Wireless Software and Hardware platforms for Flexible and Unified radio and network controL

Project Deliverable D2.3

Results of first set of showcases

Contractual date of delivery:	31-12-2015
Actual date of delivery:	04-01-2016
Beneficiaries:	IMINDS, TCD, CNIT, TUB, NCENTRIC, RUTGERS, SNU
Lead beneficiary:	TCD
Authors:	Nicholas Kaminski (TCD), Luiz DaSilva (TCD), Ingrid Moerman (IMINDS), Peter Ruckebusch (IMINDS), Spilios Giannoulis (IMINDS), Pieter Becue (IMINDS), Anatolij Zubow (TUB), Mikolaj Chwalisz (TUB), Piotr Gawłowicz (TUB), Ilenia Tinnirello (CNIT), Pierluigi Gallo (CNIT), Domenico Garlisi (CNIT), Robin Leblon (nCentric)
Reviewers:	Mikolaj Chwalisz (TUB), Spilios Giannoulis (IMINDS)
Work package:	WP2 – General requirements and showcases
Estimated person months:	8
Nature:	R
Dissemination level:	PU
Version:	3.5

Abstract:

This deliverable reports on the results of the first set of showcases that have been implemented. This deliverable also includes the specifications of the second set of showcases to be implemented by the end of Year 2.

Keywords:

showcases, proof-of-concept

Executive Summary

This deliverable reports on the showcases implemented within the first year of the project to display the functionality of the WiSHFUL platform. A brief overview of the accomplishments within each showcase is provided alongside a description of the utility offered by the WiSHFUL platform and exhibited within this showcase. As such, this document provides an initial catalogue of the accomplishments of the project, by summarizing the tangible utilities provided. Along with subsequent two work package deliverables, this document will provide a complete picture of the capabilities and utility of the WiSHFUL project in a tangible manner by applying the functionality of the project to problems currently relevant to the community in a user-friendly and consistent manner. As such, the work outlined herein will provide the basis of upcoming scientific publications and other dissemination material as a means of maximizing the impact of the project. Technical detail regarding the implementation of support for each showcase is left to the appropriate technical deliverables (D3.2, D4.2 and D6.2). Finally, this document also looks ahead to showcases envisioned for implementation within the second year of the project.

List of Acronyms and Abbreviations

AM	Aggregate Manager
AODV	Ad hoc On-demand Distance Vector
AP	Access Point
API	Application Programming Interface
BAN	Body Area Network
CPU	Central Processing Unit
CSME	Carrier Sense Multiple Access
DMT	Discrete MutiTone
DSL	Digital Subscriber Loop
DSR	Dynamic Source Routing
DT	Delay Tolerant
EM	ElectroMagnetic
EMS	Experiment Management Server
EWMA	See Figure 8
F4F	Federation for FIRE (Future Internet Research Experimentation)
FBMC	Filter Bank Multi-Carrier
Fed4FIRE	Federation for FIRE (Future Internet Research Experimentation)
FRCP	Federated Resource Control Protocol
FSM	Finite State Machine
GPIO	General Purpose Input Output
HAL	Radio Abstraction Layer
HTTPS	HyperText Transfer Protocol Secure
I/Q	In phase / Quadrature
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
KPI	Key Performance Indicator
LTE	Long Term Evolution
LTE-A	Long Term Evolution - Advanced
LQI	Link Quality Indication
PRR	Packet Reception Rate
MA	See figure 8
MAC	Medium Access Control
MTC	Machine-Type Communications
NOC	Network Operations Center

OFDM	Orthogonal Frequency Division Multiplexing
OLSR	Optimised Link State Routing
OMF	OMF Measurement Library
OML	Orbit Management Framework
PLC	Power Line Communication
QoS	Quality of Service
RAM	Random Access Memory
RF	Radio Frequency
RP	Radio Processor
RSpec	Request Specification
RSSI	Received Signal Strength Indication
RT	RealTime
SDR	Software Defined Radio
SFA	Slice Federation Architecture
SSH	Secure SHell
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TDMA	Time Division Multiple Access
TSMP	Time Synchronized Mesh Protocol
TSCH	802.15.4e
UC	Use Case
UMTS	Universal Mobile Telecommunications System (UMTS)
UPI	Unified Programming Interface
UPIR	Unified Programming Interface radio
UPIN	Unified Programming Interface network
UPIHC	Unified Programming Interface hierarchical control
URL	Uniform Resource Locator
USB	Universal Serial Bus
VPN	Virtual Private Network
Wi-Fi	Wireless Fidelity
XFSM	eXtended Finite State Machine

Table of contents

1 Introduction

This deliverable reports on the outcome of the initial showcases within the WiSHFUL project, as planned in WP2 on *General requirements and showcases.* As such the material herein is aligned with all work package objectives, but specifically targets the last one, particularly regarding the developing of convincing means to promote WiSHFUL: "***Definition of relevant and convincing showcases in view of promoting the WiSHFUL capabilities."***

This document focuses on providing a catalogue for the results of the first set of WiSHFUL showcases. The description given within this document highlights the currently relevant problem addressed by each showcase and provides an overview of the utility of the WiSHFUL framework in addressing this problem. Further, the document provides an indication of potential extensions or continuations of the work completed so far. As presented, the material herein focuses on providing a senses of the value of the WiSHFUL project to the broader community; technical details regarding the implementation of each showcase is left to the appropriate technical deliverable(s) (D3.2, D4.2 and D6.2).

Finally this document introduces concepts for the showcases that will drive the development of the WiSHFUL framework within the second year of the project.

2 Results of first set of Showcases

2.1 Efficient Airtime Management

A widely known problem experienced in IEEE 802.11 (Wi-Fi) networks is performance degradation due to co-channel interference caused by *hidden nodes*. The impact can be mitigated by preventing overlapping transmissions (in time) between co-located Access Points (APs) by efficient airtime management through interference avoidance techniques.

In order to support this showcase the Wishful framework must provide the following functionality (Figure 1):

- Detection of wireless links suffering from ***hidden node problem***, i.e. performance degradation due to packet collisions.
- ***Identification of packet flows*** using those wireless links being affected by hidden node problem,
- Setting up ***hybrid TDMA MAC*** scheme in which two wireless links suffering from the hidden node problem are getting exclusive time slots assigned.
- As we have to coordinate the medium access of a set of nodes, all wireless access needs to be ***time synchronized***.

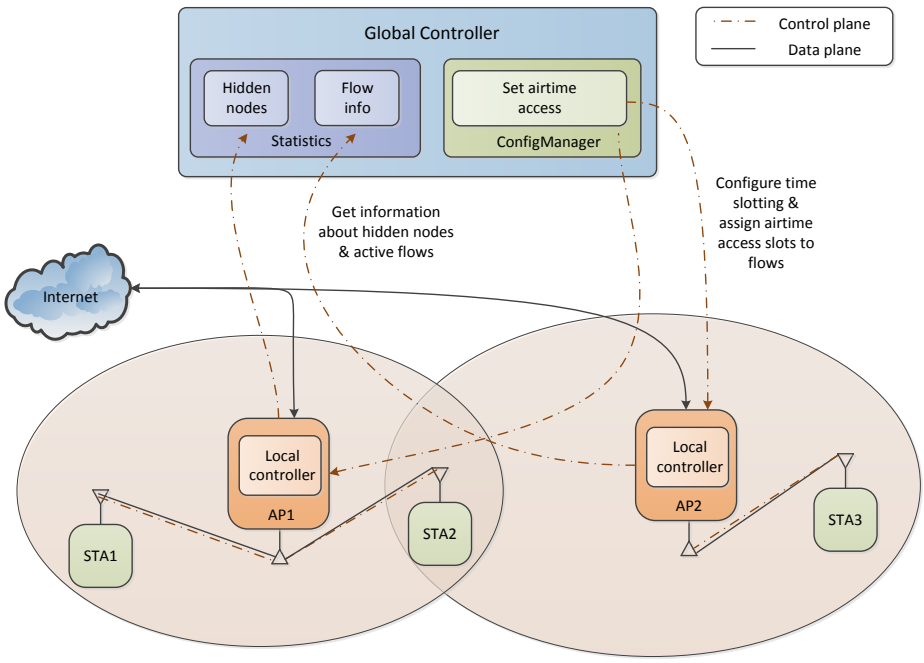

Figure 1. Illustration of the Wishful controller architecture used for this showcase. Clearly visible is the separation of control and data plane where the local agents execute control commands of the global control program.

2.1.1 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 2). 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 hidden by some other nodes.

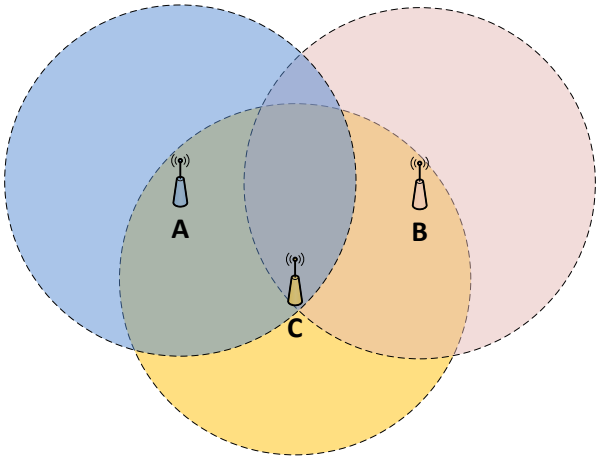

Figure 2. Example illustrating a hidden node scenario. As nodes A and B are outside their carrier sensing range the packet transmissions from A and B would collide at node C.

a. Presentation of UPIs used

For hidden node detection Wishful provides the following UPI network functions which are used by global controllers:

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

Given a set of nodes using the specific wireless interface (e.g. ath0), radio channel (e.g. 6) and detection threshold (e.g. 0.9) this functions returns a boolean matrix indicating which nodes are inside their carrier sensing range and which are outside.

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

Given a set of nodes using the specific wireless interface (e.g. ath0), radio channel (e.g. 6) and detection threshold (e.g. 0.9) this functions returns a boolean matrix indicating which nodes are inside their communication (reception) range and which are outside.

These two functions are used to detect links hidden by some node. As an illustrative example, consider the case where A and B are outside of carrier sensing range and C is inside the reception range of 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.

b. 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 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 uses the following approach. First it sets each wireless node in sniffing mode. Second, 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.

c. Next Steps

The next step is the dissemination of this WiSHFUL functionality through scientific publication and demonstration of the broader showcase.

2.1.2 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 & cloud storage. Moreover, such devices and applications require much better mobility support and higher QoS/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, we are focusing on the downlink whereas in the future also the uplink will be considered. From the literature lots of protocols are known which would benefit from such a feature.

Following the Software-defined networking (SDN) paradigm we separate the control plane from the data plane and provide an API to allow local or global control programs 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 superframe. 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 slots we can configure whether carrier-sensing is activated or not. The latter would results in the classical TDMA MAC. 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 control program which takes as input the channel access scheme specified by applications. Any application is self-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.

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 under test. The utilized Precise Time Protocol (PTP) gives an accuracy in microsecond level.

The Wishful agent running on each AP locally is responsible for coordination of channel access as configured by the local or global control program.

Wishful provides a hybrid TDMA MAC on commodity devices. So far a connector is provided for Linux boxes using Atheros Wi-Fi chips supporting the Ath9k driver. Specifically, we provide a patch to the Linux Compat Wireless. The implementation was tested with Intel x86 nodes.

a. Presentation of UPIs used

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

def setActive (self, node, interface, mac_profile)

def setParameterLowerLayer (self, node, interface, param_key,param_value)

def setInactive (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 3**).

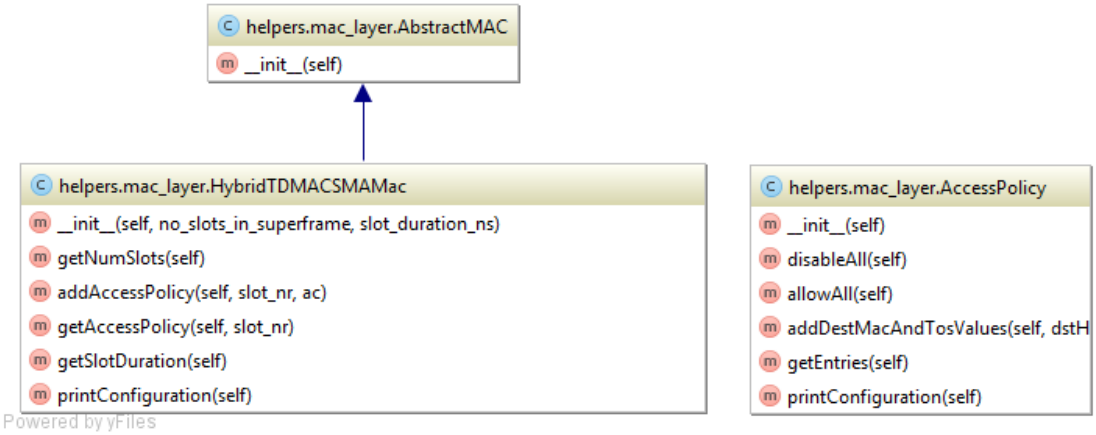

Figure 3. UML class diagram showing the hybrid MAC relevant configuration.

The following example show how to set-up a new hybrid MAC instance:

```
# create new MAC for each node
mac = HybridTDMACSMAMac(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

radioHelper. setActive (node, iface, mac)
```

Finally, the **Figure 4** 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.

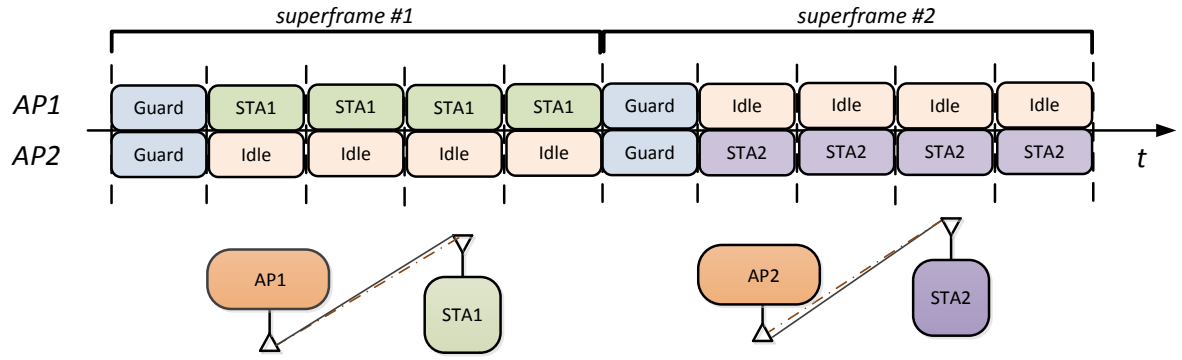

Figure 4. Illustration of exclusive slots allocation in TDMA.

b. Results

Figure 5 shows how the UPI functionality was 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 global controller (*setActive ()* command), it starts 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.

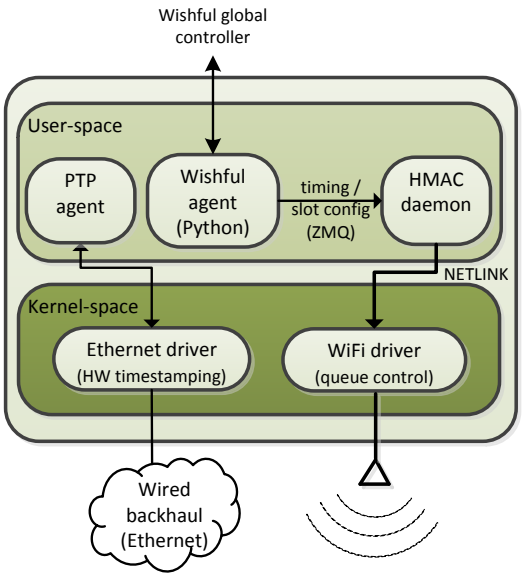

Figure 5. Overview of the components on the wireless node in the Linux-Wi-Fi prototype.

In order to evaluate the proposed efficient airtime management we ran experiments in the TWIST 802.11 testbed. We use Ubuntu 14.04, Intel i5s with a wired Ethernet NIC from Intel supporting HW timestamping and an Atheros 802.11n wireless chip.

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

In the following we present results 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 6** 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.

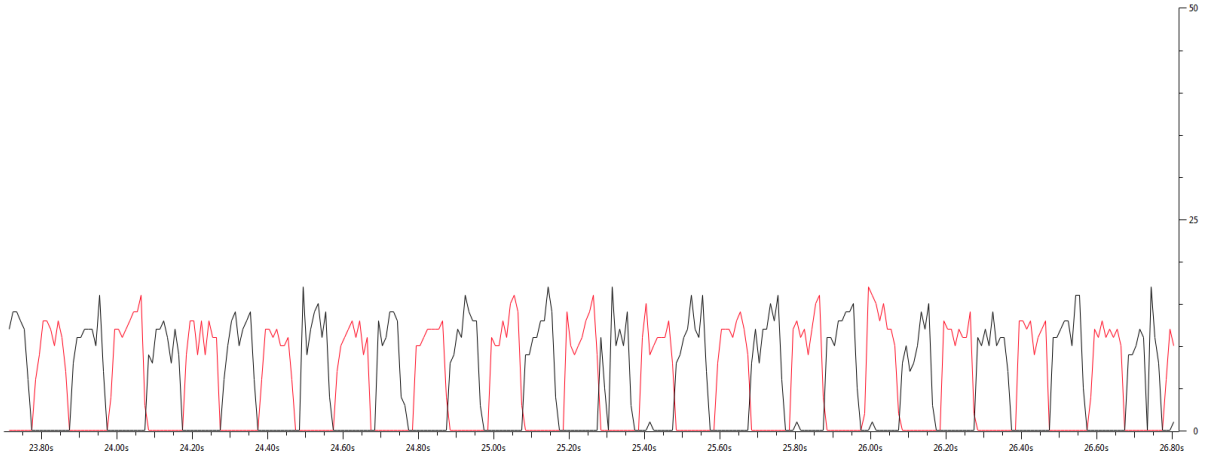

Figure 6. IO graph illustrating the number of packets sent over time. The color indicates a particular flow.

The performance improvement compared to standard 802.11 DCF is show in **Figure 7**. On this particular link the throughput could be increased by a factor of 5.2 and 2.8 respectively.

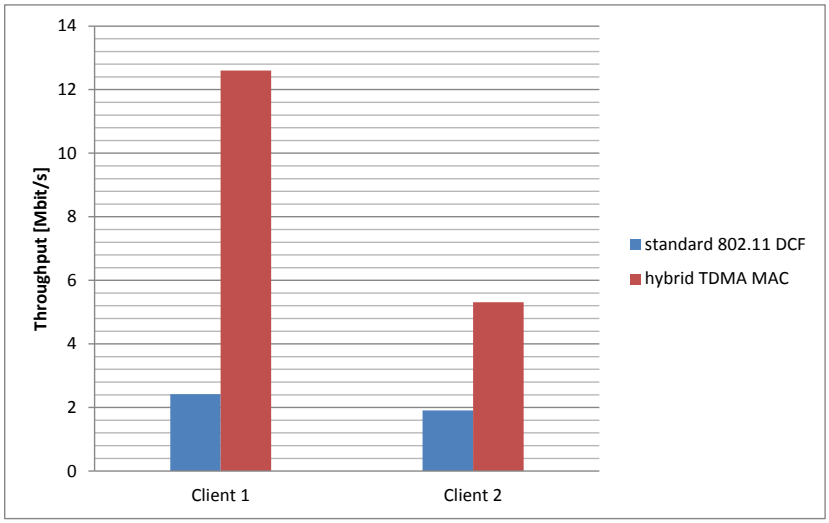

Figure 7. TCP/IP performance.

The described efficient airtime management was fully implemented. The source code is available in the project Github repository under examples/showcase1/start_sc1.py.

c. Next Steps

The following extensions to the hybrid TDMA MAC provided by Wishful are desirable:

- Make it OpenFlow like, i.e. identify flows using 5-tuple and define actions which specify how a flow should access the wireless channel,
- So far the control program works proactively which might be not sufficient for some applications using the hybrid TDMA MAC. Hence, an extension towards a reactive approach might be desirable, i.e. agents inform the control program about newly arrived flows for which no channel access was configured.

2.2 Co-Existence of Heterogeneous technologies

It is well-known that coexisting heterogeneous technologies, operating on the same wireless channel, interfere with each-other. Examples of two such technologies are IEEE-802.11 (Wi-Fi) and IEEE-802.15.4e (TSCH). In this showcase, an interference mitigation strategy is implemented on top of the WiSHFUL UPIs. Two sub-cases are considered, based on which technology is the primary user of the wireless medium, i.e. who gets priority for accessing the medium.

- TSCH as the primary user. Wi-Fi defers channel access when its transmission could collide with TSCH traffic. This sub-case requires fine-grained synchronization between the Wi-Fi and TSCH networks because the Wi-Fi nodes need to know exactly when a TSCH timeslot starts. Moreover, also the TSCH slot allocation and hopping scheme needs to be known to the Wi-Fi nodes in order to calculate when collisions could occur.
- Wi-Fi is the primary user. TSCH blacklist IEEE-802.15.4 channels that collide with the currently used IEEE-802.11 channel (multiple IEEE-802.15.4 channels are affected by one IEEE-802.11 channel). This sub-case requires that the global control program, used by TSCH, is able to retrieve the Wi-Fi channel(s) currently in use, calculate which channels need to be blacklisted and distribute the new hopping sequence (without the blacklisted channels) to each of the sensor nodes participating in the TSCH network.

Figure 8 illustrates both sub-cases. In sub-case 1, Wi-Fi delays channel access when it collides with a TSCH timeslot. In sub-case 2, the TSCH channels that collide with the Wi-Fi channel are blacklisted.

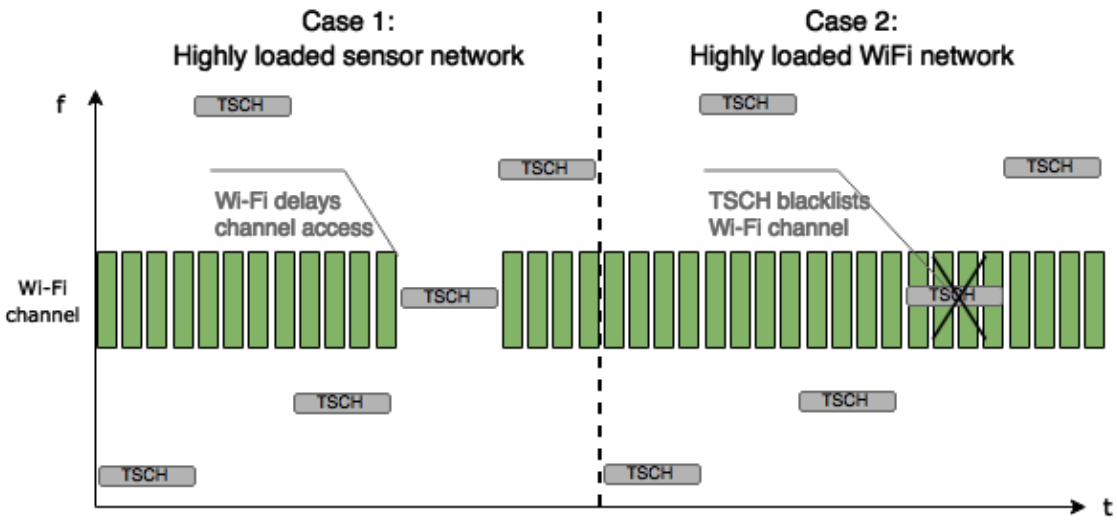

Figure 8. Co-existence scheme for TSCH (IEEE-802.15.4e) and Wi-Fi (IEEE-802.11) networks.

Due to timing constraints, sub-case 1 could not be fully implemented using the currently defined and public WiSHFUL UPI functions. The main problem is that delegation of control is required between the global control program, controlling both TSCH and Wi-Fi networks in this showcase, and the local control programs, controlling either a TSCH node or a Wi-Fi node. This is necessary because the timing requirements for this sub-case are very strict, i.e. the Wi-Fi nodes need to exactly defer transmission when the colliding TSCH slot starts (with slot duration of 10ms in the default setting). For this purpose UPI_HC will be developed and used in Y2. In the current implementation however, the building blocks are already provided for future development of this sub-case. Also preliminary results are given illustrating how TSCH and Wi-Fi networks can be synchronized.

2.2.1 Subcase1: TSCH is primary user.

In the proposed scenario it is possible to distinguish between three systems that keep track of its own time reference:

1. The IEEE802.15.4e TSCH based sensor network is a tightly synchronized wireless system where a particular node adapts its clock based on the clock of its time parent. A TSCH node does not have a direct notion of the absolute time reference.
2. Linux hosts keep own time reference in UNIX Epoch time and can synchronize it with other nodes over NTP or over PTP where high timing accuracy is required.
3. The IEEE802.11 uses Time Synchronization Function (TSF) to allow synchronization between devices in one BSS (Basic Service Set) e.g. AP puts own TSF counter value in the beacon packets, and all stations synchronize to it.

In order to achieve the tight cooperation between Wi-Fi devices and sensor nodes, it is necessary to synchronize both networks. Let's assume a simplified problem with the focus on synchronization between one sensor node (NXP JN5168 running TinyOS version of TSCH) and one Linux host machine equipped with a Wi-Fi card. There are two main options to synchronize those two devices: (a) directly using the wired connection; or (b) indirectly using an inter-technology beacon send over a wireless link. Both of the solutions have advantages and disadvantages. The wired connection promises a more accurate and easier solution but requires tight coupling between the devices. On the other hand, the wireless solution is much more flexible in terms of hardware coupling but requires more overhead in the detection of the signals sent by the different and in general incompatible technologies. In the next sections we will analyse both wired and wireless approaches.

The required synchronization accuracy depends on the time slot duration of the IEEE802.15.4e network. Although it is not fixed in the standard the typical value is 10 milliseconds. This means that Wi-Fi should be able to cease own transmissions for the duration of the TSCH timeslot. The inaccuracies of the synchronization can be adjusted with additional guard times, but it will additionally limit the performance of Wi-Fi network.

There is still a problem of schedule alignment with the synchronized networks. It is necessary to be able to calculate when the overlapping TSCH channel will be used. The straightforward solution is to defer any Wi-Fi transmission, when a TSCH time slot is scheduled to be transmitted on an overlapping Wi-Fi channel. A more advanced solution also takes the interference range of Wi-Fi into account and only defers transmissions when the TSCH nodes are in interference range. The advanced solution improves the Wi-Fi performance because the number of transmission deferrals is reduced, without having an effect on the performance of the TSCH network. This problem is orthogonal and will not be considered in Year 1.

a. Presentation of UPIs used

The following code snippets illustrate how the UPIs can be used to implement the aforementioned interactions. Currently only UPI_R/N functions are used remotely from a global control program. For this purpose UPI_G is used via the UPI_G_Helper class, which remediates some of the burden for calling UPI_R/N functions remotely from a global control program.

First, the global control program needs to retrieve a list of nodes, together with their specific role in the network (i.e. PAN coordinator or end device in the TSCH network and access point or station in the Wi-Fi network). It must first create an UPI_G_Helper and retrieve the available nodes, identifying their role within their respective networks:

```
# get reference to global UPI
global_helper = GlobalHelper()

# nodes under control
nodes = global_helper.getNodes('COEXISTANCE_SHOWCASE')
sta_nodes = []
ap_node = Node()
ed_nodes = []
pc_node = Node()
for node in nodes:
      if node.role == 'PAN_COORDINATOR':
            ap_node = node
      elif node.role == 'ACCESS_POINT':
            pc_node = node
      elif node.role == 'END_DEVICE':
            ed_nodes.append(node)
      elif node.role == 'STATION':
            sta_nodes.append(node)
```

Then it must retrieve the allocated slots (e.g. slotframe), channel hopping sequence and slotframe duration from the TSCH pan coordinator, and the channel used by the Wi-Fi access point. Note that, in the current implementation, we assume that only one Wi-Fi channel is used.

```
tsch_param_keys = ['802154E_SLOTFRAME','802154E_HOPPINGSEQUENCE',
'802154E_SUPERFRAMEDURATION']
tsch_param_key_values = global_helper.getParameterLowerLayer(pc_node,
'wpan0', tsch_param_keys)

wifi_param_keys = ['80211_CHANNEL']
wifi_param_key_values = global_helper.getParameterLowerLayer(ap_node,
'wlan0', wifi_param_keys)

global_helper.defineEvent(pc_node,   'wpan0',   '802154_SUPERFRAME_STARTED',
superframe_started_callback)
```

Then the actual interference avoidance is implemented. The current implementation requires that the TSCH pan coordinator generates an event every time a slotframe starts, indicating the absolute slot number (ASN) of the first slot in the slotframe. The global control program registers a callback function that takes as arguments the start time of the slotframe and the ASN. When the event is generated, the callback function calculates the collisions that will occur **in the slotframe following the currently active one**. This allows scheduling the freezing of Wi-Fi traffic well in advance. The example uses two functions to calculate (a) the time offsets and channels in the next slotframe in which TSCH will be active; (b) the offsets at which Wi-Fi should freeze transmissions to avoid collisions. Note that, in the current implementation, we assume that only one Wi-Fi channel is used.

```
def slotframe_started_callback(slotframe_start_timestamp,
absolute_slot_number):
      timestamp = convert_to_unix_time(slotframe_start_timestamp)
      slotframe  = tsch_param_key_values['802154E_SLOTFRAME']
      hopping_sequence = tsch_param_key_values['802154E_HOPPINGSEQUENCE']
      slotframe_duration =
tsch_param_key_values['802154E_SLOTFRAMEDURATION']
      wifi_channel = wifi_param_key_values['80211_CHANNEL']
      #returns a list with tuples [active_slot_offset,channel]
      tsch_offsets = calc_offsets_and_channels_for_active_slots(slotframe,
hopping_sequence, absolute_slot_number+len(sloframe))
      #returns a list with offsets on which the Wi-Fi should freeze
transmissions
      wifi_freeze_offsets = calc_freeze_offsets(tsch_offsets,wifi_channel)
      for offset in wifi_freeze_offsets:
            #duration is 10 ms + 10 ms guard
            param_key_values = {'80211_TXFREEZE': 20}
            #schedule 5 ms before start of slot in following slotframe
            exec_time = timestamp + slotframe_duration + offset - 5
            global_helper.setParameterLowerLayer(ap_node,param_key_values,
exec_time)
      pass

# define an event that calls the 'slotframe_started_callback' on start of
the slotframe
global_helper.defineEvent(pc_node, 'wpan0', '802154_SLOTFRAME_STARTED',
slotframe_started_callback)

while True:
      time.sleep(tsch_param_key_values['802154E_SLOTFRAMEDURATION'])
```

The code examples rely only on the scheduling functions of UPI_G, which is heavily influenced by the level of internal synchronisation between the different local monitoring and configuration engines. With the current synchronisation method it is hard to achieve reproducible results. Therefore, in Y2, it is foreseen that UPI_HC will be used to enable communication between the different local control programs and the global control program allowing a more fine grained operation.

b. Results

The most commonly used method for connecting sensor nodes to a host PC are USB and GPIO (general purpose IO). USB is very popular because it provides an easy to use and flexible interface. On the other hand GPIO allows connecting any device to the sensor node and gives full control to the developer. Using GPIO however, is platform specific and renders issues when porting to different platforms. By connecting the sensor node to the Linux host over a USB connection, and providing the synchronization over such link, allows a portable and plug and play cross-technology synchronization solution.

The biggest challenge with the USB solution is the limited predictability of the USB transmissions, namely the delay and jitter are not defined and can actually change based on how the sensor node is connected to the host. For instance, the delay will change if the sensor node is connected via a USB hub. Moreover, the jitter on the delay increases when the USB port is also used for different purposes.

The GPIO solution has the advantage of removing any additional protocol overhead, and thus promising lowest jitter and delay values. On the other hand it is much more constrained than USB. It requires from both sensor node and host PC to have external GPIO interfaces. For the sensor node

this usually means that the evaluation/development board exposes the required pins. Also not all host PCs, or in view of the full showcase also Wi-Fi routers, expose the required GPIO pins.

In order to test the applicability of wired synchronization solutions the BeagleBoneBlack (BBB) is combined with the NXP JN5168 evaluation board. The synchronization solutions were tested using both a USB and GPIO solution.

As can be seen in Figure 9, the NXP JN5168 carrier board was connected to BBB over USB and two GPIO pins (providing simple LOW/HIGH signals in both directions).

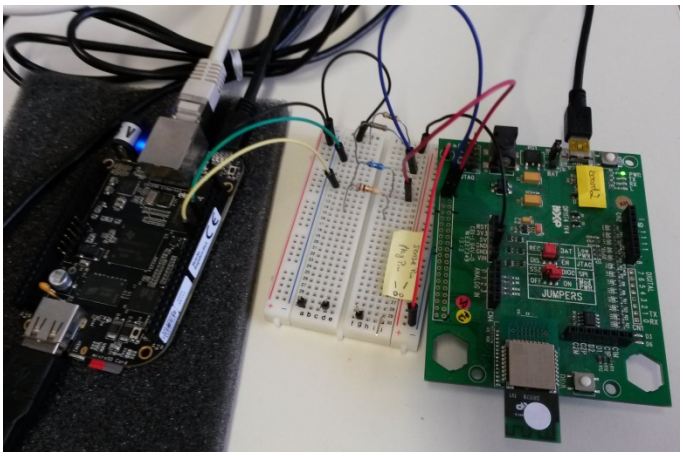

Figure 9. BeagleBoneBlack and NXP JN5168 Interfacing

The first experiment analyses the total round trip delay GPIO ping test. Namely the BBB was setting the output pin to HIGH state and measuring the time until the input pin (coming from sensor node) was set to HIGH state. The 48h experiment shows that the average delay in such experiment is 879µs. The histogram shown in Figure 10, lists the measured distribution of the round trip delays.

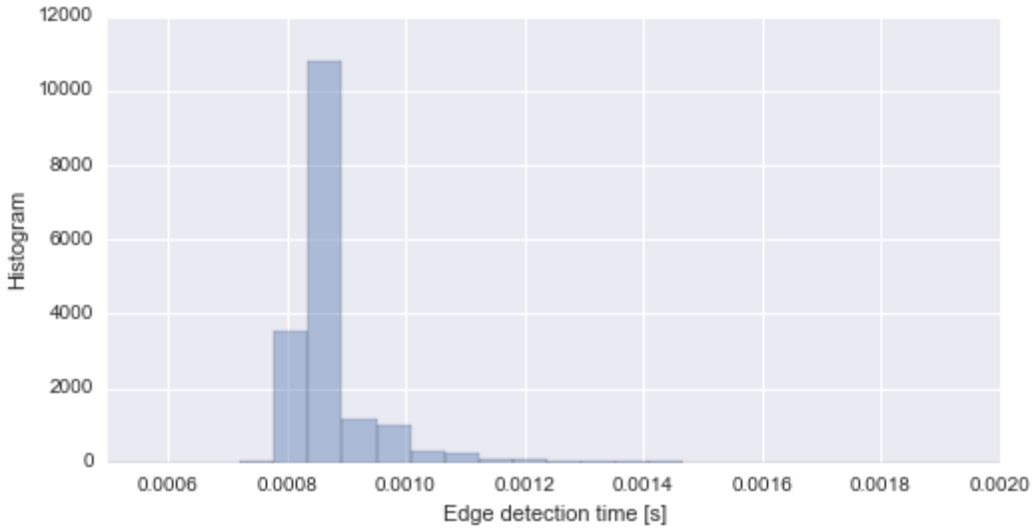

Figure 10. Measured Distribution of Round Trip Delays.

The biggest delay in the process comes from the HIGH state signal detection on the BBB side, compared to which the detection on sensor node is negligible. This is because the detection in the

BBB is done from the Linux user space, and can be expected to be improved by developing a kernel driver for clock signal detection from the sensor node.

In the second experiment the BBB collected in parallel timestamps each time the packet from sensor node with current ASN number was received over USB and each time GPIO pin was set to HIGH state by sensor node. In this scenario the sensor node was working according to the simple TSCH schedule (only one slot active in the whole slot frame) and was sending current ASN over USB and setting the output GPIO pin to HIGH every begging of slot frame. The slotframe was 1s long in this scenario. From the scatter plot, shown in Figure 11, the difference in jitter coming from a GPIO based trigger versus the USB one. On the sides there are the distributions of each particular value respectively. From the top plot we can see that the GPIO jitter is still small and stable on the other hand the delays between consecutive USB triggers varies in the range of 20ms, which is too much for our requirements and thus cannot be used directly.

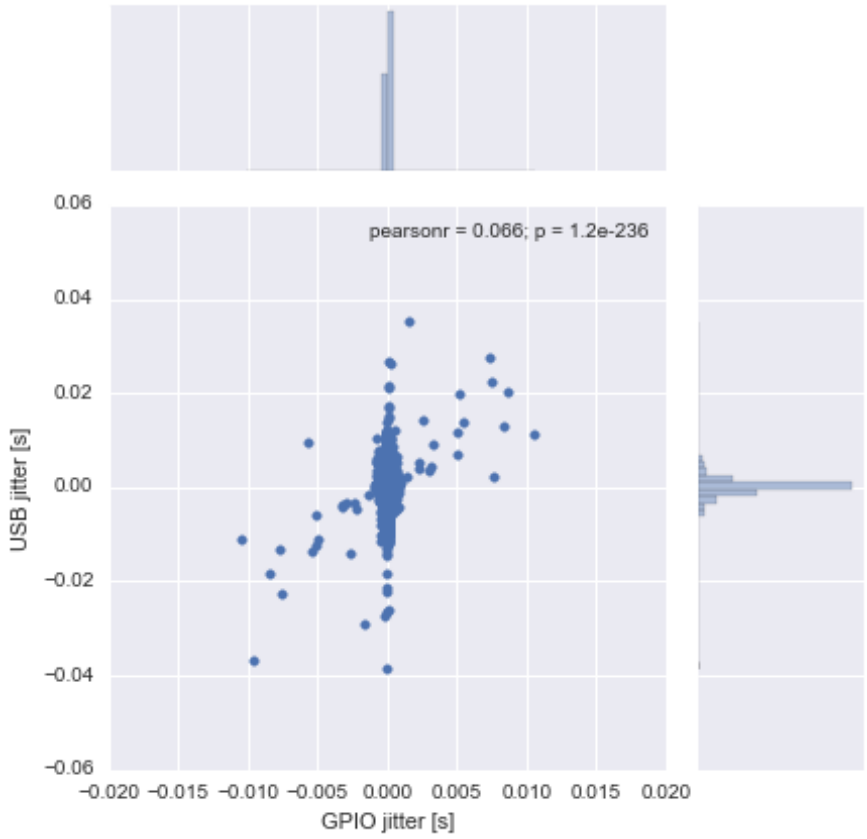

Figure 11. Scatter Plot of GPIO- vs USB-Trigger Jitter.

It is important to note that there was no other USB communication during the tests, implying that the results are an optimistic estimation for real-life jitter. This is something that was also noticed during evaluation of this showcase. A delay variance of 20 ms on the USB is too high for the aforementioned global control program to work properly. Because the sensors are attached via USB on the testbeds, no large-scale testing could be performed.

c. Next Steps

Due to the incompatibility of the PHY layers between IEEE802.11 and IEEE802.15.4 it is not possible to directly capture packets. According to [1] it is however possible to detect beacons by means of

spectrum sensing and statistical analysis of the RSSI data. The solution proposed in [1] works as long as the frequency channel is shared between those technologies. Additionally the receiving device must sense the spectrum constantly in order to properly analyse the data (to perform RSSI based beacon detection).

The proposed solution will however not work directly as the TSCH employs channel hopping and there is no fixed frequency for sending beacon packets. This makes it much more challenging to detect beacons (each beacon would be send on different frequency) by the Wi-Fi node. Also, due to the channel hopping property the sensor node cannot constantly monitor the Wi-Fi channel for its beacons. For this reasons the wireless synchronization link hasn't been investigated in detail yet and will be studied in more detail in future.

2.2.2 Subcase1: Wi-Fi is primary user.

In this sub-showcase Wi-Fi is the primary user of the network. When a Wi-Fi stream is started on a particular channel, the colliding IEEE-802.15.4e channels are blacklisted in the TSCH network. To demonstrate this, a global control program was implemented that uses UPI_R functionality to (a) configure the Wi-Fi network to operate on a particular 802.11 channel; and (b) adapt the hopping scheme in the TSCH network in order to avoid 802.15.4 channels that collide with the 802.11 channel.

A single IEEE-802.11 channel spans 4 IEEE-802.15.4 channels. The implemented blacklisting strategy avoids Wi-Fi interference by adding 5 to each blacklisted channel. If the updated channel is larger than 26 (max), the number of channels (16) is subtracted.

```
new_channel = channel+5 if channel+5 <= 26 else channel+5 - 16
```

Using this simple method, the length of the hopping sequence can remain the same. It can however lead to internal collisions when using multiple slots in one slotframe. In such advanced TSCH networks, the internal collisions should also be removed by also reconfiguring the slots. This will be investigated in Year 2.

a. Presentation of UPIs used

The global control program that implements this showcase is quite straightforward. The initialisation phase is similar to the previous subcase, and then the following three steps are taken:

1. First 90 seconds: start Iperf traffic on the TSCH end devices (iperf clients). The TSCH pan coordinator forwards to iperf traffic to an iperf server running on its Linux host-pc.
2. Second 90 seconds: configure Wi-Fi network and generate traffic
    a. Configured for high throughput by modifying the EDCA parameters and the IEEE-802.11 channel.
    b. Iperf traffic is started with the Wi-Fi station as client and the access point as server.
3. Third 90 seconds: blacklist Wi-Fi channels in TSCH network
    a. Get the hopping scheme currently used by the TSCH nodes.
    b. Calculate a new hopping scheme by using the strategy defined above.
    c. Apply the new hopping scheme by reconfiguring the TSCH nodes.

The following code snippet illustrates how this is implemented:

```
# 1) start traffic on TSCH nodes
global_helper.startIperfServer(pc_node,'wpan0')
global_helper.startIperfClient(ed_nodes,'wpan0',pc_node.getIpAddress())
time.sleep(90)

# 2) Configure Wi-Fi nodes and start traffic
```

```
# a) configure
wme_ac_vo_queueID = 4 ; wme_ac_vo_aifs=2 ; wme_ac_vo_cwmin=2 ;
wme_ac_vo_cwmax=3 ; wme_ac_vo_txop_limit=47
global_helper.setEdcaParameters([sta_nodes , ap_node], 'wlan0' ,
wme_ac_vo_queueID, wme_ac_vo_aifs, wme_ac_vo_cwmin, wme_ac_vo_cwmax,
wme_ac_vo_txop_limit)
global_helper.setRfChannel([sta_nodes , ap_node], 'wlan0' , 1)
# b) start traffic
global_helper.startIperfServer(ap_node,'wlan0')
global_helper.startIperfClient(sta_nodes,'wlan0',ap_node.getIpAddress())
time.sleep(90)

# 3) blacklist TSCH channels by modifying hopping sequence
# a) get current hopping sequence
param_keys = ['802154E_HOPPINGSEQUENCE']
tsch_param_key_values =
global_helper.getParameterLowerLayer(pc_node,'wpan0', param_keys)
tsch_hopping_sequence = tsch_param_key_values['802154E_HOPPINGSEQUENCE']
# b) calculate new hopping sequence
blacklisted_channels = [11,12,13,14]
good_channels = [16,17,18,19]
new_hopping_sequence = []
index = 0
for channel in tsch_hopping_sequence:
      if channel in blacklisted_channels:
            new_channel = channel+5 if channel+5 <= 26 else channel+5 - 16
            new_hopping_sequence.append(new_channel)
      else:
            new_hopping_sequence.append(channel)
# c) set new hopping sequence
tsch_param_key_values['802154E_HOPPINGSEQUENCE' = new_hopping_sequence
global_helper.setParameterLowerLayer([pc_node,
ed_nodes],'wpan0',tsch_param_key_values)
time.sleep(90)
```

b. Results

The sub-showcase was implemented and evaluated in a 9 node set-up on the IMINDS w-iLab.t wireless testbed located in Zwijnaarde. The TSCH network is running on 7 sensor nodes. In the testbed, each sensor node is connected to a Linux host pc. The local monitoring and configuration engine (MCE) is executed on this PC, implying out-of-band control of the TSCH network. The local MCE enables to fine-tune various parameters of the TSCH network.

There are also two Linux Wi-Fi nodes (one access point and one station) equipped with an Atheros Wi-Fi chip and using the ATH9k wireless driver. Both nodes are configured in 802.11n mode on a single channel via the UPIs. Also the EDCA parameters (aifs, cwmin, cwmax, txOP) are tweaked using the UPIs, resulting in an average throughput of +- 90 Mb/sec. This throughput is sufficient to cause serious interference on the TSCH network, as will be demonstrated in this section.

The results were collected by using a WiSpy for measuring the spectrum activity and by analysing the iperf statistics during the test.

In the first stage there is no Wi-Fi interference and all IEEE-802.15.4 channels can be used. The spectrogram shown in Figure 12 demonstrate the hopping behaviour of the TSCH network (yellow dots are TSCH packets. There was also, uncontrolled, external interference from other experiments.

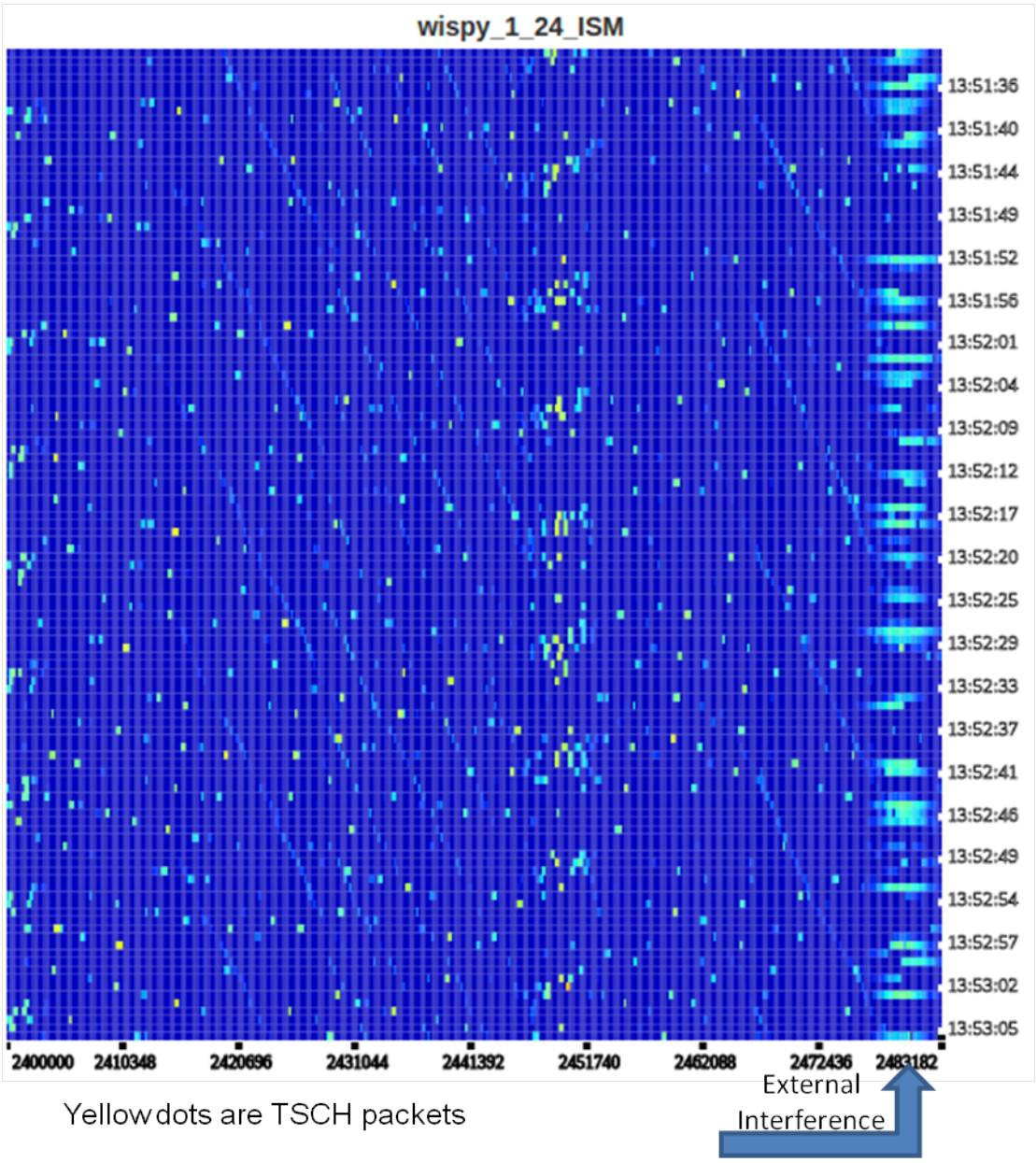

Figure 12. The spectrogram shows the activity in the 2.4 ISM band when only TSCH network is active. Note that also other experiments were conducted during this experiment. Hence there was external interference.

In the next stage, also Wi-Fi traffic was started on the first IEEE-802.11 channel. This is clearly visible on the left part of the spectrogram shown in Figure 13. As can be seen, the TSCH network also hops to channels that collide with the Wi-Fi channel.

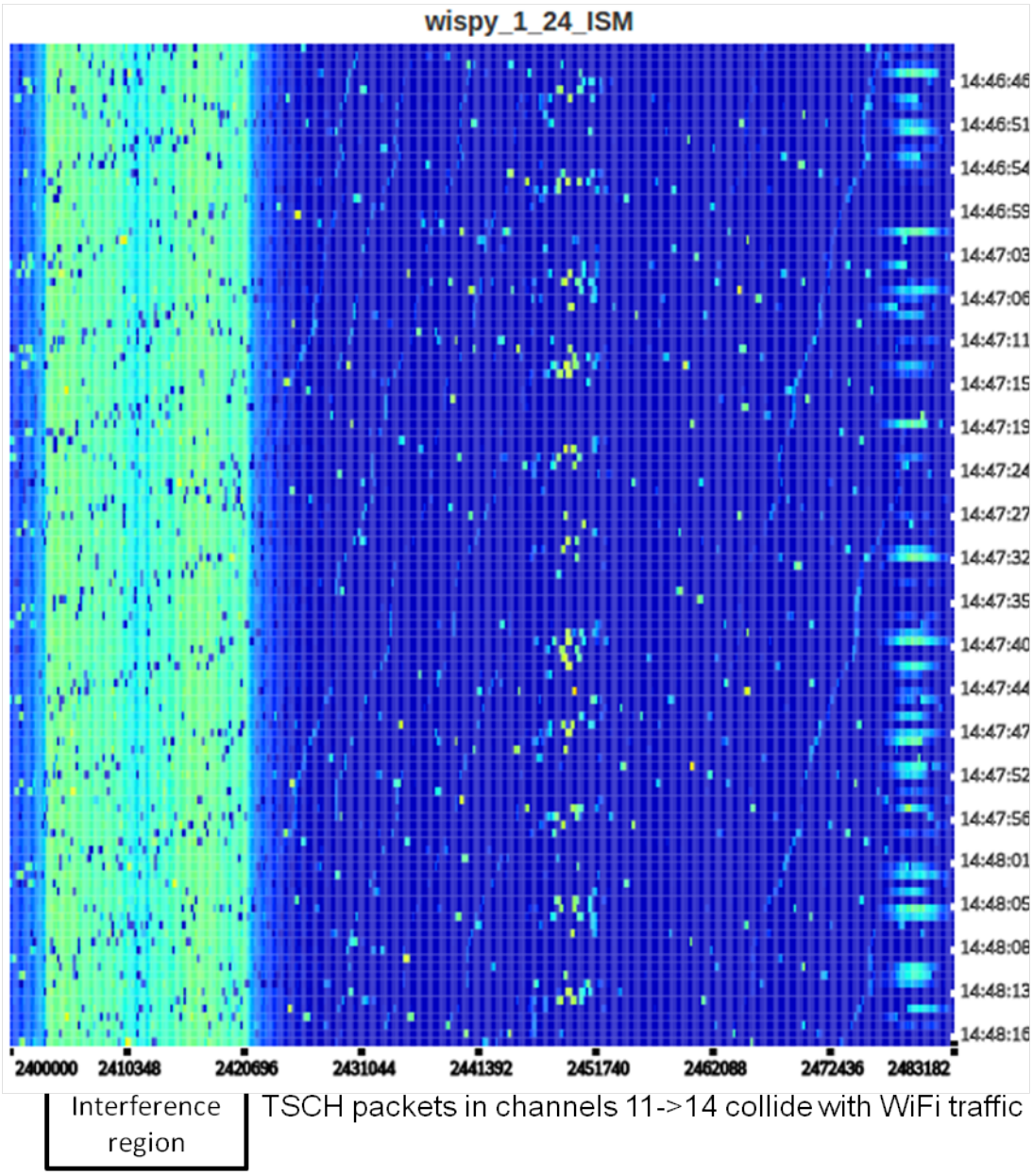

Figure 13. The spectrogram shows the activity in the 2.4 ISM band when both TSCH and Wi-Fi networks are active.

In the last stage, the blacklisting techniques are applied and the Wi-Fi channel is avoided in the channel hopping process of the TSCH network. The spectrogram shown in Figure 14 illustrates this. Now, no TSCH traffic is detected on the Wi-Fi channel and more packets can be detected on the other channels.

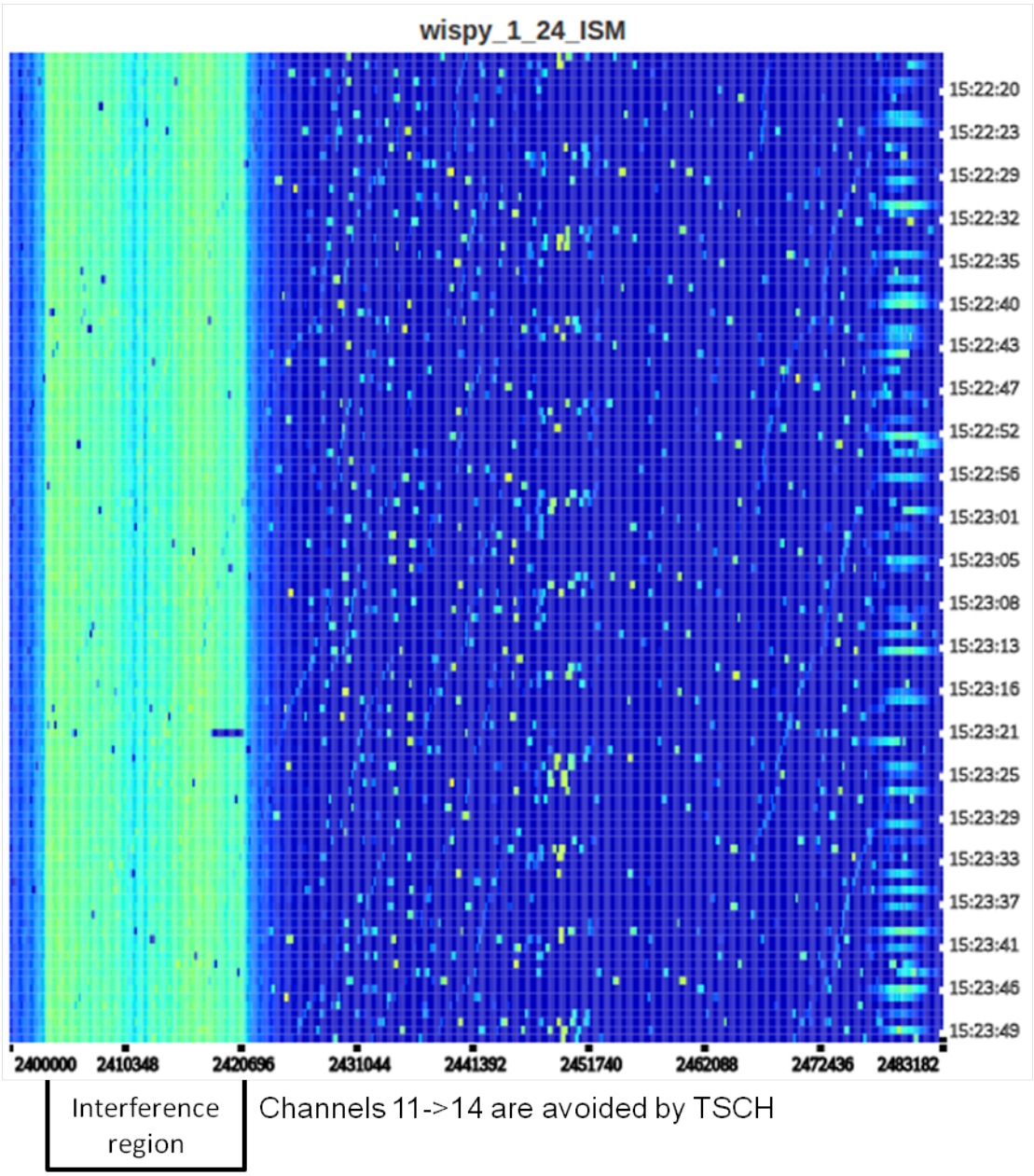

Figure 14. The spectrogram shows the activity in the 2.4 ISM band when both TSCH and Wi-Fi networks are active after blacklisting channels 11 through 14.

During these three stages also the iperf statistics (output iperf server) were logged. The left part of Figure 15 shows the overall percentage of packet loss, the right part the average network throughput in kbits/sec. In each graph the results are shown from left to right for: (a) a TSCH network operating on all 16 channels; (b) a TSCH network operating on all 16 channels with Wi-Fi interference; (c) a TSCH network with blacklisted channels 11->14; and (d) a TSCH network with blacklisted channels 11->14 and Wi-Fi interference.

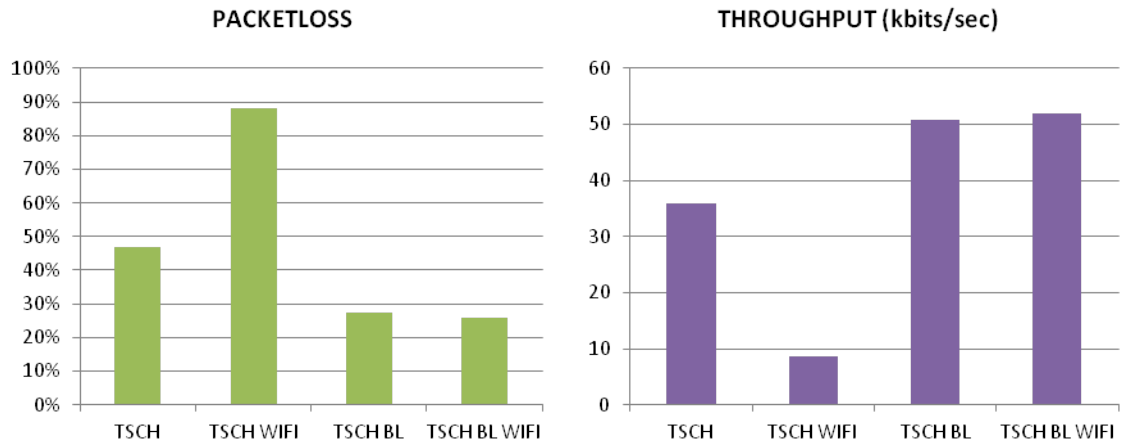

Figure 15. iPerf server statistics, packet loss (%) and throughput (kbits/sec), obtained while receiving data from 6 iPerf clients in the four different traffic scenarios (TSCH only, TSCH and Wi-Fi, TSCH with blacklisted channels and Wi-Fi). There was also external interference on channel 20.

The results show that packet loss without blacklisting in case of interference is close to 90%. The throughput drops below 10 Kbits/sec. With blacklisting the packet loss is below +- 30% and the throughput increased to 50 Kbits/sec. Note that the results are suboptimal since other traffic was also present during the experiment. In Y2 more advanced techniques will be applied that allow dynamically detecting activity on TSCH channels and blacklisting accordingly.

c. Next Steps

The implementation of the global control program is can be easily extended to allow a more dynamic behaviour. The following extensions can be considered:

- Use UPI_N to detect and steer Wi-Fi flows to particular channels.
- Use UPI_R to detect if the activity level on a Wi-Fi channel exceeds a certain threshold and should be blacklisted in the TSCH network.
- Enable advanced TSCH networks with multiple slots in a slotframe by removing internal collisions. This can be done by reconfiguring the allocated slots in each slotframe.

If local control programs are added, it would also be possible to detect traffic flows generated by non-WiSHFUL nodes. This can be done by monitoring the packet loss on each IEEE-802.15.4 channel used by TSCH. If the packet loss exceeds a certain threshold it should be blacklisted.

2.3 Load & 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; 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 can only be configured statically, and cannot be adapted to the varying network conditions.

In this showcase we want to demonstrate *how the WISHFUL UPIs can be exploited for implementing a technology-independent MAC adaptation logic* able to: i) dynamically tune the contention parameters of contention-based protocols as a function of the load and interference conditions experienced in the network, and ii) switch to a time-division access protocol in case of severe

congestion levels. The logic can work on WLAN or WPAN 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;
- Global coordination of MAC switching from CSMA to TDMA.

We demonstrate the utilization of both **local control and global control** logic. We consider a wireless network with a time-varying number of active nodes under the same contention domain (where all the nodes are in radio visibility), while a wired ethernet network is available as a control network between the global monitoring and configuration engine (running the global control program), the wireless stations and the access point. Each node runs a local optimization function that is loaded by the global control program for tuning the contention window of a CSMA protocol as a function of the network load.

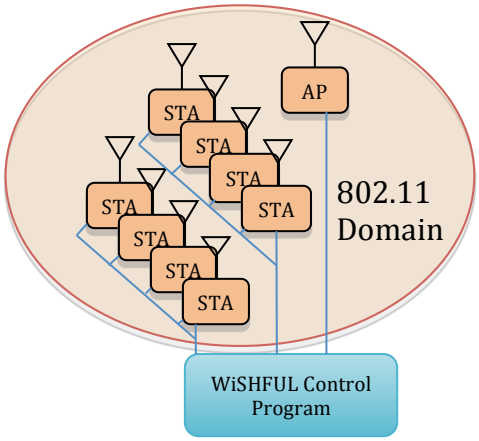

Figure 16. Network topology used in the showcase

As widely documented in literature, different optimization functions and **load metrics** can be considered according to the desired performance metric. In particular, we used a **tuning function**, called Moderated EDCA backoff (MEDCA), whose goal is the minimization of the delay jitters on the channel access times. It is well known that these jitters depend on the exponential backoff mechanism, which introduces short-term throughput unfairness among the stations, and significant variabilities on the time between two consecutive channel accesses performed by the same station. To avoid these phenomena, it is desirable to employ a fixed contention window. The moderated EDCA scheme, currently proposed as a standard amendment, is able to automatically find a fixed contention window equal to the average contention window value experienced under exponential backoff. Since the throughput performance of each station depends on the channel access probability, which in turns is only function of the average contention window, the moderated EDCA scheme is able to minimize the delay jitters while guaranteeing compatibility with legacy EDCA stations (i.e. the same average throughput).

When the number of stations overcomes a given threshold, the global control program disables the local control program (responsible of the contention window tuning) and coordinates the on-the-fly **protocol switch** from moderated EDCA to TDMA in all the nodes. The TDMA parameters, i.e. the number of slots in the periodic frame and the slot allocated to each station, are evaluated on the basis of the number of active flows, which is retrieved by means of global statistics available in the controller.

Summarizing, after the set-up of the wireless network and traffic dynamics performed by the experiment controller, the showcase works by executing the following steps:

1) Activation of a Global Control Program;
2) Sending of the local control program from the global MCE to the local MCEs;
3) Collecting of channel-level local statistics on the number of backoff freezes performed by the local MCEs (on nodes);
4) Collection of global statistics on the number of active flows;
5) Periodic tuning of the contention windows of CSMA radio programs, executed independently by the MCEs running on each network node ***(phase 1)***;
6) Stopping of the local control programs and switching to TDMA radio programs, when the number of actives flows overcomes a given threshold, under the orchestration of the global MCE;
7) Configuration of the TDMA parameters (i.e. frame length and slot allocations) on each node ***(phase 2)***.

For running this experiment, we use two different python scripts: one script is responsible of controlling the experiment, by activating the traffic flows on the wireless nodes dynamically (start_sc3_experiment_controller.py), while the second script is the most relevant one and implements the Global Control Program on top of the WiSHFUL global MCE (start_sc3.py). The two control aspects (*experiment control* for simulating some network dynamics, *wireless network control* for optimizing the system performance) are logically completely independent. This is the reason why they have been implemented in different scripts; moreover, the experiment control can, in principle, be performed in several different ways, including manual activation/deactivation of the traffic flows on the wireless nodes or other frameworks for experiment control, such as OMF/OML.

The source code that implements the load and interference aware MAC adaptation is available in Github repository of the Wishful project, in the directory examples/showcase3/start_sc.py. The same directory contains the python script to start the experiment controller (start_sc3_experiment_controller.py).

2.3.1 Local and Global Control Logic

Figure 17 shows the overall software architecture of control modules involved in this showcase. The WiSHFUL control framework provides some basic functionalities for developing different control programs, such as the possibility to discover the nodes and their capabilities, to provide a global view of the network topology, to send a local decision logic to be used by the local MCEs, and to gather different flow-level and node-level statistics by exploiting the UPI interface available in each network node.

The control program specific of this showcase includes: i) the definition of a function for tuning the contention window of a CSMA radio program, ii) the definition of a decision rule for enforcing the switching from TDMA to CSMA radio programs, and iii) the definition of a function for tuning the frame size and the slot allocations of the TDMA radio programs.

Note that the function for tuning the contention window of the CSMA radio programs is defined in the global control program (as indicated by the dashed lines in the figure), sent to the local MCEs available in each network node, and executed locally. Conversely, the function for deciding about the switching from CSMA to TDMA and for tuning the TDMA parameters is implemented in the global control program and executed by the global MCE (although its execution may also involve the call of local UPI functions). The figure shows the actions of the global MCE and the flows of global control messages in the green arrows, the actions of the local MCE in the red arrows and the data flows in the blue arrows.

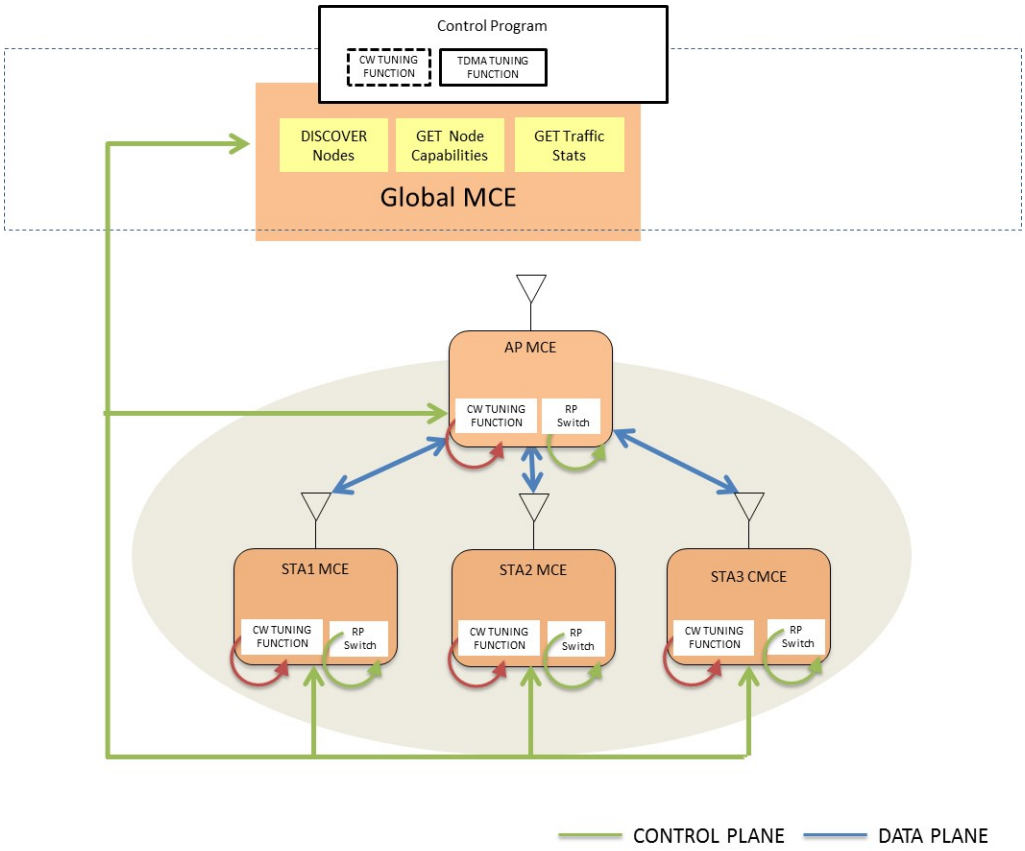

Figure 17. Relations between global and local controllers in this showcase.

a. Presentation of UPIs used

On the basis of the previous description, it is possible to easily read the following code of the control program, which is responsible of injecting the local control logic. The local logic is implemented in the *customLocalCtrlFunction(myargs)* function, which performs the tuning of the contention window, while *mytestbed* is the set of discovered nodes, and *global_mgr* is the WiSHFUL global MCE.

```
%activates the local controllers on each node
run_local_controller(mytestbed)

CtrlFuncImpl = customLocalCtrlFunction
CtrlFuncargs = {'interface' : 'wlan0'}
now = get_now_full_second()
[…]
mytestbed.global_mgr.runAt(nodes, CtrlFuncImpl, CtrlFuncargs,
unix_time_as_tuple(exec_time), callback)
```

The global logic about the decision to switch to TDMA and about the protocol configurations is implemented as follows, where *get_traffic()* is the basic utility for getting the statistics on the number of active flows and *number_nodes_threshold* is a program variable used by the experimenter for a threshold check:

```
traffic_number = get_traffic()
while traffic_number <= number_nodes_threshold:
        traffic_number = get_traffic()
        time.sleep(1)

run_local_controller(mytestbed, disable = 1)

node_index = 0
superframe_size_len = 700 * len(mytestbed.wifinodes)

for node in mytestbed.nodes:
       active_TDMA_radio_program(node,            log,mytestbed.global_mgr,
       nodes_NIC_info[node_index])
       tdma_params={'TDMA_SUPER_FRAME_SIZE'      :      superframe_size_len,
      'TDMA_NUMBER_OF_SYNC_SLOT'         :         len(mytestbed.wifinodes),
      'TDMA_ALLOCATED_SLOT': node_index}
        set_TDMA_parameters(node, log,mytestbed.global_mgr, tdma_params)
       node_index += 1
```

b. Results

Through the use of the UPIs detailed above, the control logic employed within this showcase achieves portability across platforms that support the WiSHFUL Framework, without relying on the details of the underlying hardware.

2.3.2 Tuning of the contention window

As already discussed in the general description of the control framework, the tuning of the contention window is performed independently and locally by each node, by executing the custom function *customLocalCtrlFunction(myargs).* The showcase is focused on the minimization of the delay jitters, while guaranteeing backward compatibility with legacy EDCA stations.

a. Presentation of UPIs used

The optimization strategy utilized herein is implemented in the following code, where *count_freezing* is the last measurement returned by the UPI_R monitor function, and *last_count_freezing* is a program variable used for detecting the periodic overflow of the counter.

```
#Measurement filtering
delta_freezing = count_freezing - last_count_freezing
if delta_freezing < 0 :
            delta_freezing = 65535 - last_count_freezing + count_freezing
last_count_freezing = count_freezing
ipt = ipt + a * (delta_freezing - ipt)
```

```
# #targetcw according to the heuristic formula
# determine the target CW for this IPT
targetcw = -1.3539 * ipt ** 2 + 7.6655 * ipt + 15.4545;
# calculate new smoothed CW
cw_f = cw_f + b * (targetcw - cw_f);
cw = round(cw_f);
```

After filtering the measurement, the new contention window is computed by using a heuristic formula, which relates the average contention window of legacy EDCA stations to the number of backoff freezes. It has been demonstrated that the convergence of this scheme to the average EDCA window is guaranteed for any number of stations. The computed value is filtered to avoid sudden modifications on the station access rates. Finally, the new contention window value is enforced by using the UPI_R function responsible of configuring lower layer parameters.

```
#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)
```

b. Results

To evaluate the capabilities of our contention window tuning, we activated 6 wireless nodes contending under greedy traffic sources towards a common Access Point. We first considered a legacy CSMA protocol with exponential backoff. Figure 18 shows the throughput performance achieved by each station and some regular samples of the contention window employed by the stations. Although the CSMA protocol in principle should provide an equal share of the network throughput to each station, we can observe some short-term and long-term throughput variability due to the exponential backoff mechanism (short-term) and to the location-dependent interference conditions suffered by each station (long-term).

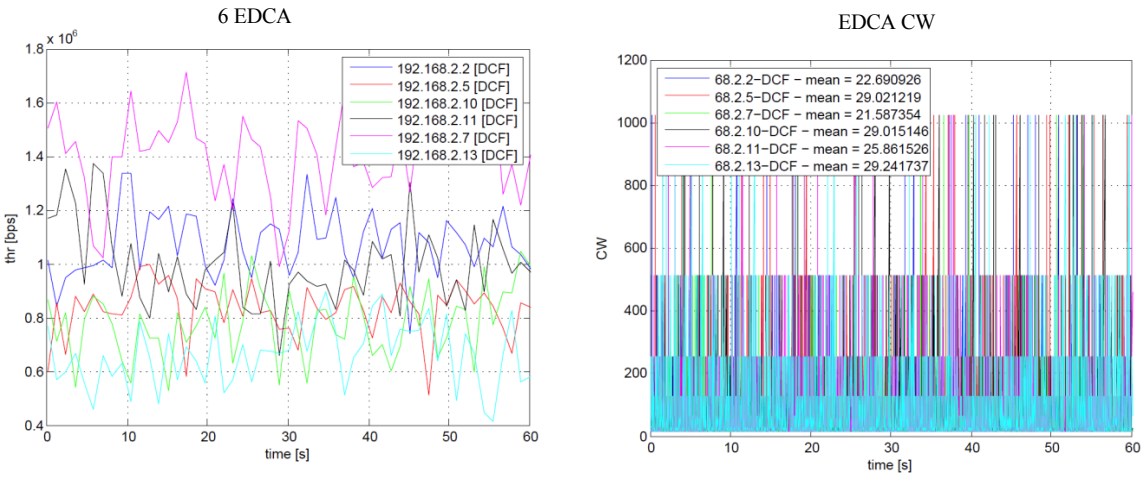

Figure 18. Throughput performance and Contention Window samples of 6 wireless nodes executing CSMA with exponential backoff.

For three of the above nodes, we then loaded the local control logic implementing the moderated EDCA backoff scheme. Figure 19 shows that the three stations achieve an average throughput comparable to the one experienced when they were using exponential backoff, but with smaller fluctuations. This is confirmed by the samples of the contention window values, that exhibit very small variations (from 20 to about 25).

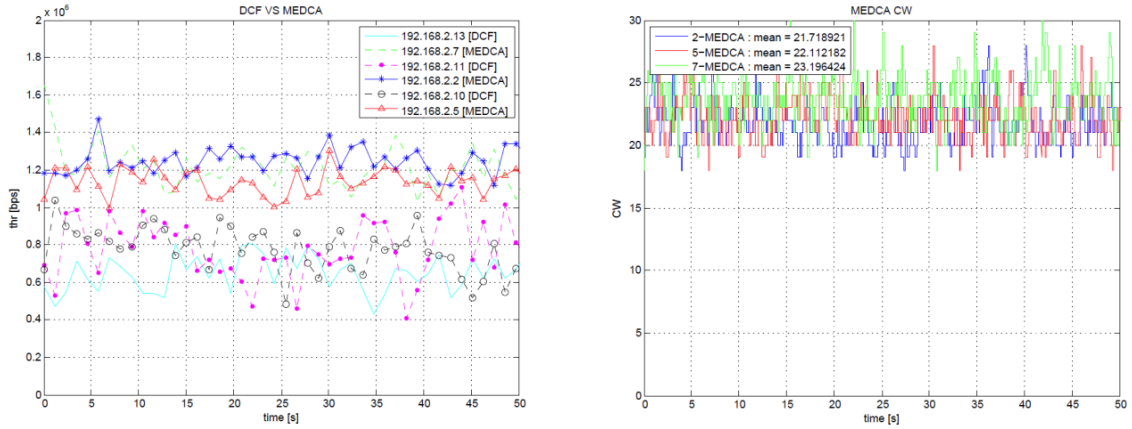

Figure 19. Throughput performance and Contention Window samples of 3 stations employing moderated backoff in contention with 3 stations employing exponential backoff.

c. Next Steps

Our current approach to contention window tuning is based on the minimization of delay jitters in a purely local fashion. This approach may be expanded in the future by exploring methods to take advantage of information provided by peer nodes.

2.3.3 Switching from CSMA to TDMA

When the global control logic detects that the number of active traffic flows overcome a given threshold, it disenables the local control functions and triggers a coordinate switch to the TDMA radio program. Presentation of UPIs used

a. Presentation of UPIs used

Before switching, the controller verifies that a TDMA radio program is available on the wireless nodes, by reading the node capabilities associated to the radio NIC:

```
for ii in range(len(current_radio_info.radio_program_list)):
        if     current_radio_info.radio_program_list[ii].radio_prg_name     ==
"TDMA" :
        #force the radio memory slot position in which the radio program is
stored, the WMP platform on broadcom card has only two memory slots to
store radio programs, addressed by position 1 and position 2
      position = '2'
[…]
```

The TDMA radio program is activated by calling the UPI_R function **setActive**, which also injects the radio program into the micro-instruction memory so the execution engine can run it.

```
UPIfunc = UPI_RN.setActive
UPIargs = {'position' : position, 'radio_program_name' : 'TDMA', 'path' : radio_program_pointer_TDMA, 'interface' : 'wlan0' }
rvalue = global_mgr.runAt(node, UPIfunc, UPIargs, exec_time)
```

Finally, the global controller configures the TDMA parameters of each station, by specifying a different temporal slot for each one by means of the relevant UPI_R function (responsible of configuring radio program parameters), which is called by the *set_TDMA_parameters* utility function.

```
tdma_params={'TDMA_SUPER_FRAME_SIZE' : superframe_size_len,
'TDMA_NUMBER_OF_SYNC_SLOT' : len(mytestbed.wifinodes),
'TDMA_ALLOCATED_SLOT': node_index}
set_TDMA_parameters(node, log,mytestbed.global_mgr, tdma_params)
```

b. Results

As an high-level **functional validation**, we monitored the channel-level measurements collected by three wireless nodes transmitting saturated traffic towards a common Access Point at regular reporting periods (samples) for phase 1 (Figure 20) and at regular time for phase 2 (Figure 21) of the experiment. For the Access Point and for the nodes executing the TDMA radio program, the counter of the backoff freezes is constant, because the backoff process is not active.

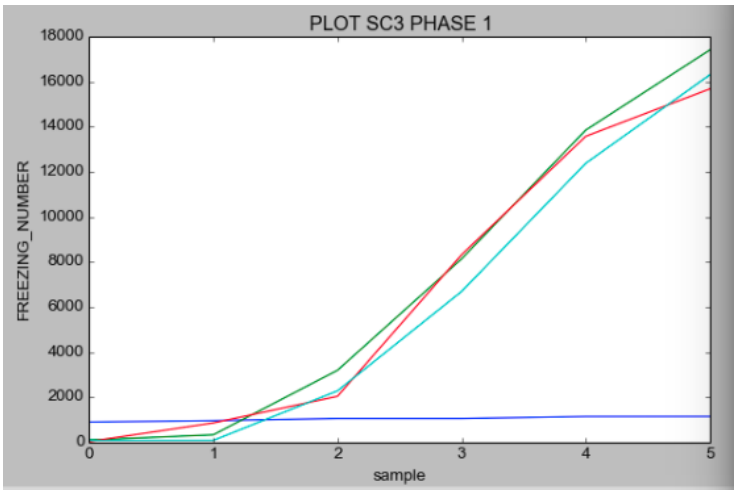

Figure 20. Number of freezes occurred during backoff procedures in phase 1 at regular report period

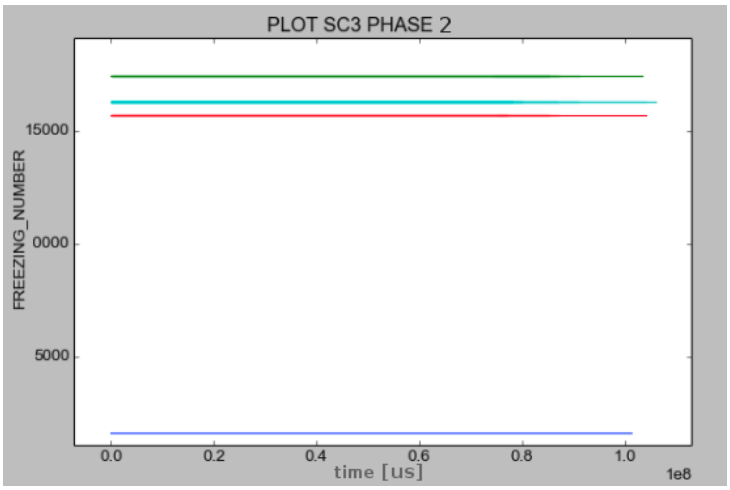

Figure 21. Number of freezes occurred during backoff procedures in phase 2 at microseconds time

To illustrate the **technology independency** of this showcase, it was also executed on the RM-090 sensor nodes. The Contiki operating system is combined with the TAISC VM to enable on-the-fly switching from CSMA to TDMA. In this case, both radio programs were pre-installed. Figure 22 shows the measured packet loss and throughput. After 90 seconds the radio program is switched from CSMA to TDMA. The results were obtained by letting 32 sensor nodes send iPerf traffic to a single sensor node acting as a sink, forwarding the iPerf streams to an iPerf server on the linux host-PC. The output of the iPerf server was used to generate the graphs. All nodes were in the same collision domain.

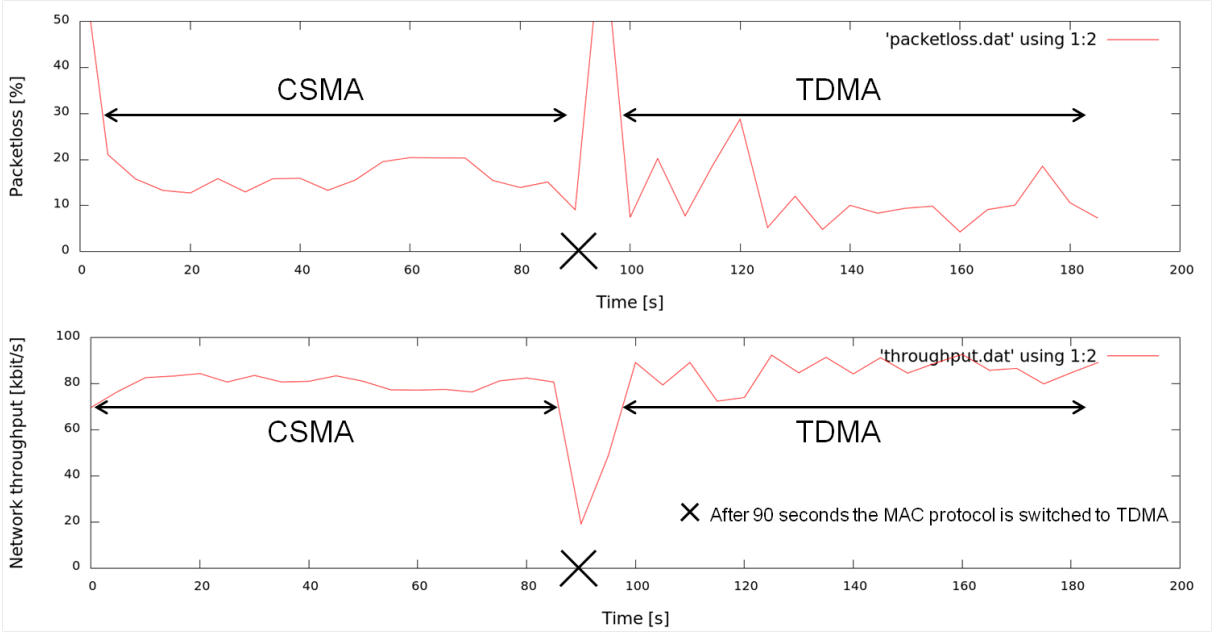

Figure 22. Switching from CSMA to TDMA on the RM-090 sensor node using TAISC inside Contiki OS. The upper graph shows the overall percentage of packet loss, the lower graph illustrates the overall throughput. The high spikes in both graphs are caused by the switch.

The overall packet loss and throughput is only slightly higher for TDMA w.r.t. to CSMA. Moreover, the TDMA radio program requires +- 30 seconds to stabilize. This indicates that the TDMA radio program as well as that the switching procedure are sub-optimal and can still improve. Note also that while the overall performance of CSMA is stable, there is a huge difference between nodes due to the back-off algorithm.

c. Next Steps

Currently protocol switching occurs according to a static threshold on the number of active traffic flows. The approach presented here may be extended by considering a multivariate utility, potentially based on delay, number of retransmissions, and number of freezes, to better model the impact of congestion on user applications. A threshold could then be learned through a genetic algorithm scheme which searches through various combinations of utility values for use as a threshold.

2.4 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.

2.4.1 Portable testbed setup

The architecture of the portable testbed is presented in Figure 23. 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. It also allows interaction with the nodes during the experiment. Section 2.4.2 describes the Backbone network in more detail.

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, 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 behaviour 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.

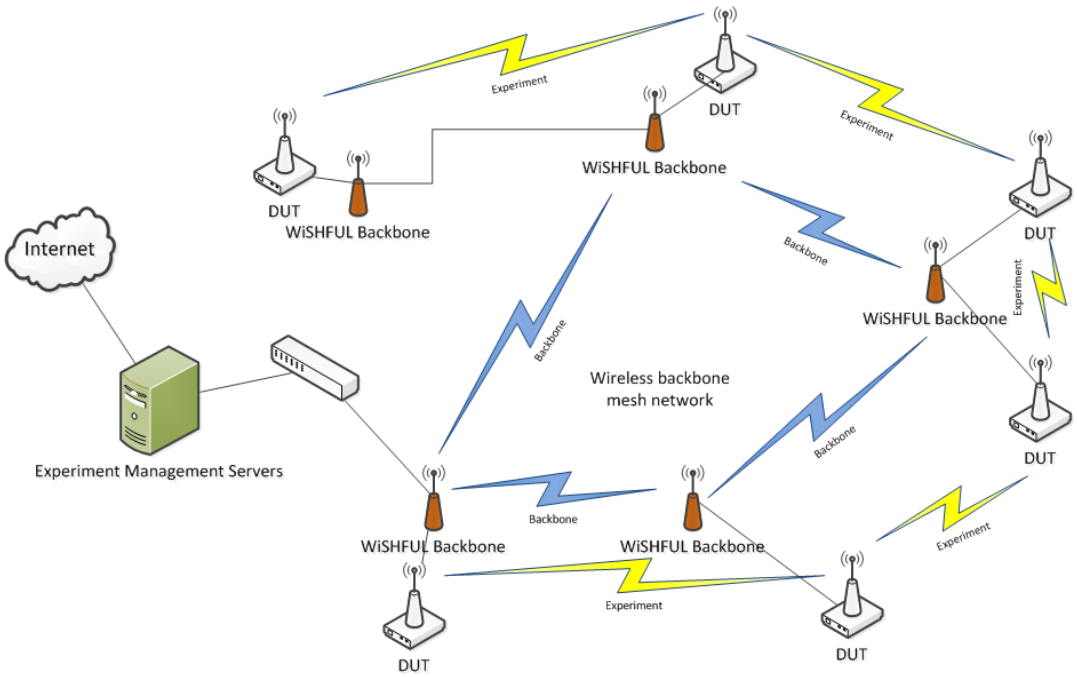

Figure 23: Portable testbed overview

a. 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 aluminium 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 mainly of an Intel Core i5 4250U processor, 4GB of ram, a gigabit Ethernet port, several USB ports, a 320GB harddisk 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.

b. Experimentation tools

The portable testbed offers almost identical functionality to the experimenters as if they would run their experiment on one of the fixed testbeds.

The experimenter is able to use one user account to access all WiSHFUL testbeds, including the portable testbed. The same user account can be used to access all Fed4FIRE testbeds. This user account is used to login with one tool (jFed) to design and setup an experiment. Again this tool can be used to access multiple other testbeds inside the Fed4FIRE federation.

The wireless backbone enables the user to interact with the nodes during the experiment. This interaction can be done by using SSH or making the DUT part of an OMF6 controlled experiment. The BN (Backbone Network) also enables measurements being collected using the OML framework. The (aggregated) live data can be sent over the wireless backbone towards an OML server, or can be stored locally (cached) and dumped to a database server after the experiment.

Deployment of the portable testbed is as easy as plug-and-play, in order to lower the boundary for experimenters. The duration of the deployment of the portable testbed can vary from several hours to several weeks or even months. Depending on the duration of the deployment and the accessibility of the environment in which the testbed is deployed, extra fail-safe mechanisms may be activated to allow for better remote management of the portable testbed. Several ways to power the DUTs are supported: AC power, Power-Over-Ethernet or 19V battery packs. Using Power-Over-Ethernet, remote rebooting of the DUT nodes is supported. On the contrary, when using AC power or 19V battery packs, this functionality (remote rebooting) is not yet supported.

A more detailed description of experimentation tools can be found in D5.2.

c. Next Steps

An integrated battery solution providing flexibility over the positioning of the nodes will be designed. An extra circuit board needs to be designed so that the BN node can instruct the DUT node to reboot while in the field. A custom enclosure will be designed for this integrated solution, containing the following components:

- DUT (with possible USB extensions)
- BN node
- Circuit board to enable the BN node to reboot the DUT node
- 19V Battery pack to power both BN and DUT node, or one battery pack for each node.
- Docking extensions with PLC to provide both Ethernet connection and recharging functionality while the nodes are plugged into the flight case.

The flight case containing the DUT nodes will also be modified to support the integrated battery powered nodes. More details can be found in D6.2.

The portable testbed will be used to demonstrate the showcases described above (2.1, 2.2, 2.3). Some of the showcases that require tight time synchronization between all nodes are not yet supported. However, a simplified version of each of the showcases can be enabled by the current implementation of the portable testbed.

2.4.2 WiSHFUL mesh backbone

The Backbone is a mesh routing network, which is used to connect TMS, EMS, SUT-NC and DUT nodes. OLSR is used as routing protocol. To make it transparent and keep all mentioned nodes unaware of being inter-connected through portable testbed, a dynamic logical L2 network over the backbone network are created.

Thanks to provided default OS image with default configuration, the Portable Testbed is operational "out-of-the-box". An experimenter has to connect pair-wise DUTs nodes with their peer BN node, boot them all, and the Backbone network establishes full connectivity.

The operation of the BN network is harmonized by a dedicated BN Controller (BNC). We implemented BNC using UPI functions defined in WiSHFUL project. An Intel NUC mini-PC with Intel i5 processor is used to run Backbone Controller application.

For BN nodes we have selected a hardware platform that is very robust and can cope with harsh, outdoor conditions. Currently we rely on the Gateworks Laguna platform - model GW2388-4.

More details about Backbone network can be found in D6.1 and D6.2.

a. Presentation of UPIs used

- ***setChannel()***

  This function is used to instruct BN nodes to change channel, that they operate on. Channel is selected during ***Channel Policy Agreement (CPA)*** procedure, which is performed before experiment start. More details of CPA procedure can be found in D6.2.

- ***installEgressSheduler()***
  This function is used in order to setup Priority Queuing Discipline – NET layer prioritization – in each BN node. First, we create description of configuration of QDisc, using developed python

package – *python-tc*. Then, function *installEgressSheduler()* is used to install proper QDisc according to passed configuration.

- ***setFlowTransmissionQueue()***
  To provide proritization in MAC layer, we exploit the fact that the EDCA classifies packet to proper queue based on TOS value it its header. The UPI function *setFlowTransmissionQueue()* is used to configure iptables rule to set proper TOS value in packet of particular flow, which is defined with 5-tuple.

b. Results

In this document we present only performance comparison between a regular mesh backbone and a backbone controlled by WiSHFUL UPI's. A detailed description of the implemented functionalities supported by Portable Testbed is provided in D6.2.

In order to collect performance measurements, we simulated traffic in Backbone network using three flows:

- F1 – generated using ping application, one ping-request is sent every 1 second; this flow mimics Backbone Mesh Management Traffic (e.g. OLSR Hello Packets), which should be served with highest priority; important performance parameters: packet loss, Round-Trip-Time (RTT)
- F2 – generated using iperf application with parameters: protocol – UDP, bandwidth – 1 Mbps; this flow mimics WiSHFUL and Experiment Control Traffic, which should be served with high priority; important performance parameters: packet loss, received throughput, jitter.
- F3 – generated using iperf application with parameters: protocol – TCP; this flow mimics Measurement Collection Data Traffic, which should be served with best-effort manner; important performance parameters: throughput.

During single test all mentioned flows were started at the same time, thus they compete for the access to spectrum. We performed experiments within topology presented in Figure 23, it is simple wireless mesh network with 3 nodes. All flows are started at Source BN node and directed to Destination BN node. The network topology was configured in such way, that transmission between Source and Destination BN nodes involved two wireless hops.

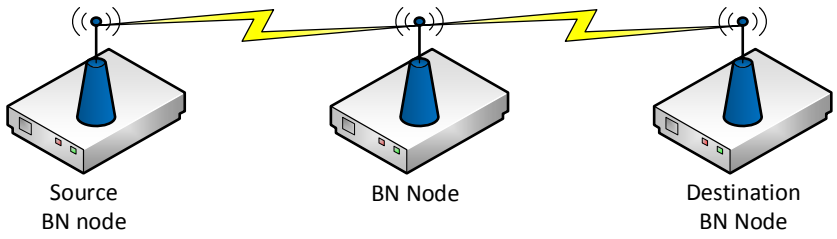

Figure 23 Topology used for performance comparison

Duration of single test was 30 seconds and we repeated each test 25 times to get credible results. Two cases were considered: *i)* with default configuration – i.e. without prioritization; and *ii)* with prioritization in NET and MAC layer. In latter case, we used BNC, that configures is able to configure prioritization in both layers using WiSHFUL UPI's - see *Presentation of UPIs used.*

In Table 1, Table 2 and

Table 3 we present performance comparison based of obtained results. As can be seen proper prioritization in NET and MAC layers improves all performance parameters of each flow.

Table 1: Performance comparison for Flow 1

Flow F1	Without prioritization		With prioritization in NET and MAC layers	
Parameter	**Average**	**Std.**	**Average**	**Std.**
Packet loss [%]	24.82	10.27	5.45	4.62
RTT [ms]	1034.61	923.57	6.98	10.98

Table 2: Performance comparison for Flow 2

Flow F2	Without prioritization		With prioritization in NET and MAC layers	
Parameter	**Average**	**Std.**	**Average**	**Std.**
Packet loss [%]	23.43	13.35	7.42	1.15
Throughput [Mbps]	0.67	0.33	0.92	0.02
Jitter [ms]	165.65	152.83	5.77	4.86

Table 3: Performance comparison for Flow 3

Flow F3	Without prioritization		With prioritization in NET and MAC layers	
Parameter	**Average**	**Std.**	**Average**	**Std.**
Throughput [Mbps]	2.75	1.23	2.98	0.39

c. Next Steps

During year 2, we will work on following improvements and extensions of Portable Testbed:

- Automatic DUT node to BN node mapping discovery
- DUT Power-supply control
- Channel Policy Agreement improvement
- Channel Access optimization

A detailed description of the extensions and improvements is provided in D6.2.

3 Definition of showcases to be implemented on Year 2

Example showcases envisioned for implementation with the second year of the project are described in an early form here.

3.1 Intelligent Download with WIFI tethering

3.1.1 Overview

Recently, with rapid growth of number of smartphones and mobile devices equipped with various wireless technology interfaces, tethering popularity gains more and more popularity. It is a very convenient, ad-hoc and low-cost wireless Internet access technology. In most cases Wi-Fi tethering is used, which allows sharing the Internet connection provided by 3G/4G technology with other devices (eg. laptops) using Wi-Fi network.

However, in most cases, there is a limit on amount of data that cellular subscriber can download every month. After exhaustion of available data transfer, user connection can slow down significantly or in worse case, one can be charged for any extra data he/she downloaded. As long as user is aware of all his/her network transmissions, there is no problem. Unfortunately, it can happen that operating system and applications will perform upgrades and download a huge amount of data that user may not even notice until he/she gets a bill from telecom company. Our idea is to recognize and prevent any "unnecessary" traffic flows. By "unnecessary", we understand flows that require downloading a huge amount of data and there is no problem to defer it to a later point in time. Operating system updates (e.g. Windows/Linux) are perfect example here.

3.1.2 Goals

- Intelligently control use of Wi-Fi network;
- Filter download requests based on their context, including connectivity situation and content priority.

3.1.3 Breakthroughs

Our basic idea is to filter out "unnecessary" traffic flows when being connected to a tethering AP (Figure 25). Therefore, we can make use of IEEE 802.11u that defines Generic Advertisement Service. GAS is a mechanism that delivers information to the STA from advertisement services. It allows stations to obtain information about network services. Standard defines few advertisement protocols that can be used with GAS: Access Network Query Protocol (ANQP), Media Independent Handover (MIH), Emergency Alert System (EAS) as well as proprietary vendor specific protocols (which will be most useful for us). What is important is that GAS mechanism allows the STA to know in advance the AP capabilities, even before associating with it.

3.1.4 Methodology

With the help of GAS we are able to block the "unnecessary" flows already on Wi-Fi end-user terminal (e.g. laptop). After reception of the specific IE from the tethering AP, the terminal should translate it into local firewall filtering rules and apply them. One possible way to achieve that is use of netlink interface and netfilter framework provided by Linux (used by iptables). Note, here both the tethering AP as well as the Wi-Fi end-user terminal need to be WISHFUL-compliant. In a second option which does not require the end-user terminal to be Wishful-compliant the blocking of

"unnecessary" flows is performed in the tethering AP which is fully transparent to the end-user terminal.

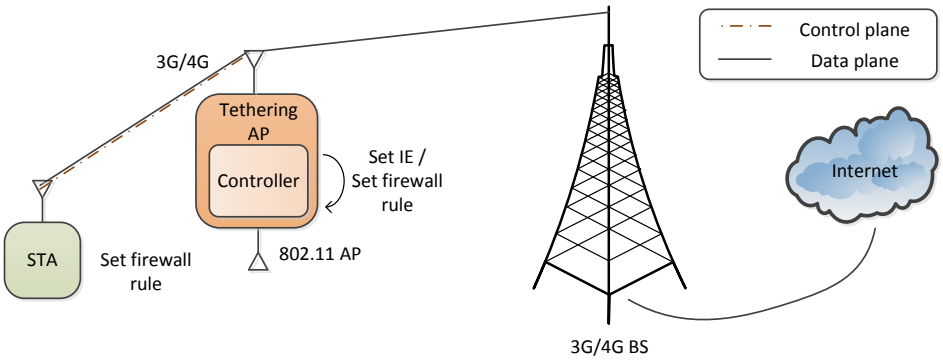

Figure 25 Controlling Wi-Fi tethering operation.

3.1.5 Use of WiSHFUL Functionality

For the first option, the WiSHFUL network interface will provide a way to program the Information Elements (IE) send in the beacon frames of the tethering AP. Moreover, on the end-user terminal WiSHFUL interfaces will offer functionality to read the received IEs, as well as to program the firewall, i.e. reject all outgoing traffic to a specific remote host.

For the second option, WiSHFUL will allow the possibility to block "unnecessary" flows in the tethering AP.

3.2 WIFI offloading

3.2.1 Overview

Although the capacity of cellular networks constantly increases thanks to technological enhancements, the throughput they provide can turn out to be insufficient, because traffic demand increases even faster. On the other hand, most of mobile devices are not only equipped with LTE interface, but also Wi-Fi chip. It is therefore promising to offload traffic from mobile networks to Wi-Fi and use them as an extension to the cellular network and telecom operators are becoming more and more interested in it. The main reasons for this approach are the high data rates provided by Wi-Fi networks.

3.2.2 Goals

- Enable offloading of cellular traffic through Wi-Fi networks;
- Empower network performance aware techniques for network selection.

3.2.3 Breakthroughs

Currently, mobile devices have simple and limited way to decide when to offload traffic to the WIFI (Figure 26). It works very simple, i.e. when mobile terminal discover and connect to WIFI network, it steer all its traffic to the WIFI. The main drawback of this solution is lack of QoS considerations that can lead to situation when mobile will switch from high data rate cellular connection to low data rate WIFI connection.

In current networks, operators do not have influence for offloading decisions of mobile stations, but the idea is so appealing, that some activity by several standardization forums was taken. They propose operator-controlled WIFI, which are deployed and managed by an operator and/or its partner. In 3GPP Release 12, some WLAN/3GPP inter-working aspects were standardized. They aim is to provide network operator control mechanisms to steer traffic offloading in downlink and uplink direction. With these solutions network can provide mobile station with parameters such as receive power level threshold. When received power is higher than this threshold mobile can offload its traffic to WLAN.

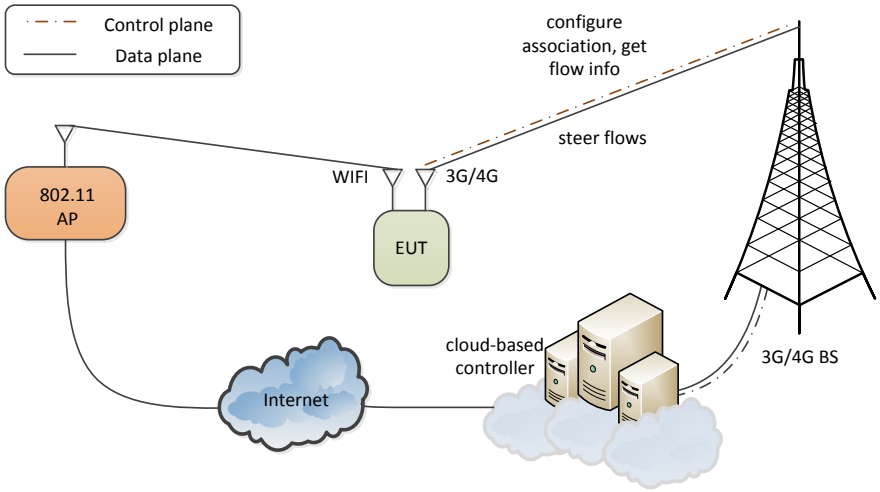

Figure 26 Controlling WIFI offloading.

3.2.4 Use of WiSHFUL Functionality

To support Wi-Fi offloading, several WiSHFUL functions are needed. First, the network operator will use WiSHFUL interfaces to define parameters (e.g. receive power thresholds) that will be send to mobile stations which are used to make the decision to connect to a Wi-Fi network. Second, the provider needs to decide which flows should be offloaded from cellular to Wi-Fi network. For example, one can decide that all VoIP traffic stays in cellular network, because it provides more robust and reliable connection, and all flexible flows using TCP protocol (eg. file transferring, web browsing) are offloaded to WLAN. This decision will be enacted through the WiSHFUL interfaces.

3.3 Load and topology aware networking

3.3.1 Overview

In dynamic wireless networks the application requirements vary over-time. Moreover, networks can grow or shrink as a result of node mobility. Network protocols designed for such networks (e.g. 6LowPan and RPL for sensor networks and IEEE802.11e and OLSR for Wi-Fi networks) have built-in support for allowing such dynamic behaviour. The standards, defining these protocols, allow fine-tuning the protocol operation via configuration parameters, enabling to make trade-offs between different performance metrics. The implementations of these protocols, however, do not provide a unified interface for this purpose.

These showcases will demonstrate how the WiSHFUL UPIs can be used to (i) dynamically monitor the network performance and topology; (ii) change network protocol configuration; or (iii) switch routing modules. Two sub showcases will be implemented:

Lowering frame loss in highly mobile networks: illustrates how *dynamic reconfiguration of frame aggregation and PHY rate adaptation parameters* can lower the frame loss. Because node mobility intensifies the time varying nature of wireless channel, the network stack requires to be re-configure d according to the degree of the mobility. For this purpose the *node mobility is monitored* and based on the level of mobility, *the aggregation level and/or modulation and coding scheme (MCS) index are reduced or increased in real-time*. By lowering the aggregation level (e.g. number of frames) or MCS index in case of higher node mobility, the chances on bit errors, due to mobility, can be reduced during frame reception.

***Selecting the optimal link estimation algorithm based on the network topology*:** demonstrates that a global control program, controlling a sensor network, can *increase the overall network performance* by *dynamically activating the optimal link estimation algorithm.* For this purpose the network topology is monitored and, the *optimal link estimator is activated in each scenario*. In sparse networks, simple link estimators (e.g. objective function 0, ETC) are preferred; in dense network more complex link estimators (e.g. 4BIT, fuzzy LQE) can be applied.

3.3.2 Goals

- Dynamically control the frame aggregation level in Wi-Fi networks by estimating the level of node mobility.
- On-the-fly selecting of the most optimal link estimator in sensor networks by monitoring the node topology.

3.3.3 Breakthroughs

Currently, in most deployments, the network stack configuration is statically defined at deploy time and requires a manual intervention to change. This showcase demonstrates how the WiSFHUL UPIs enable to ***dynamically monitor the various metrics that define the network state*** (e.g. quality of service, energy, and topology) in a unified manner. Moreover, the UPIs also allow reacting on the observed metrics, and ***dynamically reconfigure the network protocols in order to improve the network state***.

3.3.4 Use of WiSHFUL functionality

The first sub-showcase is a perfect example of how local decisions made in a local control program can improve the network performance. For this both UPI_R and UPI_N are required.

- UPI_R enables to estimate the level of node mobility by observing the link layer statistics (packet loss, BER) and combining this with link layer measurements (signal quality, medium activity).
- UPI_N is used to reconfigure the frame aggregation parameters defined by IEEE-802.11e (e.g. A-MSDU, A-MPDU).
- UPI_N is used to adapt PHY rate (MCS index) according to the network state

The second sub-showcase requires global control program to monitor the network topology and change the link estimation algorithm. UPI_G is used to execute UPI_R/N functions on a number of nodes, or obtain particular information from nodes with specific roles in the network.

- UPI_R enables detecting asymmetric links in the network by collecting link statistics between certain pairs of nodes. Moreover, also frame injection is supported for this purpose.

- UPI_N allows maintaining the current network topology by monitoring the routing tables of a certain group of nodes. Moreover, volatile links can be detected by analysing the variance in topology due to routing decisions.
- UPI_N allows to (de)-activate link estimation algorithms such as ETX, 4BIT and fuzzy LQE in the RPL routing protocol.

3.4 MAC Adaptation in multi-hop network topologies

3.4.1 Overview

Wi-Fi network performance can dramatically degrade in multi-hop connected topologies and in high-density node scenarios. The main reasons for the degradation include the starvation and unfairness phenomena of CSMA-based protocols due to a mismatch in the local views of the wireless medium among the nodes, and due to the high level of contention when the network is congested. To mitigate these problems, different approaches have been proposed, such as the adoption of rate limiters deployed on each network node, the utilization of multi-hop reservations and the exploitation of admission control mechanisms. The main goal of these approaches is to avoid consuming the whole channel capacity in different network links, thus reducing the collision probability and leaving resources for network management operations.

In this scenario, we demonstrate the prototyping of a solution for mitigating the performance impairments of CSMA/CA protocols in multi-hop topologies, based on the dynamic adaptation of the contention process experienced by nodes in the wireless network. A distributed protocol is used to negotiate the channel airtime for a node as a function of the traffic requirements of its neighbourhood, taking into account bandwidth reserved for the control operations. A mechanism is provided for a node to tune its contention window as a function of this airtime.

3.4.2 Breakthroughs

Channel allocations in wireless networks can be viewed as a resource allocation problem where transmitters correspond to demands and receivers to resources. The allocations can be performed in a disturbed manner, by considering that the capacity unused by the receiver nodes can be redistributed to the transmitter nodes as in an auction/bidder problem. When the distributed allocations are performed, each node can achieve the desired allocation by opportunistically tuning its contention window. The scheme involves the implementation of two different components: i) the negotiation protocol, for evaluating the channel resources that can be allocated to each node; ii) the tuning of the contention window for guaranteeing heterogeneous channel allocations as a function of the channel views locally observed by each node. Both the components can be prototyped by exploiting the WiSHFUL architecture and interfaces. Specifically, the negotiation protocol can be implemented as a local control program running on each node, while the contention window tuning can exploit the UPI_R interface (**Figure 24**).

Our goal is demonstrating the possibility to support advanced signalling schemes, which can inter-operate with the application level (for getting the requirements) and low level MAC operations (for tuning the contention window during run-time).

3.4.3 Use of WiSHFUL Functionality and methodology

For this showcase we use different type of Wi-Fi nodes available on WiSHFUL, both Broadcom and Atheros Wi-Fi cards. We use the experiment controller to create a wireless mesh network with different domain and we implement distributed allocation algorithm on local controller. The local

control program uses the UPI interface to get the number of transmission attempts and the air time channel. After, the local control calculates the value of contention window and uses the UPI interface to set the new value.

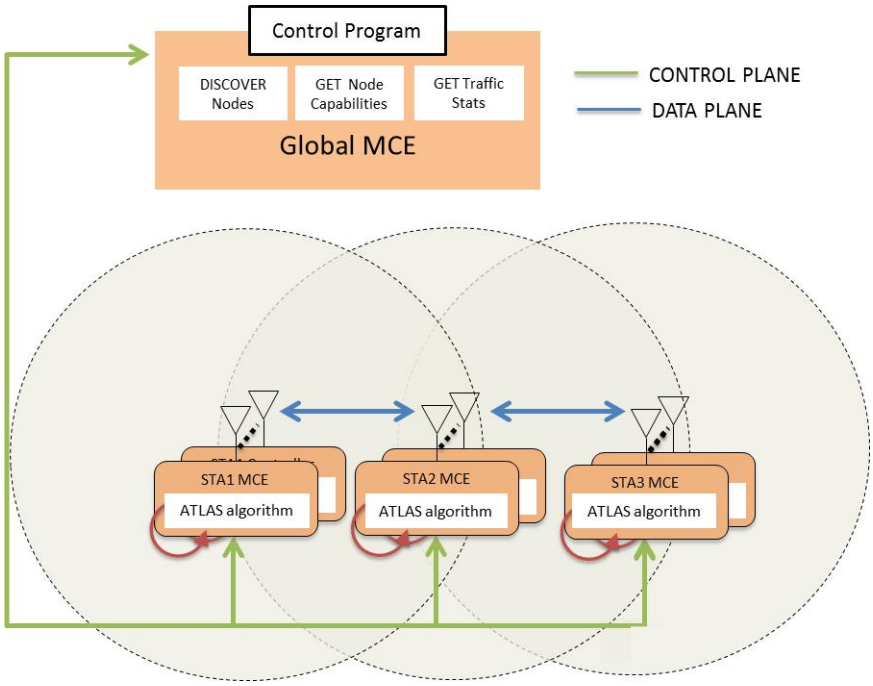

Figure 24 - 1.1.1 MAC Adaptation in multi-hop topologies

4 Conclusion

This document has provided an initial entry in the catalogue of the utilities provided by the WiSHFUL project, by summarizing the outcome of the first set of project showcases. Showcase description contained within, highlights the impact of WiSHFUL on currently relevant problems. It was addressed by discussing how WiSHFUL functionalities support the development of solutions within the space of wireless communications. Moreover, the material of this document discusses extensions or continuations of the work completed so far to further exploit the contributions of the project. This document has focused on the utility and impact of WiSHFUL in the contained description of project showcases, leaving technical details regarding the implementation of each showcase to the appropriate technical deliverable(s) (D3.2, D4.2 and D6.2).

Finally this document has introduced new showcases for the second year of the WiSHFUL project.

5 References

[1] S. M. Kim and T. He, “FreeBee: Cross-technology Communication,” ACM MobiCom, pp. 317–330, 2015.