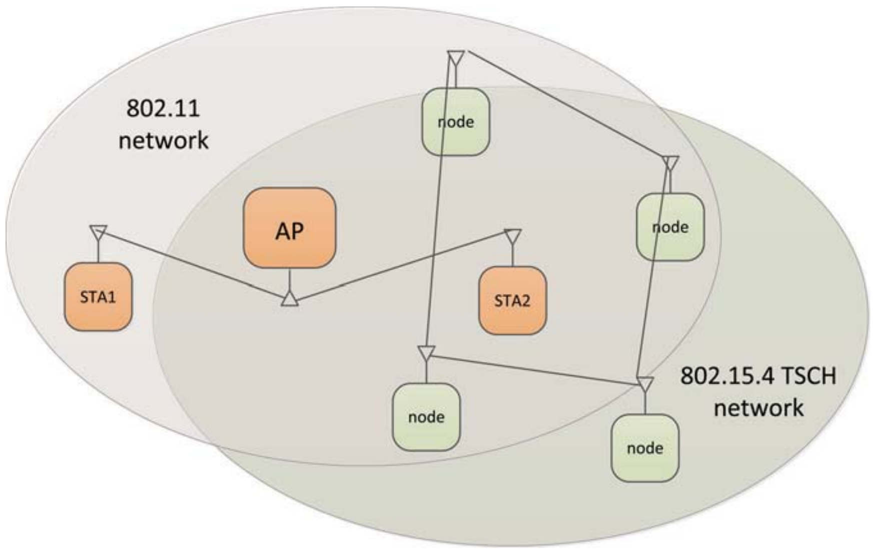


Figure 13.2 Example illustrating two co-located wireless networks of different technology.

avoidance scheme where, depending on the network load in both networks, other decisions are made. For such a scheme two possible cases, illustrated by Figure 13.3, must be considered. In the first case the sensor network is highly loaded. Here it is more efficient to perform any interference avoidance in the Wi-Fi network, thus reducing the overhead on the more loaded sensor network. To accomplish this, the sensor network would need to provide the scheduling information to allow the Wi-Fi network to delay transmissions to points in time where a collision will not occur. In the second case the network load in the Wi-Fi is high, suggesting that excluding the spectrum used by Wi-Fi from the hopping scheme of the sensor network is a more promising approach to co-existence.

More advanced approach is to use a cross-technology TDMA protocol to coordinate the transmission between both types of nodes and reduce interference to a minimum. The system runs a TDMA radio program on the Wi-Fi nodes, adapts time slots to traffic requirements, keeps free some slots that are implicitly reserved to TSCH, and uses the remainder for transmission, in order to minimize cross interferences.

To support the experimental investigation of this scenario, a great deal of functionality is required. A mechanism for the discovery of co-located wireless networks within interference range is certainly necessary to identify whether a problem exists. Furthermore, a range of mechanisms to support mutual network awareness is required, including the ability to share information regarding network load between heterogeneous networks, to expose the medium access control (MAC) schedule of the TSCH network to other coexisting technologies, as well as to notify the coexisting technologies about the wireless channel used in the IEEE 802.11 network. Moreover, mitigation

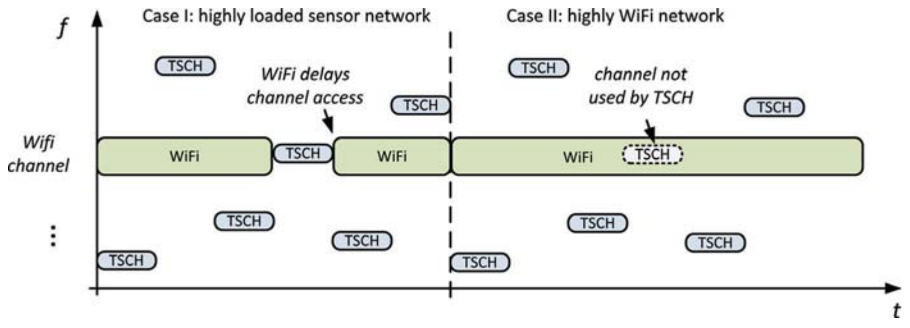

f
Case I: highly loaded sensor network
Case II: highly WiFi network
TSCH
WiFi delays channel access
channel not used by TSCH
Wifi channel
WiFi
t

Figure 13.3 The proposed co-existence scheme for avoiding interference between Wi-Fi and TSCH.

functionality must be available, potentially including the configuration of spectrum access in the Wi-Fi network, configuration of channel exclusions within the TSCH network, time synchronization between both networks, and the tuning of MAC parameters according to frames size and slot allocation.

13.3.3 Load and Interference Aware MAC Adaptation

It is well known that contention-based access protocols work better than scheduled-based protocols in case of intermittent and unpredictable traffic flows [10]. Moreover, the contention parameters can be optimized as a function of the time-varying number of nodes which have traffic to transmit. However, for most wireless technologies, the choice of contention-based or scheduled-based access protocols, as well as the configuration of the contention parameters or schedule periods can only be configured statically, and cannot be adapted to the varying network conditions.

In order to experimentally validate the possibility to perform MAC layer adaptations or to switch from one MAC protocol to another as a function of an estimate of the network topology and contention level, it is required that nodes can infer about the number of neighbors, network congestion and node visibility by monitoring elementary channel events (busy intervals, hello messages, collisions). Moreover, pre-defined MAC protocols such as CSMA and TDMA can be abstracted from the physical layer and available for different technologies by exposing the same list of configurable parameters, including the contention windows for CSMA and the frame size for TDMA protocols. Under these assumptions, different adaptation logics can be developed for maximizing the network throughput, minimizing the delay jitters or the packet losses, regardless of the specific node technology (Figure 13.4).

As an example, we initially consider only a few active wireless nodes using a CSMA base MAC.

13.3.4 In-Situ Testing

Wireless testbeds are imperative for testing innovative technologies such as protocols, hardware, and several other modules of any wireless solution. Many of these technologies will serve in dynamic wireless environments and under challenging conditions. For the sake of maintainability and experiment repeatability, however, testbed infrastructure is often fixed. Relocating nodes is difficult since their power supply and/or network connections

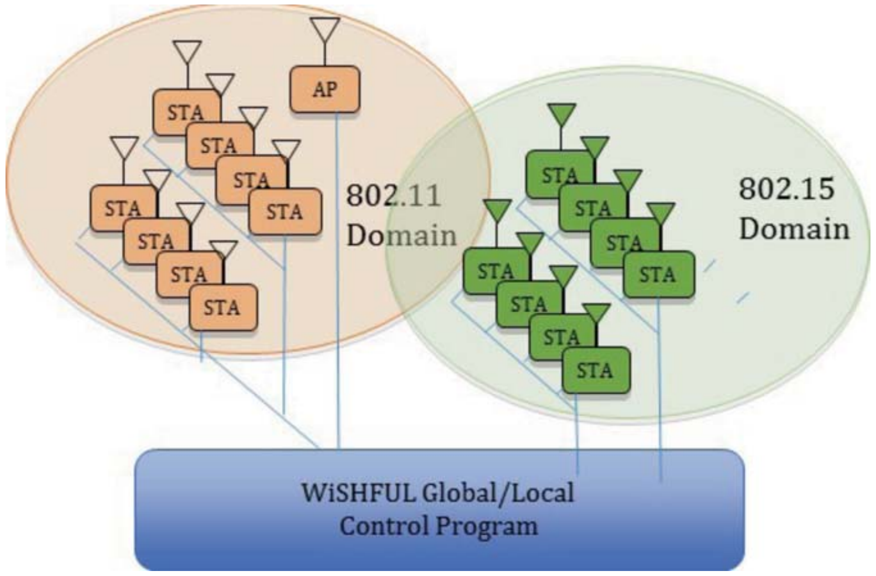


Figure 13.4 Deployment of a single local program across several platforms.

are mounted on wall sockets. The testbed environment is thus less dynamic and the conditions are more stable, thus making the evaluation of experimental wireless solutions in testbeds less realistic.

A portable testbed that can be easily deployable on remote, real-world locations is clearly necessary. Such a testbed would need to be straightforward to deploy where needed, include rugged equipment and self-contained power. Furthermore, a wireless mesh backbone to ensure connectivity between the nodes would be required to allow operation in a variety of environments. This backbone would need to employ the sort of interference management suggested by previously discussed scenarios. Finally, the portable testbed must operate in a transparent manner to allow users to examine the phenomena of interest.

Taking the successful Fed4FIRE approach [9] as a model for the use of testbed, the following steps, illustrated in Figure 13.5, would be required on the portable testbed during experiment life-cycle:

1. When the experimenter arrives at the location, the flight case is plugged into the power grid and the servers and switches boot. Optionally, the experimenter can connect the switch uplink to the Internet.
2. As the servers boot, the backbone also configures itself automatically. It creates a wireless mesh among the nodes.

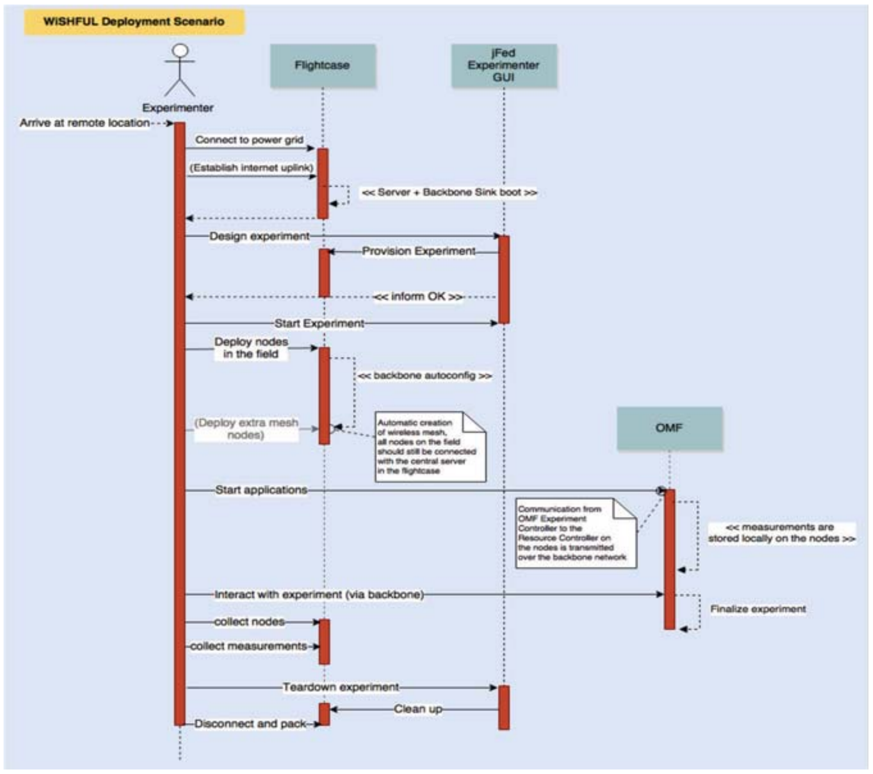

WiSHFUL Deployment Scenario
Experimenter
Flightcase
jFed Experimenter GUI
OMF
Arrive at remote location
Connect to power grid
(Establish internet uplink)
<< Server + Backbone Sink boot >>
Design experiment
Provision Experiment
<< inform OK >>
Start Experiment
Deploy nodes in the field
<< backbone autoconfig >>
(Deploy extra mesh nodes)
Automatic creation of wireless mesh, all nodes on the field should still be connected with the central server in the flightcase
Start applications
Communication from OMF Experiment Controller to the Resource Controller on the nodes is transmitted over the backbone network
<< measurements are stored locally on the nodes >>
Interact with experiment (via backbone)
Finalize experiment
collect nodes
collect measurements
Teardown experiment
Clean up
Disconnect and pack

Figure 13.5 Sequence diagram for the deployment of the portable testbed.

3. When everything is up and running, the experimenter launches the jFed tool from a laptop that is either inside the flight case or connected to the central switch. The experiment is designed or loaded from a previous run.
4. jFed will perform the needed actions via the testbed management server and the nodes will be provisioned with the desired software.
5. After this process, the user is informed and the actual experiment is started.
6. The user will deploy all nodes in the field; they remain connected and accessible via the wireless backbone.
7. If there should be a bad wireless link between one or several nodes, an extra backbone node can be added to optimize the mesh network.
8. Via Orbit management framework (OMF), the experimenter starts his experiment. OMF will make the calls to the nodes over the backbone

network. These calls can include (but are not limited to) the setup of a wireless interface, the changing of channels or the starting of an application.

9. While the experiment is running, the measurements are stored locally on the nodes.
10. As the experiment finishes, the experimenter can collect all nodes and properly dock them in the flight case, physically connecting them again with the core network.
11. The measurements are fetched from the individual nodes and the experiment can be torn down. If the throughput of the wireless mesh network is high enough, or the amount of measurement data is low, the measurement can be transported over the wireless backbone in real time to a database server in the flight case.
12. jFed will ask the central testbed server to clean the nodes up, and the flight case can be closed and plugged out.

The proposed portable testbed will allow to experiment in any given environment and to take into account the real wireless characteristics of this particular environment in the results of the experiment. Thus all the solutions targeting the aforementioned motivating scenarios can and will be tested in different environments to also test their robustness and stability in diverse wireless environments.

13.4 WiSHFUL Software Architecture

Experimentation is certainly a vital tool in the development of future wireless solutions. Furthermore, as illustrated by the above discussion of scenarios for future wireless networks, a large variety of functionality must be supported to investigate the challenges most relevant in the advancement of wireless communications. Moreover, the increasing diversity of wireless solutions and competing radio technologies, along with the ever more stringent requirements on the reliability of test results, has caused wireless test facilities to evolve to be exceedingly complicated imposing steep learning curves for new experimenters. Therefore, as the need for investigating a broad range of scenarios grows, so does the difficulty in doing so.

For these reasons, the WiSHFUL project directly targets lowering the experimentation threshold by developing flexible, scalable, open software architectures and programming interfaces to prototype novel wireless solutions. Specifically, WiSHFUL develops mechanisms for unified radio control to provide developers with deep control of physical and medium access components without requiring deep knowledge of the radio hardware platform and unified network control to allow the rapid creation, modification, and prototyping of protocols across the entire stack. These mechanisms chiefly take the form of UPIs that operate across a range of hardware platforms. In this way WiSHFUL empowers experimentation facilities with the capability to experiment with emerging wireless technologies.

13.4.1 Major Entities

The WiSHFUL architecture, illustrated in Figure 13.6, contains several entities designed to support the investigation of future networks. First and foremost within this architecture is the collection of UPIs, with each UPI providing specific functionality to experimenters. The radio interface (UPI_R) consists of a set of functions that ensure uniform control of the radio hardware and lower MAC behavior across heterogeneous devices. The functions provided herein take a generic form in order to provide experimenters with consistent operation over hardware specific implementations. The network interface (UPI_N) parallels the UPI_R with a set of functions that provides uniform control over the upper MAC and network layer protocol behavior across various devices. Again, the UPI_N consists of generic functions to provide a consistent and straightforward experimentation experience across heterogeneous platforms. The global interface (UPI_G) extends the reach of the control provided by both the UPI_R and the UPI_N across several devices in a coordinated and generic manner. The generic functions of UPI_R, UPI_N, and UPI_G are supported by monitoring and configuration engines (MCEs) that contain and manage the platform specific implementations of UPIs within WiSHFUL empowered facilities. Naturally, the UPI_R and UPI_N are supported by a local MCE, while the UPI_G employs a global MCE. Finally, the hierarchical control interface (UPI_HC) enables hierarchical communication between CPs structured in a standard manner. Note that this interface does not directly interact with hardware, but rather provides experimenters with the means to explore hierarchical control by offering a convenient method of inter-control program communication.

The separation between radio and network functionality occurs within the MAC layer of the OSI stack. In particular, WiSHFUL considers the Upper MAC and higher layers as network control functionality, relegating the Lower MAC and lower layers to radio control functionality. The Upper MAC is responsible for inter-packet states that are not time critical, including framing and management functions, where some form of negotiation between

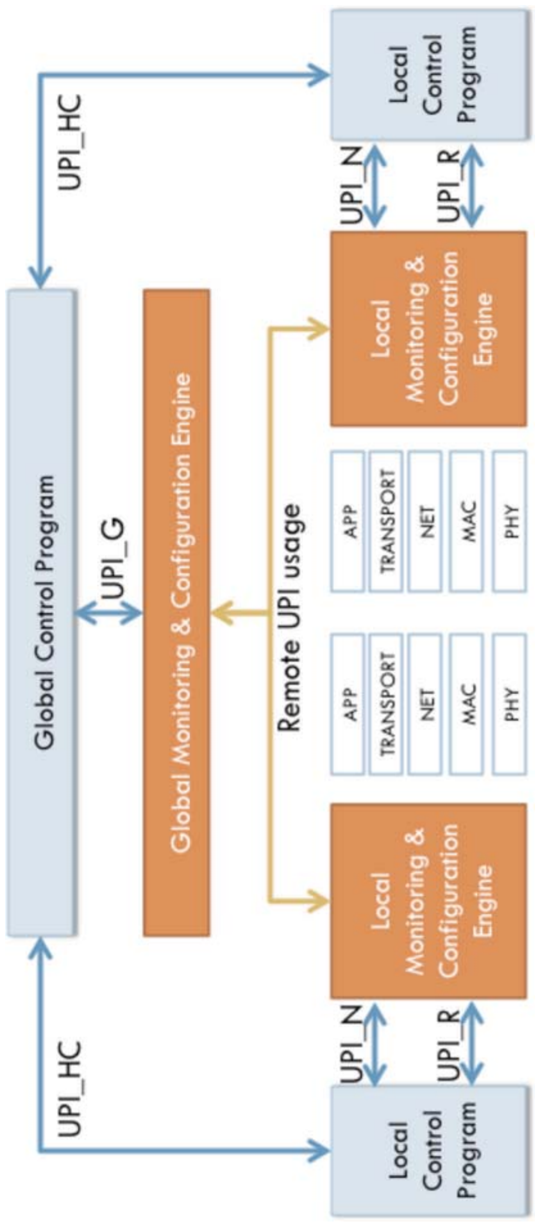


Figure 13.6 Conceptual diagram of WiSHFUL architecture.

nodes is required. The Lower MAC, on the other hand, directly interacts with the physical layer (PHY) transmission and reception operations, where minimization of processing latency is certainly critical. Typical Lower MAC functions include sending and receiving data, back-off, inter-frame spacing, and slot synchronization. As such, this distinction reflects the focus on inter-device coordination within the network control and more direct hardware operations within the radio control.

13.4.2 User Control

The interfaces of the WiSHFUL architecture are designed to support the user in controlling wireless hardware and the accompanying protocol stacks. WiSHFUL views user control as being embodied in CPs, which are either local or global in nature. In general, CPs are user defined software that implement the controlling logic for a wireless experiment and makes use of the UPI_R and/or UPI_N for hardware/protocol control. Local Control Programs (LCPs) are those that use the local information and abilities of a single device, while Global Control Programs (GCPs) interact with a group of devices.

The WiSHFUL architecture supports a two-tier control hierarchy. These two tiers work in a coordinated manner, being orchestrated at the global level. Indeed, GCPs can instantiate LCPs on wireless nodes, performing a sort of control by delegation, or can act directly on the wireless nodes in a coordinated manner. Control by delegation is needed when the reconfiguration decisions or the parameters to be monitored have strict time constraints, which cannot be guaranteed by the control network. In fact, the physical channel used for conveying control messages to/from the GCP can be unreliable and introduce some latencies. Since radio performance depends on highly variable network conditions (e.g. channel propagation, fading, interference, access timings, etc.), control by delegation is particularly important for radio control. The architecture also supports hybrid approaches, in which some control operations are managed at the global level, while some others are delegated to wireless nodes. The coordination between global and LCPs is achieved by employing the UPI_HC. Currently, the WiSHFUL framework follows a proactive approach. A CP has to trigger the execution of UPI functions on the wireless node under control. This polling-based approach might be not sufficient for every CP's implementation requirements. Therefore, it is planned to offer support also for a reactive approach in the near future. Here the user will be able to define a trigger, i.e., when a certain condition is fulfilled, a registered callback function is executed to handle the event.

13.4.3 Hardware Interfacing

Figure 13.7 illustrates how the WiSHFUL radio control works on three different platforms, namely the Iris SDR framework [6], TAISC [5] and WMP [4]. The global MCE runs remotely on a Linux machine and allows implementing node configuration that depends on network-level decisions and can be executed in a time-coordinated fashion among multiple nodes. Each of the WiSHFUL enabled nodes runs a local MCE that offers the same local services and the same UPI functions on different platforms by means of a specific connector module (CM). This unified approach unloads the experiment from the burden of dealing with a multiplicity of configuration and utility tools, such as iw, iwconfig, iptables, iwlist, iperf, b43fwdump, etc. These tools, indicated in Figure 13.7 as local control services, are heterogeneous upon platforms/operating systems and depend on the hardware and software configuration of the device under test.

The CM operates in conjunction with local MCEs to expose the uniform UPI functions on different hardware and software radio platforms.

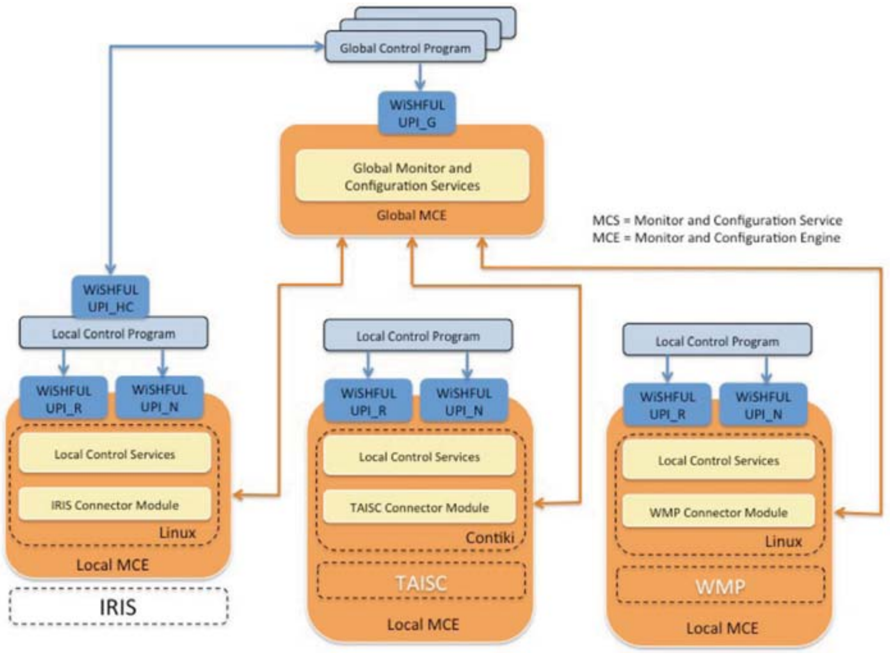


Figure 13.7 WiSHFUL architecture, UPIs, and supported platforms.

The module achieves two main goals: i) diverting platform-independent UPI calls to platform-dependent implementations and ii) providing a unified way to deal with a plethora of tools provided by heterogeneous operating systems (e.g. iw, iwconfig, iptable) or platforms (e.g. bytecode-manager for the WMP). Note that certain UPI functionality may or may not be supported by every platform, depending on the capabilities of the platform and the implementation status of the CM.

Figure 13.8 illustrates the interaction from MCE to the CM and subsequently the radio platform. The local MCE delegates each UPI call to the appropriate CM that executes the call using platform-specific sub modules. Currently, all local MCEs and CMs are implemented in Python, except from sensor nodes that, in addition to the Python implementation, also have a native implementation using GITAR [6]. The native implementation is used when the sensor nodes are decoupled. In case they have a Linux host PC (e.g. in testbeds) the Python implementation can be used. This allows to easily prototype wireless solutions for sensor networks that can also work in real deployments, when their host PCs are not available.

13.4.4 Basic Services and Capabilities

Alongside, the UPIs themselves, the WiSHFUL framework offers a number of basic services that are summarized here.

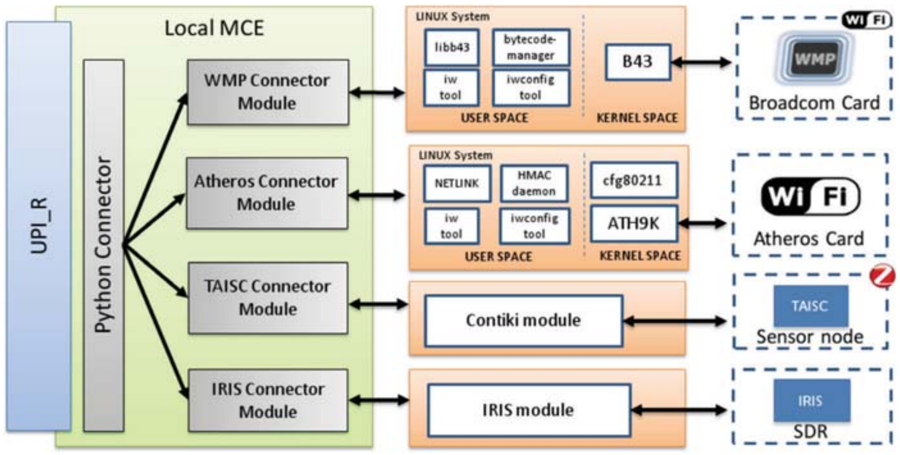


Figure 13.8 WiSHFUL adaptation modules.

13.4.4.1 Node discovery

A GCP often requires functionality for automatic node discovery. WiSHFUL provides the protocol developer an easy way to define the set of nodes he wants to control. Any wireless node belonging to the same experiment group can be controlled by a GCP using the WiSHFUL UPIs. From that set of nodes the user can either select all of them or just a sub-set.

13.4.4.2 Execution semantics

The WiSHFUL MCE (local and global) supports two execution semantics. The first is a synchronous blocking UPI call where the caller, i.e. the CP, is blocked until the callee, i.e. any UPI function, returns. The second option is an asynchronous non-blocking UPI function call. Here any UPI call returns immediately. The caller has the option to register a callback function so that he can receive the return value of the UPI call at a later point in time.

13.4.4.3 Time-scheduled execution of UPI functions

Besides the possibility of immediate execution of UPI functions either using a blocking or non-blocking scheme, the WiSHFUL MCEs also provide the possibility for time-scheduled execution of UPI functions at a particular point in time. This is important if nodes need to coordinate their actions in time, e.g. a set of nodes must perform a time-aligned switching to a new channel. The possibility for time-scheduled execution of UPI functions is especially important for GCPs if a non-real-time backbone networking system like Ethernet is used. In such networks we cannot expect that the WiSHFUL control commands are received by all nodes at the same time, e.g. due to CSMA non-deterministic behavior. Moreover, network congestion and delay are also reasons for providing hierarchical control over UPI_HC between local and GCPs.

13.4.4.4 Remote execution of UPI functions

WiSHFUL provides full location transparency. Any UPI function can be executed either locally by a LCP or remotely by a GCP. In the latter case, the WiSHFUL global MCE transparently serializes all input and output arguments. The calling semantic for both the local and remote calls is call-by-value. This has to be considered when extending the UPIs with additional functionality. Finally, as with the local execution also the execution of remote functions can be time-scheduled. This is especially important if a given UPI function needs to be executed at the same time on a set of wireless nodes.

13.4.4.5 Time synchronization

A wide range of WiSHFUL applications, like the centralized control of channel access, requires a tight time synchronization among wireless nodes. The way the wireless nodes are time synchronized is platform and architecture-dependent. Basically, we distinguish between systems based on whether a backbone network exists. Here in order not to harm the performance of the wireless network the nodes are time synchronized using the backbone (e.g. Ethernet) and some time-protocol like Precision Time Protocol (PTP). Wireless nodes without a backbone have to rely on other techniques for time synchronization (e.g. through global positioning system, GPS).

13.4.4.6 Packet forgery, sniffing and injection

WiSHFUL provides a wide range of functionality for packet forgery, sniffing and injection. A CP can use this to create and inject network packets into any layer of the network stack of a node or to receive copies of packets.

13.4.4.7 Deployment of new UPI functions

WiSHFUL provides an open and extensible architecture, which can be easily extended by new UPI functions. Any new introduced UPI function can be implemented in a different way for different platform and software architecture. Therefore, in WiSHFUL for each platform there is separate CM, as discussed above.

13.4.4.8 Global control

To enable remote usage of UPI functions using the UPI_G interface, a system supporting remote procedure calls is required. For this purpose the arguments of UPI functions have to be serialized and sent to proper node. The proposed framework provides a user-friendly interface that hides all complexity of serialization and transferring data between GCP and nodes.

13.4.4.9 Remote injection and execution of user code

To enable support for global management and control of the deployment of a WiSHFUL controlled experiment, the proposed framework supports “on-the-fly” injection of user defined functions (constituted of UPI) to be executed locally on a node directly from a GCP.

13.5 Implementation of Motivating Scenarios and Results

In order to clarify the potentialities of the WiSHFUL architecture and unified programming, in this section we present some examples of control logic and protocol adaptations developed for the motivating scenarios presented in Section 13.3. The goal is not designing a novel optimization logic for each scenario, but rather demonstrating the flexibility of the proposed approach by separating the logic for controlling the experimentation platforms from the specific transmission mechanisms running on the platform.

13.5.1 Interference Management Among Overlapping Cells

We decompose this scenario into two tasks: 1) hidden node detection and 2) hybrid TDMA-MAC management. For investigation of both tasks, we implemented a WiSHFUL enabled Wi-Fi network.

13.5.1.1 Hidden node detection

The first task to be solved for the efficient airtime management showcase is the detection of wireless links, which are suffering from performance degradation due to hidden terminals (Figure 13.9). Specifically, only flows using links, which are suffering from the hidden node problem, should be assigned to exclusive time slots. Hence, WiSHFUL provides functionality, which detects links which are suffering from the hidden node problem.

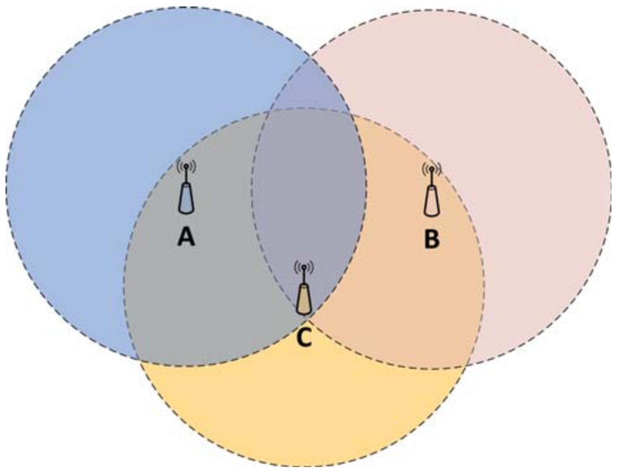


Figure 13.9 Example illustrating a hidden node scenario. As nodes A and B are outside their carrier sensing range the packet transmissions from A and B would collide at node C.

13.5.1.1.1 *Application of WiSHFUL framework*

For hidden node detection WiSHFUL provides the following UPI network functions which are used by GCPs.

Given a set of nodes using the specific wireless interface,radio channel (e.g. 6) and detection threshold (e.g. 0.9) this function returns a boolean matrix indicating which nodes are inside each node's carrier sensing range and which are outside:

```
def getNodesInCarrierSensingRange(self, nodes, wifi_intf, rfCh, detection_th)
```

Furthermore by using this function that returns a boolean matrix indicating which nodes are inside each node's communication (reception) range and which are outside it is possible to reach to a conclusion if there is a hidden node for any pair of nodes forming a link in the network:

```
def getNodesInCommunicationRange(self, nodes, Wi-Fi_intf, rfCh, detection_th)
```

These two functions are used to detect links hidden by some node. As an illustrative example, consider the case where nodes A and B are outside of carrier sensing range and C is inside the reception range of both A and B. In this case packet transmission from A to C and B to C must use exclusive time slots in order to prevent performance degradation due to packet collisions. The technical details of this functionality is further discussed in deliverable D4.2 [12].

13.5.1.1.2 *Results*

The used algorithm performs two steps. First, we use the UPI functionality to estimate which nodes are in carrier sensing range and which are outside. The algorithm uses the following approach:

- It first compares the measured isolated broadcast transmit rate of each node with the one achieved by transmitting concurrently with some other node in the network. If the latter is smaller we know that the two nodes are in carrier sensing range.
- Second, we use the UPI functionality to estimate which nodes are in communication range. The corresponding UPI function sets each wireless node in sniffing mode. Then, in each round a single transmitter is transmitting raw 802.11 broadcast frames while the other nodes are capturing the received frames.

With the information which nodes are in carrier and reception range we are able to estimate which links are suffering from hidden nodes and hence must be protected.

13.5.1.2 Hybrid TDMA MAC

Enterprise IEEE 802.11 networks need to provide high network performance to support a large number of diverse clients like laptops, smartphones and tablets as well as capacity hungry and delay sensitive novel applications like mobile HD video and cloud storage. Moreover, such devices and applications require much better mobility support and higher QoS and quality of experience (QoE).

IEEE 802.11 uses a random access scheme called distributed coordination function (DCF) to access and share the wireless medium. The advantage of DCF is its distributed and asynchronous nature making it suitable for unplanned ad-hoc networks which have no infrastructure. The main disadvantage is its inefficiency in congested networks. Moreover, it suffers from performance issues due to hidden and exposed node problem which is a severe problem in high density enterprise networks.

In contrast to DCF, in TDMA the channel access is scheduled in a synchronized and centralized manner, and hence is able to provide the required high QoS/QoE requirements of enterprise environments. WiSHFUL allows to build TDMA on top of today's off-the-shelf Wi-Fi hardware by providing a flexible and extensible software solution. Currently, the focus is being set on the downlink whereas in the future also the uplink will be considered for support from the TDMA scheme.

Following the software-defined networking (SDN) paradigm we separate the control plane from the data plane and provide an application programming interface (API) to allow local or global CPs to configure the channel access function. In particular it allows to configure the TDMA downlink channel access by defining the number and size of time slots in the TDMA super-frame. Moreover, for each time slot a medium access policy can be assigned which allows to restrict the medium access for particular stations (identified by their MAC address) and traffic identification (e.g. VoIP or video). The latter can be used to program flow-level medium access. Finally, for each time slot we can configure whether carrier-sensing is activated or not. The latter would results in the classical TDMA MAC while keeping carrier-sensing within each slot allows for transparent coexistence with legacy networks that are not aware of the TDMA scheme being used within the WiSHFUL enabled network. The data plane itself resides in each AP and is controlled by the WiSHFUL runtime system.

The control plane in our design is managed by either a global or local WiSHFUL CP which takes as input the channel access scheme specified by the applications. Any application is responsible to decide on how to map the per-flow QoS requirements on the channel access. An example would be to measure which wireless links are suffering from hidden node problem and to assign exclusive time slots for flows requiring high QoS. However, the provided centralized coordination for channel access requires a tight time synchronization among APs. In WiSHFUL time synchronization is performed using the wired backhaul network and hence is not harming the performance of the wireless network. The utilized Precise Time Protocol (PTP) gives an accuracy in microsecond level. The WiSHFUL MCE running on each AP locally is responsible for coordination of channel access as configured by the local or global CP.

13.5.1.2.1 *Application of WiSHFUL presentation of UPIs used*

The UPIs provided by WiSHFUL to set-up and control a hybrid TDMA MAC are as follows:

```
def installMacProcessor(self, node, interface, mac_profile)
def updateMacProcessor(self, node, interface, mac_profile)
def uninstallMacProcessor(self, node, interface, mac_profile)
```

The UPI functions allow the installation, reconfiguration at runtime and uninstallation of a hybrid TDMA MAC. The mac_profile is an object-oriented representation of the hybrid MAC configuration (Figure 13.10).

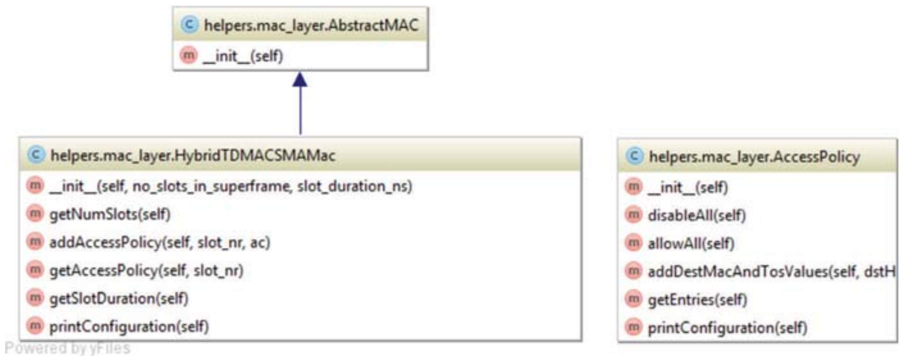

helpers.mac_layer.AbstractMAC
__init__(self)
helpers.mac_layer.HybridTDMACSMAMac
__init__(self, no_slots_in_superframe, slot_duration_ns)
getNumSlots(self)
addAccessPolicy(self, slot_nr, ac)
getAccessPolicy(self, slot_nr)
getSlotDuration(self)
printConfiguration(self)
helpers.mac_layer.AccessPolicy
__init__(self)
disableAll(self)
allowAll(self)
addDestMacAndTosValues(self, dstH
getEntries(self)
printConfiguration(self)
Powered by yFiles

Figure 13.10 UML class diagram showing the hybrid MAC relevant configuration.

The following example shows how to set-up a new hybrid MAC instance:*# create new MAC for each node* HybridTDMACSMA-Mac(no_slots_in_superframe=7,slot_duration_ns=20e3)

```
# assign access policy to slot 0
acBE = AccessPolicy()
# MAC address of the link destination
dstHWAddr = '12:12:12:12:12:12'
# best effort
tosVal = 0
acBE.addDestMacAndTosValues(dstHWAddr, tosVal)
slot_nr = 0
mac.addAccessPolicy(slot_nr, acBE)
# assign time guard slot 1
acGuard = AccessPolicy()
acGuard.disableAll() # guard slot
slot_nr = 1
mac.addAccessPolicy(slot_nr, acGuard)
# UPI call
mac = radioHelper.installMacProcessor(node, iface, mac)
```

Finally, Figure 13.11 illustrates the hybrid MAC being configured to assign exclusive time slots to two wireless links which are hidden to each other. In order to account to time synchronization inaccuracy guard slots are added.

13.5.1.2.2 *Results*

Figure 13.12 depicts how the UPI functionality is implemented on a Linux system using an Atheros Wi-Fi chip and the Ath9k wireless driver. When the locally running WiSHFUL agent receives a command for the setup of a hybrid MAC TDMA from the GCP (*installMacProcessor()* command), it starts

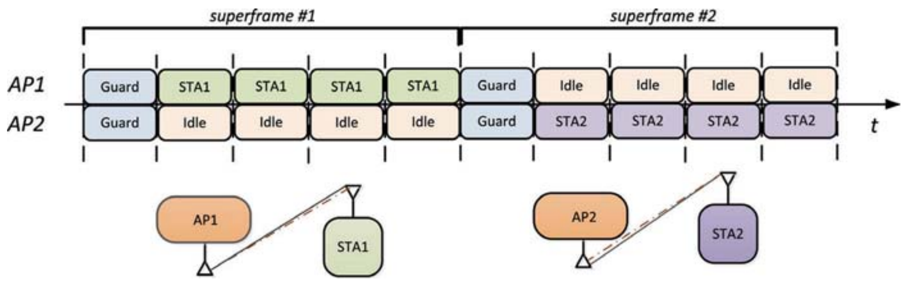


Figure 13.11 Illustration of exclusive slots allocation in TDMA.

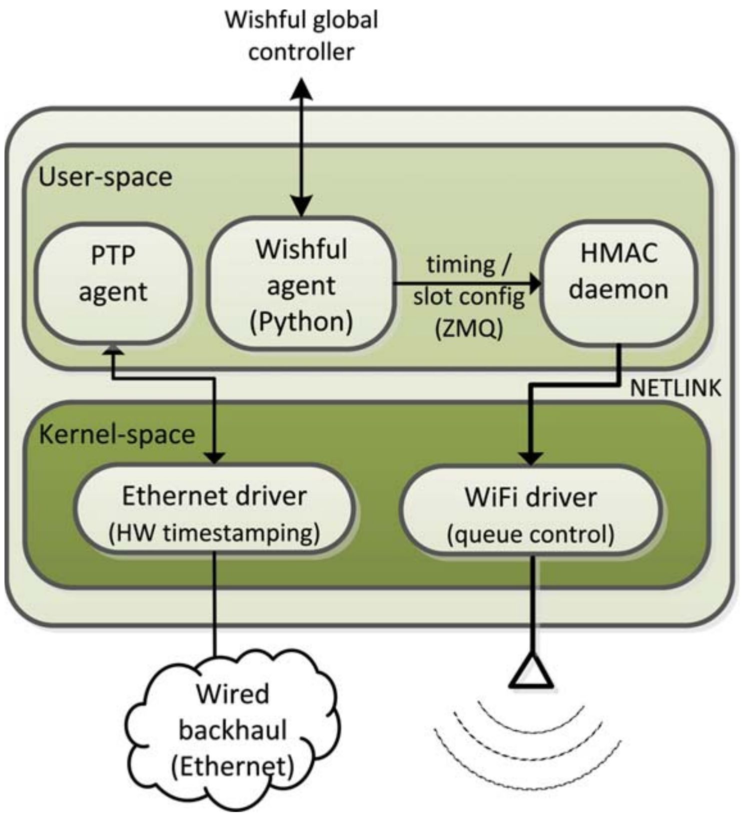

Wishful global controller
User-space
PTP agent
Wishful agent (Python)
timing / slot config (ZMQ)
HMAC daemon
NETLINK
Kernel-space
Ethernet driver (HW timestamping)
WiFi driver (queue control)
Wired backhaul (Ethernet)

Figure 13.12 Overview of the components on the wireless node in the Linux-Wi-Fi prototype.

the HMAC daemon. The agent controls the (re)configuration of the HMAC daemon using a message passing system (ZMQ). The task of the daemon is to pass slots configuration information to the wireless network driver using the NETLINK protocol. Moreover, it is responsible to inform the wireless driver about the beginning of each time slot. The patched wireless driver uses the slot configuration information to control which network queues are active and which are frozen. Only packets from active queues are allowed to be sent while the others are buffered.

In order to evaluate the proposed efficient airtime management scheme, experiments were conducted in the TWIST 802.11 testbed. Ubuntu 14.04, Intel i5s with a wired Ethernet NIC from Intel supporting HW timestamping and an Atheros 802.11n wireless chip were used in order to setup the experimental network deployment.

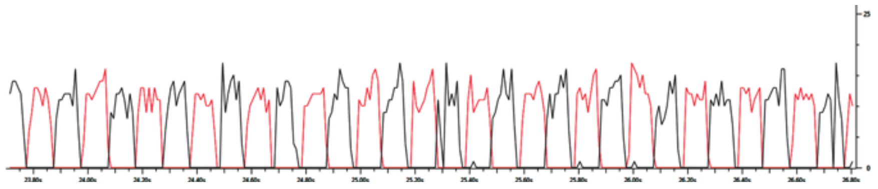


Figure 13.13 IO graph illustrating the number of packets sent over time. The color indicates a particular flow.

At the beginning of the experiment the global WiSHFUL CP used the network function UPIs in order to detect the wireless links which are suffering from the hidden node problem. Afterwards the GCP directed the hybrid MAC on these nodes in such a way that exclusive time slots were assigned.

In the following graphs results are presented for two selected wireless links which are suffering from the hidden node problem. These two links were automatically discovered by our protocol and the proper hybrid MAC was set-up. Figure 13.13 shows the IO graph where the color indicates the two different links (flows). We can clearly see that the provided hybrid TDMA scheme is able to isolate the two flows as desired.

The performance improvement compared to standard 802.11 DCF is show in Figure 13.14. On this particular link the throughput could be increased by a factor of 5.2 and 2.8 respectively which is an impressive increase in network capacity.

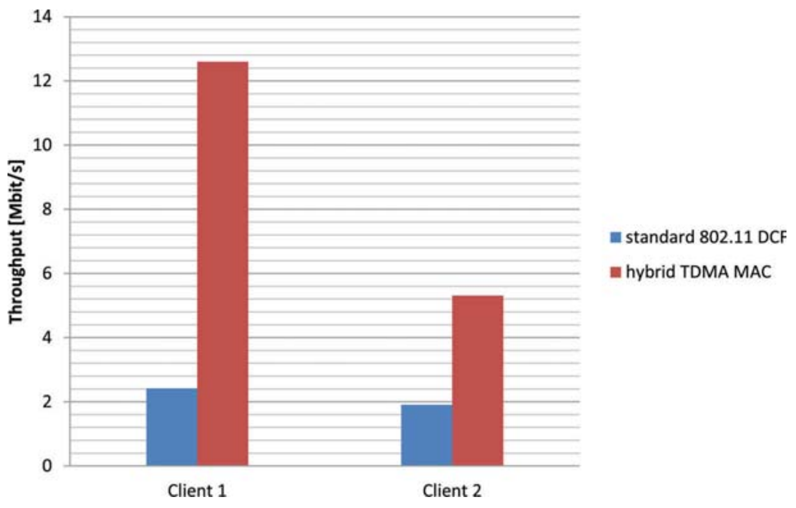


Figure 13.14 TCP/IP performance.

The described efficient airtime management scheme was fully implemented and the source code is available in the WiSHFUL project's Github public repository [13].

13.5.2 Co-existence of Heterogeneous Technologies

Up until this time the coexistence of different wireless technologies in the same domain has been inadequately supported and mostly is based on simply selecting different wireless channels to divide into the frequency plane the heterogeneous technologies that use the same spectral band in general like the ISM band at 2.4 GHz.

The WiSHFUL control framework aims to provide solutions also in the time plane based to inter-technology communication and synchronization. Table 13.1 lists the communication technologies that are currently supported and summarizes, for each technology, the available operating systems, hardware platforms and drivers.

The demonstration set-up presented in this scenario is deployed in the iMinds w.iLab.t testbed and comprises of 32 Contiki sensor nodes with an IEEE-802.15.4 radio and 14 Linux nodes with two IEEE-802.11 radios. Both showcases are executed simultaneously and can be demonstrated remotely. During execution, live measurements are taken and can be presented in two formats: 1) live graphs displaying performance statistics and b) real-time spectrum scanning plots using a universal software radio peripheral (USRP) device. The following configuration options for both showcases are possible:

13.5.2.1 Configuration options for the basic showcase

The experimenters can configure the Wi-Fi channel (2.4 GHz ISM band) and select the bandwidth (20/40 MHz) used for sending Wi-Fi frames. To mitigate interference, experimenters can choose the TSCH channels

Table 13.1 Supported platforms, OSs and drivers

	Supported Platforms, Operating Systems and Drivers		
Technology	Operating System	Platform	Driver
IEEE-802.11	Linux, Windows	Atheros, Broadcom	Ath9k, NDIS driver, WMP
IEEE-802.15.4	Contiki, TinyOS	MSP430, CC2x20, CC283x	Contiki/TinyOS drivers, TAISC
SDR	Linux, Windows	USRP, Xylink ZebBoard	Iris, LabView, GNU radio

that must be blacklisted. It is also possible to add an extra external Wi-Fi interference stream on a different channel and investigate the impact of uncontrolled cross technology interference.

13.5.2.2 Configuration options for the advanced showcase

The experimenters can also dynamically change the cross-technology TDMA schedule, e.g. allocation of slots between Wi-Fi and TSCH networks. Moreover, they can also specify a different synchronization pattern on-the-fly and add multiple concurrent streams in the TSCH network.

13.5.2.3 Results

An example of the live performance statistics monitored during execution of the first, basic showcase is given in Figure 13.15. The graph shows the overall average network throughput measured over time. From the results, it can be clearly seen that there is a substantial loss of throughput when there is Wi-Fi interference. After blacklisting the affected TSCH channels, the throughput rises up again close to its previous value. By changing the configuration parameters described in Section V. A, an experimenter can witness an immediate impact on the performance. Note that other statistics such as packet loss, jitter, TX throughput can be measured as well.

While executing the more advanced showcase it is also possible to monitor performance statistics in combination with real-time spectrum scanning using

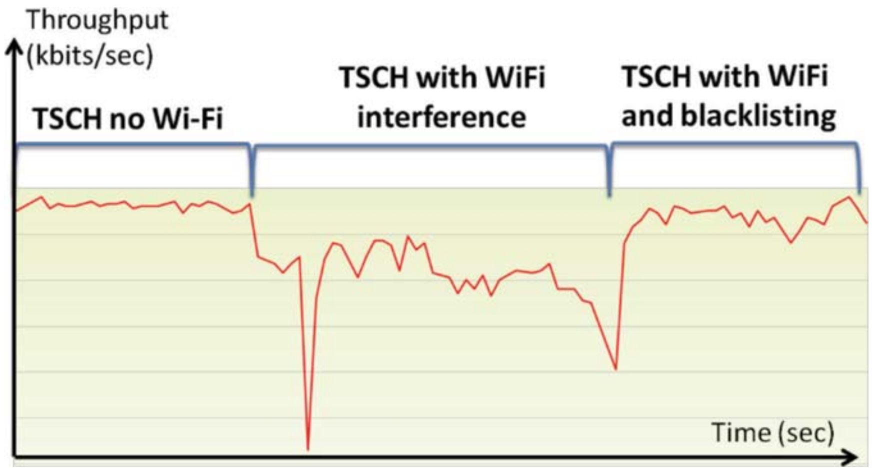


Figure 13.15 Live performance statistics showing the average network throughput (kbits/sec) over time.

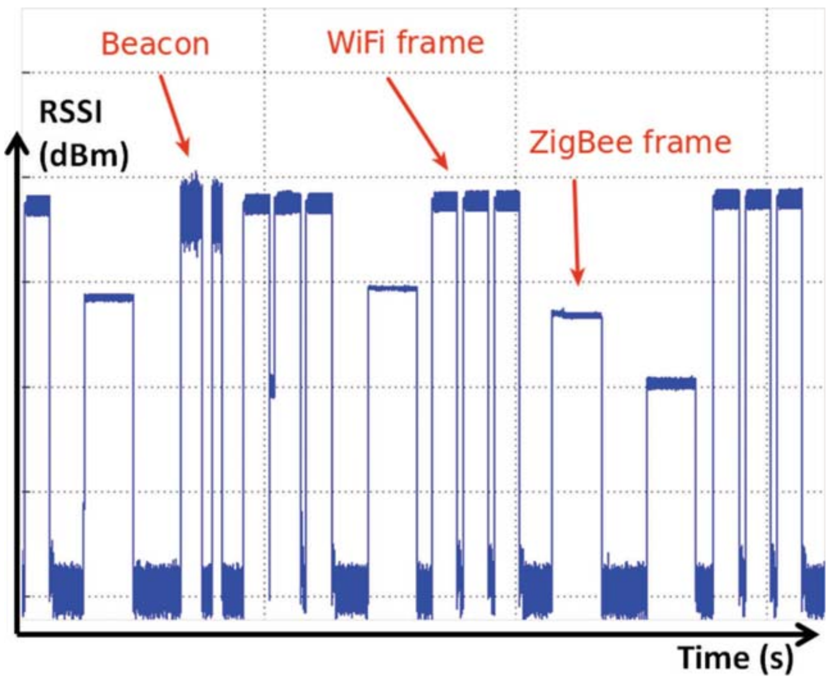


Figure 13.16 Live capture of RSSI (dBm) measured by the USRP over time.

USRP devices. Figure 13.16 illustrates the cross-technology synchronization beacon and TDMA schedule in real-time using an energy detection plot (the y-axes is RSSI in dBm). When configuring this showcase, experimenters will have an immediate feedback on the USRP plot.

The results from both showcases demonstrate the effectiveness of cross-technology interference mitigation and the ability to quickly set-up, investigate and fine-tune an interference scenario using the WiSHFUL control framework.

13.5.3 Load and Interference Aware MAC Adaptation

Here, the application of WiSHFUL in order to enable technology-independent MAC adaptation logic is presented. By employing the WiSHFUL framework it is possible to: i) dynamically tune the parameters of contention-based protocols based on load and interference conditions, and ii) switch between protocols. The logic can work on Wi-Fi or IEEE 802.15.4 nodes, regardless of the PHY layer capabilities and even on cognitive radio platforms, by exploiting the following main functionalities supported by the WiSHFUL UPI: sensing capabilities of wireless nodes, local tuning of CSMA contention windows, and global coordination of MAC protocol switching.

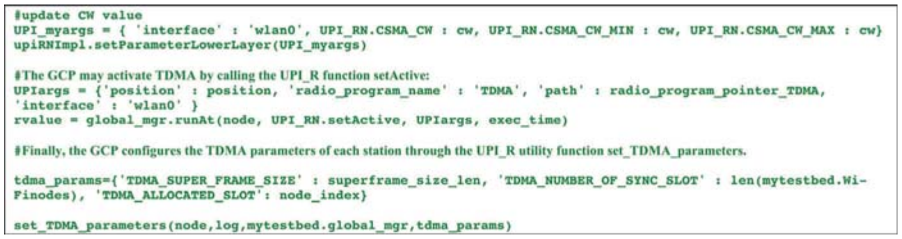
A wireless network with a time-varying number of active nodes under the same contention domain (where all the nodes are in radio visibility) is taken into account and a wired ethernet network is available as a control network between the GCP, the wireless stations and the access point. Each node runs a local optimization function that is loaded by the GCP for tuning the contention window of a CSMA protocol as a function of the network load.

In particular, a tuning function called Moderated EDCA backoff (MEDCA) is used, whose goal is the minimization of the delay jitters on the channel access times. Since these jitters depend on the exponential backoff mechanism, which introduces short-term throughput unfairness among the stations, the tuning function automatically finds a fixed contention window equal to the average contention window value experienced under exponential backoff.

When the number of stations crosses a given threshold, the GCP disables the LCP and coordinates the on-the-fly protocol switch from MEDCA to TDMA in all the nodes. As part of this, the GCP sets the TDMA parameters, such as the number of slots and the slot allocation to each station, based on the number of active flows.

13.5.3.1 Application of the WiSHFUL framework

Once calculated according to the MEDCA scheme, new contention window values are set through the UPI_R function responsible of configuring lower layer parameters as follows:



```
#update CW value
UPI_myargs = { 'interface' : 'wlan0', UPI_RN.CSMA_CW : cw, UPI_RN.CSMA_CW_MIN : cw, UPI_RN.CSMA_CW_MAX : cw}
upiRNImpl.setParameterLowerLayer(UPI_myargs)

#The GCP may activate TDMA by calling the UPI_R function setActive:
UPIargs = {'position' : position, 'radio_program_name' : 'TDMA', 'path' : radio_program_pointer_TDMA,
'interface' : 'wlan0' }
rvalue = global_mgr.runAt(node, UPI_RN.setActive, UPIargs, exec_time)

#Finally, the GCP configures the TDMA parameters of each station through the UPI_R utility function set_TDMA_parameters.

tdma_params={'TDMA_SUPER_FRAME_SIZE' : superframe_size_len, 'TDMA_NUMBER_OF_SYNC_SLOT' : len(mytestbed.Wi-
Finodes), 'TDMA_ALLOCATED_SLOT': node_index}

set_TDMA_parameters(node,log,mytestbed.global_mgr,tdma_params)
```

13.5.3.2 Results

First focusing on contention window tuning, we activated six wireless nodes running CSMA with exponential backoff contending under greedy traffic sources towards a common access point. Figure 13.17 shows the throughput performance achieved by each station. The short-term and long-term throughput variability exhibit here results from the exponential backoff mechanism (short-term) and the location-dependent interference conditions suffered by each station (long-term).

For three of the above nodes, we activated the MEDCA backoff scheme. Figure 13.18 shows that these stations achieve an average throughput comparable to that experienced with exponential backoff, but with smaller fluctuations.

Turning to MAC protocol switching, Figure 13.19 shows the measured packet loss and throughput before, during, and after the switch occurs. Radios operated in CSMA for 90 seconds then switched to TDMA. Here we examined 32 sensor nodes, which sent iPerf traffic to a single sensor node acting as a sink and used the output of the iPerf server to generate the graphs. All nodes were in the same collision domain.

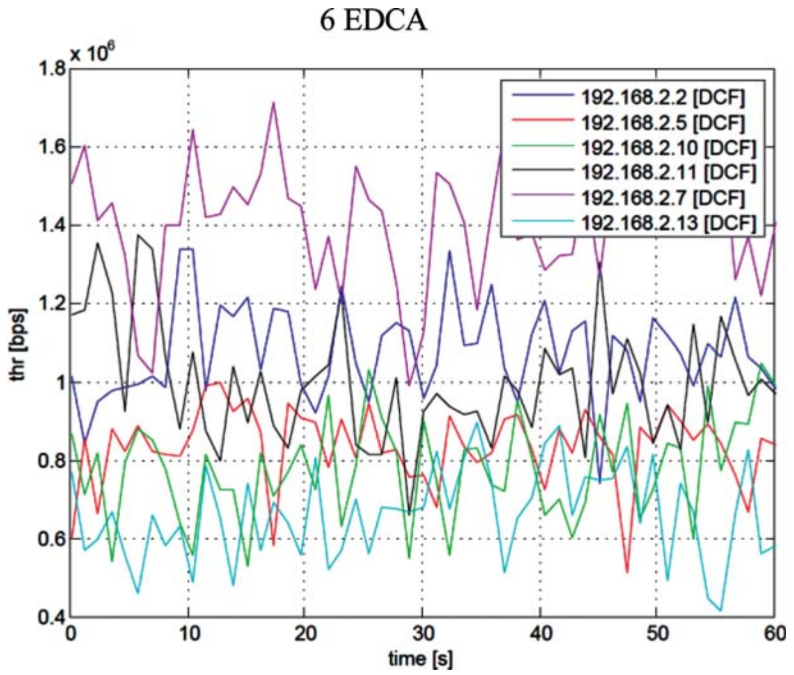


Figure 13.17 Throughput performance of 6 wireless nodes executing CSMA with exponential backoff.

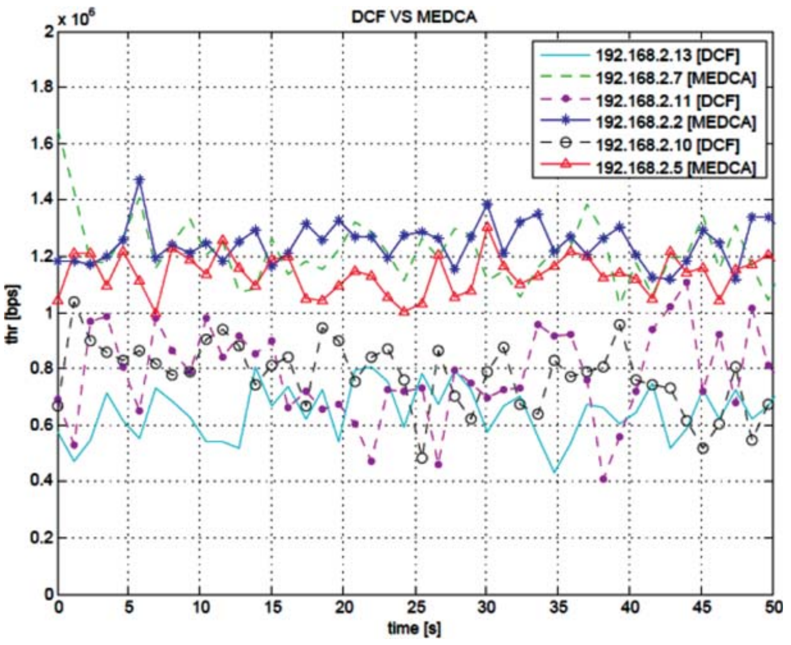


Figure 13.18 Throughput performance with 3 stations employing MEDCA backoff.

13.5.4 Wireless Portable Testbed

The WiSHFUL project offers access to several wireless testbeds, such as TWIST (TUB), w-iLab.t (IMINDS), IRIS (TCD), Orbit (Rutgers University) and a FIBRE Island at UFRJ. All of these testbeds are installed in either office environments or other dedicated testbed environments. Because some research requires doing measurement campaigns or actual testing in heterogeneous environments, the WiSHFUL project also offers a portable testbed to the community.

13.5.4.1 Portable testbed setup

The architecture of the portable testbed is presented in Figure 13.20. As can be seen there are two distinct wireless networks (blue and yellow) present in the testbed, namely BN (Backbone) network and DUT (Device Under Test, or Experiment Node) network. These two networks will be configured and controlled by the Experiment Management Servers. The blue arrows represent a highly reliable wireless backbone that allows the user to place the nodes anywhere in the field without having the practical disadvantages of using cables.

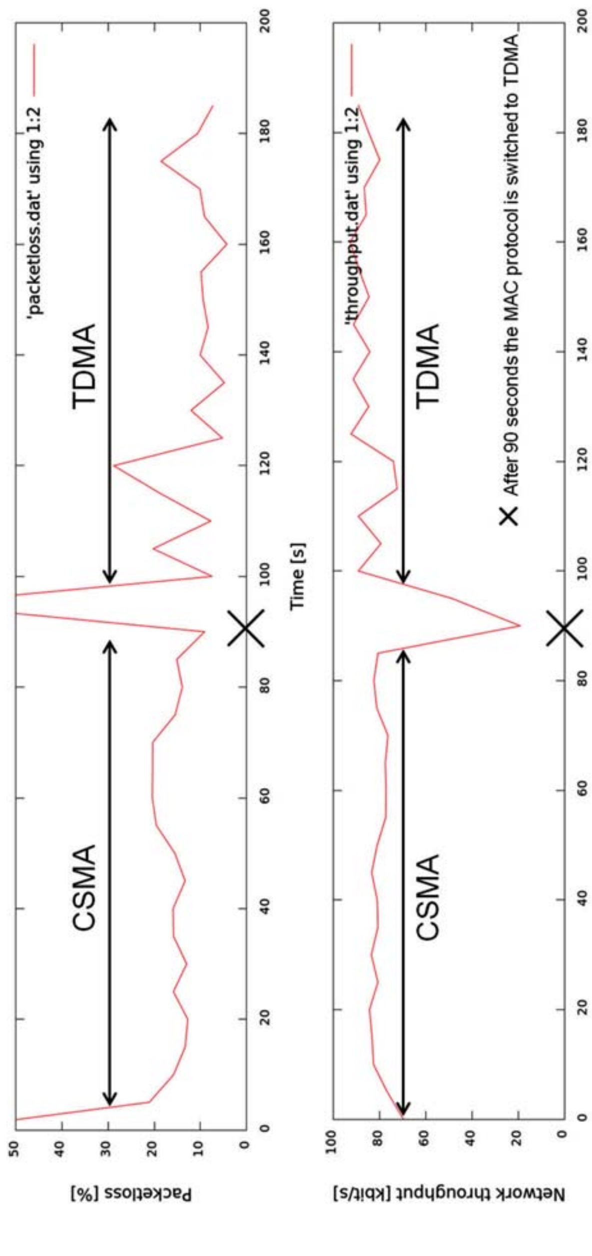


Figure 13.19 Switching from CSMA to TDMA. The upper graph shows the overall percentage of packet loss, the lower graph illustrates the overall throughput. The X marks the switch.

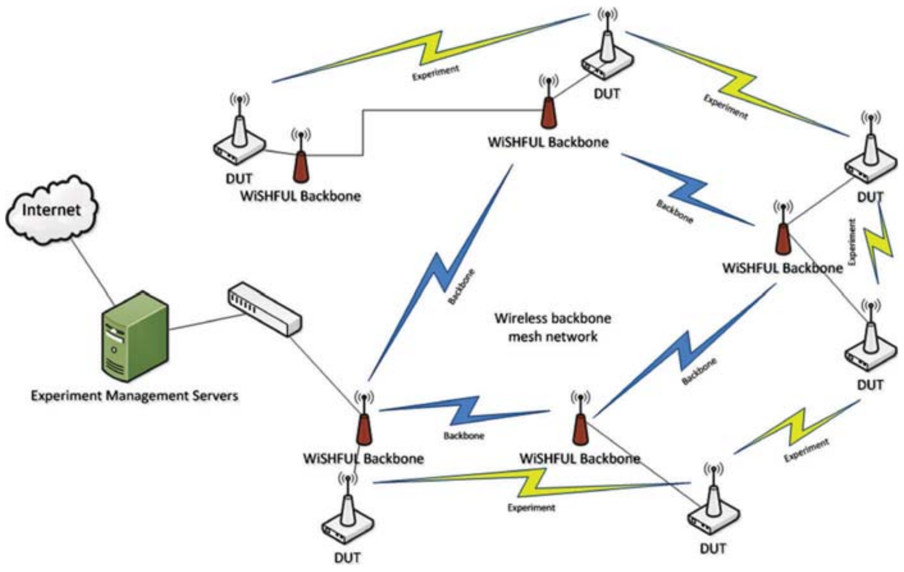


Figure 13.20 Portable testbed overview.

It also allows interaction with the nodes during the experiment. As the Portable Testbed introduces an additional network to an experiment, it is implemented in such a way that an experimenter is not overwhelmed with additional and complicated configuration procedures. In D6.1 [14], it is shown that Portable Testbed follows the "Plug and Play" approach and an experimenter should be able to use the same Testbed and Experiment Management tools as on the fixed testbeds. It has to be noted that an experimenter does not have the possibility to directly control the behavior of the Backbone network, but he is able change the channel that the Portable Testbed uses. Moreover, logical L2 networks are provided to interconnect DUT nodes in order to make them unaware whether they are connected to Portable Testbed or to a regular wired Ethernet network. This approach also reduces the required configuration because an experimenter does not have to configure any routing on his DUT nodes.

A more detailed description of the testbed setup can be found in D6.1 and D6.2 [15].

13.5.4.2 Hardware & packaging

In order to provide flexible means of transport for the portable testbed, an easy to carry, robust and spacious case is desired. It also needs protective material on the inside so the delicate electronics are not damaged during transport. Plywood flight cases are used to secure the hardware in transport.

A primary flight case hosts the central switch and experiment management servers. The EMS is a single, powerful embedded PC that hosts several VM's for each of the testbed core services. The DUT nodes are stacked in several secondary flight cases. These are made from aluminum and robust plastic and are slightly lighter than the primary case. To fix the nodes inside the case, foam is used: a base of hard foam is glued to the bottom of the case and is cut specifically to fit the DUTs. In the top of the briefcase, softer, more flexible polyurethane foam is used as its only function is to push down softly on the nodes so they stay in place while transporting the cases.

DUT devices are COTS Intel NUC (Next Unit of Computing) devices of model D54250WYKH. These are basically headless barebone PC's. They consist of an Intel Core i5 4250U processor, 4 GB of RAM, a gigabit Ethernet port, several USB ports, a 320 GB hard disk and two Wi-Fi cards: one 802.11n (WPEA-121N/W) and one 802.11ac card (WLE900VX 7AA). The nodes are by default equipped with an 802.15.4 sensor node and a Bluetooth USB dongle. The USB connections of the node can be used to attach extra hardware (e.g. LTE dongles or other USB compatible hardware). The DUT features a default embedded Linux operating system to which the experimenter can gain full (root) access. The experimenter has full control over the operating system and the software packages that are installed on the DUT. The DUT can also be used as a proxy to access all USB peripherals of the node, like sensor nodes. If the embedded PC provided by WiSHFUL does not satisfy the experimenter's needs, other hardware can be used as long as it can interface over Ethernet with the backbone nodes. A more detailed description of testbed hardware can be found in D6.2.

13.6 Conclusion

Advancing wireless communications requires overcoming several challenges. Herein, several such challenges have been examined in the form of motivating scenarios. These scenarios outline a number of requirements on experimentation platforms for investigating the future of wireless communications. The WiSHFUL project directly addresses these challenges and requirements by defining a software framework to support unified experimentation across several platforms beyond the today's standards. Examples have been provided through several case studies that apply the WiSHFUL framework to the motivating scenarios and results obtained are presented. It is evident that the use of the WiSHFUL framework provides the necessary functionality to enable advanced wireless experimentation while in parallel it lowers the learning curve for any experimenter across multiple heterogeneous wireless communication technologies.

Acknowledgment

The research leading to these results has received funding from the European Horizon 2020 Program under grant agreement n645274 (WiSHFUL project).

References

[1] Fortuna, C, et al. (2015). Wireless Software and Hardware platforms for Flexible and Unified radio and network controL. European Conference on Networks and Communications (EuCNC 2015), Workshop on 5G Testbeds and Hands-on Experimental Research, Paris, France, June 29, 2015.
[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. 2008. OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun.
[3] IETF Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification, https://tools.ietf.org/html/rfc5415
[4] ETSI TR 102 682, Reconfigurable Radio Systems (RRS); Functional Architecture for Management and Control of Reconfigurable Radio Systems, 2009.
[5] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, "Wireless MAC Processors: Programming MAC Protocols on Commo dity Hardware" IEEE INFOCOM, March 2012.
[6] Bart Jooris; Eli De Poorter; Peter Ruckebusch; Peter De Valck; Christophe Van Praet; Ingrid Moerman; TAISC: a cross-platform MAC protocol compiler and execution engine; Under submission for Computer Networks.
[7] Sutton, Paul, et al. "Iris: an architecture for cognitive radio networking testbeds." Communications Magazine, IEEE 48.9 (2010): 114–122.
[8] A. Zubow and R. Sombrutzki, "A low-cost MIMO mesh testbed based on 802.11n," 2012 IEEE Wireless Communications and Networking Conference (WCNC), Shanghai, 2012, pp. 3171–3176.
[9] P. Ruckebusch, E. De Poorter, C. Fortuna, and I. Moerman, (2015). GITAR: Generic extension for Internet-of-Things Architectures enabling dynamic updates of network and application modules. Ad Hoc Networks, January 2016, Vol. 36, Part 1, Pages 127–151.

[10] Wauters, Tim, et al. “Federation of Internet experimentation facilities: architecture and implementation.” European Conference on Networks and Communications (EuCNC 2014). 2014.

[11] A. C. V. Gummalla and J. O. Limb, “Wireless medium access control protocols,” in IEEE Communications Surveys & Tutorials, Vol. 3, No. 2, pp. 2–15, Second Quarter 2000.

[12] Deliverable D.4.2, WiSHFUL project, H2020, 2016, http://www.WiSHFUL-project.eu/deliverables

[13] WiSHFUL project Github repository, https://github.com/WirelessTestbedsAcademy

[14] Deliverable D.6.1, WiSHFUL project, H2020, 2016, http://www.WiSHFUL-project.eu/deliverables

[15] Deliverable D.6.2, WiSHFUL project, H2020, 2016, http://www.WiSHFUL-project.eu/deliverables