HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Florian Menne

Investigating Selective Wake-up Transceivers for IEEE 802.11 WLAN

Masterarbeit im Fach Computer Engineering

30. Juni 2018

Please cite as:
Florian Menne, “Investigating Selective Wake-up Transceivers for IEEE 802.11 WLAN,” Master’s Thesis (Masterarbeit), Heinz Nixdorf Institute, Paderborn University, Germany, June 2018.

Distributed Embedded Systems (CCS Labs)
Heinz Nixdorf Institute, Paderborn University, Germany

Fürstenallee 11 · 33102 Paderborn · Germany

http://www.ccs-labs.org/

Investigating Selective Wake-up Transceivers for IEEE 802.11 WLAN

Masterarbeit im Fach Computer Engineering

vorgelegt von

Florian Menne

geb. am 21. Februar 1990
in Salzkotten

angefertigt in der Fachgruppe

Distributed Embedded Systems
(CCS Labs)

Heinz Nixdorf Institut
Universität Paderborn

Betreuer:	**Johannes Blobel**
	Florian Klingler
Gutachter:	**Falko Dressler**
	Holger Karl
Abgabe der Arbeit:	**30. Juni 2018**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Florian Menne)
Paderborn, 30. Juni 2018

Abstract

In wireless LAN (WLAN), power saving is realized in a duty cycled manner. While connected to an access point (AP), the station (STA) wakes-up for the beacons sent by it, to determine if traffic is buffered at the AP. This comes along with two drawbacks. First, every time the STA wakes-up costs energy. Second, the worst case packet delay from the AP to the STA depends on the beacon interval, as the beacon inherits the traffic indication map (TIM) which is used to inform a STA about outstanding packets. In 2016, the Institute of Electrical and Electronics Engineers (IEEE) has started to address this drawbacks in a task group called IEEE 802.11tgba which is working on integrating wake-up receivers in the IEEE 802.11 standard. In this thesis, I developed a prototype which combines a commercially available WLAN adapter with a wake-up receiver. Afterwards, the system performance was measured and it turned out, that a wake-up receiver can reduce the idle energy consumption by 64 % with the used adapters at a beacon interval of 100 *ms*, which is common for most WLAN setups. Packet delay measurements, done from application level, have shown that the packet delay is lower and more stable in comparison to the built-in power save. A last measurement, in which I forced the WLAN adapter to sleep while traffic is flowing, indicates a robust behavior against bursty traffic. At the end, I can conclude that WLAN can benefit in different situations from the new standard which is developed by the task group IEEE 802.11tgba.

Kurzfassung

In wireless LAN (WLAN) wird der Energiesparmodus vergleichbar mit duty cycling realisiert. Ist eine station (STA) mit einem access point (AP) verbunden und wurde in den Energiesparmodus versetzt, so werden die Beacons welche vom AP gesendet werden genutzt um die STA darüber zu informieren ob Pakete im AP gepuffert sind. Um Beacons zu empfangen muss die STA kurz vor einem Beacon aufwachen. Aus dem Verfahren ergeben sich folgende Nachteile: Zum einen kostet es Energie den Empfänger einzuschalten um das Beacon zu empfangen. Zum anderen ist die Verzögerung bis ein Paket von dem AP an die STA ausgeliefert werden kann durch dieses Verfahren abhängig von dem Beacon-Intervall, da die STA erst ermitteln kann ob Datenpakete vorhanden sind, wenn sie das Beacon empfängt. Derzeit beschäftigt die IEEE eine Arbeitsgruppe unter der Bezeichnung IEEE 802.11tgba welche sich mit der Integration von wake-up Empfängern in den WLAN Standard beschäftigt. In dieser Masterarbeit zeige ich, wie sich ein WLAN Adapter mit einem wake-up Empfänger kombinieren lässt. Anschließend werden die Eigenschaften dieses Prototyps gemessen. Die Messungen haben gezeigt, dass unter Nutzung des wake-up Systems der stand-by Energieverbrauch um 64 % mit den genutzen WLAN Adaptern gesenkt werden kann. Messungen der Paketverzögerung auf Applikationsebene haben gezeigt, dass der entwickelte Prototyp nicht nur eine geringe, sondern auch eine stabilere Latenz im Vergleich zu dem im Standard enthaltenen Energiesparmodi aufweist. In einer letzten Messung konnte gezeigt werden, dass das wake-up system zudem robust gegen stoßweise aufkommenden Datenverkehr, während die STA schlälft, gerüstet ist. Abschließend kann ich folgern, dass WLAN von dem neuen Standard, welcher aktuell in der Arbeitsgruppe IEEE 802.11tgba entwickelt wird, profitiert.

Contents

Chapter 1

Introduction

1.1 Motivation

Wireless LAN (WLAN) (IEEE 802.11 [1]) is a well proven standard for creating a secure and reliable network across different devices including computers, smartphones, smart televisions and many more. The fact that there are only a few practical attacks over years shows the strength of WLAN and the used security standard Wi-Fi Protected Access 2 (WPA2).

In WLAN power saving is implemented in a duty cycling manner while connected to an access point (AP) [1]. The power saving works as follows: If power save is enabled, the station (STA) informs the AP that it will go to sleep. During that time, the AP buffers incoming packets for the STA. If packets are buffered, the AP notifies the STA through the so called traffic indication map (TIM) which is part of the beacons. To receive those beacons, the STA need to wake-up for a short time and stays awake to receive the buffered packets if the TIM indicates new ones. This saves energy, but instead of waking up for the beacons, it might be a better solution to keep the STA asleep until it is really necessary to wake-up the STA e.g. if a packet arrives at the AP for the sleeping STA. Another drawback of using this duty cycled approach is the increase of the packet delay. This occur because packets are buffered on the AP until the STA receives the notification through the beacon to wake-up and receive those packets. Therefore, the delay is bounded to the beacon interval, which will be shown in this thesis. Beside battery powered devices like smartphones, the growing field of internet of things (IoT) might profit from reducing the energy consumption of WLAN as it would make this standard more suitable for battery powered sensor devices.

In Wireless Sensor Networks (WSNs) the concept of wake-up receivers is used to reduce power consumption. This is done by powering off the main transceiver and the microcontroller unit (MCU) until the wake-up receiver is triggered by an

external radio signal to wake them up again e.g. to measure the environment. The wake-up receiver itself is an additional ultra low power receiver which listens to incoming wake-up signals the whole time. As commercially available wake-up receivers can typically only react to one pre-programmed pattern, a more refined version of a wake-up receiver which is called a selective wake-up receiver (SWuRx) was introduced. With the developed SWuRx it is possible to wake up one node (unicast), a group of nodes (multicast) or all nodes (broadcast) depending on the address which is sent out [2].

Besides this thesis, the Institute of Electrical and Electronics Engineers (IEEE) also formed a taskgroup (IEEE 802.11ba) which addresses the extension of IEEE 802.11 standard by wake-up receiver functionality [3].

1.2 Contribution

In this thesis I show how to combine the advantages of the well proven standard WLAN [1] with the power saving concept of wake-up receiver (WuRx). To do so, I am going to connect a commercially available WLAN adapter with the SWuRx provided by CCS Labs and adapt all necessary software on the AP and STA side to support the new media access control (MAC) scheme. As WLAN adapter I will focus on devices using the *AR9xxx* chipset from Qualcomm Atheros. The first device is called WNDA3200 manufactured by Netgear. This device consists of two chips, the first one acts as a bridge to the PC the second one is the radio. The second device is manufactured by TP-LINK and is called TL-WN722N. It is using a single chip solution which contains the MCU as well as the radio. Both devices get connected through Universal Serial Bus (USB). These adapters are chosen because of their open sourced driver (Linux) and firmware [4], which is necessary to get low level access to the chips and their general-purpose input/outputs (GPIOs). The GPIOs will be used in two ways, as STA, the GPIO will wake up the network adapter from sleep. As an AP (using hostapd), the GPIO shall trigger the sending of the wake-up signal. As WuRx, the SWuRx provided by this group [2] is used.

The next part of this thesis comprises the measurement of the properties of the developed system. The measurements will include energy consumption, delay and goodput. To measure energy, it is planned to put a shunt resistor in between the network adapter and the power source (USB-port). Then an oscilloscope is used to measure the voltage drop across the shunt to calculate the current flowing through it and thus through the network adapter. Within the scope of this thesis, energy measurement is done on the STA side only, as the AP is considered to be always on. For measuring delay and throughput tools like *ping* and *iperf* will be used. The following three scenarios are considered: In the first scenario the STA will be

always-on and the values for delay, energy and throughput will be captured. These values will be used as a reference for the following scenarios. The next scenario will use the native power save which is already implemented in the driver and the same measurements will be done again. Then the WuRx will be used to do power save. As the beacon interval should be a crucial parameter in this configuration, a parameter study will be done on that. Furthermore the question will be answered how the beacon interval influences delay, energy consumption and throughput.

The rest of this thesis is structured as follows: In Chapter 2 a focused description of the IEEE 802.11 standard regarding the power save as well as the current implementation in the Linux kernel is shown. Furthermore, the used SWuRx is described as well as the device used to develop the SWuRx sender. Afterwards the related work follows. In Chapter 3 the development of the WLAN WuRx system is described. It includes the development of the SWuRx sender, the hardware modification which is done, as well as the modification on software side in the Linux kernel and the firmware. Moreover, the modification of the SWuRx and the problem using the native power save of the ATH9K adapter is described. The used measurement setup and the measurements itself are described in Chapter 4. Afterwards, the thesis is closed with the conclusion including the future work, Chapter 5.

Chapter 2

Fundamentals

2.1 WLAN IEEE 802.11

This section explains the necessary basics of the IEEE 802.11 [1] standard which are needed to understand the currently available power saving features and their functionality in WLAN. As the standard defines a lot of details, this section will only present a reduced form of it to understand the most important concepts which are needed within this thesis.

2.1.1 Frames

This section gives a short introduction to the frame formats specified by the IEEE 802.11 standard. This part will not cover all possible frame types and their options, instead, it will focus on the most important frames and their bit fields which are needed to understand how power saving is realized in WLAN. This includes beacon frames, null frames, and PS-poll frames.

Any frame uses the MAC frame format shown in Table 2.1. The minimum frame format consists of a *Frame Control*, a *Duration/ID*, an *Address 1* and the *FCS* field.

The first two bytes of a frame belongs to the *Frame Control* shown in Table 2.2. Within this, different informations about the frame are encoded like the *Protocol version*, the *Type* and *Subtype* of the frame and so on. But also parts regarding the

Bytes	2	2	6	6	6	2	6	2	4	0 - 7951	4
Usage	Frame Control	Duration	Addr 1	Addr 2	Addr 3	Seq. control	Addr 4	QoS Control	HT Control	Frame Body	FCS

Table 2.1 – IEEE 802.11 MAC frame format

power save are already included in these fields. The *Power management* field is used by a station (STA) to inform that it is in power save mode. The field *More Data* is used to indicate if there is more data in the buffer of the AP. More information about the concrete usage of these fields will follow.

The IEEE 802.11 standard distinguishes between three types of frames, these are control frames, data frames and management frames. Each type is furthermore divided into subtypes.

The first necessary frame to discuss is the beacon frame which belongs to the management group. The beacon frame is send by the AP and publishes important network information like the *SSID*, supported data rates and so on. It includes many information elements from which only a small excerpt is shown in Table 2.3. The column *order* shows the order of occurrence within the beacon frame body. Regarding power save, the beacon interval as well as the TIM, which is described after this section, plays an important role.

The next frame to be discussed is the PS-Poll frame, this frame belongs to the control group. The frame format is shown in Table 2.4. It is used to release buffered packets from the AP. The association ID (AID), a unique number assigned during association, identifies every STA connected to an AP. The *BSSID* is the AP MAC address and *TA* the address of the STA. A detailed description of how this frame is used during power save follows in Section 2.1.3.

Bits	2	2	4	1	1	1	1	1	1	1	1
Usage	Protocol Version	Type	Subtype	To DS	From DS	More Fragments	Retry	Power Management	More Data	Protected Frame	Order

Table 2.2 – IEEE 802.11 MAC control field

Order	Information	Description
1	Timestamp	Timestamp in microseconds
2	Beacon interval	The interval a beacon is sent out
4	SSID	The name of the network
10	Traffic indication map (TIM)	Describes

Table 2.3 – Interesting beacon members

Bytes	2	2	6	6	4
Usage	Frame Control	AID	BSSID	TA	FCS

Table 2.4 – PS-Poll frame format

The null frame is a data frame without any data. It can be exploited to inform an AP that the sending STA will enter power save by setting the *Power Management* bit in the *Control Field*.

2.1.2 (Delivery) Traffic Indication Map

The traffic indication map (TIM) is an information element and part of the beacons sent out by an AP. Table 2.5 shows the structure of this element. As every information element it consists of the two obligatory fields *Element ID* and *Length* followed by the information element specific data.

Byte	1	1	1	1	1	1-251
Usage	Element ID	Length	DTIM Count	DTIM Periode	Bitmap Control	Partial Virtual Bitmap

Table 2.5 – TIM information element

The field *DTIM Count* is used to indicate if the current TIM is a delivery traffic indication message (DTIM). A value of 0 indicates a DTIM. The field *DTIM Period* informs after how many beacons a DTIM will arrive. For example, a DTIM period of 1 indicates that in every beacon the TIM is a DTIM. After this, the *Bitmap Control* follows. Table 2.6 shows the individual bits and their usage. Bit 0 is used in combination with the *DTIM count* to indicate that broadcast/multicast packets are following. Bit 1 to Bit 7 is used to indicate the offset of the partial virtual bitmap. Then the trailing partial virtual bitmap is following which indicates all STA for which data is buffered on the AP.

Bit	0	1	2	3	4	5	6	7
Usage	AID 0	Offset of the partial virtual bitmap						

Table 2.6 – Bitmap control of TIM

The partial virtual bitmap consists of 251 bytes and respectively of 2008 bits. The bit position is interpreted as the AID to identify the STAs for which packets are buffered. Only the bits 1 to 2007 are used as STAs AIDs. For example, if traffic is buffered for the STA with AID 30, Bit 30 of the partial virtual bitmap is set to 1.

As described before, the partial virtual bitmap consists of 251 bytes, but not everything is transmitted, depending on how many bytes are needed to cover all bits which are 1. For example, if no traffic is buffered, the TIM information element is reduced to 6 byte as only 1 byte is spend for the partial virtual bitmap. If there are some STAs which need to be informed about unicast traffic, the size of the partial virtual bitmap is kept as low as possible by neglecting all STAs where no traffic is

pending and therefore those bits are 0. This is done by utilizing the offset and length field described before.

The following example shows the calculation of the partial virtual bitmap. Assuming that the STA with AID 28 has unicast packets buffered on the AP. Then, according to the IEEE 802.11 standard, the highest even byte up to which all bits are 0 is searched. This byte then is denoted as byte $N1$. As Bit 28 is within byte number 3, counted from 0, $N1 = 2$. Next, $N2$ denotes the starting byte after which all trailing bytes are 0. As in this case only the Bit 28 is set, all following bytes only includes 0. Therefore the byte number 3 is the searched candidate and accordingly $N2 = 3$. Then, the length and offset can be calculated like in Equation (2.1)

$$\begin{aligned} Length &= (N2 - N1) + 4 = 3 - 2 + 4 = 5 \\ Offset &= \frac{N1}{2} = \frac{2}{2} = 1 \end{aligned} \tag{2.1}$$

2.1.3 Power saving in IEEE 802.11

This section covers the power save mode described in the IEEE 802.11 standard. As the standard is highly detailed, only the most important aspects necessary for this thesis will be covered, this includes the defined power states of a STA in a basic service set (BSS) network, the frame types used and the technique which is used to realize this power save method.

The standard defines two power states for a STA in a BSS network. The first one is called *Awake* in which the STA is fully powered, the second one is called *Doze* in which it consumes low power but is also not able to transmit or receive any data. The power save mechanism is divided between the AP and the STA therefore both have to work together to allow the STA to enter power save mode while keeping the STA in sync with the network. According to the standard, the STA has to inform the AP before it enters sleep mode. This is done by setting the *Power management* bit described in Section 2.1.1 e.g in a null frame. After this frame is successfully exchanged and acknowledged by the AP, the STA is allowed to enter sleep mode. As the AP received this frame with the power save bit set, the AP records the STA as sleeping and therefore starts to buffer packets for it.

According to the IEEE 802.11 standard the AP must distinguish between unicast and multicast/broadcast traffic. This is necessary because multicast/broadcast traffic is destined for multiple STAs and therefore, before sending this kind of traffic, all STA should listen to it. Consequently two different methods are used for each kind of traffic.

To handle multicast/boradcast traffic, the standard introduces the DTIM described in Section 2.1.2. According to the previous description the DTIM information is carried by the beacons. To be aware when the next DTIM arrives, the beacon also

carries the DTIM count and interval, refer to Section 2.1.2 for detailed information. All sleeping STAs should wake-up for those DTIMs to receive multicast/broadcast traffic afterwards.

For unicast traffic, the standard has introduced the TIM described in Section 2.1.2. In comparison to the DTIM, this message is included in every beacon. The TIM is made out of a partial virtual bit map as described in Section 2.1.2. Within this, all STAs for which packets are buffered get identified by its AID. If a STA recognizes its own AID it has to wake-up and sent PS-Poll frames to release one frame after another. To make sure that the STA can release every packet which is buffered, the AP sets the *More Data* bit if there are still packets in the buffer, otherwise it will be unset and the STA may go to sleep again. The previously described behavior is visualized in Figure 2.1. To distinguish the power save state of the STA a straight line and dashed line is used. The straight line indicates that the STA is awake, the dashed indicates it in power save mode. The short wake-ups for beacons are also visualized.

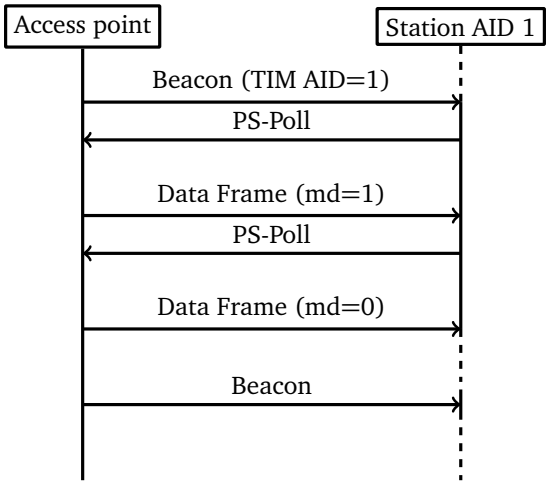


Figure 2.1 – Example of power save frame exchange defined in the IEEE 802.11 standard

2.2 Linux mac80211 and Power Save Implementation

This section discusses the state of the art wireless implementation within the Linux kernel. As the Linux kernel is an open source project, finding adequate documentation is still a problem, especially regarding the wireless implementation. Therefore most of the knowledge has been acquired by studying the source code of the necessary parts within the kernel implementation. Linux version 4.14.4 is taken as a basis for the research and the modifications like described later in Section 3.4. Within this chapter, the referenced source code belongs to the *mac80211* and therefore resides in the path *net/mac80211/*. If no other path is provided within a citation, the mentioned path is assumed.

One documentation which gives an overview of the Linux wireless implementation is a presentation by Berg from 2009[1]. He presented an architectural overview which is shown in Figure 2.2. This architecture shows four basic blocks. The *userspace*, *cfg80211*, *mac80211* and the *driver*. To interface between those blocks, four different application programming interfaces (APIs) are specified, these are *nl80211*, *wext*, *cfg80211_ops* and *ieee80211_ops*.

Starting from the *userspace* the wirless configuration and actions like scanning, changing channel and so on can be done through the *nl80211* in combination with

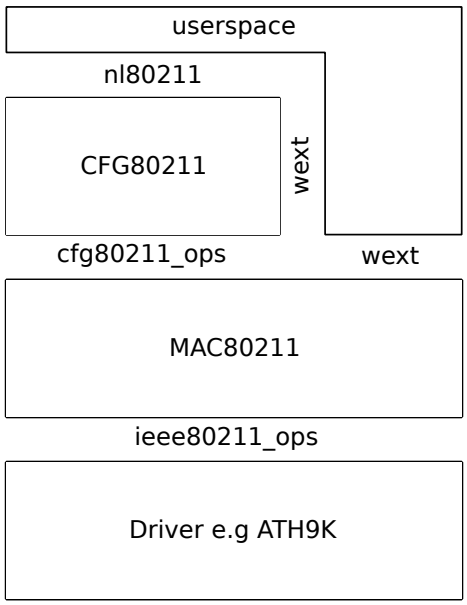


Figure 2.2 – Wireless software architecture in the Linux kernel

[1] `https://wireless.wiki.kernel.org/_media/en/developers/documentation/mac80211.pdf` Accessed: April 2018

cfg80211 or the *wext* interface. The *wext* interface is deprecated[2] and superseded by *cfg80211* and therefore neglected within this section. To guarantee backward compatibility, *cfg80211* emulates wireless-extensions (WEXT)[3].

The *cfg80211* implementation acts as a bridge between the userspace and kernel space. It provides a unified interface in form of callback functions to control wireless based network interfaces. Interested readers may refer to the documentation[4] to get an overview and description of all possible callbacks. The *mac80211* registers its implemented functions for each new device [5, `main.c`, Line 528] to the *cfg80211*.

The Linux developers differentiate between two wireless device types. The first one is called *FullMAC*, these are devices which implement the MAC in hardware. The counterpart is denoted as *SoftMAC*. The term *SoftMAC* denotes wireless devices which in comparison to a *FullMAC* do not implement the MAC on the hardware side, thus these devices directly pass raw 802.11 frames to the computer where a software side implementation has to parse and interpret them to act accordingly to the IEEE 802.11 standard. In the Linux kernel the *mac80211* module implements such a *SoftMAC*. Like the *cfg80211* the *mac80211* provides an API where the underlying driver has to register its functions on initialization. As describing every function is not necessary nor important to this thesis, interested readers might refer to the documentation[5].

As the *mac80211* implementations plays the main role in terms of power saving and understanding some parts of the *mac80211* is important for the later modification, a focused description of the important software flow paths as well as structures will be described with reference to the source code.

The first interesting flow path is the transmission path, it starts at *ieee80211_subif_start_xmit(...)* [5, `tx.c`, Line 3716]. From there, the packet runs through different processing steps in which information is added like the header [5, `tx.c`, Line 3585] or QOS information [5, `tx.c`, Line 3591]. The transmission path ends at the invocation of *drv_tx(...)* [5, `tx.c`, Line 1599] in which the packet is passed to the driver. Within the transmission path, a group of "tx handlers" are called [5, `tx.c`, Line 1679]. Two functions regarding power save belong to this group, these are *ieee80211_tx_h_dynamic_ps(...)* and *ieee80211_tx_h_ps_buf(...)* [5, `tx.c`, Line 1690]. The first one, *ieee80211_tx_h_dynamic_ps(...)*, is used to wake-up the WLAN adapter on STA side if the adapter is asleep and packets need to be sent. On the AP side, the function *ieee80211_tx_h_ps_buf(...)* checks if a STA is asleep and buffers the packets if necessary. Figure 2.3 visualize this path.

[2] `https://wireless.wiki.kernel.org/en/developers/documentation/cfg80211` Accessed: April 2018

[3] `https://wireless.wiki.kernel.org/wext-statement` Accessed: April 2018

[4] `https://www.kernel.org/doc/html/latest/driver-api/80211/cfg80211.html` Accessed: April 2018

[5] `https://www.kernel.org/doc/html/latest/driver-api/80211/mac80211.html` Accessed: April 2018

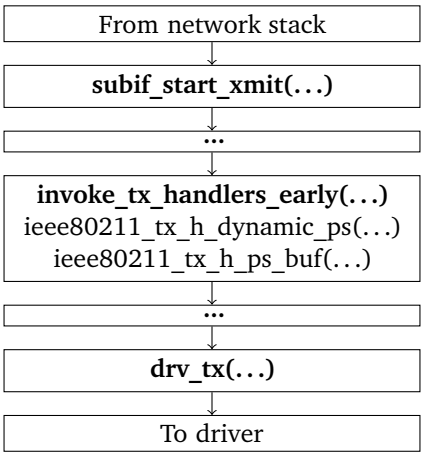


Figure 2.3 – Reduced transmission path call graph of *mac80211*

The receiving path starts by invoking *ieee80211_rx(...)* [5, `include/net/mac/mac80211.h`, Line 4033] from the driver side. After this, multiple sanity checks are done like checking if the packet is destined for that interface. If the packets passes these checks an ethernet frame is passed to the network stack by invoking *ieee80211_deliver_skb(...)* [5, `rx.c`, Line 2257]. Like in the transmission path, again a group of "rx handlers" are called [5, `rx.c`, Line 3363]. Within those, three handlers regarding power save can be found. These are *ieee80211_rx_h_check_more_data*, *ieee80211_rx_h_uapsd_and_pspoll* and *ieee80211_rx_h_sta_process* [5, `rx.c`, Line 3392]. The first one, *ieee80211_rx_h_check_more_data*, is used on STA side to check if the AP has more data by checking the *More Data* bit described in Section 2.1.1. This function triggers the sending of PS-Poll frames to release remaining packets buffered on the AP side. The function *ieee80211_rx_h_uapsd_and_pspoll* is part of the U-APSD which is not used and therefore not discussed within this thesis. The last function, *ieee80211_rx_h_sta_process*, updates the STA information. Therefore the power save status is changed within this function [5, `rx.c`, Line 1628]. Figure 2.4 visualize this path.

To keep track of a STA status the *mac80211* uses the *sta_info* structure defined in [5, `sta_info.h`, Line 474] where different attributes of a STA are composited. Within this structure a flag field is used to describe different states including the power save (WLAN_STA_PS_STA). A description of all flags can be found above the enumeration at [5, `sta_info.h`, Line 26].

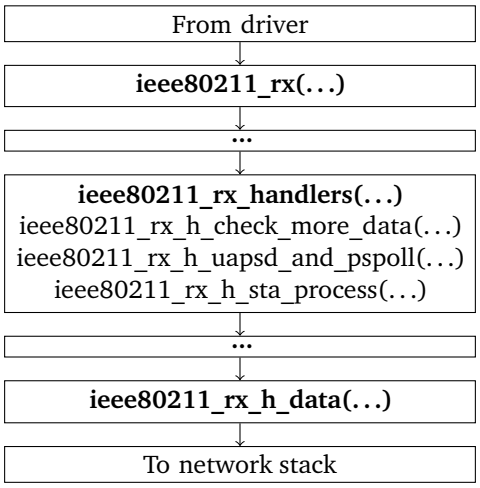


Figure 2.4 – Reduced receiving path call graph of *mac80211*

2.2.1 Power Save

The *mac80211* implements different power saving mechanisms including the one described in Section 2.1.3. Which one is used depends on the hardware capabilities which needs to be correctly set by the underlying driver while registering to the *mac80211* module. On driver level these capabilities can be set through the macro *ieee80211_hw_set(...)* [5, `include/net/mac80211.h`, Line 2265] which prefixes *IEEE80211_HW_*. The capabilities regarding power save are:
IEEE80211_HW_SUPPORTS_PS,
IEEE80211_HW_PS_NULLFUNC_STACK,
IEEE80211_HW_SUPPORTS_DYNAMIC_PS.
With *IEEE80211_HW_SUPPORTS_PS* the driver signalizes to the *mac80211* that the hardware supports power save. According to that, the code to set the interface asleep is enabled. The flag *IEEE80211_HW_PS_NULLFUNC_STACK* implies that null frames needs to be handled in the *mac80211*. The last flag *IEEE80211_HW_SUPPORTS_-DYNAMIC_PS* signalizes that the hardware will take care of the dynamic power save feature which effectively disables the dynamic power save in the *mac80211* which is described in the following.

The *mac80211* implements a so called 'dynamic power save' feature[6]. This power save does now follow the IEEE 802.11 standard completely. As described in Section 2.1.3 power saving in a BSS network is done by setting the STA asleep after the AP has been informed. While asleep the AP buffers packets for the sleeping STA and notifies it through the beacons which are sent frequently. Up to this point the dynamic power save follows the IEEE 802.11 standard, but instead of releasing the packets by PS-Poll frames, the dynamic power save leaves the power save mode and

informs the AP about that by sending a null frame with the power save bit unset. Thus, the AP releases all frames buffered for the STA. After a specified time, by default 100 *ms*[6] [5, `mlme.c`, Line 1481], the STA enters power save again if no traffic is following [5, `rx.c`, Line 2657]. The AP is informed by a null frame with the power save bit set. The delay after which the STA goes to sleep again is controlled through the old WEXT system by the userspace *iwconfig* tool. Figure 2.5 shows the frame exchange using the dynamic power save and Figure 2.6 visualizes the described behavior.

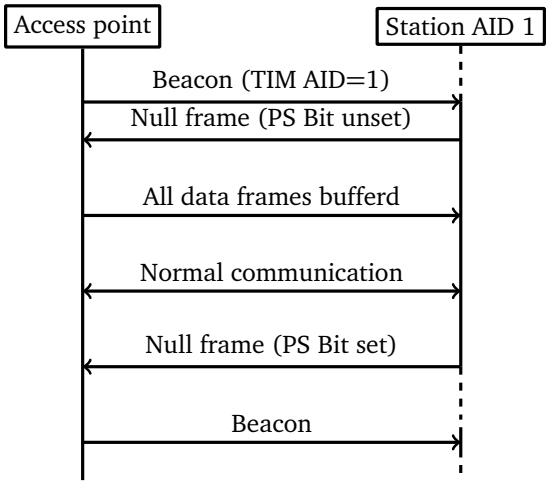


Figure 2.5 – Example of power save frame exchange in *mac80211*

[6] `https://wireless.wiki.kernel.org/en/users/documentation/dynamic-power-save` Accessed: April 2018

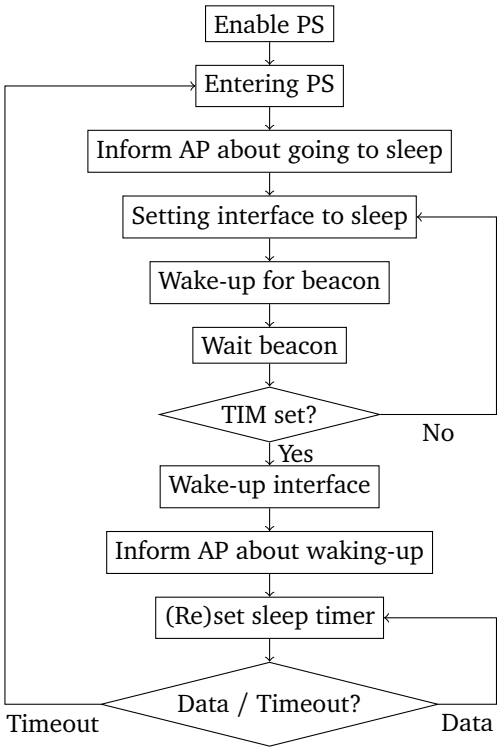


Figure 2.6 – Dynamic power save flow of *mac80211* on STA side

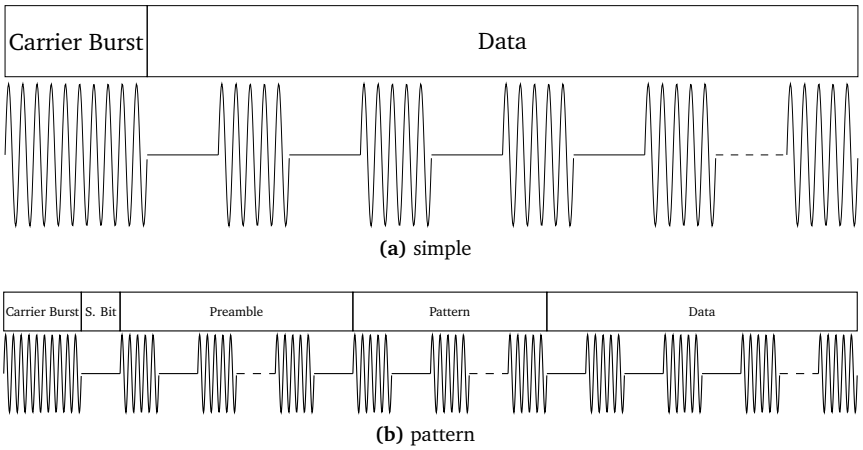
2.3 Selective Wake-up Receiver

In WSNs the new concept of WuRx was introduced to further reduce power consumption. To achieve this, an additional receiver is added to the sensor node. The purpose of this receiver is to listen for a wake-up signal which signalizes the node to wake-up the main MCU and transceiver from low power state to initiate the communication. The selective wake-up receiver introduced in [2] is constructed around the commercial available RFID wake-up receiver AS3933. The main features of the AS3933 are a low current consumption (1.7 μA if all three antenna channels are used) and two different wake-up protocols. Furthermore the device supports different frequency bands (15 kHz to 150 kHz) and symbol rates. All this configuration is done through a Serial Peripheral Interface (SPI).

As already mentioned, the device supports two wake-up protocols. Figure 2.7 shows both protocols. The first one is just a simple frequency detection. After a specified carrier burst is sent, the wake-up pin of the AS3933 toggles to high. Additionally trailing data can follow after the carrier burst.

The second wake-up protocol is more complex. Again, the transmission is initiated by a carrier burst followed by a separation bit denoted as 'S. Bit' in the figure.

Then a preamble and pattern is added. The pattern is changeable by reconfiguring the associated registers [6]. After the correct pattern has been recognized, the wake-up pin switches to high state. An additional data stream may follow after the pattern.



(a) simple

(b) pattern

Figure 2.7 – Wake-up protocols supported by the AS3933

A full block diagram of the selective wake-up receiver is shown in Figure 2.8. It consists of four basic blocks. The AS3933 presented before, an envelop detector which is connected to one antenna input of the AS3933, an address logic and a MSP430 on a TI Launch pad to control the AS3933 and the pattern.

The envelop detector is tuned to 868 MHz and feeds the demodulated signal into one of the three antenna inputs of the AS3933. To control the AS3933 through the 868 MHz an On-Off-Keying (OOK) modulated signal is sent. The 868 MHz carrier is modulated with the carrier for the AS3933 which can be in a range of 15 kHz to 150 kHz depending on the configuration. This approach is called subcarrier modulation.

Figure 2.9 shows a block diagram of the evaluating logic. The address logic implements the new addressing scheme proposed in [2]. It uses shift registers which get fed from the AS3933 data and clock lines. Another shift register is used to set a predefined pattern. The pattern is set by the MSP430.

The evaluation logic works as follow: If data is sent through the AS3933, every bit is applied to the *data* synchronized by the clock sent through the *clock* signal. Therefore, the following shift registers gets filled with the data accordingly. To prevent toggling of the output, which would lead to wrong wake-up signals, while shifting in the data, the *counter* is used. The counter is configured so that after eight clock cycles – 4 bit address and 4 bit mask – it output switches to *high* level and therefore allows a possible *high* signal from the evaluation logic to pass the last

and-gate in the chain. To reset the counter to the initial state, a timer created of a RC element is used.

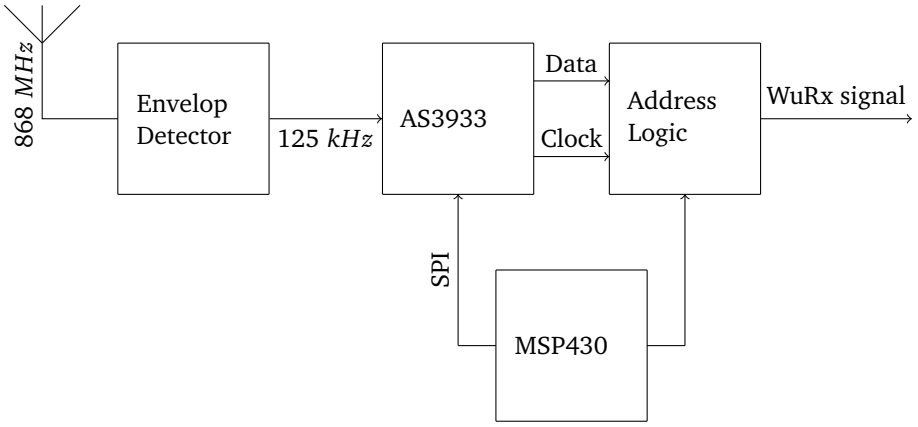


Figure 2.8 – Block diagram of the SWuRx proposed in [2]

The received data is divided into two parts, the address itself and a mask. The mask is used to decide which bit of the address must be equal to the corresponding bit in the pattern, all other bits are treated as don't care.

The last block, in Figure 2.9 denoted as *MSP430*, is a TI Launchpad from Texas Instruments. To send a wake-up signal the authors used a software defined radio (SDR).

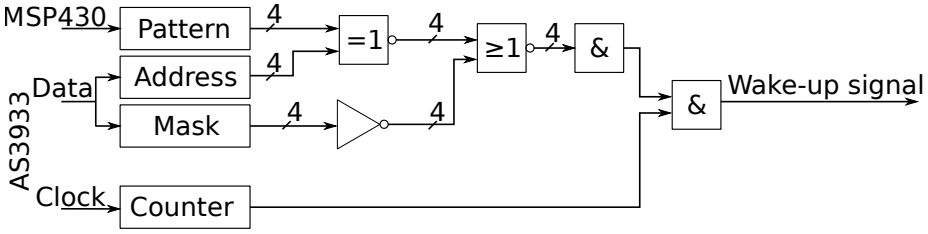


Figure 2.9 – Address evaluation logic of the SWuRx proposed in [2]

2.4 ESP32 Remote Module

One part of this thesis is to create an wake-up sender which can control the SWuRx described before. This is done with the help of an ESP32. The ESP32 is a feature rich MCU which comprises a so called remote module. This peripheral is intended to be used as a remote controller e.g for infrared based communication[7]. Due to the versatility of this module it is possible to generate arbitrary digital signals. The

[7] `http://esp-idf.readthedocs.io/en/latest/api-reference/peripherals/rmt.html` Accessed: April 2018

benefit of using this module is that the real-time requirements become easy to achieve as the hardware module takes care of it. To use the remote module it has to be configured at first. To do so, a C-structure is defined which is passed to the initialization function. Within the context of this thesis, the most important members of the configuration structure are shown in Table 2.7.

Member	Description
clk_div	Configures clock division from source clock (1-255)
$carrier\_freq\_hz$	Configures carrier frequency
$carrier\_duty\_percent$	Duty cycle of carrier
$carrier\_en$	Enables/Disables the carrier

Table 2.7 – Relevant configuration structure members of ESP32 remote module

After the remote module has been configured, the structure from Listing 2.1 can be used to define signal items. Each signal item is defined by the members *duration0*, *level0*, *duration1* and *level1*. The *duration* members define how long the levels (0/1) defined by the *level* members are hold on the selected GPIO. More complex signals can be achieved by concatenating multiple items into one array. The duration is measured in ticks, the period of the clock. Equation (2.2) is used to calculate the real time elapsed for a specified number of ticks, where f_{RMT} denotes the frequency, the remote module is driven by, and clk_div the clock divider which is configured by software in a range of 1 to 255.

$$t = T * ticks = \frac{clk\_div}{f_{RMT}} * ticks \tag{2.2}$$

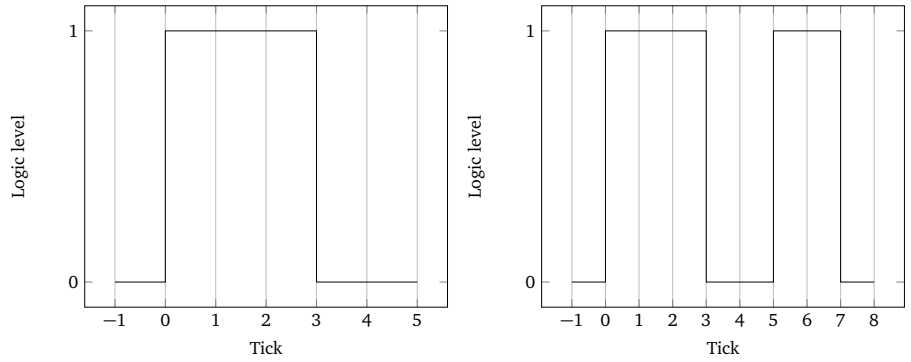
As an example Listing 2.2 shows two definitions of signal items. The first one is a single item which generates 3 ticks *high* and 2 ticks *low*. The second one combines two signal items and leads to 3 ticks *high* and 2 ticks *low* followed by 2 ticks *high* and 1 tick *low*. Figure 2.10 shows the resulting waveform for each of the defined items.

```
typedef struct {
    union {
        struct {
            uint32_t duration0 :15;
            uint32_t level0 :1;
            uint32_t duration1 :15;
            uint32_t level1 :1;
        };
        uint32_t val;
    };
} rmt_item32_t;
```

Listing 2.1 – RMT item structure for signal definition

```
rmt_item32_t item = {1, 3, 0, 2};
rmt_item32_t multiItem[] = {{1, 3, 0, 2}, {1, 2, 0, 1}};
```

Listing 2.2 – Example signal definition



(a) Single element ($item$) **(b)** Two elements ($multiItem[]$)

Figure 2.10 – Generated signal according to Listing 2.2

2.5 Related Work

As WSNs nodes operate with limited resources, reducing the energy consumption is one key attribute for the network lifetime. Thus, reducing the power consumption of the nodes within the WSN is crucial [7]. One factor which influences energy consumption is idle listening. Idle listening occurs if the node listens to the medium but no traffic is intended for it. To face this problem, a common way is to use a duty-cycling like MAC protocol.

In such type of protocols, the energy consumption is reduced by disabling the main transceiver and the MCU which comes along with some drawbacks. First, using such protocols is always a tradeoff between energy consumption and responsiveness,

therefore those types of protocols might not be qualified for the usage in time critical scenarios [8]. Second, duty cycling needs synchronization which rises the complexity of the MAC, especially in huge WSNs. Furthermore control frames are needed which also introduce additional energy consumption within the WSN [9]. As choosing a good balance between energy consumption and responsiveness is highly application dependent, different applications led to different MAC protocols in the field of WSN. The magnitude of this has already been paraphrased as the "MAC alphabet soup" [10].

Two examples of such protocols are described in the following: The first simple example for this type of MACs is the B-MAC [11]. Like other duty-cycled protocols, B-MAC saves energy by disabling the main transceiver and only turns it on from time to time, specified by the duty-cycle. To detect a wake-up request, the transceivers of a node performs a so called low power listening, in which a simple energy detection, similar to clear channel assessment (CCA), is done. If the channel is detected as clear, the node directly enters sleep mode again until the next period. Otherwise, if a carrier is detected, the receiver waits until the end of the preamble and switches to full power mode to receive the trailing data. To achieve a reliable communication, the transmitting node must send a preamble with a length at least as long as the sleep period of the receiving node.

Another example for such types of MACs is X-MAC [12]. X-MAC again is a duty-cycling MAC protocol which uses preambles to signalize communication attempts. But in contrast to B-MAC, X-MAC splits up the long preamble into small strobes with pauses in between. Within the preamble, an address is used to identify the destined node. Using the address reduce the overhearing problem as nodes for which the data is not intended can quickly enter sleep mode again. In between the strobed preambles, the intended receiver can interrupt the sender by sending an acknowledge after which the sender transfers the data. This reduces energy consumption and delay, as it is not necessary to wait for the full preamble to end like in B-MAC.

The answer to this conceptional drawbacks described before is a so called wake-up receiver. One of the first paper which introduced this novel concept to face this problem was written by Gu and Stankovic, they named their concept "radio-triggered wake-up system" [8]. By adding an additional, receiver which is only responsible to wake-up the main circuitry of the sensor node, they achieved a huge amount of energy saving potential. They were able to reduce the energy consumption by 70% compared to duty-cycled MAC implementations. Today, these additional low power receivers are better known under the term wake-up receiver.

Following this promising approach, researchers have started investigate the benefits of WuRx MAC protocols in contrast to duty-cycled ones. Furthermore

different addressing schemes and different methods to realize such a WuRx system in hardware were investigated.

In [13] the authors compared different duty-cycled MAC protocols including B-MAC and X-MAC to a WuRx like MAC protocol. Those kind of protocols including an always-on solution and a WuRx solution has been compared within three different scenarios. Those are a single hop network, where a mobile station collects data, a static multihop scenario where the network is build like a binary tree and a dynamic multihop scenario, where again a mobile node should collect data but now from multiple sensors which are only reachable though multiple hops. They could show that a WuRx based MAC protocol is clearly superior in comparison to duty-cycled MAC protocols in all three scenarios with a small exception which could be neglected as it consider the receiver to be always-on. From this they concluded that a WuRx system can improve energy consumption, delay and packet delivery ratio (PDR) drastically and furthermore reduces the complexity of the software, only to the cost of a little more hardware.

In [10] the authors proposed a wake-up receiver consisting of three general blocks, a RF front-end, a back-end and a RF energy harvester. Within the RF front-end, the whole receiving circuitry is covered like a matching network, amplifiers and filters, a demodulator etc. In the backend they proposed to be an ultra low power (ULP) MCU, a correlator as well as an interrupt generator. An additional RF energy harvester is also proposed to reduce the external power consumption of the wake-up receiver partly or completely. Figure 2.11 shows the general architecture as proposed in [10]. Beside the proposed general wake-up receiver the authors also state basic attributes

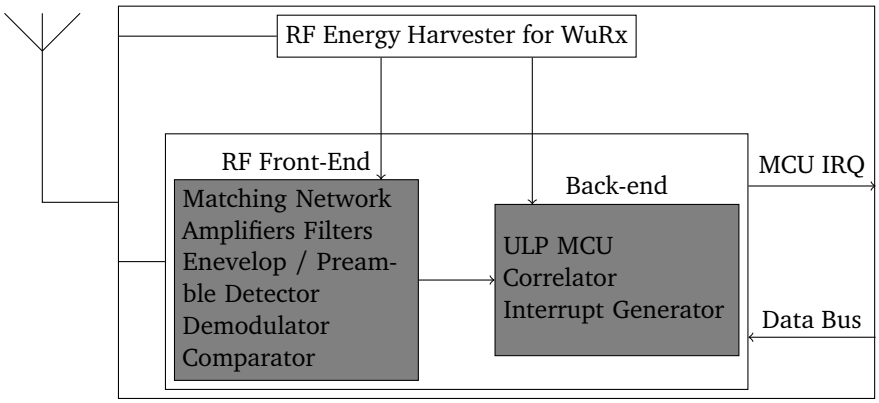


Figure 2.11 – Proposed [10] general wake-up receiver architecture

and requirements which should be considered while working with wake-up receivers. One of the most important attribute of a wake-up receiver is the power consumption as the receiver is considered to be always on. Therefore, according to [10], a trade-off between power consumption and sensitivity must be made. Reducing the energy

consumption can be achieved by keeping the modulation schemes simple. Another way to reduce energy consumption is to remove 'unnecessary' parts of the receiver like the low noise amplifier (LNA) which is responsible for amplifying the received signal. But removing the LNA will come at the cost of reducing the sensitivity and therefore the range if a constant transmit power is assumed. This becomes more crucial if the sensor node itself should be able to wake-up another node as a higher transmission power is needed with worse receiver sensitivities assuming the same distance. As stated above, an additional RF energy harvester can reduce these shortcoming by powering some parts of the wake-up receiver.

In [8] the authors proposed a simple energy detection circuitry which is selective to a specified frequency. Doing it this way, no additional power is needed to drive the wake-up receiver and therefore it can be categorized as a passive one. The drawback of this simple structure is the missing selectivity and addressing scheme, as this system just reacts to any energy on the pre-configured frequency. Assuming that there are more than one node using this circuitry, all would wake-up if they share the same frequency configuration. To face this problem, the authors further proposed to use multiple energy detecting circuits with different frequencies followed by logic circuits. By sending different combinations of frequencies at the same time, different nodes are addressable. With this configuration they were able to achieve a simple addressing scheme at the cost of rising the hardware effort as every frequency needs its own energy detection circuitry including the digital logic behind it.

Another way of supporting selectivity without additional hardware was proposed in [14]. They called their method "Time-Knocking". To decode the wake-up address the MCU itself is used which reduces hardware effort. At the first glance, using the MCU to decode the address seems like wasting energy. But instead of keeping the MCU on while decoding the wake-up address, they are exploiting the internal timer to decode it. To do so, they are starting the internal timer after the MCU first receives a wake-up signal via its interrupt. Afterwards the MCU directly goes to sleep again. If a second pulse arrives, the MCU wakes-up for a second time to capture the time elapsed which is used to decode the address.

In [2] a so called selective wake-up receiver was presented. They proposed a wake-up receiver which is build around an AS3933 [6]. The addressing scheme allows to wake-up a single node or a group of nodes by sending an address and a mask. The bits of the mask decide which bit of the address should be compared with a predefined pattern. As the mask is also transmitted from sender side, it is possible to select and create different groups without reconfiguring the nodes which saves energy. They concluded that this refined version can reduce the energy consumption even more with just a small hardware effort. A more detailed description of this wake-up receiver will follow in Section 2.3 as this device plays an essential role within this thesis.

As WLAN was designed for high speed communication between different computers, the energy consumption was a secondary objective. Nevertheless, also in this field researchers investigated the power save capabilities and their effects on network performance.

In [15] the authors studied the downlink delay – AP to STA –. If in power save mode, this delay is bounded to the beacon interval as the STA gets notified about buffered packets through the beacons. They addressed this problem by sending PS-Poll frames as an proactive way to the AP to release possibly buffered packets. Upon this technique, they developed an algorithm which dynamically changes the PS-Poll frame interval depending on the packet interarrival time. They have shown that their algorithm can hold an upper bound while keeping signaling load (PS-Polls) low and are therefore independent from the beacon interval.

In [16] the authors proposed a wake-on-wireless system to face energy problems on Personal Digital Assistants (PDAs). They have examined how to improve the energy consumption of PDAs while keeping it reachable through Voice over IP (VOIP). They have created a prototype which uses additional hardware to send a wake-up signal and receive it on the PDA. Their architecture consists of two modified PDAs, a laptop connected to their wake-up sender and one server to coordinate registrations and VOIP calls. If device A wants to call B, A informs the server which then informs the laptop with the connected sender to send the wake-up signal. Subsequently the PDA of B turns the WLAN adapter on and is reachable for the call from A. Using this system they were able to improve the battery lifetime by 115% over the native power save mode provided by the IEEE802.11 standard [1]. Basically that work is similar to this thesis but there are still huge differences. In their work they focused on a VOIP scenario which they especially created a server for. In contrast to that, this thesis investigates the general case where the AP WLAN adapter directly gets the capability of sending the wake-up signals and the STA WLAN adapter the capability of receiving them. Furthermore modifications in the firmware and the Linux kernel are made instead of building a system on top of it, making the system agnostic to the application layer.

Researchers also have started to examine the feasibility of WLAN in WSNs. In [17] they used an G2M5477 WLAN module to investigate the energy consumption of a possible wireless sensor node. As a scenario they have chosen a sensor node which only needs to send data to the AP, therefore once asleep the sensor is no longer reachable for external requests. Within their setup they have shown that it is possible to keep the sensor running for years, depending on the wake-up interval. Furthermore they have concluded that in comparison to the energy needed by sending and receiving is less important than the energy used during the wake-up procedure.

But researchers did not stick to the energy saving on STA side only, they flipped this concept to save energy on the AP side too. In [18] the authors started such an approach. They showed how to modify an already available AP to bring wake-up functionality to it. They used a Linksys WRT54G connected to an additional circuitry to control the power state of the WLAN adapter. On the other side a laptop is used as a STA connected to a low power radio which is used to send the wake-up request to the AP.

As already noticeable in some research results, one of the main contributor to the energy consumption in WLAN is idle listening [19] [16] which could be prevented by a WuRx system.

Since 2016 the IEEE has started to extend the IEEE 802.11 standard by wake-up receiver functionality [3]. The new standard IEEE 802.11ba propose an additional receiver to wake-up the main transceiver based on IEEE 802.11. As the work group is currently working on this topic, only few documentation mainly in form of presentations are available from the website showing the current progress of the group [3]. Figure 2.12 shows the WLAN wake-up architecture proposed in one presentation[8] held in the context of the IEEE 802.11ba task group. Like in WSNs an additional receiver is added to the traditional IEEE 802.11 transceiver to wake it up. Using the proposed architecture, the task group wants to address the energy consumption in IEEE 802.11. At the same time, they also addresses the problem with delay while the STA is in power save mode. Also use cases were already presented, like IoT sensors running on coin batteries[9].

This thesis is highly related to the work currently done by the task group. In contrast to the task group, this thesis implements a prototype consisting of a commercially available WLAN adapter and a wake-up receiver presented later. Afterwards the performance of the prototype is evaluated.

[8] `https://mentor.ieee.org/802.11/dcn/16/11-16-0722-01-0000-proposal-for-wake-up-receiver-study-group.pptx` Accessed: April 2018

[9] `https://mentor.ieee.org/802.11/dcn/16/11-16-1465-00-0wur-wur-usage-model.pptx` Accessed: April 2018

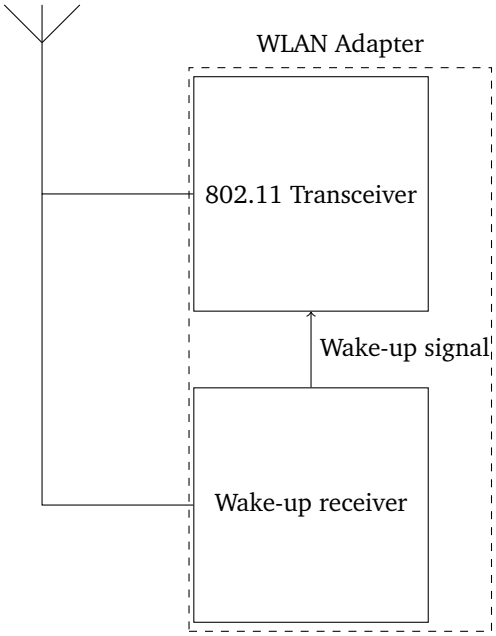


Figure 2.12 – Proposed[8] wake-up architecture for IEEE 802.11

Chapter 3

Developed WuRx System

The starting point for the developed system is the BSS mode of WLAN. This results in a star topology with one AP and zero or more STAs. With this setup it is already possible to use the built-in dynamic power save which is provided by the *mac80211* implementation of the Linux kernel described in Section 2.2.1. This setup is taken as a base for the extension with a wake-up receiver and a sender which generates and receives the wake-up signal to save energy on the STAs side. Figure 3.1 shows a block diagram of the proposed system with the wake-up sender as well as the wake-up receiver attached.

The figure shows four components: a STA, an AP and additionally the wake-up receiver marked with *SWuRx* as well as the SWuRx sender marked as *SWuTx* connected to the AP. On the left-hand side resides the STA which is connected with two wires to the wake-up receiver discussed in Section 2.3 before. One signal is connected from the SWuRx to one of the GPIOs of the WLAN adapter to wake it up. The second signal is used to create a serial connection between the WLAN adapter and the wake-up receiver. This connection is used to set the address of the wake-up receiver. The AP resides on the right-hand side of the figure. It is connected to the SWuRx sender. This signal is used to inform the SWuRx sender for which address it should generate a wake-up signal.

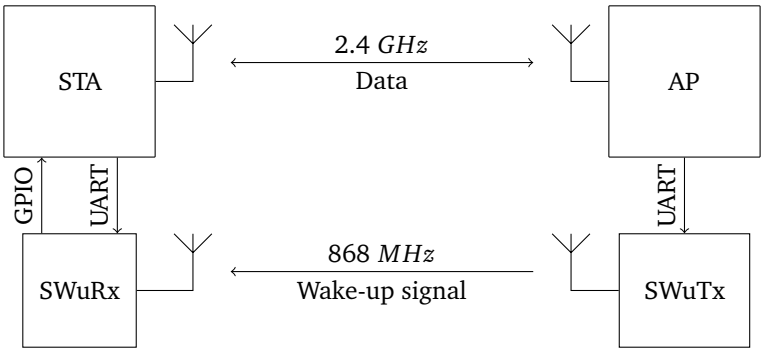


Figure 3.1 – Block diagram of BSS scenario with wake-up extension

3.1 SWuRx Sender

As mentioned in Section 2.3 the wake-up receiver was controlled by a SDR before. To simplify the communication between the AP and the wake-up receiver and furthermore create a more realistic prototype, a wake-up transmitter has been developed to control the wake-up receiver.

The base of the SWuRx sender is the ESP32 connected to a simple 868 MHz OOK transmitter. The transmitter has just three terminals, two for power supply and one for the data, whereas a *high* level on this connection starts the transmission of the carrier and a *low* level stops it.

To generate the signal, the remote module of the ESP32 described in Section 2.4 is used. To control the AS3933 the selected carrier frequency of 125 kHz must be generated. The remote module of the ESP32 is already capable of doing this by setting $carrier\_en = True$. The frequency is selected by setting $carrier\_freq\_hz = 125000$. Lastly the duty-cycle is set to $carrier\_duty\_percent = 50$. Now, with this configuration being set, the remote module does not only set the selected GPIO to *high* if a *high* level is requested, instead it generates a square wave with a duty-cycle of 50 % and a frequency of 125 kHz. This signal is fed to the 868 MHz OOK transmitter which then, according to the signal, is toggled on and off with a frequency of 125 kHz. Therefore, the 868 MHz carrier is modulated by the 125 kHz carrier which is known as subcarrier modulation.

As described in Section 2.4 the remote module uses a C-structure to describe the levels and the timings of a signal. According to [6] data is transferred in a manchester encoded manner. Therefore two symbols are defined: *one* and *zero*. These symbols are composited of two bits. The symbol *one* is represented by first sending a one bit followed by a zero bit. The symbol *zero* is represented by sending it the other way round, first a zero bit followed by a one bit. Figure 3.2 shows an encoding example. With respect to this, two basic signal blocks were defined to

represent those symbols and one for representing the carrier burst at the beginning. These are listed in Listing 3.1.

The element *BURST* is used to generate the burst followed by one separation bit. The minimum burst length denoted as $BURST\_LENGTH$ depends on the configured operating frequency range and the carrier frequency as well as the clock frequency. For the used frequency range from 95 kHz to 150 kHz the minimum burst length can be calculated by Equation (3.1) [6]. The clock frequency denoted as f_{clk} is given by the internal RC oscillator and f_{carr} is the carrier frequency used, in this case 125 kHz.

$$BURST\_DURATION = 16 * T_{clk} + 16 * T_{carr} = 16 * \frac{1}{f_{clk}} + 16 * \frac{1}{f_{carr}} \tag{3.1}$$

To evaluate the BIT_LENGTH the bit duration is needed. The AS3933 allows to set the bit duration in multiples of the clock generator periods, from 4 to 21 periods. To be as fast as possible, the lowest value 4 and therefore the highest bitrate of 8192 Bit/s is used. As two bits are used to represent one symbol, the effective data rate is reduced to 4096 $Symbols/s$. Equation (3.2) is used to calculate the bit duration.

$$BIT\_DURATION = 4 * T_{clk} = 4 * \frac{1}{f_{clk}} \tag{3.2}$$

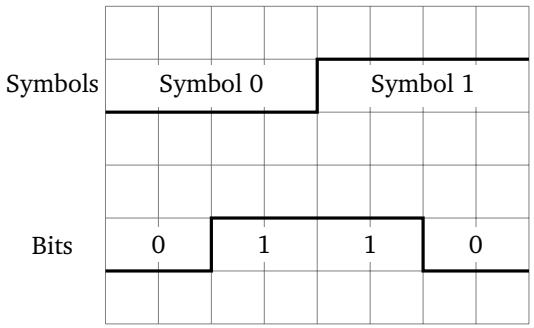


Figure 3.2 – AS3933 manchester code

```
1 static const rmt_item32_t BURST = {.level0 = 1, .duration0 =
      BURST_LENGTH, .level1 = 0, .duration1 = BIT_LENGTH};
2 static const rmt_item32_t logicOne = {.level0 = 1, .duration0 =
      BIT_LENGTH, .level1 = 0, .duration1 = BIT_LENGTH};
3 static const rmt_item32_t logicZero = {.level0 = 0, .duration0 =
      BIT_LENGTH, .level1 = 1, .duration1 = BIT_LENGTH};
```

Listing 3.1 – Definition of all necessary symbols

Using Equation (2.2) from the remote module described in Section 2.4, the BIT_LENGTH and $BURST\_LENGTH$ can be calculated by Equation (3.3) and Equation (3.4) respectively. Whereas f_{RMT} is the clock frequency of the remote module and clk_div the selected divider.

$$BIT\_LENGTH = BIT\_DURATION * \frac{f_{RMT}}{clk\_div} \tag{3.3}$$

$$BURST\_LENGTH = BURST\_DURATION * \frac{f_{RMT}}{clk\_div} \tag{3.4}$$

Using those basic blocks, an array is created starting with the burst and followed by the symbols representing the user data.

Due to a bug in the AS3933 the usage of the simple wake-up protocol is not possible, therefore the complex one is used with the drawback of a higher delay. As described in Section 2.3 the complex protocol needs the carrier burst followed by a preamble and the configured pattern, which shall not to be confused with the pattern of the external logic block. Then the trailing data, in this case the address and mask for the logic behind the AS3933, is following.

As for this thesis only unicast is needed, the program on the ESP32 accepts a hex character between 0 and F through the universal asynchronous receiver transmitter (UART). According to this, the program generates the appropriate signal with all mask bits set to one and the address bits to the transferred value. To keep the delay introduced by UART as low as possible, no special protocol is used. After receiving the character, the program starts to generate followed by sending the signal.

3.2 WLAN Adapter

As WLAN adapter the previously mentioned WNDA3200 from netgear is used. These adapters use two chips, the first one is called AR7010 and can be seen as a MCU which main purpose is to glue together USB with Peripheral Component Interconnect Express (PCI-E) as the radio chip AR9280 is a PCI-E WLAN adapter. If the WLAN adapter is set to power save, only the radio reduces the current consumption. These devices are used because the driver (Linux) as well as the firmware [4] is available as open source code and therefore easily changeable to the necessary needs within this context.

3.2.1 Hardware Modifications

The printed circuit board (PCB) of the WLAN adapters have two light emitting diodes (LEDs) and one button populated. Furthermore there is also one test pad to the UART of the AR7010.

As described at the beginning of this chapter, at the AP side the UART is used to send the address which should be woken up by the wake-up transmitter. On the STA side it is used to configure the address the wake-up receiver should listen to. The electrical connection is done through a fine isolated copper wire to reduce any mechanical stress to the pad.

For the STA side it is necessary to get the wake-up signal back from the wake-up receiver to the WLAN adapter. To create that connection a second wire of the same type is used and soldered to the already existing button contact, as this is easier to do instead of soldering it directly to a chip with such a small pitch. Figure 3.3 shows the modified hardware.

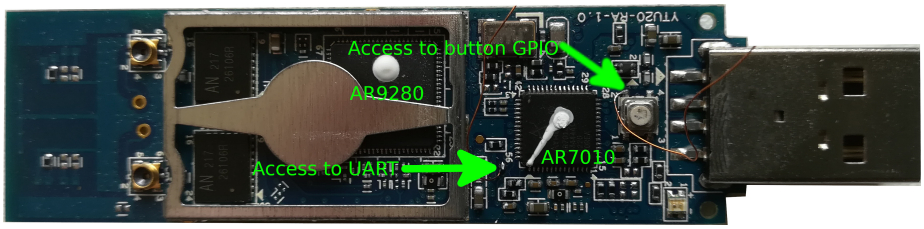


Figure 3.3 – Opened WLAN adapter with one wire connected to the UART testpad and one to the populated button

To connect the SWuRx to the WLAN adapter the circuit shown in Figure 3.4 is used. The external part consists of the resistor R_b and the transistor Q_1. The SWuRx is connected to the transistor through the resistor. The dashed line defines the equivalent circuit diagram on the WLAN adapter. The resistor R_p pulls the GPIO against the supply voltage which leads to a logical one of the corresponding bit in the input register. If voltage is applied to the input denoted as *SWuRx*, the transistor becomes conductive and shorts out the voltage which leads to a logical zero. The resistor R_p limits the current to prevent damage. This way, the functionality of the button for manual interaction is kept as the open-collector-circuit behaves like the button and represents an open circuit until the base is powered. The wake-up receiver is connected to the base of the transistor. The whole circuit works as follows: If the wake-up receiver receives a correct wake-up address, the wake-up signalizes this by rising a high level on its output, thus the transistor becomes conductive and the GPIO of the AR7010 is pulled low which can be recognized by reading the GPIO input registers. This is used to detect a wake-up request on firmware and kernel level.

Bit 13 of the GPIO input register [5, `drivers/net/wireless/ath/ath9k/reg.h`, Line 1181] represents the button.

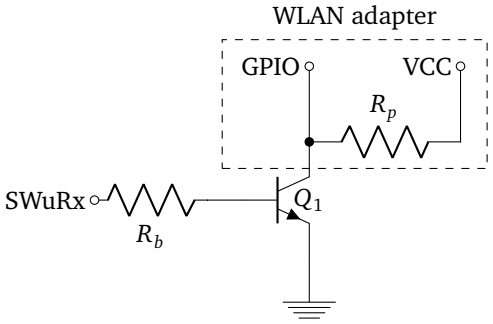


Figure 3.4 – Open collector circuit used for connecting the SWuRx to the WLAN adapter

3.2.2 Firmware

The firmware of these adapters is divided into two parts, the first one is pre-programmed onto the on-board flash-chip and the second one resides on the computer as a binary file. The purpose of the first part is to initialize the MCU to a simple state where it waits for the second part of the firmware which will be transferred by the driver through USB. After transferring the firmware, the second part is executed. As the second part is just a file on the computer it can be easily exchanged without complex reprogramming of the internal flash-chip.

To clarify what changes were made in the firmware, this section will refere to the original code [4]. As base the tag 1.4.0 is used. The base path for referenced files is *open-ath9k-htc-firmware/target_firmware/wlan/*. If files are not located there the full path is provided.

The main task of the firmware is to create a bridge between the Linux kernel driver and the PCI-E radio chip. To do this, the original authors introduced a command like facility. Different functions are enumerated [4, `include/wmi.h`, Line 131]. A lookup table [4, `include/if_ath.c`, Line 1745] connects these commands accordingly to the target functions through pointers. Listing 3.2 shows an example for such a function. Every function gets the command which was issued, the sequence number, a buffer for data from the host and a length field indicating the data length in the buffer. Also, every function ends by calling *wmi_cmd_rsp(...)* which takes the command executed, the sequence number, as well as a buffer and the length of data which should be returned to the host. It is also possible to return nothing by setting $Length = 0$. The counter part to this command system is in the kernel driver described in Section 3.2.3

```
1 static void handle_echo_command(void *pContext, A_UINT16
      Command, A_UINT16 SeqNo, A_UINT8 *buffer, a_int32_t Length)
2 {
3   wmi_cmd_rsp(pContext, WMI_ECHO_CMDID, SeqNo, buffer, Length);
4 }
```

Listing 3.2 – WMI callback function example

The firmware was extended by two features. The first feature enables the firmware to pass characters to the UART of the AR7010. As described at the beginning of this chapter this functionality is needed on the STA side to configure the wake-up pattern and on the AP side to control the SWuRx sender. To implement this, the function shown in Listing 3.2 is used as a template. In this function, the buffer sent from host side is just passed to the already provided function *A_PUTC(...)* which passes the charter to the UART. The enumeration was extended by *WMI_UART_WRITE_CMDID* and connected through the lookup table described before. Beside sending a character through UART this function also triggers a short pulse to one of the GPIOs. This feature was implemented to do the wire measurement described in Section 4.2.3.

The second extension is used to detect button presses. Reading the corresponding bit of the button is already possible through the register read commands from the kernel side, but it turns out that polling is to slow to get the short wake-up pulse. Therefore, the input register is polled in firmware and a flag is set if the GPIO is pulled low. Then, like before an additional function is written and connected to the command *WMI_BUTTON_PRESSED_CMDID*. If the host calls this command the state of the flag is returned and reset.

3.2.3 Driver

To use the new capabilities of the firmware described before, the ATH9K driver needs to be extended. The driver holds the counter part of the command facility used within the firmware. To execute commands on the firmware, the function *ath9k_wmi_cmd(...)* [5, `drivers/net/wireless/ath/ath9k/wmi.c`, Line 291] is used in the kernel driver. Like in the firmware, an identical enumeration [5, `drivers/net/wireless/ath/ath9k/wmi.h`, Line 81] is used to list all available commands. To use the new commands, this enumeration has been extended by *WMI_UART_WRITE_CMDID* and *WMI_BUTTON_PRESSED_CMDID* and two new function were added. The ATH9K driver registers these to the new callbacks of the *mac80211* which are described in Section 3.4.

3.2.4 Native Power Save Malfunction

The already build-in power save showed an erroneous behavior. As described in Section 2.2.1, the dynamic power save sets the WLAN adapter to sleep mode which works properly. But during sleep, the WLAN adapter should wake-up for receiving the beacons coming from the AP which then are passed to the *mac80211* implementation to check if there is the pending traffic bit set in the TIM. But at this point it fails, as the WLAN adapter does not wake-up for those beacons. As a comparison including the native power save is planned, a working native power save is needed. A simple fix can be found in [10].

This fix is verified by using the measurement system described afterwards. Figure 3.5 shows the erroneous behavior where the WLAN adapter does no wake-up correctly. It is noticeable that the WLAN adapter wakes-up after a while, but this happens because the beacon loss counter triggers after a duration of $7 * beacon\_interval$ [5, `/net/mac80211/mlme.c`, Line 63]. Furthermore there are small current peaks jumping up, this indicates the sending of data, again this is understandable as the *mac80211* tries to probe for the AP The measurement was taken with a beacon interval of 100 *ms*, as expected in the erroneous case it wakes-up after approximately $7 * 100\ ms = 700\ ms$. The fixed version on the other hand, shown in Figure 3.6, wakes up before every beacon within 100 *ms*. Furthermore it does not send anything, as it only needs to listen to the beacons.

[10] `https://github.com/doru91/linux-stable/commit/3eb6c30`

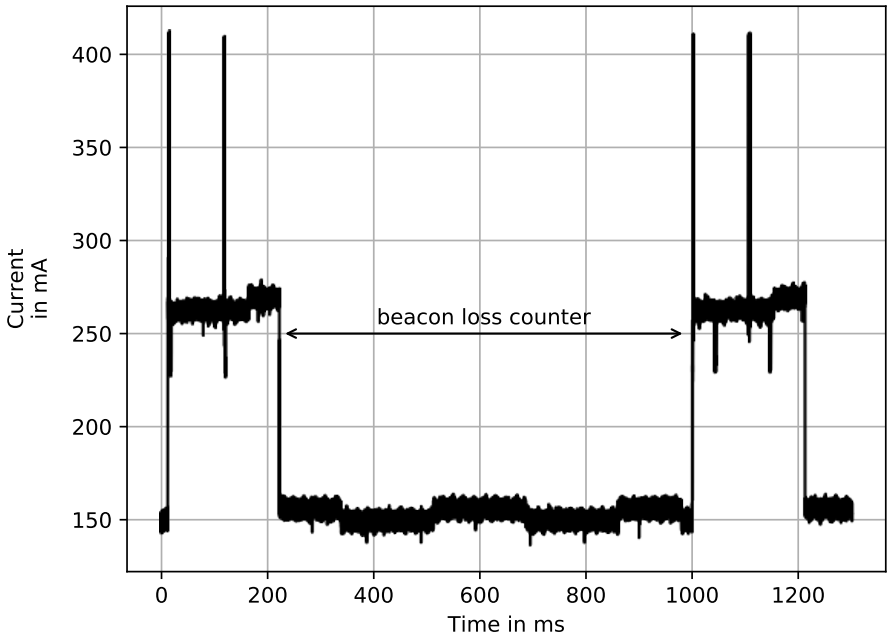


Figure 3.5 – Broken ATH9K power save

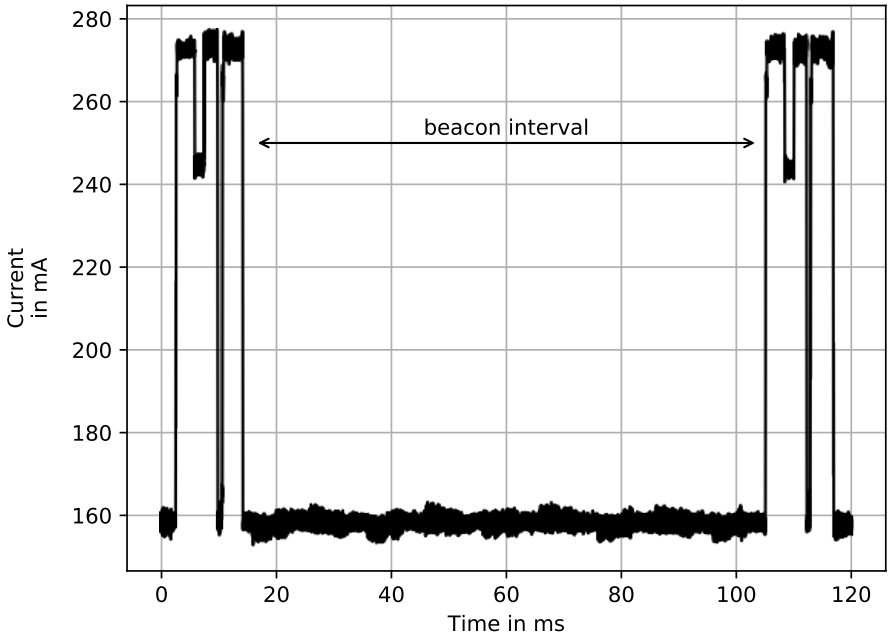


Figure 3.6 – Fixed ATH9K power save

3.3 SWuRx Modifications

The selective wake-up receiver only needed minor modifications in software. The necessary modification was to enable the selection of the wake-up pattern through the UART. This is done to allow the STA adapter to select the wake-up pattern. Like described in Section 3.1 again the software on the MSP430 reacts to hex symbols from 0 to *F* and sets the pattern in the shift register accordingly.

3.4 Modification of MAC80211 Implementation

As described in Section 2.2 the *mac80211* plays the main role in terms of power management on the AP as well as on the STA side. Therefore modifying the ATH9K driver is not enough. To realize the WuRx system it is necessary to consider the following modifications. From the AP point of view, the *mac80211* implementation needs to be extended by the capability of using the SWuRx sender described before, to send a wake-up request. The second modification addresses the reception of the wake-up signal on the STA side.

The modification starts by adding two more callback functions to the *mac80211* implementation listed in Listing 3.3. The callback function *send_SWuRx(...)* is used to pass a possible wake-up candidate down to the driver. Two arguments are passed to this function. Like all other functions, the first one is used to differentiated between different *mac80211* instances, as multiple WLAN adapters might be connected to the machine. The second argument is intended to transfer the AID of the STA which should be woken-up. As the AID is limited to a range of $1-2007$ [1] a 16 bit unsigned integer is sufficient. The second one, *get_wurx_state(...)*, is used to check for a possible received wake-up signal. A returned *True* signalizes the reception of a wake-up request.

To generate the correct wake-up signal, the transmission path of the *mac80211* described in Section 2.2 is used. To do so, the transmission path, at the position of the "tx handlers", has been extended by a function which checks if a packet is destined for a sleeping STA like in *ieee80211_tx_h_ps_buf(...)*. If this is the case, the new function *send_SWuRx(...)* is called with the AID according to the station information structure. To prevent multiple calls to that function it is locked afterwards, as otherwise every packet would trigger a new wake-up request. This lock is released after a timeout

```
1 void (*send_SWuRx)(struct ieee80211_hw *hw, u16 address);
2 bool (*get_wurx_state)(struct ieee80211_hw *hw);
```

Listing 3.3 – Appended callbacks to *mac80211*

in which the STA did wake-up to allow a resend of the wake-up signal. The lock is also released if the STA signalizes the sleep mode as this is the initial state. The new behavior is visualized in Figure 3.7.

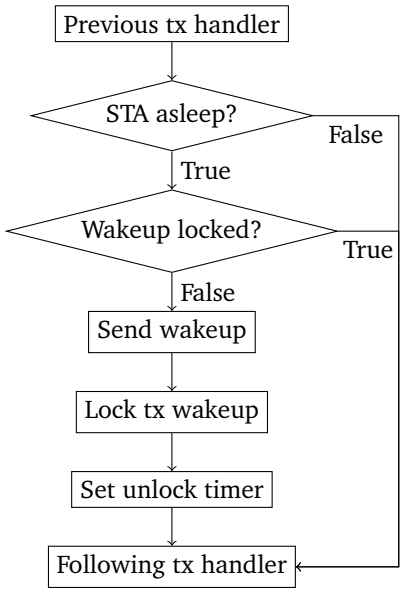


Figure 3.7 – Sending wake-up procedure on AP side

To receive a possible wake-up signal, a new function has been introduced which polls the *get_wurx_state* callback. This function gets enabled after the STA enters power save and polls until the *get_wurx_state* returns a *True*. Then, like implemented in the code for handling the TIM, a null frame is sent to the AP and the sleep timer is set. Figure 3.8 shows the described behavior. The shown visualization is similar to the power save flow chart described in Section 2.2. But there is one difference, instead of checking the TIMs in the beacons sent by the AP, the new implementation neglects this and just polls the new WuRx function to check if there is any traffic intended for the STA.

To control everything without recompiling the module, the *debugfs* is used. A new file is allocated for every WLAN adapter and a small interpreter was implemented behind it. It is used to control the WuRx system but also allows to disable some default behaviors like the beacon interval timeout which is necessary to use the WuRx system without getting disturbed by the received beacons.

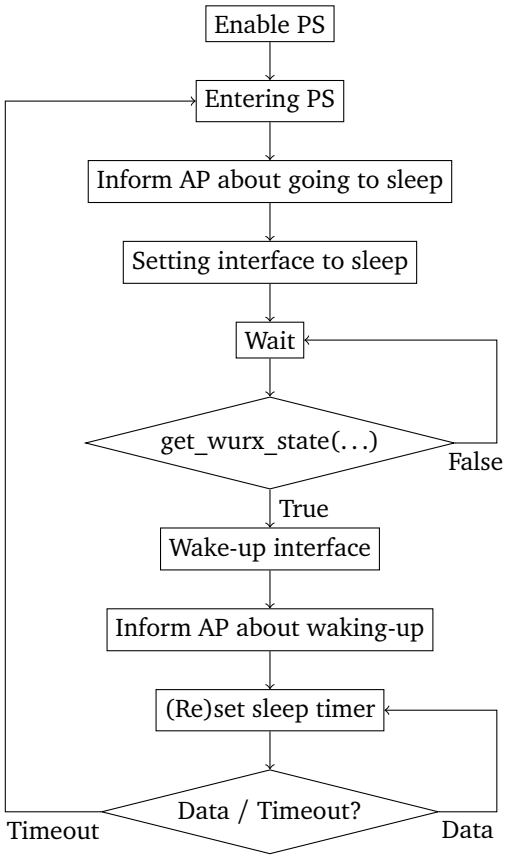


Figure 3.8 – Flowchart of WuRx power save mode on STA side

3.5 UDP Ping

To measure the delay from the AP to the STA in one direction, a program called UDP ping was written. The purpose of this program is to measure the delay from one interface to another by using an UDP packet in only one direction. To do so, a server side and client side, controllable through arguments were implemented into one program. On server side a socket is opened to listen to UDP packets. After opening the socket, the client can send UDP packets to that server. To measure the time between sending the packet and receiving it on the server side, the client adds a timestamp as payload to the packet. After the server has received the packet, it will calculate the difference between the current timestamp, after reception, and the timestamp carried by the packet. As clock source, the CLOCK_MONOTONIC, a continuously counting up timer since boot, is used. This can be done, as the server and client are running on the same machine.

Chapter 4

Evaluation

4.1 Measurement Setup

This chapter presents the measurement setup followed by the measurements itself and their results. The completed system is shown in Figure 4.1. The measurement setup consists of the following devices:

- A Rigol DS1054Z digital oscilloscope
- A current to voltage converter[11]
- Two WNDA3200 WLAN-sticks using the AR7010 connected to an AR9280.
- The selective wake-up receiver described in [2]
- The created SWuRx sender described in Section 3.1
- A computer with the modified kernel and firmware

To measure and calculate the energy of the STA within a specified time frame, the voltage as well as the current of the WLAN adapter is captured by an oscilloscope. To tap the voltage, the probe has been connected to the USB power supply lines. As the oscilloscope itself can only measure voltage a conversation between current and voltage is necessary. This can be done by using a resistor and solving the Equation (4.1).

$$V(t) = R * I(t) \tag{4.1}$$

Instead of a simple resistor the so called µCurrent[11] was used. This device also uses a shunt resistor, but has additional amplifiers built-in to amplify the voltage drop measured across the shunt. Measurements have shown that the noise is reduced by

[11] `https://www.eevblog.com/projects/ucurrent/` Accessed: April 2018

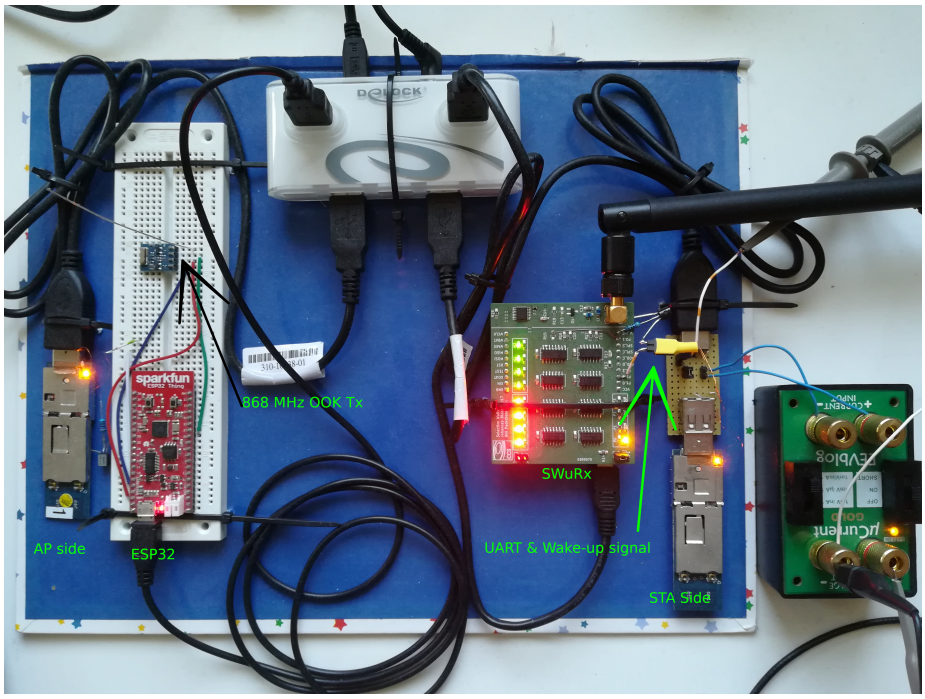


Figure 4.1 – Developed prototype.

using the µCurrent for amplification instead of using the internal amplifier of the oscilloscope. To download the data captured by the oscilloscope pyVISA[12] is used. It provides a simple interface to the NI-VISA[13] library. The oscilloscope is connected to the computer through USB and gets controlled through a measurement script written in python. A block diagram of the measurement setup on the STA side is shown in Figure 4.2. The oscilloscope measures the voltage V_{USB} from the USB and the voltage $V_{current}$ from the µCurrent. As the converter was configured to convert 1 mA into 1 mV the current evaluates to $I = V_{current}$.

To calculate the energy used by the STA within a specified timeframe Equation (4.2) is used. In python this is done by firstly multiplying the collected voltage and current point by point and calculating the area with the help of the numpy *trapz(...)* function with respect to time.

$$E = \int_0^T P(t)dt = \int_0^T V(t) * I(t)dt \tag{4.2}$$

[12] `https://pyvisa.readthedocs.io` Accessed: April 2018
[13] `http://www.ni.com/visa/` Accessed: April 2018

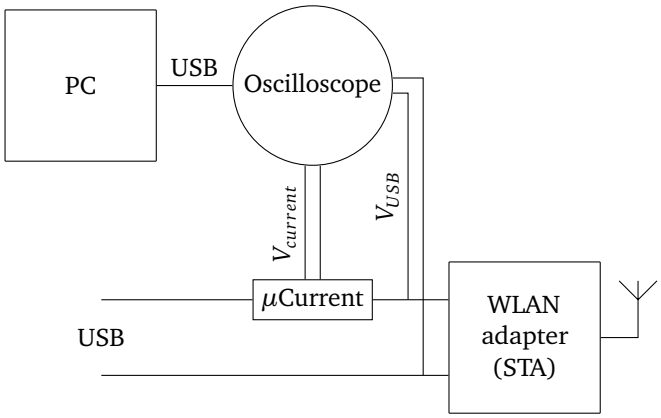


Figure 4.2 – Model of STA side measurement setup.

4.2 Measurements

The measurements are separated into two steps. In the first step reference and validation measurements were made. The reference measurements are necessary to measure initial current consumption. The current and energy measurements excludes the SWuRx as it draws nearly nothing in comparison to the WLAN adapter. The following currents are captured by collecting the current draw with the oscilloscope within 1 min, afterwards the mean was calculated. As the WNDA3200 has two chips the current consumption of the radio and the MCU are not clearly distinguishable from the outside. The best approach to get close to the radio chip current consumption is to disable the radio chip and measure the current consumed in that state. To do this, the WLAN adapter was disabled from the host side by using the Linux *ip* tool. After issuing *ip link <interface> down* the current consumption was reduced to 153.24 *mA*. This current now is assumed to be I_{AR7010}, therefore to get to the radio chip current Equation (4.3) is used. Figure 4.3 visualizes the assumed current draw model.

$$I_{AR9280} = I_{sum} - I_{AR7010} \tag{4.3}$$

According to this, the current used for the energy calculation will be I_{AR9280}. Thus, using Equation (4.2) and Equation (4.3), the consumed energy of the radio chip can be calculated by Equation (4.4) whereas $I_{sum} = V_{current}$. A good argument of separating the current consumption is that there exists mini PCI-E cards featuring the AR9280 radio chip where no AR7010 is needed as it gets directly connected to

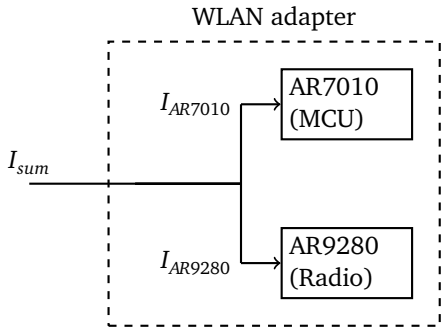


Figure 4.3 – Current draw model.

the PCI-E bus.

$$E = \int_0^T V_{USB}(t) * I_{AR9280}(t)dt = \int_0^T V_{USB}(t) * (I_{sum}(t) - I_{AR7010})dt \tag{4.4}$$

After enabling the adapter again, by issuing *ip link <interface> up*, the current consumption was increased to 159.05 *mA*. The next measurement which was done is the current consumption while the STA is connected to the AP without any power save mode and no data transfer. The current measured is about 271.86 *mA*. Then the STA is set to power save mode and the current is measured without waking up for the beacons. The current consumption in this state, called "network sleep" in the ATH9K source, is about 158.95 *mA*. This seems to be effectively the same current like in idle mode without a connection to an AP. Table 4.1 summarizes all measured currents in the different modes.

Mode	Current in mA
Interface down	153.24
Interface up	159.05
Connected to AP	271.86
Network sleep	158.95

Table 4.1 – Current consumption summary.

4.2.1 System Validation

After measuring the basic operating parameters, verification measurements were made to prove that the new WuRx system works as expected. To verify the system, the oscilloscope has been connected with all four channels to the following measurement points: One channel is used to measure the UART communication from the WLAN adapter which acts as an AP to the SWuRx sender described in Section 3.1. A

second channel is used to capture the signal between the ESP32 and the 868 MHz sender. Third, the output of the selective wake-up receiver is measured to capture the incoming wake-up signal. Finally, the current of the STA is probed to verify that it changes the power state upon receiving the wake-up signal.

Then, the test is done by setting the STA into power save mode with the new wake-up receiving implementation enabled. Afterwards, one unicast UDP packet was sent from the AP side to the STA side. After sending the UDP packet the AP adapter sends the AID of the STA through UART to the ESP32. Then, the ESP32 generates the signal and sends it out through the 868 MHz OOK sender. Afterwards, the SWuRx toggles its wake-up output and the STA changes its power state to active, which results in an increasing current consumption.

Figure 4.4 shows an overview of the all four signals. Due to a noisy signal of the current, the data was filtered using *lfilter* from the scipy library[14] to create more readable plots. The figure shows four subplots ordered from bottom to top with respect to the events which are occurring. The lowest one shows the UART communication from the AP to the ESP32 as described above. After a short time, the wake-up signal is generated according to the AID received before through UART which can be observed in the second subplot. As a result of the correct reception of the wake-up address, the SWuRx signalizes this to the STA through the GPIO by pulling it low, observable in the third subplot. As described in Section 3.4 the *mac80211* detects this through the new function and disables the sleep mode of the adapter which can be observed in the fourth subplot. Due to the coarse timescale in comparison to the high frequency of the UART and the generated wake-up signal, the signals are barely to observe. Therefore two sections are extracted from the same data. To easily evaluate the delay from those plots the abscissa, presenting the time, has been aligned to 0. Figure 4.5 gives a closer look to the UART and the beginning of the generated wake-up signal. It shows that after the UART communication is done – static high level –, it takes about 0.8 *ms* until the signal is generated. Figure 4.6 shows a closeup between the wake-up signal an the current consumption of the STA. It is observable that after the wake-up signal is received – falling edge of the wake signal – it takes about 8 *ms* until the WLAN adapter is woken-up – rising current consumption – by the new *mac80211* implementation. The capacitive behavior which is observable in the wake-up signal is introduced through a RC element which resets the counter of the SWuRx as described in Section 2.3.

[14] `https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.signal.lfilter.html`

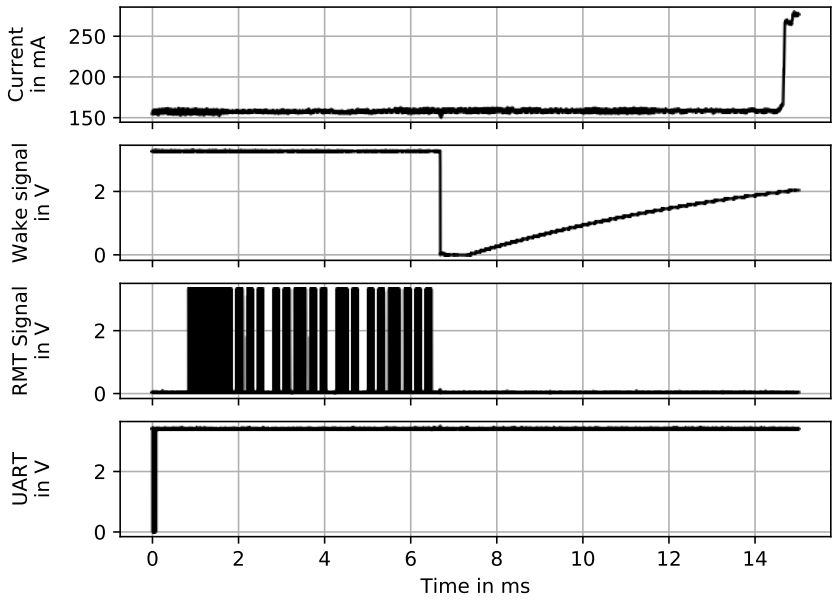


Figure 4.4 – System validation overview.

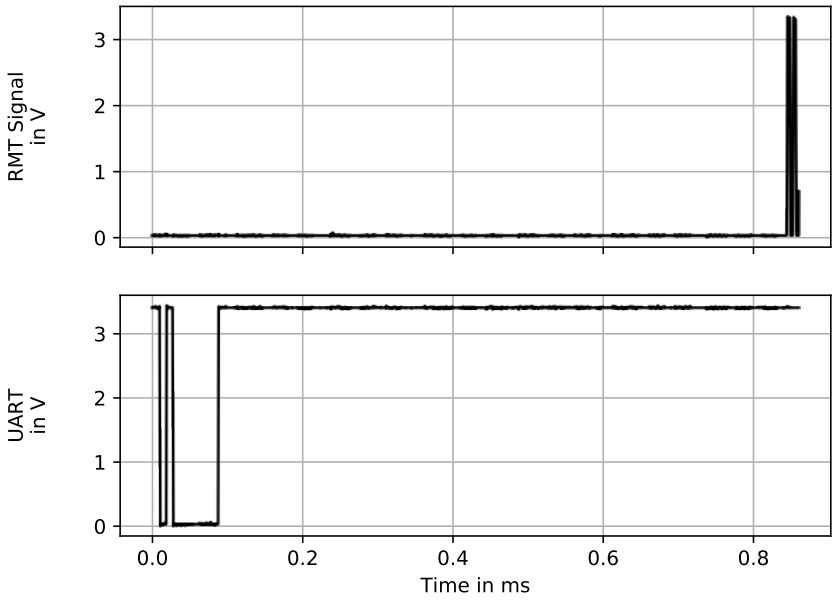


Figure 4.5 – UART to wake-up signal closeup.

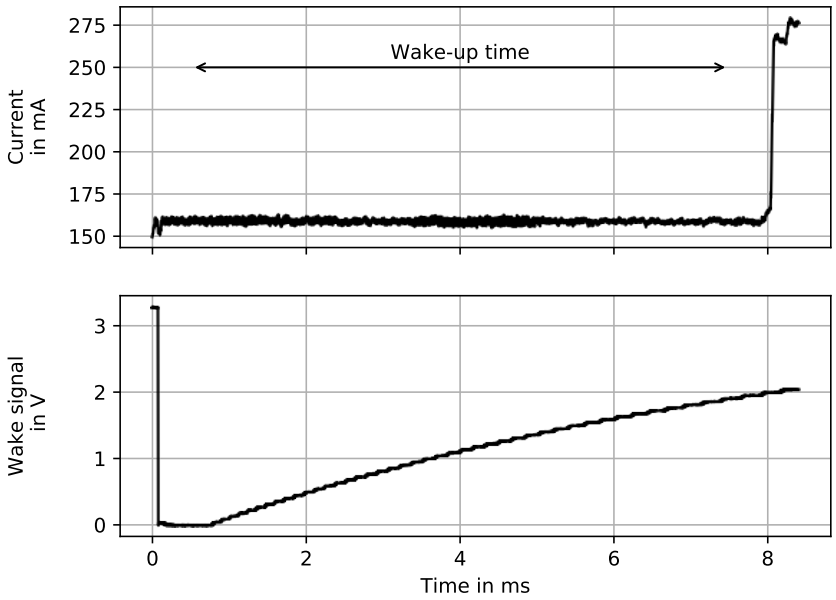


Figure 4.6 – Wake signal to wake-up closeup.

The next measurements focus on energy consumption, delay, and throughput. Therefore a parameter study regarding the beacon interval was made. The measurements are listed below:

- Idle measurement to compare the energy consumption between all three modes if no data is transferred
- Application layer packet delay measurement
- Goodput measurement with resetting the sleep timer on data and forcing the STA to sleep while traffic is flowing

4.2.2 Idle Energy Consumption

In the idle measurement the energy consumption of the STA while no data is transferred is measured, only necessary frame exchanges are handled like receiving the beacons in the native power save. The measurements were done within a time frame of 1 min. The measurement starts with a beacon interval of 50 *ms* and increases in a step size of 10 *ms* until the beacon interval reaches 100 *ms*, from there the step size is increased by 100 *ms* until 1000 *ms* is reached. The measurement was repeated 5 times for every beacon interval in every mode. The results of the measurements

are shown in Figure 4.7, all standard deviations are less than 1 J and therefore not shown.

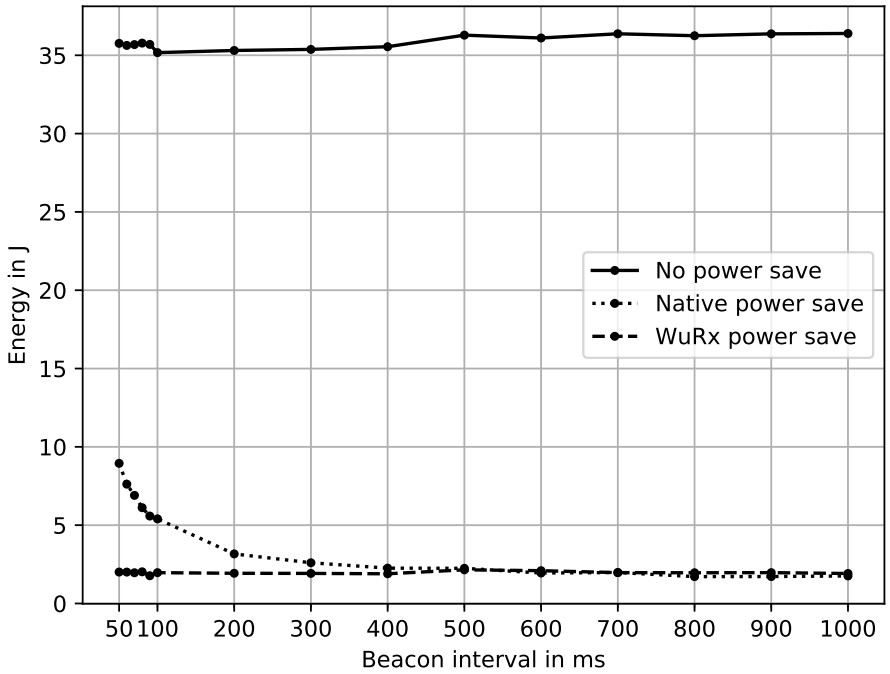


Figure 4.7 – Energy consumption while no data is transferred except necessary beacon reception in native power save mode.

A clear difference between all three modes is observable. As expected, the highest energy consumption is drawn if no power save is used. This occurs due to the idle listening of the receiver chain of the WLAN adapter. There is no dependency between the beacon interval and the energy consumption, as the STA stays awake the whole time.

The native power save mode shows the second largest energy consumption but in comparison to the native power save and WuRx power save it has a decreasing exponential tendency with an increasing beacon interval. This behavior can be explained with the fact that the STA has to wake-up to listen to the beacons to check the TIM as described in Section 2.2.1. With the increase of the beacon interval the STA has to wake-up less often within the time frame of 1 min, thus less energy is needed. As the exponential tendency is counterintuitive an example is used to show why this behavior occurs. Assuming that the beacons can only be sent in discrete positions like every 100 *ms* within a limited time interval, shown in Figure 4.8, and every beacon costs energy denoted as E, then changing the beacon interval to 200 *ms* would skip every second beacon. Therefore only 5 instead of 10 beacons would be in the row. Increasing it the interval to 300 *ms* would skip every third beacon. Now

only 4 beacons are sent. A continuation, up to 600 *ms*, of the series is shown in Table 4.2. Like observed in Figure 4.7 the decreasing series shows an exponential tendency.

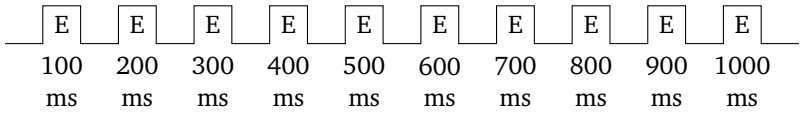


Figure 4.8 – Example to outline the exponential energy fall off.

Beacon Interval in ms	100	200	300	400	500	600
#Beacons	10	5	4	3	2	2

Table 4.2 – Beacon interval series.

The WuRx system is treated as a special case. As shown in Figure 3.8, with the new WuRx system, it is not necessary that the STA wakes-up for beacons. Therefore, it is assumed, that a STA does not need to listen to the WLAN beacons sent by the AP, instead the WuRx system may provide a separate beaconing system to detect if an AP is still in range. This is not realized within this thesis, but it is possible to extend it on the SWuRx with some hardware changing effort. Nevertheless, to simulate a system which does not depend on the WLAN beacons, the beacon wake-up was disabled by disabling the fix described in Section 3.2.4. Furthermore the beacon timeout has been disabled by resetting the timer and circumvent the error handling [5, `mlme.c`, Line 4007]. Although the beacon wake-up was disabled, the connection was still stable but the out of range detection was disabled with it. As the STA stays asleep the whole time, the lowest energy consumption is achieved and the system is independent from the beacon interval.

4.2.3 Application Level Delay Measurement

In the next measurement, the impact of the beacon interval to the application level delay of packets is shown. To remove the delay which is taken from the STA back to the AP while using a normal ping, the UDPPing program described in Section 3.5 was used.

The measurement was done by sending 115 UDP packets within an interval of 500 *ms*. Then, this measurement has been repeated 20 times to improve the reliability of the energy measurement. Compared to the idle measurement, this measurement has one more curve to show. The fourth curve results from using a wire connected from the AP side to the STA side to eliminate the delay introduced by the SWuRx.

The measurement was started at 100 *ms* and ended at 1000 *ms* beacon interval in a step size of 100 *ms*. Figure 4.9 shows the mean of the delay of 115 UDP packets. The errorbars represents the standard deviation. One thing to note is that the error bar is only enabled at a standard deviation threshold above 6 *ms*. The results show that by using no power save, the lowest delay is achieved. This is followed by the wake-up system using a wire between the AP and the STA. Afterwards, if the WuRx system is used the normal way, with the SWuRx, an additional delay is introduced. The contributor of this delay can be observed in Figure 4.4 from the system validation overview. It is the time from sending the AID to the ESP32 to the end of the generated wake-up signal. But again, compared to the native power save, this small delay can be neglected.

The delay of the native power save shows an increasing tendency. The source of this behavior can be traced back to the dependency of the native power save to the beacon interval. As already described in Section 2.2.1, the STA must wait for the next beacon to detect if packets are pending and afterwards it has to inform the AP to release the packets by sending a null frame with the power save bit unset. Therefore, if the beacon interval is increased, the worst case waiting time also increases. A better insight of the distribution is given by the ECDFs shown in Figure 4.10 and Figure 4.11 for 100 *ms* and 1000 *ms* beacon interval. The native power save is separated from the others to prevent the huge scaling to be applied to them. One thing to notice is, that even at the 1000 *ms* beacon interval using the native power save, some packets arrive quiet quickly. This phenomenon occurs especially at higher beacon intervals because the application level is not synchronized with the beacon interval in any way, therefore it may happen that some packets are sent just before the next beacon arrives. Furthermore, the relation in time between the sending of a UDP packet and the beacon interval also drifts, due to a lack of real time capabilities of Linux.

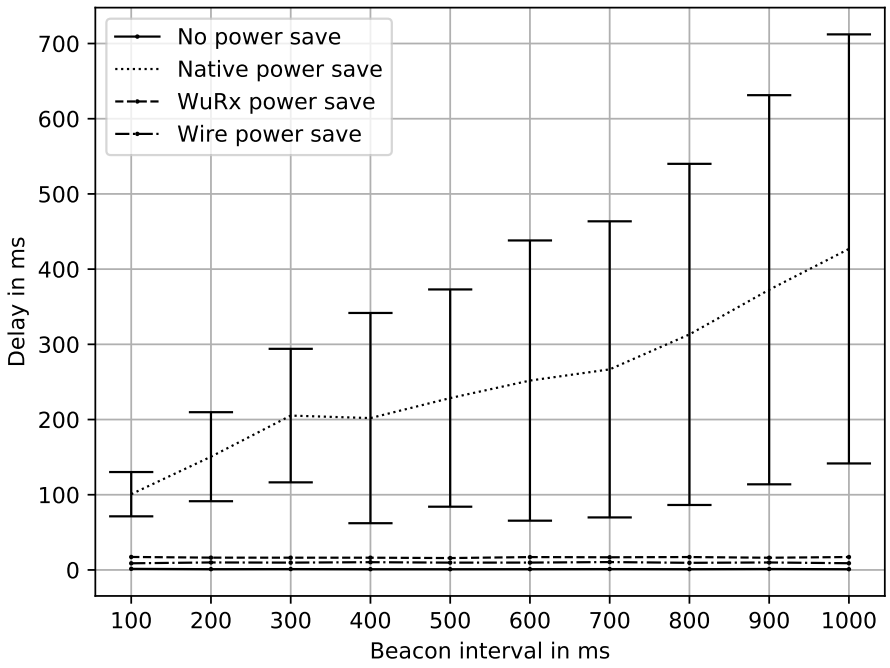


Figure 4.9 – Results of delay measurement with 115 samples. Error bars represents standard deviation. A standard deviation less than 6 *ms* is neglected.

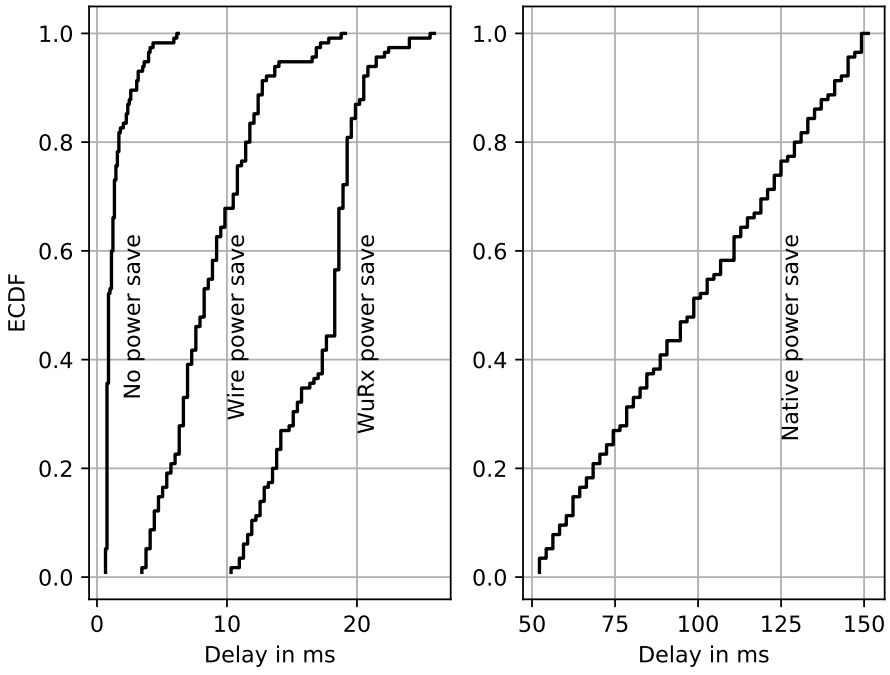


Figure 4.10 – Results of delay measurement at 100 *ms* beacon interval presented as ECDF.

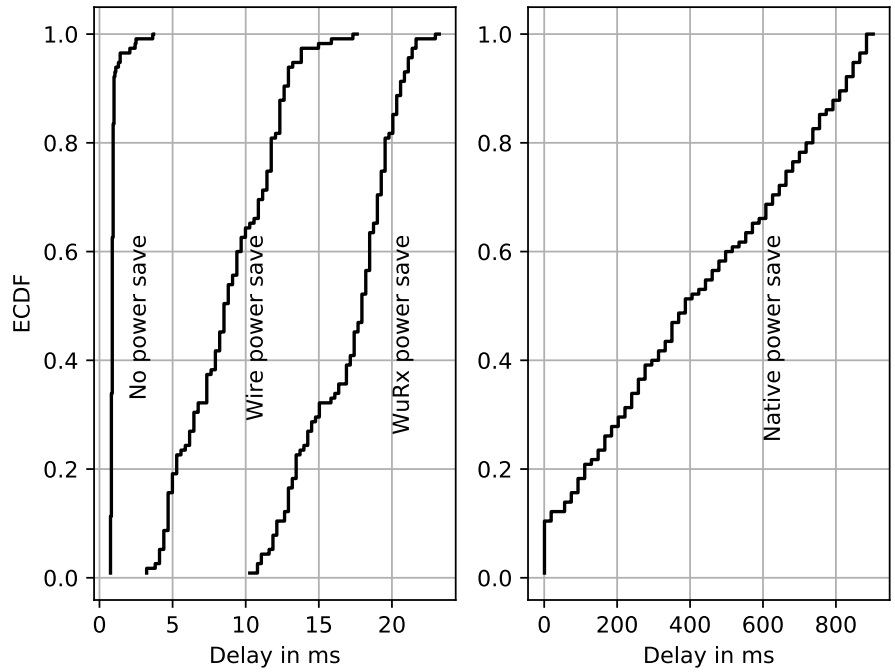


Figure 4.11 – Results of delay measurement at 1000 *ms* beacon interval presented as ECDF.

Figure 4.12 shows the energy consumption of the delay measurement. The standard deviation is disabled if it is less than 1 J. The highest energy consumption is used if no power save is enabled. Like in the idle measurement, the energy consumption is about 36 J. The WuRx and wire power save have nearly the same energy consumption, which can be reasoned by the fact that in both situations the STA will wake-up for every UDP packet which comes through, regardless of the few milliseconds saved by the wire. The native power save instead has a decreasing tendency. This happens because the main contributor to the energy consumption is idle listening, which in this context is introduced by the timeout of the dynamic power save. As the beacon interval increases, there are less wake-ups, accordingly the energy consumption decreases. Nevertheless all packets go through as they are buffered on the AP during sleep. For lower beacon intervals, the energy consumption of the native power save is higher than the energy consumption of the WuRx system. This behavior occurs because the oscilloscope always recorded a time frame of 1 min, but the delay measurement does not occupy it the whole time. A few seconds were left in which during the native power save beacon wake-ups were recorded. The outlier at 600 *ms* beacon interval is the result of one measurement point, therefore it can be neglected.

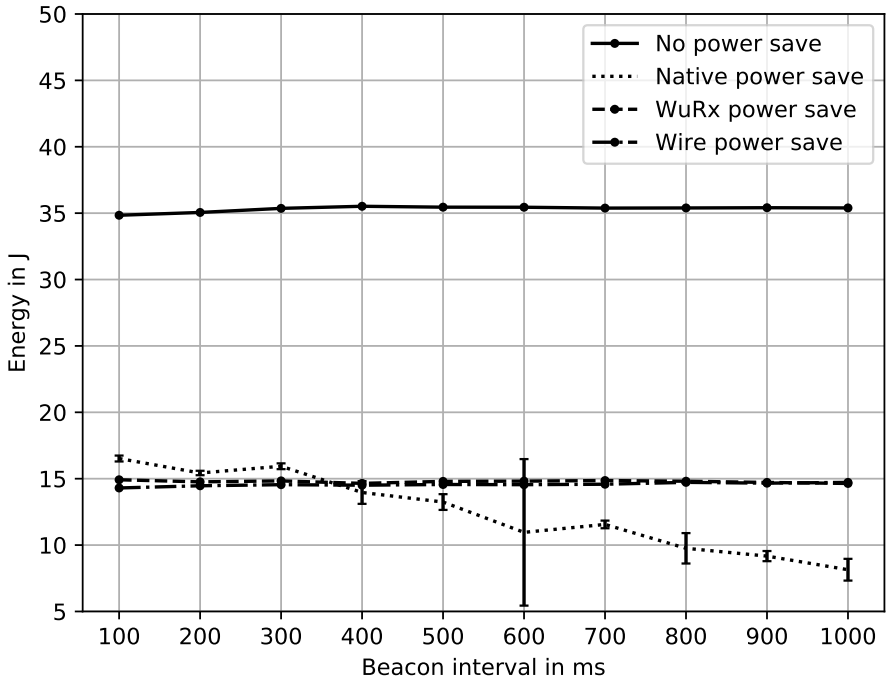


Figure 4.12 – Energy consumption of delay measurement. Standard deviation disabled if less than 1 J.

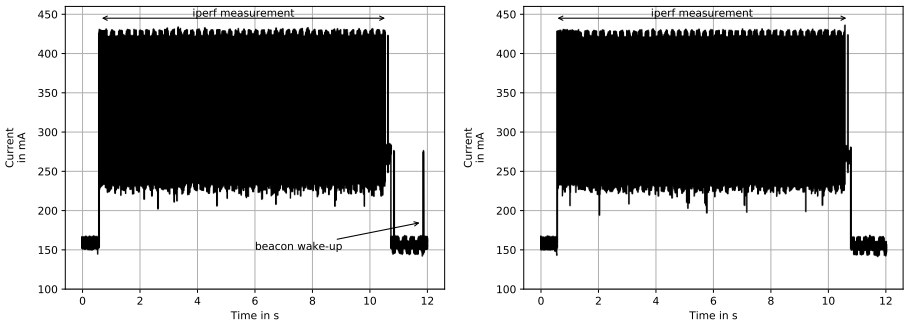
4.2.4 Goodput Measurement

In the goodput study, it is shown how the new WuRx system behaves at higher goodputs. This study is created by using the *iperf* utility. All studys are done in UDP mode and a target goodput of 5 MiB has been set. The time frame used for the current measurement is 12 s as within this time the *iperf* measurement finishes. This also means that the energy consumption always includes the full 12 s even if the *iperf* measurements ends earlier. The measurements were repeated 20 times for every beacon interval and every mode. Two cases were taken into account. In the first case, the normal behavior of the *mac80211* implementation regarding the sleep timer is kept. In the second case, the implementation is changed that way, that the sleep timer triggers even if there is still data flowing.

4.2.4.1 No Sleep on Data

As described in Section 2.2.1 and shown in Figure 2.6 the implemented behavior in the *mac80211* is to reset the sleep timer if data is flowing. According to this, it is not expected to save much energy while using the native power save or the WuRx power save, as a goodput measurement will lead to a continuous reset of the sleep timer. Figure 4.13a shows the current consumption during the measurement within

the native power save. It is starting with a low current consumption followed by a block of acknowledgments which could not be clearly observed due to the coarse timescale. The acknowledgments were sent while the STA receives the packets from *iperf* during the measurement. As expected, the WLAN adapter is not going to sleep until *iperf* is done. Afterwards, the current consumption is low again except small peaks which occur as a result of the beacon wake-ups necessary in the native power save mode. The same applies for the WuRx power save with the exception that it is not necessary to receive beacons. Figure 4.13b shows the *iperf* measurement in WuRx power save mode.



(a) Native power save **(b)** WuRx power save

Figure 4.13 – Current consumption during goodput measurement for native power save and WuRx power save at 1000 *ms* beacon interval.

Figure 4.14 shows the goodput for no power save, native power save and WuRx power save. Except a small artefact on the native power save, the goodput is equal between all three modes as expected. For some reason *iperf* calculates a higher goodput on the server side while using native power save. One assumption is, that the first UDP packet, which also carries measurement configurations for the server side, resides with the other packets in the buffer and therefore the server side calculates a wrong receiver side goodput. This phenomenon was not further investigated. Instead, the sender side was rechecked to assure that in all three modes the same goodput was sent.

In Figure 4.15 the results of the energy consumption measurements are shown. As expected, the energy consumption between the native power save and the WuRx power save are nearly equal. As the *iperf* measurement does not take the full 12 *s*, the calculated energy includes a few seconds of idle energy consumption. Thus, the native power save shows a decreasing energy consumption with a rising beacon interval due to the captured beacon wake-ups. One unexpected behavior can be observed at the point where the native power save energy consumption falls below the WuRx power save. Like shown in Figure 4.13, the behavior of both measurements

are the same. The expectation was to see a behavior like in the idle measurement in which the native power save approaches the WuRx power save as a few beacon wake-ups are captured before and after the *iperf* measurement. One explanation could be a changed temperature, which might have offset the WuRx energy consumption, as the measurements took hours.

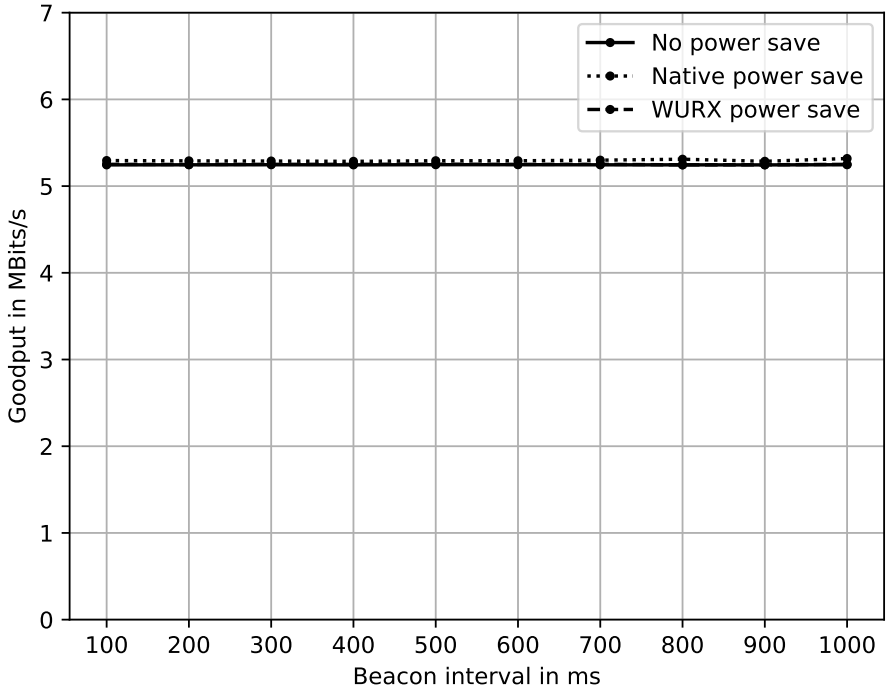


Figure 4.14 – Goodput measurement without forcing the STA going to sleep while data is flowing.

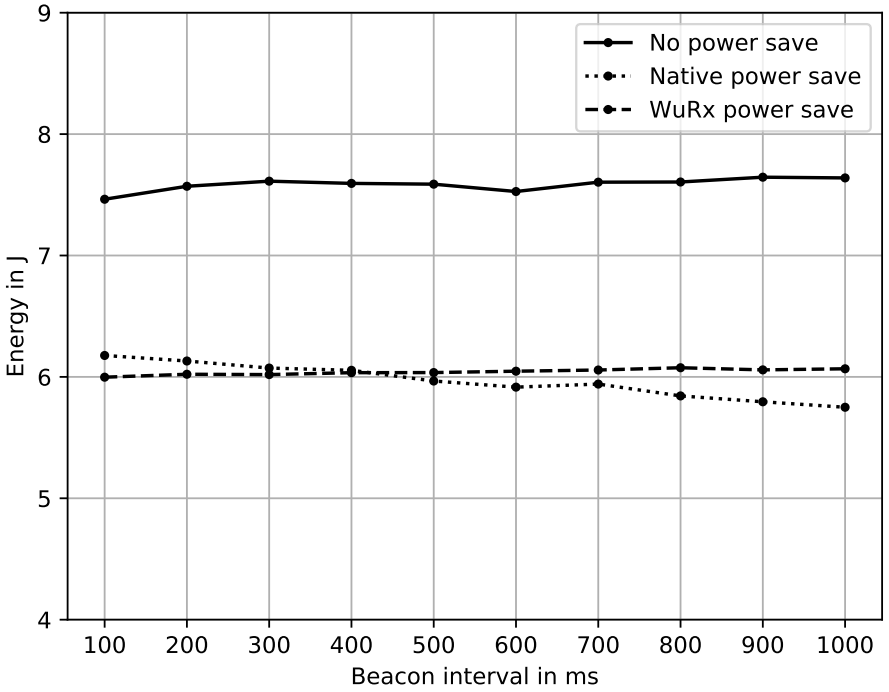


Figure 4.15 – Energy consumption of goodput measurement without forcing STA to sleep. A standard deviation less than 0.5 J is ignored.

4.2.4.2 Forced Sleep

The same measurement has been done again, but this time the sleep timeout is not reset by flowing traffic and therefore the STA will go to sleep even if data is flowing. Conducting this measurement seems to be useless at the first glance, but forcing the system to sleep while data is still flowing at high speed can simulate the system behavior during a beginning bursty traffic while the STA is asleep. Figure 4.16 shows the goodput measured by *iperf*. A standard deviation below 0.1 $MBit$ is now shown.

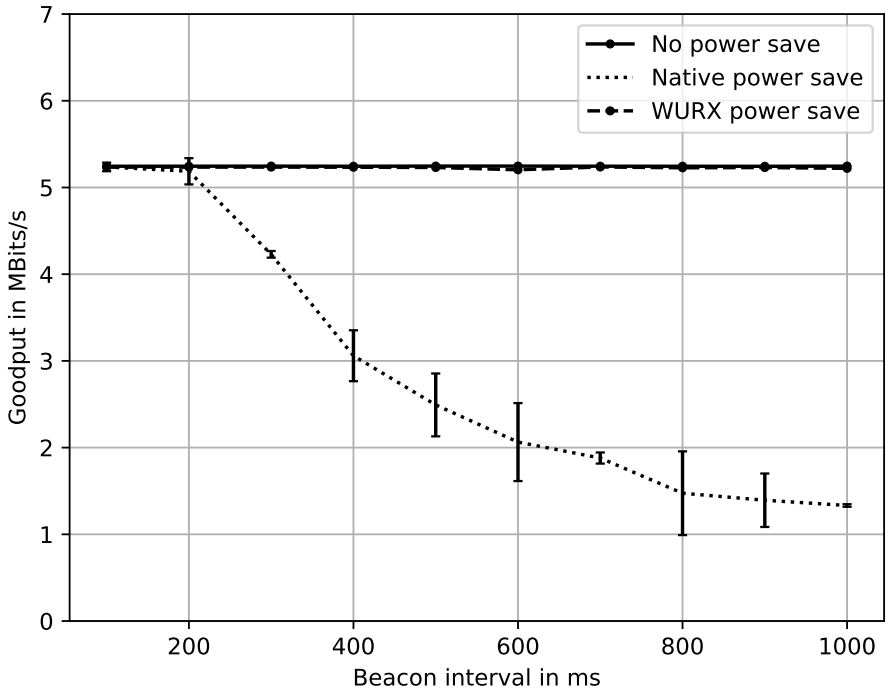


Figure 4.16 – Goodput measurement with STA forced to sleep. A Standard deviation less than 0.1 $MBit$ is not shown.

The results compare no power save, native power save and the new WuRx power save. It should be noted that the no power save base data is taken from the measurement before as in no power save mode the sleep timer has no influence to the system. The results clearly show the benefits of the WuRx power save in comparison to the native power save. While the WuRx power save achieves the same goodput like using no power save, the goodput of the native power save clearly decreases with an increasing beacon interval. This behavior can be explained by the power save technique described in Section 2.2.1. After the STA has notified the AP, it starts to buffer packets for the STA. As in this measurement the system is forced to go to sleep while data is flowing, a lot of packets arrives at the AP. This leads

to dropped packets as the buffer has a limited size. In case of the *mac80211* this limit is set to 64 [5, `/net/mac80211/sta_info.h`, Line 653] frames. Therefore the goodput depends on the wake-up time which is achieved by the systems. As the native power save needs the beacons to detect if there is traffic pending, the wake-up time depends on the beacon interval. Accordingly, a high beacon interval leads to a high wake-up time where packets get lost on the AP due to the limited buffer. Taking a look on the energy consumption shown in Figure 4.17, an artefact can be seen at 100 *ms* and 200 *ms* beacon interval. It turns out, that using the sleep timer modification leads to problems informing the AP about the power state of the STA in the 100 *ms* and 200 *ms* interval. Hence the STA enters sleep mode while the AP records it as awake. At the first glance, it should be clear that all the data would get lost during this phase as the STA is asleep and the AP just sends all the data to it. But this is not the case which is observable in Figure 4.16. The reason for that can be found in the beacon wake-up in which, besides the beacons, also data can be received. As the STA stays awake way shorter than it would if a beacon notifies about pending packets, the energy consumption is much lower. This erroneous behavior indicates that it should be possible to transfer 5 MiB with even less energy consumption. Nevertheless, as this is not the default behavior, the more meaningful observations start at 300 *ms* as at this point the AP gets correctly informed about the current power save mode of the STA and therefore stops the transmission as expected in the normal system. The energy consumption decreases with the increase of the beacon interval as the STA stays asleep for longer periods at the cost of packet loss and therefore a decreasing goodput. Figure 4.18 shows the current consumption of the STA while the native power save measurement is done with a beacon interval of 1000 *ms*. The figure clearly shows the wake-ups for the beacons at the beginning and the end of the time as well as the *iperf* measurement in between. The high peaks shows the transmission of the acknowledgments for the packets coming from the AP. Furthermore, the expected beacon interval of 1000 *ms* is observable.

In contrast to the native power save, the WuRx power save performs much better regarding the goodput. This is the result of a fast wake-up after the STA is forced to sleep mode, it can immediately be woken-up again by the AP through the WuRx system. It is not possible to save energy anymore, but this is not the intend in this scenario as it is necessary to get as most packets through as possible. Figure 4.19 shows the current consumption during the measurement with *iperf*. Like in Figure 4.18 the STA is forced to go to sleep during the transfer of data, observable by the peaks going down to about 150 *mA*, but instead of staying asleep for a maximum of the beacon interval, the STA is directly woken-up again and data can continue to flow.

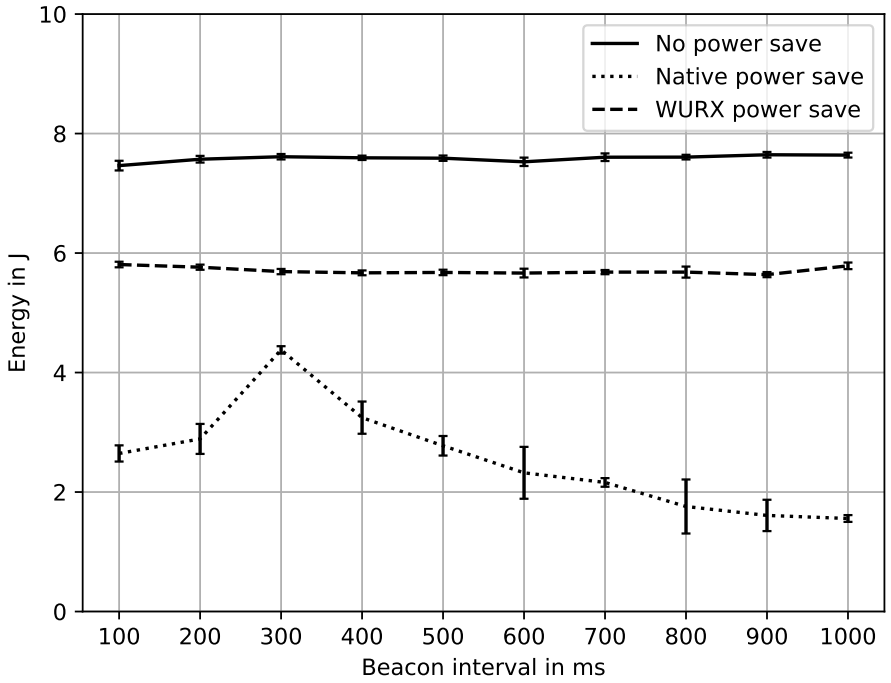


Figure 4.17 – Goodput energy consumption with forced sleep.

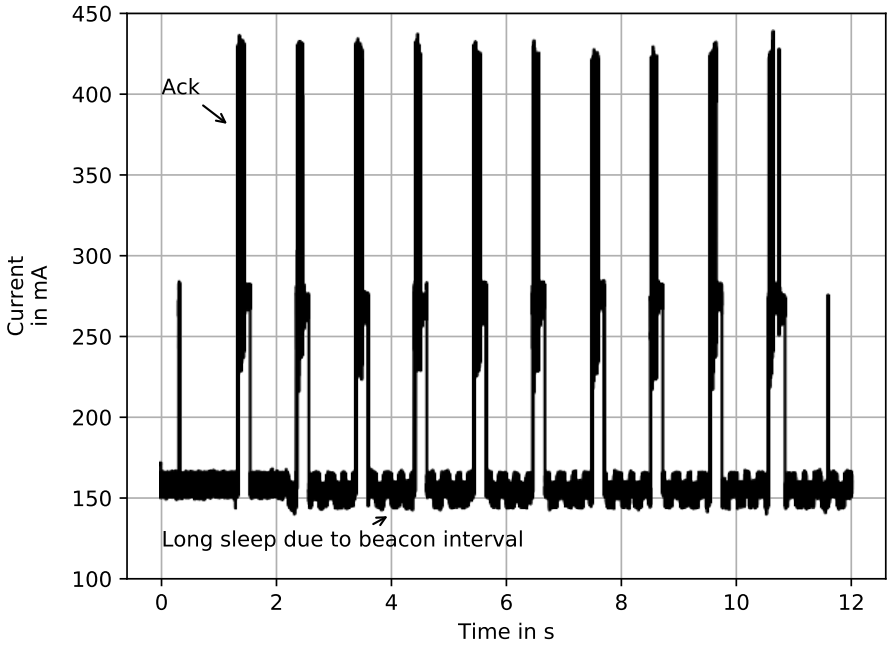


Figure 4.18 – Current consumption during goodput measurement with forced sleep at 1000 *ms* beacon interval for native power save.

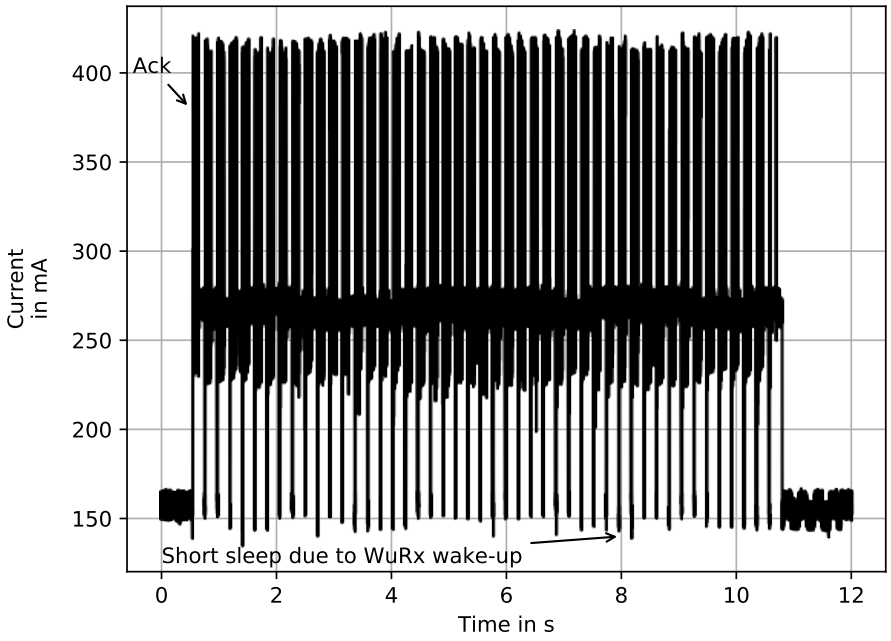


Figure 4.19 – Current consumption of the WuRx system during goodput measurement with forced sleep at 1000 *ms* beacon interval.

Chapter 5

Conclusion

In this thesis I could show a way how to extend an already existing WLAN adapter with a wake-up capability. To do so, it was necessary to do multiple modifications on hard- and software side, in firmware and the Linux kernel. After the system validity was proved by reference measurements, the system was characterized by energy consumption, delay and goodput.

The first measurement was the idle measurement in which no data was send or received with exception for the beacons in the native power save mode. It is clearly observable that the energy consumption of the no power save is the highest while the energy consumption of the WuRx power save is the lowest. In between the native power save consumes the second largest amount of energy, depending on the beacon interval. But assuming a beacon interval of 100 *ms* which for most deployed WLAN environments is the default configuration, the native power save consumes 5.3 J, due to the wake-ups to receive the beacons, and the WuRx power save 1.9 J. Therefore the WuRx power save saves up to 64 % in comparison to the native power save. Like other researchers have already shown, again this measurement emphasize that idle listening, this also includes listening to the beacons, is the main contributor to the overall energy consumption of WLAN.

The delay measurements also shows the benefits of the WuRx system in comparison to the native power save. While the packet delay in the native power save is bounded to the beacon interval, the WuRx system provides a low and stable latency. An additional measurement, with a wire connected between the AP and STA, has shown that a WuRx system should be designed to be as fast as possible to reduce the delay more. But even with the WuRx system shown in this thesis, a pretty low, about 20 *ms*, and a stable delay was achieved especially compared to the native power save.

The last measurement has shown how the system reacts to bursty traffic. The goodput measurement by default would not be very interesting at all, as after waking-

up the STA stays awake until no data is flowing anymore. After disabling this behavior, the goodput gets bounded to the wake-up time as it becomes necessary to wake-up as fast as possible if data is still flowing. The reason for that is the limited buffer size of the AP which gets quickly overflowed if the STA does not wake-up to receive them. Again, the measurements have shown that the WuRx system reacts faster as the AP can just request a wake-up at any time. This indicates a robust behavior against bursty traffic as the STA can be woken up directly after the first packet arrived, without waiting for the next beacon which might be too late.

Summing up, I could show that extending a WLAN adapter including the firmware and the Linux kernel by WuRx capabilities is possible with a moderate effort. Furthermore the thesis shows that the goals of the task group of IEEE 802.11ba want to address are reachable even with an already existing WLAN adapter. The energy consumption is reduced to a minimum, specially if no data is transferred. IoT sensors, like already proposed in some presentations within the task group, could benefit a lot as most of these devices does not need to communicate very often. Using the native power save instead is not a good solution for this type of devices, as they need to wake-up for the beacons to determine if traffic is pending. It is possible to ignore beacons or increase the beacon interval drastically but this would lead to a higher delay. Assuming an IoT actor like a lamp would react slowly on switch requests. This problem is also solved as shown in the delay measurements where a stable delay of about 20 *ms* was achieved. The goodput measurement, with forced sleep, has shown that bursty traffic while the STA is asleep is no problem for the WuRx power save, as the AP can request a wake-up directly which addresses the problem of dropped packets due to a limited buffer size on the AP side.

5.1 Future work

Currently the wake-up prototype can be denoted as an off the band wake-up radio system as two different frequencies are used, 2.4 GHz for data transfer and 868 MHz for the wake-up signal. This comes with the drawback of an unnecessary hardware effort, in this case on the sender side. In the electronic letter [20] they have shown, that it is possible to control an AS3933 through an AP by using correctly timed broadcast packets. To realize this, it is necessary to disable the CCA. One research question would be how the surrounded network environment is disturbed by the wake-up signal generated this way. Furthermore it would be interesting to investigate if the distances between the wake-up signal and the data signal are somehow equivalent or completely off. As detecting if an AP is still in range is an important feature, clarifying this question would influence the way how to do it. Currently this is done through the beacons sent out by the AP using the default WLAN radio. The

energy measurements have shown that waking-up for those beacons costs a lot of energy. Thus, as already mentioned in the evaluations chapter, one idea would be to replace the WLAN beacons by beacons which are sent through the WuRx system. To implement this, it is necessary to differentiate between a wake-up request and a beacon on the WuRx side.

As maintaining the connection between a STA and an AP also comes along with key exchange and furthermore refreshing those keys, while using WPA2, investigating how to exchange those keys more energy efficient using the new wake-up system would be necessary.

Another interesting topic to investigate would be to transfer this system to an already existing IoT chip. The ESP32, which in this thesis is only used for generating the wake-up signal, is an example for such an IoT chip. Compared to the ATH9K, the ESP32 code regarding the WLAN low level control is delivered as a binary blob and therefore not used within this thesis. But this matter might change in the future.

List of Abbreviations

AID	association ID
AP	access point
API	application programming interface
BSS	basic service set
CCA	clear channel assessment
DTIM	delivery traffic indication message
GPIO	general-purpose input/output
IEEE	Institute of Electrical and Electronics Engineers
IoT	internet of things
LED	light emitting diode
LNA	low noise amplifier
MAC	media access control
MCU	microcontroller unit
OOK	On-Off-Keying
PCB	printed circuit board
PCI-E	Peripheral Component Interconnect Express
PDA	Personal Digital Assistant
PDR	packet delivery ratio
SDR	software defined radio
SPI	Serial Peripheral Interface
STA	station
SWuRx	selective wake-up receiver
TIM	traffic indication map
UART	universal asynchronous receiver transmitter
ULP	ultra low power
USB	Universal Serial Bus
VOIP	Voice over IP
WEXT	wireless-extensions
WLAN	wireless LAN
WPA2	Wi-Fi Protected Access 2

WSN	Wireless Sensor Network
WuRx	wake-up receiver

List of Figures

List of Tables

Bibliography

[1] IEEE, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," IEEE, Std 802.11-2012, 2012.

[2] J. Blobel, J. Krasemann, and F. Dressler, "An Architecture for Sender-based Addressing for Selective Sensor Network Wake-Up Receivers," in *17th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2016)*, Coimbra, Portugal: IEEE, Jun. 2016. DOI: 10.1109/WoWMoM.2016.7523516.

[3] Status of Project IEEE 802.11ba, Std, http://www.ieee802.org/11/Reports/tgba_update.htm Accessed: April 2018.

[4] ATH9K Firmware, https://github.com/qca/open-ath9k-htc-firmware/tree/1.4.0/ Accessed: April 2018.

[5] Linux kernel Tag v4.14.4, https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git/tag/?h=v4.14.4 Accessed: April 2018.

[6] "AS3933 - 3D Low Frequency Wakeup Receiver," ams AG, Tech. Rep., 2015, http://ams.com/eng/Products/Wireless-Connectivity/Wireless-Sensor-Connectivity/AS3933 Accessed: April 2018.

[7] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, Aug. 2002. DOI: 10.1109/MCOM.2002.1024422.

[8] L. Gu and J. A. Stankovic, "Radio-triggered wake-up for wireless sensor networks," *Real-time Systems*, vol. 29, pp. 157–182, 2005.

[9] R. C. Carrano, D. Passos, L. C. S. Magalhaes, and C. V. N. Albuquerque, "Survey and Taxonomy of Duty Cycling Mechanisms in Wireless Sensor Networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 181–194, Jun. 2014. DOI: 10.1109/SURV.2013.052213.00116.

[10] R. Piyare, A. L. Murphy, C. Kiraly, P. Tosato, and D. Brunelli, "Ultra Low Power Wake-Up Radios: A Hardware and Networking Survey," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2117–2157, Jul. 2017. DOI: 10.1109/COMST.2017.2728092.

[11] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," in *2nd ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, Baltimore, MD: ACM, 2004, pp. 95–107. DOI: 10.1145/1031495.1031508.

[12] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: a Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks," in *4th ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, CO: ACM, Nov. 2006, pp. 307–320. DOI: 10.1145/1182807.1182838.

[13] J. Oller, I. Demirkol, J. Casademont, J. Paradells, G. U. Gamm, and L. Reindl, "Has Time Come to Switch From Duty-Cycled MAC Protocols to Wake-Up Radio for Wireless Sensor Networks?" *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 674–687, 2015. DOI: 10.1109/TNET.2014.2387314.

[14] J. Oller, I. Demirkol, J. Paradells, J. Casademont, and W. Heinzelman, "Time-Knocking: A novel addressing mechanism for wake-up receivers," in *8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2012)*, Barcelona, Spain: IEEE, Oct. 2012, pp. 268–275. DOI: 10.1109/WiMOB.2012.6379086.

[15] X. Perez-Costa and D. Camps-Mur, "APSM: bounding the downlink delay for 802.11 power save mode," in *IEEE International Conference on Communications*, vol. 5, Seoul, South Korea: IEEE, May 2005, pp. 3616–3622. DOI: 10.1109/ICC.2005.1495091.

[16] E. Shih, P. Bahl, and M. J. Sinclair, "Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices," in *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '02, Atlanta, GA, USA: ACM, Sep. 2002, pp. 160–171. DOI: 10.1145/570645.570666.

[17] S. Tozlu, "Feasibility of Wi-Fi enabled sensors for Internet of Things," in *2011 7th International Wireless Communications and Mobile Computing Conference*, Istanbul, Turkey: IEEE, Jul. 2011, pp. 291–296. DOI: 10.1109/IWCMC.2011.5982548.

[18] I. Haratcherev, M. Fiorito, and C. Balageas, "Low-Power Sleep Mode and Out-Of-Band Wake-Up for Indoor Access Points," in *2009 IEEE Globecom Workshops*, Honolulu, USA: IEEE, Nov. 2009, pp. 1–6. DOI: 10.1109/GLOCOMW.2009.5360732.

[19] L. M. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, vol. 3, Anchorage, USA: IEEE, Apr. 2001, pp. 1548–1557. DOI: 10.1109/INFCOM.2001.916651.

[20] J. Oller, E. Garcia, E. Lopez, I. Demirkol, J. Casademont, J. Paradells, U. Gamm, and L. Reindl, "IEEE 802.11-enabled wake-up radio system: design and performance evaluation," *Electronics Letters*, vol. 50, no. 20, pp. 1484–1486, Sep. 2014. DOI: 10.1049/el.2014.2468.