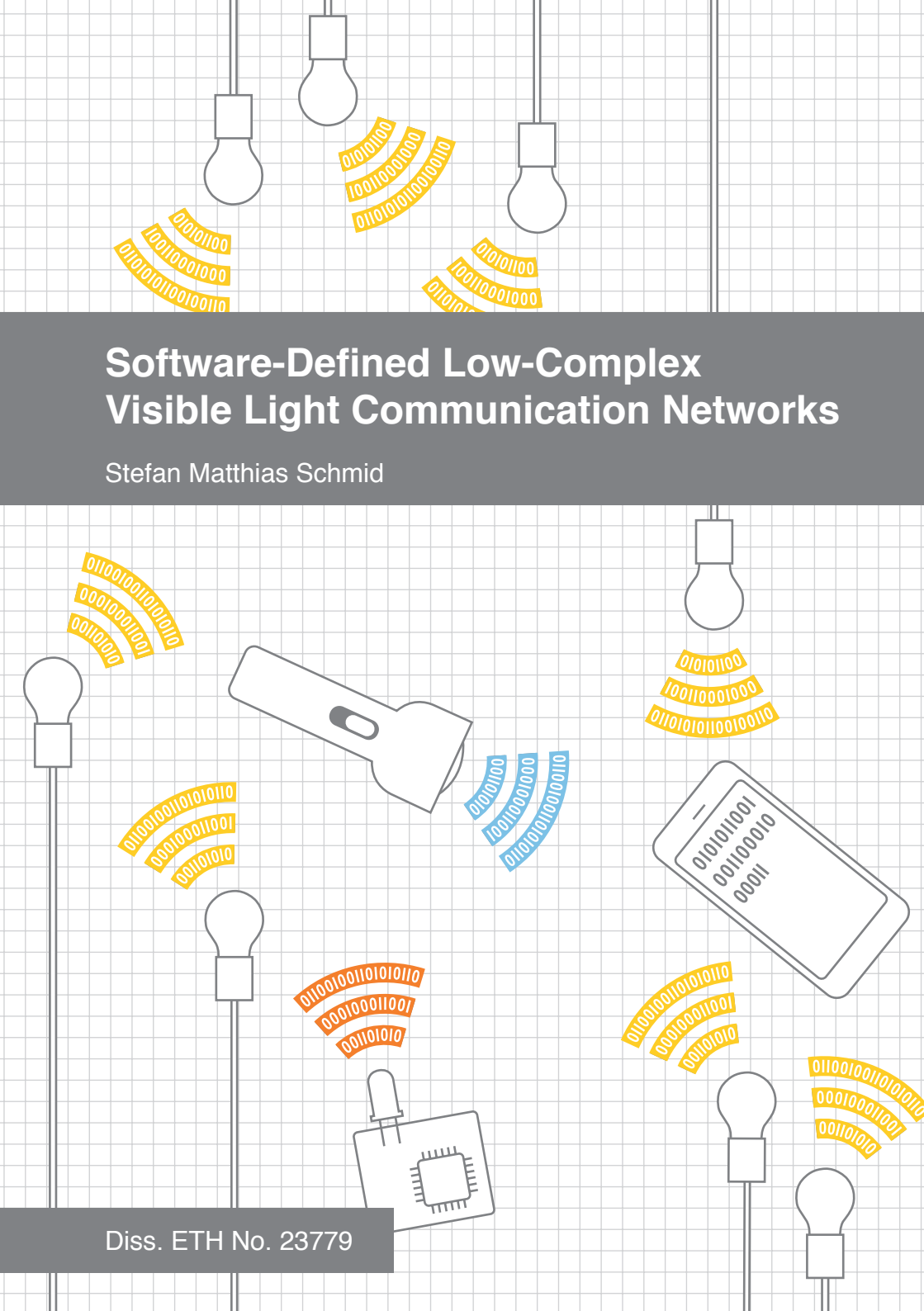
Software-Defined Low-Complex Visible Light Communication Networks

Stefan Matthias Schmid



Diss. ETH No. 23779

Diss. ETH No. 23779

Software-Defined Low-Complex Visible Light Communication Networks

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

Stefan Matthias Schmid

Master of Science ETH in Computer Science

born on 20 December 1983

citizen of
Schaffhausen SH and Rorschacherberg SG
Switzerland

accepted on the recommendation of

Prof. Dr. Thomas R. Gross, examiner
Dr. Stefan Mangold, co-examiner
Prof. Dr. Koen G. Langendoen, co-examiner
Prof. Dr. Edward W. Knightly, co-examiner

2016

ABSTRACT

Today, everything is connected. The Internet transports a massive amount of data every second, and the traffic created by social media, instant messaging, and video streaming is growing day by day. The last step to end user devices, such as mobile phones or laptops, is usually bridged via a wireless connection. Data is transmitted wirelessly using electromagnetic waves with carrier frequencies within the radio spectrum, which is strictly regulated by government organizations. The wireless communication industry tries to keep up with the data growth and regularly releases new standards based on technological improvements to increase the achievable data rates. The communication channel capacity, and thus the maximum possible data rate, depends on the available bandwidth. As only limited spectrum slices are free for commercial and private use, a certain data rate cap will be reached at some point. Fortunately, there is a large piece of the electromagnetic spectrum at hand, also known as *Visible Light*, that is not regulated and can be exploited for wireless communication.

Visible light as communication medium has many promising characteristics. Since the medium can be seen, communication becomes directable and provides visible feedback. Light cannot pass (most) solid objects and can therefore be simply contained within a room, providing a secure communication channel. Furthermore, light sources are available everywhere, enabling the reuse of existing infrastructure to combine illumination and communication.

In contrast to other works in the field of Visible Light Communication (VLC), addressing new modulation schemes and data rate improvements, this thesis focuses on a low-complex and software-defined approach and presents inexpensive VLC systems for different scenarios. Basic microcontrollers, off-the-shelf Light Emitting Diodes (LEDs) used as sender *and* receiver, and software-based communication protocols provide a solid basis for VLC

networking. The communication protocols presented in this thesis simultaneously provide illumination (without flickering) and communication and can be applied to various consumer devices, such as toys, mobile phones and lighting infrastructure, while reusing hardware already in place. The capabilities of the introduced communication protocols are further demonstrated with a system based on modified LED light bulbs, called *EnLighting*, that can be used for illumination and at the same time provides a room area network. It allows communication with visible light and also represents a promising platform for indoor localization.

As this thesis aims for low-cost, low-complex, and software-centric system designs, a diversity of devices can be interconnected with a single set of protocols at moderate data rates, enabling new interaction techniques and applications. Moreover, an unintrusive and ubiquitous system like *EnLighting* can provide a communication fabric for the many devices of the envisioned *Internet of Things* without relying on the crowded radio spectrum.

ZUSAMMENFASSUNG

Heutzutage ist alles vernetzt. Jeden Tag werden riesige Datenmengen durch das Internet transportiert. Der Datenverkehr, erzeugt durch die sozialen Medien, Sofortnachrichten und Video-Streaming, nimmt mit jedem Tag zu. Der letzte Abschnitt des Übertragungsweges zu einem Endbenutzergerät, wie beispielsweise einem Mobiltelefon oder Laptop, erfolgt meistens drahtlos. Daten können drahtlos mit Hilfe von elektromagnetischen Wellen übermittelt werden. Für die Trägerwellen werden Radiofrequenzen verwendet, die durch Regierungsorganisationen streng reguliert werden. Die Telekommunikationsindustrie versucht mit dem Datenwachstum Schritt zu halten und veröffentlicht regelmässig neue Kommunikationsstandards, die auf technischen Verbesserungen beruhen und höhere Datenraten ermöglichen. Die Kapazität eines Kommunikationskanals definiert die maximale Datenrate und ist abhängig von der verfügbaren Bandbreite. Da nur kleine Teile des Radiospektrums für private und kommerzielle Zwecke freigegeben sind, wird irgendwann eine Obergrenze der möglichen Datenübertragungsrate erreicht werden. Erfreulicherweise steht aber noch ein anderer grosser Teil des elektromagnetischen Spektrums, auch besser bekannt als *sichtbares Licht*, zur Verfügung. Dieser Teil des Spektrums ist nicht reguliert und kann auch für drahtlose Kommunikation genutzt werden.

Ein Kommunikationsmedium basierend auf sichtbarem Licht hat vielversprechende Eigenschaften. Da das Medium sichtbar ist, kann die Kommunikation einfach in bestimmte Richtungen gelenkt werden. Es ist auch möglich Licht auf einfache Weise nur auf einen Raum zu begrenzen. Dadurch kann ein sicherer Kommunikationskanal entstehen, der von ausserhalb des Raumes nicht abgehört werden kann. Ausserdem sind Lichtquellen an vielen Orten bereits vorhanden. Diese bereits existierende Infrastruktur kann verwendet werden, um Beleuchtung mit Kommunikation zu kombinieren.

Andere Arbeiten im Bereich der Kommunikation mit sichtbarem Licht behandeln und verbessern Modulationsarten mit dem Ziel, die Datenrate zu erhöhen. Im Gegensatz dazu befasst sich diese Arbeit mit möglichst einfachen und softwarebasierten Ansätzen für kostengünstige und lichtbasierte Kommunikationssysteme, die in verschiedenen Szenarien eingesetzt werden können. Simple Mikrokontroller, handelsübliche Leuchtdioden, die zum Senden *und* Empfangen benutzt werden können, und softwarebasierte Protokolle bilden eine stabile Grundlage für lichtbasierte Kommunikationsnetzwerke. Diese Arbeit stellt Kommunikationsprotokolle vor, die Beleuchtung und Kommunikation miteinander kombinieren (ohne sichtbares Flackern) und von verschieden Endkundengeräten, wie zum Beispiel Spielzeug, Mobiltelefone oder Leuchten, verwendet werden können, indem möglichst Hardware, die bereits vorhanden ist, ausgenutzt wird. Das *EnLighting* System, bestehend aus modifizierten Leuchten, zeigt die Möglichkeiten der vorgestellten Protokolle auf: Die Lampen können zur Beleuchtung eingesetzt werden, zeitgleich wird aber auch ein Netzwerk innerhalb des Raumes aufgebaut, das Kommunikation mit sichtbarem Licht ermöglicht. Weiter beweist sich das System auch als eine vielversprechende Plattform zur Lokalisierung innerhalb von Gebäuden.

Das Ziel dieser Arbeit ist es, einfache, kostengünstige und softwarebasierte Systemkonzepte zu definieren, die es ermöglichen, eine Vielzahl von Geräten mit denselben Protokollen zu vernetzen. Die erreichten Datenraten fördern neue Anwendungsbereiche und Interaktionstechniken. Ausserdem bietet ein unaufdringliches und allgegenwärtiges System wie *EnLighting* eine Kommunikationsstruktur, die für das *Internet der Dinge* genutzt werden könnte, ohne das dicht belegte Radiospektrum zu belasten.

PUBLICATIONS

The following publications are included in part or in an extended version in this thesis:

- S. Schmid et al. "An LED-to-LED Visible Light Communication System with Software-based Synchronization." In: *Proc. IEEE Globecom Workshops*. Dec. 2012, pp. 1264–1268
- S. Schmid et al. "LED-to-LED Visible Light Communication Networks." In: *Proceedings of the Fourteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. MobiHoc '13. Bangalore, India: ACM, 2013, pp. 1–10
- S. Schmid et al. "(In)Visible Light Communication: Combining Illumination and Communication." In: *ACM SIGGRAPH 2014 Emerging Technologies*. SIGGRAPH '14. Vancouver, Canada: ACM, 2014, 13:1–13:1
- S. Schmid et al. "Using Consumer LED Light Bulbs for Low-cost Visible Light Communication Systems." In: *Proceedings of the 1st ACM MobiCom Workshop on Visible Light Communication Systems*. VLCS '14. Maui, Hawaii, USA: ACM, 2014, pp. 9–14
- S. Schmid et al. "Continuous Synchronization for LED-to-LED Visible Light Communication Networks." In: *Proc. 3rd Int Optical Wireless Communications (IWOW) Workshop in*. Sept. 2014, pp. 45–49
- S. Schmid et al. "From Sound to Sight: Using Audio Processing to Enable Visible Light Communication." In: *Proc. IEEE Globecom Workshops (GC Wkshps)*. Dec. 2014, pp. 518–523
- S. Schmid et al. "Linux Light Bulbs: Enabling Internet Protocol Connectivity for Light Bulb Networks." In: *Proceedings*

of the 2nd International Workshop on Visible Light Communications Systems. VLCS '15. Paris, France: ACM, 2015, pp. 3–8

- S. Schmid et al. "EnLighting: An Indoor Visible Light Communication System Based on Networked Light Bulbs." In: *Sensing, Communication, and Networking (SECON), 2016 13th Annual IEEE International Conference on*. June 2016

- S. Schmid, L. Arquint, and T. R. Gross. "Using Smartphones as Continuous Receivers in a Visible Light Communication System." In: *Proceedings of the 3rd International Workshop on Visible Light Communications Systems*. VLCS '16. New York, NY, USA: ACM, 2016

In addition, the following works were also published while pursuing the Ph.D. but are not further discussed, since their topics lay outside of the scope of this thesis.

- S. Schmid, S. Mangold, and T. R. Gross. "Wireless LAN in Paired Radio Spectrum with Downlink-Uplink Separation." In: *Proc. IEEE Wireless Communications and Networking Conf. (WCNC)*. Apr. 2014, pp. 3349–3354

- G. Corbellini et al. "Connecting Networks of Toys and Smartphones with Visible Light Communication." In: *IEEE Communications Magazine* 52.7 (July 2014), pp. 72–78

- F. Zünd et al. "Unfolding the 8-bit Era." In: *Proceedings of the 12th European Conference on Visual Media Production*. CVMP '15. London, United Kingdom: ACM, 2015, 9:1–9:10

ACKNOWLEDGMENTS

While perusing my undergraduate studies I had a fixed plan: get the degree as fast as possible and afterwards find work in the industry as a software engineer. Fortunately, during the time as a master's student, my future started to diverge from the path I naively predicted for myself.

Thanks to a series of fortunate events, I got the chance to work in Stefan Mangold's Wireless Communications and Mobile Computing team at Disney Research Zurich as an intern, where I, for the first time, came in contact with applied research in the area of wireless communication systems. Recognizing that I was still not convinced that research was the right thing for me, Thomas Gross (at this time my master's thesis advisor) and Stefan Mangold tried to talk me around to prolong my stay at ETH Zurich for several years. As the reader might probably guess, they were rather successful. I am immensely thankful for their efforts at persuasion. Had I not started my doctoral studies five years ago, I would have missed a great opportunity to learn so many things. I would also like to truly thank them for the privilege to conduct my research as a joint Ph.D. student at Disney Research and ETH Zurich, for the freedom to pursue my own ideas, and the constant support and guidance.

I would like to express my gratitude to Koen Langendoen and Edward Knightly for having kindly agreed to become co-examiners of my work, and for the time they have taken to read my thesis and participate in the examination. Their feedback in the final stages of my work was highly appreciated.

I wish to thank my colleagues at Disney Research and ETH Zurich who made my Ph.D. journey an unforgettable and fun trip: the members of the "Wireless Group" at Disney Research, Domenico Giustiniano, Vladimir Vukadinović, Fabio Zünd, Adriano Galati, Lito Kriara, Virág Varga, and Manuela Hitz and the members at the "Laboratory for Software Technology", Zoltán Majó, Faheem Ullah, Luca Della Toffola, Martin Bättig, Michael

Faes, Aristeidis Mastoras, Remi Meier, and Ivana Unković. Special thanks to Giorgio Corbellini, together with whom I started the project that later became my thesis work. I gladly recall the days we spent together in front of the computer and oscilloscope, jointly exploring the frontier of an emerging research area. He also acted as a valuable source of feedback and advice as my work progressed. Another special thanks to Ronnie Gaensli who helped me with many projects and taught me the basics of electronics and soldering, and Jan Wezel who was an outstanding resource for 3D modeling and printing.

Also, I would like to thank my friends Fabian Dreier and Robin Loop for the evenings we went out for beers, BBQs (also with beers), and lunches (sometimes with beers), where I could speak about the challenges I currently faced in my research, but even more important, where we also talked about all the other things not related to work. Of course, I should not forget to mention my oldest friend (I even have to since he did the same in his thesis). I would like to thank Matthias Geel for being a good friend and especially for the enduring discussions (few of them with a meaningful outcome) we had over the many years we have known each other.

Finally I would like to thank my family for their unconditional support, in particular, my partner in life, Sandra Bodenmann. She has always been there for me during my years of study. Thank you, Sandra, for always believing in me and for bearing with me, even during the stressful time when I had to write this thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

Not so long ago, you could sit in the movies and enjoy the newest feature starring your favorite actor without being disturbed by a silly ringtone. At lunch with your friends, you could discuss endlessly about a piece of trivia without someone grabbing their mobile phone to lookup the truth. And during your daily commutes, you could look out the window and dream about the future, instead of continuously refreshing your preferred news app in fear of missing the latest rumors. (Un)fortunately, those days are gone.

Today everything is connected, namely wirelessly. We carry the Internet in our pockets with a smartphone permanently connected to cellular networks, wherever we are, allover the world. Computers, laptops, television sets, and tablets are connected to Wi-Fi networks at home, streaming data to and from the Internet. The amount of data consumed by each person is growing every day. Social media, instant messaging, daily news, video streaming, and e-mail traffic data is mostly transmitted wirelessly for the last part to the end device.

The wireless communication industry tries to keep pace with the exponentially growing data demands, which is not at all an easy task. As the required data rates grow, the available resources stay mostly constant. Wireless communication is based on electromagnetic waves and various modulation schemes. Nearly all communication takes places in the radio spectrum from 3 kHz to 300 GHz, which is strictly regulated by government organizations. Commercial and private use is not allowed without a license, except several tiny chunks of the spectrum, the so called Industrial, Scientific and Medical (ISM) bands, that are free to adopt by everyone. For instance, Wi-Fi networks, cordless telephones and microwave ovens employ frequencies belonging to ISM bands. The bigger part of the radio spectrum is in use for military purposes, satellite communication, or navigation services,

or it is licensed to radio and television broadcasting companies and mobile network operators.

The wireless channel capacity and thus the maximum achievable data rate is directly proportional to the available bandwidth (width of the spectrum slice). As spectrum is obviously a scarce resource and additional chunks are rarely released by the governments, the wireless communication industry uses other methods to increase data rates, such as increasing the spectrum efficiency, optimizing shared medium access, or decreasing the area covered by a cell to increase signal quality and to reduce the number of served clients.

Technology can keep up for now, but soon, the theoretical limits will be reached. Furthermore it seems that the Internet of Things (IoT) is eventually happening, adding many more wirelessly connected devices to the already crowded pool of end devices. It can be concluded that more spectrum is required to fulfill all mobile data needs. In addition to the ISM band at 60 GHz (which might be in heavy use soon), a tremendous piece of the electromagnetic spectrum from 430 to 770 THz is unregulated and free to use. It is also known as *Visible Light*.

1.1 MOTIVATION

First attempts using visible light as a communication medium took place in the late nineteenth century. Alexander Graham Bell (who also made significant contributions to the invention of the telephone) constructed devices that enabled the first voice transmission over a wireless channel, using sunlight as communication medium. He called it the *Photophone*. A person could speak into a mouthpiece that guided the sound waves towards a mirror. The mirror was deformed by the sound waves and either scattered or condensed reflected sunlight. At a distance of more than 200 m the reflected light was collected with a parabolic mirror where the modulated light hit a material that changed electrical resistance when illuminated. The whole apparatus was hooked up to an ear peace of a telephone prototype to make the electrical signal audible again.

After Bell's successful experiments, the visible light spectrum and its possible applications for wireless communication went unnoticed for more than a century as radio communication became more popular. Only at the beginning of the twenty-first century, researchers started thinking again about the almost forgotten piece of spectrum, having the dwindling resources, the radio spectrum, in mind.

With better tools on hands now, researchers are working on continuously increasing the capacity of Visible Light Communication (VLC) links. Light Emitting Diodes (LEDs) are power-efficient light sources and can be modulated electronically at high frequencies. Their counterparts, photodiodes, used to convert modulated light back to electrical signals, are built for high sensitivity and optimized for quick response times.

Visible light as a communication medium also provides other promising characteristics aside from the vast amount of bandwidth available. In contrast to radio waves, communication based on light is visible. As communication can be seen, it becomes directable adding visible feedback. This enables new interaction and control methods, but also grants additional security. As the communication reach is assessable with the naked eye, it is more difficult for eavesdroppers to intercept messages. Due to the short wavelength, visible light cannot pass through most solid materials and can therefore be contained without effort, e.g., within a room, enabling the forming of small cells with well-defined borders and providing a secure communication channel. The most practical aspect about VLC is that light sources, thus possible transmitters, are already available everywhere: street lamps, lighting within buildings, flashlights, indication lights in consumer electronics, and experience enhancing lights in toys, all based on efficient and low-cost LEDs.

Analog to radio waves, light needs to be modulated as well to transfer data from a source to a receiver. In VLC, mostly intensity-based modulation schemes are applied. The light is repeatedly switched on and off, which can be detected by a receiver. The receiver can extract digital information, depending on the used frequencies or the position of "on" and "off" pulses. If the frequency of those intensity changes is high enough (above 100 Hz), and the

average light output is constant, a human observer does not note any flickering and only observes a steady light. Having a light output that seems constant while still data is transmitted enables the combination of illumination and communication. Lamps and lights used for illumination and indication can at the same time communicate with other devices in the immediate vicinity, serving as a ubiquitous communication infrastructure.

This thesis focuses on how to reuse those already in place communication-enabling components to realize an unintrusive communication system, interconnecting various devices with different capabilities with a single communication protocol. The goal is to always maintain the primary task of lighting and illumination at the same time. The repurposing of LEDs to transceivers and the utilization of software-based communication layers running on off-the-shelf microcontrollers provide bidirectional communication facilities to formerly unconnected devices, often without the addition of extra hardware. A software-defined system relying on as few as possible supplementary hardware components, implements a flexible, low-cost, and low-complex approach to interconnect many devices as, e.g., envisioned by the IoT, without stressing the already scarce radio spectrum resources.

1.2 CONTRIBUTIONS AND THESIS STRUCTURE

This thesis contains the listed contributions and is structured as follows:

Chapter 2 provides context regarding VLC and discusses recent advances in the area of communication and communication systems using visible light as a medium. The focus is on complete communication systems and their applications and to a lesser extent on the development of new Physical (PHY) layer modulation schemes. Related work is summarized, advantages and disadvantages are pointed out and compared to similar work discussed in this thesis.

Chapter 3 introduces communication protocols that enable LED-based networking. The PHY layer protocol is designed so that communication and illumination is separated to provide a well-defined wireless channel assessment method, although light out-

put for a human observer is kept constant. Furthermore, LEDs are employed as transceivers, thus act as transmitters and receivers while at the same time maintain their illumination capabilities. A Medium Access Control (MAC) layer protocol defines device addressing, retransmissions and shared medium access and enables efficient VLC networking. The aforementioned protocols are implemented in software and run on an off-the-shelf 8-bit microcontroller. The two protocol layers are integrated in a communication library called *libvlc*, separating application from communication implementation. The library provides an intuitive Application Programming Interface (API) that enables rapid prototyping for VLC applications. The communication protocols, also including forward error correction, are evaluated in multiple scenarios and the results are visualized an discussed.

Chapter 4 presents the integration and adaptation of *libvlc* for light bulbs networks. Commercially available light bulbs are modified to host a VLC controller (running *libvlc*), connected via a serial interface as external peripheral to a System-on-a-Chip (SoC) board (running an embedded Linux distribution). The VLC controller is transparently integrated into the Linux network stack (as an Ethernet interface) to enable Internet Protocol (IP) layer data traffic over VLC links. As the light bulbs high-power LEDs are not capable light receivers, four photodiodes pointing in four different directions, are added to the system. The photodiodes are also built into *libvlc* as a receiving channel, together with a channel multiplexing method to sense light from all different directions virtually at the same time. The SoC board comes with a Wi-Fi module which is used to establish a control channel. Testbed software manages deployed light bulbs, implementing functionality to start and stop measurements, to collect data, and to upgrade the VLC controller's firmware over the wireless control channel. The system, called *EnLighting*, is evaluated for different transport layer protocols and different scenarios (direct link and multihop), using the VLC channel as communication link. The measurement results are visualized and discussed. Furthermore, the capabilities for indoor localization are explored, exploiting the fact that the light bulbs can not only transmit, but also receive.

Chapter 5 explains how to extend *libvlc* to add support for additional hardware platforms, by introducing a hardware adaption architecture. The PHY and MAC layers are implemented on top of this abstraction to simplify the adoption of new processors. A prototype board with a 32-bit microcontroller is designed and built and the corresponding hardware abstraction for *libvlc* is implemented. Furthermore, *libvlc's* PHY layer is complemented with three additional PHY layer modes to increase the data rate for good channel conditions. The PHY mode can be changed dynamically based on an automatic adaptation scheme. An additional synchronization correction method enables high data rates also for light bulb at large distances, or for no line of sight scenarios, where light is reflected at doors or walls. Several scenarios for LED-to-LED networks and light bulb networks are evaluated and the results are visualized and discussed.

Chapter 6 discusses three VLC-based applications. The first application describes the design an implementation of a peripheral device for smartphones. The peripheral device consists of an LED, photodiode, and an energy harvesting circuit and is plugged into the smartphone's audio jack. An application running on the mobile phone generates audio waveforms to modulate the peripheral's LED according to *libvlc's* PHY layer. The light signal picked up by the photodiode is fed into the microphone input and processed by the application to decode the bits transmitted by the PHY layer protocol. The presented peripheral device enables bidirectional communication with any device running *libvlc*.

The second application is based on a smartphone software that exploits the rolling shutter effect. Reflected light from a light source using *libvlc's* PHY layer is recorded with the smartphone's camera. The resulting barcode-like video frames can be decoded in real-time, establishing a light source to mobile phone communication channel driven only by an application, without hardware changes to the smartphone itself. Current smartphone cameras with high video recording frame rates enable the direct integration into the *EnLighting* system based on *libvlc's* continuous protocols.

The third and last application explores the use of VLC as interaction and control method. An LED flashlight is modified to

implement *libvlc's* protocol to transmit short beacons which trigger predefined actions in receiving devices. A prototype flashlight can be used to switch lamps on and off by simply pointing towards their light bulbs. A user study is conducted for which candidates complete tasks such as enabling specific lamps or creating requested light configurations, once with an ordinary light switch interface and once with the flashlight as a remote control. The time needed to complete the tasks is recorded and a questionnaire is filled in by each candidate. The results of the user study are presented and discussed.

Chapter 7 discusses future work, summarizes all results and concludes the thesis.

2 RELATED WORK

In this chapter, the related work is summarized and commented and analyzed in respect to the VLC system proposed in this thesis. The chapter is structured into three parts. The first part discusses the Institute of Electrical and Electronics Engineers (IEEE) 802.15.7 standard [34], which defines PHY and MAC layer schemes for VLC systems to guarantee interoperability. The second part presents an overview of VLC core communication systems. Such a system provides the foundation for any VLC communication link (and networking) and relies on hardware and/or software to operate transmitter and receiver. The third and last part discusses the usage of VLC for specific applications. Some systems are exclusively built with a specific scenario in mind and others are built on top of adapted communication systems to be exploited for different objectives. This part is subdivided into three sections each focusing on different applications. Indoor localization applications can benefit from the omnipresent lighting devices already in place and utilize the rather short range communication capabilities to form small cells. Current mobile devices employ fast processors and are already equipped with sensors and extension interfaces, providing an excellent playground for prototyping communication systems. Furthermore, novel interaction and sensing methods take advantage of the visibility and directionality of communication using the visible light spectrum.

2.1 THE IEEE 802.15.7 STANDARD

The IEEE 802.15.7 standard [6, 34] has influenced various aspects of recent VLC research. The document specifies a PHY [74] and MAC protocol layer [61] with different optional transmission schemes and protocol configurations for supporting a variety of use cases. The PHY layer is split into three different operating modes. PHYI is optimized for larger communication distances

in outdoor applications (e.g., car or traffic lights) and uses On-Off Keying (OOK) (with Manchester codes to prevent flickering) and Pulse-Position Modulation (PPM) schemes at data rates up to 260 kb/s. PHYII is targeted for higher rates and indoor point-to-point applications. Again, OOK and PPM schemes are employed to achieve data rates up to 96 Mb/s. PHYIII is based on a Color-Shift Keying (CSK) modulation scheme to be used together with Red Green Blue (RGB) light sources (providing white light) for indoor usage and data rates up to 96 Mb/s.

Light dimming [69], when using OOK, is realized by inserting compensation time between modulated data, decreasing the light output but also reducing the data rate. This is similar to the adaptive brightness approach, which will be introduced in Section 5.2.1. Light dimming for PPM is implemented by reducing the duration of the light emitting pulses while keeping the data rate constant. Also while not transmitting, a constant light level needs to be maintained. The standard defines two available options: in-band and out-of-band idle patterns. The duty-cycle of the idle pattern can be adapted to assume variable brightness levels. The in-band idle pattern uses the same frequency as when transmitting data and is therefore visible for a receiver (within the applied bandpass). An out-of-band pattern uses lower frequencies (also using a Direct Current (DC) bias only is allowed), which are not visible for a receiver, to control luminosity.

The MAC layer protocol foresees three typical topologies: peer-to-peer, star and broadcast. In a peer-to-peer topology, the devices are allowed to directly communicate with each other using random medium access with optional Clear-Channel Assessment (CCA). The standard does not further specify how devices synchronize their transmitting and receiving attempts. In a star or broadcast topology, only data transfers between a coordinator and participating devices are allowed. Data is either transferred within a super frame structure embedded between beacons sent by the coordinator or directly via random medium access (when beaconing is disabled).

Many works focus on providing results based on simulations for the PHY and MAC layer proposed by the standard [32, 54, 59–61, 75, 89]. Some works also propose changes to the introduced

MAC layer [30, 31, 47] to improve performance or to adapt to particular scenarios, e.g., combining the PHY and MAC layer to a cross-layer protocol [58]. There are also several implementations available, built on top of Software Defined Radio (SDR) [5, 23, 29], Field Programmable Gate Array (FPGA) [8], or realized as SoC [9]. These efforts completely focus on the PHY layer only and evaluate direct link scenarios. MAC and networking for multiple devices are not further investigated.

The approach to VLC, which will be described in this thesis, follows a different direction. It aims towards low-complexity and the reuse of existing hardware components to upgrade existing devices with VLC networking capabilities. The simplicity of the system design is reflected by the fact that LEDs are employed as transceivers and microcontrollers are used instead of dedicated communication hardware. The system implementation is based on a flexible and inexpensive software-based platform and therefore only achieving comparably low data rates (which are enough for the envisioned scenarios). The system that will be presented in this thesis clearly separates illumination from communication to enable a proper MAC layer protocol with CCA. According to the standard, a CCA mechanism is optional and not required for random medium access although detecting a busy channel is fundamental for a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol to prevent unnecessary collisions and increase channel efficiency. Additionally, the introduction of idle patterns to support light dimming makes distinguishing communication from illumination more difficult.

The standard lays a solid foundation for developing VLC systems, but has not been widely adopted so far. It has not been updated since 2011 and therefore misses recent advancements in VLC such as the inclusion of an Orthogonal Frequency-Division Multiplexing (OFDM) modulation scheme [1, 17, 19]. Furthermore, VLC is often applied for niche scenarios where special requirements demand a custom tailored communication system (as the work to be discussed in this thesis) and the VLC standard cannot be applied directly. Since there are no commercial products available implementing the standard, researchers tend to develop

their own version of a VLC system, deviating from the proposed PHY and MAC layer schemes.

2.2 CORE COMMUNICATION SYSTEMS

VLC faces many challenges [64], but several systems based on hardware and software [18, 20, 65] emerged during the last years. This section discusses approaches categorized in hardware-based systems that focus on performance, and software-based systems for flexible and cost-efficient solutions.

2.2.1 *Hardware-Based Systems*

The main difference between hardware-centric systems is the employed modulation scheme. The most straight forward scheme is OOK where the light is switched on and off (or not completely off to maintain a certain light intensity). A data rate of 230 Mb/s for phosphor-based white LEDs [94] for distances less than 1 m was demonstrated. By using an RGB LED (for higher bandwidth) the data rate can be improved up to 477 Mb/s [22] for distances also less than 1 m. Both system use dedicated hardware for transmitter and receiver and employ photodiodes as light detectors. When used in conjunction with Run Length Limited (RLL) codes (e.g., Manchester) to mitigate flickering, OOK provides constant brightness levels. Dimming can be implemented by redefining the light on and off levels or by introducing compensation periods, at the expense of a lower data rate.

Other popular modulation techniques are PPM [62, 63] and CSK [93, 103], which are also part of the VLC standard. The most prominent scheme mentioned in recent literature is OFDM [1, 55]. OFDM is already widely used in Radio Frequency (RF) communication and is known to mitigate inter-symbol interference, which is a big problem for the aforementioned modulation schemes. OFDM employs multiple orthogonal subcarriers within the available bandwidth. Each subcarrier carries data in parallel at a lower symbol rate (still preserving a higher total data rate) increasing the robustness against severe channel conditions. The individ-

ual carriers can be modulated with regular modulation schemes. Data rates up to 3 Gb/s can be achieved with single special LEDs (high bandwidth) [92]. Medium access can be handled by using Orthogonal Frequency-Division Multiple Access (OFDMA) [7], splitting the available subcarriers up between the present networking devices. OFDM-based modulation schemes require dedicated (complex) hardware for both, transmitting and receiving path, and provide only restricted dimming functionality [101].

The researchers who first introduced OFDM for VLC founded a company to commercialize their technology. The company is called pureLiFi[1] and is offering VLC devices[2] to form VLC networks. The devices support data rates of 5 to 40 Mb/s at distances up to 3 m. The communication channel is bidirectional, using an infrared-based backchannel to communicate from a receiving device back to the light source. The devices pack extensive hardware and sensors (with optics) and therefore do not come with a small form factor (and probably also price tag).

The systems discussed in this hardware section mostly focus on the PHY layer aspect of VLC. Networking and thus also MAC is not often mentioned since only single link scenarios are considered. The reported performance and communication distances can be achieved for special cases where the conditions are optimal and the necessary hardware and space is available. The literature discusses many additional modulation schemes and hardware-centric systems. Only the most relevant are discussed here, since this thesis highlights the software and networking aspect for VLC systems. The systems that will be presented in this thesis do not compete with the full-fledged communication hardware. With a software-based and low-complex approach, they take a different path and explore the usage of only basic hardware together with software-defined protocols to provide VLC connectivity for scenarios where inexpensive solutions are required. A flexible software system can be adapted without effort for different applications and scenarios to investigate the applicability of VLC.

1. http://purelifi.com/
2. http://purelifi.com/lifi-products/lifi-x/

2.2.2 *Software-Based Systems*

That LEDs can also be used as light detectors has already been known for some time [56]. Dietz et al. [16] picked up on this idea and built a VLC system based on basic microcontrollers with LEDs as transmitters and receivers, directly connected to the microcontroller's General-Purpose Input/Output (GPIO). The authors use an OOK scheme together with Pulse Width Modulation (PWM) to synchronize two devices (no networking of multiple devices is supported) and transfer data at a rate of 250 b/s. To sense incident light, the same reverse bias and charge method as will be described in Chapter 3 is used. To note is that the remaining voltage (after a charge) is not determined with an Analog-to-Digital Converter (ADC), but with a digital GPIO pin (the voltage needs to drop below the digital input threshold to distinguish bit 0 and 1. This makes their system vulnerable to ambient light changes, whereas the approach discussed later in this thesis uses light level differences to encode bits to filter out constant lighting.

Giustiniano et al. [24, 90] picked up on the LED-only approach and implemented an LED-to-LED communication system with a MAC layer based on a CSMA/CA and Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol. Results for a network throughput of approximately 800 b/s for up to four participating devices are reported. The system supports the elimination of flickering during data transmissions and also while idling or receiving to always provide constant light output. The implemented mechanism to enable constant light output leads to two problems: First, since devices are not continuously synchronized, a preamble preceding the data is used to timely align transmitter and receiver. It works for two devices but as soon as more devices are present in a network, their (unsynchronized) idle patterns superimpose and prevent a receiver from recognizing the preamble pattern. As a result, the transmission fails due to the lack of synchronization. Second, due to the randomly aligned idle patterns of participating devices, a transmitter cannot detect if the channel is free or busy since the outcome of a CCA is random. These two issues will be solved in this thesis with the separation of illu-

mination and communication and the introduction of continuous synchronization.

OpenVLC[3] [96, 98–100] is an open source (at least the software part) software-based VLC research platform. The system is based on a BeagleBone Black[4], which comes with a 1 GHz Advanced RISC Machines (ARM) processor and runs Linux. An additional board that connects to the BeagleBone is necessary for the VLC links. It hosts high power LEDs, a standard LED and a photodiode. The device can either transmit using the high power LEDs or the standard LED. Both signals, from the photodiode and LED, are fed into a transimpedance amplifier connected to an external ADC unit. The system's PHY and MAC layers are implemented as a device driver interfacing with the Linux networking stack. The PHY layer uses an OOK modulation scheme with direct detection at the receiver. The MAC layer is based on CSMA/CA and CSMA/CD. There is also the option (in case the standard LED is used as transmitter and receiver) to allow the embedding of frames into ongoing transmissions to achieve full-duplex communication [95, 97].

OpenVLC can reach data rates up to 20 kb/s and a maximum communication range of 5 m, depending on the used transmitter and receiver pair. Using a standard LED as transmitter and a photodiode as receiver results in the longest communication distance thanks to the LED's narrow light beam. Network User Datagram Protocol (UDP) throughput of total 12 kb/s is demonstrated for two transmitting devices (each 6 kb/s) and one receiving device.

With OpenVLC, a performant software-based solution for VLC networking is freely available. The platform is promising and achieves good results, but there are also a few drawbacks. The PHY layer needs to fulfill real-time requirements. As the used PHY layer slots are only 20 µs long, the operating system needs to guarantee timely code execution within a range of a few microseconds. A heavy load on the system could influence the PHY layer behavior (even for a kernel with real-time extensions). The *EnLighting* system (introduced in Chapter 4), will present a solution where the VLC PHY and MAC layers are integrated into a Linux system as an external device. The VLC system does not

3. http://www.openvlc.org
4. https://beagleboard.org/black

draw from the resources of the Linux system and is completely independent in the execution of the two communication layers. Another disadvantage of OpenVLC is that there is no support for continuous illumination (the light output is only enabled while transmitting). VLC aims at reusing existing lighting infrastructure to provide communication *and* illumination. The protocols that will be presented in this thesis can provide constant light output while idling, transmitting and receiving and can be applied in scenarios where lighting and communication is required. Furthermore, OpenVLC's photodiode-based receiver does not work when a certain level of ambient light is present due to oversaturation (either of the sensor or the amplifier, which is not clear from the papers since the hardware schematics are not published). The sensors used in the *EnLighting* system are DC-filtered before amplification to prevent oversaturation in the amplification stage.

Shine [37] provides similar features as OpenVLC. A custom designed Printed Circuit Board (PCB) hosts photodiodes (amplified) hooked up to an ADC component and LEDs supported by transistors as drivers. The PCB can be connected to any processor (with the necessary interfaces) that runs the PHY layer (OOK via LED drivers and light sampling using the ADC unit) and possible higher layer protocols. What differentiates the proposed platform from OpenVLC and other systems is the use of multiple LEDs (arranged outward facing in a circle) together with four photodiodes to provide a 360° coverage. Groups of LEDs can be controlled individually to emit light only in a specific direction and save energy in return. A prototype implementation using an 8-bit microcontroller achieves approximately 1 kb/s data throughput at distances of not more than 1 m. An additional evaluation demonstrates multi-hop capabilities. Aside from the dependence on several hardware parts, the system provides great flexibility since the transceiver front end is separated from the control processor, which can be substituted based on requirements and usage scenarios. The presented receiver implementation relies on a detection threshold to decode measured light levels to bits. To avoid errors due to sudden ambient light changes and to simplify the receiving procedure, PPM can be used as will be discussed in Chapter 3. Instead of maintaining a threshold and comparing

against it, two consecutive samples (the two pulse positions) can be compared and the resulting bits can be decoded without the influence of ambient light. Also a CCA for MAC can be obtained without comparing samples to a threshold value, namely as long as consecutive samples are different, the channel can be assumed as busy.

VLC systems based on SDR [5, 15, 29, 40, 66, 67] can use more complex modulation schemes and achieve higher data rates and still provide a certain degree of flexibility. Drawbacks are the amount of hardware needed (FPGA, ADC, DAC) and the required computational power for signal processing. All systems that will be demonstrated in this thesis focus on reusing existing hardware (e.g., LEDs as transceiver, smartphone camera as light receiver) to build low-complex, inexpensive, and flexible communication systems. Simple microcontrollers already pack enough hardware and computational power to achieve reasonable data rates for most envisioned applications.

2.3 APPLICATION SPECIFIC SYSTEMS

Wireless communication technology is predominated by RF-based systems. Radio waves can penetrate obstacles, are bent by sharp edges, or even reflected at the ionosphere to reach further. Furthermore, they are invisible. For certain applications, a visible communication medium, which can easily be blocked by its surroundings, can be more useful. The directionality and visibility enables new ways of wireless interactions with other devices. Additionally, data exchange between devices can be considered as secure, since a probable eavesdropper can be easily identified thanks to the visibility and containability of the communication medium, with limited room for attacks [11]. Due to its very short short wavelength (400 to 700 nm), light cannot pass through objects and can therefore be easily contained within a certain area. This enables the formation of cells with exact boundaries, usable for localization or for location-based event triggering. In the following, related work for some VLC applications is summarized. The focus is on application tangent to the work that will be pre-

sented in this thesis, such as indoor localization, smartphone-based VLC, and interaction methods based on VLC techniques.

2.3.1 *Indoor Localization*

As localization based on Global Positioning System (GPS) does not work in most indoor scenarios due to too weak signals, alternatives are required. Indoor localization based on Wi-Fi has been studied extensively in recent work [10]. The reuse of already deployed Wi-Fi access points can lead to a localization precision of a decimeter [39], but requires extensive calibration and is prone to errors caused by the multi-path behavior [88] of radio waves. Similar to localization technology based on Wi-Fi, the existing lighting infrastructure can be exploited for positioning [4]. As there are usually more different light sources within range of a receiver than Wi-Fi access points, a higher triangulation accuracy can be expected.

A localization system called Epsilon [50] uses Received Signal Strength Indication (RSSI) values and trilateration to determine the position of mobile devices. Light sources transmit, using a Binary Frequency Shift Keying (BFSK) modulation scheme, location information (beacons) in fixed time intervals which can be received by mobile devices (smartphones). The luminaries consist of a custom designed control circuit attached to a commercial available 10 W LED and do not provide any sensing capabilities. As multiple light sources are present, channel hopping is used to avoid collisions. At the receiving side, a photodiode together with an amplification circuit and a battery is used to feed a signal into the audio jack (microphone input) of a mobile phone. The signal is sampled by the sound processing hardware and processed by an application on the smartphone. The location information together with the retrieved RSSI value (from at least three different light sources) serves as input for trilateration. The system achieves a positioning accuracy of approximately 40 cm.

Luxapose [41] employs the camera of a smartphone as a light receiver. The light sources transmit beacons containing identification data using OOK. Multiple light sources are captured with the smartphone camera and isolated using computer vision tech-

niques. The individual light sources are identified based on the transmitted beacon with the help of the rolling shutter effect (see Section 6.2.1 for more information). Based on the known location of the transmitting luminaries, the smartphone orientation and the angle of arrival, the location in 3D-space can be calculated. The authors demonstrate a 10 cm positioning accuracy for their testbed setup.

Yang et al. [105] take a different approach when modulating light sources. Their system does not depend on LED-based luminaries. Polarization filters are employed to modulate the light of any light source (even sunlight). A smartphone camera with an applied polarization filter captures the incoming light and uses image processing to decode the transmitted information. The same localization algorithm as used by the aforementioned Luxapose can be applied: multiple light sources (in a captured video frame) can be identified by the broadcast beacons and together with their relative positions, the algorithm determines the receiving mobile phone's position with a sub-meter accuracy.

SpinLight [104] uses infrared light, but is also briefly mentioned since the presented concepts could also be applied to a VLC-based system (with some modifications). An infrared light source is covered with a hemispherical shape. The shape is divided into rings and each ring into different cells. A cell can either be open (representing a 1) or closed (representing a 0). When the covered light source (on the ceiling) is enabled, the cell pattern in projected on the floor. If a device to localize can identify the cell in which it is located, and the position and height of the infrared light source is known, its position can be determined. As the shade is rotating and each ring has a different pattern of open and closed cells, the receiving device can determine the corresponding ring by identifying the predefined pattern. To determine the cell within a ring, a synchronization mechanism is employed to mark the pattern's starting point. If the corresponding cell within a certain ring can be determined, then the position is roughly known and can be further refined to achieve an accuracy of 4 cm. The authors use infrared light since the rotating shade would introduce heavy flickering for a source emitting visible light. The system could be adapted for visible light by replac-

ing the mechanically turning shade with a display pixel matrix as found in projectors, which support refresh rates above 100 Hz. The matrix can be used to generate a similar pattern (without rotating) at a higher frequency to eliminate flickering.

All discussed systems can achieve a localization accuracy below 1 m. Preliminary results for *EnLighting* (discussed in Chapter 4) show that the system is also a promising candidate for localization based on RSSI measurements and trilateration. Compared with the other systems, *EnLighting* has some important advantages. Since the system consists of regular light bulbs, it can be easily deployed anywhere where sockets (or floorlamps) are available. Furthermore the ability to sense incident light enables synchronization and therefore MAC for transmitted beacons, preventing collisions and increasing the localization frequency. The flexible design and the networking capabilities make the system useful for other applications (e.g., neighbor discovery for maintenance purposes) that can be executed concurrently with the localization service.

2.3.2 *Light Sensing for Mobile Devices*

The localization systems reviewed above all require an appropriate receiving device to be useful. Common mobile device such as smartphones and tablets come with light sensors (to control display brightness settings) but these sensors are either not directly accessible or cannot be sampled at the demanded frequencies. Therefore either an additional peripheral connected to the smartphone is required (as proposed in Section 6.1), or another light sensitive part of the device, the camera, can be used. This section summarizes efforts that explore the usage of cameras embedded into mobile devices as endpoints for VLC.

Current smartphone cameras support capturing rates from 30 to 240 frames per second which is not high enough to sample VLC signals, usually modulated above 100 Hz to prevent visible flickering. Furthermore, the camera is controlled by the mobile phone's operating system and does not allow fine grained control of the frame capturing rate (which can even deviate from the chosen setting, depending on the system load) and there-

fore does not allow to synchronize the frame rate to the signal emitted by a light source. Nevertheless, two (successful) approaches to establish a communication link from a flicker-neutral light source to smartphone cameras are discussed in recent work: Undersampled Frequency Shift OOK (UFSOOK) and the exploitation of the rolling shutter effect.

When undersampling a signal with a certain fixed frequency, an alias signal with a different frequency occurs depending on the camera sampling offset [73]. This enables the mapping of alias frequencies to data symbols. A light source can use OOK with several flicker-free frequencies to encode data that are recognized as alias frequencies at the smartphone when analyzing light intensities in a series of consecutively captured video frames [53, 73]. To achieve robust results, the random camera's sampling offset needs to be taken into account [72]. The approach also allows the capturing of multiple light sources in parallel to increase the data rate [71]. Bit rates up to 400 b/s at a communication distance of several meters are reported.

Another approach exploits the rolling shutter effect. A camera's Complementary Metal-Oxide-Semiconductor (CMOS) sensor is read line by line and is not evaluated at once. A light source using an OOK modulation generates a barcode-like pattern contained in a captured video frame. The rolling shutter effect can be seen as an extension of the sampling rate, allowing the processing of a flicker-free light signal. Some efforts discuss the usage of multiple light sources together with a Frequency Shift Keying (FSK) modulation scheme [48, 68]. Data symbols are encoded with multiple frequency which can be retrieved from the captured images by isolating the light sources and analyzing the pattern created by the rolling shutter. When capturing reflected light, the complete video frame can contain data. This enables modulation schemes such as OOK and PWM [14, 21]. The data is extracted by measuring and comparing the width of the bright and dark bars, frame by frame. A time gap between individual video frames where no light is recorded requires the usage of redundant codes. A data rate of approximately 700 b/s at distances up to 3 m can be achieved. If RGB light sources are available, the scheme can be extended with CSK [28] to reach data rates of about 5 kb/s. Color-

flickering is prevented by ensuring that the amount of red, green, and blue light is constant within a critical amount of time. The camera-based system that will be discussed in this thesis (Section 6.2.1) makes use of current smartphones with higher capturing rates (240 frames per second), which reduces the gap between individual frames. This enables the integration of mobile phones as continuous receivers into the existing *EnLighting* system, relying on the same communication protocols.

2.3.3 *Interaction and Sensing*

VLC also enables new interaction techniques based on the visibility of the communication process. Section 6.3 discusses a method to intuitively control light sources by using a flashlight as remote control. A user study shows that common tasks, such as switching on and off lights or creating a certain light configuration, can be executed more efficiently. Schmidt et al. follow a similar approach using a pico projector as a light source [86]. The projector projects a user interface on other devices (e.g., a lamp) equipped with light sensors. The data transfer is started by pressing a button and received by the device's sensor, triggering an action. Data is embedded into the projected images using gray code patterns. Due to the low refresh of the the projector (30 to 60 Hertz), flickering is introduced when transmitting data.

Optical remote controls based on infrared light have been used for decades as a low-cost technology to operate many consumer devices and appliances. While infrared remotes rely on high intensity output to reach the receiver, VLC-based devices can make use of the visible feedback and therefore reduce their power output. Another proposed device can determine its pointing direction with the help of cameras and position sensors and interact with objects by using voice and gesture recognition [102].

VLC can also be applied in human sensing. Zhang et al. [106] present a system called Okuli to extend the user interface of a mobile device to any nearby surface. An LED and two photodiodes (to the left and right of the LED) are placed in a small box. Holes on one side let the light in and out. When moving a finger in front of the box's slits, the light from the LED is partially reflected

back and can be sensed by the photodiodes. The received light information from the two photodiodes is used to calculate the relative position of the finger at an accuracy of 1 cm. This concept can be extended to reconstruct human postures in real-time [51, 52, 107]. LEDs in the ceiling emit light that is partly blocked by a human, creating a shadow on the floor. Photodiodes distributed on the floor capture light (and shadow) values (implementing a low-resolution camera) that are further processed to reconstruct the human posture. The number of required photodiodes can be reduced when using many densely arranged LEDs on the ceiling, each transmitting a distinct beacon (using FSK). The beacon information is used by the system to identify individual LEDs to reconstruct a more accurate shadow map (using fewer photodiodes). The human posture can be tracked in real-time at a refresh rate of 40 to 60 Hz. Furthermore, it has been demonstrated that the system can recognize people [2] with an 80 % success rate.

3 LED-TO-LED NETWORKS

LEDs are everywhere. They are used as indicators in consumer electronics, as flash for smartphone cameras, in modern light bulbs for illumination, and in toys to create play experiences. LEDs are of low-cost and power efficient and nowadays available in various form factors, luminosities and colors which explains the increase in popularity during the past decades. In addition to the ability to emit light, they can also be used to sense light [16] and thus are able transform modulated light into electrical signals.

LEDs as transmitters and receivers, paired with off-the-shelf microcontrollers, provide a novel approach to enable low bit rate wireless communication for short distances [80, 90]. Communication devices assembled from the two aforementioned building blocks can form VLC LED-to-LED networks. They communicate with each other over free-space line of sight channels and achieve a total network throughput in the order of kb/s at distances of no more than a few meters. Except of an LED and a microcontroller, there is no other additional hardware needed. The complexity of a communication device can be kept low by implementing necessary protocol logic in software, embedded into the microcontroller's firmware.

Low-complexity LED-to-LED communication can be applied to sensor networks, home automation systems, and smart illumination, or can provide a fabric to connect consumer devices and toys, being a part of the IoT. These networks can exploit the ubiquitous presence of LEDs and leverage their ability to act as cost-effective transceivers – while allowing the LEDs to continue to operate as lighting devices.

In such VLC networks, photodetectors like phototransistors or photodiodes as light sensors are not required anymore. The VLC devices use off-the-shelf 8-bit microcontrollers, powerful enough to operate the required communication protocols to handle PHY and MAC layer. The LED-to-LED network hides the exchange of

messages within the illumination. A message transfer has no effect on the level of brightness (for a human observer), so that an LED appears to be switched on all the time.

This chapter describes the design, implementation and evaluation of an LED-only VLC system that also enables networking for multiple participating devices. The system includes a software-based PHY layer and a contention-based MAC protocol layer. The communication between two devices requires a PHY layer protocol with focus on robustness and accurate wireless synchronization. For the MAC layer, a CSMA/CA protocol is defined based on the IEEE 802.11 [33] standard.

The chapter is structured as follows: Section 3.1 provides a description of the overall system design. Section 3.2 and Section 3.3 describe the hardware and software (protocol) building blocks used to stitch together a working VLC communication system. Section 3.4 discusses implementation details and concepts and Section 3.5 provides insides regarding testbed and system evaluation. Section 3.6 summarizes all findings and concludes the chapter.

3.1 SYSTEM DESIGN

The system design builds upon two main ideas: flexibility and low-complexity. Flexibility is important since a research platform is continuously developed further, refined and extended. Switching to different hardware while prototyping can be done with less effort if hardware and software is decoupled. Changing communication protocols, fine tuning parameters, and maintenance is straightforward when implemented as a software system. High flexibility can be achieved when implementing most system parts in software while using only basic hardware components. At the same time, the system complexity is reduced. Low-complex systems have the advantages that they can be well and clearly defined and since fewer (hardware) parts are involved, cost can be kept at a minimum. Having a low-complex and low-cost footprint makes the system more applicable for consumer electronics, toys and the IoT.

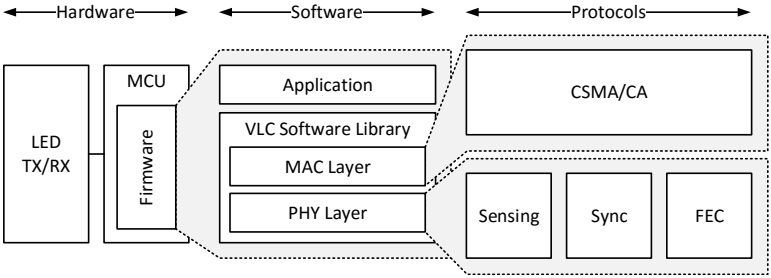


Figure 3.1: Overall system design. An off-the-shelf LED is directly connected to the Input/Output (I/O) pins of a Microcontroller (MCU) and can be used for Transmitting (TX) and Receiving (RX) data. The communication protocols for the PHY and MAC layer run directly on the MCU's firmware encapsulated into the VLC software library. Wrapped into these layers are protocols for light sensing, synchronization and Forward Error Correction (FEC) (PHY layer), and for CSMA/CA (MAC layer).

Figure 3.1 illustrates the overall system design for a software-based VLC system using LEDs as transceivers. Off-the-shelf LEDs are used as light sending and receiving front end. Why LEDs can be used as receivers and how they can be employed in a communication system is outlined in Section 3.2.2. The transmitting and receiving LED is directly connected to the microcontroller's Input/Output (I/O) pins without the necessity of an intermediate electronic circuit. This improves the system flexibility and hardware independence. The microcontroller requirements and type are explained in Section 3.2.1.

The firmware running on the microcontroller consists of two parts: first, the VLC software library and, second, the application. The library contains the PHY and MAC layer protocols and provides a high-level API (usable by the application), which offers facilities to send and receive messages and to collect statistics. The application contains the program logic for a certain use case. Separating the application from the communication protocols makes it possible to independently work on both parts (application and library). As long as the API stays consistent the communication library can be extended and developed further without influenc-

ing the application code. Section 3.3.3 describes the core APIs functionality.

The PHY layer incorporates a light sensing protocol that defines when and how light is modulated and sensed. Light is modulated by turning on and off the LED in predefined patterns, whereas light is sensed by measuring the photocurrent generated by incident light. Sensing and transmitting devices need to be aligned to allow the receiving device to sense at the right moments when the transmitting device is modulating the LED. The synchronization protocol is able to correct the initial timing off-set and can also keep the devices synchronized over time counteracting the the devices' drifting clock. Another building block is the Forward Error Correction (FEC) based on Reed-Solomon [70] error correcting codes. It is able to correct a predefined number of errors (byte errors in this specific implementation). The PHY layer is explained in detail in Section 3.3.1.

Since multiple devices (not only two) should be able to form a network and communicate with each other, there will be competition for the shared communication channel. Packets sent at the same time have a chance to collide and therefore can most likely not be decoded correctly by the receiver. A MAC protocol is able to handle distributed medium access and can decrease the collision probability significantly. The implemented protocol is based on the IEEE 802.11 [33] standard, supporting random contention windows, Request to Send (RTS)/Clear to Send (CTS), retransmissions and device addressing, and is further described in Section 3.3.2.

3.2 HARDWARE BUILDING BLOCKS

The VLC communication device depends on two hardware building blocks: a main processor to run all communication protocols and an LED for light emission and sensing. There are certain requirements for a microcontroller to be able to run the proposed communication protocols: there has to be enough computation power to handle all protocol levels and timer and ADC peripherals need to be available. Almost any off-the-shelf LED is applicable as a transceiver front end. The sensitivity depends on the LED's

color, the housing form factor (lens), and the housing filter properties.

3.2.1 *System Processor*

Communication systems usually employ a system processor (can be a programmable microcontroller to increase flexibility) to run the application logic and driver for the communication front end. In the approach presented in this thesis, the system processor is also responsible to run the VLC front end, consisting of the PHY and MAC layer. Getting rid of additional hardware reduces cost and complexity and the software-centric solution increases flexibility and maintainability. Although there is an additional workload, a basic 8-bit microcontroller is still powerful enough to run the communication protocols presented in this chapter. Since only a basic processor is needed, devices that are already equipped with microcontrollers and LEDs could be extended with communication capabilities by a software update only.

For successful prototyping, not only a processor is necessary but also the processor's pins need to be accessible, and there should be a convenient way to program and debug the software running on the microcontroller. For this purpose, manufacturers often provide evaluation boards containing all the needed hardware for programming, debugging and serial communication (for logging and debug output) together with processor pin breakouts for easy access. The recent Maker Movement [3, 26] also produced many different kinds of prototyping and evaluation boards with good availability and support by large communities.

One of those prototyping boards is the Arduino[1]. In the meantime, the Arduino makers are selling a variety of boards with different processors. When referring to the Arduino in this thesis, the classic Arduino UNO[2] board is meant. The Arduino board is shown in Figure 3.2. It is equipped with a ATmega328P[3] microcontroller from Atmel. The processor's specifications are summarized in Table 3.1. It uses Atmel's own AVR (8-bit) instruction set,

1. https://www.arduino.cc
2. https://www.arduino.cc/en/Main/ArduinoBoardUno
3. http://www.atmel.com/devices/atmega328p.aspx

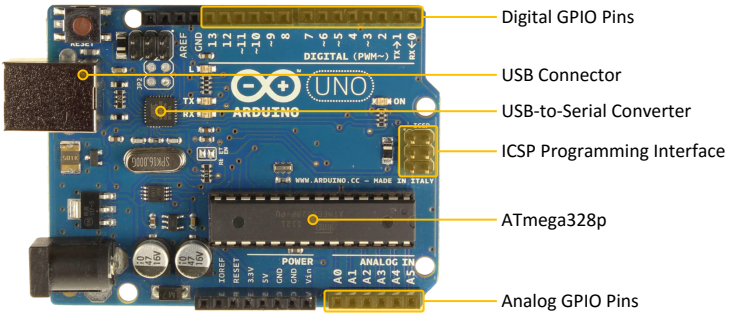


Figure 3.2: Arduino UNO prototyping board equipped with an ATmega328P processor. The board exposes six pins connected to the ADC peripheral and 14 digital only pins. The processor can be programmed through the In-Circuit Serial Programming (ICSP) interface or the serial connection (via the USB-to-serial converter).

based on a Reduced Instruction Set Computer (RISC) architecture. The clock speed can be set up to 20 MHz, whereas the Arduino uses a 16 MHz quartz crystal as clock source. This means that one instruction cycle on the processor roughly amounts to 63 ns.

The timer and ADC are the two peripherals necessary to build a robust communication system. The timers can be used to trigger actions with a certain periodicity, such as turning on and off an LED (modulating light) or sensing incident light. The ADC can convert an analog voltage to a digital value with a resolution of 10 bit. When using a reference voltage of 5 V, every ADC unit relates to approximately 4.9 mV, accurate enough for a system where the sensitivity of the LED is the bottleneck. An analog-to-digital conversion is not instant, but needs some time while the ADC is approximating the resulting voltage step by step. The conversion speed can be configured to take from milliseconds to a few microseconds, with reduced accuracy for faster conversion times. Eight of the 23 GPIO pins are connected to the ADC peripheral whereas six are accessible on the Arduino. Since there is only one ADC available, the analog pins are multiplexed and thus only one conversion can be processed at a time.

FEATURE	SPECIFICATION
Architecture	8-bit AVR
Operating Voltage	1.8 to 5.5 V
Current per Pin (max.)	40 mA
Clock Speed (max.)	20 MHz
GPIO	23
Timer	8-bit (two), 16-bit (one)
ADC	10-bit (one)
Serial Interface	USART (one)
EEPROM	1 kB
Flash Memory	32 kB
SRAM	2 kB

Table 3.1: ATmega328P specifications. The microcontroller uses the 8-bit AVR instruction set and can be clocked up to 20 MHz. It provides 23 GPIO pins and integrates timer, ADC and serial communication peripherals. Furthermore, 32 kB of program memory and 2 kB of Static Random-Access Memory (SRAM) is available.

The Universal Synchronous Asynchronous Receiver Transmitter (USART) interface allows for serial data exchange with other devices. Since modern computers do not have classic serial ports anymore, an intermediate device is employed to connect to a common USB port, and a standard serial port is emulated in software on the target machine. The Arduino already comes with a USB-to-serial converter that enables communication with other devices, e.g., for logging on a connected computer to gather measurement results or for debugging purposes. With the help of a bootloader, it is also possible to upload a program to the microcontroller using the serial connection. For more complex and larger programs, using the In-Circuit Serial Programming (ICSP) interface can speed up the program uploading process. It can be used to directly write the program memory without relying on a bootloader.

The ATmega328P offers 32 kB of flash memory, which is used to store the program and constant data. This is enough for programs of a decent complexity. Important for a communication system is the 2 kB Static Random-Access Memory (SRAM), which is reserved for dynamic data, stack, and heap during runtime. Since communication is based on mostly dynamic data (messages), the SRAM size is limiting the number of message buffers and their sizes, but it is enough for a prototype demonstrating the concepts developed in this thesis.

The processor can be powered with 1.8 to 5.5 V. The Arduino prototyping board provides a voltage regulator, supporting also higher voltages, aside from the 5 V available from the USB connector. Since the processor also allows for lower operating voltages, it can also be run directly from single cell batteries. The GPIO pins are able to source approximately 40 mA, enough to directly supply an LED, fitting the plan to use no intermediate circuitry.

Except from the Arduino UNO board, no other infrastructure offered by Arduino is used. As will be explained later in this chapter (Section 3.4), neither the Arduino Integrated Development Environment (IDE) nor the Arduino Libraries are used for the implementation of the software building blocks.

3.2.2 *Transceiver Front End*

As LEDs are based on the same principles as photodiodes, they also have similar properties [16]. Indecent light generates a photocurrent proportional to the light intensity, although less pronounced than in photodiodes. The LED can act as replacement for the photodiode and the usual circuits for forward (photovoltaic) or reverse bias (photoconductive) mode can be applied, but the photocurrent can also be measured without a dedicated circuit [16]. Figure 3.3 depicts such a light sensing process. Part 1 shows an LED with the anode (A) pin and cathode (K) pin. Part 2 and 3 show simplified replacement circuits [25] for an LED, consisting of a current source modeling the photocurrent, I_{photo}, with a capacitor, C_1, in parallel. To start a sensing sequence (Part 2), a voltage is applied in reverse bias. Reverse-biasing means that a positive voltage is applied to the cathode of the LED. This

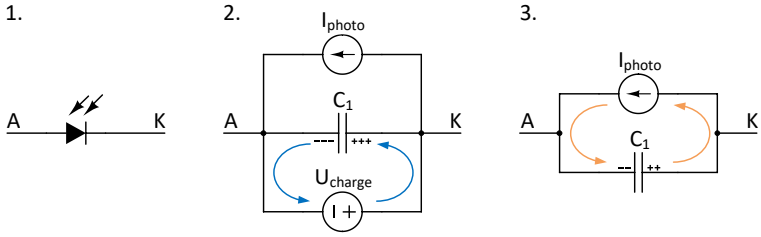


Figure 3.3: Simplified LED replacement circuit. Part 1 shows an LED with anode (A) and cathode (K). Part 2 and 3 show replacement circuits with the current source I_{photo}, modeling the photocurrent, and capacitor C_1. Part 2 demonstrates reverse biasing the LED and charging up the internal capacitance. In part 3, incident light is generating a photocurrent discharging capacitor C_1. The remaining voltage measurable over C_1 is proportional to the amount of light received.

charges the LED's internal capacitance (when looking at the replacement circuit). Indecent light will generate a photocurrent and discharge the capacitance. As the photocurrent is proportional to the light intensity, it is possible to infer the amount of light received by measuring the remaining voltage over C_1 (after a fixed time period).

This measuring sequence can be executed by directly connecting the LED's anode and cathode to a microcontroller pin each. It is not only possible to measure the photocurrent generated by the LED but also to operate it in its usual way when it is emitting light. The full process of emitting and sensing light while connected to a microcontroller is illustrated in Figure 3.4. Anode and cathode are connected to GPIO pins (the cathode is connected to an ADC-capable pin). Most microcontrollers support digital and analog behaviors for a certain number of pins. Part 1 shows the LED in light emitting mode. Both microcontroller pins are used in output mode. The pin connected to the anode is set to `HIGH` and the cathode's pin is set to `LOW` (ground) to drive the LED in forward bias mode, during which it emits light. Current (and thus LED luminosity) could additionally be limited with an appropriate resistor in series. Since the LED is operated with duty cycling, and

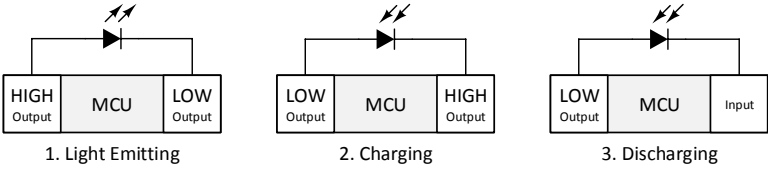
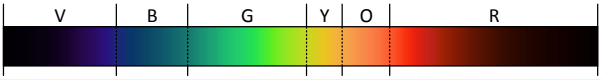


Figure 3.4: Microcontroller-based LED transceiver. Part 1 illustrates an LED in light emitting state. Both microcontroller pins are in output mode, anode set to `HIGH` and cathode set to `LOW`. While charging the internal capacitor (part 2), the LED is reverse biased; pins are still in output mode, but the polarity is inverted. Part 3 shows the discharging state. Photocurrent caused by incident light is discharging the capacitance again. The remaining voltage, proportional to the amount of light received, can directly be measured at the cathode pin with the connected analog microcontroller pin.

the current is also limited by the microcontroller itself, a resistor is omitted here. Part 2 shows the pin settings during the charge sequence. The anode is set to `LOW` and the cathode to `HIGH`, reverse biasing the LED and charging up the internal capacitance. In part 3, the cathode pin is set to input mode. The capacitance can now discharge via two paths: current leaks through the high impedance input pin and the photocurrent generated by incident light breaks down charges. After the discharge period, the remaining voltage can directly be measured using the analog pin connected to the cathode.

LEDs come in different colors, many form factors, and different housings. All these properties influence the light sensitivity. First, the sensing characteristics for LEDs of different colors is discussed.

Figure 3.5 shows the visible light spectrum. Each color is associated with a wavelength and frequency. Light is not only an electromagnetic wave, but can also be explained with particles called photons. Each color is also related to a photon energy E_p described by Equation 3.1. Hence, LEDs producing light of different colors generate photons with different energies. A photon represents the energy freed when an electron recombines with a hole (absence of an electron) [87] within the LED's semiconduc-



Wavelength [nm]	380	450	495	570	590	620	750
Frequency [THz]	789	668	606	526	508	484	400
Photon Energy [eV]	3.26	2.75	2.50	2.17	2.10	2.00	1.65

Figure 3.5: Visible light spectrum showing the colors violet (V), blue (B), green (G), yellow (Y), orange (O), and red (R) and the associated values for wavelength, frequency and photon energy. An LED of a certain color is only sensitive to light from the left part of the spectrum starting at its color's position. Figure based on Wikipedia's Visible Spectrum article.[4]

tor material. Applying a forward bias voltage to the LED pushes electrons and holes together and forces the emissions of many photons, creating light. When sensing light, this process is reversed. An incoming photon provides energy to create electron hole pairs, allowing the electron to move from the valence band to the conductive band. This provides free charges, which contribute to the photocurrent.

$$E_p = \frac{hc}{\lambda} \tag{3.1}$$

The energy needed to create an electron hole pair is the same that is released in form of a photon when an electron and hole combines (for the same material). Therefore, to induce a photocurrent in a certain LED, at least photons of the same wavelength, as can be produced by this LED, or shorter are required. This makes an LED only sensitive to light of the same color or colors of shorter wavelength. E.g., a red LED is sensitive to light from a red, green, and blue LED, but red photons do not have enough energy to induce photocurrent in a blue LED.

The light color emitted by an LED depends on different semiconductor materials and doping (intentionally added impurities) used. LEDs are available for all main colors from violet to red. White light, a combination of lights of different wavelength is a

4. https://en.wikipedia.org/wiki/Visible_spectrum

special case. LEDs producing white light can be built in two different ways:

First, wavelength converting materials like phosphor can be used. When phosphor is illuminated by blue light, it will emit yellow light. Mixing this yellow light with blue unconverted light is resulting in white light. Another method involves three different phosphor coatings that emit red, blue and green light when shown on by ultraviolet light [87]. Using blue or ultraviolet LEDs coated with phosphor can be used to produce white light but these LEDs do not have good sensing properties due to the phosphor coating and the short wavelength of blue and ultraviolet LEDs.

The second way to build white LEDs is more straight forward. Multiple LEDs of different colors can be built into one LED where the colors are mixed, providing white light. An example is an RGB LED, housing a red, green and blue LED with common cathode and three anodes or common anode and three cathodes for individual control. Enabling all LEDs will result in white light, but in most cases the three colors are still recognizable as well. An RGB LED can be used to generate all colors by combining the three colors at different luminosities. In addition, the part of the LED responsible for the red photons is a good candidate for light sensing since it is sensitive to light of equal or shorter wavelength than itself, which is the complete visible spectrum.

The sensitivity also depends on the housing built around the LED. Lenses shape the produced light beam and thus also widen or narrow the field of view of the LED, influencing sensitivity for different viewing angles. A diffuse housing distributes emitted light more uniformly but also scatters indecent light so that fewer photons reach the actual sensing area. Colored casings can be used to change the color of an LED but they also act as color filters when sensing light.

The main LED used throughout this thesis is a device from Kingbright[5]. It is a red LED with a 5 mm clear housing. With the integrated lens, it achieves a viewing angle of 20°. This generates a narrow beam of light, which is still compact after a few me-

5. https://octopart.com/l-7113sec-j3-kingbright-55402831

ters, increasing the photon concentration per area. But the lens also restricts the sensitivity for incident light. The LEDs require to be precisely aligned for longer distances to guarantee successful communication. Overall the selected LED worked well enough to support the proof of concept system presented in this thesis. Any other device, which fulfills the wavelength sensitivity property discussed earlier in this section and as long as enough photons can reach the sensing surface, will achieve similar results.

3.3 SOFTWARE BUILDING BLOCKS

The main contribution of this thesis is the design and implementation of a software-based PHY and MAC layer. All functionality is encapsulated into a software library called *libvlc*. Hence, the communication core is logically separated from a possible application. The library presents the communication service through an API supporting actions for creating communication channels (if multiple transceiver front ends are present), sending and receiving data packets, and for collecting statistics.

With *libvlc* it is possible to create a full-fledged communication system based on only the off-the-shelf parts described in Section 3.2, a basic microcontroller and an LED. The dependence on only a few hardware parts and the combination of transmitter and receiver into one transceiver device produces a low-complex and cost-effective system. The limited achievable data rates fulfill the requirements for the envisioned applications in the area of consumer electronics, toys, and the IoT. The software-defined nature of the system enables fast prototyping and provides a powerful tool for VLC research due to its flexibility. Already deployed systems using LEDs and microcontrollers can be enhanced with communication abilities by changing only software, and existing VLC systems can be improved via software updates.

In the following, the PHY and MAC layers are described and explained in detail, and the proposed API, visible for potential application, is defined. The *libvlc* version presented in this thesis represents the latest and most stable version. Many concepts and implementation details changed over time but are all well documented (in chronological order) [80–82].

3.3.1 *Physical Layer*

The PHY layer itself is composed of smaller building blocks. Important parts are sensing, device synchronization and FEC, described in detail later in this section. The key concept behind the PHY layer is the slotting infrastructure that separates illumination and communication and introduces simple modulation, sensing and synchronization.

Light Sensing and Modulation

The light sensing concept is explained with the aid of Figure 3.6 using the help of an oscilloscope. An LED is operated by a microcontroller as described in Section 3.2.2. One of the scope's probes is connected to the LED's cathode measuring the voltage versus ground. The resulting voltage is shown as the upper black curve labeled with *Light Sensing*. The LED is periodically reverse-biased

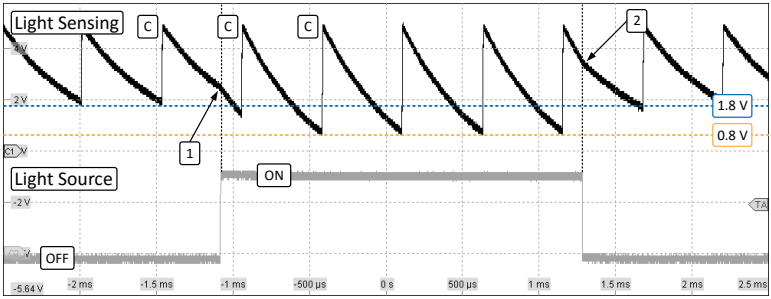


Figure 3.6: LED light sensing concept illustrated with an annotated oscilloscope screenshot. The LED is reverse-biased (charged) periodically (black channel), labeled with C. After charging, the LED is discharging itself reaching 1.8 V before being charged again. The second channel (gray) shows the state of a light source, first disabled, and then enabled after a certain time and last disabled again. While the light source is enabled, the additional photocurrent is discharging the LED's capacitance further down to 0.8 V. Label 1 and 2 emphasizes the kink in the discharge curve caused by the enabling/disabling of the light source.

(charged) for a short instance, labeled with *C*. After charging, the LED is given time (500 µs) to discharge itself again. The heavy discharge visible is mostly due to the oscilloscope, drawing current to feed its ADCs. Without a connected probe, the discharge will depend largely on the photocurrent generated by the ambient light. Little current will also be drawn by the high impedance input pin, and the drifting electrons and holes in the LED are also breaking down charges. The second channel (gray) of the oscilloscope is showing the state of a light source. It is either enabled or disabled. While it is enabled, the photocurrent further accelerates the discharge, leading to a voltage level (before the next charge) of 0.8 V. Compared to the voltage level when the light source is off (1.8 V), there is a clear difference visible. Label 1 and 2 are pointing at the kinks in the discharge curve that are caused by the state changes of the light source.

The microcontroller can now be used to measure the remaining voltage at the end of each discharge cycle. The PHY layer protocol is constructed around these charge and discharge cycles as described in the following section. The information from comparing voltage levels at the end of each cycle will be used for synchronization and data symbol decoding.

Slot-based Communication and Illumination

Aside from communication, the main purpose of lighting devices, the illumination, should be maintained. A VLC system should be able to hide the communication within the illumination and act at the same time as a lighting device. The proposed sensing technique requires the LED to be turned off while receiving light. Therefore, communication and illumination needs to be multiplexed in time as illustrated with Figure 3.7. During an Illumination (ILLU) slot, the LED is enabled and contributes to lighting, while the Communication (COM) slot is responsible for various light sensing sequences, called intervals. The slots are each 500 µs long, resulting in a 1 kHz on and off pattern. The fast light changes are not visible to a human observer. The eye cannot resolve fast on and off patterns at a certain frequency, which is called the flicker fusion threshold or critical flicker frequency [45,

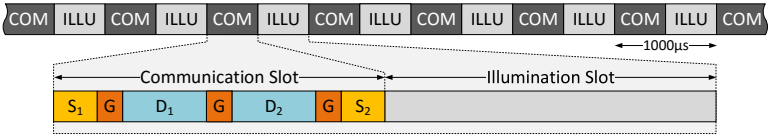


Figure 3.7: Slot-based PHY layer communication protocol. ILLU and COM slots alternate at a frequency of 1 kHz, too fast for a human observer to notice the COM slots, during which the LED is turned off. The COM slot consists of several shorter light sensing intervals.

46]. For the human eye, the threshold was determined at about 60 Hz [27]. The COM slot duration of 500 µs is chosen to fit in the proposed sensing intervals including microcontroller processing time. The ILLU slot duration can be extended for higher light intensity (duty cycling) at the cost of fewer COM slots per time. To demonstrate the concept and for simplicity, both are kept the same length for this discussion.

A COM slot is partitioned in short light sensing intervals that are responsible for synchronization and data transmission and reception. The synchronization (S_1, S_2) and data (D_1, D_2) intervals are light sensing sequences as shown in Figure 3.8. The start of a sensing sequence is marked with a *C*, indicating that the LED is being charged. At the end, labeled with *M*, the remaining voltage is measured. The two synchronization intervals at the start and at the end of the slot are used to align the COM slots of communicating devices. This synchronization process is explained later

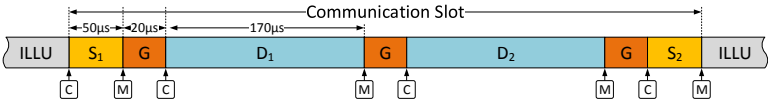


Figure 3.8: A COM slot is partitioned into shorter light sensing intervals. S_1 and S_2 are used for synchronization, D_1 and D_2 for data encoding, and the guard intervals G protect the other intervals from light leaking caused by inaccurate synchronization. At the beginning of a sensing interval, the LED is charged (C) and at the end, the remaining voltage is measured (M).

in this section. The guard intervals G prevent light leaking into other intervals in case of inaccurate synchronization. The data slots are either used to receive or to transmit data.

Modulation Scheme

The applied modulation scheme is based on OOK and PPM. OOK is a basic version of Amplitude-Shift Keying (ASK) where data symbols are represented as the presence or absence of light during a certain time, which can be detected using the proposed methods. Also the transmitter implementation is straight forward, since a digital GPIO pin is enough to enable or disable light output. More elaborate modulation schemes like Phase-Shift Keying (PSK) require more powerful microcontrollers with a Digital-to-Analog Converter (DAC) and additional peripherals to generate the necessary waveforms. In PPM the position of on-pulses is used to encode digital data.

Figure 3.9 illustrates how OOK and PPM are applied to the PHY layer protocol. The transmitting device enables light output during the D_1 or D_2 intervals of its COM slots (light gray area) while the LED stays off for the rest of the slot (dark gray area). A receiving device measures its two data intervals and compares the two values. If D_1 is significantly larger than D_2, meaning that

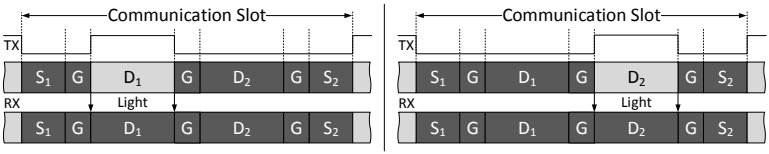


Figure 3.9: Modulation scheme based on OOK and PPM. The Transmitting (TX) device either enables light during D_1 (left side), encoding a bit 0, or during D_2 (right side), encoding a bit 1. Light output is denoted with a light gray area, and the absence of light emissions is signalized with a dark gray color. In addition, the light signal is shown on top of the TX device. The Receiving (RX) device senses light during D_1 and D_2 and compares the results. Measuring more light during D_1 than during D_2, results in a bit 0, else a bit 1 is decoded.

there was more light present during D_1, the received symbol is decoded to a bit 0. If D_2 is larger than D_1 the received symbol results to a bit 1. Because there are always light emissions involved in transmitting a bit 0 or 1, it is straightforward to derive a medium busy or idle scheme CCA. As long as light is detected in one of the data intervals, there is a transmission going on and the medium is busy. If D_1 and D_2 are very close, only ambient light is present and the medium is therefore idle.

Using the data intervals for light output during a transmission increases the amount of light emitted per time (when compared to the case when no data is transmitted and therefore no light is emitted during a COM slot) for the duration of the transmission. This is perceived by a human observer as an increase of brightness. To get rid of the unwanted visible brightness changes (flickering), the average light output must remain constant while transmitting, receiving, or idling. To achieve this, the same amount of light emitted during one of the data intervals must be compensated during the following ILLU slot.

The compensation method is shown in Figure 3.10. Light gray areas denote enabled light output and dark gray areas mark disabled light output. The first row shows a device while idling or receiving for reference, resulting in a 50 % duty cycle. The device shown in the second row starts transmitting and thus adds light

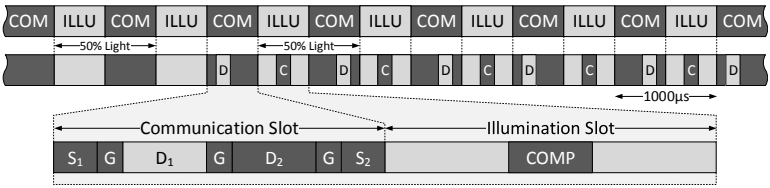


Figure 3.10: Flicker avoidance while transmitting. The first row shows a device while idling and transmitting with constant light output at 50 % duty cycle. The second row shows a transmitting device, which adds light output during the COM slot to modulate a data bit. This additional light output is compensated during the next ILLU slot to achieve the same duty cycle, avoiding a brightness change.

output to the COM slot, increasing the light duty cycle, which results in brightness changes. Removing the same amount of light (switching the light of for the same duration as a data interval) is decreasing the duty cycle again and compensating for the brightness changes. This prevents flickering during data transmissions and successfully hides communication from the human eye.

Continuous Synchronization

Looking at the modulation scheme makes it clear that accurate synchronization of transmitter and receiver is necessary. The ILLU and COM slots need to be aligned, so that the data intervals of communicating devices mostly overlap. The synchronization procedure needs to be simple enough to run on the microcontroller without consuming too many valuable processing cycles. A common approach in communication system is to use a Phase-Locked Loop (PLL) to match the phases of two signals. A PLL is usually implemented directly in hardware and relies on a sampled input signal. Since the LED sensing mechanism does not support direct sampling of the modulated light signal and the synchronization is required to be implemented in software, the following, much simpler but sufficient synchronization procedure is presented.

There are two reasons why synchronization is necessary. First, devices are switched on at random times and therefore have a phase offset from the beginning. Second, internal resonator or quartz crystals do not have exactly the same frequency and drift over time. This drift is higher for resonator circuits and low-cost crystals. Common practice is to synchronize devices at the beginning of a data frame with a preamble. After the preamble, devices are expected to be in sync for the duration of a frame. For devices with inexpensive clock infrastructures, a single synchronization phase (preamble) is not enough when transmitting longer packets. The devices will drift apart while transmitting and receiving. The method presented in this thesis provides continuous synchronization, also while sending or receiving a frame.

Figure 3.11 illustrates the synchronization procedure. Devices A and B are out of sync. Their COM and ILLU slots are not aligned. While their phase is offset by ϵ, the previous or next ILLU slot

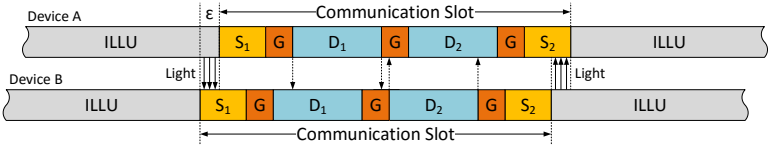


Figure 3.11: Device synchronization process. Two devices are out of sync by ϵ. Device A measures additional light during S_2 (from Device B's ILLU slot) and no additional light during S_1, meaning that the COM slot started too late. Device B receives additional light during S_1 and no light during S_2, which means that the COM slot started too early. The synchronization process compares S_1 and S_2 and as long as they differ, the phase is corrected step by step until the devices are in sync.

overlaps with a synchronization interval S_1 or S_2. The synchronization intervals behave the same as the data intervals. The LED is charged at the beginning and in the end, the remaining voltage is measured to deduct the amount of light received. Listing 3.1 shows the algorithm that uses the measured light values to continuously synchronize devices. After each COM slot, the light values for S_1 and S_2 are compared. If their difference is smaller or equal to a certain margin, nothing happens. Similar light values for S_1 and S_2 mean that no light is leaking from neighboring ILLU slots, hence the devices are synchronized. If the difference between S_1 and S_2 increases, one of the synchronization intervals is receiving more light than the other, which means that the devices are out of sync. One interval overlaps the following or previous ILLU slot, receiving light, while the other is taking place during a COM slot, receiving no light. The amount of light received during S_1 and S_2 can also be used to determine the direction of the phase correction. If more light is received during S_1 than during S_2, the previous ILLU slot overlaps and the change from ILLU to COM slot happened too early, requiring the system to shift its following slot changes towards right. If light values for S_2 are higher than for S_1, the slot changes need to be shifted towards left. The shifting of the signal is achieved by slightly adapting the duration of the following ILLU slot. When shifting left, the slot duration is short-

Listing 3.1: Synchronization algorithm (pseudo code). The algorithm is run after each COM slot.

```
/* if S1 and S2 differ more than margin */
if (abs(S1 - S2) > margin) {
    if (S1 > S2) {
        /* too early */
        shift towards right
    } else {
        /* too late */
        shift towards left
    }
}
```

ened and when shifting right, the slot is prolonged. The shifting step size is in the range of a few microseconds. The higher it is, the faster an initial synchronization is reached. Using smaller steps increases the stability of continuous synchronization.

Figure 3.11 shows also that if two devices are out of sync, the phase correction is always applied so that the devices eventually converge in synchronization. The measured light values for S_1 and S_2 are always the opposite (for two devices), forcing the algorithm to shift their slot changes towards each other. This prevents that the devices shift their signals in the same direction, never reaching synchronization, or shifting them apart from each other, which results in a false equilibrium. The light values for S_1 and S_2 are not only similar if the COM slots (and ILLU slots) are aligned, but also if the signals have a phase offset of one slot (ILLU slot aligned with a COM slot). This could happen by chance at device startup and prevent the synchronization algorithm from immediately correcting the phase offset due to the similarity of the detected light values for S_1 and S_2 (both receiving the same amount of light from the aligned ILLU slot). But as soon as the two signals slightly move apart (due to the clock drifts), the synchronization mechanism will force the correct slot alignment.

The synchronization algorithm is very simple, consuming only a few processor cycles, and is directly implemented in software as part of the PHY layer. It is used to correct the initial phase off-

set and is also continuously running while idling, transmitting, or receiving. Due to the sensing mechanism and the algorithm's simplicity, the synchronization is not perfect. The phase offset is always changing within a few microseconds, also depending on the light intensity. The more light, the more distinct the sensed intervals are and the more accurate the synchronization is. For devices further apart, less light is received and the resulting synchronization is less stable. Successful communication is still possible, given the synchronization inaccuracy is within the duration of a guard interval G. The guard intervals are located between synchronization and data intervals and between the two data intervals. The dashed arrows in Figure 3.11 show that although the two devices are offset, a data slot cannot leak light into a synchronization or data interval and possibly interfere with synchronization or provoke bit flips. The guard intervals are not used to measure light but provide a buffer zone between the active sensing intervals.

The proposed slot and interval structure and synchronization process was designed with networking in mind. When coordinating shared medium access in a network, a solution to detect communication activity is required. Using the same medium at the same time for another purpose, namely illumination, makes shared medium access more demanding. The strict separation of communication and illumination into COM and ILLU slots and the synchronization provides a common place in time where communication can be detected independently from illumination. When two devices are synchronized, communication can only happen during aligned COM slots where it can be detected using the data intervals D_1 and D_2. Further, two or more in sync devices produce the same light pattern as a single device. Hence, a synchronized network looks and behaves the same as one device when observed by another device joining the network. The arriving device can synchronize to the network as it were a single device. The presented simple synchronization method not only provides reasonable device synchronization but also scales with the number of network nodes.

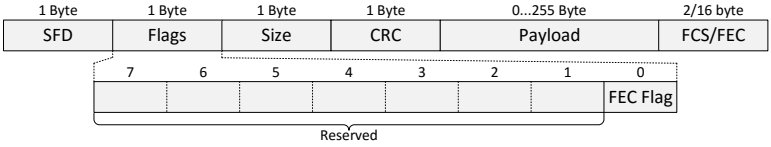


Figure 3.12: PHY layer frame header. It consists of four fields with a size of 1 B each. The start of a frame is detected by looking for a predefined Start Frame Delimiter (SFD) byte, followed by a byte reserved for future bit flags. One flag is occupied for the FEC state. The PHY layer payload size is defined by the size field. In addition, the 3 B are protected by an 8-bit Cyclic Redundancy Check (CRC).

Physical Layer Frame Format

The PHY layer frame starts with a 4 B header followed by the payload containing the MAC layer frame as shown in Figure 3.12. To detect the beginning of a new frame, a Start Frame Delimiter (SFD) is introduced. The SFD is a predefined bit sequence with a length of 1 B. Since a 0 and 1 bit is well-defined, and not only defined as presence or absence of light, any bit sequence works as an SFD. If an SFD is found in the continuously decoded bit stream, the decoder switches to receiving mode and receives the following three PHY header bytes. The first real header byte contains bit flags and is mostly reserved for future extensions. Still, one of the flags is used for FEC. It informs the frame receiver whether FEC is enabled for the PHY payload or a Frame Check Sequence (FCS) is used. The Flags byte is followed by another 1 B field, defining the size of the PHY payload. The field's size limits the maximum payload to 255 B. Using larger payload sizes is not reasonable at this stage due to the memory constraints of microcontrollers and the transmission duration of already approximately two seconds (at 1 kb/s) for a PHY layer frame of maximum size. The flags and payload size byte is protected by an 8-bit Cyclic Redundancy Check (CRC) sequence. It is calculated using an already implemented (and optimized in assembler for AVR) *avr-libc* method[6],

6. http://www.nongnu.org/avr-libc/user-manual/group__util__crc.html

developed by Dallas Inc.[7] (now Maxim Integrated). If the transmitted CRC does not match with the CRC computed at the receiver, the frame is ignored.

The PHY layer header is followed by a 0 to 255 B payload that contains the MAC layer frame including the application data. The transmitter decides (e.g., depending on the payload size) whether to add a 16-bit CRC sequence or additional redundancy for FEC. The transmitter also sets the FEC flag in the PHY according to the used method. The FCS uses again an already implemented and optimized *avr-libc* method and is used by the receiver to validate the received payload. With the addition of 16 B redundancy and some computation at the transmitter and receiver, the receiver is able to not only validate the payload but also to correct up to eight byte errors in the received data. The used FEC method is explained in the following section.

Forward Error Correction

The transmitter decides, based on a configurable FEC threshold, whether to apply FEC or not. For PHY layer payload sizes equal to or below the FEC threshold, an FCS is used since it is more efficient to retransmit short data frames. For payload sizes above the FEC threshold, Reed-Solomon error correcting codes [70] are applied. They are widely used in consumer technologies and communications, provide byte-level error detection and correction, and are reasonable to implement in software on a microcontroller.

The unit on which all operations take place is called codeword. The maximum codeword length n is given by Equation 3.2. Where s represents the symbol size. The length of uncoded data k is determined by Equation 3.3 and the number of provided parity symbols is $2t$. This allows to detect and correct up to t symbol errors. If the error positions are known, $2t$ errors can be corrected. It is practical to use a symbol size $s = 8$, which is equal to a byte, allowing to directly find and correct byte errors.

$$n = 2^s - 1 \tag{3.2}$$

$$k = n - 2t \tag{3.3}$$

7. https://www.maximintegrated.com/en/app-notes/index.mvp/id/27

Using $s = 8$ gives a codeword length of $n = 2^8 - 1 = 255$. The number of parity symbols can be defined when compiling *libvlc*. By default it is set to $2t = 16$ resulting in a maximum data length of $k = 255 - 16 = 239$. These constants produce an FEC code generally written as $RS(255, 239)$. It limits the maximum PHY payload size to 239 B. Larger payloads must be split up in chunks of size k symbols which are each processed as a codeword. Due to the small difference to the previous maximum payload size of 255 B, the PHY layer only supports one codeword per frame. The encoding and decoding algorithms can directly be applied to payload sizes smaller than k. In theory, the payload is padded with zeros up to the size of k, which are omitted when transmitting and then added again by the receiver for processing.

The encoder and decoder implementations are optimized to run on a microcontroller with memory limitations. Buffer sizes and copy operations are minimized and calculations are sped up by static lookup tables for the logarithm and exponentials for every value in the Galois Field[8] of 2^8. The first stage of the decoder algorithm (syndromes calculation to determine error locations) is computed step by step (while receiving) and is continued as soon as another received byte is available, improving on the overall processing time for a completely received PHY layer frame. The additional time needed for processing after receiving a frame can influence higher layer protocols as discussed in Section 3.5. The current *libvlc* implementation can find and correct t byte errors ($2t$ defined at compile time), and it only supports correcting errors if their locations are known. Since the complete PHY layer logic is part of the same software it can be upgraded without effort to mark received and possibly corrupted bytes. As discussed earlier, the bit-level decoder is comparing voltage values and decides to produce a bit 0 or bit 1. Close decisions can be recorded and fed into FEC to correct $2t = 16$ errors.

8. https://en.wikipedia.org/wiki/Finite_field

3.3.2 *Medium Access Control Layer*

The literature almost exclusively discusses point-to-point VLC systems. The system proposed in this thesis supports the interconnection of multiple devices. Due to its unique slotting mechanism, separating communication and illumination, and providing a wireless synchronization method, a contention-based protocol based on the IEEE 802.11 MAC standard [33] can be applied to handle medium access.

Listen-Before-Talk

In a network where many devices share a common communication channel (slice of the spectrum) a MAC protocol takes care of handling medium access so that communication attempts do not interfere with each other, wasting precious air time. A common approach to handle shared medium access is to employ a CSMA/CA protocol where collisions are avoided by listen-before-talk and random backoff times. Listen-before-talk means that a transmitter observes the channel before sending, and if the medium is already occupied, the transmission is suspended. In wireless radio systems, detecting energy (or the absence of energy) in a certain frequency band and determining the channel state is straight forward. In a VLC system where the same channel is used for illumination and communication, it is more complicated to detect a busy channel (caused by communication and not illumination). The PHY layer presented in Section 3.8 has been designed with this problem in mind. The separation of illumination and communication and the synchronization of ILLU and COM slots for all devices in range defines a common place in time where all participating devices can expect only communication without being affected by the illumination part of the system. The channel is detected busy if the difference of measured light during D_1 and D_2 is above a configurable threshold, meaning that either communication is ongoing or any other interference is present.

The MAC protocol is implemented as an independent layer on top of the PHY layer. An incoming data frame is first processed by

the PHY layer where a byte buffer is assembled from the incoming bits. After checking the frame for consistency, either with the help of FCS or FEC, the PHY layer payload (containing the MAC frame) is forwarded to the MAC layer for further actions. When transmitting a packet, the application hands the data packet over to *libvlc*, which forwards it as frame payload to the MAC layer where the following protocol steps are executed (protocol parameters and default values are summarized in Table 3.2):

- The payload forwarded by *libvlc* is prepended with the MAC header, containing 4 B of additional data more closely described in a later section.
- Before forwarding the MAC frame to the PHY layer where it is eventually transmitted, the channel is tested (similar to carrier sensing in IEEE 802.11) to recognize ongoing communication or other interference. If the channel has been clear for a certain timespan called Distributed Interframe Space (DIFS), the random backoff process starts. Two parameters span the Contention Window (CW). The lower threshold is defined by CWmin and the upper threshold is set by CWmax. The actual CW (size) starts at CWmin. The backoff process starts with the generation of a random backoff time bound by the current contention window CW, hence a number between 0 and CW. One backoff time unit relates to the timespan between the starts of two COM slots (unit) which is 1 ms in this setup. While counting down the backoff timer, the channel is successively sensed during each COM slot. The MAC frame is handed to the PHY layer after the backoff time is expired.
- Sensing a busy channel before starting the backoff timer or during counting it down interrupts the backoff process. If the busy channel is caused by another transmission and the SFD and the following PHY header can be decoded, the backoff timer is resumed after receiving the complete packet plus an additional DIFS. Is the channel busy due to interference, the backoff is continued after detecting no signal for the duration of a DIFS. Resuming the backoff timer instead

PARAMETER	DESCRIPTION	DEFAULT
SIFS	waiting time before ACK	4 units
DIFS	waiting time before data frame	8 units
CWmin	contention window starting value	32 units
CWmax	maximum contention window	1024 units
ACKto	ACK timeout	84 ms
RTNS	number of retransmissions	3

Table 3.2: Available MAC layer parameters. SIFS is the time always waited before sending an ACK and before a data frame, the transmitter pauses for the duration of DIFS. The contention window starts from CWmin and is doubled on every transmission failure until CWmax is reached. A unit is defined as the timespan between two COM slot beginnings. Within the ACK timeout an ACK control frame is expected and if not received, the current data frame is resent at most RTNS times.

of reseting it provides some fairness for the medium access, since it allows all devices to transmit eventually.

- When receiving a MAC frame from the PHY layer and the destination field in the header matches the device's address, an Acknowledgment (ACK) frame is transmitted. The ACK is a control frame and consists of only the MAC header without payload. The frame is sent out after a Short Interframe Space (SIFS), which is always shorter than DIFS. This prevents a collision with other data frames and gives the ACK a higher priority.

- The transmitting device expects to receive an ACK within the ACK timeout (ACKto) period. Receiving no ACK means that the sent frame collided with another frame (when backoff timers expire at the same time) or that the packet could not be received correctly due to interference or bad reception and therefore no ACK was generated at the receiver's end. It can also happen that the ACK is generated but never reaches its destination, again due to interference or bad re-

ception. If no ACK is received, the process is repeated until the number of Retransmissions (RTNSs) for the same frame is reached. For every unsuccessful trial, the current contention window is doubled until CWmax is reached. A successful transmission resets the contention window back to CWmin.

Figure 3.13 illustrates the described MAC process. Part 1 shows a successful data frame transmission answered with an ACK. First the channel is sensed, after being clear for DIFS a random backoff value within the contention window is generated, after which the data frame is transmitted. The receiving device waits for a SIFS before sending the ACK. In part 2, the data is sent after a backoff process, but due to interference or collision, matching the CRC at the receiver fails. Therefore no ACK is generated at the receiver and the transmitter starts the backoff process again (with an extended CW) after the ACK timeout to resend the lost data frame.

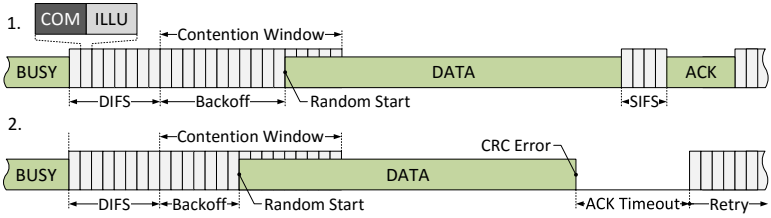


Figure 3.13: MAC process illustrated by means of examples. Part 1 shows a successful data frame transmission after a backoff phase answered with an ACK. Part 2 demonstrates the protocol behavior in case of a frame loss. No ACK is issued by the receiver and the transmitting station starts another backoff process after the ACK timeout to resend the last frame. The units used in the backoff process (also for SIFS and DIFS) consists of an ILLU and COM slot and therefore the timespan between two COM slot beginnings.

Forward Error Correction and Acknowledgment Timeout

The FEC is part of the PHY layer. If FEC is enabled by the transmitter, the frame is first tested for consistency and possible errors are corrected, provided that the error number lies within the code's capabilities. If a correct payload can be decoded, it is handed over to the MAC layer where an ACK is generated. The time between receiving a complete frame on the PHY layer and sending the resulting ACK therefore heavily depends on the FEC runtime.

Figure 3.14 plots the time until an ACK is received by the transmitter depending on number of errors and payload size. The results show that the ACK time increases for higher error numbers and larger frame payloads. The ACK time is prolonged due to higher FEC processing load for larger packets and higher er-

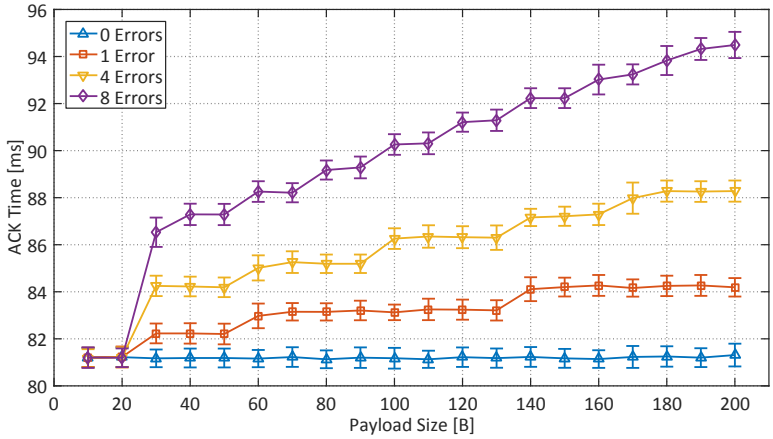


Figure 3.14: Time passed until a device receives an ACK for a transmitted data frame. The ACK time is set in relation to frame payload size and introduced byte errors. Since the FEC codes operate on a byte level, an error means at least one flipped bit in a byte. The ACK time increases for higher payload sizes and higher error numbers. The additional time is introduced at the frame receiver where the ACK is generated, due to the FEC processing time. The error bars indicate the standard deviation.

ror numbers. For these experiments, the FEC threshold was set to 30, enabling FEC only for PHY payloads larger or equal to 30 B, clearly visible in the plot. For a 200 B payload and 8 introduced errors (which is the maximum number of errors correctable for the tested configuration), the ACK time amounts to more than 94 ms. Per unit, one bit can be transmitted, the ACK including all headers and SFD equals to 10 B or 80 units. In the used setting where one unit is 1 ms, the total air time for an ACK is 80 ms. Therefore, for an ACK time of 94 ms the time passed between completely receiving a frame and sending out the ACK results in 14 ms, considerably longer than the budgeted SIFS of 4 units.

This problem can be solved in several ways. First, SIFS can be increased to 96 units covering the maximum FEC processing time. Second, the ACK can be sent earlier, while still running FEC. The time consuming part in the FEC algorithm is locating the errors and correcting them. The time needed to compute the number of errors detected is independent of the number of errors and payload size and can be computed within the 4 units of a SIFS on the used hardware (ATmega328P). If the resulting error number is within the bounds where the complete frame can be reconstructed, the ACK can already be dispatched, before patching the faulty payload. To successfully construct and address an ACK, the source address field of the received frame's MAC header must not be erroneous, so that it can be sent to the correct destination. Otherwise, the ACK is sent to the wrong receiver and the original sender will trigger a retransmission. The probability that one out of k uniformly distributed errors in a payload of n bytes falls on a specific byte can be calculated with Equation 3.4.

$$P_{n,k} = \frac{\binom{n-1}{k-1}}{\binom{n}{k}} = \frac{k}{n} \tag{3.4}$$

Figure 3.15 plots the probability for different error numbers and payload sizes. In addition to the stated payload size, a 4 B MAC header and 16 B of FEC redundancy are used for the calculation. For higher payload sizes where losing a packet due to the damaged source address is more serious, probabilities are below 5 %. Even for medium payload sizes, the probabilities are still below 10 % for 8 possible errors. Furthermore, the probabilities shown

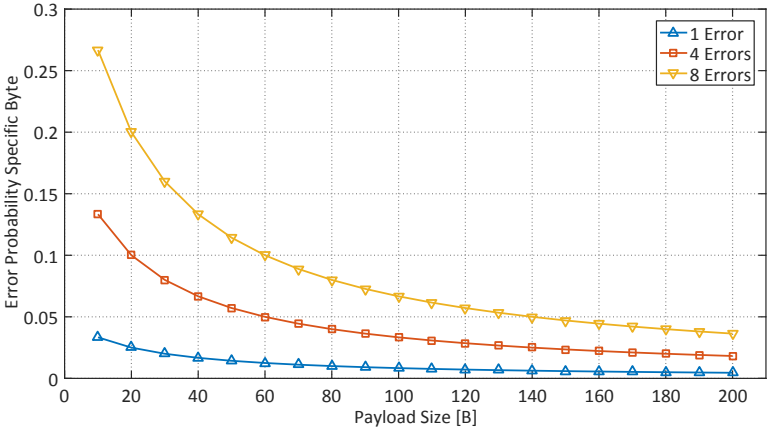


Figure 3.15: Probability that, for a given number of uniformly distributed errors and payload sizes, one error falls on a specific byte. For the computation, the payload is complemented with 4 B MAC header and 16 B FEC redundancy.

in Figure 3.15 are based on the assumption that a specific number of errors exists. To get the definite probability for a defective source address, it has to be multiplied with the probability that a certain number of errors exists, leading to even lower values. These calculations show, that the probability is reasonable low, that a data frame is being lost due to wrongly addressed ACKs. The third, last, and most useful solution proposed is to use a similar but more powerful microcontroller, introduced later in this thesis.

Hidden Station Problem

The hidden station or hidden terminal problem [91] is a well-known issue in radio communication especially where a listen-before-talk MAC scheme is employed. For LED-based VLC systems it is even a bigger problem because most light sources are very directive and additional optics further narrow light beams and restrict the field of view.

Figure 3.16 depicts an example for a hidden station scenario. Devices A, B and C are equipped with an LED as sender and

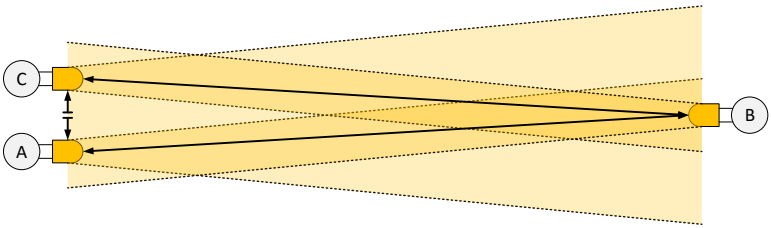


Figure 3.16: Hidden station problem. LEDs with a narrow viewing angle reach further but are more vulnerable to the hidden station problem. Stations A,B and C are all equipped with a narrow beam LED. The light cones from A and C overlap with B and therefore enable B to receive data from both communication partners. Reverse, B can also reach A and C, but between A and C, no communication is possible. Communication attempts from A and C have a high probability to collide since their channels are hidden from each other.

receiver. The light beams of A and C both reach B's receiving LED and therefore enable communication from A and C to B. In reverse, B can also communicate with A and C due to the overlapping field of view. A and C however do not see each other although being arranged close to each other. The standard CSMA/CA MAC protocol fails in this scenario. The following simple scenario shows that the protocol breaks: A wants to transmit a data frame, listens to the channels, sees a free channel for the duration of DIFS and starts transmitting. At the same time, C is already transmitting, but since its communication is hidden from A, A could not recognize that the channel actually was busy. The two data frames collide and B will not be able to decode the incoming data. While both A and C have data to send, no reasonable and stable communication from A and C to B is possible. An advantage of a VLC system over a radio system is that due to the visibility property of the communication range, hidden stations can easily be identified.

An approach to mitigate the collision probability for hidden station scenarios is to extend the existing MAC protocol with an RTS/CTS scheme [36, 38], also implemented by the IEEE 802.11 standard [33]. The proposed MAC protocol is enhanced with two ad-

ditional control frames, an RTS frame and a CTS frame. The RTS packet consists of the MAC header plus one additional byte payload. A device with a data frame in queue first sends an RTS with the data frame's size as payload to the data frame's receiver. The RTS competes for the channel like a standard data frame: the device waits for DIFS and if the channel is clear afterwards, it starts the backoff process. After a successful backoff, the RTS is sent to its receiver, which is replying with a CTS. It also consists of the standard MAC header with an additional byte of payload (copied from the RTS). The CTS is directly sent, after waiting for a SIFS, without a backoff process as it is the case for an ACK. If a CTS is issued, the requesting device successfully reserved the channel and can transmit its data frame after receiving the CTS and after waiting for another SIFS. Any other device overhearing the CTS is reserving the channel for the duration of the requested data frame, calculated from the size value sent with the CTS. If no CTS is received after a timeout, the RTS is retransmitted for the configured number of retransmission. If no RTS gets through, the current data frame is dropped.

This scheme introduces additional overhead but also prevents collisions of data frames and therefore the loss of precious channel time. Back to Figure 3.16. If device A wants to send a data frame, it first sends an RTS. Since the RTS is much shorter than an average data frame, the probability that it will collide with an RTS from device C is low. If the two RTS from A and C collide, only little channel capacity is lost and another collision is even less probable for the RTS retransmissions after increasing the CW. On receiving the RTS, device B is issuing a CTS which can also be overheard by C. Device C knows now that A reserved the channel for the following data frame and therefore will defer for the calculated airtime, although it cannot sense a busy channel during this time.

The protocol extension is implemented as part of the MAC layer of *libvlc*. Since the additional overhead is not reasonable for short packets with a lower collision probability, triggering the transmission of RTS is based on a payload threshold. Only data frames with a MAC payload size equal or larger than specified by the RTS threshold will lead to an RTS/CTS handshake.

Frame Format

This section further explains the MAC header and summarizes the format of data and control frames. The MAC frame is shown in Figure 3.17 together with the encapsulating PHY frame. The MAC header consists of four additional 1 B fields. The control field is used to identify frame types and to mark a frame as a retransmission. The remaining bits are reserved for future protocol additions. Half of the control byte is used to determine the frame type, distinguishing between Data, ACK, RTS, and CTS frames. There is still room for additional frame types since only four out of 16 possible identifiers are used. The source field stores the MAC address of the transmitting device and can be used by the receiver to address replies, e.g., an ACK or CTS frame. The destination field labels the frame's receiver. It is used by the MAC layer to filter received frames. Only frames with matching destination address are accepted, all other frames are ignored. The sequence number consecutively numbers every new outgoing frame. The sequence number is used by the receiver to sort out frame duplicates, which can happen when frames are successfully received but are still retransmitted because the ACK was lost. The MAC header is followed by a MAC payload with a maximum of 200 B as discussed earlier.

3.3.3 *Application Layer*

The described PHY and MAC layer are encapsulated into *libvlc* together with additional infrastructure for serial communication, logging and GPIO abstraction. For measurements to evaluate the PHY and MAC performance or for basic device-to-device communication, no additional networking layers are needed. The program logic can directly be implemented on the same microcontroller using the API exposed by *libvlc*.

The core functionality is summarized in Listing 3.2. The API can be structured into three groups. The first three methods belong to the *control* group. The method `vlc_init` initializes all library parts including PHY and MAC layer. Calling `vlc_start` enables the timer interrupts used to drive the PHY layer. Calling `vlc_process`

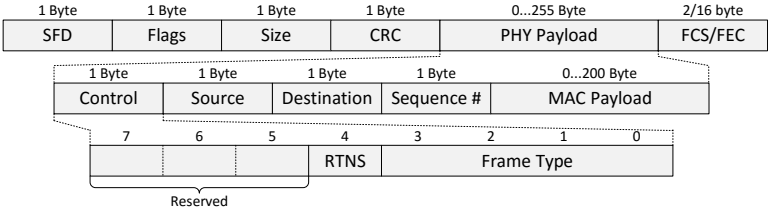


Figure 3.17: MAC frame structure as part of a PHY layer frame. The MAC header is built from additional 4 B. The source field contains the address of the transmitting device and the destination field is set to the receiving station's address. The control field's four lower order bits identify the different frame types, the following bit determines an RTNS, and the rest of the byte is reserved. The MAC payload with a maximum size of 200 B is appended to the MAC header.

will process all pending tasks for the MAC layer, FEC, logger and all callbacks, as they are not handled during the timer interrupt context of the PHY layer. To start processing as soon as tasks are available, this method is usually called during the application's main loop.

The following three methods are part of the *callback* group. A callback method for incoming messages is registered through `vlc_register_rx_cb`. The registered function will be called from the MAC layer after frame processing has finished and provides a pointer to the payload and the payload size. The data needs to be copied immediately so that it is not overwritten by the next incoming message (memory is a scarce resource on a microcontroller). With the next method, `vlc_register_tx_cb`, a hook for the TX event can be registered, triggered when the PHY layer finished transmitting a frame. It is called for all messages, also retransmissions and control frames. This callback can be used to log statistics during measurements. The method registered through `vlc_register_tx_done_cb` is called when a message dispatched using the API is completely handled, meaning either reception was confirmed with an ACK or the maximum number of retransmissions was reached. This callback is helpful when implementing saturation traffic for measurements. Due to limited

memory on microcontrollers, the outgoing data queue can only handle a single packet, hence when the TX done event is triggered, the next packet can overwrite the current data in the queue.

The last three methods build the *action* group. An LED is abstracted in *libvlc* as a communication channel. With the method `vlc_add_channel`, a new communication interface is initialized. It connects the hardware (LED pins) with the software-based PHY

Listing 3.2: Basic functionality exposed by *libvlc's* API. Only the core functionality is listed. The API can be structured in three groups: The first three methods are part of the *control* group, followed by the next three being part of the *callback* group, and the last three fall in the *action* group.

```
/* initializes libvlc */
void vlc_init(void);

/* enables libvlc's interrupts */
void vlc_start(void);

/* processes pending async tasks */
void vlc_process(void);

/* registers callback for incoming messages */
void vlc_register_rx_cb(void (*cb) (uint8_t *data, uint8_t size);

/* registers callback for sent messages */
void vlc_register_tx_cb(void (*cb)());

/* registers callback for completely handled messages */
void vlc_register_tx_done_cb(void (*cb)());

/* adds a communication channel */
void *vlc_add_channel(uint8_t id, const pin_t *cat, const pin_t *ano,
     Channel *ch);

/* sends a message */
int vlc_send_message(uint8_t id, uint8_t* msg, uint8_t size, uint8_t
     dest);

/* configures protocol parameters */
void vlc_set_parameter(int group, int param, int32_t value);
```

PHY GROUP [0]		MAC GROUP [1]		LOG GROUP [2]	
0	Preamble length	0	# RTNS	0	Verbosity level
1	FEC threshold	1	DIFS	1	Prefix
2	Busy threshold	2	CWmin		
3	Light emission	3	CWmax		
		4	RTS threshold		
		5	MAC address		

Table 3.3: Configurable parameters through *libvlc's* API. The parameters are divided into configuration groups. The PHY configuration group includes parameters to configure FEC, and light emission and sensitivity. The MAC group consists of the parameters discussed in Section 3.3.2 and additional parameters to set the RTS threshold and device MAC address. The logging group parameters can be used to influence logging behavior.

layer which operates the LED through GPIO pins. The method also provides the infrastructure to add multiple communication channels. As the name suggests, `vlc_send_message` provides the functionality to dispatch messages to the MAC layer. The message, represented as a byte array, can be sent using a specific channel (id) to any device MAC address. The method returns after the handover to the MAC layer is completed with a non-negative value if successful and with a negative value if the message was dropped due to a full queue. All parameters for the PHY and MAC layer and other infrastructure can be adjusted with the `vlc_set_parameter` method. Parameters are divided into groups and within a group identified by a numeric value as listed in Table 3.3.

The PHY group includes parameters to change the PHY layer behavior. The preamble length is used for multi-channel devices and is further explained later in this thesis. The FEC threshold defines the PHY layer payload length starting from which FEC is applied. The busy threshold is used to define the detection sensitivity. If light-level differences are above this threshold, the channel is busy and the data intervals introduced in the PHY layer section are considered as data symbols. Light emission is con-

trolled through the last parameter of the PHY group. It defines whether light is enabled during an ILLU slot or not. Most parameters in the MAC group were already discussed. In addition, the RTS threshold defines the MAC payload size starting from which the RTS/CTS protocol extension is enabled. The MAC address parameter is used to overwrite or set the device's address. The logging group parameters influence logging behavior and are useful for debugging and measurements. The verbosity level can be set from 0 (none) to 7 (silly). The prefix parameter can be used to prepend a specific character to each logged line.

3.4 SYSTEM IMPLEMENTATION CONCEPTS

The implementation is mostly written in C, which is the commonly used programming language for microcontrollers due to its closeness to the hardware. The C code is complemented with C++ classes where encapsulating functionality into objects makes sense and to create individual namespaces for method names. There are no additional libraries needed for the implementation of the discussed communication layers. The basic functionality, provided by *avr-libc*[9], which is the standard C library shipped with the *avr-gcc*, the compiler suite for the AVR architecture, is sufficient enough to implement the envisioned system.

The PHY and MAC layer are implemented as strictly separated and independent parts with a small and clearly defined interface to each other for data and status exchange. Therefore, individual layers can be adapted separately or can be completely exchanged without interfering with their counterparts, and additional layers can be inserted between existing layers without effort. The following sections describe additional implementation concepts used to keep the software system flexible and to increase maintainability.

3.4.1 *Pin Abstraction*

The ATmega328P microcontroller supports multiple digital and analog GPIO pins. The pins are spread over different pin banks,

9. http://nongnu.org/avr-libc/

each holding, since it is a 8-bit architecture, eight pins. One bank is responsible for the analog pins connected to the ADC peripheral, whereas the digital pins are held by two other banks. The pins are controlled via two 8-bit registers per bank. One register controls the pin direction, whether it is used as input or output pin, and the other register can be used to set the pin to a low or high logic level (if used as output pin) or to enable or disable a pull-up resistor (if used as input pin). The individual pins are controlled by setting the respective bit within those registers.

To increase the usability of *libvlc* and to support the communication channel infrastructure described in the next section, the pin software mapping is introduced. During the initialization phase of an application based on *libvlc*, the software representation of hardware pins can be loaded. A method provides the functionality to map a specific pin specified by using an identifier to the software structure later representing the pin within *libvlc*. The mapping function assigns the respective registers to their software representation, which will be used for all pin operations in the PHY layer.

3.4.2 *Communication Channels*

A communication channel is the abstraction of an LED (as sender and receiver) within *libvlc*. It bundles an anode and cathode pin together, using the pin abstraction described in the last section. Since the microcontroller provides several GPIO pins, it is possible to have multiple communication channels per device making it reasonable to model a channel as a class to enable the creation of multiple instances. How multiple channels are handled within *libvlc* is discussed later in this thesis.

The channel class provides the basic functionality to operate the LED as a communication front end. Light emission can be enabled or disabled invoking channel methods to create ILLU slots and to trigger light output during data intervals. The light sensing process is controlled with the aid of three class methods. The `charge` method is reverse biasing the LED marking the starting point of a sense cycle. At the end of a sensing interval, an ADC measurement can be triggered with the `sense` method. An ADC

conversion is not done within an instant. Depending on the requested granularity, it can take from microseconds to milliseconds. The `measure` method, responsible for returning the measurement result, takes this fact into account. The `measure` method immediately returns the result if it is available or blocks until the value becomes available. In addition, the channel stores all measurements conducted during a COM slot for later synchronization and bit detection processing.

3.4.3 *Timer and Interrupt Handling*

The PHY layer's slot and interval scheme is implemented with a timer. All intervals are fed into the timer abstraction. The timer fires after the desired interval length and calls the appropriate method (implemented as sequential state machine explained in the next section) to handle the interval specific tasks, like enabling light output or triggering an ADC measurement, executed in an interrupt context. The interrupt handling methods are kept as short as possible to not interfere with following intervals. Since the PHY layer actions and processing are time critical (e.g., synchronization or bit detection has to be processed and executed before the next COM slot), it is directly handled within the interrupt routines at places where the needed computing time fits between consecutive intervals.

Since the FEC and the MAC layer operate on the completely received data buffer, further processing is not time critical and can be handled outside the timer interrupts during the normal program execution. Therefore, the required tasks are bundled in process methods, each executed within `vlc_process` which is invoked in the application's main loop. The processing tasks are controlled by flags set during the interrupt routines.

3.4.4 *Physical Layer State Machines*

The PHY layer is based on a hierarchical sequential state machine as shown in Figure 3.18. Three super states are defined: Transmitting (TX), Receiving (RX) and idle. The starting point is

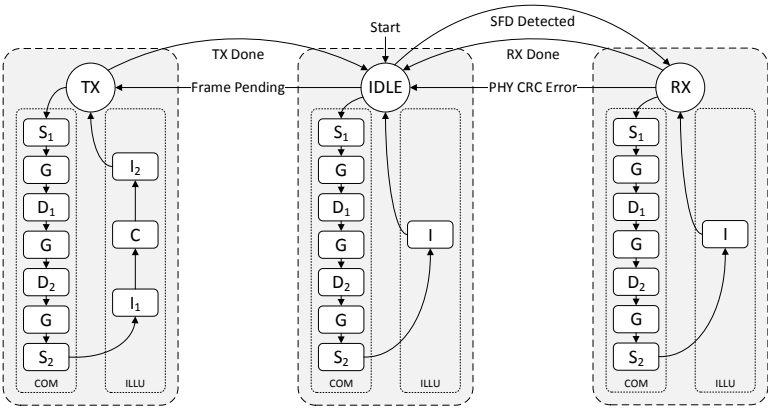


Figure 3.18: PHY layer hierarchical state machine. The three super states Transmitting (TX), Receiving (RX) and idle describe the current activity of the PHY layer. The PHY layer starts in the idle state. Whenever a frame from the MAC layer is pending, it transitions to the TX state, transmits the frame an switches back to idle. While in idle, the channel is scanned for an SFD and when detected, the state is switched to RX where the frame is received. The PHY layer falls back to idle on a CRC error in the PHY header or when the frame is completely received. While in the super state, the PHY layer cycles through sequential finite state machine handling the slots and interval actions.

the idle state. Whenever the MAC layer finishes a backoff process, a frame becomes pending in the PHY layer and the state is switched to TX. As soon as the last bit is sent, the PHY layer transitions back into the idle state. While in the idle state, every COM slot is decoded as if receiving, generating a continuous bitstream. When the last received eight bits match the SFD, the PHY layer switches to the RX state. While in the RX state, the following three bytes (PHY header) are awaited, where the last one contains the CRC protecting the PHY header. If the CRC fails, the frame reception cannot be continued and the PHY switches back to the idle state. If the PHY header is correct and therefore the correct payload size is known, the PHY state machine stays in the RX state

until the complete payload is received and subsequently transitions back to the idle state.

When changing the super state (idle, TX, RX), the underlaying state machine handling the intervals is also exchanged. E.g., the actions taken at the start and end of the data intervals are different for the TX and RX or idle super state: while in the RX super state, the channel is sensed during the data intervals and while in the TX super state, the LED is switched on and off according to the bit that is currently transmitted.

For efficiency reasons, the intervals used for the underlaying state machines do not match the schematic figures introduced earlier (for an ILLU slot). During the TX super state, the duration of I_1 and I_2 is chosen so that the compensation interval C falls at the same position as the first data interval. This generates a long (continuous) interval I_2, which can be used for PHY layer processing. For the idle and RX super state, the ILLU slot consists only of one single state.

Each super state is implemented as a separate sequential state machine. The states (representing intervals) are described as function pointers, stored in an array. Running the state machine is straight forward since all states have to be traversed in sequence. Hence, the array of function pointers can be iterated in order (using timer interrupts), invoking each function to process the corresponding interval. After handling the last state, the sequence starts again from the beginning at the first array element. Every super state owns its own sequential state machine wrapped in an array of function pointers, which is exchanged when super states are switched. This implementation has the advantage that new functionality (super states, interval states) can be added, and existing implementations replaced, without effort. Additionally, common functionality, e.g., the processing function for the synchronization intervals, can be used throughout all state machines by using a common function pointer, referencing the same implementation.

3.5 SYSTEM EVALUATION

In this section, the presented VLC system is evaluated to demonstrate the feasibility of a software-based LED-only communication system. A single link scenario is measured to assess throughput performance for variable communication distances. This experiment primarily demonstrates the PHY layer capabilities. To test the MAC layer protocols, a full network setup is evaluated. Throughput and delay measurements give insights about the network behavior and protocol characteristics. Additional measurements demonstrate the implemented RTS/CTS protocol extension in a hidden station scenario. The following Sections 3.5.1 to 3.5.3 describe the testbed configurations and the used hardware components and summarize the measurements and findings for the individual scenarios.

3.5.1 *Single Link*

The devices used in the testbed consist of an Arduino Uno board equipped with an ATmega328P microcontroller as described earlier. A red (640 nm wavelength) LED from Kingbright (see Section 3.2.2 for more details) is directly connected to the microcontroller's pin and operated by *libvlc*. The devices are connected via USB to a control computer. The USB connection is used to emulate a serial connection. The microcontroller runs a measurement application on top of *libvlc* using the specified API. The control computer communicates with the measurement application through the available serial communication link and can start, stop, configure PHY and MAC parameters, and change payload sizes. The measurement application generates saturation traffic (next packet is sent as soon as the previous packet is completely processed) with random payload content according to the specified size received from the control computer. All packet transmissions and receptions are logged on the control computer together with statistical and time information. These logs contain all the information to produce a detailed analysis, e.g., for a data throughput and delay evaluation. Listing 3.3 shows typical log lines received from the communicating devices and printed to a console on the

Listing 3.3: Typical logs collected during an experiment. The information contained is (from left to right): mode (reception or transmission), frame type, source, destination, payload (complete size on PHY layer), sequence number, CW, maximum CW, RSSI, corrected errors (if FEC is enabled), channel number (if multiple channels are present), reserved, reserved, dispatch time, RX/TX time.

```
[T,A,0A->0B,0(9),2,0,32,0,0,0,0,0,856430,856511]
[R,A,0A->0B,0(9),2,0,0,24,0,0,0,0,0,854010]
[R,D,0B->0A,200(209),3,0,0,32,0,0,0,0,0,858219]
[T,D,0B->0A,200(209),3,0,32,0,0,0,0,0,854011,855718]
```

control computer while running an experiment. The statistical information is generated through instrumentation within *libvlc*. Each device keeps its own real-time clock, which is used to log frame dispatching times on different layers as well as timing information for TX and RX complete events. Additional statistics such as RSSI values or number of errors corrected by FEC provide information about the quality of the current VLC link.

Testbed Setup

The testbed setup used for the single link experiments is shown in Figure 3.19. Two devices equipped with a measurement application running on top of *libvlc* and a single red LED are placed opposite each other on a rail system mounted on a wooden board. The devices are screwed to a block of wood that can move back and forth on the rail. One block is fixed at one end of the rail, whereas the other can be moved back and forth to assume various communication distances to determine the maximum communication range and run experiments for different receiving signal strengths. As the light intensity is inversely proportional to the square of the distance from the source, the number of photons per area also decreases when increasing the distance, leading to a smaller photocurrent induced in the receiving LED. Hence, the difference between on and off signals is less prominent, so that it is more difficult to distinguish received bits, which results

in a higher error probability. The height of the wooden blocks (around 5 cm) is chosen so, that reflections on the ground do not contribute to the overall signal strength and quality, and therefore provides an accurate free-space optical communication scenario.

The measurement application running on the testbed devices is remote-controlled from the control computer with the help of Python scripts. This tool suite allows to start and stop the testbed devices, enable and disable traffic generators and apply different MAC and PHY layer settings, providing a convenient tool to script and run experiment collections on the control computer. Measurement logs are printed to `stdout` from where the data can be piped into additional processing tools or saved to files.

The single link experiments include the following measurements: One device is fixed to the rails and the other device is moved to different communication ranges, starting from 10 cm with a step width of 10 cm. The fixed device generates saturation traffic, whereas the other device receives and acknowledges incoming data frames. A data frame counts as successfully transmitted if the corresponding ACK is also received. Since there is only one transmitter and therefore only one device competing for medium access, the MAC protocol could be disabled. How-

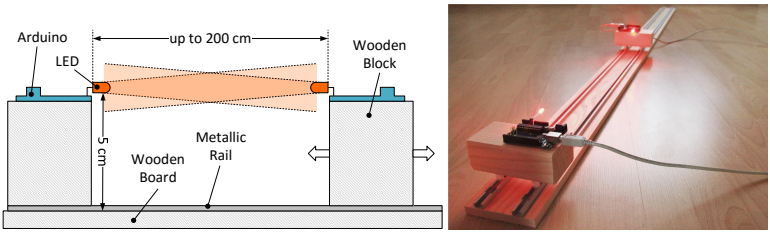


Figure 3.19: Single link distance measurement testbed setup. The left part depicts a simplified sketch, whereas the right part of the figure shows a photograph taken from the actual testbed. Two metallic rails are fixed on a wooden board. Two wooden blocks, each equipped with an Arduino, are mounted on the rails and can be moved back and forth to realize various communication distances. The devices are both connected to a control computer to collect the measurement data.

ever, the configuration is kept to be able to compare results to the following hidden station and network scenarios. At each distance, experiments for different MAC layer payload sizes (1, 10, 20, 50, 100, 150, and 200 B) are conducted. FEC is disabled and the default MAC layer parameters (see Table 3.2) are used. The experiments at all distances are repeated for an FEC threshold set to 20, meaning that for a PHY layer payload of 20 B or more, FEC will be applied. The experiments at each distance and for each payload size are run for 120 s, providing enough data to assess the PHY layer performance for the various communication ranges. The measurements were conducted during normal day lighting conditions. No other VLC system or light source using duty cycling, and possibly interfering with the testbed devices, were present.

Results

The results for the single link distance measurements are shown in Figure 3.20. The y-axis denotes the measured average throughput in b/s (bit per second). Only real payload (without layer headers) is accounted for in the average throughput. The x-axis denotes the communication distance (between the tips of the LEDs) in cm. The lines of different colors (and markers) stand for the different payload sizes used in the experiments. The error bars show the standard deviation from the average throughput.

The theoretical PHY layer bit rate can be computed from the duration of a COM and ILLU slot. They together sum up to a duration of 1 ms, meaning that every millisecond one bit of data can be transfered, leading to bit rate of 1 kb/s. As expected, higher throughput can be reached with larger payload sizes. The additional PHY and MAC header and the executed CSMA/CA protocol (backoff process) makes smaller payload sizes less efficient due to the large relative overhead per data frame. For a payload size of 1 B, an average data throughput of 40 b/s is reached. When increasing the payload size, the throughput also increases, reaching 900 b/s for a maximum payload size of 200 B, which is close to the theoretical maximum taking the PHY and MAC layer overhead into account. The throughput stays stable up to a distance

of 130 cm. Communication is still possible at a distance of 140 cm for all packet sizes, but with reduced throughput due to lost data frames. At 150 cm, the error possibility for larger payloads than 50 B is too high so that data exchange for these payload sizes is almost impossible. For small payloads (50 B and smaller), some data frames still reach their destination and are acknowledged successfully, reaching a maximum throughput up to 150 b/s. At a distance of 160 cm and onwards, no communication is possible anymore for this measurement setup.

Earlier work [81] based on a previous protocol version reports communication distances larger than 200 cm. In this protocol version, data and synchronization intervals filled a complete 500 µs slot. Longer intervals mean more time to collect incident photons, hence discharging the sensing LED further and therefore increasing the sensitivity. The current protocol is optimized for networking, and thus data and synchronization intervals are significantly

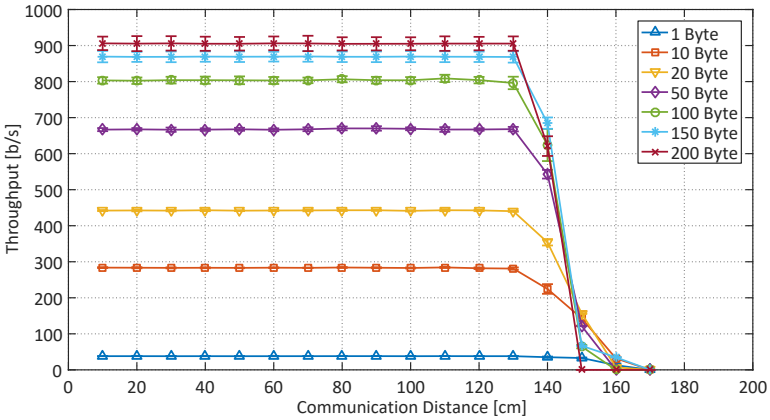


Figure 3.20: Single link measurement results for different communication distances with FEC disabled. Average data throughput is shown over distance for various payload sizes. The error bars show the standard deviation. Higher throughput can be reached with larger payload sizes due to the overhead introduced by PHY and MAC layer frame headers. Communication is stable up to 130 cm.

shorter and all contained within a single 500 µs COM slot. Based on the earlier results, increasing the duration of COM and ILLU slots (and therefore also scaling the data and synchronization intervals within a COM slot) increases sensitivity and therefore also communication distance. Of course this improvement has also a price: Fewer data intervals will be available within a certain amount of time, decreasing the PHY layer bit rate.

The statistics logged through *libvlc* for each received frame also contain information about the received signal strength. For each received bit, an RSSI value can be computed. When the two data intervals within a COM slot are compared, the difference of incident light during those two slot is calculated and used as RSSI value. These bitwise RSSI values are average over all received bits resulting in a single value for each received frame.

Figure 3.21 and Table 3.4 show the average RSSI value for all frames received at the same distance. The y-axis denotes the RSSI in ADC units. The ATmega328P's integrated ADC has a 10-bit resolution. For a reference voltage of 5 V, an ADC unit amounts to 4.9 mV. As expected, the RSSI values follow the inverse square law for increasing distances. From this follows that the chosen RSSI metric is directly correlated to the received light intensity. The x-axis denotes the distances between sending and receiving LED. As indicated by Figure 3.21, the RSSI drops quickly when the

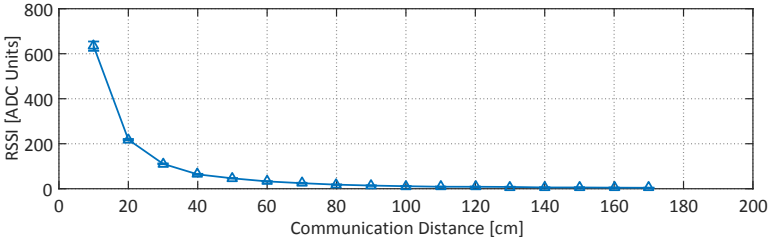


Figure 3.21: RSSI values for data frames received at various distances. For exact values for all distances see Table 3.4. RSSI is measured in ADC units (roughly 5 mV per unit). The curve follows the inverse square law when increasing the distance. From approximately 100 cm onwards, the RSSI values already fall below 10. The error bars denote the standard deviation.

DISTANCE	RSSI	STDEV	DISTANCE	RSSI	STDEV
10 cm	633.6	20.8	90 cm	14.1	1.3
20 cm	218.2	2.9	100 cm	11.2	1.1
30 cm	110.7	2.0	110 cm	8.9	1.0
40 cm	63.8	1.0	120 cm	8.7	1.3
50 cm	46.2	2.4	130 cm	8.0	1.0
60 cm	32.6	1.1	140 cm	5.6	0.9
70 cm	24.7	1.0	150 cm	5.6	0.9
80 cm	18.1	1.8	160 cm	4.8	0.7

Table 3.4: RSSI values with standard deviation for various communication distances. Stable communication is possible up to 130 cm with a RSSI value of only 8 ADC units which corresponds to approximately 40 mV.

distance is increased but communication stays stable (as seen in Figure 3.20) up to a distance of 130 cm. This is remarkable since the RSSI at 130 cm only amounts to 8 ADC units or approximately 40 mV. For small payload sizes, communication still works up to 150 cm or an RSSI value of 5.6. In the end, the communication fails because of flipped bits caused by the low signal strength. Synchronization still works up to 200 cm, which is confirmed by oscilloscope readings. These results show that although LEDs are used as receivers and only a simple and low-cost microcontroller drives the PHY layer, the system still works at low light conditions and is accurate enough to enable stable communications.

To demonstrate the implemented FEC based on Reed-Solomon codes, the single link distance measurements are repeated with an FEC threshold of 20. Hence, for every packet with at least 20 B PHY layer payload the additional 16 B of redundancy is computed and appended at the end of the PHY payload. Additionally, the FEC flag in the PHY header control field is set to true to inform the receiving device about the enabled FEC.

The measurement results are shown in Figure 3.22. The y-axis denotes data throughput in b/s, whereas the x-axis denotes the

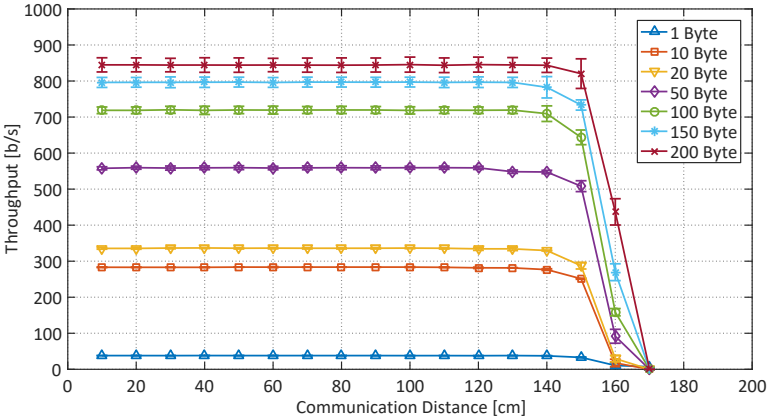


Figure 3.22: Single link measurement results for different communication distances with FEC enabled. Average data throughput is shown over distance for various payload sizes. The error bars show the standard deviation. In comparison to the results shown in Figure 3.20, less throughput is achieved for all packet sizes due to the additional FEC overhead. Thanks to the error correction capability, communication is now stable up to 150 cm.

various distances between sender and receiver. The line colors and markers stand for the different payload sizes. The error bars show the standard deviation. A comparison of the throughput to the results depicted in Figure 3.20 clearly shows the effect of the additional overhead caused by the added FEC redundancy. The overall throughput is slightly reduced by approximately 50 to 100 b/s, whereas the decrease is more prominent for smaller payload sizes. The throughput for payload sizes 10 B and 1 B is not affected since their PHY layer payload size is below the FEC threshold.

The positive effect caused by FEC starts to arise from a distance of 140 cm. The throughput stays stable at this distance, whereas at the same distance and without FEC, the average throughput already significantly dropped. Figure 3.23 depicts the number of corrected errors for all the different payload sizes and for all distances. Only successfully received data fames are accounted

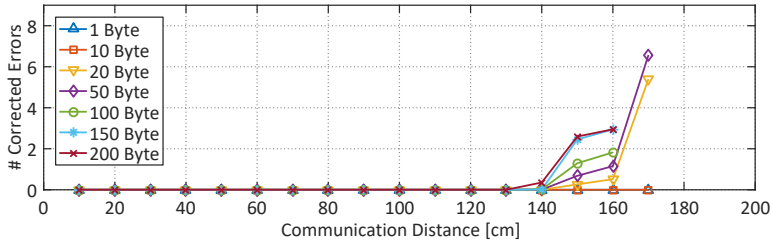


Figure 3.23: Number of corrected errors in a single link scenario for different distances and various payload sizes. No errors are detected and corrected up to a distance of 130 cm. From 140 cm, the number of corrected errors starts to rise. For 160 cm and larger distances, data frames only get through occasionally (comparing with 3.22), if the detected number of errors lays within the correction capabilities. This plot only captures successfully received (and corrected) data frames. Uncorrectable data frames are not logged by *libvlc*.

for. The corrected error number for 140 cm is small but existent, reflecting what is shown in the throughput plots. Without FEC, already one flipped bit can invalidate a complete data frame, resulting in a throughput drop in case many frames are affected. With active FEC these few errors can be corrected which stabilizes the average throughput at this distance. Also at a distance of 150 cm, throughput is only slightly reduced and stable communication is still possible. The error plot shows that the corresponding values are between 1 and 3 detected and corrected errors. For 160 cm, reasonable data throughput is still possible, e.g., for a 200 B payload, an average of 450 b/s is achievable. At 170 cm, only a few packets reach their destination with error numbers close to 8, which is the maximum number of correctable errors for the used FEC scheme.

Using software-based FEC increases communication distances for a single link LED-only scenario by 15 to 20 cm. As stated in Table 3.4, RSSI values at theses critical distances are already low (between 4 and 5 ADC units), so that distinguishing between different bit values is challenging. For larger distances, the decoded bit stream will contain more random bits than actually received

bits and therefore makes it impossible for the FEC to recover correct data frames. Although FEC could improve communication distance, it will be more helpful in a noisy environment where interference is present.

3.5.2 *Hidden Station*

With a second experiment series, *libvlc's* capabilities in a hidden station scenario is evaluated. As discussed in the MAC layer section, *libvlc* implements an RTS/CTS protocol to decrease the packet loss rate when hidden stations are present. The same hardware and software configurations as for the single link scenario are used. To demonstrate the hidden station effect as explained with the help of Figure 3.16, at least three devices are needed. The following two sections describe the testbed used to evaluate this scenario and the measured results.

Testbed Setup

The testbed setup is shown in Figure 3.24 and analog to the explanatory Figure 3.16. Three devices are fixed to a wooden board. Device B is positioned between device A and C. Its transceiver is made of two LEDs soldered together in parallel in a 90° angle. The two LEDs in parallel roughly behave as a single bigger LED, but with the capability to transmit to and receive from different directions. The devices A and C are placed at different corners of the wooden board so that their LEDs point towards B's LEDs. Device B is now able to transmit to device A and C and can also receive messages from both, whereas A and C are not in each others field of view. In other words, devices A and C are hidden from each other. To completely block every possible reflection, an additional view blocker is installed between A and C. Since the outcome of this scenario depends on packet collisions, the capture effect [49] needs to be suppressed. Due to the capture effect, one of two colliding data frames could still be received if the signal strength for one of the transmitted signals is significantly stronger and overlays the weaker signal. Therefore, the distance from device B to A and C is kept the same, namely 10 cm each, to

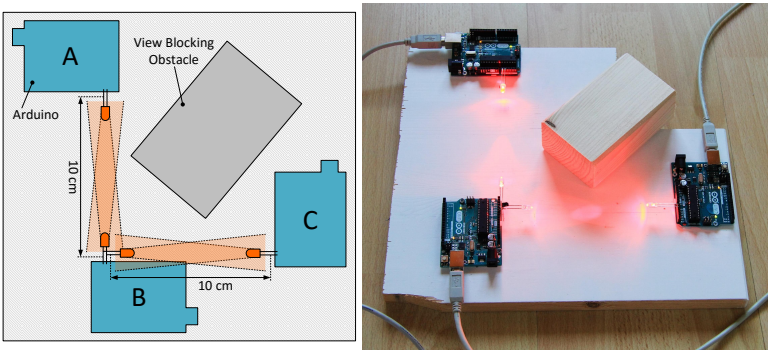


Figure 3.24: Hidden station scenario testbed. Device B is equipped with two LEDs soldered together in parallel. These two LEDs point in different directions with an angle of 90° between them. The LEDs of devices A and C point towards B's LEDs and are positioned so that they are not in the field of view of each other but at the same time are able to communicate with device B. An additional obstacle blocks the view between device A and C.

achieve similar signal strength at device B when receiving from both other devices. Enforcing equal signal strength prevents the capture effect and colliding packets will most likely not reach their destination.

The measurements for the hidden station scenario are conducted with the same software and devices used for evaluating the single link scenario. Experiments are executed for the already introduced payload sizes and the default MAC parameters are applied. Devices A and C generate saturation traffic and device B receives and acknowledges incoming data frames. Due to the fact that A and C are hidden from each other, they will not sense a busy channel when the other device is transmitting, which will lead to collisions. Since the outcome of this scenario is partly based on chance (collision probability depends on random contention window), the duration for each experiment (payload size) is set to 10 min. The experiments for all payload sizes are once conducted with RTS/CTS disabled to demonstrate the hidden station effect. In a second run, all experiments are repeated with

the RTS threshold set to 0, enabling the protocol extension for all payload sizes.

Results

The results for the hidden station scenario are shown in Figure 3.25. There are two y-axes, one on the left and one on the right. The left one, associated with the blue color and blue lines denotes data throughput in b/s. The right y-axis, associated with the red color and lines plotted in red, denotes the packet loss rate. For both axis, the error bars show the standard deviation. The x-axis stands for the different packet sizes used in this scenario. Results for both experiments, with and without RTS/CTS protocol extension, are displayed in this plot. Throughput and packet loss rates are summed up for both devices, A and C.

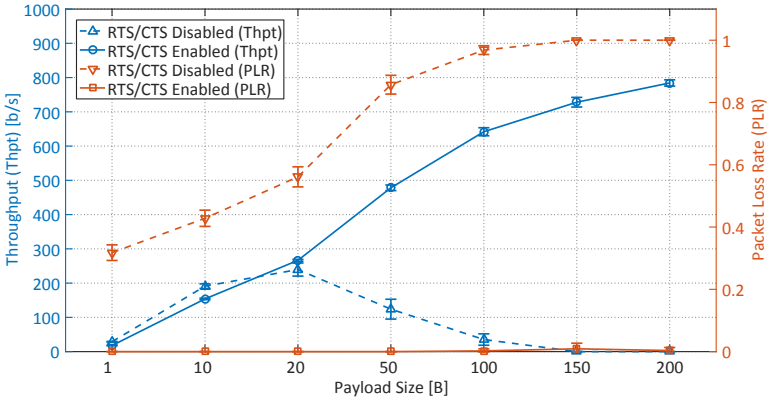


Figure 3.25: Measurement results for the hidden station scenario showing data throughput and packet loss rate. The left y-axis denotes data throughput and shows the scale for the blue lines and the right y-axis denotes the packet loss rate and is valid for the red lines. The dashed lines show the results for disabled RTS/CTS. Throughput drops down towards zero from 20 B payload size due to packet collisions. The solid lines show results for RTS/CTS enabled. Thanks to the protocol extension, collisions are prevented, resulting in an almost nonexistent packet loss rate.

The dashed lines show the results for disabled RTS/CTS. For small payload sizes up to 20 B, approximately half of the packets are lost, resulting in a data throughput of 200 b/s. Although both devices A and C are generating saturation traffic and are not aware of each other (and the most likely occupied channel), data frames (and their retransmissions) reach their destination and are acknowledged. This can be explained with the short transmission time of small packets. The short airtime and the increased contention window for retransmitted packet lead to a lower collision probability. When increasing the payload size further, collision probability also increases and throughput drops down to zero, and the packet loss rate goes up to 100 %. The data frames collide due to uncontrolled medium access caused by the hidden station problem.

The solid lines show the results for the same scenario with enabled RTS/CTS. As for the single link measurements, the throughput increases when increasing payload size and therefore reducing the overhead effect. The packet loss rate is close to zero for all payload sizes. The maximum achievable data throughput of about 800 b/s is reached with a payload size of 200 B. When comparing with the results from the single link scenario, there is a throughput difference from 100 to 150 b/s, depending on the payload size caused by the protocol overhead for the RTS/CTS. The throughput results for both experiments show that it makes sense to enable RTS/CTS for all packet sizes if hidden stations are to be expected, also for small payload sizes, since the achieved throughput without RTS/CTS is not significantly higher.

These results show that also for a VLC network, hidden stations are a problem. Since communication is visible, it is less difficult to identify hidden stations than it is for radio communication. For scenarios where hidden stations are present, the implemented MAC protocol extension using RTS/CTS significantly reduces packet collision and increases data throughput. The results also demonstrate that *libvlc* operates as intended and protocols are implemented correctly.

3.5.3 *Network*

The third evaluated scenario explores *libvlc's* MAC layer capabilities. As described earlier, *libvlc* implements a CSMA/CA similar to the one described in the IEEE 802.11 standard. The MAC layer protocol uses a listen-before-talk approach together with random backoff. For this evaluation, a network with twelve VLC devices is built. All devices are connected and have access to the same channel (all devices are in the field of view of each other). Eleven devices are simultaneous transmitting data to the twelfth device and competing for the medium. The following two sections describe the custom built testbed and explain the measurement results.

Testbed Setup

The core part of the testbed is a wooden frame shown and described in Figure 3.26. It encloses a round USB hub with more than 20 USB sockets. The top of the frame is completed with a cir-

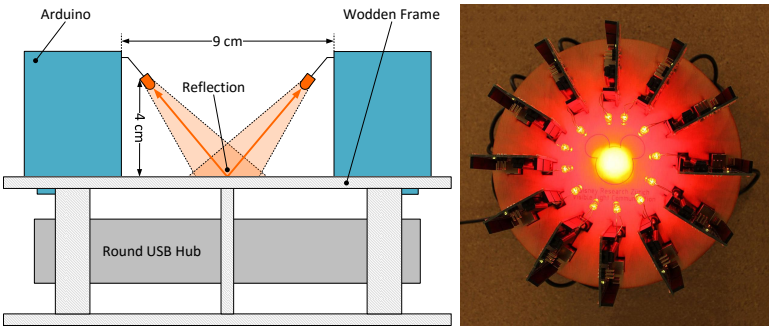


Figure 3.26: Network scenario testbed. A wooden frame is built around a round USB hub where all participating devices can be plugged in. On top of the wooden frame is a circular platform with holes fitting the profile (USB and power plug) of an Arduino. Twelve devices can be sticked into the top platform, forming a circle. The connected LEDs point towards the center of the platform where the light is reflected in all directions, reaching all other receivers.

cular platform providing enough space for twelve devices. Holes in the front panel with the shape of an Arduino connector side (USB port and power plug) makes it possible to directly stick the devices through the panel and connect them to the USB hub from the other side. The exactly fitting notches hold the Arduinos in place without any other fastening mechanism. The twelve devices are arranged in a circle with connected LEDs pointing towards the center of the top platform where the light is reflected in all direction. The reflection of one LED is strong enough to reach all other devices so that they all share the same channel.

To evaluate the MAC layer behavior for different numbers of competing devices, a series of eleven experiments is conducted. For every new experiment, the number of simultaneously transmitting devices is increased by one, from one active device to eleven. For every experiment, random traffic for the known payload sizes from 1 to 200 B is generated. Since successful medium access and collisions depend on the random contention window, the experiments need to run long enough to generate meaningful results. Hence, the experiment duration is 10 min for each number of transmitting devices and for each payload size. All devices transmit to the same device, which acknowledges incoming data frames. A data frame only counts for the average throughput in case the corresponding ACK is successfully received. For a MAC layer evaluation, it does not matter whether a device transmits to a random communication partner or to a predetermined fixed one, since the competition for the medium is not influenced by the destination of a data frame. To simplify the testbed setup and configuration and the result evaluation, a fixed data sink is used. The hardware and software is the same as used for the previous scenarios, and the default MAC layer parameters (Table 3.2) are applied.

Results

To verify the MAC protocol before running extensive measurements, a test setup with four transmitting devices is used (this time transmitting to a random other device). Additional instrumentation inside *libvlc* makes the MAC protocol visible on an

oscilloscope (in logic analyzer mode). A GPIO pin (debug pin) is configured to output a low logic level while the device is in idle mode, a high logic level while it is transmitting, and quickly switching between high and low while a backoff process is active. The debug pin for each device is hooked up to the oscilloscope's logic probes. While the experiment is running the actual protocol can be inspected on the oscilloscope when hitting the stop button to pause the acquisition of new signals. The result is shown as an augmented screenshot shown in Figure 3.27. The output signals for the individual devices are labeled on the left side. The dark gray ares stands for the backoff process during which the signal is switched on and off very quickly. Label 1 shows that device D0, D2, and D3 have a queued data frame and start the backoff process. Device D2 wins the competition for the medium (shorter contention window, label 2) and finishes the transmission at label 3. Device D1 and D2 immediately start with another backoff

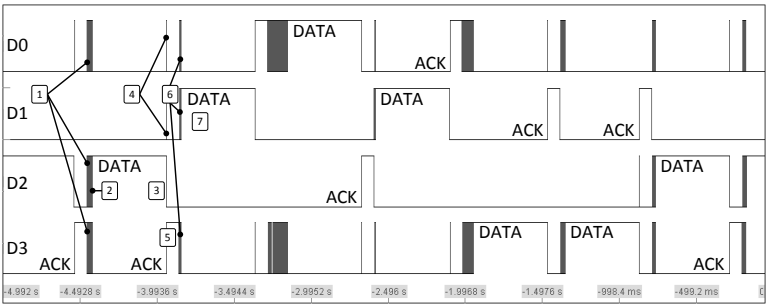


Figure 3.27: MAC protocol visualized on an oscilloscope. Data and ACK frames together with the backoff process are shown. 1) Ongoing backoff process. 2) Device D2 wins medium access race and starts transmitting. 3) End of a data frame sent by device D2. 4) As soon as the medium is clear again, device D0 and D1 start counting down their DIFS. 5) The ACK for the previously sent data frame has priority and is immediately sent. 6) After the ACK has been sent, the channel is free and D0, D1, D3 compete for the medium by starting another backoff process. 7) Device D1 wins the race for the medium and starts transmitting the next data frame.

process after the channel is released (label 4), but are interrupted by an ACK from D3 (at label 5), which is acknowledging the data frame sent by D2. This shows a correct prioritizing of the ACK in respect to a data frame. Label 6 shows that as soon as the ACK is transmitted and the channel is free again, the backoff processes from D0, D1 and D3 (from label 1) are resumed. This time, device D1 wins the race for the medium an transmits its data frame (label 7). More examples of a working listen-before-talk and backoff process can be identified in the screenshot, confirming a working MAC protocol.

The experiment series for the network scenario is evaluated for data throughput and packet delivery delay. Figure 3.28 illustrates the collected and processed measurements for one to eleven devices competing for medium access. The y-axis denotes

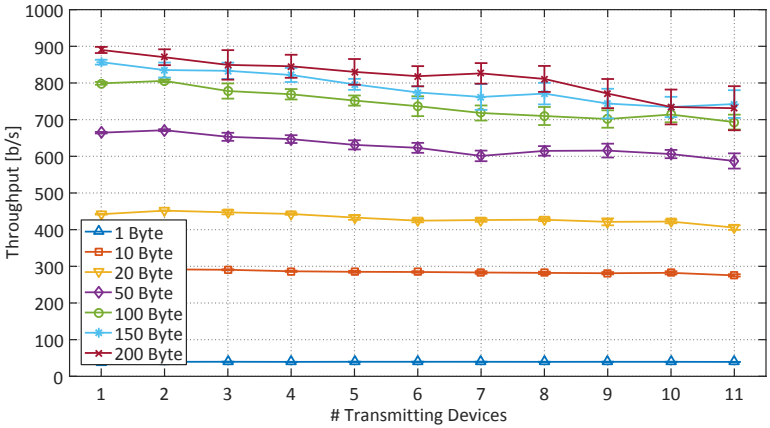


Figure 3.28: Full network scenario measurement results. Average data throughput is plotted versus number of transmitting stations. One to eleven devices are simultaneously transmitting and competing for channel access. The MAC protocol, which is part of *libvlc*, limits packet collisions to a minimum so that, although eleven devices try to access the medium, only few collisions happen and an average throughput of 750 b/s for the maximum payload size of 200 B is achievable. The error bars show the standard deviation.

data throughput in b/s and the x-axis stands for the one to eleven transmitting devices. The variously colored lines with different markers show the results for the different payload sizes. The error bars show the standard deviation. The results for one transmitting device confirms the results from the single link measurements. Increasing the number of stations for payload sizes of 20 B and shorter does not significantly influence the average throughput. Since the packets are short, the channel time loss caused by a collision only marginally influences the average throughput. Although all devices can sense the same channel, data frames can still collide due to contention windows with the same size. The more participating transmitting station, the higher is the collision probability. This probability can be influenced with the CWmin and CWmax parameters. They have to be chosen in a way, so that for a given network, the devices do not have to wait for too long before transmitting, and that the collision probability is low enough to guarantee stable data exchange. The default parameters are optimized for smaller networks of around a dozen devices, which is confirmed by the measurements. Although collision probability is higher for longer packets, the average throughput only drops approximately 100 to 150 b/s for larger payload sizes. For eleven transmitting devices competing for the medium, an average throughput of 750 b/s is achievable for the maximum payload size of 200 B.

Figure 3.29 renders the probabilities of packet delivery delays for three different packet sizes (1 B, 50 B, and 200 B) for one and eleven sending devices. The results are plotted as Cumulative Distribution Functions (CDFs) to better visualize the delays caused by contention. The delay is defined as time passed between the transmission of a packet and the reception of the corresponding ACK. The y-axis denotes the probability that the delay is longer than the corresponding x-axis value. The x-axis therefore denotes time in ms. The three solid lines show results for only one transmitting station (no competition for the medium). The time indicated by the top part of the lines show the minimal possible delay given by the length of the packet and the time it takes to transmit it. The lines go almost straight from top to bottom, because there is no contention and packets can only be more delayed through

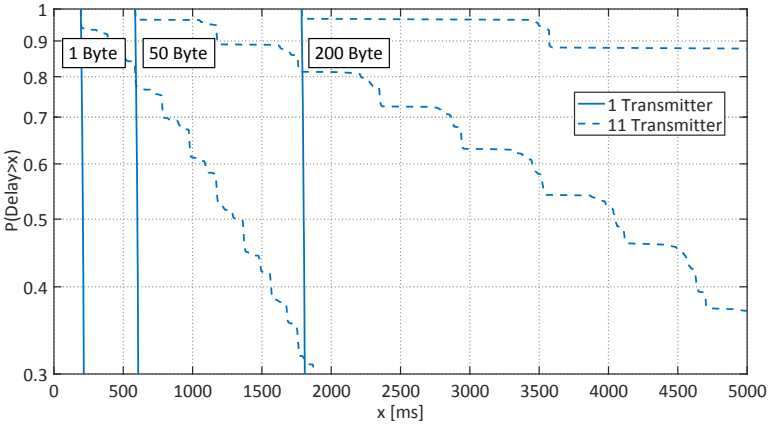


Figure 3.29: Data frame delivery delay probabilities for a full network scenario. The delay includes a full round trip including the reception of a corresponding ACK frame. The graph shows plotted CDFs of delays for payload sizes of 1 B, 50 B, and 200 B, and for one or eleven transmitting devices. The curves for eleven transmitters show the typical step-like shape of a delay CDF for a contention-based protocol.

retransmissions caused by bit errors, which did not occur during the experiments (close distance and therefore high RSSI value). The steps in the dashed lines (eleven transmitting devices) indicate a correctly working contention based protocol. The width of the steps is again the packet transmission duration. Every step means that the competition for the medium access was lost and the transmitter must wait for another device to finish its transmission before trying again to transmit its own data frame.

The two plots showing measurement results for data throughput and data frame delivery delay and the oscilloscope visualization illustrate, that the implemented MAC control protocol works as intended. Additionally, they also show that the same MAC concepts used in radio-based networks can also be applied in the domain of VLC.

3.6 DISCUSSION AND CONCLUSION

The system discussed in this chapter is software-based and builds upon few and basic hardware parts. An off-the-shelf 8-bit microcontroller is paired with an unmodified LED to create a communication system capable of connecting multiple devices to a network. All communication logic is encapsulated into a firmware software library, called *libvlc*, running on the microcontroller. It provides a flexible prototyping environment to build applications for VLC networks. An LED is employed as a communication front end, used to send and receive data modulated with light. The LED is directly connected to the microcontroller's GPIO pins without the need of any other hardware parts of electronic circuits. The software library *libvlc* implements the complete PHY layer responsible for data modulation, reception, and synchronization, designed under the premise that the LEDs are always perceived (by human observers) as switched on at constant brightness. Together with the MAC layer implemented on top of the PHY layer, full network scenarios are supported, while still using the lights for illumination purposes.

The measurements results presented in the evaluation section confirm a working and stable system for LED-to-LED VLC networking. The achievable throughput is mainly limited by the processing power of the 8-bit microcontroller used and is in the order of 900 b/s at a distance of approximately 1.5 m. The communication range can be improved by prolonging communication slots at the cost of data throughput. The proposed MAC layer successfully handles medium access for networks of a dozen devices and a protocol extension using RTS/CTS control frames significantly improves and stabilizes communication in case of the presence of hidden stations. VLC based on LEDs as transceivers is an approach that enables low bit rate wireless networking based on consumer electronic equipment that is often already present in many environments. The simplicity of this approach lies in the reuse of existing components. Obscuring the communication within the visible light and hiding the communication in the illumination makes it possible to also reuse existing lighting system and to augment them with networking capabilities.

The presented software system, *libvlc*, provides a solid baseline for software-defined, low-complex and flexible VLC communication and networking. It can be adapted to any kind of LED-based lighting system and its use cases, extending it with communication and networking capabilities. Given the fact that lighting is omnipresent within buildings, in transportation and on the streets, VLC could be a technology providing the necessary communication fabric for the IoT. This chapter describes the base functionality of *libvlc*. The following chapters explain and evaluate library extensions and adaptations.

4 LIGHT BULB NETWORKS

The IoT envisions that many devices, such as appliances, wearables, sensors, utilities, and toys, can connect to a network to communicate with data centers, controllers, and other devices. Many of these devices have modest data rate requirements, but all-direction, uplink, downlink, and mesh connections are often a central requirement to adapt and control the device behavior, and to provide network redundancy and communication reliability. However, a communication infrastructure that aims to connect many devices should be low-cost, non-intrusive, and ubiquitous.

VLC is an attractive choice and has many desirable properties. LEDs allow the construction of low-cost communication systems [81] as shown in Chapter 3. VLC does not interfere with the use of the scarce radio spectrum, and VLC cannot be easily overheard in another room – to observe a message exchange that is communicated via a light channel, the eavesdropping party needs (direct or indirect) line of sight access. This chapter introduces *EnLighting*, a system of distributed and fully connected LED light bulbs that communicate through free space optics to enable indoor communication and localization services.

LED light bulbs combine multiple LEDs and are significantly brighter than single LEDs. Lenses or diffuse bulbs mounted on top of the LEDs transform them into an omni-directional light source. Such light bulbs are low-cost [79, 84] and have been proposed for many novel indoor applications. But some of the desirable properties for illumination, such as dispersion of light in all directions and the usage of white LEDs, make them unsuitable to directly receive signals from other light sources as described in Chapter 3. To enable the sensing of incoming signals from other light-emitting devices – light bulbs, single LEDs, or smartphones [12], LED light bulbs can be enhanced with simple light receiving electronics based on photodiodes. Such LED light bulbs are powerful enough to establish a communication link over sev-

eral meters, but are still based on the same software-defined PHY and MAC layer [82] as introduced with *libvlc*. These light bulbs can communicate with other light bulbs as well as with low-cost LED-only systems that can be integrated in many devices.

LED light bulbs mounted on the ceiling (or in free-standing floor lamps) easily cover a room, still serving as illumination devices, and at the same time create a room area network that allows data exchange between light-emitting devices. Some light bulbs may even act as gateway to the Internet to provide connectivity beyond a single room. Each light bulb contains a simple SoC, running an embedded version of Linux. The light bulbs provide the physical communication channel between the devices in the room.

This chapter discusses the design, implementation and evaluation of a room area network based on Linux-enabled light bulbs and shows that simple devices distributed in a room can provide an attractive solution for today's communication challenges without relaying on the already crowded radio spectrum. The chapter is structured as follows. Section 4.1 discusses the design of a Linux- and VLC-enabled light bulb together with the necessary software changes to *libvlc* and Linux. Section 4.2 describes a testbed architecture with distributed and networked light bulbs and software tools to enable the operation of large-scale light bulb network deployments. Section 4.3.1 presents a communication link and protocol evaluation for different network topologies: Traffic for the Internet Control Message Protocol (ICMP) UDP and Transmission Control Protocol (TCP) are evaluated and analyzed. Section 4.3.2 discusses and evaluates an indoor localization service built on top of the proposed communication system. Section 4.4 summarizes and concludes the chapter.

4.1 LIGHT BULB DESIGN

This section summarizes the VLC light bulb hardware and software protocol design. Commercial off-the-shelf LED light bulbs are used as a starting point and are then modified to host a SoC running Linux, a microcontroller running *libvlc* with an application on top, and an additional power supply to support the

added electronics. Since the light bulbs are usually used for illumination, they use white LEDs with a yellow-orange filter to make the light look warmer. These white LEDs together with the diffuse bulb on top makes it impossible to directly use the LEDs as a transceiver. Therefore, photodiodes are added to the system. Each light bulb system is built from parts of the original light bulb and additional custom built parts, circuitry, and casing. The main motivation for using consumer light bulbs as starting point is that they are readily available at low cost and can be used in any lamp with standard sockets. This leads to an easy-to-setup and flexible testbed with quick replication.

4.1.1 *System Architecture*

The VLC firmware and protocols described in Chapter 3 implement the PHY and MAC layers and enable low-level networking between multiple devices. As described, *libvlc* already supports the use of LEDs as transceivers. The library is extended, as described later, with the capabilities to employ photodiodes together with the available light bulb LEDs as a communication channel. To make use of higher-level network protocols, the microcontroller running *libvlc* (VLC controller) is extended with a SoC running a Linux distribution for embedded wireless systems called OpenWrt[1].

Figure 4.1 shows the overall system architecture. A SoC (label 1) from Qualcomm Atheros with the identification AR9331[2] is used to handle upper layer protocols such as IP and TCP. It also integrates a Wi-Fi module on the same chip and runs a Linux distribution for embedded systems. The Wi-Fi interface provides a control channel for testbed operations and firmware updates but is not meant to interconnect the light bulbs via a radio channel. The SoC connects to the VLC controller via the USART interface (label 2). The VLC controller (label 3), a microcontroller (ATmega328P) running a serial communication application on top of *libvlc* to accept commands via the USART interface. It is abstracted (from the

1. https://openwrt.org/
2. https://wikidevi.com/wiki/Atheros_AR9331

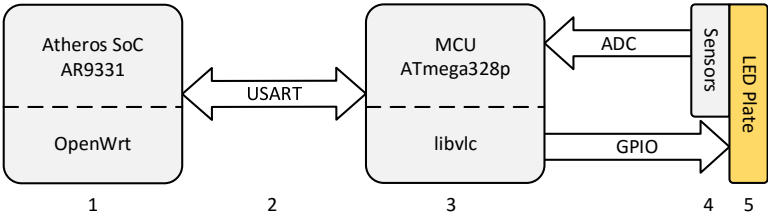


Figure 4.1: Light bulb system architecture: 1) Wi-Fi-enabled SoC running OpenWrt, a Linux distribution for embedded devices. 2) A USART interface connects the Linux system with the VLC controller. 3) VLC controller, consisting of a microcontroller (ATmega328P) running *libvlc*. 4) Photodiodes connected to a transimpedance amplifier provide light dependent voltage values, sampled by the VLC controller's ADC. 5) Board equipped with several LEDs used for illumination and communication, controlled by the VLC controller via a GPIO pin.

Linux side) as a regular Ethernet interface, implemented as a kernel driver module. Therefore most applications using TCP or UDP sockets will work out of the box and can make use of the VLC link. Photodiodes together with a transimpedance amplifier [25] build the sensing unit (label 4), converting the generated photocurrent into a light intensity dependent voltage value. The sensors are sampled by the VLC controller using the ADC. The LED plate (label 5) contains several LEDs connected in series, acting as light source for illumination and communication. The light is controlled via a GPIO pin from the VLC controller.

Since the VLC firmware is real-time critical, it is kept on a dedicated device, in the same way as Wi-Fi modules or any other networking device. In a different approach, the protocols are directly implemented on top of Linux using the available GPIOs as already demonstrated [99]. However, such a system works only as long as the operating system workload is kept low; as soon as the Linux host is used for other tasks (which motivated the addition of an operating system to VLC), timings might deviate and consequently influence the behavior of the VLC layers.

4.1.2 *System Components*

The system is built from parts of a light bulb[3] manufactured and sold by IKEA and additional custom-built parts, circuitry, and casing. This particular light bulb has been chosen because it has been widely available and the electronics are easily accessible and extendable. In the following, the most important parts are discussed: LED plate, power supply, sensors and amplification circuitry, VLC controller, SoC board, and the additional casing.

LED Plate

The LED plate included in the original light bulb is reused together with the diffuse bulb that is covering the LEDs. The plate consists of a single layer PCB equipped with 14 LEDs connected in series. It has a round shape with screw holes on the side and power connectors in the center. The LEDs, based on Surface-Mount Technology (SMT), are additionally covered with a yellow-orange filter to produce warm white light.

The diffuse bulb distributes the light in all directions and therefore makes it possible to send data to devices all around and not only in a single direction. Since the light intensity is also distributed, receiving a clean signal at a distance of several meters is challenging (see light sensor description). Furthermore, a heat sink is attached to the LED plate, since the LEDs produce not negligible thermal discharge when powered on. The original light bulb uses the socket as a heat sink, but since the LED plate is not connected to the socket in the new light bulb design, it needs to be cooled differently.

Power Supplies

The switching power supply built into the light bulb socket transforms the high Alternating Current (AC) voltage from the power grid to DC voltage of around 40 V. It is dimensioned so that it provides just enough current for the 14 LEDs in series. To provide

3. LED1221G7, E27, 6.3 W, 400 lm

enough power for the additional electronics, VLC controller, sensors, and Linux Board, a second power supply is added to the light bulb. The custom-built device is shown in Figure 4.2, part 1. It is built from an AC/DC converter which is transforming the grid voltage to 9 V DC. A PCB is directly attached to it, hosting a voltage regulator (1c) which is providing a low-ripple 3.3 V for the additional electronics. The low-ripple source is specifically important for the analog amplification circuitry (for further explanations see sensor description). There are connectors to attach power from the grid (1a) and for regulated 3.3 V power output (1b).

When testing the complete light bulb system, problems with the included light bulb power supply emerged. Due to its cost-

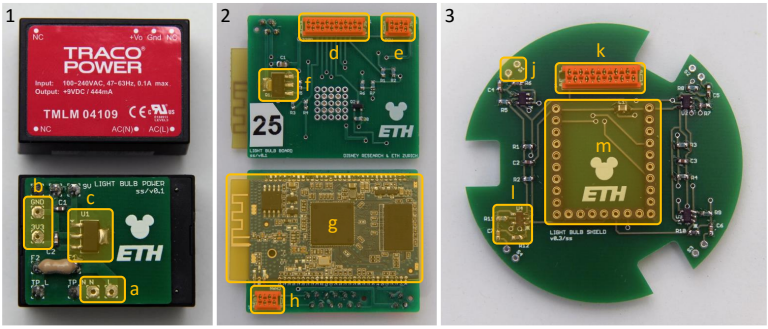


Figure 4.2: Light bulb electronic components. Part 1 shows the power supply for the additional electronics: 1a) Power grid connectors. 1b) Regulated 3.3 V output. 1c) Regulator for low-ripple voltage. Part 2 displays the SoC breakout board: 2d) Connectors interfacing with the VLC controller. 2e) Power connector for the electronics and light bulb power supply. 2f) Metal–Oxide–Semiconductor Field-Effect Transistor (MOSFET) used to modulated the LED plate from the VLC controller. 2g) SoC board with PCB antenna. 2h) LED plate connector. Part 3 shows the VLC controller and sensor board: 3j) Phototdiode connectors (four in total). 3k) Connector to interface with the SoC host board. 3l) Transimpedance amplifier (four in total). 3m) Pin headers to connect to the microcontroller running *libvlc*.

effective design and open built (no casing and shielding) radiation from the switching circuitry affected the photodiode amplifiers. As additional shielding did not serve the purpose, the power supply was exchanged with a shielded version with similar form factor (still fits into the light bulb socket) and properties. This finally solved the radiation problems and lead to clean sensor reading. When designing such a light bulb from scratch, the power supply could be designed so that all necessary voltages and current constraints could be fulfilled by a single power supply.

Light Sensors and Amplification Circuit

Every light bulb is equipped with four photodiodes from Osram[4], pointing in four orthogonal directions to sense incoming light from all around. The photodiode is sensitive to light for wavelength from 400 to 1100 nm, covering the complete visible light spectrum. The sensor has a sensitive area of $1\,mm^2$ and a half angle of 75°, marking the angle at which the sensitivity reaches 50 % of the peak sensitivity. This wide open field of view helps providing an omnidirectional "light antenna" with only four sensors.

The incoming signal is strongly amplified by a transimpedance amplifier to increase the maximum communication distance. The amplifier can be built with only a few simple components (resistors and capacitors) and an operational amplifier. It converts the photocurrent generated by the photodiode to a voltage that can be sampled by an ADC. Before amplification, the signal picked up from the photodiode is DC-filtered. Every photodiode is connected to its own amplifier to keep the paths on the board as short as possible and thus the signal less prone to noise and interference. The signals from the four amplifiers are fed into four different GPIO pins that are connected to the ADC of the VLC controller. There is only one ADC available on the microcontroller and it can only convert one value at the time. The necessary changes within *libvlc* to multiplex light sensing is described in the following section about the VLC controller. Figure 4.2, part 3 shows the

4. https://octopart.com/sfh203p-osram+opto-56009445

PCB hosting the light sensors (3j), transimpedance amplifiers (3l) and VLC controller (3m). It connects to the SoC board via a 12-wire interface (3k).

System-on-a-Chip Board

Each light bulb includes a SoC board from DPTechnics.[5] The module consists of a Quaclcomm Atheros (AR9331) SoC, a 400 MHz Microprocessor without Interlocked Pipeline Stages (MIPS) processor with built-in Wi-Fi including an on-board antenna, 64 MB of Random-Access Memory (RAM) and 16 MB of flash memory for persistent storage to host the operating system. The available USART interface is used to connect to the VLC controller. In addition, the module provides 20 GPIO pins directly operable from Linux. By default, the SoC runs OpenWrt, an embedded Linux distribution specifically adapted for routers and other networking devices. Since it is a complete Linux distribution it ships with the entire Linux network stack, providing a complete network- and transport-layer to be used for VLC. A Linux kernel driver module, further explained in Section 4.1.3, interfaces the VLC controller with the Linux network stack.

The Wi-Fi connectivity provided by the SoC is useful for a testbed environment. The wireless radio channel can be used as a control channel. The devices can be deployed and later configured and reprogrammed (SoC operating system and VLC controller firmware) remotely. Further, measurement series can be remotely controlled and data can be collected without disassembling the testbed again. It is important to know that the objective was not to build light bulbs that communicate by using a Wi-Fi channel. The light bulbs are meant to interact with each other using VLC only. Some of the provided GPIO pins are connected to the VLC controller to enable firmware upgrades using ICSP emulation via a software tool called `avrdude`[6]. It also supports serial programming via the bootloader.

Figure 4.2, part 2 shows the board hosting the SoC module (2g) from both sides. It provides several connectors to hook up the

5. https://www.dptechnics.com, DPT-MOD-001
6. http://www.nongnu.org/avrdude/

VLC controller and sensor board (2d) and the two power supplies (2e). One powers all the electronics (part 1) and the other is responsible to drive the LED plate. The light output is controlled by the VLC controller using a Metal–Oxide–Semiconductor Field-Effect Transistor (MOSFET) (2f) placed between light bulb power supply and LED plate connector (2h). Additional electronics serve as a resetting infrastructure for the VLC controller and to convert logic-level voltages (the SoC module only supports a maximum of 2.5 V on the input pins).

Visible Light Communication Controller

The VLC PHY and MAC layers [81, 82], as described in Chapter 3, are software-based and encapsulated into *libvlc* and hosted on an 8-bit microcontroller. Instead of the Arduino Uno, a different board called Microduino Core[7], with smaller form factor and without USB-to-serial converter, but equipped with an identical microcontroller (ATmega328P) is used. It directly plugs into the sensor and amplification PCB, connecting to the amplified sensors and the Linux board.

The newly added hardware (sensors, LEDs driven via MOSFET) also need to be represented and their behavior implemented into *libvlc*. Consequently, the communication channel is further abstracted into two subclasses, `LEDChannel` and `SensorChannel`. The subclass responsible to handle the LED transceiver case, called `LEDChannel`, implements light modulation and sensing as discussed earlier. The `SensorChannel` provides the abstraction for the case where a sensor and an LED are combined. It implements light modulation and sensing using a driver pin (connected to MOSFET's gate) and a sensor pin (connected to the photodiode amplifier's output). An application using *libvlc* can decide whether to use an `LEDChannel` or a `SensorChannel`, hence the API needs to be extended as shown in Listing 4.1. It is not supported to mix the channel types.

Light modulation and sensing need to be implemented differently for a `SensorChannel`. Controlling the light source is now handled with a single pin connected to the MOSFET on the SoC

7. https://www.microduino.cc/product/core

Listing 4.1: Adapted *libvlc* channel API for *EnLighting*. The communication channel class is further abstracted and subclassed to either hold an `LEDChannel` or a `SensorChannel`. Corresponding infrastructure to create specialized channels is also added to the API.

```
/* adds an LED (only) communication channel */
void *vlc_add_LEDChannel(uint8_t id, const pin_t *cat, const pin_t *
     ano, LEDChannel *ch);

/* adds an LED and sensor communication channel */
void *vlc_add_SensorChannel(uint8_t id, const pin_t *sensor, const
     pin_t *driver, SensorChannel *ch);
```

board. The MOSFET's source is connected to the common ground of the system whereas the drain is hooked up to the LED plate's minus pole. Setting the gate pin to high connects the ground with the LED plate's minus pole, closing the circuit and therefore enabling the light. Doing the opposite, providing a low signal, disconnects the ground from the LEDs, consequently disabling the light. Instead of heaving a cathode and anode as before which need to be set to the right mode and output level, a single pin, called driver pin, can be used to control the light, independent from the sensing.

The amplifier circuit connected to the photodiode generates a light intensity dependent voltage signal (previously done by the charge and discharge phase of the LED used as receiver). This signal can be sampled directly with the microcontroller's ADC. Figure 4.3 shows the modified light sensing procedure implemented for a `SensorChannel`. The interval distribution and duration is kept the same to ensure compatibility with the `LEDChannel`. The light is measured once at the beginning and once at the end of the COM slot. It is obvious that these measured values can be used as input for the already discussed synchronization algorithm. They even provide a more accurate starting position since they represent light values captured directly at the slot boundaries. During the two data intervals, the sensor is sampled multiple times and the collected measurements are averaged to provide a single

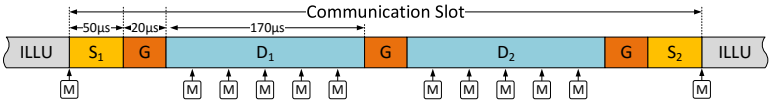


Figure 4.3: COM slot sensor light measurement procedure. The interval structure is kept the same for compatibility. The light is measured at the beginning and at the end of the COM slot to get input values for the synchronization algorithm. The sensor is sampled multiple times during the two data intervals and subsequently averaged to have a single value per data interval.

value for each, D_1 and D_2. These two values can be used by the bit detection procedure as before. To summarize, the only changes to *libvlc* are the introduction of the `SensorChannel` and how and when light measurements are collected. The result (the collected values) have the same format in the end and the specific channel type used is completely transparent for the rest of *libvlc*.

The API provided by *libvlc* already supports the creation of multiple channels, which is now required for the light bulb since it employs four individual sensors pointing in different directions. The driver pin can be shared between the four channels to operate the single available light source. The microcontroller's ADC can only convert one voltage value at the time and is also not fast enough to provide a series of values during a short time period. To sense light from the four directions (almost) at the same time, the ADC is multiplexed in software within *libvlc's* PHY layer. Figure 4.4 illustrates the used technique. A system with four available channels is shown. The curves show the signal received by the corresponding photodiode channel. The COM and ILLU slots are shown as reference. While in idle mode, *libvlc's* PHY layer cycles through all available channels, switching to the next channel at the end of a COM slot. The active channel is highlighted with a light blue color. If the channel is switched, also the corresponding sensor pin is switched and the ADC is configured to take readings of this specific pin. When detecting a signal, meaning that either during interval D_1 or D_2 light is detected, the PHY layer does not switch to the next channel and stays at the same channel for the

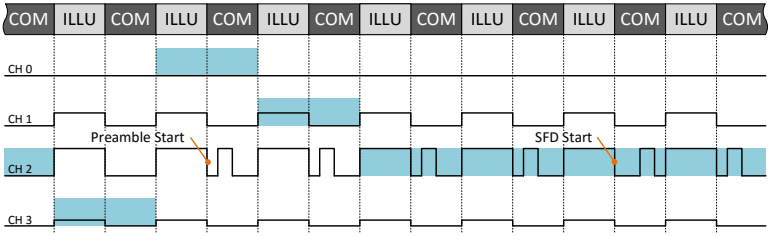


Figure 4.4: Light bulb channel multiplexing. Four channels and the signal from the corresponding photodiodes are shown. When in idle mode, *libvlc's* PHY layer cycles through all available channels, always switching to the next at the end of a COM slot. The active channel is marked with a blue overlay. If a busy channel is detected (light during D_1 or D_2), the channel is not switched at the end of the slot. Using a preamble of zeros, with at least the length of the number of participating channels, prevents channel switching at the beginning of a frame and guarantees that the PHY layer tries to receive from a channel with enough signal strength.

next ILLU and COM slot pair. A valid signal is detected when the difference of the light measured during D_1 and D_2 is above a configurable threshold. This makes the system stay on the same channel when receiving data, but does not guarantee that a signal is captured from the beginning, possibly missing the start of an SFD. To prevent this, a preamble precedes the actual SFD and PHY layer frame. It consists of multiple bits 0 (bit 1 could also be used, since the channel just needs to stay busy), which tune the system into a channel with a valid signal before the SFD is transmitted. The preamble has to be at least as long as the number of available channels. This does not guarantee that the channel with the highest RSSI values is chosen, but provides a basic solution to automatically stay at a channel where the signal is strong enough for reception. After receiving the SFD, the PHY layer switches to the RX state, preventing further channel switches. The channel cycling is resumed after the reception is completed and the system has switched back to the idle state. There are many ways to optimize the channel selection when receiving, e.g., using a history

with channel numbers, RSSI values, and frame source addresses, but the version discussed here provides a simple and clean solution that works well enough.

Light Bulb Casing

The off-the-shelf light bulbs are tightly packed. To create additional space for the modifications, casing extensions are designed and 3D-printed to provide room for the newly added parts. Figure 4.5 shows the different components and the fully assembled light bulb. The bottom casing shown in part 1 hosts the power supply (1c) installed at the lower part of the case and held in place by a crossbar. The sensor board with the attached VLC controller (1a) sits on top of this crossbar, tightly fitting into the casing. The holes (1b) on the side provide room for the photodiodes. On top, there is space to exactly fit the SoC board, which is held in place without screws or glue. The bottom casing is directly screwed to the light bulb socket (part 3) using the holes previously occupied by the screws securing the LED plate. On top sits an additional casing (shown in part 2), accommodating the LED plate (2d) and the heat sink (2e), which is directly attached to the plate. Additional slits (2f) provide improved air flow. The diffuse bulb is screwed on top of the LED plate and held in place by a 3D-printed thread. The top case is attached to the bottom part with screws and incorporated nuts, using the designated holes (visible in part 1). Part 3 shows the original light bulb socket with the power supply for the LED plate. It is replaced with a more sophisticated substitute to reduce interference with the transimpedance amplifier.

The fully assembled light bulb is displayed in part 4. The new casing contains all the necessary parts for full operation. It seems clunky but works as a prototype. When designing a VLC-enabled light bulb from scratch, the electronics could be designed to fit into a smaller more elegant enclosure. Indeed, it is also possible to build a VLC-capable illumination device with another form factor, but the light bulb has two important advantages: First, it is compact and comes in one single part. Second, the only additional part to operate a light bulb is a lamp. Lamps come in many

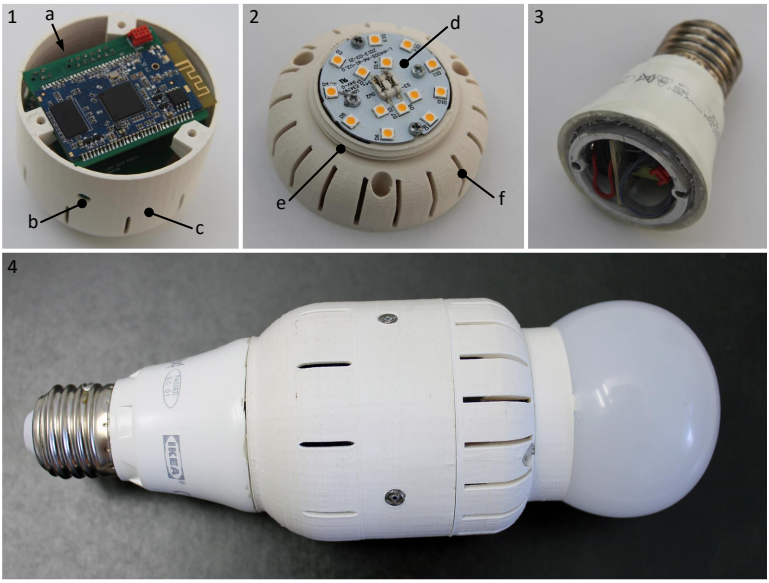


Figure 4.5: 3D-printed light bulb casings. Part 1 shows the bottom casing hosting the additional power supply (1c), the light sensors (1b), the VLC controller board (1a), and the SoC board. Part 2 depicts the top casing with slits (2f) for additional air flow, housing the LED heat sink (2e) and the LED plate (2d). Part 3: original light bulb socket with integrated (replaced) power supply for the LED plate. Part 4 presents the completely assembled light bulb.

flavors and sizes and can be arranged effortless for arbitrary scenarios within a room or building.

4.1.3 *Linux Integration*

The transparent integration of the VLC communication channel, based on *libvlc*, into the Linux networking stack demands the implementation of a data link layer, interfacing with both sides, Linux and the VLC controller. As illustrated with the left part of Figure 4.6, the data link layer consists of three core functional blocks: the VLC network driver, *libvlc's* MAC protocol, and USART

communication interface, which interconnects the two aforementioned blocks through a serial link. This section describes the design and implementation of the Linux-based network driver for the VLC controller.

The VLC Linux network driver is implemented as a loadable kernel module and has two main functions. It handles communication over the USART interface with the VLC controller, and it abstracts the VLC device within Linux as a networking device, interfacing with the Linux network stack. The driver implements a full-fledged serial stack and communication protocol. On the other side, the VLC controller runs an application on top of *libvlc* to handle the serial communication protocol, exposing *libvlc's* API. The serial communication protocols implements commands to, e.g., transfer a packet between the VLC controller and the driver, or to initialize and startup *libvlc*. The serial communication interface is capable of reaching data rates higher than 100 kb/s which are significantly higher than the rates achievable with VLC and therefore should not lead to bottlenecks.

The other part oft the VLC network driver deals with transparently integrating the VLC communication link (based on *libvlc's* PHY and MAC layer) into the Linux network protocol suite. It abstracts the VLC controller as an Ethernet-class network interface within Linux. Having this abstraction in place enables the transparent usage of the VLC link by any IP-based Linux program or tool. Programs such as `ping` or even `ssh` can be used without knowing the underlying MAC or PHY layer. Incorporating VLC into new programs can be done by using standard sockets[8] as communication endpoints. Most programming languages directly support UDP and TCP based on sockets, facilitating rapid development of new VLC-based applications.

The network interface registers itself within specific kernel data structures [13], to be invoked when packets are exchanged with the outside world. It responds to asynchronous requests received from the outside (incoming packets from the VLC controller), as well as to requests triggered by the kernel for outgoing packets to the MAC and PHY layers. The right part of Figure 4.6 fur-

8. https://en.wikipedia.org/wiki/Network_socket

ther explains the driver's functionality and implementation. The interface to the Linux network stack consists of two functional blocks, which are inherent in every Linux-based network driver: the `net_device` and the `sk_buff` data structures. The `net_device` represents the driver module within Linux. It is used to initialize and to disable the network device. Additionally, a number of device parameters can be set and initialized through this data structure, e.g., the hardware address and the name of the network interface as it is exposed to user space.

Network packets are transferred between the driver and the Linux network stack packed into the `sk_buff` data structure. A callback function captures all incoming packets from the Linux network stack. The content of every arriving `sk_buff` is sent to the VLC controller using the implemented USART communication

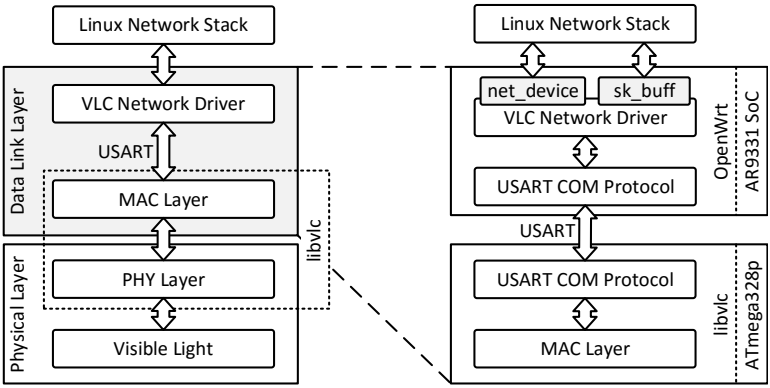


Figure 4.6: VLC controller integration with the Linux network stack. The left part shows the PHY and data link layer (containing *libvlc's* MAC layer and the Linux driver) interfacing with the Linux network stack on the top. The right part presents a zoomed in look of the data link layer. The application implemented on top of *libvlc* provides a USART communication protocol connecting to the counterpart on the driver's side. The driver communicates with the Linux network stack via two data structures: `net_device` is used to control the network device and network packets are transferred wrapped into `sk_buff` structures.

protocol. On the controller's side it is dispatched to the MAC layer using the API provided by *libvlc*. When receiving data on a VLC link, the payload is forwarded over the USART link to the Linux driver, where it is wrapped into an `sk_buff` structure and committed to the network stack.

The Linux driver and the USART communication protocol interconnects *libvlc's* lower communication layers with the higher layers of the Linux network, providing direct access to a well-tested networking infrastructure with all its tools and programs. Additionally, it distributes the computational load, keeping the VLC controller simple and low-cost.

4.2 ENLIGHTING TESTBED

After designing and building a communication system, it is usually tested and evaluated. The testing and measuring process can be simplified by building custom testbed software tools. The tools can be employed to configure the system, run tests and measurements, graphically visualize results, and to apply software updates. Certain configuration steps can be automatically repeated or applied to multiple devices all at once, saving precious time. Each light bulb employs a SoC with Wi-Fi connectivity, providing wireless access to each device. Well-designed testbed software reduces time spent for tedious system configurations and helps identifying and solving problems. This section describes the testbed infrastructure and software built for *EnLighting* which is used for software and firmware deployment and protocol debugging as well as to run measurement campaigns.

4.2.1 *Testbed Infrastructure*

Building, debugging, and running a testbed with devices connected directly to the power grid is complicated and dangerous. Wired connections from a computer to the light bulbs must be avoided. Also taking down all light bulbs and disassembling and reassembling them when updating software or collecting measurement results is not an option. Fortunately, the available wire-

less control channel enables remote actions, such as conducting administrative tasks, efficient debugging, software updates, and data logging for experiments and measurement campaigns.

All light bulbs (and the underlaying OpenWrt installations) are configured to join a dedicated wireless access point which integrates them into an IP-based network, enabling remote login into the Linux system. Having a wireless control channel also helps scaling testbed deployments, where individual devices are further apart and an extensive wired installation is not possible. While remote access to every light bulb gives great flexibility, additional software is necessary to simplify testbed management and to make running experiments more efficient. These software tools are explained in the following subsection.

4.2.2 *Testbed Software*

The testbed consists of three different software parts: The *EnLighting* web interface, the *EnLighting* server, and the *EnLighting* service. Figure 4.7 illustrates the communication infrastructure used to relay information between these program parts. The web interface provides a graphical user interface. It collects and visualizes information gathered from the testbed devices (light bulbs). Furthermore it can be used to trigger actions in a single or in multiple deployed devices. The actual implemented functionalities are explained later in this section. The web interface is implemented in JavaScript and uses websockets[9] to communicate with the *EnLighting* server. The server is also implemented in JavaScript using the *node.js*[10] environment. It can be run on the same machine as the web interface (control machine) or on a dedicated server within the same network. The *EnLighting* server is used as message broker to communicate with the *EnLighting* service running on each light bulb. The intermediate step via a server is required since the light bulbs run only a plain *node.js* environment without additional libraries to keep the memory and flash storage footprint small. The server uses UDP or TCP sockets (already in-

9. https://en.wikipedia.org/wiki/WebSocket
10. https://nodejs.org/

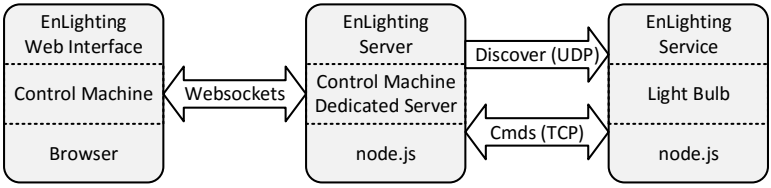


Figure 4.7: *EnLighting* testbed communication architecture. The testbed is controlled via a web interface from the control machine. It uses websockets to communicate with the *EnLighting* server running on the same machine or on a different dedicated machine in the same network. The server connects to the *EnLighting* service running on each light bulb via UDP (only for discovering) or TCP sockets to send and receive commands to trigger actions or gather information. Results are asynchronously reported via the server back to the web interface.

cluded in the *node.js* runtime) to communicate with the *EnLighting* service. To discover all available light bulbs, the server sends a discovery message to the network broadcast IP address. The light bulbs present (and running the *EnLighting* service) receive the discovery message and reply (using TCP) to the server. With receiving the reply, the server also picks up the IP address of the corresponding light bulb. The reply also contains additional information about the light bulb. The collected data is forwarded to the web interface where it is visualized together with all the other discovered light bulbs.

If a light bulb is discovered, its IP address is known to the system and the web interface can directly communicate with it (via the *EnLighting* server). Actions and data are transferred between web interface and light bulbs via commands. A command consists of the target light bulb's address, a unique command identification (number), and optional payload. A command is dispatched from the web interface via websockets to the *EnLighting* server where a new TCP connection to the target light bulb is opened. The command is wrapped into a TCP packet and relayed to the light bulb. If it is a onesided command, the TCP connection is closed by the light bulb and the action associated with the command is executed. If a response is expected, a reply command is

generated and sent back over the still open TCP connection. The server closes the TCP connection to the light bulb and forwards the command to the web interface, where the view is updated with the information provided by the reply command.

The web interface shown in Figure 4.8 gives an overview of the testbed functionality. The buttons on the top left side (a) are used to control the testbed. The discover button refreshes the light bulb view (c) and discovers new light bulbs joining the testbed. The light bulbs are represented as discs labeled with their hardware address. They appear yellow when the VLC driver is enabled and white when it is disabled. The discs can be moved around within the light bulb view to build a map of the testbed to faster find

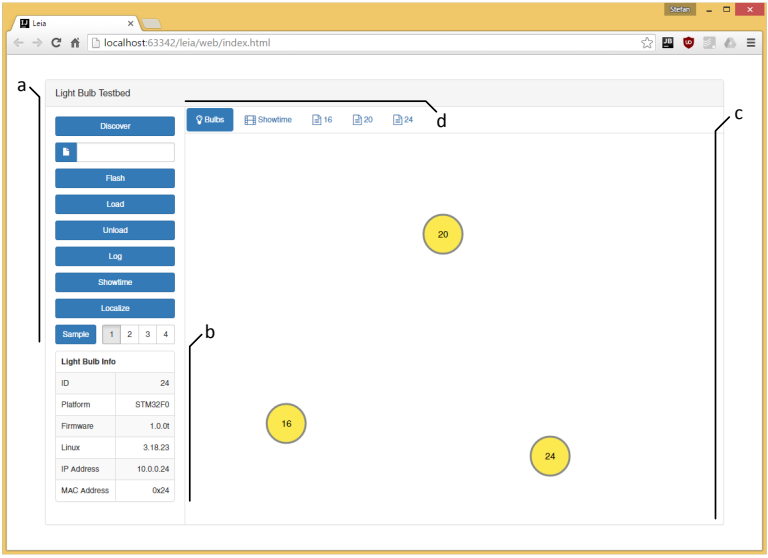


Figure 4.8: *EnLighting* web interface. The testbed is controlled using the various buttons on the top left side (a). The information area (b) below the buttons provide information about the currently selected light bulb. All discovered light bulbs are shown in the main area (c). The main area view can be exchanged using the tabs (d) to show device logs or a real-time packet visualization.

and correlate light bulbs from the testbed with their representation in the web interface. Other buttons can be used to upload new firmware to the VLC controller, load and unload the driver module, open the remote kernel log for a selected light bulb, or start *Showtime*, the real-time VLC packet visualizer. Single or multiple selected light bulbs can be a target of all these actions. The panel (b) below the buttons shows additional information for the selected light bulb. The light bulb view can be replaced using the tabs (d) with a view containing different information such as light bulb kernel logs or real-time VLC data traffic.

Figure 4.9 shows two additional screenshots from the *EnLighting* web interface. On the left side, the screenshot displays the view of an activated kernel log. It is updated in real-time and can be configured to show VLC packet statistics as explained in Chapter 3, which are printed to the kernel log by the VLC Linux driver. The kernel logs also provide additional information about the VLC driver module state. The screenshot on the right shows an activated *Showtime* view, the real-time VLC packet visualization. All activated kernel logs are scanned for VLC packet statistics and parsed to generate the *Showtime* visualization. The view can be panned and zoomed and provides a time-based (x-axis) view of all sent and received VLC packets, together with additional infor-

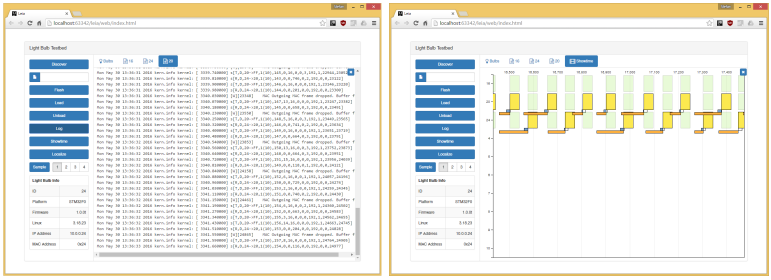


Figure 4.9: Additional web interface views. The left side shows an activated light bulb kernel log, updated in real-time. The right side shows the web interface with activated *Showtime*, the real-time VLC packet visualizer. All currently active kernel logs are parsed for packet logs and displayed in the *Showtime* view.

mation about the packets, such as size, source, destination, RSSI value, number of corrected errors, etc. *Showtime* can be helpful when analyzing protocol behavior when more than two devices are involved and reading and understanding console logs start to be complicated and confusing.

4.3 SYSTEM EVALUATION

Two dimensions were considered for an evaluation of the communication system based on the light bulb hardware and software discussed in this chapter. First, results for the light bulb communication link are reported. The measurements were conducted at Linux level for different protocols and for several network topologies. In the second part, a localization service built on top of *EnLighting* is described and evaluated to illustrate the flexibility offered by light bulbs that can send *and* receive.

4.3.1 *Communication and Networking*

A testbed built from four LED light bulbs as described earlier is used. Figure 4.10 shows a schematic representation of the testbed setup. Only neighboring light bulbs can communicate with each other. To send data from the sender to the receiver, intermediate light bulbs help to bridge the extended communication distance

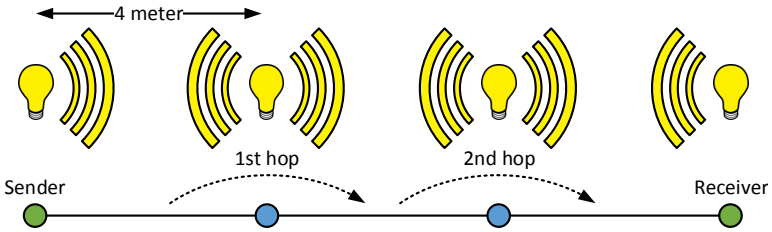


Figure 4.10: Schematic representation of a multi-hop light bulb testbed. Light bulbs are placed in a line at 4 m distance from each other. Data sent from the sender to the receiver is forwarded by intermediate light bulbs.

and forward the sent packets. The light bulbs are placed at a distance of 4 m from each other to still bridge a reasonable distance but also provide a stable communication link (see next section). The figure also explains the notion of multi-hop communication. If network packets are forwarded by one or two intermediate network node(s), the link is called a one or two hop communication link. If there is no intermediate node participating, and packets are directly sent to the receiver, the term direct communication is used. This scenario was chosen for evaluation since light sources arranged in a line (corridor) or grid (room) are common and could be exploited to transfer data between, e.g., a gateway light bulb and other devices further away using multiple hops.

The testbed setup entails the hidden station problem, considering that each light bulb can only communicate with and receive from its direct neighbors. Since this evaluation focuses on a proof of concept for IP communication over a VLC channel, demonstrating that reliable communication is possible using inexpensive devices, the experiments conducted in this evaluation are designed so that (most of the time) only one packet is in transfer and hidden stations are not an issue. The MAC protocol used in *libvlc* also implements an RTS/CTS extension to decrease the collision probability in case hidden stations are present. Due to the nature of the conducted experiments, this extension is disabled for the conducted measurements.

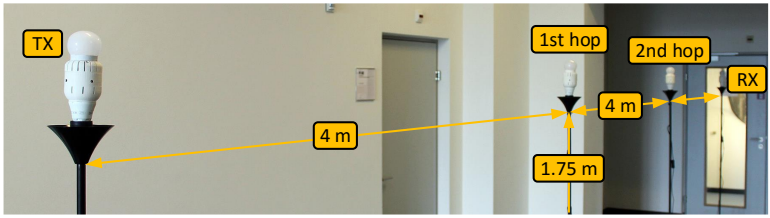


Figure 4.11: Deployed light bulb multi-hop testbed. The light bulbs are placed in floor lamps at 1.75 m above the floor level. The torchieres are setup in a line with a pitch of 4 m between each light bulb. A light bulb sensors orientation towards the next light bulb is chosen with a 45° offset (the middle between two sensors).

The setup of Figure 4.10 is realized by a simple testbed shown in Figure 4.11. Each light bulb is placed in an off-the-shelf floor lamp (torchiere) with a height of 1.75 m. The lamps are placed in a line, each bulb four meters apart from its neighbors. This setup also illustrates a benefit obtained by using modified consumer light bulbs. They can be placed in any available lamp with a fitting socket. The lamps again can be placed anywhere in a room as long as a power connection is available, shaping a flexible and uncomplicated testbed.

Experiments for direct link and multi-hop topologies, for different network traffic types (ICMP, UDP, and TCP), and different communication ranges, are explained and commented in the following sections.

ICMP Network Traffic

To better understand the possible communication range, direct link performance for different communication distances is evaluated. ICMP data traffic is used to measure the Round-Trip Time (RTT) for bidirectional data exchanges using `fping`[11] (a more powerful version of the standard `ping` program). Only two of the testbed light bulbs are used for these measurements. All measurement results always show the payload at the specific protocol level and the VLC MAC layer payload for better comparison with previous results. The payload sizes on the Linux level are chosen so that MAC layer payload sizes are better comparable with earlier results. The maximum payload supported by *libvlc* is 200 B. Experiments are run for 5 min each, at different distances and for different protocol payload sizes. The default MAC layer parameters are used and FEC is disabled to better evaluate the PHY layer performance.

Results are depicted in Figure 4.12. The y-axis denotes RTT in s (and is scaled for better comparison with the multi-hop scenario results) and the x-axis denotes communication distance in m. As expected, a larger payload size leads to a longer RTT. Results for the RTTs are in the range from 1 to 4 s. The RTTs reflect the PHY layer bit rate of 1 kb/s (for the used COM and ILLU slot duration

11. http://fping.org/

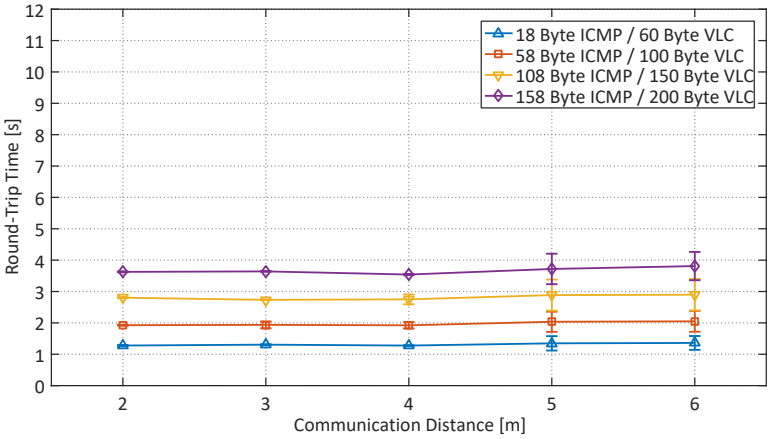


Figure 4.12: Light bulb direct link ICMP measurements results. RTTs for different communication distances and payload sizes are shown. Payload sizes are indicated for the current protocol level and the VLC MAC layer. RTTs lie between 1 to 4 s, depending on the payload size. Communication stays stable up to a distance of 6 m.

of each 500 µs). The communication link stays stable up to 6 m, a distance that is reasonable to be bridged by a VLC link within a room. The communication range could be significantly extended compared to the LED-only case (1.5 m, 1.3 m without FEC), thanks to the photodiode and amplifier circuit. One might also think that the higher light intensity of the light bulb contributed to the increased communication distance, but this is not true. Only the total light intensity is higher which is then distributed in all direction by the diffuse bulb covering the LED plate. This results in a similar light intensity per opening angle as for a single LED. There are only minimal changes for RTTs at different distances, but at five meters, the error bars (indicating the standard deviation) show more variation for the RTTs together with a slight increase of the average RTTs. This is a result of the longer distance and the therefore decreased RSSI value, introducing bit flips at the PHY layer. Lost packets are retransmitted by the MAC layer which increases the RTT significantly. At distances larger than 6 m,

the error probability increases profoundly so that no further measurements are possible.

The collected results show that stable communication is possible up to a distance of 6 m. Bridging this distance is sufficient to communicate with other devices within a (residential) room or small office. To reach further, e.g., to communicate within large open spaces, or to create communication links to neighboring rooms, multiple light bulbs and multi-hop communication can be used.

Figure 4.13 shows ICMP RTT measurement results for a direct link as well as for the one and two hop topologies (as described in the section's introduction), for different packet sizes, and a direct link covering a distance of 4 m. The y-axis denotes RTT in s and the x-axis denotes different payload sizes. To route the network traffic, static IP routes are used. Since the VLC controller is abstracted as a Linux Ethernet interface, any routing proto-

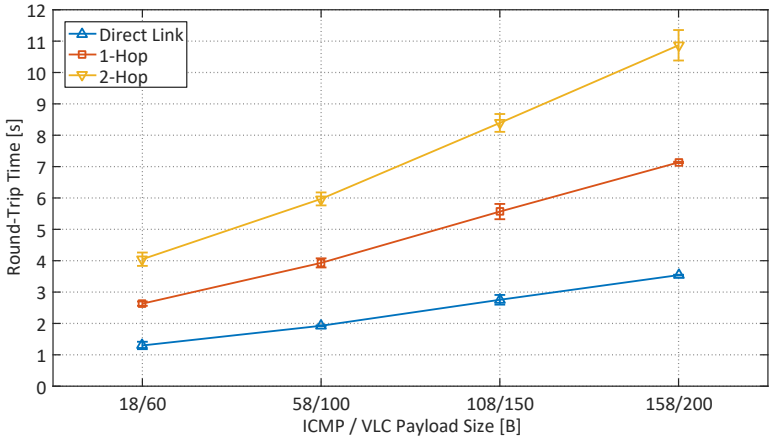


Figure 4.13: Light bulb multi-hop ICMP measurement results. RTTs for a direct link as well as for 1-hop and 2-hop topologies and different payload sizes are shown. Payload sizes are indicated for the current protocol level and the VLC MAC layer. The results show that, as expected, RTT roughly doubles for each additional hop and communication stays stable, also for larger payload sizes.

col available for Linux e.g., Optimized Link State Routing (OLSR), could be used to discover routes (with some parameter tuning due to the low PHY layer data rate when compared to Ethernet or Wi-Fi). The default MAC layer parameters are used and an experiment runs for 5 min for each payload size and number of hops. The results show that RTTs are approximately doubling for every additional hop, and communication stays stable also for larger payload sizes. Two hops are mostly sufficient to bridge large distances within bigger rooms and offices but the results also indicate that stable communication can also be expected for a higher number of intermediate hops, given that channel conditions are reasonable.

UDP and TCP Network Traffic

UDP and TCP measurements are executed with the well-known `iperf`[12] tool. The default MAC layer parameters are used and an experiment runs for 5 min for each payload size and number of hops. FEC is disabled to reduce the overhead since bit errors are less likely to happen at a distance of 4 m (see ICMP distance measurement results). To route the network traffic in multi-hop topologies, static IP routes are used.

Figure 4.14 shows saturation throughput for a direct link as well as for one hop and two hop topologies. The y-axis denotes data throughput in b/s and the x-axis denotes the different payload sizes at the Linux protocol level and for the complete VLC MAC layer payload. The error bars show the standard deviation. For small packet sizes the protocol headers are limiting throughput. For an 18 B UDP payload (Ethernet frames are padded up to a required minimal size), data throughput of approximately 200 b/s can be reached. For the maximum UDP payload of 158 B (200 B VLC payload), a throughput of 600 b/s is achievable. Compared with the MAC layer throughput result from the last chapter (900 b/s), 300 b/s less are measured. This is due to the additional protocol overhead introduced by the Linux network stack (42 B overhead for each UDP packet). Also, for one hop and two hop communication links, UDP data transfer still works, reaching a

12. https://iperf.fr

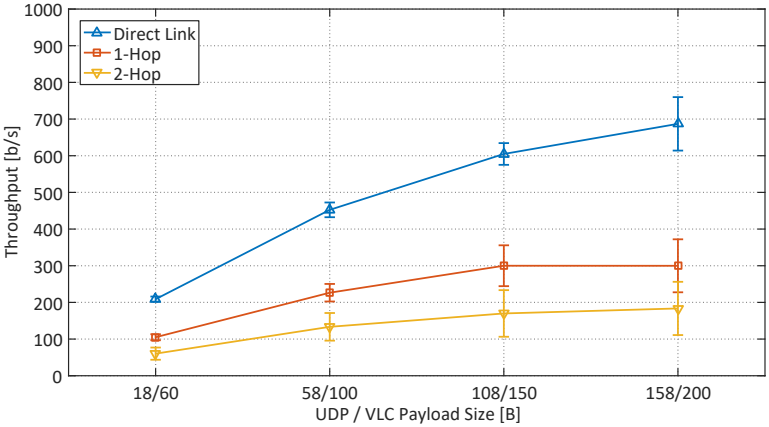


Figure 4.14: Light bulb multi-hop UDP measurement results. UDP data throughput for direct link, one hop, and two hop topologies and different payload sizes is shown. Payload sizes are indicated for the current protocol level and the VLC MAC layer. A stable UDP communication link over two hops with a maximum throughput of approximately 200 b/s is achievable.

maximal throughput also at the maximum packet size. For the one hop topology, 300 b/s is reached, and for the two hop case, a maximum of 200 b/s is possible.

TCP measurements are conducted for a static TCP window size of a single packet, i.e., the next packet is only dispatched if the corresponding TCP ACK has been received. The Maximum Segment Size (MSS) is also fixed to 34, 84, and 134 B, reflecting a 100, 150, and 200 B VLC MAC payload size. The measurement results are shown in Figure 4.15. The y-axis denotes throughput in b/s and the x-axis denotes the different payload sizes at the Linux protocol level and for the complete VLC MAC layer payload. The error bars show the standard deviation. TCP produces more overhead (mostly due to the additional ACK) than UDP which is visible in the achieved maximal data throughput. For a direct link topology, up to 400 b/s are possible. For one hop and two hop topologies 200 b/s, respectively 150 b/s, are possible. The throughput is low, but a stable TCP communication channel is available. Since

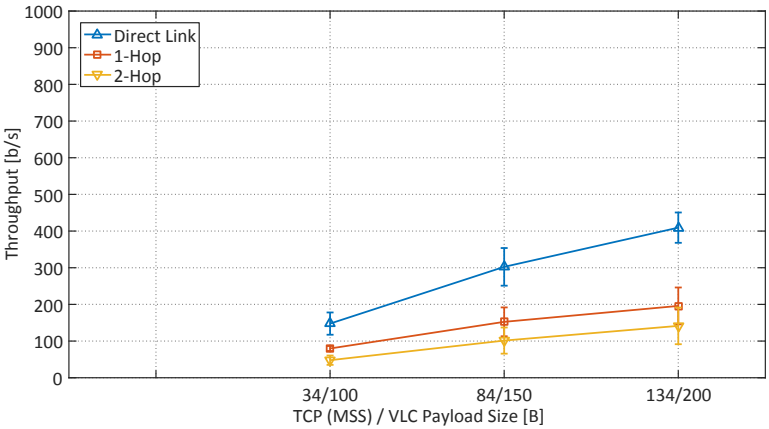


Figure 4.15: Light bulb multi-hop TCP measurement results. TCP data saturation throughput for direct link, one hop, and two hop topologies, a static MSS of 34, 84 and 134 B, and a static TCP window with the size of 1 is shown. MSS and VLC MAC payload size is indicated. A stable TCP communication link over two hops with a maximum throughput of approximately 150 b/s is possible.

there is always only one packet at a time transferred in the channel (due to the static TCP window with size 1), there are no collisions caused by hidden terminals.

These measurement results provide a proof of concept; they show that stable communication over a VLC channel using a standard Linux environment and protocols is possible. Employing faster microcontrollers could further improve the physical data rate, and protocol optimization (e.g., reducing protocol overhead for an IoT deployment) could increase the overall system data throughput.

4.3.2 *Indoor Localization*

In addition to basic networking, VLC-enabled light bulbs can also support other services. This section discusses and evaluates how a sample application for indoor localization is implemented on

top of the networked light bulbs. VLC has been used for localization in specialized scenarios (see Chapter 2) before. The results presented here provide evidence that the light bulb's ability to send *and* receive yields tangible benefits. The service is implemented at the Linux level and does not require changes in *libvlc*. Since the VLC link is transparently integrated into Linux as an Ethernet interface, any technology supporting IP sockets can be used as a basis. The application described in the following is based on the *node.js* platform.

Given a room illuminated by the LED light bulbs, the RSSI value (of a received packet) can be used to estimate the distance from the receiving to the transmitting light bulb. The RSSI is defined as the difference between a low and a high voltage level, measured by the ADC at the PHY layer when receiving a bit, and averaged over a complete data frame. The RSSI value is provided in the VLC controller's statistic logs, printed to the kernel log by the VLC Linux driver. Any application can read and parse the kernel log to retrieve the RSSI and additional information about the corresponding packet. The localization service running on every light bulb is repetitively transmitting a beacon with a fixed and configurable time interval. A beacon consists of a UDP packet sent to the broadcast address.

A receiver placed somewhere in the room can now estimate its position by looking at the received UDP packet, which contains the sender's address, and the kernel log to retrieve the RSSI value for the received packet. When receiving beacons from multiple light bulbs, it is possible to refine the estimated location with well known trilateration techniques [50]. The receiver can either lookup the location of the sender in a database using the received address, or the sender can directly provide its location as payload in the transmitted beacons. The system is deployed as shown in Figure 4.16. The left part shows a sketch viewed from above and the right part displays the deployed testbed. The light bulbs are set up in standard floor lamps at a height of 1.75 m and arranged in a triangle. Light Bulb (LB)1 and LB3 are placed 6 m apart from each other and LB2 is located at approximately 4 m distance from LB1 and LB3.

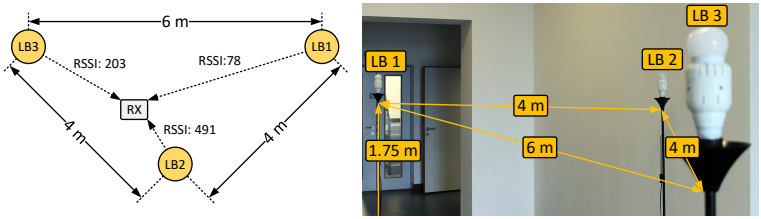


Figure 4.16: Indoor localization testbed setup. The left side shows a sketch of the testbed viewed from above. The augmented photo on the right shows the deployed testbed. The light bulbs are deployed in floor lamps arranged in a triangle and run the localization service. A receiver (RX) receives the beacons and records the corresponding RSSI values.

RSSI Quality Measurements

For the first measurement setup, only LB1 and LB3 are used, LB2 is disabled. With these measurements, the quality of the RSSI values is assessed to determine its behavior under different conditions and whether it is usable for localization or not. The light bulbs are sending location beacons at an interval of 1 s plus a random jitter between 0 and 200 ms. Since the light bulbs can send and receive, they synchronize to each other and also receive each other's beacons. A receiving device RX, based on the same hardware as the light bulbs and with only one photodiode and amplifier (also without LED plate and battery powered), is placed at different locations on the direct line between the two light bulbs (from 40 to 560 cm). At every position, the device receives and logs beacons for a duration 2 min. Furthermore, the receiving photodiode is pointed towards the ceiling, not favoring any direction.

Figure 4.17 displays the measured RSSI values for different receiver locations at a height of 100 cm, emulating a device held by a human. The y-axis denotes the RSSI value and the x-axis denotes the distance from LB1 in cm towards the location of LB3 at 600 cm. The measurements shown were recorded concurrently, i.e., both light bulbs are transmitting beacons at the same time. The RSSI values follow the inverse square law as already has been confirmed for the LED transceiver case and are plausible for values

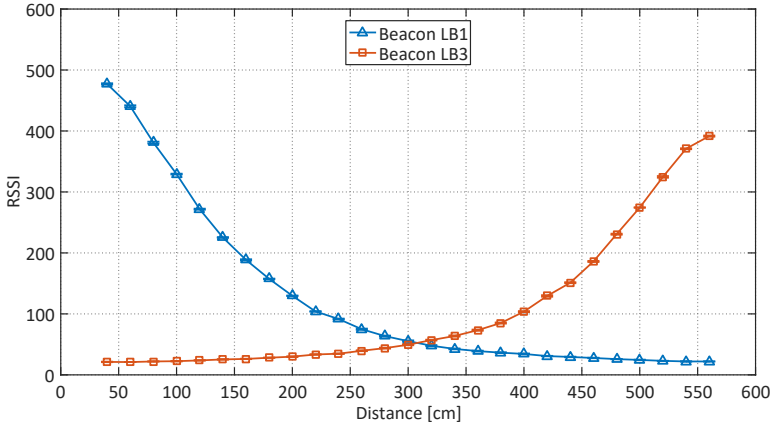


Figure 4.17: RSSI quality measurement results with enabled synchronization. RSSI values for a receiver moved stepwise from the position of LB1 at 0 cm to the position of LB3 at 600 cm at a height of 100 cm. Thanks to the synchronization and working MAC protocol, beacons are coordinated and can be received from both light bulbs at any position.

based on light intensity. The almost inexistent error bars (showing the standard deviation) also underline the stability of the measured values. Since the two light bulbs are synchronized, the MAC layer can handle medium access, and beacons only collide with a small probability. Therefore beacons from both light bulbs can be received at the same time (one shortly after each other). Even when the receiver is close to one light bulb, it is still possible to receive beacons from the light bulb further away. Having clean RSSI readings and not suffering from the capture effect, thanks to a working MAC layer (also in extreme conditions), helps refining the localization process and positioning accuracy.

Figure 4.18 shows the same experiment, but this time with *disabled* PHY layer synchronization. This setup mimics a VLC system in which the participating light bulbs can send but not receive. No real networking is possible and the light bulbs only act as broadcasting devices. As a result, beacons are not coordinated by the MAC protocol which increases the collision probability dras-

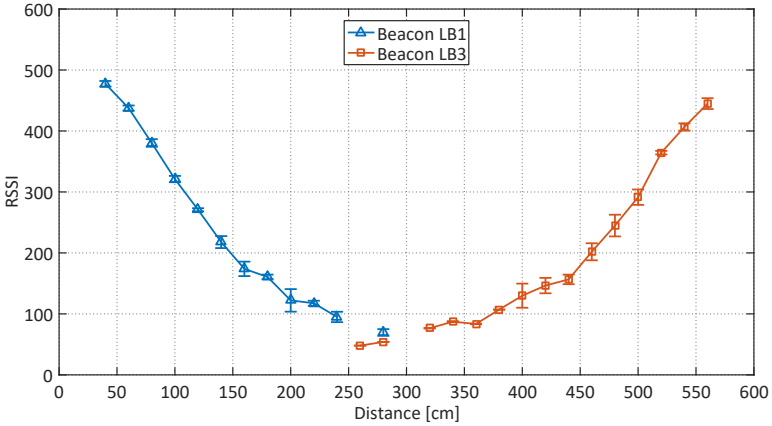


Figure 4.18: RSSI quality measurement results with disabled synchronization. RSSI values for a receiver moved stepwise from the position of LB1 at 0 cm to the position of LB3 at 600 cm at a height of 100 cm. Since synchronization is disabled, beacons are are not transmitted in a coordinated way and collide. Due to the capture effect, only beacons from the closer light bulb are received and RSSI values are falsified.

tically. The plotted results show that most of the time only beacons from the closer light bulb with the stronger signal are received. During a collision, the beacon from the closer light bulb might still be received due to the capture effect. Furthermore, the suppressed beacon provides additional energy, polluting the RSSI reading. This evidence is visible in the curve shape and the standard deviation depicted by the error bars.

These reported measurements show that the chosen RSSI metric and the proposed VLC system can be used for localization applications. Furthermore, the light bulbs' ability to receive clearly improves the quality of the RSSI readings and thus also enhances localization performance.

Towards Trilateration

Measurement results involving all three light bulbs shown in Figure 4.16 are reported in the following. The receiving device is

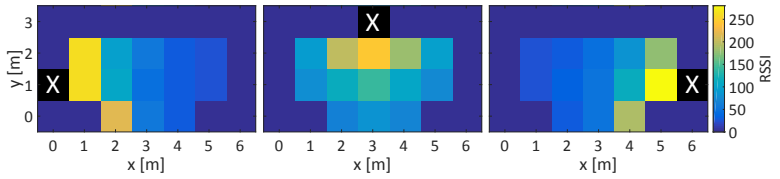


Figure 4.19: RSSI measurement results for three anchor light bulbs. A view from above is shown. The white X marks the position of each light bulb; from left to right: LB1, LB2, and LB3. The axis denote position within the room. The color code is a gradient from blue (low RSSI value) to yellow (high RSSI value). The measurements for the same square in each figure were collected at the same time when moving a receiving device around within the room. The results clearly suggest that the RSSI values can be used as input for a trilateration algorithm to estimate the location.

moved around at a height of 100 cm recording beacons from all light bulbs. PHY layer synchronization is enabled for all devices. Figure 4.19 summarizes the results. To improve the figure's readability, the RSSI values for each light bulb are shown in separate plots, but the data was recorded at the same time. The view is from above (the ceiling); the axes show the x-direction and y-direction (within the room) in m, and the white X denotes each light bulb's position. The results clearly show that the RSSI values are highest closest to the corresponding light bulb and decay over distance. The figure also shows that the light bulb networking scales at least up to three devices. When combining the three figures, it is obvious that the retrieved RSSI values can be used to determine the receiver's position by applying trilateration techniques.

4.3.2.1 *Practical Issues*

The previous section discussed the contribution of synchronization to the quality of the localization service. Because light bulbs can receive, the localization service can be extended with useful features for setting up and managing the lighting infrastructure. There is no need for any special wiring of the room (other than

to provide power), and configuration can be done through the optical links and does not interfere with any Wi-Fi or other radio systems that may be deployed. Furthermore, the light bulbs overhear each other's beacons and therefore can store the transmitter's IP addresses (could be used for light bulb identification) together with the corresponding RSSI values. If the lighting infrastructure provides a localization service and a light bulb fails and must be replaced, the replacement bulb (when inserted into the failed bulb's socket) can ask the neighbors for the predecessor's address. The neighbors can use their history of received beacons and RSSI values to infer a possible address for the replaced bulb. This capability simplifies maintenance and minimizes configuration errors (in case of single failures which should be the common case). Furthermore, light bulbs can also be configured wirelessly using a VLC link through a mobile light source (e.g., flashlight). Such a flashlight, programmed with configuration information, can simply be pointed towards the light bulbs that should be reconfigured or updated, providing an intuitive interface and connectivity.

4.4 DISCUSSION AND CONCLUSION

Networked light bulbs are combining illumination and communication. While for a human observer, the light bulb appears to illuminate its surroundings with a constant brightness, devices with appropriate sensors can receive data encoded into the light. *EnLighting*, the VLC system described in this chapter, provides a software-defined platform based on *libvlc*, which can be hosted on inexpensive hardware building blocks. LED light bulbs equipped with photodiodes offer an ideal platform for room area communication networks that also allow communication with low-cost LED-only systems.

Current SoC chips are powerful enough to run complex operating systems (e.g., Linux), and combined with a VLC controller based on *libvlc* and an off-the-shelf light bulb result in smart light bulbs that support higher layer protocols, such as TCP/IP. Without any tuning or customization, such a system provides stable connections and adequate performance for low-bandwidth ap-

plications in room area setups. *EnLighting* builds the basis for communication services and can be enriched with custom built applications that can be implemented without effort as demonstrated in this chapter.

The light bulbs can not only transmit but also receive, providing a bidirectional communication link. This property is crucial not only to support a wide range of two-way communication protocols but also for other application scenarios. As neighboring light bulbs can synchronize their activities, the number of collisions is reduced and the connections are stable. Furthermore, the ability to synchronize makes the system flexible. There is no need for another control channel or specialized wiring so that the light bulbs can be placed everywhere, in floor lamps or mounted on the ceiling, without any additional constraints.

As the number of connected devices increases, the available radio bandwidth stays constant and the available spectrum is becoming a scarce resource. For applications with low and moderate bandwidth demands, which is true for scenarios of the envisioned IoT, connected light bulbs can set up a room area network that provides a communication fabric. Supporting higher layer protocol stacks is important as it allows many other applications, that are built on top of these protocols, to work without modifications. Many issues remain to make the vision of the IoT a reality, but the proof of concept provided by the simple VLC system described in this chapter is an encouraging result that demonstrates that VLC could be a part of the technological foundation for the IoT.

5 ADAPTIVE VISIBLE LIGHT COMMUNICATION

VLC combines illumination and communication and is therefore an attractive technology for a ubiquitous indoor communication system. An already deployed lighting infrastructure can, without effort, be reused for communication purposes. Light sources built from LEDs are a logical choice for such a VLC system as LEDs are inexpensive, readily available, economical, and can be used to emit light (for data transmission) as well as to sense light (for data reception). Communication based on visible light provides further promising characteristics as discussed in earlier chapters.

To allow the use of VLC for emerging scenarios like the IoT, networked toys [12], and home automation, the endpoints of a VLC system must be as simple as possible. A minimal VLC node should be able to operate just with a single LED and survive on a battery for a reasonable amount of time. Chapter 3 introduced a system deployed on a simple 8-bit microcontroller. For scenarios where devices only need to communicate sporadically, an even simpler and smaller, less expensive, and more energy efficient processor could be used.

LED light bulbs combine multiple LEDs and are significantly brighter than single LEDs radiating light in different directions; they are low-cost [79, 84] and have been proposed for many novel indoor applications. To enable the sensing of incoming signals from other light-emitting devices, LED light bulbs can be enhanced with simple light receiving electronics based on photodiodes as further described in Chapter 4. Since these light bulbs are directly connected to the power grid, they have an adequate power budget and can operate faster processors. Furthermore, improved computational power paired with accurate light sensing capabilities can be exploited to introduce new and faster PHY layer data rates (PHY modes).

The key insight that allows different LED-based systems to communicate with each other (LED-only devices or LED-based light

bulbs) is to let software handle the communication protocols. The software-based PHY layer faces two core problems: First, all devices within range must be synchronized, i.e., agree when to engage in communication, and second, the analog signal detected by an LED or a photodiode must be sensed and digitized. Both of these problems, synchronization and sensing, can be accomplished in software as realized with *libvlc*, forging a flexible system.

This flexibility can now be exploited when introducing new processor types and new PHY modes to improve the achievable data rates. This chapter discusses the design, implementation, and reports on a practical evaluation of a VLC system (*libvlc* improvement) that dynamically adapts PHY data rates based on channel (environment) conditions and the capabilities of communication partners (not all devices can or need to support all available data rates). To support higher data rates, a faster processor needs to be supported. Also to optimize the system for a specific use case, different microcontrollers could be employed. To provide a basis for supporting multiple hardware platforms, a hardware adaptation layer is introduced, abstracting the necessary peripherals to run *libvlc*.

The chapter is structured as follows. Section 5.1 presents the design of a Hardware Adaptation Architecture (HAA) for *libvlc* together with the design of a new VLC controller board hosting a modern 32-bit microcontroller. Section 5.2 introduces higher PHY data rates and improved sensing techniques. The newly introduced PHY mode can be dynamically adapted, providing an optimal performance based on channel conditions and device capabilities. Section 5.3 evaluated the communication link an protocol for different network setups and device types. Section 5.4 summarizes and concludes the topics discussed in this chapter.

5.1 HARDWARE INDEPENDENT PLATFORM

A VLC system that connects devices, e.g., in the context of consumer devices and the IoT, must have minimal hardware requirements (so that low-cost/low-part count implementations are possible) yet allow inter-operation with other devices. A software-

centric approach is attractive as it isolates hardware dependencies and allows the reuse of the software base. The PHY layer implemented for *libvlc* was tailored for the ATmega328P processor. To increase the portability and decouple the software building blocks from the hardware, a general PHY layer implementation is introduced built on top of a device-specific HAA. The system design overview figure from Chapter 3 can be updated as shown in Figure 5.1. The sensor and amplifier RX introduced for *EnLighting* and the hardware abstraction layer are marked yellow.

The software-centric approach offers a communication system running on off-the-shelf and low-cost microcontrollers, but is still robust and stable, and fulfills the data rate requirements for scenarios requiring moderate data rates. Many devices already use a microcontroller and LEDs and could benefit from a hardware independent VLC software solution that relies on software changes only. The methods described in Section 5.2 highly benefit from the software-based approach. All changes described can be done in software and it is not required to exchange or add hardware to the existing systems. The following two sections describe the HAA and introduce a custom built system processor board based on an ARM[1] microcontroller.

1. https://www.arm.com

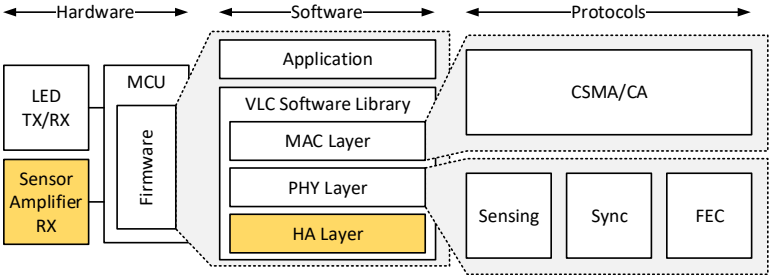


Figure 5.1: Extended VLC controller overall system design. Updated version of Figure 3.1 additionally showing optional sensor and amplifier RX path used by the *EnLighting* system, and the newly created HAA to support multiple microcontroller architectures.

5.1.1 *Hardware Adaptation Architecture*

To keep the software protocols hardware (microcontroller architecture) independent, the processors peripherals (such as timer, ADC, GPIO, etc.) are abstracted and the actual software (starting from the PHY layer) is built on top of this abstraction. The HAA consists of three layers between the hardware and application as illustrated in Figure 5.2 on the left side. While the Hardware Adaption Layer (HAL) and Hardware Presentation Layer (HPL) are platform specific, the Hardware Interface Layer (HIL) exposed to the application is platform independent.

The HPL is a thin layer sitting above the bare metal hardware, it presents the underlying hardware to the programmer in the form of human readable function calls and symbolic memory addresses and definitions. It does not keep any state and presents only available hardware features to the programmer. It is often supplied by the chip manufacturer or available as open source projects. A HAL is the implementation of the hardware interface for a specific platform, it is allowed to keep state and is built on top of the platform specific HPL. The HIL is a common interface to specific hardware components and peripherals; it sits above the HAL. To support a new hardware platform for an application, only the platform specific hardware adaption layer needs to be implemented, as the HPL is provided by the manufacturer, and the HIL as well as the application is platform independent. Due to its generic nature, the interfaces are narrow and hide hardware features that may not be available on all platforms or emulate missing hardware features in software on inferior platforms.

Figure 5.2, on the right, shows the HAA concept applied to two micrcocontroller platforms that are supported by *libvlc* (at the moment). The prototype board hosting the ARM processor from STMicroelectronics (STM)[2] is introduced in the next section. The HPL for the AVR microcontroller is part of the *avr-gcc* suite and the processor from STM uses *libopencm3*[3], which is an open source project covering HPLs for ARM processors of different manufacturers. The two HALs are implemented as part of *libvlc*, providing the

2. http://www.st.com/content/st_com/en.html
3. http://libopencm3.org

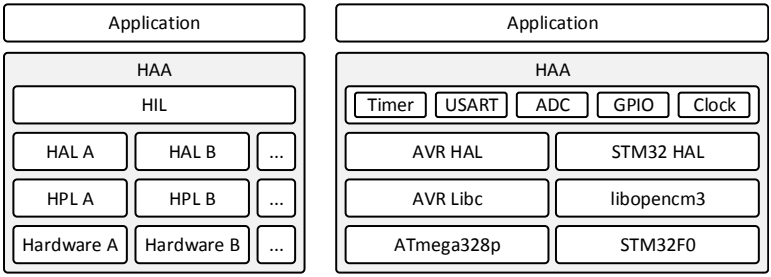


Figure 5.2: HAA for a hardware independent VLC controller. The left part shows a schematic visualization of the HAA based on three layers: HPL, HAL and HIL. Only the HPL and HAL are hardware specific, whereas the HIL presents a common hardware interface. The right part shows the HAA applied to the two hardware platforms currently supported by *libvlc*.

actual platform specific implementation that is then used by the HIL to provide a common interface for the available peripherals. There is almost no additional computational overhead since it is already known at compile time which HPL and HAL is used.

When introducing additional hardware platforms, it is only necessary to create a HAL for the new architecture and available peripherals. The application does not need to be changed since it only depends on the HIL. This decouples the software-based VLC protocols from the hardware, provides flexibility in the choice of hardware platform, and makes it possible to switch to new platforms with minimal effort.

5.1.2 *ARM Prototype Board*

The FEC runtime evaluation already demonstrated that the used ATmega328P processor is already close to its maximum computational capacity. Also *libvlc's* RAM requirements are close to the 80 % mark of the total available 2 kB of RAM, with not much left for the actual application running on top of *libvlc*. To support more complex applications and to enable the implementation of higher (faster) PHY modes, to increase data throughput as discussed in the following section, a faster processor is required.

The requirements are as follows: the processor needs to support all necessary peripherals (timer, ADC, GPIO, and USART for debugging and logging), and it should be available in a small package to fit on a small prototype board (to eventually fit into the light bulb). Additionally, the amount of available RAM and computational power should be increased when compared to the AVR platform.

Today, many low-cost and power-efficient microcontrollers are based on the ARM architecture. Many manufactures license the ARM and chip layouts to build their own microcontroller versions. STM is a chip manufacturer selling various ARM-based microcontrollers. The Cortex-M0 is the smallest available ARM processor in a similar price range as legacy 8-bit processors (like the ATmega328P) but with 32-bit performance. STM offers several Cortex-M0 types with different memory and flash sizes. All requirements can be met with the STM32F051[4] at even a lower price tag than the AVR-based model.

Table 5.1 lists the specifications for the ARM processor (the specs for the AVR processor from Table 3.1 are also shown for comparison). The microcontroller is based on a 32-bit architecture and can be clocked up to 48 MHz, providing a performance boost when compared to the 8-bit AVR processor. The necessary peripherals are all available and of a similar nature. The 64 kB of flash memory and the 8 kB of RAM can increase the complexity of application built on top of *libvlc*. Furthermore, the processor can be operated up to 3.6 V and GPIO pins provide up to 25 mA current. Since less voltage and less current is available per pin than for the AVR processor, the LEDs (when directly operated by the GPIO pins) are less bright, decreasing the maximum communication distance. There is no difference when using an LED driver, as it is the case for the introduced light bulb system.

STM also offers an evaluation board, the STM32F0Discovery[5], which includes pin header, LEDs, push buttons, and an on-board programmer. It is suitable for prototyping and developing but the form factor is too big to be included in consumer devices, toys, and light bulbs. Consequently a smaller prototyping board

4. https://octopart.com/stm32f051k8t6-stmicroelectronics-24958778
5. https://octopart.com/stm32f0discovery-stmicroelectronics-22099547

FEATURE	STM32F051	ATMEGA328P
Architecture	32-bit ARM	8-bit AVR
Operating Voltage	2.0 to 3.6 V	1.8 to 5.5 V
Current per Pin (max.)	25 mA	40 mA
Clock Speed (max.)	48 MHz	20 MHz
GPIO	25	23
Timer 8-bit	N/A	2
Timer 16-bit	5	1
Timer 32-bit	1	N/A
ADC	12-bit (two)	10-bit (one)
Serial Interface	USART (two)	USART (one)
EEPROM	N/A	1 kB
Flash Memory	64 kB	32 kB
SRAM	8 kB	2 kB

Table 5.1: STM32F051 specifications (ATmega328P specifications shown for comparison). The microcontroller uses an 32-bit ARM instruction set and can be clocked up to 48 MHz. it provides 25 GPIO pins and integrates timer, two 12-bit ADCs and serial communication peripherals. Further, 64 kB of program memory and 8 kB of SRAM is available.

with just the microcontroller and a few necessary electronic parts is designed, based on the STM32F0Discovery board schematics. The fully assembled custom-built PCB is shown in Figure 5.3. Top and bottom part of the completely assembled board are shown. The board has the same form factor as the Microduino Core used for the *EnLighting* system. Also the pin layout is plugin compatible, meaning that the ARM board can be used with the light bulbs sensor board without additional changes. The processor (a) is clocked with an external 8 MHz oscillator (b) which can be increased up to 48 MHz by a microcontroller-internal PLL. Two indicator LEDs (c) are available on the board. One is used as a power on indicator (green), the other (blue) is connected to a GPIO pin and can be used as desired. The board can be directly

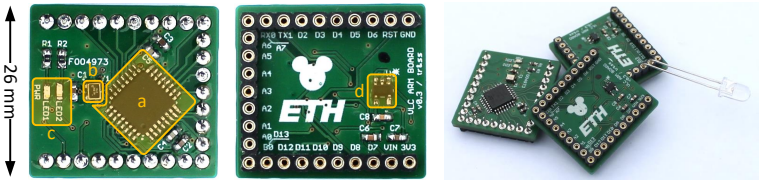


Figure 5.3: Custom designed ARM board based on the STM32F051 microcontroller from STM. Top and bottom part of the assembled PCB are shown; a. microcontroller; b. 8 MHz oscillator, can be scaled up up to 48 MHz by an internal PLL; c. (power) indicator LEDs; d. voltage regulator (3.3 V). The right part shows some assembled board together with a 5 mm LED for size comparison.

operated with a microcontroller-compatible voltage or via the onboard voltage regulator (d) which provides stable 3.3 V output for an input voltage of maximum 16 V. The right part of Figure 5.3 shows three assembled ARM boards together with a 5 mm LED for a size comparison.

5.2 ADAPTIVE ILLUMINATION AND SENSING

The modular and layered structure of the VLC software platform allows effortless porting of the VLC system to different microcontroller families and architectures. As of the moment, an AVR (ATmega328P) and ARM (STM32F051) are supported by the HAA. Neither of them currently reaches its limit in terms of computational power and memory. Therefore good channel conditions allow a more densely modulated medium (shortening data interval duration, at the cost of additional computation), which will increase the data rates of the system. This section describes how a fully adaptive system that dynamically adapts to current channel conditions and capabilities of participating communication partners can be built by extending *libvlc*. In addition to the adaptive hardware and PHY modes, an adaptive LED brightness control mechanism is introduced for the existing VLC protocol, briefly discussed in the following section.

5.2.1 *Adaptive Brightness Control*

With the VLC protocol introduced in this thesis (scheme shown in Figure 3.7), a lighting device outputs light only half of the time, being only half as bright as originally possible. If the light source only needs to provide static brightness, the lighting system can be designed such that a 50 % duty cycle is equal to the target brightness. An adaptive light source can follow the concept depicted in Figure 5.4. Light output can be reduced replacing ILLU slots with slots where no light is emitted or even replace ILLU slots with COM slots to increase communication capacity. To increase brightness, additional ILLU slots can be introduced at the cost of COM slots and communication performance.

For *libvlc*, only a few things need to be changed. Synchronization is not affected at all, also for devices at different brightness levels. As long as some ILLU or COM slots exists (edges from dark to bright or reverse), the system is able to synchronize. In case the clock drift between devices cannot be compensated anymore since fewer synchronization chances are available (when chang-

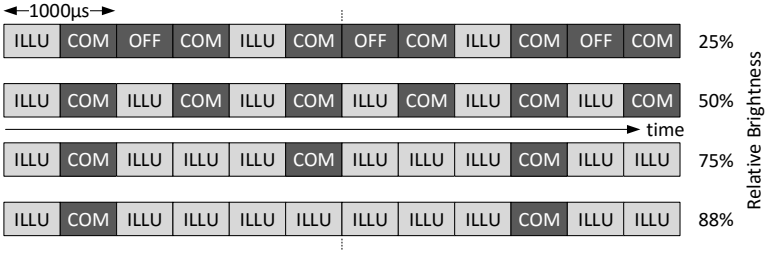


Figure 5.4: Adaptive brightness control. Light brightness can be controlled by adding or removing ILLU slots. Dark gray areas symbolizes no light output and light gray areas depict light emission. The second row (50 %) displays the normal behavior. The brightness can be increased by replacing COM slots with ILLU slots with the side effect of loosing data throughput performance. Deactivating light output for certain ILLU slots decreases the brightness. The scheme can be continued in both directions.

ing ILLU and COM slot ration), the synchronization step can be increased with the brightness change. When making the light source brighter, the number of COM slots is reduced, which needs to be known by a transmitter so that only available COM slots are used for transmission. To solve this problem, each device keeps track of all neighbors and brightness levels. If a device changes brightness, it informs all neighbors. Reception is not affected by brightness changes, also not in systems with mixed brightness levels, because light level differences and not absolute values are used when evaluating and decoding data intervals.

For a lighting system where the brightness of all participating (and communicating) devices is always changed simultaneously, a different approach can be applied. Changing the duration of the ILLU slots, whereas the duration of the COM slots stays constant, also affects the brightness. Obviously, prolonging ILLU slots lead to more light output and therefore increases brightness, and reversely shortening ILLU slots decreases the brightness level. ILLU slots cannot be shortened indefinitely since there has to be enough room for compensation while communicating to prevent flickering. Changing the ILLU slot duration does not affect communication and can be applied to *libvlc* without any changed to the rest of the communication protocols.

This section only presents initial thoughts and concepts. Brightness control is not yet implemented in *libvlc* and therefore can also not be discussed in the evaluation section of this chapter. Brightness control is an important feature of VLC systems and illumination devices and is therefore briefly addressed here for completeness.

5.2.2 *Physical Layer Modes*

To make use of available processing power and good channel conditions, the data interval duration within a COM slot can be shortened to construct COM slots with different numbers of data interval pairs to increase the PHY data rate. The resulting PHY modes are shown in Figure 5.5; different PHY modes can be built by changing the number of data intervals and their duration inside a COM slot. The top of the figure shows the `SINGLE` or `BASIC`

PHY mode; it is identical to the COM slot structure introduced in Chapter 3 and uses identical timings. By equally partitioning D_1 and D_2 into two intervals each, PHY mode `DOUBLE` is composed, containing four data subintervals $D_{1,1}$, $D_{1,2}$, $D_{2,1}$, and $D_{2,2}$. By repeating this procedure, the construction of PHY modes `QUAD` and `OCTA` is straightforward (see Figure 5.5). There are in total 16 subintervals for the `OCTA` PHY mode. Since the overall COM slot duration stays the same, the subintervals are getting shorter for higher PHY modes, e.g., 21 µs for an `OCTA` subinterval. Splitting into more subintervals is not useful since the necessary accuracy in synchronization cannot be achieved with simple hardware and software-base protocols. Shorter slots mean that less light is received (for LED-only systems), and the synchronization must be (more) accurate. Hence, the better the channel condition (mostly depending on the sender receiver distance) the higher the PHY mode that can be used.

The guard intervals (20 µs) are still in place to increase the stability of the `SINGLE` PHY mode. Also for the higher PHY modes they are still in place to simplify the protocol implementation (not influencing the achievable data rate) but are not necessary anymore. The duration of the guard intervals can also be decreased with minimal effect on the signal quality for the LED-only case (duration of data subintervals slightly increased). When receiving, the subinterval start and end times are slightly offset inside

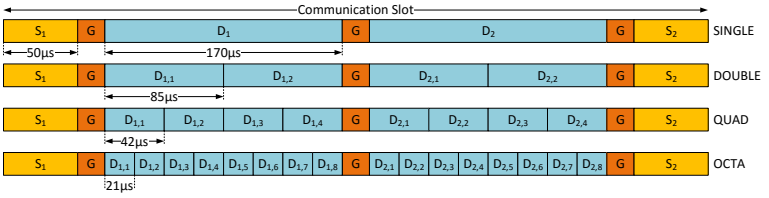


Figure 5.5: PHY layer data subintervals. The data intervals D_1 and D_2 are further partitioned into shorter subintervals to compose different PHY modes, called `SINGLE`, `DOUBLE`, `QUAD`,and `OCTA`. Construction is done by simply splitting one data interval into two, repetitively. More data intervals per communication slot increase the PHY data rate.

the actual subinterval, implementing short guard spaces to provide protection from light leaking into neighboring subintervals in case of imprecise synchronization. But in general, synchronization needs to be more accurate to successfully apply faster PHY data rates using shorter data subintervals.

To enable a completely dynamic PHY mode adaptation, each frame has to carry information about the used PHY rate. Consequently, the PHY header is always transmitted in PHY mode `SINGLE`, using two bits in the flags field to inform the receiver about the data rate of PHY layer payload. A receiver always starts decoding in `SINGLE` mode. After successfully validating the PHY header with the provided CRC the receiver switches to the required PHY mode to decode the rest of the frame. Furthermore, MAC layer ACKs are transmitted one PHY mode lower as the corresponding data frame, e.g., for a data frame received in `OCTA` mode, `QUAD` is used to transmit the ACK. Lowering the PHY mode increases the probability of a successful delivery and therefore prevents possible unnecessary retransmissions. This is important since channel conditions are not necessarily the same in both directions. A data frame can still be delivered (and repaired by FEC) using a particular PHY mode in one direction, but the ACK sent in the other direction with the same PHY mode cannot be received due to possible worse channel conditions and not available FEC.

Data Encoding

For all PHY modes, the two subintervals with the same (second) index encode a bit 0 or 1 together. Figure 5.6 shows an example of an `OCTA` communication slot. The dark gray background symbolizes lights turned off, whereas the light gray areas mean lights turned on. Below the slot visualization, the on and off signal from the modulated light is shown. The corresponding subintervals are labeled with the same numbers. A single bit is encoded with the two subintervals $D_{1,k}$ and $D_{2,k}$, for $k \in \{1, \ldots, N\}$. Corresponding subintervals are always encoded with inverted light levels so that for the decoder, a simple light value comparison is enough to determine the resulting bit. Spatially splitting up the corresponding subintervals makes the decoder also more robust

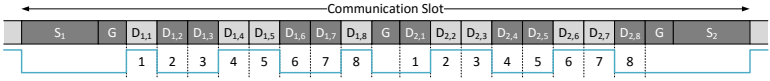


Figure 5.6: PHY mode data encoding. Example communication slot, showing 1 B of data. The first row displays the slot layout and the second row shows the on and off signal of the modulated light. Intervals $D_{1,k}$ and $D_{2,k}$, for $k \in \{1, \ldots, N\}$ together encode a single bit.

against light leaking into neighboring slots during imprecise synchronization.

Theoretical Speedup

The theoretical data throughput improvement $S_{N,P}$ for a given slot layout with N subintervals and a MAC layer payload of size P is obtained by dividing the throughput (theoretical MAC layer throughput) achieved with N subinterval pairs by the throughput of the base case with only one subinterval pair as stated by Equation 5.1.

$$S_{N,P} = \frac{R_{N,P}}{R_{1,P}} \tag{5.1}$$

The throughput $R_{N,P}$ itself is calculated by dividing the payload size (in bit) by the time $T_{N,P}$ it takes to successfully transmit that payload (and all included headers), as shown in Equation 5.2.

$$R_{N,P} = \frac{P}{T_{N,P}} \tag{5.2}$$

The time $T_{N,P}$ for a successful transmission of a payload of size P (using N subintervals pairs) depends on whether FEC is enabled or the frame is validated by CRC only. It is described by Equation 5.3 (FEC) and Equation 5.4 (CRC). H_{PHY} denotes the 4 B PHY header (including the SFD). The 4 B MAC header is accounted for by H_{MAC}. Depending on whether FEC is used or not, additional redundancy (E_{FEC}, 16 B) or E_{CRC} for a 2 B CRC is used. P signifies the payload size, which amounts to 200 B for maximum throughput

(as seen in previous results), and N is the number of subinterval pairs used in the actual PHY mode.

$$T^{FEC}_{N,P} = \frac{\overbrace{\left(H_{PHY} + \frac{H_{MAC} + P + E_{FEC}}{N}\right)}^{\text{Data}} + \overbrace{\left(H_{PHY} + \frac{H_{MAC} + E_{CRC}}{max(1, N/2)}\right)}^{\text{ACK}}}{C} \tag{5.3}$$

$$T^{CRC}_{N,P} = \frac{\overbrace{\left(H_{PHY} + \frac{H_{MAC} + P + E_{CRC}}{N}\right)}^{\text{Data}} + \overbrace{\left(H_{PHY} + \frac{H_{MAC} + E_{CRC}}{max(1, N/2)}\right)}^{\text{ACK}}}{C} \tag{5.4}$$

The first addend of the numerator in both equations describes the data frame containing payload P. Its MAC header, the payload and the redundancy or FCS is sent using the given number of subinterval pairs N, while the PHY header is transmitted with PHY mode `SINGLE`. The second addend in the numerator represents the MAC layer ACK, which needs to be received by the sender, before a data frame is considered to be successfully processed and a new frame can be prepared. The ACK does not contain any MAC payload and is validated via CRC only. It is transmitted using one PHY mode lower than the corresponding data frame (meaning that only half of the subinterval pairs is used to transmit ACK). The numerator calculates the number of COM slots needed to construct a complete data frame and ACK. A COM slot carries 1 to N bit depending on the used PHY mode. Therefore all data amounts in the formula are applied as a value in bit (multiplied by 8). Dividing the numerator by the number of COM slots per second (1000 for 500 µs ILLU and COM slot pairs), yields the time $T_{N,P}$ it takes to successfully transmit a data frame with payload P, using a PHY mode with N subinterval pairs. The interframe time and random contention window introduced by the MAC layer are neglected for this calculation, since only a benchmark result for comparison to real measurements is of interest.

Table 5.2 shows the theoretical transmission times, as well as the resulting throughput and maximum speedup, for the introduced PHY modes, based on Equations 5.3 and 5.4. The maximum PHY data rate, as opposed to the throughput, grows linearly

FCS				FEC			
N	T_N [s]	R_N [b/s]	S_N	N	T_N [s]	R_N [b/s]	S_N
1	1.76	909.10	1.00	1	1.87	854.70	1.00
2	0.94	1709.40	1.88	2	0.99	1612.90	1.89
4	0.50	3200.00	3.52	4	0.53	3030.30	3.55
8	0.28	5623.76	6.24	8	0.30	5405.41	6.32

Table 5.2: Theoretical transmission time, throughput and speedup, for a maximum payload P of 200 B, depending on the number of subintervals N; using a FCS (with CRC) or FEC. For both cases, a maximum theoretical speedup of more than 6 can be reached for 8 subinterval pairs.

with the the number of subinterval pairs N, reaching 8 kb/s for N=8, meaning that 8 bits are transmitted per COM slot. Since the PHY header is transmitted in PHY mode `SINGLE` and due to the overhead caused by the PHY, MAC, and FCS or FEC, the resulting theoretical throughput is significantly lower. For the maximum PHY mode (`OCTA`), a throughput of approximately 5 kb/s and a speedup of 6 is to be expected.

Receiving Strategies

Different receiving and decoding methods can be used depending on the receiving channel (LED or dedicated sensor) and the microcontroller's capabilities. Figure 5.7 summarizes the three approaches (for PHY mode `OCTA`, the same is valid also for the other PHY modes). The first example (1) shows the layout of a COM slot for a receiving LED channel. The first row (TX) shows the on and off signal of the modulated light from a transmitter, and the second row (RX) visualizes the reception logic. Since the LED is not paired with an electrical amplifier, the channel cannot directly be sampled. The LED is first charged in reverse bias and the remaining voltage is measured after a short period of time as introduced in Section 3.3.1. The remaining voltage correlates to the received light since the last charge. The blue arrows in-

dicate a charge and the red arrows denote the starting point of a voltage measurement. The subinterval control loop configures pins and handles the ADC inclusive result collection. For the PHY modes `QUAD` and `OCTA` a more powerful processor, like the introduced ARM microcontroller, is required to handle the processing within the short duration of a subinterval. The less performant 8-bit AVR processor can handle PHY modes `SINGLE` and `DOUBLE`. The blue and red arrows do not match with the subinterval bounds, but are slightly offset within the subinterval to provide additional guard space in case of an inaccurate synchronization.

The second communication slot (2) illustrates the implementation of a dedicated sensor channel. Here, the signal provided by the modulated light can directly be read from the output of the transimpedance amplifier. The microcontroller triggers an ADC sample (red arrow) at the center of each subinterval. Samples are collected and later processed at the end of the communication slot. Sensor-based receiving employs a less complex control

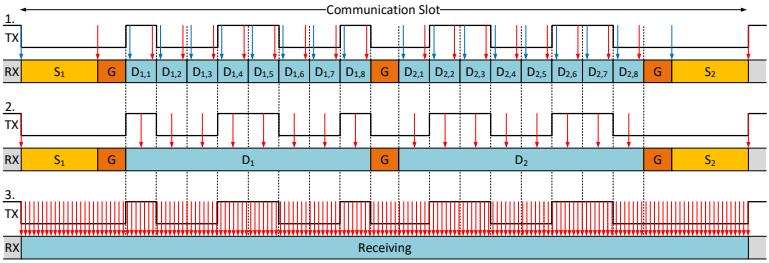


Figure 5.7: Receiving strategies for LED and sensor channels. 1. LED-based receiving. The LED is charged in reverse bias (blue arrow) at the beginning of the (sub)interval and the remaining voltage is measured (red arrow) at the end of the (sub)interval. 2. Sensor-based receiving. The sensor is sampled in the middle of each subinterval. 3. Sensor-based receiving with DMA. The complete COM slot is sampled and results are transferred to memory by DMA. Samples can be assigned to synchronization intervals and each data subinterval using the sample number (position in memory).

loop using fewer computation resources. Hence, all discussed PHY modes also work on less performant 8-bit processors.

For a microcontroller supporting DMA, the last shown method (3) can be applied. The complete communication slot can be simplified into one state. A timer is instructed to trigger samples (reading voltage values via ADC) during the complete slot. The conversion results are continuously transferred to memory via DMA without involving the processor. Using DMA allows faster sampling so that for a 500 µs COM slot 250 samples can be collected. They are processed later (during the following ILLU slot) where they are assigned to synchronization and data subintervals and averaged according to the active PHY mode. This method removes all processing from the communication slot, and the microcontroller can be used to process higher layers (MAC or application logic) during this time. This method also opens up space for further improvements discussed in the next section. DMA is supported by the introduced ARM processor from STM.

Synchronization Correction

The higher the PHY mode, the smaller the subintervals and therefore more precise synchronization is required. The synchronization is done in software and only relies on the simple method described earlier. Devices communicating over large distances (several meters), e.g., light bulbs, suffer from imprecise synchronization. Due to the lower received signal strength, the synchronization is easily influenced by noise and can shift several microseconds back and forth. This additional shift can be compensated when using the DMA receiving strategy as explained in the following.

Figure 5.8 shows such a scenario where the synchronization offset is ϵ. When using the DMA sampling method, this offset can be recognized and corrected. For the transmission of the PHY header, PHY mode `SINGLE` is used with the two data intervals D_1 and D_2. For all PHY header bits, there is either a falling edge at the end of D_1 or a rising edge at the beginning of D_2, depending on the transmitted bit. Since it is known where this edge should be inside the COM slot (when perfectly synchronized) and

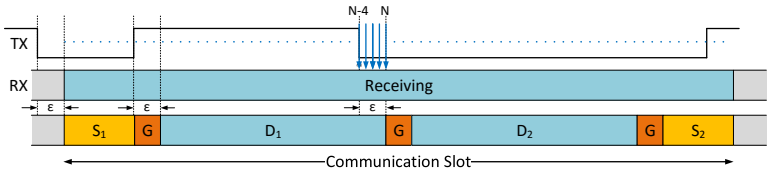


Figure 5.8: Synchronization offset calculation. Transmitter and receiver are slightly misaligned by ϵ. An offset to correct the sample assignment during the PHY payload can be calculated during the transmission of the PHY header (sent using PHY mode `SINGLE`). The edge is expected at sample N but detected at sample N-4. This offset is calculated and averaged for each bit of the PHY header and then applied to decode the PHY payload probably sent in a higher PHY mode.

the edge can be detected (at a resolution of 2 µs, 250 samples per COM slot) using the collected samples, the resulting offset (in samples) can be calculated and be considered when assigning samples to subintervals. The example in Figure 5.8 shows that the edge is expected at sample N but detected at sample N-4. This offset is calculated for all PHY header bits and averaged to be applied when decoding the PHY payload. The PHY header can still be decoded correctly, also if synchronization is significantly off. It is transmitted in the most resilient PHY mode where many samples are averaged to a single value per data interval. Outliers collected outside the corresponding data interval do not carry as much weight as in higher PHY modes where data subintervals are shorter.

Figure 5.9 displays the results of an experiment where the packet delivery ratio for a transmitting and receiving light bulb is measured. The light bulbs are set up at a distance of 5 m for a direct link scenario. One is used as transmitter, the other as a receiver. For all PHY modes, 100 data frames with a MAC layer payload of 200 B are sent. The frames are counted as delivered if they are successfully decoded at the receiver. The y-axis denotes the packet delivery ratio in % and the x-axis denotes the four different PHY modes. When using the standard synchronization method, for PHY mode `QUAD`, 20 %, and for PHY mode `OCTA`, more

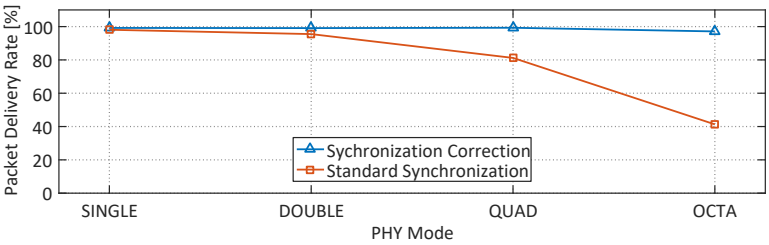


Figure 5.9: Packet delivery rate comparison with and without synchronization correction. Packet delivery rate is measured for 100 data frames with maximum payload for each PHY mode, once with standard synchronization, and once with enabled synchronization correction. Using the synchronization corrections drastically improves the successful reception for PHY mode `QUAD` and `OCTA`.

than 60 % of all data frames sent are lost. Applying the offset correction leads to drastically improved results: almost every packet is received successfully. This method demonstrates another way to improve the achievable data rate by selecting the correct samples.

Revised Physical Layer Header

Figure 5.10 depicts the revised PHY header based on the header introduced in Figure 3.12. The flags field is complemented with two additional values (marked yellow), each covering two bits. The three most significant bits are still reserved. Bit 3 and 4 encode the PHY mode used for the PHY payload (`00:SINGLE`, `01:DOUBLE`, `10:QUAD`, `11:OCTA`). Bit 1 and 2 define the capabilities (supported PHY modes) of the transmitting device. These two bits are always transmitted with every frame, informing all neighbors about the PHY modes supported by this specific device. The neighbor manager, a part of *libvlc*, is keeping track of all neighboring devices and their capabilities. The PHY header is always transmitted using the `SINGLE` PHY mode to allow the receiver to decode its contents and then switch to the requested PHY mode to receive the rest of the frame encoded with the requested PHY mode.

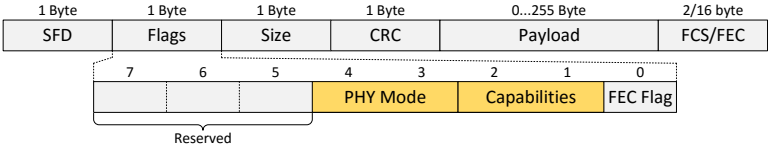


Figure 5.10: Revised PHY layer frame header based on the frame format introduce in Figure 3.12. Two values are added to the flags field (marked yellow), each covering two bits. The two PHY mode bits encode the PHY mode used for the PHY payload and the two capabilities bits announce the supported PHY modes of the transmitting device.

Dynamic Physical Layer Mode Adaptation

To be able to react to changes in the environment, the information available from collected packet statistics is leveraged to switch between PHY modes in an automated manner. One of the first published algorithms to make use of transmission and failure counters to adapt the transmission mode was Auto Rate Fallback (ARF) [35] developed for a predecessor of IEEE 802.11 Wi-Fi. This algorithm works by periodically checking whether a faster transmission mode would work, using so-called probing packets. Whenever a certain number of probing packets are transmitted successfully, the sender switches to the faster mode. Reversely, when the number of lost frames exceeds a given threshold, the sender scales back to a slower mode.

In a comparison of different adaptation schemes [57] and in the introductory paper for ARF [44], the disadvantages of ARF are elaborated. The major drawback of the algorithm is its inability to leverage long-term stability. ARF probes the channel capabilities frequently, regardless of the result of former inquiries. This behavior leads to an unnecessarily high number of failed probing packets, which diminishes throughput. Adaptive Auto Rate Fallback (AARF) tries to eliminate this problem by adapting the interval between probing. Whenever a given number of probing packets fail, the threshold determining when to send the next probing packet is doubled (up to a certain limit). This yields a substantial reduction in wasted probing packets in a stable envi-

ronment and still allows AARF to use a faster PHY mode, should the conditions improve. Since the AARF scheme is straightforward to implement and also successfully employed in other wireless systems, an adapted version thereof is built into *libvlc*.

5.3 EVALUATION

This section discusses measurements for different VLC-enabled devices and setups. LED-only devices with different capabilities, light bulbs communicating with LED-only devices, and light bulb networks are evaluated to show the effectiveness of the presented adaptive PHY layer together with the CSMA/CA-based MAC layer. All results show MAC layer throughput. The testbed setups are the same as already introduced in Chapter 3 and Chapter 4 when not stated otherwise. The following section addresses the issue about the MAC ACK timeout and FEC processing discussed in Section 3.3.2 again, this time for the newly introduced ARM processor.

5.3.1 *Forward Error Correction Processing Time Revisited*

As discussed earlier, the FEC processing time depends on the number of errors to correct and on the payload size. The results for the AVR 8-bit processor unveiled that the FEC processing time took too long for one or more errors to be compatible with the MAC protocol. The verification and error correction of a received data frame takes so long that the corresponding ACK cannot be sent within the necessary time to conform with the SIFS. The problem can be handled by releasing the ACK before the actual error correction, as soon as the information is available whether the data frame can be recovered (hoping that the frame's source address is not erroneous and the ACK can be delivered to the correct destination).

Since *libvlc* now supports a faster 32-bit ARM processor from STM, which is introduced in this chapter together with the HAA, the ACK timeout measurements are repeated to verify if the problem still exits. The results are shown in Figure 5.11. The time it

takes from the moment when a data frame transmission is complete until the corresponding ACK is received is plotted versus different payload sizes for different error numbers. Due to the 32-bit architecture and higher clock rate, the ARM microcontroller can handle the FEC processing within 83 µs (compared to 95 µs for the AVR processor, see Figure 3.14) for all payload sizes and error numbers. This ensures that the ACK can be dispatched in time so that it can be delivered after SIFS. For devices using the ARM-based STM processor, it can therefore be guaranteed that the MAC protocol requirements can be fulfilled (without dispatching the ACK before errors are corrected) and the ACK is sent to the correct receiver.

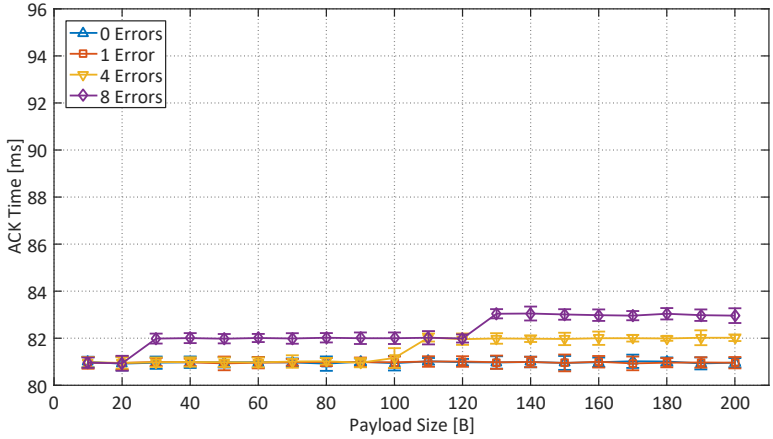


Figure 5.11: FEC processing time for STM ARM microcontroller. The time between frame transmission completion and the reception of a corresponding ACK is set in relation to frame payload size and introduced byte errors. The ACK time increases for higher payload sizes and higher error numbers. The additional time is introduced at the frame receiver where the ACK is generated, due to the FEC processing time. The error bars are indicating the standard deviation. The processing time for all combinations is short enough to ensure a timely ACK dispatch within SIFS. See Figure 3.14 for the previous results (AVR processor).

5.3.2 *LED-to-LED Communication*

This section discusses measurement results for LED-only devices using the same LED to send and receive. The four PHY modes introduced in this chapter are evaluated for single link communication and various distances and in a network with up to twelve devices. The experiments are mostly conducted with devices based on the STM ARM processor since this platform supports all available PHY modes. Some measurements also demonstrate the interoperability between both microcontroller platforms (AVR and ARM) using the capability announcements.

Single Link

The single link performance is measured in terms of throughput for an LED-only setup. Here, both sender and receiver are based on the ARM prototype boards, using a single LED each to transmit and receive. Except from the different microcontroller boards, the same hardware and testbed setup is used as discussed in Section 3.5 and illustrated in Figure 3.19. Messages of various sizes between 1 B and 200 B are transmitted from a dedicated sending device and acknowledged by a dedicated receiver. The default MAC layer parameters are used. The ACK is always sent with a more resilient PHY mode than the corresponding data frame was received with (e.g., if the data frame was sent with PHY mode `OCTA`, the ACK is sent using PHY mode `QUAD`). Data frames are transmitted at saturation, i.e., the next frame is immediately sent whenever the previous frame is completely processed (either acknowledged or the maximum number of retransmission reached). Only successfully acknowledged data frames count towards the throughput. The FEC threshold is set to 30, enabling FEC for PHY payloads with a length larger or equal to 30 B. Measurements are conducted for each of the four PHY modes, first fixed without dynamic adaptation. Experiments for various payload sizes and distances are executed for each fixed PHY mode. Each of those experiments (for each distance and each payload size) was evaluated for a duration of 5 min.

Figures 5.12 to 5.15 depict the results for fixed PHY modes `SINGLE` through `OCTA`. The y-axis denotes MAC layer throughput in b/s and the x-axis denotes distance in cm. The error bars visualize the standard deviation. PHY mode `SINGLE` (Figure 5.12) works reliably up to 160 cm. These results confirm the measurements from Chapter 3 depicted in Figure 3.20. The more accurate oscillator on the ARM board improves synchronization and therefore slightly increases communication distance about 10 cm. Using 200 B messages, a maximum throughput of approximately 850 b/s for the aforementioned distances can be achieved. This result matches the theoretical maximum throughput as calculated in Section 5.2.2. The maximum distance where successful

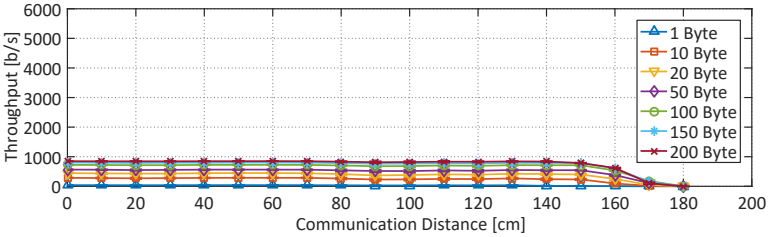


Figure 5.12: Single link throughput measurement results using PHY mode `SINGLE` for different packet sizes and variable distances. A maximum throughput of 850 b/s is achievable and communication is stable up to a distance of 170 cm.

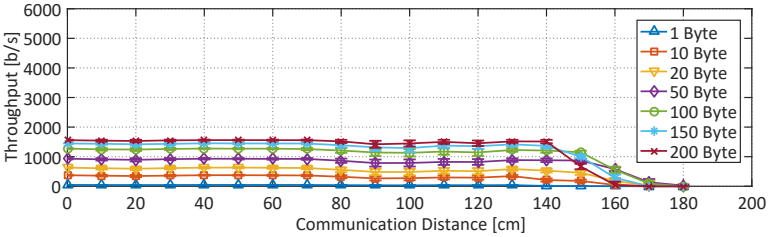


Figure 5.13: Single link throughput measurement results using PHY mode `DOUBLE` for different packet sizes and variable distances. A maximum throughput of 1.55 kb/s is achievable and communication is stable up to a distance of 150 cm.

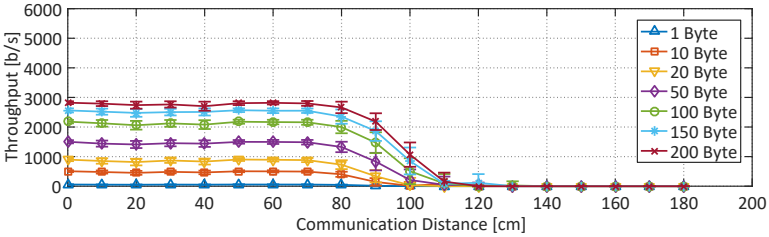


Figure 5.14: Single link throughput measurement results using PHY mode QUAD for different packet sizes and variable distances. A maximum throughput of 2.85 kbit/s is achievable and communication is stable up to a distance of 90 cm.

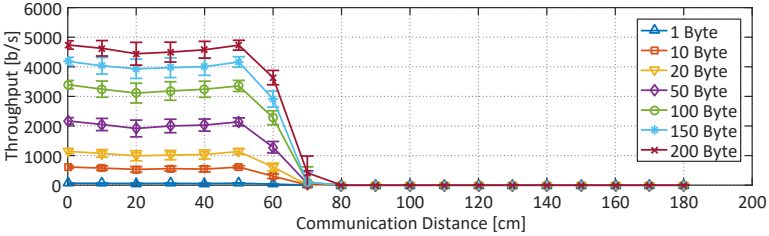


Figure 5.15: Single link throughput measurement results using PHY mode OCTA for different packet sizes and variable distances. A maximum throughput of 4.72 kbit/s is achievable and communication is stable up to a distance of 60 cm.

transmissions are still possible is at about 170 cm, albeit with a throughput of only 100 b/s.

PHY mode DOUBLE (Figure 5.13) achieves a maximum throughput of 1.55 kb/s up to a distance of 140 cm. Again, this result lies within the magnitude of the theoretical maximum of 1.61 kb/s, deviating only a few percent from the theoretical value. Communication is possible up to 160 cm, but with heavy losses in throughput. PHY mode QUAD (Figure 5.14) works reliably up to 90 cm with a maximum throughput of 2.85 kbit/s (more than 90 % of the theoretical maximum). Maximum throughput is achieved with PHY mode OCTA (Figure 5.15). The throughput has increased to 4.72 kbit/s, which is equal to more than 85 % of the theoretical

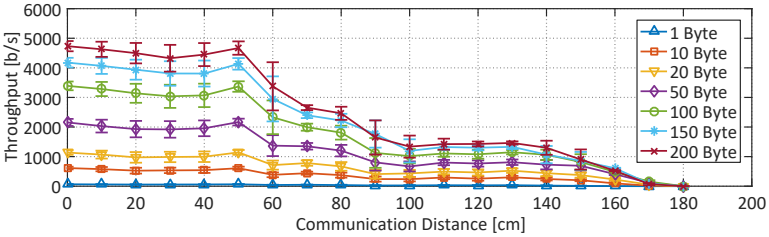


Figure 5.16: Single link throughput measurement results using AARF to dynamically adapt the PHY mode for different packet sizes and variable distances. A maximum throughput of 4.70 kbit/s is achieved at distances up to 50 cm. At the same time thanks to the dynamic PHY mode adaptation the communication stays reliable up to a distance of 160 cm.

maximum. The communication link stays reliable up to a distance of 60 cm.

The results can be summarized as follows: the higher the PHY mode, the shorter the communication distance that can be covered. Increasing the PHY mode leads to shorter subintervals and therefore less time to collect incoming photons. The light intensity decreases with communication distance so that fewer photons can reach the sensing LED. Hence, at a certain distance, not enough photons can be collected during a subinterval to significantly contribute to the LED's discharge and therefore no valid bit can be decoded. Additionally, larger payloads reach higher throughput because of the PHY and MAC layer overhead. The point at which shorter messages perform better than longer ones is reached when the probability for bit errors is so high that sending shorter messages with a higher framing overhead yields a higher expected throughput, due to fewer lost messages.

To evaluate the effectiveness of the dynamic PHY mode adaptation, the following results are compared with the results obtained with a fixed PHY mode. The throughput is measured again with the same setup as before, but without specifying the mode. Thus, *libvlc* uses the modified AARF algorithm to select the optimal PHY mode dynamically. The results of this experiment are shown in Figure 5.16. The adaptive PHY mode selection procedure always

starts at the slowest mode and probes for the applicability of the next higher mode after a couple of frames have been transmitted successfully. If the probing packet succeeds, the PHY mode is changed accordingly and an attempt to transmit faster occurs again some frames later. Should the probing packet fail, the current PHY mode is maintained and the waiting period for the next probing packet is doubled. This behavior has the advantage that it is always able to detect the best possible PHY mode for a certain distance (and signal strength), as becomes evident when comparing the graphs. One drawback of this tentative approach is the increased uncertainty in the transmission duration, and thus throughput, which is visible in the considerably higher error bars in Figure 5.16. Nonetheless it shows that the adaptive method succeeds to chose the best available PHY mode at all distances resulting in optimal throughput.

Network

In this measurement series, the adaptive PHY mode selection in a network of twelve devices arranged in a circle is evaluated. An LED is used as sender and receiver. The network consists of six AVR and six ARM boards, which are added one by one to the network in an alternating fashion. For each number of devices, network throughput for various payload sizes is measured. An experiment for each packet size lasts 5 min. The same testbed setup as shown in Figure 3.26 is used (with the described hardware changes). The AVR boards are capable of using PHY mode `SINGLE` and `DOUBLE`. The ARM boards support up to PHY mode `QUAD` in this configuration. These capabilities are set by the measurement application and used by *libvlc* to transmit at optimal rates (determined by AARF). For this measurement, all devices transmit at saturation, selecting a random receiver (not themselves) for each transmission. Furthermore, default MAC layer parameters are used and the FEC threshold is set to 30 B.

The measurement results are shown in Figure 5.17. The y-axis denotes throughput in b/s and the x-axis denotes the number of participating (and transmitting) devices. The error bars show the standard deviation. When only two devices (one AVR and one

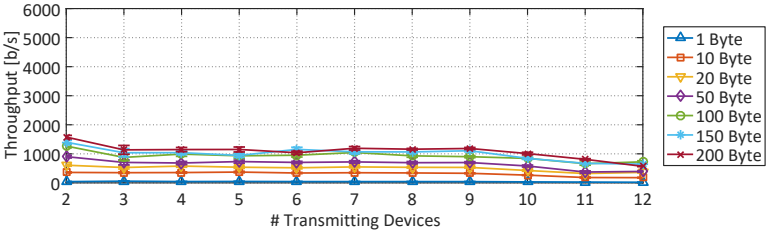


Figure 5.17: Network throughput measurement results for mixed (AVR and ARM) devices. AARF is used to dynamically adapt the PHY mode. Results for different payload sizes and for two to twelve transmitting devices are shown. A throughput of approximately 900 b/s can be achieved with twelve transmitting devices.

ARM board) are active, the maximum supported PHY mode by both platforms is PHY mode `DOUBLE`. The two devices can transmit using short contention windows, since collisions in this setup are unlikely, leading to a throughput close to the theoretical maximum of approximately 1.6 kb/s. Adding more devices to the network diminishes the throughput due to the higher probability of collisions. The stable behavior of the system for a large range of number of devices (from three to nine), shows that the collision avoidance provided by the MAC layer works reliably, also for dynamic PHY modes. Nevertheless, for more than nine transmitters the total system throughput considerably decreases, since collisions and congestion become more likely. With 12 transmitting devices, a maximum throughput of approximately 900 b/s can be achieved. Having a network of devices based on different hardware cannot exhaust the theoretical limits, but nevertheless *libvlc* can provide a stable network performance.

Figure 5.18 shows results for the same testbed setup for two to twelve devices, but this time using only devices with ARM processors. They are configured to use AARF to dynamically choose the appropriate PHY mode. The communication partner for each transmission is randomly selected. The highest selectable PHY mode is `OCTA` (only ARM-based devices are present). The resulting throughput is significantly higher than with a mixed setup.

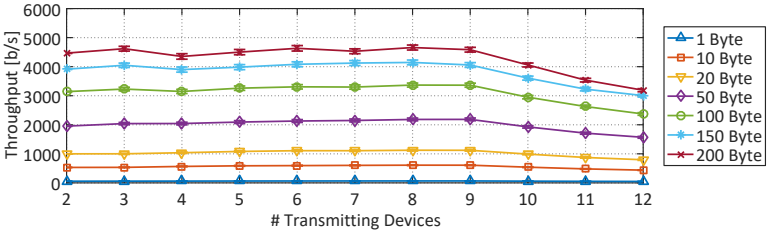


Figure 5.18: Network throughput measurement results for ARM devices. AARF is used to dynamically adapt the PHY mode. Results for different payload sizes and for two to twelve transmitting devices are shown. A throughput of approximately 3.2 kb/s can be achieved with twelve transmitting devices.

The network shows the typical behavior of a CSMA/CA protocol. Throughput is slightly increasing (when adding more devices) until saturation is reached; at this point, collisions start to be more probable and throughput decreases again for each additional device, but communication stays stable. For 200 B packets, throughput can reach up to almost 5 kb/s for nine devices participating in the network. For the maximum number of twelve devices, a four times higher throughput of approximately 3.2 kb/s is reached when compared to the results of Section 3.5.3.

5.3.3 *Light Bulb-to-LED Communication*

Since *libvlc* does not only support LED-to-LED channels, but also allows the use of LED-only devices in combination with LED light bulbs equipped with dedicated sensors as introduced in Chapter 4, such a heterogeneous setup is evaluated in the following. The throughput performance for a system where a light bulb transmits data to a device with an LED as receiver is measured. Unlike before, data is sent as broadcast messages to also present the performance for unacknowledged data transmissions. The testbed setup is illustrated in Figure 5.19. The light bulb is inserted into a table lamp and aligned with the receiving LED as shown in the figure. Both devices employ an ARM processor to enable all available PHY modes and can be moved back and forth

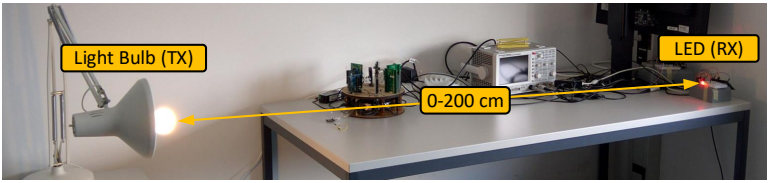


Figure 5.19: Light bulb-to-LED communication testbed setup. The light bulb is aligned with the receiving LED as shown in the picture. The lamp and the LED device is moved back and forth to cover communication distances up to 200 cm. The receiving device uses an ARM processor to be able to support all available PHY modes.

to realize communication distances between 0 to 200 cm. Experiments for various distances and payload sizes are conducted for a duration of 5 min each. All experiments are repeated for the four available PHY modes (fixed without AARF). The FEC threshold is set to 30 B and default MAC parameters are applied.

Figures 5.20 to 5.23 show the results of the measurement campaign for PHY mode `SINGLE` to `OCTA`. The y-axis denotes MAC layer throughput in b/s and the x-axis denotes communication distance in cm. The error bars indicate the standard deviation. Since the adaptive PHY mode selection is based on transmission statis-

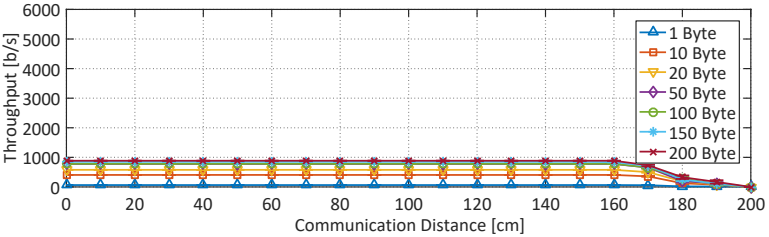


Figure 5.20: Throughput measurement results for light bulb-to-LED communication using PHY mode `SINGLE`. Results for different packet sizes and variable distances are shown. A maximum throughput of approximately 950 b/s is achievable and communication is stable up to a distance of 180 cm.

tics, it cannot be used for broadcast messages due to the lack of ACKs. It becomes evident from the plots that the maximum throughput is higher than with acknowledged messages, which is understandable, since the sender does not need to wait for an ACK before sending the next packet. The difference in throughput is most noticeable for PHY mode `OCTA`, where the LED setup with ACK achieves a maximum throughput of about 4.72 kb/s and the light bulb-to-LED setup reaches 5.46 kb/s for a 200 B payload, an increase of about 15 %. Thanks to the higher light intensity produced by the light bulb, further communication distances than

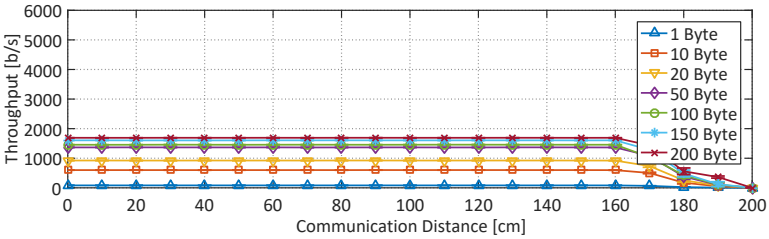


Figure 5.21: Throughput measurement results for light bulb-to-LED communication using PHY mode `DOUBLE`. Results for different packet sizes and variable distances are shown. A maximum throughput of approximately 1.7 kb/s is achievable and communication is stable up to a distance of 170 cm.

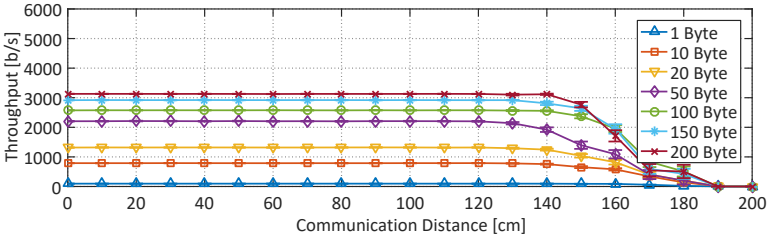


Figure 5.22: Throughput measurement results for light bulb-to-LED communication using PHY mode `QUAD`. Results for different packet sizes and variable distances are shown. A maximum throughput of approximately 3.1 kb/s is achievable and communication is stable up to a distance of 160 cm.

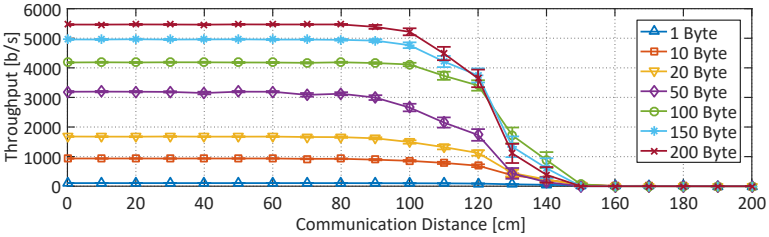


Figure 5.23: Throughput measurement results for light bulb-to-LED communication using PHY mode `OCTA`. Results for different packet sizes and variable distances are shown. A maximum throughput of approximately 5.5 kb/s is achievable and communication is stable up to a distance of 120 cm.

for the LED-only case can be reached. Yet, it is limited by the sensitivity of the LED, leading to a relatively modest improvement in the maximum distance of about 20 cm for PHY mode `SINGLE` and `DOUBLE`. Nevertheless, for PHY mode `QUAD` and `OCTA`, the maximum communication distance is almost doubled to 160 cm respectively 120 cm when compared to the LED-only case. The results demonstrate that *libvlc* can also support heterogeneous communication links.

5.3.4 *Light Bulbs*

This section presents results for measurements conducted with LED light bulbs equipped with sensors for reception as introduced in Chapter 4. A network formed by up to six light bulbs is evaluated for the four available PHY modes and AARF. Furthermore it is demonstrated that light bulb networks can also be formed using only indirect light. No direct line of sight is necessary for two light bulbs communicating with each other.

Network

Whereas Section 5.3.3 reported results for a mixed setup of LED light bulbs and LED-only devices, the content of this section describes the testbed and results for a network of light bulbs. For

this measurement campaign, six light bulbs (in floor lamps) are arranged in a star-like shape, each 2 m apart from the center), using the devices on the edges as dedicated senders and the light bulb in the center of the star as the dedicated receiver. Experiments for various payload sizes and one to five transmitting devices are each conducted for a duration of 5 min. These measurement series are repeated for all available PHY modes and the dynamic PHY mode selection based on AARF. FEC is enabled for PHY layer payloads of 30 B and larger, and default MAC parameters are used.

Figures 5.24 to 5.28 show the results for each fixed PHY mode as well as the effect of adaptive mode selection. The y-axis denotes the total network throughput, which is the sum of throughputs achieved for each sender, in b/s. The x-axis denotes the number of transmitting devices. The results for all four PHY modes show a stable throughput for the various number of senders, demonstrating a working MAC protocol. The achieved maximum throughput can be compared with the results for single link throughput for all PHY modes and is within a few percent of the theoretical maximum throughput. For PHY mode `OCTA`, a payload size of 200 B, and five devices, a throughput of more than 4.5 kb/s is reached.

The dynamic PHY mode selection, using a modified AARF, provides slightly inferior results than PHY mode `OCTA` for more than three transmitters and packet sizes above 50 B. The maximum achievable throughput is 4.65 kbit/s for three transmitting de-

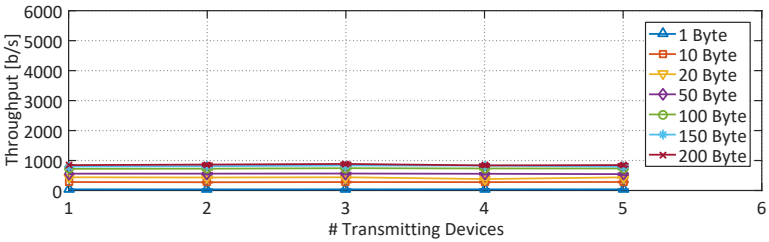


Figure 5.24: Throughput measurement results for a light bulb network using PHY mode `SINGLE`. Results for different payload sizes and one to five transmitting devices are shown. A throughput of 850 b/s is reached for five transmitting devices.

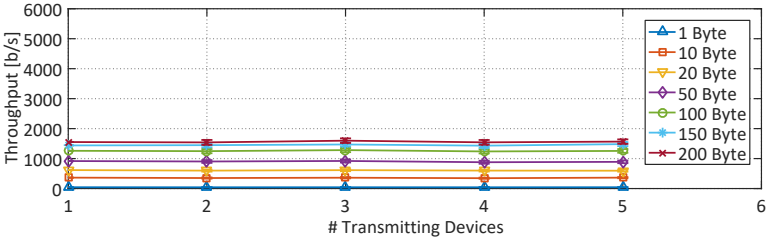


Figure 5.25: Throughput measurement results for a light bulb network using PHY mode `DOUBLE`. Results for different payload sizes and one to five transmitting devices are shown. A throughput of 1.7 kb/s is reached for five transmitting devices.

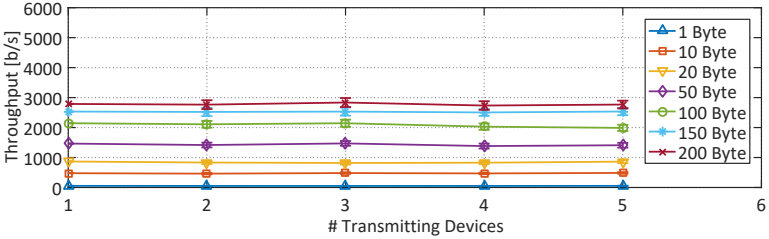


Figure 5.26: Throughput measurement results for a light bulb network using PHY mode `QUAD`. Results for different payload sizes and one to five transmitting devices are shown. A throughput of 2.8 kb/s is reached for five transmitting devices.

vices. This suggests that the parameters for the adaptive selection, which determines how the algorithm reacts to dropped or retransmitted frames, are set too conservatively in a scenario with higher contention probability. This leads to hasty scale-backs, and a slower PHY mode is selected too quickly, and the waiting time for the next probing packet is too long to quickly adapt to the recovery characteristics of a congested channel. Still, the results for the adaptive PHY mode selection are close enough to the results achieved with PHY mode `OCTA` to consider it a working prototype.

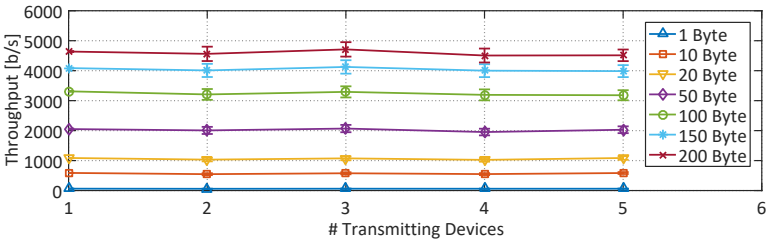


Figure 5.27: Throughput measurement results for a light bulb network using PHY mode `OCTA`. Results for different payload sizes and one to five transmitting devices are shown. A throughput of 4.6 b/s is reached for five transmitting devices.

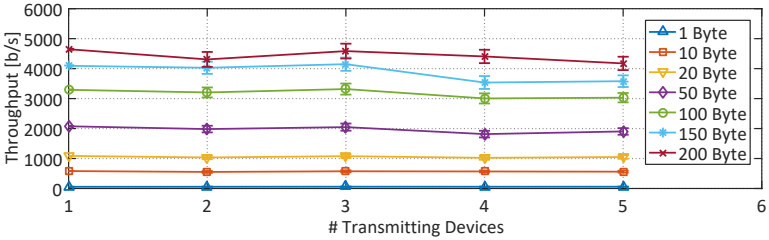


Figure 5.28: Throughput measurement results for a light bulb network using AARF to dynamically determine the optimal PHY mode. Results for different payload sizes and one to five transmitting devices are shown. A throughput of 4.2 b/s is reached for five transmitting devices.

No Line of Sight Communication

The highly sensitive light bulb sensors and the improved synchronization methods allow for line of sight communication over several meters. One of the most often heard critique points for VLC system is that only line of sight communication is possible. Figure 5.29 shows that also no line of sight communication is possible if accurate enough sensing is available. LB1 is moved from line of sight to a no line of sight location while communicating with the static LB2. The plot shows RSSI values retrieved from received data frames for various line of sight and no line of sight positions of LB1. While in line of sight, LB1 moves closer to LB2,

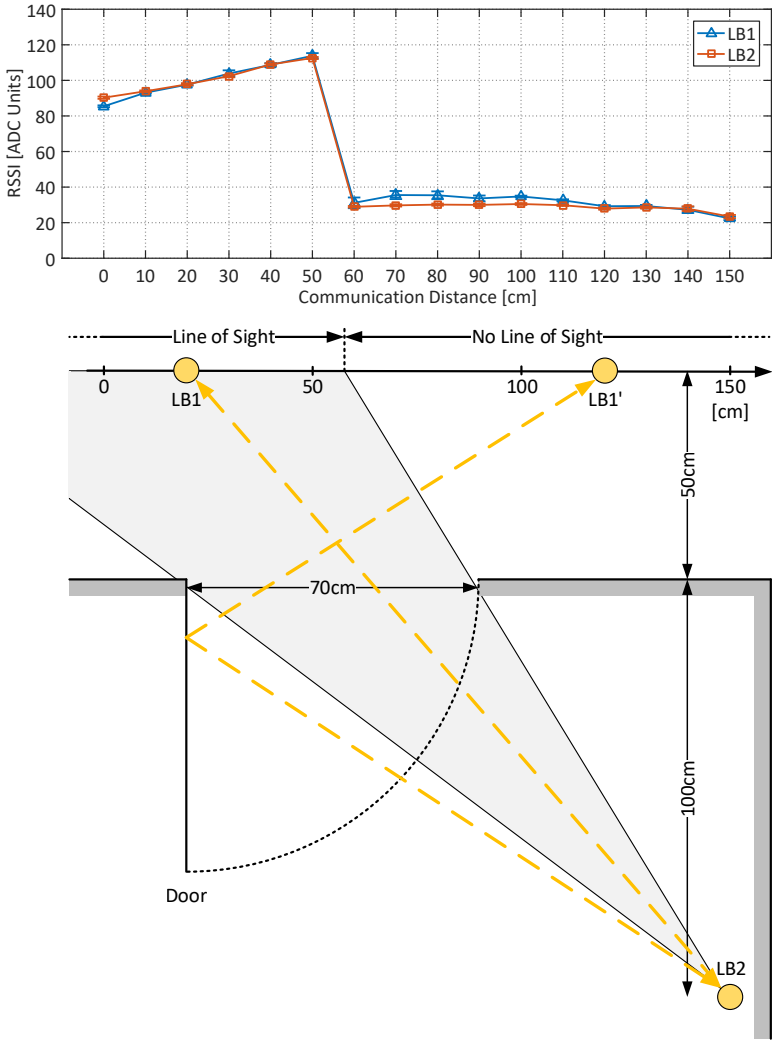


Figure 5.29: No line of sight communication via reflection on a room door. LB1 is moved from line of sight to no line of sight (from left to right). The plot shows RSSI values retrieved from received data frames in relation to the position of LB1. At the transition point (60 cm and further) from line of sight to no light of sight, communication is still possible.

and therefore the RSSI value is slightly increasing. At a distance of 60 cm, the RSSI value suddenly drops to 30. This is the point where the light bulbs start to communicate via the reflection on the door (regular wooden door, painted white). The communication also stays stable when moving LB1 further away from the door. At 150 cm, LB1 is at the same horizontal coordinate but separated by a wall and still communicates with LB2. To reach even further, the application running on top of *libvlc* can configure a PHY parameter specifying the light level difference at which bits can still be distinguished. This value is closely related to the RSSI value. Setting it to a value of 10 and above still guarantees stable communication and enables reception at an RSSI level close to 10. Reflection not only works on doors but also an walls and ceilings and evens allow communication between different floor levels.

5.4 DISCUSSION AND CONCLUSION

This chapter introduced various PHY layer modes and a dynamic PHY mode selection scheme based on AARF. To allow the higher PHY modes to work properly, an additional hardware platform based on a 32-bit ARM processor from STM is proposed. The processor provides higher performance due to the 32-bit architecture and higher clock rate, whereas the price level is similar as for classic 8-bit microcontrollers. To run *libvlc* on different hardware and to reduce the effort needed to port the communication library to another platform in the future, a HAA is integrated in the existing VLC software stack. Furthermore, improved receiving strategies increase the software-based synchronization accuracy leading to higher performance and larger communication ranges. Data throughput could be increased more than 5 times for all channel types (LED, sensor, mixed) with mostly software changes only, demonstrating the capabilities of a flexible software-based system.

A software-based VLC system can work with a wide range of sensing hardware. In scenarios that must work with minimal hardware components (e.g., because cost or energy consumption are critical parameters), a single LED can be used as a sender and receiver. If the environment allows the usage of an LED light bulb

(with a dedicated sensor for receiving), larger distances can be covered. As LED light bulbs and LED-only systems use the same PHY and MAC layer integrated into *libvlc*, they can inter-operate and form a convenient platform for indoor room-area networks.

Realistic VLC systems must work across a wide range of environmental conditions; close to a window that brings sunlight into a room, for mobile devices (maybe attached as tags to physical objects), or without direct line of sight (either because of the device placement or because a moving object or person blocks temporarily the view). The software-centric approach described allows to easily adapt the link resilience based on the strength of the input signal. Adaptivity can deal with varying distances between sender and receiver, the hardware capabilities (sensor properties and processor features), or environmental conditions. Compared to a static system, an adaptive VLC system can translate the increased channel capacity into up to 8 times higher bit rates, can communicate over significantly larger distances, or allow communication without a direct line of sight, allowing VLC to be used to communicate around a corner or between different floors of a building.

6 EXPLORATION AND APPLICATIONS

The previous chapters introduced a prototype VLC system that builds upon basic microcontrollers running a software-defined communication stack called *libvlc*. The proposed communication protocols are evaluated for different scenarios such as LED-to-LED networks or connected light bulbs. The results demonstrate that the proposed concepts are working and that they are ready to be to applied to real-world devices, extending them with VLC capabilities.

This chapter is about exploring the many application areas of VLC and demonstrating how existing devices can be used to interact with VLC networks. Sections 6.1 and 6.2 discuss how to establish a VLC link between smartphones and VLC-enabled devices, such as toys or LED light bulbs. The first approach described in Section 6.1 uses a battery-less extension device that plugs into the phone's audio jack, enabling bidirectional communication using an LED and photodiode driven by audio signals. The second approach discussed in Section 6.2 goes even further and does not rely on additional devices. It employs the integrated smartphone camera as a sensor and exploits the rolling shutter effect to enable software-based real-time decoding of VLC data streams generated by devices running *libvlc*.

Section 6.3 explores the ability of VLC to make communication visible. It describes an intuitive approach how to use a modified flash light as a remote control to configure lighting systems by simply pointing at the illumination sources. The light is employed at the same time to guide the user (to provide visual feedback) and to carry information (to send commands) from the flash light to the targeted light source. The applications addressed in this chapter showcase only a small selection of many direction that can be investigated explored to effectively apply VLC as a communication technology.

6.1 FROM SOUND TO SIGHT

This section presents the design, implementation and evaluation of a miniature low-cost passive device that can be plugged into an audio jack connection of a mobile phone to enable two-way VLC communication based on the protocol introduced in Chapter 3. The device uses an LED and a photodiode to transmit respectively receive light signals. The use of a photodiode instead of a receiving LED is discussed in Section 6.1.1. When connecting such a device with a phone's headset audio jack, the phone can exchange data through light at a data rate of 700 b/s in both directions. The miniature VLC device uses the audio output of the phone to drive an LED (TX), and the microphone input to receive from a photodiode (RX).

The audio jack device is battery-free and operates without the involvement of a microcontroller. A software running on the mobile phone is responsible to generate the audio signals to drive the LED and modulate the light. The incoming light signal sensed by the photodiode and recorded by the microphone is also directly processed on the phone. Since the application can receive and generate data frames in real-time it is able to maintain a bidirectional communication link with other devices running the *libvlc* communication stack.

This section is structured as follows. Section 6.1.1 presents the hardware design of a VLC peripheral extension device for smartphones using the audio jack as interface. The device is battery-free and only powered and operated through audio signals generated by the mobile phone. Further, the device is equipped with a photodiode to feed incoming modulated light as electrical signals into the microphone input. Sections 6.1.2 describes the smartphone application software operating the audio jack peripheral device through audio signal processing only; there is no need for an additional microcontroller. Microphone input data is analyzed and decoded in software and arbitrary data packets can be generated in real-time using the peripheral's LED as a communication front end. Section 6.1.3 discusses the evaluation of the designed

hardware together with application software running on Apple iOS[1] devices; results for different host devices are reported.

6.1.1 *Hardware Design*

This section focuses on how to extend mobile devices, such as smartphones and tablets, with VLC capabilities. Although these devices are already equipped with a flash light (light emitter) and camera (light receiver), which can be used for communication [12, 14], they do not provide enough flexibility to work well together with other VLC-enabled devices. Smartphone operating systems cannot support real-time scheduling, and control over the flash light and camera is often restricted. These constraints limit performance and stability of a VLC link. The system described in the following is based on a passive peripheral device that plugs into a smartphone's audio interface and can emit and receive light by using the phone's audio system. The peripheral device is battery-free and powered only through the audio signals, yet the communication protocols are handled without additional microcontroller or signal processors – light is directly modulated through audio signals generated in real-time by the smartphone, and incoming signals are converted by the microphone and analyzed by the smartphone's software. The system described can interact with existing VLC-enabled toys or other consumer devices that implement *libvlc's* VLC protocols.

Audio signals are AC-coupled, hence it is not possible to directly generate an on and off signal to drive an LED. Further, even with the loudest audio settings, the peak voltage values of the audio output signal is still in the millivolts range (around 100 to 200 mV, depending on the device) and therefore not large enough to emit light with reasonable intensity using a standard LED. The proposed peripheral device uses a hardware design based on the Hijack project[2] [42, 43]. The device's schematics are shown in Figure 6.1 on the left side. The schematics present a low-complex design with only a handful of components (no microcontroller

1. https://de.wikipedia.org/wiki/Apple_iOS
2. http://web.eecs.umich.edu/~prabal/projects/hijack/

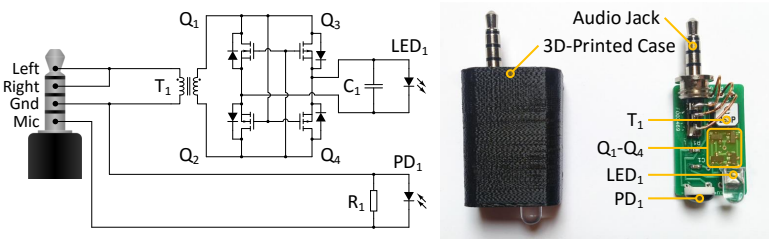


Figure 6.1: Audio jack peripheral schematics and assembled PCB. Right part: Coupled inductors (T_1) transform the audio signal so that the voltage is high enough to drive the LED (LED_1). An efficient MOSFET rectifier bridge (Q_1-Q_4) inverts the negative part of the signal to provide additional forward bias for the LED. The photo current generated by the photodiode (PD_1) is converted to a voltage using a resistor (R_1) and sampled through the microphone input. Left part: fully assembled PCB with 3D-printed case.

needed). The audio signals of the left and right (stereo) channel are joined together to increase the available current and power. The signals are transformed by the coupled inductors (T_1)[3]. These inductors are passive components that increases the voltage at the same electric power (current is decreased). The higher voltage is needed to drive the LED. Already with this raw signal, the LED emits light, albeit with low intensity. To increase the light emission further, the signal is rectified (Q_1-Q_4). This step makes it possible to also use the negative parts of the generated audio signal to emit light instead of reverse biasing the LED. The rectifier is built as an efficient MOSFET rectifier bridge to minimize voltage loss. Finally, a capacitor is used to smooth and stabilize the LED's input voltage (the same LED from Kingbright as mentioned in previous chapters is also used here).

Instead off using the LED also for reception, the device uses a photodiode to convert modulated light back to electrical signals. Using the same LED to send and receive is not trivial for this setup. There is a bias voltage of 2 V applied to the microphone input and therefore, to switch between emitting and receiving light,

3. https://octopart.com/lpr6235-253pmlc-coilcraft-11914358

the LED needs to be attached and detached from and to the microphone signal line. As the goal is to keep the hardware as simple as possible, the focus is on how to modulate the light with the help of audio signals and to allow bidirectional communication with a low-complex circuit. The peripheral device uses the microphone (and electronic circuit behind it) as an ADC to measure the voltage over a resistor (R_1). Incident light generates a photo current in the photodiode (PD_1). The current is proportional to the light intensity and can be measured as voltage drop over the resistor. Figure 6.1 on the right shows the assembled PCB with and without casing. The device is still small, fits on a board of 1.7 cm by 2.7 cm, requires no battery, and can be built with only a handful of inexpensive electronic parts.

6.1.2 *Smartphone Software*

The software part of the VLC system implemented for iOS is responsible for generating the waveforms needed to modulate the peripheral device's LED to conform with *libvlc's* PHY layer. It also analyzes the incoming signal from the photodiode and decodes it. The software consists of three principal modules: the main module, the sender module, and the receiver module. Because sender and receiver use the same interface to the hardware, they are partly implemented in a common transceiver module. Data passed from the application to the VLC main module is encapsulated in VLC frames and added to a message queue. The main module is also in charge of the MAC layer as specified by *libvlc*. From there the frames are passed to the sender in First In – First Out (FIFO) order. Incoming messages are processed by a decoding pipeline and finally delivered to the main module, which dispatches a MAC layer ACK if necessary and delivers the data to the application.

Sender

The core of the sender is a callback function that is invoked by the hardware whenever it needs to output sound buffers. Sound buffers contain the values that are output via the sound system's

DAC to generate the audio signal. To ensure that the callback function returns before a buffer underrun occurs, the templates for the three appearing patterns (idle COM slot and COM slot for bit 0 and 1) are pre-built at start-up and only need to be copied into the target buffers (this system supports only PHY mode `SINGLE` due to the limited sampling rate of 48 kHz). A template always comes with the following ILLU slot (with possible compensation) attached. The built-in sound processor of iOS devices smooths quick on and off patterns. It was determined that with a sample rate of 48 kHz and a signal frequency of 10 kHz the effect could be reduced to an acceptable level while still delivering enough power to the LED. The 10 kHz signal is transformed and rectified by the peripheral device to deliver a constant forward voltage to the connected LED.

An example of a generated audio signal based on the predefined wave forms is shown in Figure 6.2. An oscilloscope screenshot shows the smartphone's stereo audio output signal. To produce an idle pattern, the 10 kHz signal (generating an ILLU slot) and no signal (creating a COM) is alternating. The audio signals to generate light patterns for COM slots carrying bits and ILLU slots

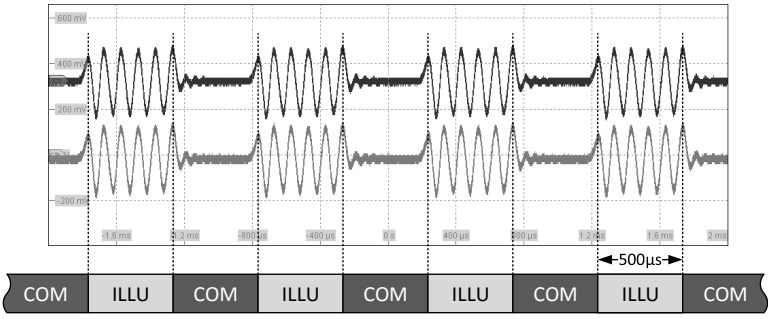


Figure 6.2: Stereo audio signal to generate alternating ILLU and COM slots. An oscilloscope screenshot is illustrating the smartphone's audio output signal. A 10 kHz signal (transformed and rectified by the peripheral device) drives the LED during an ILLU slot. An idle COM slot (no light output) is simply created by the absence of any signal.

with compensation parts can be created analogously by enabling the 10 kHz audio signal for a certain amount of time. There is no synchronization mechanism integrated in the smartphone software. It is not necessary for the receiver since the incoming signal is oversampled by the microphone part of the audio system. When transmitting, it is expected that a receiver synchronizes to the produced light signal, if necessary.

Receiver

Whenever the hardware has input audio buffers ready, containing the captured waveforms from the microphone input, a mechanism is invoked to preprocess these buffers and to copy them to user memory where they are forwarded to the decoding pipeline. The first stage of the pipeline is the physical decoding stage. It compares the audio frames to a threshold with alternating sign. A value below the negative threshold value is considered to originate from incoming light, while a value above the positive threshold means there was no light detected. A change between light and no light is called a flip. The decoder then calculates the number of samples between the flips. These run lengths correspond to the on and off pattern of the transmitting LED. By analyzing these patterns, individual bits can be decoded. The decoded bits are accumulated to bytes and passed to the next stage.

Due to a high level of noise in the signal, short intervals cannot be reliably detected with a resolution of only 24 samples per slot (48 kHz is the maximum sampling rate). Thus, a simplified decoding scheme is introduced that works as follows: if there is no flip during a COM slot, no bit is detected. If there is a flip, the decoder counts the samples and determines in which half of the COM slot the flips occurs. With this information, it can be concluded whether the light belongs to D_1 or D_2 to successfully decode a bit. Furthermore, the inability to detect short intervals prevents the system from being able to synchronize to another VLC device (as it cannot detect the synchronization intervals which are part of *libvlc's* PHY layer). As long as only one smartphone is in a network, this limitation does not pose a problem as the other VLC devices can synchronize to the smartphone's pattern. The

second pipeline checks for the SFD in the decoded bit stream and if detected, the headers are decoded and the payload is retrieved.

Signal Feedback and Filtering

To drive the LED and to follow the alternating ILLU and COM slot pattern, the audio system is continuously generating the necessary waveforms. Because of the simple circuitry included in the peripheral device, the audio output signal is leaking into the receiver (microphone), leading to two problems: First, there is a 10 kHz feedback signal during ILLU slots, making it impossible to reliably detect the end of ILLU slots. Second, while transmitting data, the generated audio signals mimicking the data and compensation intervals are also fed back into the receiver, leading to more signal decoding problems. To prevent both problems, the waveform buffers from the microphone are preprocessed before decoding. By keeping track of the COM slot sample times when generating the output waveforms, the corresponding input buffers are filtered to remove the leaked signal.

The smartphone application also includes an oscilloscope-like visualization. The microphone input signal can be paused or displayed in real-time. This visualization tool is used in Figure 6.3 to illustrates the filtering process. If the peripheral's light output is not enabled, no audio signal is leaking into the receiver, hence a clean input signal can be reported (1). The negative part of the signal denotes light input. The dashed red line shows the run lengths between signal flips used by the decoder pipeline to retrieve the encoded bits. If the light output is enabled, the software produces a 10 kHz signal to drive the LED and to follow the ILLU an COM slot pattern, also leaking into the microphone input (2). Since it is known when light output is enabled, it can be filtered by modifying the retrieved audio samples (3). The COM slot beginnings and endings are still influenced by the audio signal used to generated light output. To provide a clean signal for the decoder pipeline, those parts of the slot are masked out (4). This can be done without consequences since the signal flips used to decode the bit stream occur within the slot and not at the borders. While transmitting, the same filtering procedure is applied to the ILLU

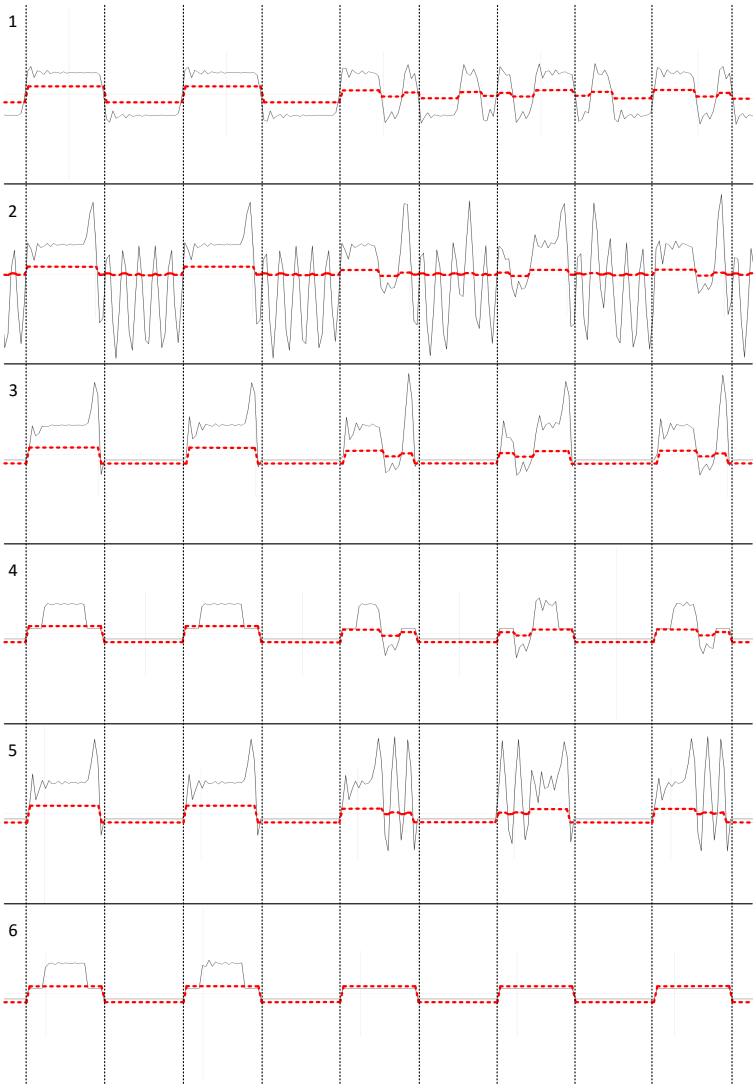


Figure 6.3: Signal filtering and masking. The dashed red line indicates the run length pattern found. 1) Input signal while LED off; 2) input signal while LED on; 3) filtered ILLU slot; 4) masked COM slot borders; 5) filtered ILLU slot while transmitting; 6) filtered and masked COM slots while transmitting.

slots (now including the compensation interval) and COM slots carrying data (6).

6.1.3 *Evaluation*

The evaluation testbed consists of an iPhone 5S and an iPad mini (both equipped with Apple's A7 processor, running iOS 7) and a VLC prototype device that employs an LED as transceiver and runs *libvlc*. The audio jack peripheral device is platform independent. Other smartphones or tablets, even laptops or desktop computers (independent of operating systems), could be used for this evaluation as long as they provide a 3-channel audio jack plug with a matching pinout. All experiments are conducted in an office space and without special shielding from artificial light or sunlight.

Acknowledgment Timeout

The VLC device and smartphone both run a MAC layer capable of data frame acknowledgments and retransmissions. After transmitting a data frame, the transmitter waits for the ACK timeout. If no ACK from the data frame's destination is received within this time period, the frame counts as lost and the transmitter retransmits the same frame again. This procedure is repeated until an ACK is received or a fixed number of retransmissions is reached. The VLC device running *libvlc* can keep ACK timeouts short since the frame processing only lasts a few milliseconds. With a short ACK timeout, the communication channel can be used more efficiently guaranteeing higher data throughput.

A smartphone operating system is not a system with real-time guarantees and the main processor is used for several different tasks at the same time. Also, it may take some time to analyze incoming data from the peripheral device and decode the content. Furthermore, the audio signals needed to transmit an ACK are generated on demand specifically for the received data frame; this step takes additional time. Hence, *libvlc's* ACK timeout must be adjusted to enable a successful and optimized data exchange with a smartphone using the audio jack extension device.

To find a proper value for the ACK timeout the following experiment is conducted: the VLC device is generating data packets (saturation) with the smartphone as destination. The smartphone needs to acknowledge this data. If the ACK does not arrive on time, the data frame is retransmitted, resulting in throughput drop. To find an optimal value, the timeout is increased step by step. The same experiment is also repeated for different packet sizes so see if the processing time (on the smartphone) has any impact on the delay. The results for the iPhone 5S are shown in Figure 6.4 (results for the iPad mini are very similar). The y-axis denotes throughput in b/s and the x-axis denotes the chosen ACK timeout in ms. The error bars show the standard deviation. For timeouts of 125 to 130 ms throughput is stable but not close to the theoretically reachable maximum, meaning that the ACK arrives too late and packets are always retransmitted (and duplicates dropped at the receiver). Between 130 ms and 150 ms throughput is increasing slowly, but the plot also shows increased

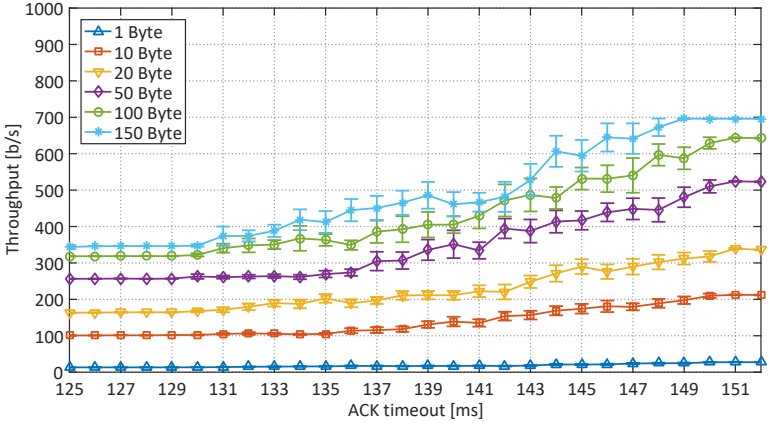


Figure 6.4: Throughput measurement results for various ACK timeouts and payload sizes for an iPhone 5S. For a timeout of up to 150 ms, the ACK might arrive too late and the corresponding data frame is retransmitted reducing throughput. From 151 ms, the ACK arrives in time, triggering no retransmission and therefore maximizing data throughput.

error bars, leading to the conclusion that the ACK reaches the destination sometimes within the timeout window. For 151 ms and longer, the error bars are disappearing again and throughput stays stable. Measurements for higher timeouts are omitted since the throughput does not increase anymore. Also, the packet length and therefore the decoding time on the smartphone seems not to have any impact. In summary, the measurements show that a delay of around 150 ms (*libvlc's* default is 84 ms) is optimal to maximize throughput for a single communication link. As these experiments were conducted with older hardware, this value might be lower (and closer to the default value) with current smartphones.

Distance Measurements

To be useful for some use cases, the peripheral device must be able to cover a certain communication distance and achieve a stable and reasonable data throughput. Figures 6.5 and 6.6 show

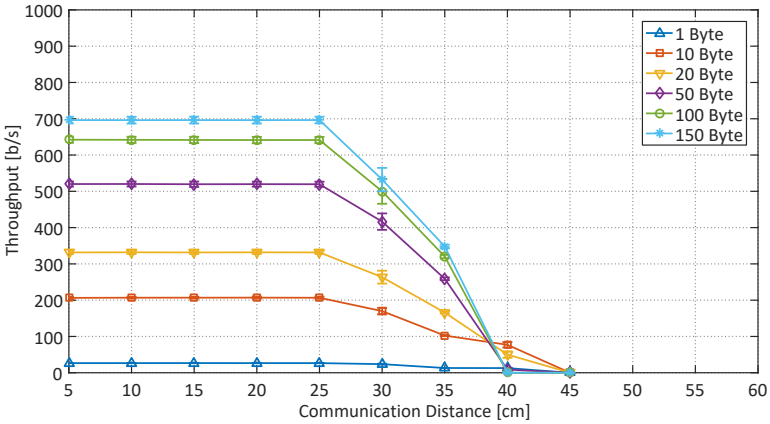


Figure 6.5: Throughput measurement results (without FEC) for various distances and payload sizes and for a communication channel between an iPhone 5S, using the introduced audio jack peripheral, and a VLC device based on *libvlc*. A maximum throughput of 700 b/s for a payload size of 150 B is achieved. The communication link stays stable up to a distance 35 cm.

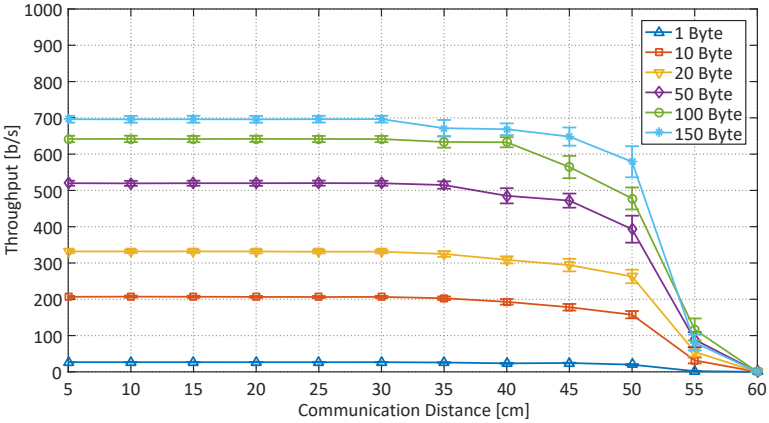


Figure 6.6: Throughput measurement results (without FEC) for various distances and payload sizes and for a communication channel between an iPad mini, using the introduced audio jack peripheral, and a VLC device based on *libvlc*. A maximum throughput of 700 b/s for a payload size of 150 B is achieved. The communication link stays stable up to a distance 50 cm.

measurements for an iPhone respectively an iPad and various payload sizes. The y-axis denotes data throughput in b/s and the x-axis denotes communication distance in cm. The error bars show the standard deviation. The VLC device running the *libvlc* firmware acts as data frame generator (saturation). The smartphone or tablet receives the data frame and sends back acknowledgments. A transmitted data frame is only accounted for when also the corresponding ACK is received. The measurement results for the iPhone show that the throughput stays stable up to 25 cm at a maximum throughput of 700 b/s for a payload size of 150 B. With the retransmission scheme in place, it is also possible to achieve reliable communication up to a distance of 35 cm, but with losses in throughput. The iPad measurement results show an increased communication range. This is due to the more powerful audio amplifier included in the iPad which increases the intensity of the light emissions. The iPad achieves stable throughput in the same order as the iPhone at a communication dis-

tance of up to 50 cm. These measurements show that the mobile device's audio system has also an impact on the possible communication range. In conclusion, for a communication channel from the mobile device to a VLC device, 50 cm is a reasonable distance to remotely interact with other devices such as toys or other smartphones and tablets. When receiving broadcast data (e.g., from a light bulb), where no back channel is required, distances of several meters are possible.

Power Consumption

Highest light emissions are achieved by using the loudest audio output settings. This setup puts additional stress on the device's battery. Also, the computational power consumed to decode and create data packets cannot be neglected (although checking the system monitor during measurements always shows a processor utilization below 10 %). Figure 6.7 shows the battery level for both iPhone and iPad over time while transmitting and receiving. The y-axis denotes the battery level as a percentage and the x-axis denotes the time since the last full charge. The measurements show that the battery lifetime equals to more than four hours for the iPhone, and more than five hours for the iPad (due

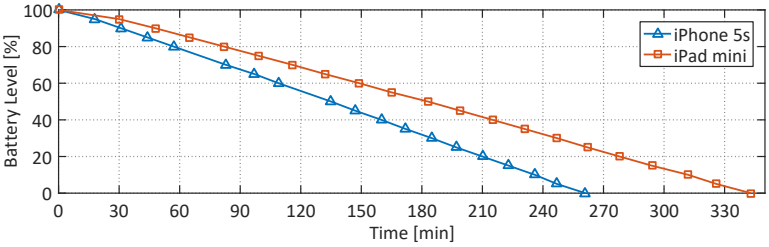


Figure 6.7: Mobile device battery lifetime measurement results for an audio jack peripheral in operation. Results for an iPhone and iPad are shown. For continuous operation, the battery of the mobile device lasts more than four hours, respectively, five hours. A general use case assumes only sporadic use of the peripheral device which is not affecting battery lifetime significantly.

to higher battery capacity). When assuming that the audio jack peripheral device is not used more than 10 % of the overall usage time per battery charge, it can be concluded that the peripheral device does not impact battery lifetime significantly.

6.1.4 *Conclusion*

Section 6.1 reports on the design, implementation and evaluation of a smartphone VLC extension device. It uses a smartphone (or tablet) audio jack as interface and is operated by audio signal processing. The key design constraints are low-complexity, low-cost, battery-free operation, and interoperability with the existing VLC systems based on *libvlc*. A simple and passive plug-in device is presented based on only a handful of electronic components powered by audio signals. Its LED is modulated without the help of an additional microcontroller, directly via audio signals generated in real-time by an application running on the mobile device. The evaluation results demonstrate the VLC communication protocols implemented in software on the smartphone or tablet provide stable and reliable communication for distances up to 50 cm, depending on the device used (and its audio system). These results show that smartphones and tablets can also be integrated into existing VLC networks by the addition of only a small passive component.

6.2 FROM BARS TO BITS

VLC systems based on *libvlc* support networking a wide range of devices with a single protocol. Simple stations (such as toys or other consumer devices) might have only a single, simple LED that is used for both, transmitting and receiving data [81, 82, 85]. LED light bulbs enhanced with photo detectors (and including a SoC that runs an embedded version of Linux) may serve a room-area network [79, 83, 84]. LED flash lights allow a user to point to another station and to transmit data (e.g., a command) embedded in the light beam. A system like *EnLighting* can support various

services, either on top of the well-known IP suite or directly using the underlying VLC protocol.

Smartphones are ubiquitous and could be an attractive platform to host various services based on VLC. E.g., a VLC system could provide positioning information or could distribute access keys. Unfortunately, current smartphone models do not have (accessible) dedicated sensors to handle VLC links. The previous chapter proposed the addition of an external device to enable bidirectional communication. This section explores the use of a smartphone's camera (without the help of an additional device) to allow data reception based on existing communication protocols employed by *libvlc*. Those cameras include a CMOS sensor array that provides the capability to sense light, usually used to take photographs or to record videos. Synchronizing the smartphone's sensing efforts with the VLC protocols is a challenge for an on-board camera, as the camera is driven by the operating system and therefore cannot be accessed directly for fine granularity control. Additionally, recent smartphone cameras support a capture rate up to 240 frames per second, which is a sampling rate too low for many VLC systems. Fortunately, such cameras suffer from the rolling shutter effect, where the CMOS sensor is read out line by line, representing a blinking light source as a set of bright and dark bars [14, 21, 48]. Although a rolling shutter might create problems when photographing fast moving objects, its side effects can provide the input for a suitable software solution to reconstruct the blinking light source. The frame capturing rate can be multiplied by the number of lines available per frame, providing a high enough sampling rate to reconstruct VLC signals.

This section describes the operation of simple, consumer-grade shutters in relation to *libvlc's* PHY layer (Section 6.2.1), followed by the description of the design and implementation of a robust decoder based on the rolling shutter effect (Section 6.2.2). Section 6.2.3 reports an evaluation, demonstrating possible communication distances and achievable data throughput.

6.2.1 *Rolling Shutter*

Smartphones usually employ a commodity camera with a simple CMOS sensor array. Many of these cameras use a rolling shutter: The columns (or rows) of the sensor array are read one at a time and not all at once (as would be the case for a global shutter). As a result, a frame captured with a rolling shutter relates to multiple points in time, because each line of the image sensor is read out one after the other. This effect can be exploited to increase the sampling rate of the camera. An LED modulated with an OOK scheme and captured with a rolling shutter camera generates a barcode-like pattern for each video frame. The width of the generated bars directly relate to the on and off times of the LED and can therefore be used to extract encoded data. Figures 6.8 and 6.9 show the effect when the camera is directed at a white wall that is illuminated by a VLC transmitter following the *libvlc's* PHY layer protocol (for PHY mode `SINGLE` and PHY mode `DOUBLE`). The COM and ILLU slots as well as the data and compensating (sub)intervals are visible and distinguishable just by looking at the captured frame.

When capturing a frame, light intensities are measured during a predefined time interval, which is known as the exposure time. By choosing a shorter exposure time and therefore reducing the influence of ambient light, the edges between bright and dark pixel bars are sharpened. The camera system's frame rate gives

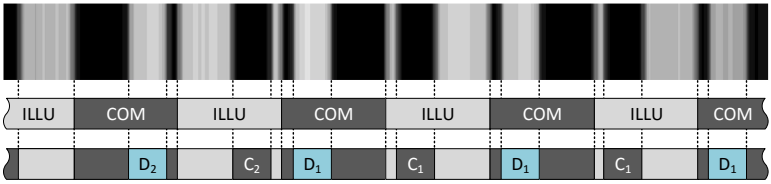


Figure 6.8: Alternating bright and dark bars caused by the rolling shutter effect when recording PHY mode `SINGLE`. The captured light source follows *libvlc's* PHY layer as described earlier. COM slots, ILLU slots, data, and compensation intervals are clearly visible.

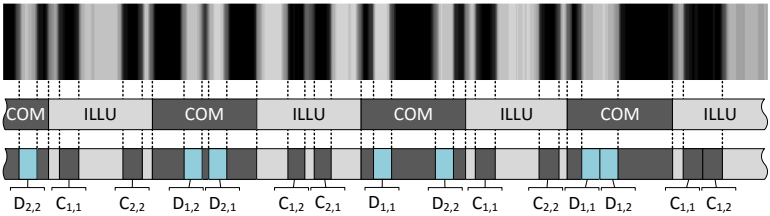


Figure 6.9: Alternating bright and dark bars caused by the rolling shutter effect when recording PHY mode `DOUBLE`. The captured light source follows *libvlc's* PHY layer as described earlier. COM slots, ILLU slots, data, and compensation subintervals are clearly visible. The subdivided data intervals lead to narrower bars.

an upper bound on how much time is required to read out all lines of a frame, since the last line has to be reached before starting a new frame. In addition, measurements on an iPhone 6S and iPhone 5 show that there is a gap between reading out the last line of a frame and reading out the first line of the following frame[4]. In other words, the shutter has the (undesirable) property to appear to pause after reading the matrix of photo detectors that make up the camera's CMOS sensor. During this time, no measurements are kept, i.e., no line is captured. This idle time must be considered in the design of an appropriate protocol, because any signal that is transmitted during this gap is lost.

To empirically assess the length of this gap, an iPhone 6S captures a scene in slow motion, i.e. at 240 frames per second, with an exposure time of 6 µs (and an ISO setting of 736). A light source is turned on every second for 2083 µs, which corresponds to a bright bar with a width of approximately 50 % of a frame. Depending on when a frame is captured, the bar can appear fully within one frame (Figure 6.10, plot 1) or span over two frames

4. There exists a wide variety of sensor arrays and camera designs built into many different smartphones [48]. This exploration focuses on iPhone devices as those are readily available, provide high video capturing frame rates, represent a large segment of the market (with a slow motion capable camera system), and are subject to common specifications. Results for Android devices are expected to be similar.

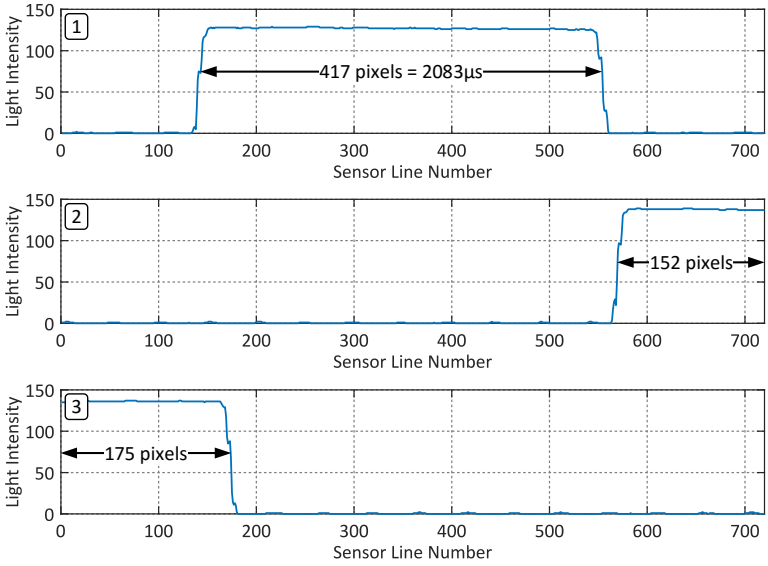


Figure 6.10: Averaged line intensities for light pulses of a fixed duration. Plot 1 shows a pulse completely visible within a single frame resulting to a width of 417 pixels. Plot 2 and 3 show the same light pulse, but split over to consecutive frames resulting in a total width of only 327 pixels. The pixel difference can be used to calculate the gap duration between two captured frames.

(Figure 6.10, plot 2 and 3). The plots show the averaged light intensities of each line of a captured frame. It is obvious that the width of the two partial bright bars is not equal to the width of the fully visible bar. The width difference can now be calculated in pixels and converted to a time value. It is concluded that there is a gap of about 440 µs (with few microseconds of variation) between consecutive frames when no signal can be received. These measurements were conducted while the phone was mostly idle and no other applications were running.

As a consequence, two challenges can be formulated when using a commodity camera built into a smartphone as an input device for VLC: First, it is impossible to synchronize camera sen-

sor readings with incoming light signals since the camera is controlled by the operating system, and second, the rolling shutter is not continuous, there is a gap between reading the last line of a frame and the first line of the following frame.

6.2.2 *Decoder*

The video frame decoder is an iOS application that first configures the camera to capture frame buffers at 240 frames per second with a resolution of 720 lines per frame, maximal ISO value (736), and an exposure time of 6 to 60 µs, depending on the light source's intensity. While capturing, the application receives for each video frame a callback with a pixel buffer in the Y-Cb-Cr[5] biplanar color space, where only the Y-plane is of any interest, because the decoder is only interested in light intensities and not in the pixels' actual color values.

Instead of working with the intensities of an entire frame, the obtained matrix is reduced to an array that represents a row in the original frame (reducing each captured line to one pixel). The array is computed by averaging every eighth pixel's intensity (of a given line). Considering multiple pixels in a line helps reducing noise, which exists due to the high sensor sensitivity, i.e. the high ISO value. Only every eighth pixel is taken into account to reduce processing time, as accessing every pixel in the frame would already exceed the time budget defined by the frame rate.

The averaged intensities (720 values) are pushed to a different thread preventing the operating system from skipping frames in case the capturing callback takes too long. As a first step, the worker thread thresholds the intensities based on their mean. Hence, finding gradients to identify bright and dark bars is afterwards a trivial task by iterating over the thresholded input array and comparing each pair of neighboring entries. Whenever the value of an entry changes, a negative respectively a positive edge is found and its relative position in the captured frame is maintained in corresponding lists.

5. https://en.wikipedia.org/wiki/YCbCr

To start decoding a transmitted bit pattern, the COM slots carrying the information need to be identified. For each frame, the starting points of COM slots can be obtained by checking for each negative gradient the following three cases:

1. The next gradient is positive and occurs at a distance of about 100 pixels, which corresponds to roughly 500 µs.
2. There is a positive gradient at a distance of about 10 pixels and a negative gradient is about 50 pixels away.
3. The following positive gradient is at a distance of about 50 pixels, and the following negative gradient is approximately 90 pixels away.

This list can be extended analogously to cover all cases (subdivided data intervals) when handling PHY mode `DOUBLE`. Case 1 handles an idle COM slot, which is completely dark, and therefore the next strong gradient belongs to the transition to the following ILLU slot. Cases 2 and 3 test for the existence of strong gradients at the start and at the end of a D_1 or D_2 data interval. To enhance stability, the decoder incorporates a fail-safe procedure to test whether additional fully visible COM slots can be inserted at the start, in between, or at the end of an already detected COM slot.

Given the start of a fully visible COM slot, decoding the transmitted symbols is now straight forward. The average intensity of the D_1 interval is compared against the intensity of the D_2 interval. If the computed difference is large enough (above a certain threshold), the corresponding COM slot is decoded to a bit 0 or 1. In case the difference is too small, the COM slot is considered to be idle. This threshold is again evaluated on a per-frame basis and is the mean of the minimal and maximal intensity of the frame. In case the decoder has received the PHY mode `DOUBLE` flag in the PHY header, the average intensity of the $D_{1,1}$ and $D_{1,2}$ interval are checked against the intensity of $D_{2,1}$ respectively $D_{2,2}$.

As discussed earlier, there is a gap between two frames where no light can be received. The decoder must ensure that no symbol is lost during this gap between two consecutive frames. The PHY layer protocol implemented by *libvlc* is slightly adapted to

add additional redundancy during ILLU slots to be able to bridge this gap. Instead of compensating the additional light output of a COM slot in a single block during the following ILLU slot, it is compensating in smaller intervals, inversely mirroring the pulses as they occur in the COM slot. The ILLU slot contains now the same information (inversed) as the preceding COM slot, in addition to still compensating for the light output to avoid flickering. Since the ILLU slot is ignored by other VLC devices, the protocol is still compatible with the original version. Although a gap of about 440 ms was concluded, a larger loss of 500 ms is assumed, simplifying the following explanation to exploit the redundancy of ILLU slots.

Figure 6.11 shows a COM and ILLU slot that cannot be completely received due to a gap of 500 ms (between two consecutive frames). Depending on the location of the gap, one of the following situations sets in:

- In the simplest case, the gap coincides with the ILLU slot, causing the preceding COM slot to be fully visible. Thus, the COM slot can be decoded as every other fully visible COM slot.
- The COM slot is completely lost, inducing the successive ILLU slot being fully visible. Since the ILLU slot is the inverse of the associated COM slot, the ILLU slot can be handled like a COM slot after inverting its intensities.
- In case the gap starts during a COM slot, not only parts of the COM slots, but also parts of the ILLU slot are affected. Fortunately, the lost COM parts exist in inverted form in the ILLU slot and vice versa, because C_1 and C_2 begin at the same relative offset as D_1 respectively D_2. Therefore by combining COM and ILLU slot, there is enough information for reconstruction.
- The last case concerns the gap beginning during an ILLU slot. The antecedent COM slot is fully visible and can thus be normally processed, but some parts of the next COM slot are not received. These parts can be restored, using the corresponding ILLU slot.

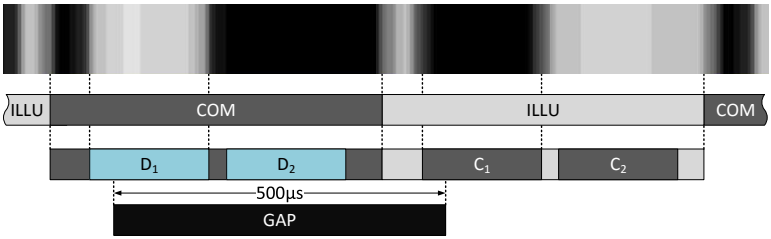


Figure 6.11: A gap of up to 500 ms can start at an arbitrary location during the transmission of a data packet in PHY mode SINGLE. 75 % and 100 % of the D_1 respectively D_2 interval are lost in this case. However, the encoded bit can be reconstructed by exploiting the redundancy in the 75 % visible respectively completely visible compensation interval C_1 and C_2.

In practice, the gap is smaller than 500 ms (as shown for an iPhone 6S), providing the decoder even more information for reconstruction. The above cases are simplified in the implementation by storing the average intensities for every D_1 and D_2 interval occurring in a COM slot close to the end of a frame that is visible for at least 50 %. The stored information is then combined with the intensities of the compensation interval of the first ILLU slot in the next video frame if the compensation interval is also

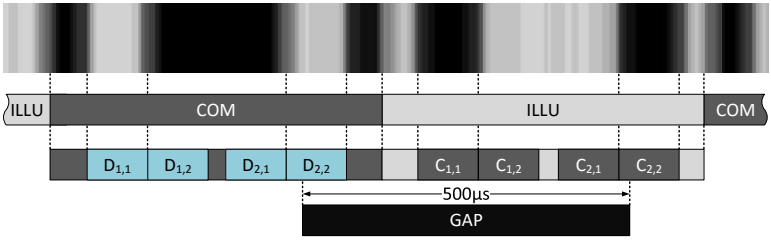


Figure 6.12: A gap of up to 500 ms can start at an arbitrary location during during the transmission of a data packet in PHY mode DOUBLE. Although 75 % of $D_{2,2}$ are lost in this case, the encoded two bits can be reconstructed by exploiting the redundancy in the visible 75 % of the compensation subinterval $C_{2,2}$ in the subsequent ILLU slot.

visible for more than 50 %. Considering only intervals visible for at least 50 % guarantees stable averages, since at least 18 (`SINGLE`) respectively 9 pixels (`DOUBLE`) are used for averaging. In addition, this restriction has no impact on decoding, because, independent of the gap's location, there is an interval whose majority is visible for each pair of data and compensation interval (as illustrated in Figure 6.11).

The reconstruction of partially visible slots is achieved similarly for PHY mode `DOUBLE`. The only difference is that more information has to be stored when changing video frame, because there are two additional data subintervals which might be needed for the next captured frame. Figure 6.12 shows a case for which the average intensities of the first three data subintervals have to be stored to be combined with the inverted intensity of the partly visible $C_{2,2}$ slot in the next frame.

The application starts collecting the decoded bits as soon as the SFD is detected and the PHY header's CRC turns out to be correct. The PHY payload's soundness is verified via the method provided by the transmitter and indicated in the PHY header's flags field. Either FCS is used to verify whether the payload is correct or FEC is employed to find possible errors and correct them. A correctly received PHY payload is forwarded to the MAC protocol layer where additional headers are removed and the resulting payload can be retrieved.

6.2.3 *Evaluation*

The concepts described in Section 6.2.2 are used to implement a software-based real-time decoder as an iOS application running on an iPhone 6S. The employed light source is a commercially available LED strip (63 LEDs, 24 V, 1500 mA from Luminary Design GmbH[6]). The experiments are conducted in a darkened room with an ambient light value measured as 9 lx (there is no need for a dark room for the software to work, but controlling the room lighting keeps it constant for the duration of the measurements). The light source is mounted vertically at various dis-

6. http://luminarydesign.ch

tances pointing to a white wall. The iPhone 6S is arranged next to the light source, with the same distance to the wall and captures the illuminated part of the wall (Figure 6.13). The LED strip is connected to a microcontroller running the *libvlc* firmware to generate VLC traffic for different payload sizes. For each distance and each payload size, at least 200 data packets are sent. For PHY layer payload sizes of up to and including 50 B, an FCS is used, and for larger payloads, FEC provides additional redundancy to find and correct possible errors.

Figure 6.14 shows the throughput for PHY mode `SINGLE` for different payload sizes. The y-axis denotes throughput in b/s and the x-axis denotes the distance between smartphone / light source and the reflecting wall. The error bars show the standard deviation. A maximum throughput of 750 b/s can be achieved with a payload size of 100 B. Since the operating system behavior cannot be controlled completely, it might happen that the gap between two captured frames is increased because the smartphone is using its processing power somewhere else. Therefore, for larger packets, the probability of losing bits in between frames is so high, that the average throughput cannot be increased anymore.

As mentioned in Section 6.2.2, PHY mode `DOUBLE` is also supported by the decoder and can reach a maximum throughput

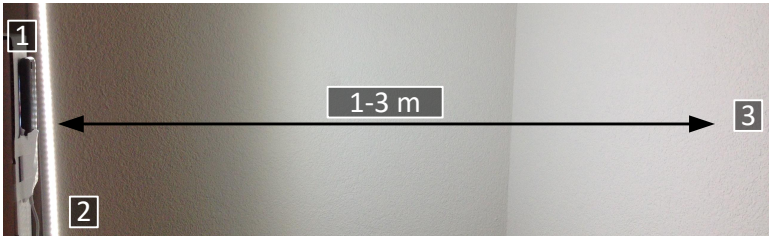


Figure 6.13: Testbed setup with an iPhone 6S (1) capturing the illuminated part of the wall (3). The light source (2), is placed next to the smartphone pointing to the same wall area (at the same distance to the wall). The distance between light source / smartphone and wall is modified to achieve different communication ranges.

of 1250 b/s as shown in Figure 6.15. The communication link is stable for distances up to 2.75 m, as demonstrated by the results. The sudden loss of throughput at about 2.9 m is due to less light penetrating the iPhone's camera, which in turn leads to smaller brightness differences between dark and bright bars in the captured frames. Although the comparison of D_1 and D_2 intervals is stable and can still be done for these light conditions, thresholding the input array leads to a noisy results where slots and intervals cannot be recovered. The standard deviation increases with the packet size due to the longer transmission duration and the unswayable influence of the operation system (length of the gap between frames) being accountable for video frame and hence packet loss.

6.2.4 *Conclusion*

This section describes a simple approach how to integrate smartphones with commodity cameras into a VLC network based on

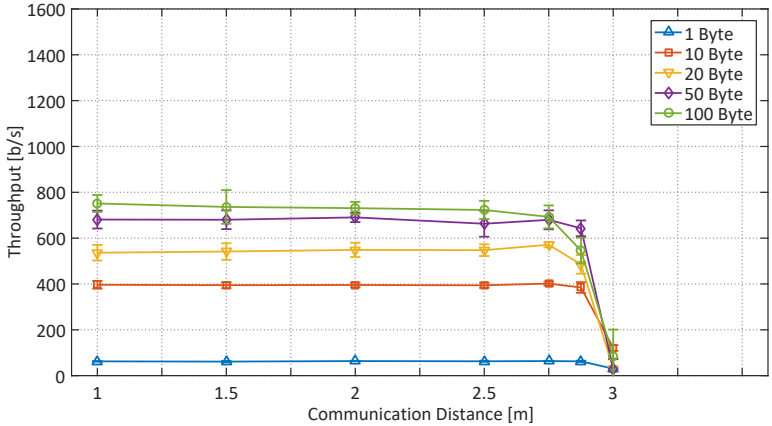


Figure 6.14: Throughput for LED to smartphone communication using different payload sizes, variable distances and PHY mode `SINGLE`. The distance is measured from the light source (and smartphone) to the wall. A maximum data throughput of 750 b/s can be achieved with a payload size of 100 B.

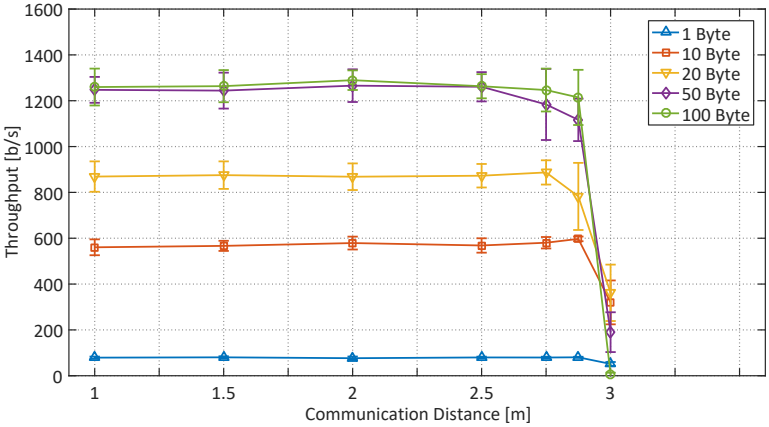


Figure 6.15: Throughput for LED to smartphone communication using different payload sizes, variable distances and PHY mode `DOUBLE`. The distance is measured from the light source (and smartphone) to the wall. A maximum data throughput of 1250 b/s can be achieved with a payload size of 100 B.

libvlc. The smartphone uses an application to continuously process captured video frames, decoding a data stream made visible through the rolling shutter effect. The protocol used for communication between LEDs or light bulbs can directly be understood by the smartphone's camera without any customizations, providing a unified communication fabric for different devices with different hardware and light sensors. The presented approach works even if the VLC protocol hides communication and creates (for a human observer) the impression that the light source is always on. Avoiding flickering is important when using VLC in environments occupied by humans.

The ability to directly and continuously receive data transmitted by other devices in a VLC system, e.g., on a smartphone, opens many opportunities for new services and usage models. Being able to directly employ the smartphone's camera as an additional communication front end through software only enables flexible and rapid prototyping of new applications for location-based services, indoor localization, and the IoT.

6.3 INTUITIVE LIGHTING CONTROL WITH LIGHTTOUCH

Light sources based on VLC technology bring new communication features to consumer devices such as smartphones and toys, introducing new play patterns and experiences [85]. The visibility and directionality of light paired with the communication ability can be exploited to create a new type of user interface. The directionality of light makes it easy to point towards possible interaction points (objects equipped with light receivers - LEDs or photodiodes) and the generated light beam acts as visible feedback to determine devices in reach.

This section introduces *LightTouch*, a lighting control user interface based on VLC. A prototype based on an off-the-shelf LED flashlight (modified to host a microcontroller, running *libvlc*) and VLC-enabled light bulbs (based on the *EnLighting* system), demonstrates a proof-of-concept system for VLC-based user interfaces. The light bulbs used as interaction end points are only one of many possible applications and use cases. A user study compares *LightTouch* with a conventional switch-based interface implemented on a touch screen. The experiments and results are described and followed by a discussion.

6.3.1 *System Description*

Today, many flashlights are already equipped with LED-based light sources since they are more energy-efficient and still can produce a high light intensity. Hence, the use of an LED can increase battery lifetime significantly. Furthermore, LEDs also have the property that they can be switched on an off at high frequencies, making them good candidates for VLC.

Usually, a large flashlight is powered by two or three D-type batteries. Figure 6.16 shows a custom 3D-printed case (b) of the same size as two batteries (a). The case provides metal contacts in the front (d) and at the back (c) to mimic one large battery, still fitting inside the flashlight's battery compartment. Inside the 3D-printed case, there is space for a microcontroller (e) running *libvlc*, a MOSFET (f) to switch high currents, and a Lithium Polymer (LiPo) battery (g) to provide power for the electronics and LED.

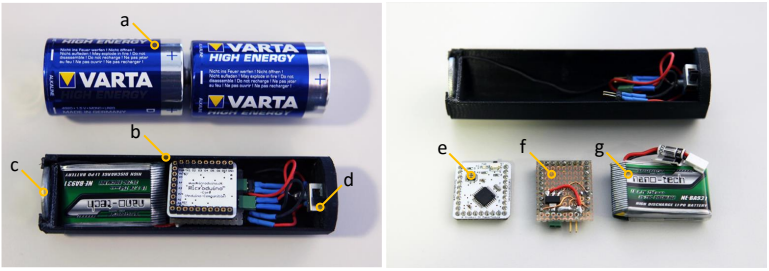


Figure 6.16: 3D-printed case for the flashlight battery compartment. The flashlight is usually operated by two D-type batteries (a) and now replace by a 3D-printed hollow case (b). The printed case uses a washer (c) as negative pole and a screw as positive pole (d). A microcontroller (e) runs *libvlc* and is connected to an additional prototyping board with a MOSFET (f). Furthermore, a LiPo battery (g) provides power for the LED and electronics.

A GPIO pin from the microcontroller is triggering the MOSFETs to switch on and off the LED, following *libvlc's* PHY layer protocol. Providing a hight signal to the MOSFET's gate connects the LiPo's negative pole to the negative pole of the battery case closing the circuit and providing current to the flashlight's LED. Since the 3D-printed case has exactly the size of two D-type batteries, it can be used with every two-battery LED flashlight, transforming it into a VLC device. Different cases can be printed without effort to also provide fitting compartments for other LED-based flashlights.

The flashlight (when switched on) continuously transmits a predefined 1 B pattern, called beacon. A beacon is just a plain byte without any additional headers and implemented in *libvlc* as follows: Beacons are predefined and need to be registered by the receiving device's application via *libvlc's* API. The PHY layer compares every completed byte buffer with the list of registered beacon patterns (similar as the SFD is detected). If a beacon is recognized, a callback is initiated, informing the application running on top of *libvlc* about the match. The VLC-enabled light bulbs listen for incoming beacons, and if the predefined pattern is received they are switched on or off depending on their previous

state. The flashlight can now toggle the light bulbs it points to, acting as an intuitive light switch. Since this one byte pattern can also be changed (dynamically), the flashlight could also trigger different actions in the light bulbs, e.g., changing light intensity or color.

The prototype application described above is only one of many use cases. When the hardware modifications are in place, only the software running on the microcontrollers must be updated to implement new functionality.

6.3.2 *User Study*

The experiment is structured into two tasks: *The One Lamp* task and the *The Light Configuration* task. The tasks are explained in the following sections. After completing the two tasks, the participants were asked five questions, described in the questionnaire section.

Experiment Setup

The experiment setup is shown in Figure 6.17. Three desk lamps equipped with VLC-enabled light bulbs are placed on a table and podium in a room. A podium with a touchscreen-enabled computer is setup 2 m in front of the lamps. The participants stand in front of the touch panel. The touchscreen shows a graphical user interface with three switches labeled from one to three. The desk lamps are also labeled from one to three, each at a different position. The three switches on the touchscreen can be used to switch on and off the corresponding lamp. The numbering is chosen so that no natural mapping for the switches (e.g., lamps left to right correspond to switches one to three) exists. The computer is connected to power sockets, which can be enabled and disabled through Ethernet. A modified VLC-enabled flashlight as described in the last section is handed (switched off) to a participant when necessary. The tasks are explained immediately before the experiment. Participants did not know the nature or purpose of the experiment and did not have any prior knowledge of the functionality of the flashlight. The time to complete the two tasks

when using the touchscreen computer interface or the flashlight interface is measured. Task 1 and task 2 are executed immediately after each other.

Task 1 - One Lamp

Initially, all three lamps are switched off. The participant stands in front of the touch screen. The three lamps are introduced and it is explained that the switches on the touchscreen can be used to toggle the lamps. One random lamp is now chosen and the participant is asked to enable it using the switches on the touch screen. The lamps are identified verbally as left, middle, or right. The time is measured until the selected (and only the selected) lamp is switched on.

For the start of the second part of task 1, the lamps are switched off again. The flashlight is handed (switched off) to the participant and she or he is told to use the flashlight (without explain-

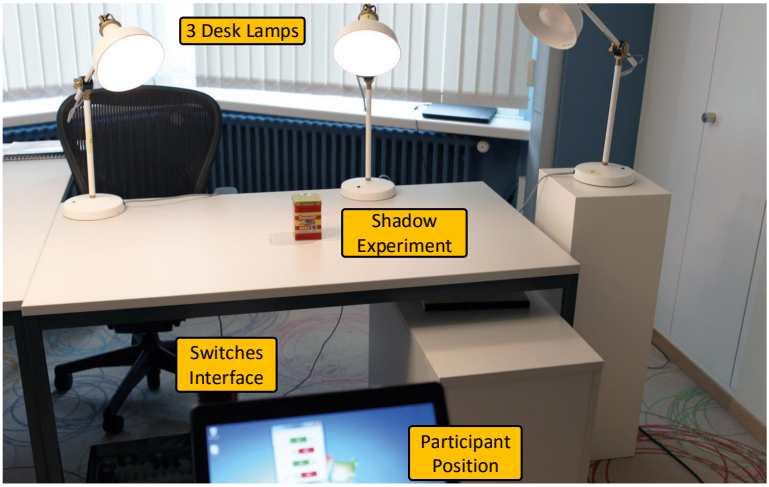


Figure 6.17: User study experiment setup. Three desk lamps equipped with VLC-enabled light bulbs, a podium with a computer on top, providing a touchscreen interface at 2 m distance, and the shadow casting box (for the second task).

ing anything else) to switch on another randomly chosen lamp. The time is again measured until the task is completed.

Task 2 - Light Configuration

For the second task, an object is placed on the table in range of the three lamps (see Figure 6.17). A shadow configuration is drawn on the table. This shadow is cast by the object under a certain lighting condition. As example, if only the left and the middle lamps are switched on, the shadow cast on the desk is different from the shadow created by the light of two other lamps. The participant is asked to find the lighting setup that matches the shadow shape drawn on the table. The lamps are once operated by the switches and once by the flashlight. Again, the time it takes to complete each part is measured.

Questionnaire

Immediately after the two tasks, every participant is asked to answer the questions as listed below. Every question can be answered with the following three options: (1) switches, (2) equal, and (3) flashlight.

Q1 Which interface is more intuitive?

Q2 Which interface felt more efficient to complete the tasks?

Q3 Which interface was easier to use?

Q4 Which interface would you prefer for the tasks at hand?

Q5 Which interface is more fun to use?

Experiment Results

Overall 21 subjects (3 females, 18 males) of ages between 22 and 35 participated in the experiment and completed the two tasks and the questionnaire as described above. The results are presented in the following paragraphs.

Task 1 - One Lamp – The aim of this task is to compare the two different input methods (touchscreen switches and flashlight). Only three participants recognized the labeling on the lamps and

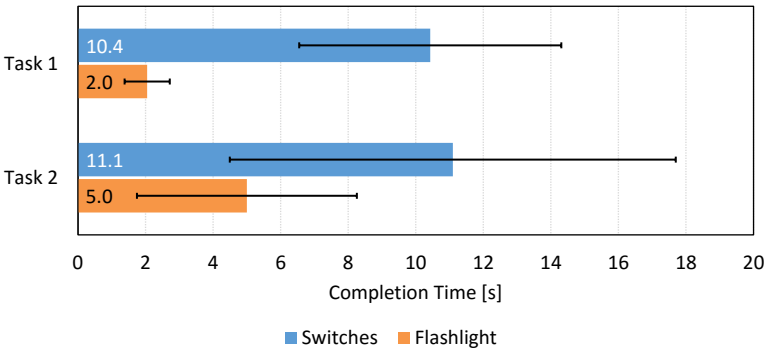


Figure 6.18: Time measurements results for task 1 (one lamp) and task 2 (light configuration) for the two compared interfaces, switches and flashlight. The error bar shows the standard deviation.

could use it to find the corresponding switch. Most participants assumed a natural mapping of the switches and lamps and therefore did not succeed in the beginning. After trying out most or all the switches, eventually the right lamp was switched on. Figure 6.18 shows the timing measurements. On average it took each participant around 10 s. For the second part of this task, the participants had to use the flashlight to switch on a specific lamp. Although not having more than this information, they switched on the flashlight and naturally pointed it towards the lamp they were asked to switch on. On average this task was completed within 2 s.

Task 2 - Light Configuration – The workload of this task is chosen to emulate a more complex scenario, like creating a certain light configuration in a room with multiple lamps, to evaluate the practically of the two interfaces. This task also includes switching off lamps that the participants (wrongly) think to be part of the wanted lamp configuration. Figure 6.18 shows also the time measurements for the second task. Although participants were now familiar with the mapping of switches to lamps, using the switches still took more than twice the time to complete the task (11 s) than using the flashlight (5 s). This result shows that also

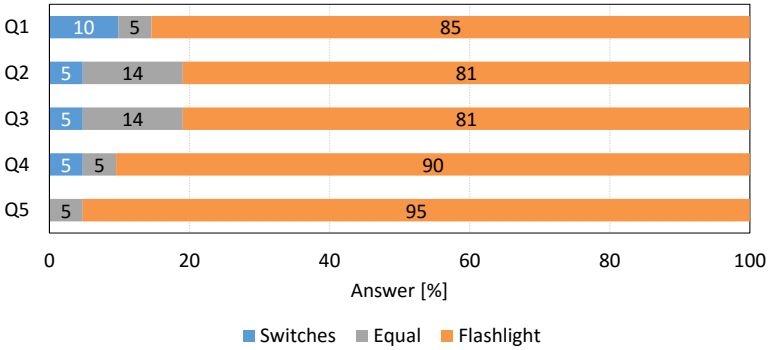


Figure 6.19: Questionnaire answers in percent for the three possible answers (switches, equal, flashlight). Q1 to Q5 corresponds to the questions listed in the questionnaire section.

for a complex use case, the flashlight is an appealing candidate for a more natural interface.

User Feedback – Figure 6.19 summarizes the results of the questionnaire. Almost all participants (85 %) thought of the flashlight as a more intuitive interface for operating the lams. Also, over 80 % of the subjects replied that the flashlight felt more efficient to complete the given tasks. The same percentage stated that the flashlight is also easier to handle. 90 % of the participants would prefer the flashlight to complete the tasks at hand (and therefore also similar tasks). The flashlight seems to also have an entertaining effect, since 95 % named it as the interface that is more fun to use.

6.3.3 *Discussion and Conclusion*

The experiment results show that even when unaware of the functionality, the flashlight was successfully used to complete the tasks within a shorter timespan. According to the answers to the questionnaire, it is also thought of a good interface for the proposed tasks and easy and practical to use. Hence, a flashlight-like pointing device could improve over complicated buttons and switches interfaces (for many lights) where the correspondence

of switch to light is done by manual mapping instead of simple pointing. The flashlight was operated correctly without previous knowledge whereas with the switches interface the participants first needed to learn and understand which switch operated which lamp.

The *LightTouch* interface also scales well with an increasing number of light sources. A common switch interface is usually restricted in space often leading to not having a separate switch for every available light source, but instead for groups of several light sources. Also mapping light sources to switches gets complicated fast, if an increasing number of lights should be operated autonomously. The *LightTouch* interface does not face such problems. It can operate independent of the number of light sources present. Further it is also not only limited to simply toggling the light sources. The light is transporting encoded data and by changing this information, different actions can be trigged, e.g., dimming the lights or changing colors.

Deploying a lighting system can also be complex. Cables for switches or bus systems need to be planted apart from the power grid wiring. Using a flashlight as light system control represents a very intuitive and low-cost wireless interface and does not need additional wiring. Using a flashlight as remote control for lights can also have a downside. Switches are stationary and will always be at the same place and do not need external power sources. In case of the flashlight, it can be laying around anywhere and has to be found first before it can be used, or it can even run out of battery when most needed.

This section presents a novel user interface based on VLC to interact with light sources. A VLC-enabled flashlight can be used to switch on and off lamps by simply pointing towards them. A user study compares the proposed *LightTouch* interface to a common switch interface and the results emphasize that using a VLC-enabled pointing device can compete and even improve existing interfaces. Only one example of many possible use case are presented, underlining the feasibility of such a system. The directionality and visibility of the communication turns a VLC-enabled pointing device into a unique interface for wireless interaction with remote objects.

Dumbledore turned and walked back down the street. On the corner he stopped and took out the silver Put-Outer. He clicked it once and twelve balls of light sped back to their street lamps so that Privet Drive glowed suddenly orange and he could make out a tabby cat slinking around the corner at the other and of the street.

— J.K. Rowling, *Harry Potter and the Philosopher's Stone*

7 CONCLUSION AND FUTURE WORK

The chapters in this thesis describe different VLC systems, based on different hardware, and with different capabilities. They have one thing in common, namely *libvlc*. The goal of this thesis was to develop software-based protocols that can be applied to many different devices, enabling VLC-based networking without significant modifications. With the help of *libvlc*, devices such as light bulbs, flashlights, smartphones, and different VLC prototype boards (mimicking possible future toys and consumer devices), based on only LEDs or additionally equipped with photodiodes, can all speak the same language. Due to this common ground, facilitating efficient data exchange and networking, using a communication channel based on visible light, is straight forward.

7.1 ANALYSIS

The design of *libvlc's* protocols originated from investigating the reasons of a failed MAC experiment, conducted with an earlier version of the communication protocols. One of the requirements for the PHY layer protocol is, along with enabling communication, to enable constant flicker-free light output for human observers, so that the lighting device does not abandon its main purpose. A MAC protocol based on a CSMA/CA scheme heavily relies on a CCA to determine if the communication channel is free or busy. The analysis of the failed experiment attempt uncovered that the employed protocol was not able to differentiate between light used for illumination and light used for communication. In more complex systems, this problem could be solved by modulating the light with different frequencies and apply filters to identify the light contributions. As the objective was to create a low-complex system using only basic microcontrollers, filtering was not an option. This lead to the idea to separate illumination and communication in time.

The PHY layer, implemented as part of *libvlc*, partitions the timeline in ILLU and COM slots. As the name already suggests, the illumination slots provide the necessary light flux when idling, receiving, and transmitting to maintain constant light output. Each ILLU slot is followed by a communication slot, where light is sensed, but also emitted in case of an ongoing data transmission. The ILLU and COM slots are alternating with a frequency above the flicker fusion threshold so that only a steady light is visible by the naked eye. The additional light output during the COM slots are compensated during the following ILLU slot to keep the same average light brightness.

Together with continuous synchronization to align the ILLU and COM slots of participating devices, this simple approach, separating the light used for illumination from the light employed for communication, allows the devices to attribute any sensed light emissions to communication. This allows to define a CCA scheme to reliably determine the current channel state. The ability to detect a busy communication channel reduces data frame collisions and thus improves the overall network throughput and reduces packet delivery delays.

The evaluation results demonstrate that it is possible to assemble a communication system capable of networking, based just on an off-the-shelf 8-bit microcontroller and conventional LEDs. The heavy lifting is done by software-defined protocols. The PHY layer together with FEC allow communication distances close to 2 m at a bit rate of 850 b/s. The working network experiments show that the CCA and the MAC protocol fulfill their duty. For the maximum number of eleven devices, simultaneously trying to access the medium, a resulting throughput of still 750 b/s is possible. As this result is only a little worse than the direct link result, it can be concluded that only few collisions happen thanks to the clearly defined CCA and the efficient MAC protocol. Protocol extensions such as RTS/CTS additionally support the MAC layer in case of hidden stations present.

The software-centric approach allows the extension of *libvlc*, with four different PHY layer modes to increase channel capacity, together with a dynamic adaptation scheme, without effort. Neither additional hardware is required nor changes to existing hard-

ware need to be applied. The dynamic PHY mode scheme adapts to local channel conditions and can increase data throughput by a factor of 5, reaching bit rates that already support the transmission of human voice (digital), still using only LEDs as transceivers. The flexibility of *libvlc* is further increased by introducing a HAA to quickly adopt new hardware platforms.

EnLighting demonstrates a successful application of *libvlc* to a communication system consisting of interconnected light bulbs. Again, thanks to the flexibility of the software-based approach, only minimal changes are necessary to integrate photodiodes as sensing devices with an additional multiplexing scheme to handle four photodiodes with a single ADC. To add IP traffic support, the *libvlc*-based VLC controller is integrated (as an external device) into Linux as an Ethernet device, interfacing with the Linux network stack. As the VLC controller is transparently abstracted as an Ethernet device, any software using IP-based networking can now employ the VLC link for communication. This makes it possible to reuse existing higher layer protocols, as for instance, routing protocols used in a mesh network.

When working with many devices, deployed and distributed over a large area, testbed software can simplify developing, debugging and running experiments immensely. The testbed system developed for *EnLighting* connects all light bulbs via a Wi-Fi control channel to a dedicated network. This allows to remotely control all testbed devices, to collect data and to display real-time protocol information. While working on *EnLighting*, the testbed infrastructure was heavily used and helped to work more efficiently on the devices and protocols.

The proposed light bulbs can still be used for illumination and at the same time broadcast data. What differentiates them from other systems is the ability to also receive light and accept incoming data traffic. The capability to detect light and being able to synchronize to other light bulbs in the vicinity can improve existing applications and enable new use cases. While exchanging data, the light bulbs can bridge communication distances of several meters, also for no line of sight scenarios, using walls, floors, and open doors as light reflectors. Data can be forwarded along floors using multi-hop communication to connect multiple

offices or to eventually reach a gateway connecting to a different network based on other technologies. Furthermore, thanks to the ability to perform a CCA, multiple light bulbs in a room can use *libvlc's* MAC protocols for efficient networking.

Using VLC for indoor localization has high potential. Light naturally forms communication cells with well-defined borders and light sources are already numerously available inside buildings, enabling accurate localization based on trilateration. Light bulbs continuously transmit beacons with location and identification information, which are used together with a corresponding RSSI value to calculate a position. As *EnLighting* also supports MAC, the system does not rely on a random scheme to prevent beacon collisions, thus beacons can be sent at a higher rate, which can lead to faster localization. Additionally, light sources can collect information about their neighbors (extracted from beacons). In case of a failure, the light bulb can be replaced without further configuration steps. As soon as a new bulb is deployed, it can request the necessary configuration data from its neighbors.

The low-complex nature of the discussed communication system and the associated communication protocols make it applicable to various sorts of devices. Mobile phones can use a batteryless peripheral device plugged into the audio jack and adopt the built-in audio system and software to follow *libvlc's* protocols to send and receive data. Instead of an additional peripheral, the smartphone camera can directly be used as a light receiver. The redundantly designed PHY layer protocol allows continuous reception, exploiting the rolling shutter effect, even as gaps between captured video frames are introduced, during which no light can be received. Moreover, LED-based flashlights or similar devices can act as remote controls, transmitting commands to devices they point to, and at the same time provide visual feedback. Visible light communication could change how people interact with devices and provide more intuitive user interfaces.

This thesis presents a software-defined, low-complex and low-cost approach to visible light communication. Thanks to the low-complexity and software-centric concepts, the presented communication protocols for MAC and PHY layer can be applied to various devices, reusing hardware already in place, and extending

their capabilities with VLC at low-cost. The software library, *libvlc*, connects a diversity of devices with the same single set of protocols, forming heterogeneous networks and enabling new interactions and applications. Due to the targeted low-cost and low-complexity environments, applications requiring high data rates, e.g., video streaming, cannot be addressed, but with an unintrusive and ubiquitous system like *EnLighting*, a possible communication fabric for the IoT can be provided, relieving the overloaded radio spectrum.

7.2 FUTURE WORK

With the introduction of the different PHY modes, the achievable data rates could already be improved. The used data interval width for PHY mode `OCTA` is already so narrow that an additional synchronization correction had to be introduced. This prevents the implementation of additional PHY modes with again narrower slots to further increase data rates (at least for a low-cost microcontroller-based solution). Instead of trying to improve in the time space, RGB light sources could encode data symbols using the color space. Different colors can encode multiple bits and thus increase the overall data rate. The light source can still appear to emit white light (for a human observer) if the share of red, green, and blue emissions are equal and constant within a short interval. For the light sensing, either multiple receivers (LEDs or photodiodes) with color filters can be employed or it could be explored if it is possible to use the registered light intensity to derive the color.

The concept, described in Section 5.2.1, to dynamically control the light source's brightness has not been implemented so far. Such functionality is required in scenarios where the light sources used for communication and illumination need to be dimmable. An implementation based on *libvlc* could either modify the duration of ILLU slots, or replace certain ILLU slots with COM slots, to decrease brightness and possibly also increase the data rate, and replace COM slots with ILLU slots, to increase the brightness but also decrease the data rate.

Section 4.3.2 describes how *EnLighting* could be used for indoor localization. The aforementioned section does not evaluate a complete indoor localization but only verifies that RSSI values could be used as a light source to receiver distance estimator. A most attractive candidate for a possible receiver in a real-world scenario is the smartphone, since it can retrieve additional information from other wireless networks and directly display the actual (calculated) position. As described in Section 6.2, a smartphone can be integrated into the *EnLighting* network without any hardware modifications, exploiting the rolling shutter effect to receive data. With the two building blocks, *EnLighting* and the smartphone camera receiver application, all components to implement an indoor positioning system based on RSSI values and trilateration are ready, making localization a topic worth to explore further.

A

CREDIT AND ATTRIBUTION

This appendix lists all the third-party contributions to this thesis and gives credit to the respective authors. The implementation of *libvlc* and *EnLighting* used a number of resources provided by third parties which are released under various licenses that may require to attribute the creator of the material. For this purpose, all employed resources together with the relevant student contributions are listed in this appendix.

A.1 STUDENT CONTRIBUTIONS

The technical realization of *libvlc* and *EnLighting* and the prototype applications would not have been possible without the help of hardworking students that explored the concepts described in this work as part of their Bachelor's, Master's or semester projects. In this section, students who significantly contributed to the work described in this thesis, are listed in the following. All students were supervised by the author of this thesis. The list is ordered alphabetically.

- **Linard Arquint** explored the application of the rolling shutter effect for a smartphone-based VLC receiver as part of his Bachelor's thesis: *Implementation of a Smartphone-based Visible Light Communication System using the Camera as a Receiver*. Based on his findings, he implemented a prototype decoder application as described in Section 6.2.2.
- **Theodoros Bourchas** implemented a first version of the Linux driver used for *EnLighting* (described in Section 4.1.3) and conducted measurements for different transport layer protocols as part of his Master's thesis with the title *Enabling TCP/IP over Visible Light Communication Networks*.

- **Benjamin von Deschwanden** implemented the PHY layer modes introduced in Chapter 5 and helped conducting several measurement campaigns (the results are shown in Section 5.3). He worked on this subject as part of his Master's Thesis with the title: *Light Resource Management for Visible Light Communication Networks*.

- **Thomas Richner** worked on the testbed software for *EnLighting* as part of a semester thesis and helped restructuring *libvlc* for the HAA (Section 5.1.1) as part of his Master's thesis: *Design, Implementation and Evaluation of a Hardware Independent Software Stack for Visible Light Communication*. In addition, he was also involved in designing and assembling the ARM prototype board described in Section 5.1.2.

- **Daniel Schwyn** completed the smartphone audio encoder and decoder application (Section 6.1.2) used together with the audio jack peripheral device. The work was done as part of his Bachelor's thesis with the title: *Implementation of a Smartphone-based VLC System using the Audio Jack as a Communication Front-End*.

- **Josef Ziegler** explored how consumer LED light bulbs can be modified to work together with the software-based communication protocols. As part of a semester thesis he developed a first prototype light bulb for the *EnLighting* system.

A.2 ATTRIBUTIONS

libopencm3. The hardware library libopencm3[1] is used as HPL for the STM ARM processor in *libvlc*. The library code is released under LGPL, version 3[2].

OpenWrt. The Linux distribution used for the *EnLighting* SoC is based on OpenWrt[3]. If not otherwise stated in the source files, the

1. http://libopencm3.org
2. https://opensource.org/licenses/LGPL-3.0
3. https://openwrt.org

OpenWrt build environment is provided under the GPL, version 2[4] terms.

Bootstrap. The *EnLighting* web interface uses HTML and CSS files from the Bootstrap[5] framework. The Bootstrap code is released under the MIT license[6] and copyright by Twitter, Inc.

D3.js. Parts of the visualization displayed in the *EnLighting* web interface are based on D3.js[7] created by Mike Bostock. The D3.js library is released under the BSD[8] license.

Font Awesome. The icons used in the *EnLighting* web interface are provided by Font Awesome[9] created by Dave Gandy. The Font Awesome font is licensed under the SIL Open Font License[10] 1.1.

4. https://opensource.org/licenses/gpl-2.0.php
5. http://getbootstrap.com/
6. https://opensource.org/licenses/MIT
7. https://d3js.org/
8. https://opensource.org/licenses/BSD-3-Clause
9. http://fontawesome.io/
10. http://scripts.sil.org/OFL

BIBLIOGRAPHY

[1] M. Z. Afgani, H. Haas, H. Elgala, and D. Knipp. "Visible Light Communication Using OFDM." In: *Proc. TRIDENTCOM 2006. 2nd Int. Conf. Testbeds and Research Infrastructures for the Development of Networks and Communities*. 2006, 6 pp.–134. DOI: 10.1109/TRIDNT.2006.1649137.

[2] C. An, T. Li, Z. Tian, A. T. Campbell, and X. Zhou. "Visible Light Knows Who You Are." In: *Proceedings of the 2Nd International Workshop on Visible Light Communications Systems*. VLCS '15. Paris, France: ACM, 2015, pp. 39–44. DOI: 10.1145/2801073.2801078.

[3] C. Anderson. *Makers: The New Industrial Revolution*. Crown Business, Apr. 2014.

[4] J. Armstrong, Y. A. Sekercioglu, and A. Neild. "Visible Light Positioning: A Roadmap for International Standardization." In: *IEEE Communications Magazine* 51.12 (Dec. 2013), pp. 68–73. DOI: 10.1109/MCOM.2013.6685759.

[5] J. Baranda, P. Henarejos, and C. G. Gavrincea. "An SDR Implementation of a Visible Light Communication System based on the IEEE 802.15.7 Standard." In: *Proc. 20th Int Telecommunications (ICT) Conf*. May 2013, pp. 1–5. DOI: 10.1109/ICTEL.2013.6632076.

[6] A. C. Boucouvalas, P. Chatzimisios, Z. Ghassemlooy, M. Uysal, and K. Yiannopoulos. "Standards for Indoor Optical Wireless Communications." In: *IEEE Communications Magazine* 53.3 (Mar. 2015), pp. 24–31. DOI: 10.1109/MCOM.2015.7060515.

[7] H. Burchardt, N. Serafimovski, D. Tsonev, S. Videv, and H. Haas. "VLC: Beyond Point-to-Point Communication." In: *IEEE Communications Magazine* 52.7 (July 2014), pp. 98–105. DOI: 10.1109/MCOM.2014.6852089.

[8] F. Che, B. Hussain, L. Wu, and C. P. Yue. "Design and Implementation of IEEE 802.15.7 VLC PHY-I Transceiver." In: *Proc. 12th IEEE Int Solid-State and Integrated Circuit Technology (ICSICT) Conf*. Oct. 2014, pp. 1–4. DOI: 10.1109/ICSICT.2014.7021249.

[9] F. Che, L. Wu, B. Hussain, X. Li, and C. P. Yue. "A Fully Integrated IEEE 802.15.7 Visible Light Communication Transmitter With On-Chip 8-W 85 % Efficiency Boost LED Driver." In: *Journal of Lightwave Technology* 34.10 (May 2016), pp. 2419–2430. DOI: 10.1109/JLT.2016.2535319.

[10] K. Chintalapudi, A. Padmanabha Iyer, and V. N. Padmanabhan. "Indoor Localization Without the Pain." In: *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*. MobiCom '10. Chicago, Illinois, USA: ACM, 2010, pp. 173–184. DOI: 10.1145/1859995.1860016.

[11] J. Classen, J. Chen, D. Steinmetzer, M. Hollick, and E. Knightly. "The Spy Next Door: Eavesdropping on High Throughput Visible Light Communications." In: *Proceedings of the 2nd International Workshop on Visible Light Communications Systems*. VLCS '15. Paris, France: ACM, 2015, pp. 9–14. DOI: 10.1145/2801073.2801075.

[12] G. Corbellini, K. Akşit, S. Schmid, S. Mangold, and T. R. Gross. "Connecting Networks of Toys and Smartphones with Visible Light Communication." In: *IEEE Communications Magazine* 52.7 (July 2014), pp. 72–78. DOI: 10.1109/MCOM.2014.6852086.

[13] J. Corbet, A. Rubini, and G. Kroah-Hartman. *Linux Device Drivers*. O'REILLY & ASSOC INC, 2005.

[14] C. Danakis, M. Afgani, G. Povey, I. Underwood, and H. Haas. "Using a CMOS Camera Sensor for Visible Light Communication." In: *Proc. IEEE Globecom Workshops*. Dec. 2012, pp. 1244–1248. DOI: 10.1109/GLOCOMW.2012.6477759.

[15] P. Deng and M. Kavehrad. "Real-time Software-Defined Single-Carrier QAM MIMO Visible Light Communication System." In: *Proc. Integrated Communications Navigation and Surveillance (ICNS)*. Apr. 2016, pp. 1–11. DOI: 10.1109/ICNSURV.2016.7486354.

[16] P. Dietz, W. Yerazunis, and D. Leigh. "Very Low-Cost Sensing and Communication Using Bidirectional LEDs." In: *UbiComp 2003: Ubiquitous Computing: 5th International Conference, Seattle, WA, USA, October 12-15, 2003. Proceedings*. Springer Berlin Heidelberg, 2003, pp. 175–191. DOI: 10.1007/978-3-540-39653-6_14.

[17] H. Elgala, R. Mesleh, and H. Haas. "Indoor Broadcasting via White LEDs and OFDM." In: *IEEE Transactions on Consumer Electronics* 55.3 (Aug. 2009), pp. 1127–1134. DOI: 10.1109/TCE.2009.5277966.

[18] H. Elgala, R. Mesleh, and H. Haas. "Indoor Optical Wireless Communication: Potential and State-of-the-Art." In: *IEEE Communications Magazine* 49.9 (Sept. 2011), pp. 56–62. DOI: 10.1109/MCOM.2011.6011734.

[19] H. Elgala, R. Mesleh, H. Haas, and B. Pricope. "OFDM Visible Light Wireless Communication Based on White LEDs." In: *Proc. IEEE 65th Vehicular Technology Conf. - VTC2007-Spring*. Apr. 2007, pp. 2185–2189. DOI: 10.1109/VETECS.2007.451.

[20] O. Ergul, E. Dinc, and O. B. Akan. "Communicate to Illuminate: State-of-the-Art and Research Challenges for Visible Light Communications." In: *Physical Communication* 17 (2015), pp. 72–85. DOI: http://dx.doi.org/10.1016/j.phycom.2015.08.003.

[21] J. Ferrandiz-Lahuerta, D. Camps-Mur, and J. Paradells-Aspas. "A Reliable Asynchronous Protocol for VLC Communications Based on the Rolling Shutter Effect." In: *Proc. IEEE Global Communications Conf. (GLOBECOM)*. Dec. 2015, pp. 1–6. DOI: 10.1109/GLOCOM.2015.7417229.

[22] N. Fujimoto and H. Mochizuki. "477 Mbit/s Visible Light Transmission Based on OOK-NRZ Modulation Using a Single Commercially Available Visible LED and a Practical LED Driver with a Pre-Emphasis Circuit." In: *Proc. Optical Fiber Communication Conf. and Exposition and the National Fiber Optic Engineers Conf. (OFC/NFOEC)*. Mar. 2013, pp. 1–3. DOI: 10.1364/NFOEC.2013.JTh2A.73.

[23] C. G. Gavrincea, J. Baranda, and P. Henarejos. "Rapid Prototyping of Standard-compliant Visible Light Communications System." In: *IEEE Communications Magazine* 52.7 (July 2014), pp. 80–87. DOI: 10.1109/MCOM.2014.6852087.

[24] D. Giustiniano, N. O. Tippenhauer, and S. Mangold. "Low-complexity Visible Light Networking with LED-to-LED Communication." In: *Proc. IFIP Wireless Days (WD)*. Nov. 2012, pp. 1–8. DOI: 10.1109/WD.2012.6402861.

[25] J. Graeme. *Photodiode Amplifiers: Op Amp Solutions*. MCGRAW HILL BOOK CO, 2009.

[26] M. Hatch. *The Maker Movement Manifesto: Rules for Innovation in the New World of Crafters, Hackers, and Tinkerers*. 1st ed. McGraw-Hill Education, Sept. 2013.

[27] S. Hecht and E. L. Smith. "Intermittent Stimulation by Light VI. Area and the Relation between Critical Frequency and Intensity." In: *The Journal of General Physiology* 19.6 (1936), pp. 979–989.

[28] P. Hu, P. H. Pathak, X. Feng, H. Fu, and P. Mohapatra. "ColorBars: Increasing Data Rate of LED-to-Camera Communication using Color Shift Keying." In: ACM, 2015.

[29] W. Hussain, H. F. Ugurdag, and M. Uysal. "Software Defined VLC System: Implementation and Performance Evaluation." In: *Proc. 4th Int Optical Wireless Communications (IWOW) Workshop*. Sept. 2015, pp. 117–121. DOI: 10.1109/IWOW.2015.7342278.

[30] V. V. Huynh, L. N. Tuan, and Y. M. Jang. "Priority MAC Based on Multi-Parameter for IEEE 802.15.7 VLC." In: *Proc. ICTC 2011*. Sept. 2011, pp. 257–260. DOI: 10.1109/ICTC.2011.6082592.

[31] V. V. Huynh and Y. M. Jang. "Multi-Parameters Based CSMA/CA for Priority in Visible Light Communication." In: *Proc. Fourth Int. Conf. Ubiquitous and Future Networks (ICUFN)*. July 2012, pp. 13–14. DOI: 10.1109/ICUFN.2012.6261655.

[32] J. Hwang, T. Do, and M. Yoo. "Performance Analysis on MAC Protocol Based on Beacon-Enabled Visible Personal Area Networks." In: *Proc. Fifth Int. Conf. Ubiquitous and Future Networks (ICUFN)*. July 2013, pp. 384–388. DOI: 10.1109/ICUFN.2013.6614847.

[33] "IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." In: *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)* (Mar. 2012), pp. 1–2793. DOI: 10.1109/IEEESTD.2012.6178212.

[34] "IEEE Standard for Local and Metropolitan Area Networks Part 15.7: Short-Range Wireless Optical Communication Using Visible Light." In: *IEEE Std 802.15.7-2011* (Sept. 2011), pp. 1–309. DOI: 10.1109/IEEESTD.2011.6016195.

[35] A. Kamerman and L. Monteban. "WaveLAN ®-II: A High-Performance Wireless LAN for the Unlicensed Band." In: *Bell Labs Technical Journal* 2.3 (1997), pp. 118–133. DOI: 10.1002/bltj.2069.

[36] P. Karn. "MACA - A New Channel Access Method for Packet Radio." In: *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*. Vol. 140. 1990, pp. 134–140.

[37] L. Klaver and M. Zuniga. "Shine: A Step Towards Distributed Multi-Hop Visible Light Communication." In: *Proc. IEEE 12th Int Mobile Ad Hoc and Sensor Systems (MASS) Conf.* Oct. 2015, pp. 235–243. DOI: 10.1109/MASS.2015.78.

[38] K. Kosek-Szott. "A Survey of MAC Layer Solutions to the Hidden Node Problem in Ad-hoc Networks." In: *Ad Hoc Networks* 10.3 (2012), pp. 635–660. DOI: 10.1016/j.adhoc.2011.10.003.

[39] M. Kotaru, K. Joshi, D. Bharadia, and S. Katti. "SpotFi: Decimeter Level Localization Using WiFi." In: *SIGCOMM Comput. Commun. Rev.* 45.4 (Aug. 2015), pp. 269–282. DOI: 10.1145/2829988.2787487.

[40] Y.-S. Kuo, P. Pannuto, and P. Dutta. "System Architecture Directions for a Software-defined Lighting Infrastructure." In: *Proceedings of the 1st ACM MobiCom Workshop on Visible Light Communication Systems*. VLCS '14. Maui, Hawaii, USA: ACM, 2014, pp. 3–8. DOI: 10.1145/2643164.2643166.

[41] Y.-S. Kuo, P. Pannuto, K.-J. Hsiao, and P. Dutta. "Luxapose: Indoor Positioning with Mobile Phones and Visible Light." In: *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*. MobiCom '14. Maui, Hawaii, USA: ACM, 2014, pp. 447–458. DOI: 10.1145/2639108.2639109.

[42] Y.-S. Kuo, T. Schmid, and P. Dutta. "Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface." In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. SenSys '10. Zurich, Switzerland: ACM, 2010, pp. 389–390. DOI: 10.1145/1869983.1870037.

[43] Y.-S. Kuo, S. Verma, T. Schmid, and P. Dutta. "Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface." In: *Proceedings of the First ACM Symposium on Computing for Development*. ACM DEV '10. London, United Kingdom: ACM, 2010, 24:1–24:10. DOI: 10.1145/1926180.1926210.

[44] M. Lacage, M. H. Manshaei, and T. Turletti. "IEEE 802.11 Rate Adaptation: A Practical Approach." In: *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. MSWiM

'04. Venice, Italy: ACM, 2004, pp. 126–134. DOI: 10.1145/1023663.1023687.

[45] C. Landis. "Determinants of the Critical Flicker-Fusion Threshold." In: *Physiological Reviews* (1954).

[46] H. de Lange Dzn. "Eye's Response at Flicker Fusion to Square-Wave Modulation of a Test Field Surrounded by a Large Steady Field of Equal Mean Luminance." In: *Journal of the Optical Society of America* 51.4 (Apr. 1961), pp. 415–421. DOI: 10.1364/JOSA.51.000415.

[47] N. T. Le, S. Choi, and Y. M. Jang. "Cooperative MAC Protocol for LED-ID Systems." In: *Proc. ICTC 2011*. Sept. 2011, pp. 144–150. DOI: 10.1109/ICTC.2011.6082569.

[48] H.-Y. Lee, H.-M. Lin, Y.-L. Wei, H.-I. Wu, H.-M. Tsai, and K. C.-J. Lin. "RollingLight: Enabling Line-of-Sight Light-to-Camera Communications." In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '15. Florence, Italy: ACM, 2015, pp. 167–180. DOI: 10.1145/2742647.2742651.

[49] J. Lee, W. Kim, S.-J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi. "An Experimental Study on the Capture Effect in 802.11a Networks." In: *Proceedings of the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*. WinTECH '07. Montreal, Quebec, Canada: ACM, 2007, pp. 19–26. DOI: 10.1145/1287767.1287772.

[50] L. Li, P. Hu, C. Peng, G. Shen, and F. Zhao. "Epsilon: A Visible Light Based Positioning System." In: *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*. 2014, pp. 331–343.

[51] T. Li, C. An, Z. Tian, A. T. Campbell, and X. Zhou. "Human Sensing Using Visible Light Communication." In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. MobiCom '15. Paris, France: ACM, 2015, pp. 331–344. DOI: 10.1145/2789168.2790110.

[52] T. Li, Q. Liu, and X. Zhou. "Practical Human Sensing in the Light." In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '16. Singapore, Singapore: ACM, 2016, pp. 71–84. DOI: 10.1145/2906388.2906401.

[53] P. Luo, Z. Ghassemlooy, H. L. Minh, X. Tang, and H. M. Tsai. "Undersampled Phase Shift ON-OFF Keying for Camera Communication." In: *Proc. Sixth Int Wireless Communications and Signal Processing (WCSP) Conf*. Oct. 2014, pp. 1–6. DOI: 10.1109/WCSP.2014.6992043.

[54] K. A. Mehr, S. K. Nobar, and J. M. Niya. "IEEE 802.15.7 MAC Under Unsaturated Traffic: Performance Analysis and Queue Modeling." In: *IEEE/OSA Journal of Optical Communications and Networking* 7.9 (Sept. 2015), pp. 875–884. DOI: 10.1364/JOCN.7.000875.

[55] R. Mesleh, H. Elgala, and H. Haas. "Performance Analysis of iIndoor OFDM Optical Wireless Communication Systems." In: *Proc. IEEE Wireless Communications and Networking Conf. (WCNC)*. Apr. 2012, pp. 1005–1010. DOI: 10.1109/WCNC.2012.6213920.

[56] F. M. Mims. *LED Circuits and Projects*. Howard W. Sams, 1973.

[57] A. W. Min and K. G. Shin. "An Optimal Transmission Strategy for IEEE 802.11 Wireless LANs: Stochastic Control Approach." In: *Proc. Mesh and Ad Hoc Communications and Networks 2008 5th Annual IEEE Communications Society Conf. Sensor*. June 2008, pp. 251–259. DOI: 10.1109/SAHCN.2008.39.

[58] R. K. Mondal, N. Saha, and Y. M. Jang. "Joint Scheduling and Rate Allocation for IEEE 802.15.7 WPAN System." In: *Proc. Fifth Int. Conf. Ubiquitous and Future Networks (ICUFN)*. July 2013, pp. 691–695. DOI: 10.1109/ICUFN.2013.6614909.

[59] A. Musa, M. D. Baba, and H. M. A. H. Mansor. "The Design and Implementation of IEEE 802.15.7 Module with ns-2 Simulator." In: *Proc. Int Computer, Communications, and Control Technology (I4CT) Conf*. Sept. 2014, pp. 111–115. DOI: 10.1109/I4CT.2014.6914157.

[60] A. Musa, M. D. Baba, and H. M. A. H. Mansor. "Performance Analysis of the IEEE 802.15.7 CSMA/CA Algorithm Based on Discrete Time Markov Chain (DTMC)." In: *Proc. IEEE Malaysia Int Communications (MICC) Conf*. Nov. 2013, pp. 385–389. DOI: 10.1109/MICC.2013.6805859.

[61] S. K. Nobar, K. A. Mehr, and J. M. Niya. "Comprehensive Performance Analysis of IEEE 802.15.7 CSMA/CA Mechanism for Saturated Traffic." In: *IEEE/OSA Journal of Optical Communications and Networking* 7.2 (Feb. 2015), pp. 62–73. DOI: 10.1364/JOCN.7.000062.

[62] M. Noshad and M. Brandt-Pearce. "Expurgated PPM Using Symmetric Balanced Incomplete Block Designs." In: *IEEE Communications Letters* 16.7 (July 2012), pp. 968–971. DOI: 10.1109/LCOMM.2012.042512.120457.

[63] M. Noshad and M. Brandt-Pearce. "Application of Expurgated PPM to Indoor Visible Light Communications — Part I: Single-User Systems." In: *Journal of Lightwave Technology* 32.5 (Mar. 2014), pp. 875–882. DOI: 10.1109/JLT.2013.2293341.

[64] D. C. O'Brien. "Visible Light Communications: Challenges and Potential." In: *Proc. IEEE Photonic Society 24th Annual Meeting*. Oct. 2011, pp. 365–366. DOI: 10.1109/PHO.2011.6110579.

[65] P. H. Pathak, X. Feng, P. Hu, and P. Mohapatra. "Visible Light Communication, Networking, and Sensing: A Survey, Potential and Challenges." In: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2047–2077. DOI: 10.1109/COMST.2015.2476474.

[66] Y. Qiao, H. Haas, and K. Edward. "Demo: A Software-defined Visible Light Communications System with WARP." In: *Demo at the ACM Workshop on Visible Light Communication Systems*. 2014.

[67] M. Rahaim, A. Miravakili, S. Ray, V. Koomson, M. Hella, and T. Little. "Software Defined Visible Light Communication." In: *Wireless Innovation Forum Conference on Communications Technologies and Software Defined Radio (WInnComm SDR)*. 2014.

[68] N. Rajagopal, P. Lazik, and A. Rowe. "Visual Light Landmarks for Mobile Devices." In: *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*. IPSN '14. Berlin, Germany: IEEE Press, 2014, pp. 249–260.

[69] S. Rajagopal, R. D. Roberts, and S. K. Lim. "IEEE 802.15.7 Visible Light Communication: Modulation Schemes and Dimming Support." In: *IEEE Communications Magazine* 50.3 (Mar. 2012), pp. 72–82. DOI: 10.1109/MCOM.2012.6163585.

[70] I. S. Reed and G. Solomon. "Polynomial Codes over Certain Finite Fields." In: *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), pp. 300–304.

[71] R. D. Roberts. "A MIMO Protocol for Camera Communications (CamCom) Using Undersampled Frequency Shift ON-OFF Keying (UFSOOK)." In: *Proc. IEEE Globecom Workshops (GC Wkshps)*. Dec. 2013, pp. 1052–1057. DOI: 10.1109/GLOCOMW.2013.6825131.

[72] R. D. Roberts. "Space-Time Forward Error Correction for Dimmable Undersampled Frequency Shift ON-OFF Keying Camera Communications (CamCom)." In: *Proc. Fifth Int. Conf. Ubiquitous and Future Networks (ICUFN)*. July 2013, pp. 459–464. DOI: 10.1109/ICUFN.2013.6614861.

[73] R. D. Roberts. "Undersampled Frequency Shift ON-OFF Keying (UFSOOK) for Camera Communications (CamCom)." In: *Proc. 22nd Wireless and Optical Communication*

Conf. May 2013, pp. 645–648. DOI: 10.1109/WOCC.2013.6676454.

[74] R. D. Roberts, S. Rajagopal, and S. K. Lim. "IEEE 802.15.7 Physical Layer Summary." In: *Proc. IEEE GLOBECOM Workshops (GC Wkshps)*. Dec. 2011, pp. 772–776. DOI: 10.1109/GLOCOMW.2011.6162558.

[75] E. Sarbazi and M. Uysal. "PHY Layer Performance Evaluation of the IEEE 802.15.7 Visible Light Communication Standard." In: *Proc. 2nd Int Optical Wireless Communications (IWOW) Workshop*. Oct. 2013, pp. 35–39. DOI: 10.1109/IWOW.2013.6777772.

[76] S. Schmid, S. Mangold, and T. R. Gross. "Wireless LAN in Paired Radio Spectrum with Downlink-Uplink Separation." In: *Proc. IEEE Wireless Communications and Networking Conf. (WCNC)*. Apr. 2014, pp. 3349–3354. DOI: 10.1109/WCNC.2014.6953092.

[77] S. Schmid, D. Schwyn, K. Akşit, G. Corbellini, T. R. Gross, and S. Mangold. "From Sound to Sight: Using Audio Processing to Enable Visible Light Communication." In: *Proc. IEEE Globecom Workshops (GC Wkshps)*. Dec. 2014, pp. 518–523. DOI: 10.1109/GLOCOMW.2014.7063484.

[78] S. Schmid, L. Arquint, and T. R. Gross. "Using Smartphones as Continuous Receivers in a Visible Light Communication System." In: *Proceedings of the 3rd International Workshop on Visible Light Communications Systems*. VLCS '16. New York, NY, USA: ACM, 2016.

[79] S. Schmid, T. Bourchas, S. Mangold, and T. R. Gross. "Linux Light Bulbs: Enabling Internet Protocol Connectivity for Light Bulb Networks." In: *Proceedings of the 2nd International Workshop on Visible Light Communications Systems*. VLCS '15. Paris, France: ACM, 2015, pp. 3–8. DOI: 10.1145/2801073.2801074.

[80] S. Schmid, G. Corbellini, S. Mangold, and T. R. Gross. "An LED-to-LED Visible Light Communication System with Software-based Synchronization." In: *Proc. IEEE Globe-*

com Workshops. Dec. 2012, pp. 1264–1268. DOI: 10.1109/GLOCOMW.2012.6477763.

[81] S. Schmid, G. Corbellini, S. Mangold, and T. R. Gross. "LED-to-LED Visible Light Communication Networks." In: *Proceedings of the Fourteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. MobiHoc '13. Bangalore, India: ACM, 2013, pp. 1–10. DOI: 10.1145/2491288.2491293.

[82] S. Schmid, G. Corbellini, S. Mangold, and T. R. Gross. "Continuous Synchronization for LED-to-LED Visible Light Communication Networks." In: *Proc. 3rd Int Optical Wireless Communications (IWOW) Workshop in*. Sept. 2014, pp. 45–49. DOI: 10.1109/IWOW.2014.6950774.

[83] S. Schmid, T. Richner, S. Mangold, and T. R. Gross. "EnLighting: An Indoor Visible Light Communication System Based on Networked Light Bulbs." In: *Sensing, Communication, and Networking (SECON), 2016 13th Annual IEEE International Conference on*. June 2016.

[84] S. Schmid, J. Ziegler, G. Corbellini, T. R. Gross, and S. Mangold. "Using Consumer LED Light Bulbs for Low-cost Visible Light Communication Systems." In: *Proceedings of the 1st ACM MobiCom Workshop on Visible Light Communication Systems*. VLCS '14. Maui, Hawaii, USA: ACM, 2014, pp. 9–14. DOI: 10.1145/2643164.2643170.

[85] S. Schmid, J. Ziegler, T. R. Gross, M. Hitz, A. Psarra, G. Corbellini, and S. Mangold. "(In)Visible Light Communication: Combining Illumination and Communication." In: *ACM SIGGRAPH 2014 Emerging Technologies*. SIGGRAPH '14. Vancouver, Canada: ACM, 2014, 13:1–13:1. DOI: 10.1145/2614066.2614094.

[86] D. Schmidt, D. Molyneaux, and X. Cao. "PICOntrol: Using a Handheld Projector for Direct Control of Physical Devices Through Visible Light." In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. UIST '12. Cambridge, Massachusetts, USA: ACM, 2012, pp. 379–388. DOI: 10.1145/2380116.2380166.

[87] E. F. Schubert. *Light-Emitting Diodes*. Cambridge University Press, 2006.

[88] S. Sen, J. Lee, K.-H. Kim, and P. Congdon. "Avoiding Multipath to Revive Inbuilding WiFi Localization." In: *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '13. Taipei, Taiwan: ACM, 2013, pp. 249–262. DOI: 10.1145/2462456.2464463.

[89] R. Singh, T. O'Farrell, and J. P. R. David. "Performance Evaluation of IEEE 802.15.7 CSK Physical Layer." In: *Proc. IEEE Globecom Workshops (GC Wkshps)*. Dec. 2013, pp. 1064–1069. DOI: 10.1109/GLOCOMW.2013.6825133.

[90] N. O. Tippenhauer, D. Giustiniano, and S. Mangold. "Toys Communicating with LEDs: Enabling Toy Cars Interaction." In: *Proc. IEEE Consumer Communications and Networking Conf. (CCNC)*. Jan. 2012, pp. 48–49. DOI: 10.1109/CCNC.2012.6181045.

[91] F. Tobagi and L. Kleinrock. "Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution." In: *IEEE Transactions on Communications* 23.12 (Dec. 1975), pp. 1417–1433. DOI: 10.1109/TCOM.1975.1092767.

[92] D. Tsonev et al. "A 3-Gb/s Single-LED OFDM-Based Wireless VLC Link Using a Gallium Nitride." In: *IEEE Photonics Technology Letters* 26.7 (Apr. 2014), pp. 637–640. DOI: 10.1109/LPT.2013.2297621.

[93] J. Vučić, C. Kottke, K. Habel, and K. D. Langer. "803 Mbit/s Visible Light WDM Link Based on DMT Modulation of a Aingle RGB LED Luminary." In: *Proc. and the National Fiber Optic Engineers Conf. Optical Fiber Communication Conf. and Exposition (OFC/NFOEC)*. Mar. 2011, pp. 1–3.

[94] J. Vučić, C. Kottke, S. Nerreter, K. Habel, A. Büttner, K. D. Langer, and J. W. Walewski. "230 Mbit/s via a Wireless Visible-Light Link Based on OOK Modulation of Phosphorescent White LEDs." In: *Proc. Conf Optical Fiber Communi-*

cation (OFC), collocated National Fiber Optic Engineers Conf. (OFC/NFOEC). Mar. 2010, pp. 1–3.

[95] Q. Wang and D. Giustiniano. "Intra-Frame Bidirectional Transmission in Networks of Visible LEDs." In: *IEEE/ACM Transactions on Networking* PP.99 (2016), pp. 1–13. DOI: 10.1109/TNET.2016.2530874.

[96] Q. Wang, D. Giustiniano, and D. Puccinelli. "An Open Source Research Platform for Embedded Visible Light Networking." In: *IEEE Wireless Communications* 22.2 (Apr. 2015), pp. 94–100. DOI: 10.1109/MWC.2015.7096291.

[97] Q. Wang and D. Giustiniano. "Communication Networks of Visible Light Emitting Diodes with Intra-Frame Bidirectional Transmission." In: *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*. CoNEXT '14. Sydney, Australia: ACM, 2014, pp. 21–28. DOI: 10.1145/2674005.2675000.

[98] Q. Wang, D. Giustiniano, and O. Gnawali. "Low-Cost, Flexible and Open Platform for Visible Light Communication Networks." In: *Proceedings of the 2nd International Workshop on Hot Topics in Wireless*. HotWireless '15. Paris, France: ACM, 2015, pp. 31–35. DOI: 10.1145/2799650.2799655.

[99] Q. Wang, D. Giustiniano, and D. Puccinelli. "OpenVLC: Software-defined Visible Light Embedded Networks." In: *Proceedings of the 1st ACM MobiCom Workshop on Visible Light Communication Systems*. VLCS '14. Maui, Hawaii, USA: ACM, 2014, pp. 15–20. DOI: 10.1145/2643164.2643167.

[100] Q. Wang, S. Yin, O. Gnawali, and D. Giustiniano. "Demo: OpenVLC1.0 Platform for Research in Visible Light Communication Networks." In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. MobiCom '15. Paris, France: ACM, 2015, pp. 179–181. DOI: 10.1145/2789168.2789173.

[101] Z. Wang, C. Yu, W. D. Zhong, J. Chen, and W. Chen. "Performance of Variable M-QAM OFDM Visible Light Communication System with Dimming Control." In: *Proc. 17th Opto-Electronics and Communications Conf. (OECC)*. July 2012, pp. 741–742. DOI: 10.1109/OECC.2012.6276607.

[102] A. Wilson and S. Shafer. "XWand: UI for Intelligent Spaces." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '03. Ft. Lauderdale, Florida, USA: ACM, 2003, pp. 545–552. DOI: 10.1145/642611.642706.

[103] F.-M. Wu, C.-T. Lin, C.-C. Wei, C.-W. Chen, Z.-Y. Chen, and H.-T. Huang. "3.22-Gb/s WDM Visible Light Communication of a Single RGB LED Employing Carrier-Less Amplitude and Phase Modulation." In: *Proc. Optical Fiber Communication Conf. and Exposition and the National Fiber Optic Engineers Conf. (OFC/NFOEC)*. Mar. 2013, pp. 1–3.

[104] B. Xie, G. Tan, and T. He. "SpinLight: A High Accuracy and Robust Light Positioning System for Indoor Applications." In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. SenSys '15. Seoul, South Korea: ACM, 2015, pp. 211–223. DOI: 10.1145/2809695.2809713.

[105] Z. Yang, Z. Wang, J. Zhang, C. Huang, and Q. Zhang. "Wearables Can Afford: Light-weight Indoor Positioning with Visible Light." In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '15. Florence, Italy: ACM, 2015, pp. 317–330. DOI: 10.1145/2742647.2742648.

[106] C. Zhang, J. Tabor, J. Zhang, and X. Zhang. "Extending Mobile Interaction Through Near-Field Visible Light Sensing." In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. MobiCom '15. Paris, France: ACM, 2015, pp. 345–357. DOI: 10.1145/2789168.2790115.

[107] X. Zhou and A. T. Campbell. "Visible Light Networking and Sensing." In: *Proceedings of the 1st ACM Workshop on Hot Topics in Wireless*. HotWireless '14. Maui, Hawaii, USA: ACM, 2014, pp. 55–60. DOI: 10.1145/2643614.2643621.

[108] F. Zünd, P. Bérard, A. Chapiro, S. Schmid, M. Ryffel, M. Gross, A. H. Bermano, and R. W. Sumner. "Unfolding the 8-bit Era." In: *Proceedings of the 12th European Conference on Visual Media Production*. CVMP '15. London, United Kingdom: ACM, 2015, 9:1–9:10. DOI: 10.1145/2824840.2824848.

ACRONYMS

AARF	Adaptive Auto Rate Fallback
AC	Alternating Current
ACK	Acknowledgment
ADC	Analog-to-Digital Converter
API	Application Programming Interface
ARF	Auto Rate Fallback
ARM	Advanced RISC Machines
ASK	Amplitude-Shift Keying
BFSK	Binary Frequency Shift Keying
CCA	Clear-Channel Assessment
CDF	Cumulative Distribution Function
CMOS	Complementary Metal-Oxide-Semiconductor
COM	Communication
CRC	Cyclic Redundancy Check
CSK	Color-Shift Keying
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CTS	Clear to Send
CW	Contention Window

DAC	Digital-to-Analog Converter
DC	Direct Current
DIFS	Distributed Interframe Space
DMA	Direct Memory Access
FCS	Frame Check Sequence
FEC	Forward Error Correction
FIFO	First In – First Out
FPGA	Field Programmable Gate Array
FSK	Frequency Shift Keying
GPIO	General-Purpose Input/Output
GPS	Global Positioning System
HAA	Hardware Adaptation Architecture
HAL	Hardware Adaption Layer
HIL	Hardware Interface Layer
HPL	Hardware Presentation Layer
I/O	Input/Output
ICMP	Internet Control Message Protocol
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
ILLU	Illumination
IoT	Internet of Things
IP	Internet Protocol
ISM	Industrial, Scientific and Medical

ISO	International Organization for Standardization
LB	Light Bulb
LED	Light Emitting Diode
LiPo	Lithium Polymer
MAC	Medium Access Control
MIPS	Microprocessor without Interlocked Pipeline Stages
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor
MSS	Maximum Segment Size
OFDM	Orthogonal Frequency-Division Multiplexing
OFDMA	Orthogonal Frequency-Division Multiple Access
OLSR	Optimized Link State Routing
OOK	On-Off Keying
PCB	Printed Circuit Board
PD	Photodiode
PHY	Physical
PLL	Phase-Locked Loop
PPM	Pulse-Position Modulation
PSK	Phase-Shift Keying
PWM	Pulse Width Modulation
RAM	Random-Access Memory
RF	Radio Frequency
RGB	Red Green Blue
RISC	Reduced Instruction Set Computer
RLL	Run Length Limited

RSSI	Received Signal Strength Indication
RTNS	Retransmission
RTS	Request to Send
RTT	Round-Trip Time
RX	Receiving
SDR	Software Defined Radio
SFD	Start Frame Delimiter
SIFS	Short Interframe Space
SMT	Surface-Mount Technology
SoC	System-on-a-Chip
SRAM	Static Random-Access Memory
STM	STMicroelectronics
TCP	Transmission Control Protocol
TX	Transmitting
UDP	User Datagram Protocol
UFSOOK	Undersampled Frequency Shift OOK
USART	Universal Synchronous Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VLC	Visible Light Communication

COLOPHON

This thesis was typeset in LaTeX using the typographical look-and-feel `classicthesis`. The bibliography was generated using biblatex.

Final version as of December 15, 2016 (version 1.0).