Filzmaier Josef, BSc

Design and Implementation of Pikrit: A Home Automation Source Routing Protocol for Wireless Networks

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institute for Technical Informatics

Graz, April 2017

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Abstract

Im Bereich der Heimautomatisierung steigt das wirtschaftliche Interesse an kabellosen Sensor- und Aktornetzwerken stetig. Besonders im Anwendungsgebiet der Heimautomatisierung wurden in einer relativ kurzen Zeitspanne eine große Menge an Innovationen voran getrieben. Ziel ist hierbei meist eine Vereinfachung der alltäglichen Aufgaben im Haushalt. Zur Realisierung dieser Aufgaben existieren viele generische Protokolle, allerdings wird der fehlende Kommunikationsstandard für Entwickler solcher Systeme zum Stolperstein. Es gibt eine Vielzahl an generischen Protokollen, jedoch sind diese meist untereinander Inkompatibel. Die Abstimmung auf einen allgemein gültigen Standard erweist sich als besonders schwierig, da jede Firma seine eigene Kreation als Norm für zukünftige Systeme deklariert wissen will.

Diese Masterarbeit stellt eine neue Protokollspezifikation namens Pikrit vor. Pikrit wurde für ein spezielles Anwendungsgebiet entwickelt und versucht, im Gegensatz zu den meisten kommerziell verwendeten Protokollen, keine allgemeine Lösung für alle Produkte zu sein. Der Designfokus liegt auf statisch montierte Sensoren und Aktoren in der Heimautomatisierung. Grundsätzlich versucht Pikrit ein besonders ressourcenschonendes Protokoll zu sein um eine Paketweiterleitung über mehrere Teilnehmer auch für eingebettete Systeme effizient zu gestalten. Es wird dabei die Prämisse ausgenutzt, dass alle Netzwerkteilnehmer statisch montiert sind und sich innerhalb des Gebäudes nicht bewegen. Ein weiteres Ziel von Pikrit ist es, erweiterbar zu sein, um es Anwendungsentwicklern zu ermöglichen ihre eigenen Protokolle aufbauend auf dem Pikrit Protokoll zu realisieren. Anhand einer Referenzimplementierung wird die Performanz mit anderen gängigen Routingprotokollen verglichen. Basierend auf Simulationen wird gegen Ende der Arbeit auf die Vor- und Nachteile des Protokolls eingegangen und es wurde gezeigt, dass das Protokoll die Designziele erfüllt.

Abstract

Wireless sensor and actor networks are steadily gaining public interest. Especially within the home automation line of business there is a plethora of innovations to be found within a relatively short time period. Typically the goal is to trivialize housekeeping. A big problem innovators are facing when designing new home automation products is which communication standard to use. There are a lot of general purpose standards currently in use which are mostly incompatible with each other. So the vision to make housekeeping a unified and simple experience is hindered until a common standard is agreed upon. Achieving this is a very hard task because there is a lot of competitive interest between companies to have their standards declared as the norm.

This thesis introduces a new protocol specification named Pikrit. Contrary to the desire of many companies Pikrit does not try to be a general purpose solution. Instead it centers design focus around statically mounted sensors and actors within home automation. The basic idea behind Pikrit is to be very lightweight in order to allow for a reliable low latency communication even when using multi-hop routing on embedded devices. To achieve this it makes use of the premise that participants are generally non-moving parts. Extensibility is also a key design goal to allow developers to realize their own higher level applications using the Pikrit stack. Simultaneously to drafting a specification a reference implementation has been built to perform a direct comparison with competitors. Real world and simulation experiments were performed to showcase the protocols strengths and weaknesses and to prove that the protocol fulfills its design goals.

Acknowledgement

At this juncture I want to express my gratitude to my supervisor Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger for the useful comments, remarks and engagement throughout the process of creating this thesis. Furthermore I would like to thank Manuel Stanglechner, Dipl.-Ing. (FH) Wolfgang Profer and all other involved employees and stakeholders at HELLA for tendering this thesis, the active involvement and giving me the opportunity to work on this exciting topic.

As a very important part of my life I also want to thank my friends which kept me sane throughout the years. Their continuous endorsement enriched my personal life by keeping a good balance to my professional life. They also helped me overcome setbacks and find new motivation. I greatly value all of my friendships.

Most importantly, I want to thank my lovely family which has been supportive, motivational and inspiring for the duration of my whole life. I am especially grateful to my parents, Josef and Ramona Filzmaier. Their fairness and dedication was very influential, they backed my time at university and are my close friends.

Without all of you this thesis would not have been possible.

Thank you.

Contents

Chapter 1

Introduction

"Perplexity is the beginning of knowledge"[1]

1.1 Motivation

Humanity is the most influential species on planet Earth, measured by the amount of space we control and call our own. The population steeply increases since centuries and we do not generally fear natural predators, besides ourselves. People have discovered and surveyed almost all places on earth and there are even plans to conquer space and other planets. Stubbornness, curiosity and problem solving instincts are, besides from anatomical and cognitive advantages, some peoples strong points which led to this dominance. We are by nature experts at identifying potential for improvement, adapting to our environment and inventing tools to generalize and solve arising problems. Even though the solution to a problem can seem impossible at times, people are resilient enough to keep on trying until it is solved. These human properties shaped the world we live in today, especially when focusing on the topic of technology.

The advancements in this very matter have fundamentally changed our lifestyle, and are doing so at an accelerated pace as of the past few decades. Traveling around the globe has been trivialized by cars, air planes and other vehicles. But the need to travel dwindles as more and more tasks can be done over the internet. Borders between the real world and imagination are melted when using special virtual reality headsets which allow for an immersive dive into imaginary worlds.

The promise of a lot of these products is to make solving every day tasks in life easier, or even automated, and more efficient. Technology also helps people with

[1] Khalil Gibran

disabilities improve their lives and let them do things which they would not be able to do otherwise. In general a lot of investigation is made in order to simplify day to day living.

Recent developments also increasingly automate the lifestyle within our living area. Many helpful components inside and around the house are equipped with sensors and actors which are connected to the internet. On the one hand such components can easily automate things like switching off the lights if they are not needed in order to save energy. On the other hand people can always check the status of their components from everywhere over the internet using their smartphone. This enables a new, ideally easier and more secure way of living.

So the motivation for me is being able to interconnect things using technology and making peoples lifes easier. It also feels like an interesting topic as it has just started emerging and is still in its infancy. Manufacturers are unconsolidated about which communication standard to use and a lot of things are up in the air. These uncertainties and the continuous changes through improvements in technology in combination with new ideas that frequently arise are exciting to me, and are in part reason why I chose to work on this topic.

1.1.1 Goals

A primary goal of this thesis is to clarify the trade-off between developing a custom application-adapted routing protocol and using already established protocols within wireless embedded mesh routing. In order to shed some light on this essential question the requirements of embedded devices within building automation are refurbished. Given this context a basic protocol design shall be forged and then translated into a reference implementation example. Reasoning on why and in which areas the suggested protocol is better than already available ones is also important. Last but not least some critical impacts of the digital lifestyle shall be discussed as sometimes the more important question is the 'why?' as opposed to the 'how?'.

1.2 Challenges within digital culture

As software is getting central in the way we communicate with each other and interact with our environment, I think it is important to stress the cultural implications of the modern digital lifestyle. Technology makes the life easier in a variety of ways but it is also abused on a regular basis in unfair or unethical ways, which a lot of users do not even notice. Major companies often take advantage of their monopoly position, enforcing certain restrictions on the user, and sometimes even

spy on and monitor their customers behavior. This topic is in itself a rather large one, so I will only broach some concerning topics.

1.2.1 The value of privacy

Throughout the history of mankind there has never been a time where data about individuals were so widely available as in the present. Data collection, indexing and calculating metrics from it is trivialized through advancements in technology. The journal article [Ste+12, p. 1755] describes an hypothetical "highly-tracked" day in the life of an individual person, from leaving the house in the morning to going to bed in the evening. We are almost at the point where every activity throughout the day could be recorded by diverse connected devices. In addition to this continual data collection our society grew into one which voluntarily broadcasts a lot of personal information. Personal photos, opinions, locations, activities, major life events and even trivialities like what food was served for dinner or some mundane thoughts form a stream of data which represents our life online. The combination of the tracked and voluntarily shared data draws a stunningly detailed picture of an individual life. This information is very valuable for governments and businesses alike as described in the following Sections.

1.2.2 Government and security agencies

Governments use personal data in order to detect illegal activities or knowing early about the possibilities of terroristic activity from certain people. While this sounds like a security enrichment to our society, data collection and indexing can also be dangerous depending on the form of usage of the data. It is important to keep a good balance between security and the collection of private data. In the second world war, as an extreme example, the Gestapo (German secret service) collected lists identifying homosexual people. The lists were later used to chase these people in order to castrate them or commit them to psychiatric institutions. At this time it was not very easy to gather such lists, but with today's technology such data acquisition could easily be misused to chase minorities. Hopefully humanity has learned its lesson from history and embraces basic human rights like privacy and many others.

1.2.3 Trading privacy for corporate controlled services

The financial model of software companies has evolved over time. In the beginnings fixed prices were charged for snapshots of software implementations. But as anyone knows who has had some experience in programming, software can always be

improved and there is no such thing as a "final" product. Some companies provided a set of updates for their already sold products and then eventually released an overhauled version for which users were charged again. While this pricing model is still widely used today, a lot of new ways of charging for software have been explored. Product as a service, where the updates are free but the user pays monthly in order to use the product, or micro transactions within products, where the product itself is free but advanced features, templates, assets or even only cosmetic changes have to be paid for, are some common examples.

A more controversial payment method, which especially pertains social interaction services, is the trade of private information for products. These products are usually completely free of charge, but not without cost as they enforce you to provide basic private data like name or birthday, and encourage you to define and share all your social contacts, your hobbies, your habits, which things you enjoy and dislike amongst other personal information. Every bit of personal information given to these companies is very valuable for them. As the information is indexed and easily available, advertisements can be placed much more precisely to a target audience. Other companies take advantage of this and spend huge amounts of money in order to advertise their product to the most relevant target groups. Many users of such services are not even aware that all their activities are being monetized, because most people aren't lawyers who read the End-User License Agreement (EULA), they accept it blindly. From this point of view private information can be seen as currency with which the service is paid for.

1.2.4 Free and proprietary software models

Software licenses can be fundamentally differentiated between free (often also called "libre" or "open") and proprietary licenses. In the very early days of computing the source code of algorithms was shared by university students, researchers and companies. Companies then proposed the proprietary model, where the source code to the software is only available to the company and the sold product is the compiled executable binary. Using this proprietary model is very beneficial to companies as development costs are usually nonrecurring and the distribution of software is very inexpensive. While most customers are satisfied with this way of distributing software, there are also a lot of concerns among developers and people with a certain amount of know-how. If the source code is not available, users can not actually know if the program is just doing what it is supposed to do or if there are some malicious features baked in. Also, the company providing the executables has complete control over the things a customer can do with his computer. So what can be achieved with a computer is limited by the bundled software, which implies a restriction on the users freedom.

This lead to mistrust within parts of the technical community which then decided to author licenses which embraced this kind of freedom and the availability of source code. Developing software this way encouraged a lot of people around the globe to participate, which lead to some very dynamic development processes. Openness attracted new developers to projects who might contribute free of charge. Since it was created this open way of distributing software has had the issue that companies struggled to find a way to earn money because the available source code made it easy to create an executable binary by the users themselves. Some companies had success in selling support for their products and in recent history this method is being well received by the maker community and companies alike. Both, proprietary and open source software models have examples for very good and very bad software maturity. When developing a software product it is often easier, cheaper (or even free) and less restrictive to join or fork open projects. Proprietary solutions shift the responsibilities to external companies which can also be advantageous.

1.2.5 Digital divide between cultures

The adoption rate of cutting edge technology is not equally spread throughout the different cultures of our planet. A good overview for this topic is given in [PS16], which explains and compares four common theories about unevenness in access to information and telecommunication technologies between individuals and culture groups. As an excerpt two out of the four theories are summarized here:

- **Adoption-Diffusion Theory**
  This theory is not directly bound to the digital divide, but is rather a general approach to analyze processes, like the introduction of innovation within a single social system, as a local market. Once a new and innovating technology product is released for the public, early adopters and innovators are among the first to acquire it. Those people are very influential in that they tend to form “diffusion cells” over small neighborhood areas. The trend of buying this product therefore increases over time as the knowledge about its existence spreads among the people. After this period the demand for the product begins to dwindle as only laggards are left to acquire the product. In other words the market share will eventually reach saturation.

- **Theory of Digital Technology Access and Societal Impacts**
  A 10 year long study shows the connection between initial personal and positional background of a person and the therefore resulting access to resources. Here personal background among others refers to human properties like age, gender and health while positional background relates to labor, education, nation and household. The better the initial condition within this system, the more resources are generally available. Resources which provide skill and material access to information and communication technology. This access also leads to more participation within society, as activity within social networks, politics and other.

1.2.6 Information and disinformation

Sharing information is easier than ever. The speed, breadth and ease of spreading knowledge has opened up unprecedented possibilities. Most of those opportunities have had widespread positive impacts on the lives of billions of people around the globe. But the ease of distributing data also led to its misuse. Frequently, false information is ably disguised and presented as the truth. Such messages, also often called hoax as described in [KWL16], do play a detrimental role in our culture. Reasons for people to spread misinformation vary. A goal of spreading misinformation can be to let people believe in irrefutable theories which may corruptly bend their view on reality. Other times people denote false facts to deliberately deceive or betray an audience. If misinformation about some fact gains wide spread adoption or even acceptance the implications can be fatal. Myths about medical issues can be dangerous to victims of such false claims. Another more extreme example would be incorrect political propaganda which can polarize communities and in the worst case can lead to deadly attacks, revolts or even wars.

1.3 Partner

This Master thesis is done in cooperation with HELLA, sun- and weather protection systems Gesellschaft mit beschränkter Haftung (GmbH). The official Logo of the company can bee seen in Figure 1.1. HELLA is located in Abfaltersbach in East Tyrol and has, as of this writing, about 1200 employees. The product palette includes blinds, awnings and roller shutters.

Figure 1.1: Official HELLA logo

1.3.1 Problem statement

HELLA is developing a home automation solution called ONYX. It is a solution with focus on convenient brightness control throughout the house. Other use cases are planned for the future. In order to be easily extensible for current houses it uses wireless communication. This product is already available on the market with the following components:

- **ONYX.CENTER**
  As the heart of processing commands from clients it sends the corresponding commands wirelessly to other participants. It also is responsible for calculating and prioritizing commands that are generated from automatic scenarios.
- **ONYX.NODE**
  This piece is placed between the power supply of a motor for sun blocking components like raffstores, roller shutters or awnings. It receives commands from ONYX.CENTER, executes them and gathers information about the state of the product it controls.
- **ONYX.CONNECTOR**
  Is a similar component to ONYX.NODE but with a special form factor allowing it to be placed within the frame of shading products.
- **ONYX.WEATHER**
  In order to react to changes within the environment ONYX.WEATHER provides data like temperature, brightness and wind speeds and reports those back wirelessly to a ONYX.CENTER.

Figure 1.2: Small house using ONYX

Additionally the product involves intuitive smartphone apps for all major platforms. The system is generally being well received from its customers. However, a problem that is reported frequently is the lack of communication range for larger buildings.

Figure 1.3: Large house using ONYX

Consider a small single family house as shown in Figure 1.2. The area highlighted in green represents the wireless communication range of an ONYX.CENTER. This range is heavily influenced by blocking objects, reflection and general position of the box within the house. As shown in the figure in this case there are no problems with the communication range as the green area stretches across the whole house.

Where there do arise a lot of problems is within larger buildings as shown in Figure 1.3. Compared to Figure 1.2 one can see that the green highlighted wireless range is not covering up the whole building. This scenario can occur in larger buildings with thick ferroconcrete walls. Such large buildings often have a lot of components installed which all directly communicate with the ONYX.CENTER component. The idea is to let all those devices spread throughout the building communicate with each other in order to route the necessary data to its destination.

The basic problem to solve is to extend the range of wireless communication partners in home automation devices by using their resources more efficiently.

1.4 Outline

The remainder of this thesis is structured as follows. In Chapter 2, a brief introduction including a history of embedded wireless mesh networks is given. Also, popular tools for developing and simulating wireless sensor and actor networks are explored. The requirements and design aspects of Pikrit are discussed in Chapter 3. With these requirements in mind, Chapter 5 evaluates simulations of diverse protocols like Routing Protocol for Low power and Lossy Networks (RPL) and Contiki Rime mesh regarding their performance in comparison to Pikrit. A reference implementation of Pikrit is demonstrated in Chapter 4 and the conclusion of this thesis with a discussion about the achieved results including an outlook for possible developments in the future is shown in Chapter 6.

Chapter 2

Wireless sensor and actor networks

"All of the biggest technological inventions created by man says little about his intelligence, but speaks volumes about his laziness."[1]

2.1 Introduction

Transferring data from one place to another requires some sort of medium to establish a transmission channel on. There are currently three competitive data transfer mediums. The first is the electrical method, where the impulse of electrons in conducting metal alloys is used to carry information. Secondly there are optical solutions, where the information is transferred via light modulation. Light is an electromagnetic wave which has a wavelength close to the wavelengths visible to the human eye. But the most relevant option for this thesis are wireless communication channels. Here electromagnetism is used in different modulations in order to transfer data. While both light and wireless transmissions are based on electromagnetism their wavelengths differ greatly which gives them unique properties.

Each of these mediums have their own unique set of technical, economical and convenience related advantages and disadvantages, depending on the requirements to the communication system. A big problem when choosing the electrical solution is upgradability, despite being a very reliable choice otherwise. If cables have to be laid subsequently this often implies infeasible costs. Optical solutions have basically the same problem, with the exception of a few special cases where line of sight is granted between the communication partners.

[1] Mark Kennedy

2.1.1 Wireless Communication

Due to this fact a lot of companies focus on wireless solutions, which do not share this drawback. Wireless solutions are easily upgradable and have additional positional flexibility. However, the spatial range is restricted by physical limitations and the allowed transmission power is regulated by legal entities. Especially in constructions with shielding elements, for example buildings with thick and ferroconcrete walls or underfloor heatings, the desired communication distance often cannot be reached.

2.1.2 Mesh Networking

One promising idea is to solve this problem by using what is called mesh networking or routing. The idea behind a meshed network is that there are often more than one communication devices in range, but not all devices can talk to each other directly. Figure 2.1 shows an example mesh network. The circles with imprinted identifiers represent communication partners (nodes), while the arrows between two nodes in the network indicate that those communication devices can talk to each other directly. So there is no way for node A to share data with node D using direct communication channels only.

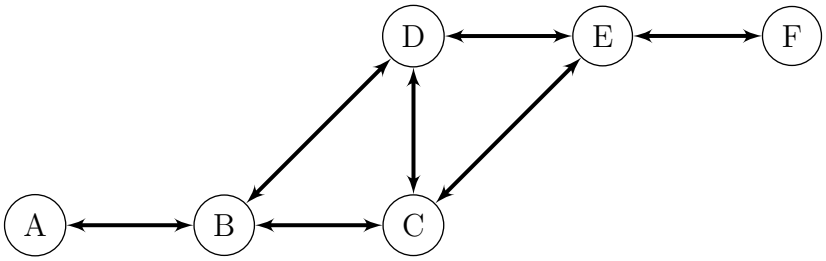

Figure 2.1: Example mesh network

When using a mesh network node A has the possibility to to ask node B if it can forward its data to node D. Node D will respond to node B which then in turn forwards the response to node A. By using this concept, node A is able to talk to all participants of the network, even though only node B is in direct communication range. So the spatial range problem is solved by an improved utilization of the available resources and redistributing the transmission task to other participants in the network. There are a lot of possibilities and ideas on how to establish these routes, which will be discussed in later sections.

2.1.3 Embedded devices

Embedded devices are application specific computers which are part of a larger system or device. In order for the device to fulfill its purpose the components of the system often need to communicate with each other. An ongoing trend is to embed wireless transmission devices into larger systems to form a network of embedded devices. Systems designed with embedded devices do offer a lot of benefits:

- **Low unit cost**
  Microcontroller units (MCUs) are very generic and can solve a wide range of problems. Therefore they are built in large quantities which reduces the production cost of a single MCU significantly.
- **Flexibility**
  Software written for a MCU architecture can usually be replaced easily by another one. This comes in handy when fixing bugs or introducing new features. Also, if one embedded device is faulty it can be easily replaced.
- **Small size and low weight**
  Because embedded devices often consist merely of some input and output pins, a power supply and a network interface they are very small and lightweight and can be used in space constrained constructions.

A lot of the time there are special constraints which limit the designer's options, as described in [CD11, p. 1].

- **Power and energy**
  Embedded devices often operate in places where a power supply is not available or only available with a significant financial investment. Sometimes they are even powered by battery or solar cells. Especially battery supplied embedded devices need to reduce their power consumption as much as possible, because it is inconvenient for the user to change the batteries frequently and empty batteries also have environmental shortcomings.
- **Size and weight**
  Mobile application are becoming exceedingly important. The smaller and less weight a device has, the better is its mobility. This is also very important in automotive systems, as computing in cars is often very space constrained and the less weight a car has the better it is for acceleration, braking and fuel efficiency.

- **Environmental**
  The environment in which embedded devices are used may vary vastly. There can be extreme temperatures, water, strong vibration, or even spark plugs which generate radio frequency that can interfere with the electronics.

2.1.4 History and Applications

The first concepts for creating a technical device for communicating wirelessly over larger distances were smoke signals, drums and beacons. While they did use very primitive communication channels, the encoding of information is in some ways similar to modern concepts. Most of the time these prehistoric methods were used to signalize an upcoming dangerous situation.

The next step in wireless communication was the Photophone, invented by Alexander Graham Bell and Charles Sumner Tainter in 1880. In Alaxander Graham Bells journal article [Bel80, p. 404] it is described that modulated light was used, in contrast to regular cable telephones, to transfer the analogous sound information wirelessly. Incremental improvements to the concept were made in order to gain more wide spread adoption, but the advantages of long distance radio transmission eventually took over the wireless communication.

The beginning of the 20$^{\text{th}}$ century marks the discovery of electromagnetic radiation as a communication medium. It has first been formally described by James Clerk Maxwell and has been attested by Heinrich Rudolf Hertz. Ever since it has gained attraction as it allows communication without necessarily having line of sight between the communication partners. In the early 20$^{\text{th}}$ century it was used primarily for analog audio transmission. But as time passed it has gotten more widespread adoption in audio and video broadcasting, (mobile) telephony, two-way radios, alarm systems, home and commercial building automation and many more. The beginning of the 21$^{\text{st}}$ century is characterized by the invention and wide spread adoption of the internet. As described in report of Adam Lella [Lel14, p. 4] around 60 percent of the people in the United States of America (U.S.) are accessing the internet with their mobile phone or tablet, which are most commonly connected wirelessly. Also, a lot of laptops and desktop computers are connected via Wireless Local Area Network (WLAN), so it is safe to say that most of the communication of computer end users is happening without the use of cables.

2.2 Networks and routing

The term routing describes the process of selecting paths in a network. A network is a collection of interconnected nodes, which is more theoretically explained in Section 2.2.1. In most cases the ideal route from one node in the network to another is as cheap as possible, by the means of time it takes until the node is reached and reliability in reaching it. There are numerous concepts for routing data in a network, most of them are either link-state or distance-vector protocols which discussed in Section 2.2.2 and 2.2.3 respectively. Some protocols share some principles of both worlds and are therefore called hybrid protocols.

2.2.1 Graph theory

Mathematically speaking, a network can be described as a graph. A Graph G is an ordered pair $G = (V, E)$ comprised of vertices V and edges E. Most commonly, one edge is related to two vertices. There are a multitude of different properties which graphs can have. All the properties which are used throughout this thesis will be explained here.

- **Weighted**
  The edges or vertices of a weighted graph are given a value which is called weight. For example, the weight of an edge in a network can describe its average latency.

- **Directed or undirected**
  In a directed graph the edges between vertices V_1 and V_2 are only allowed to have one specific order (Figure 2.2b). Edges in undirected graphs do not differentiate between the possible orders of the associated vertices (Figure 2.2a).

(a) Undirected graph (b) Directed graph

Figure 2.2: Directed and undirected graphs

- **Simple**
  Simple graphs do not allow loops, which means that an edge E is not allowed to connect from and to a single vertex V_1. Also multiple edges E_1 and E_2 are not allowed between two vertices V_1 and V_2.

- **Acyclic**
  A graph is acyclic if there is no route from one vertex V_1 over one or more other vertices $\{V_1, \dots V_N\}$ which leads back to the vertex V_1 itself.

2.2.2 Distance-vector routing

Every node in a routing protocol of the distance-vector family limits its knowledge about the network to the distance to possible destinations and directions which have to be taken to reach them. So a routing table entry for a specific destination node contains the fields *route*, *cost* (sometimes called *distance*) and *direction*. The route entry in the lookup table is used to determine where the packet needs to be sent in order to be delivered successfully. Once the destination address has been found the corresponding direction does give information about where to send the packet next in order for it to be delivered eventually. There are numerous different ways in which the cost can be interpreted, as described in [Lu+03, p. 2]. Reaching from the number of hops to the latency of links along the path or other metrics, all can be interpreted as cost, depending on the application. It does not matter if such a packet originates from the node itself or has been received by a secondary node. Figure 2.3 shows an example of two routers, A and B with the addresses 3 and 25 respecctively. Router A's table shows that it has knowledge about the distance to router B which can be reached directly (1 hop). The table entry also shows that there is a third destination C with address 14 which A can also directly reach. But if router B wants to send a message to C it knows that it has to send the message towards A and that A will have to forward it.

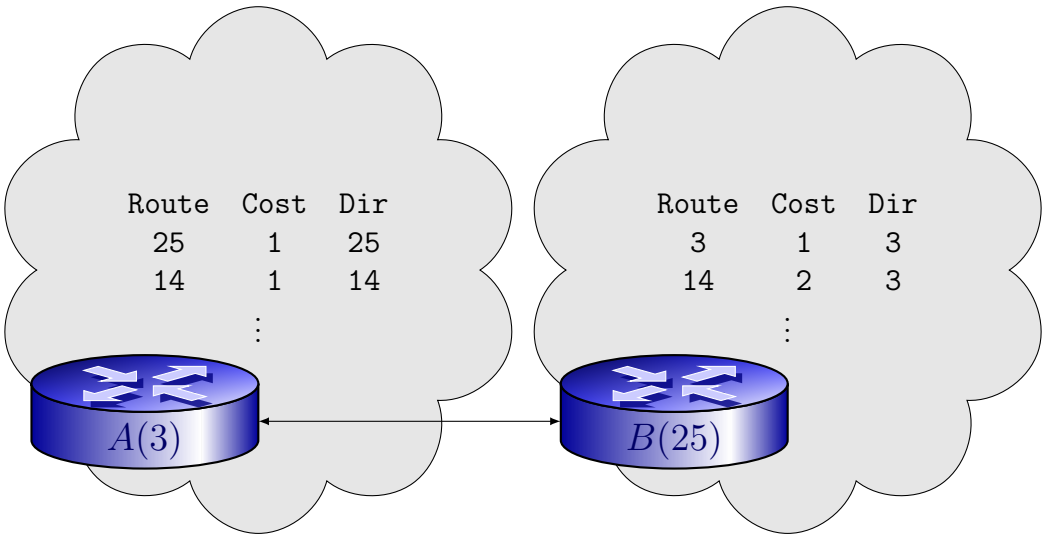

Figure 2.3: Distance-vector topology visualisation

2.2.3 Link-state routing

Contrary to the minimalistic approach of distance-vector routing protocols link-state protocols have more overhead when propagating routing information in the network. The reason for this is that every node requires complete information about the topology of the network. Links between nodes are weighted and propagated throughout the complete network as described in [AK06, p. 1]. So, every node knows all possible routes and can therefore calculate the optimal route to a specific destination. A problem this approach faces is the enormous amount of network traffic required to process the Link State Advertisement (LSA) when there are lots of nodes in the network. Figure 2.4 shows an example router A with address 3 that stores the route to every accessible network participant. It also remembers the link weights in order to calculate good paths. A similar algorithm as described in 3.6.2 is needed in order to calculate optimal paths to routes.

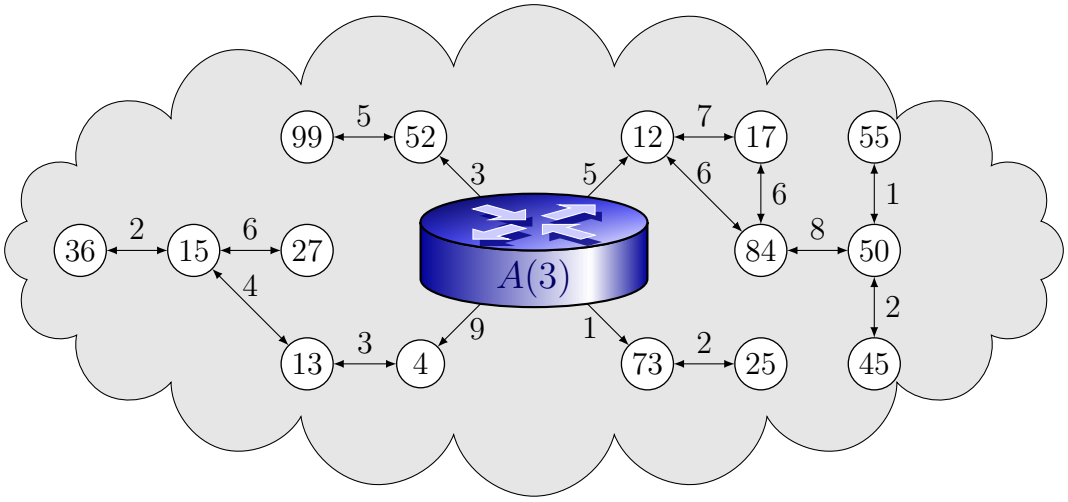

Figure 2.4: Link-state topology visualisation

2.3 Modern mesh network designs

Focusing on the topic at hand, there are already some implementations for routing in wireless networks for low power embedded devices. This section offers a short summary of the most important specification details of the more popular ones in order to have a good understanding of some common approaches.

2.3.1 RPL

The RPL is specified by the Internet Engineering Task Force (IETF) and conceptually follows a distance-vector approach. It extends the Internet Protocol version 6 (IPv6) specification to be the standard routing protocol for Low Power and Lossy Networks (LLNs). RPLs objective, according to [CHP11, pp. 365 sq.], is to support a network comprised of thousands of nodes with constrained resources, while being managed by few central root nodes. Figure 2.5 shows a Destination Oriented Directed Acyclic Graph (DODAG), wich is the basic theoretic construct which RPL relies upon. Node A represents a root node. In a converged LLN each node has identified a list of parents, including one preferred parent (indicated by P), which are potential next-hops towards a root node. In order to propagate routing information throughout the network each node emits DODAG Information Object (DIO) messages. These messages indicate the respective rank of the node in the network, which is some kind of metric explaining the distance to a root node (e.g. hop count).

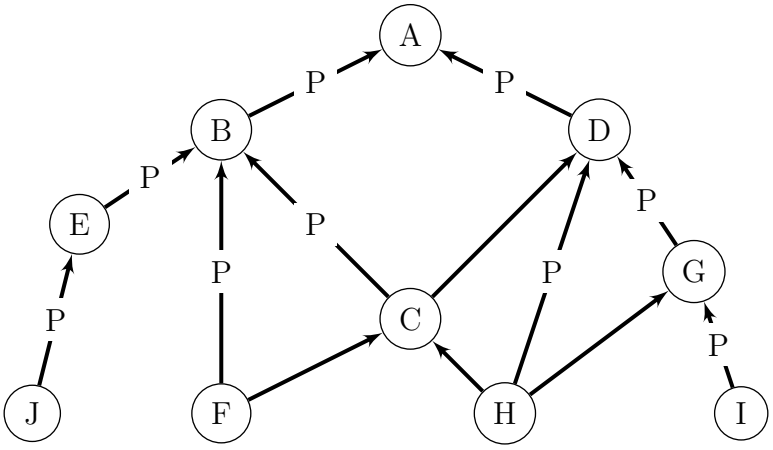

Figure 2.5: DODAG, where P indicates the preferred parent

The root node initiates the DODAG formation, as it is initially the only node participating, and spreads the information to gradually cover the whole LLN. Nodes send out DIOs and choose their parents and preferred parents. Once the DODAG creation process has been finished the network is optimized for multipoint to point routes, as all the nodes know a way to the root node. This is also called *upward routing* as the communication using this technique can only happen from standard nodes to the root node. In order to achieve a communication from the root node to one of the other nodes (*downward routing*) RPL uses so called Destination Advertisement Object (DAO) messages. These messages describe which prefix belongs to and can be reached via which Node in the network. All nodes in the network must work in either of two modes of operation

- **Non-storing mode**
  The DAOs messages are created by a RPL router and advertise information about one or more of its parents. This information is then unicast to a DODAG root. Once the root node has collected information about all nodes between itself and the desired destination it can use source routing for reaching destinations inside the LLN. So here the root is the only network participant which is storing routes to destinations in the LLN.

- **Storing mode**
  Here every node in the path between the root node and the originator of the DAO stores a route to the prefixes advertised in the DAO and the next hop towards these prefixes. Using this method a node can maintain its routes to all destinations in its created sub-DODAG. When this operation is complete, the node forwards the DAO to its preferred parent.

Trickle

RPL nodes generate messages to update their routes based on timers and on the amount of communication done by their neighbors. This procedure is called Trickle. The root node is able to configure the back-off intervals which define when timers are triggered and DIO messages are sent. Trickle's goal is to minimize the duration it takes for the network to converge, which means that there are as little updates via DIOs as possible and every node in the network has correct and up to date information about its parents. In order to reach this state the amount of DIO traffic a node recognizes about its neighbors and a back-off time given by the root node are essential, as it could be redundant for a node to send out its DIO if all neighbors already know the necessary information.

While RPL is generally a very promising standard for LLNs there are some scenarios in which it does not perform optimal. Especially the use of DAO messages for bi-directional and point to point traffic which has to be sent through the root node can be detrimental for certain applications.

ZigBee

The most popular use of RPL is within one of the currently three available ZigBee specifications, namely ZigBee IP. (Excerpt from [All14, p. 6]). ZigBee is being created and maintained by the ZigBee Alliance which is a collaboration of over 230 companies. It can be seen as an extension of the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 protocol that widens its functionality by routing and secure key exchange. Figure 2.6 gives an overview of its used layers

which are mostly standardized by IEEE and IETF. Design-wise it is closely related to the internet protocol. The core differences are the use of IPv6 over Low power Wireless Personal Area Networks (6LoWPAN), a lightweight IPv6 variant, as well as the use of RPL. A lot of options are available in the higher layers like encryption using Transport Layer Security (TLS), automatic service discovery using Multicast Domain Name System (mDNS) or Domain Name System - Service Discovery (DNS-SD) and many more.

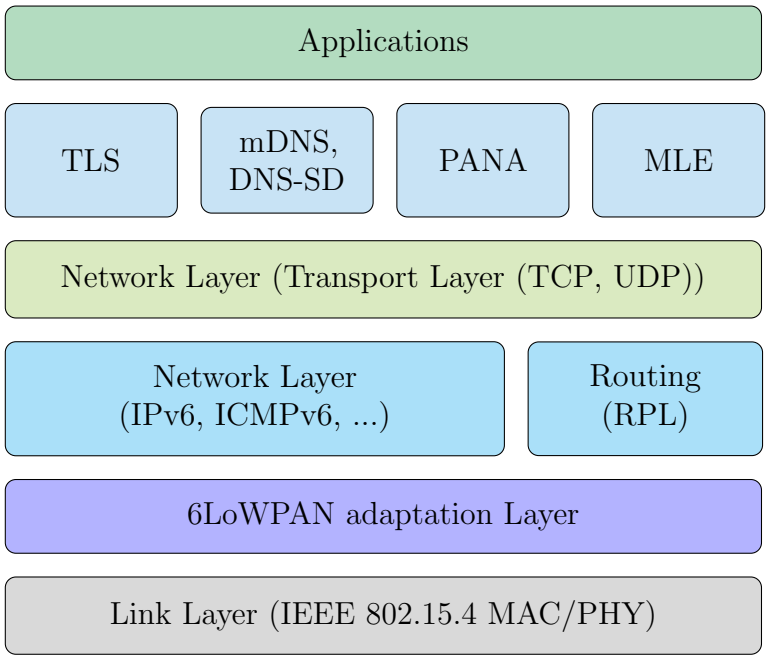

Figure 2.6: ZigBee IP protocol layers

2.3.2 Z-Wave

A popular american standard for home automation is Sigma Design's Z-Wave protocol. For a very long time only paid premium members could access the official documentation but as of recently the specification has been partially opened up. Unlike ZigBee, Z-Wave's network stack has a more lightweight design. In order to uniquely identify a node within Z-Wave there are two separate IDs:

- **Home ID**
  Common identification scheme of all nodes that belong to a logical Z-Wave network. It is 4 bytes in size which allows for $2^{4 \cdot 8} = 4294967296$ distinct networks.

- **Node ID**
  The address of a single node within a network. It has a size of 1 byte which allows for $2^8 = 256$ members corresponding to a Home ID.

Z-Wave knows two different kinds of network participants: *Controllers* and *Slaves*. A controller's job is to deliver orders to slaves. Controllers are provided with unique Home ID built in as factory defaults. Slaves are given their Node ID from a controller. The routing strategy of Z-Wave follows a source routing approach with up to 5 hops within a network. In case no route to a node is found, an exploration frame can be sent which is propagated throughout the network as a last resort to check whether the desired device can be found. A single network can support multiple controllers, which can be statically placed or embedded within portable remote controls. Using controllers within remote controls, however, does not guarantee that all nodes can be reached due to possible attenuations caused by surrounding objects.

2.3.3 LOADng and related concepts

LLN On-Demand Ad hoc Distance-vector - next generation (LOADng) is building upon the foundations of Ad Hoc on demand Distance Vector (AODV). Therefore, AODV will be explained beforehand.

AODV

AODV is a highly popular ad-hoc network specification based on Distance-vector routing (Section 2.2.2) formulated in 1999. As described in [PR99, p. 2] it is a pure on-demand route acquisition system, which means that nodes which are not on active paths in the network do not participate in any periodic routing exchanges. Every node maintains a *Node Sequence Number* and a *Broadcast ID*. Also, it does not maintain any routes to other nodes until the need to communicate arises. When a node is in need of communication and it does not know any routing information to the desired destination, a Route Request (RREQ) is broadcasted. A RREQ consists of the following fields:

- Source address
- Source sequence number
- Broadcast ID
- Destination address
- Destination sequence number
- Hop count

The pair (*Source Addreess*, *Broadcast ID*) uniquely identifies a RREQ. Local neighbors either satisfy the RREQ and respond with a Route Reply (RREP) or rebroadcast the RREQ to its own local neighbors after increasing the *Hop*

count. Nodes which receive the same RREQ have to check if the *Broadcast ID* in combination with the *Source Address* was used recently. If this is the case, the RREQ is dropped in order to avoid redundant RREQs. There are two mechanisms in order to assure that a communication channel is eventually found between two nodes in a network:

- **Reverse Path Formation**
  As a RREQ travels from a specific source to various destinations in order to find the desired *Destination Address* it automatically determines the reverse path from all nodes back to the source, which are stored by each node separately. The reverse path route entries are stored long enough for a RREQ to traverse through the network and send a reply to the originator of a message.

- **Forward Path Formation**
  The RREQ will eventually arrive at either the destination directly or at another intermediate node which knows a up to date route to the destination. In the latter case, in order to figure out if the route to the destination is current, the intermediate node compares its stored *Destination sequence number* with the one from the RREQ. If the stored *Destination sequence number* from the intermediate node is smaller than the one from the RREQ the node must not use its recorded route to respond to the RREQ. Instead it has to continue to broadcast the RREQ as if it had no idea about the route. Otherwise it unicasts a RREP back to the sender from which it received the RREQ.

Nodes which are not on the Forward path will timeout eventually and delete their reverse route. If there are multiple routes, the RREPs will be triggered again if a route has a smaller *Hop count*. This way a communication channel to an arbitrary node in the network can be established.

LOADng

AODV is a rather successful and mature protocol which is widely used, but it is still particularly performance intensive especially for the case of more recent trends like low power embedded mesh networks. In an effort to modernize AODV the LOADng specification was developed, as mentioned in [CYV12, p. 2]. The main area of improvement is the better support of more constrained environments, by the means of computational power and energy consumption. LOADng inherits the basic properties of AODV, like RREQs for discovering a route, RREPs when the desired destination of a RREQ has been detected and unicast forwarding of RREPs

to the originator (Subsection 2.3.3). It can be seen as a set of extensions to and simplifications of the older AODV approach. The extensions include:

- **Modularity**
  LOADng has a flexible packet format which permits the addition of arbitrary attributes and information via Type-Length-Value (TLV) fields.
- **Optimized flooding**
  This reduces the overhead incured by RREQ forwarding. Jitter reduces the probability of packet loss due to collision on lower layers.
- **Variable Address lengths**
  The length of addresses in LOADng are supported to be between 1 and 16 octets. Requirement for this to work is that within a given routing domain all addresses are of the same length.
- **Metrics**
  Link information can be better represented due to a variety of available metrics.

The simplifications include:

- **Disallow intermediate RREPs**
  The destination node is exclusively allowed to answer a RREQ, all other nodes are prohibited to do so. All messages by a given node share a single, unique and increasing sequence number. This simplification is reasoned with reduced complexity of protocol operation and smaller message sizes. The paper [CYV12, p. 3] states that this simplification come without significant influence on performance.

Other improvements of LOADng over AODV include the maintenance of routes. If a route failure is detected, for example when a specific node cannot be reached anymore, a Route Error (RERR) message is sent as a unicast along the route to the source of the packet. In AODV, all neighbor nodes contained in the precursor list are informed about the route failure, which causes a lot of transmission overhead. So overall, LOADng can be seen as a simplified and gentrified version of AODV with special focus on constrained environments.

2.4 Popular operating systems for IoT

In order to achieve networking on any device it is advisable to make use of already existing operating systems largely reducing implementation expenses. There is a wide variety of systems tailored at diverse use cases. The focus of this section is on operating systems for networked, memory constrained, low power wireless Internet of Things (IoT) devices. A lot of the available operating systems fall short of this requirement, but there is still a hand full of choices.

2.4.1 Contiki OS

Contiki Operating System (OS) is a popular operating system for these kind of embedded environments. It is also the chosen base operating system for the reference implementation of the routing protocol this thesis introduces, as described in Chapter 4. This fact qualifies a closer look into the internals of this operating system.

Contiki is an open source, portable, multi-tasking friendly, networked embedded operating system with a very low memory footprint. As [PK09, p. 1] states, a typical configurations of Contiki uses about 2 kb of RAM and 40 kb of ROM. It features an event driven kernel, where processes can be dynamically loaded and unloaded at run time. The programmer has the choice of different network protocols, which is chosen at compile time. These choices include IPv6 using 6LoWPAN, IPv4, and the Contiki specific Rime stack.

Process

In Contiki a process is basically a composition of the process control block and the process thread. The control block, as shown in Listing 2.1, which is stored in Random Access Memory (RAM) only, provides information about a process, such as the process name, a reference to its thread and its internal state. It is designed to be only for internal use and should never be directly accessed by a process. Contiki is using a linked list of processes in order to schedule them. This is why there is a reference to the next process in the definition of a process.

Process thread

A process thread contains a pointer to the code which shall be executed. It can be seen as a single protothread which is invoked by the scheduler. Listing 2.2 shows a

```
struct process {
   struct process* next;
   const char* name;
   int (* thread)(struct pt*,
                  process_event_t,
                  process_data_t);
   struct pt pt;
   unsigned char state, needspoll;
};
```

Listing 2.1: Contiki process control block

bare bone implementation of a process thread. The actual code which is executed is encapsulated by the `PROCESS_BEGIN();` and `PROCESS_END();` macro keywords.

Protothread

Multitasking is a core requirement of modern operating systems. Most of them use a process and thread model, where a process is started with its own context and can have one or more threads which all share this context. One downside to this approach is that every process needs to have its own context and stack, which implies a rather large memory overhead when used in constrained environments. Contiki makes use of the concept of protothreads in order to achieve a lower memory footprint while still providing an environment which is similar to threads. protothreads share a common memory stack in RAM, which is contrary to usual multithreading where each thread provides its own stack. Another design principle which is defining protothreads is an event-driven model for context switching. Also, protothreads can be advised to wait for certain events (see Section 2.4.1) before continuing their execution.

Events

Events are constructs which inform processes that a certain circumstance has occured and can be posted synchronously or asynchronously to an event queue. There are numerous predefined events like `timer`, `poll`, `continue` and `message` events. Listing 2.2 shows a Contiki process named `example_process` with a protothread which is defined in the closure of the `PROCESS_THREAD(...)` body brackets. `AUTOSTART_PROCESSES(&example_process)` is a macro function that instructs Contiki to automatically start the protothread corresponding to the process. The `PROCESS_WAIT_EVENT()` macro suspends (schedules another process) the protothread until any event occurs. There are also other macros where you can specify event IDs to wait for.

```
#include "contiki.h"

PROCESS(example_process, "Example process");
AUTOSTART_PROCESSES(&example_process);

PROCESS_THREAD(example_process, ev, data)
{
  PROCESS_BEGIN();

  while(1) {
    PROCESS_WAIT_EVENT();
    printf("Got event number %d\n", ev);
  }

  PROCESS_END();
}
```

Listing 2.2: Example Contiki process waiting for an event

Network stack

As networking is an important part for many embedded device applications, Contiki provides a modular network stack. Figure 2.7 provides a high level overview of the five layers that form the stack. There are several implementation options to choose from for every layer.

A short description of the steps to take for an outgoing packet in order to be sent by the radio module:

- **Network Driver**
  This is the stage which defines the addressing scheme. It contains higher level operations such as routing, providing a callback to the application to inform about received data or preparing the packetbuf for a new outgoing packet. The possible options which Contiki provides for this layer are IPv6 using 6LoWPAN, the modular and extensible Rime stack and there also exists an Internet Protocol version 4 (IPv4) implementation. Pikrit is the network driver which will be introduced in this thesis in Chapter 3.

- **Link-Layer Security**
  More Recent versions of Contiki provide a separate security layer in the stack that is responsible for the encryption of sent data, as well as for the avoidance of possible replay attacks. While some radio modules already provide built-in encryption, not all of them do. So this layer provides a more modular approach to security within Contiki. Encryption is, as of the time of this writing, a relatively new addition to Contiki, so there are not many implementation options. Besides the nullsec implementation, which

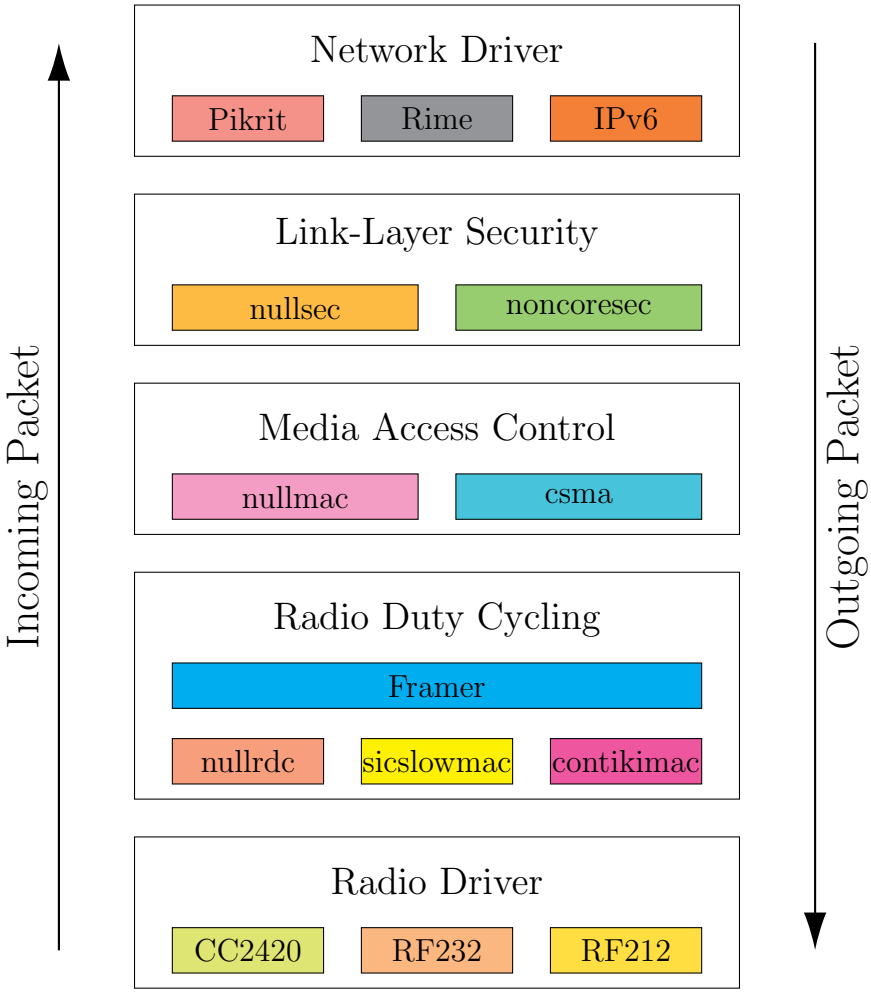

Figure 2.7: Overview of the Contiki network stack

does nothing, there is also the noncoresec implementation. This is a security implementation, standardized by IEEE 802.15.4, which uses a network-wide key, that has to be distributed to every network participant in a secure manner.

- **Media Access Control (MAC)**
  A problem which frequently occurs when sending data, especially in a wireless medium, is if two network participants send out a packet at the same time, which creates a so called collision. This corrupts the transmitted data of both senders. The MAC-layer is dedicated to avoiding such collisions as good as possible. There is no guarantee in avoiding collisions, as a sender can never know when another network participant sends out data. Contiki provides a generic Carrier Sense Multiple Access (CSMA) implementation as well as the nullmac driver which does nothing. CSMA checks the medium before sending packets. If the medium is busy at this moment it backs off the sending.

- **Radio Duty Cycling (RDC)**
  Many embedded devices are using batteries as their power source, so it is critical to save as much power as possible to achieve a long network lifetime. Just using power efficient network hardware is often not enough, as it is usually the most power consuming part of the device. In order to improve the situation the operating system should disable the network hardware as much as possible. While the network hardware is disabled, no data packets can be transmitted or received. The network hardware is enabled periodically to sense incoming packets. If data transfer is detected, the network hardware is kept on for a larger amount of time in order to receive consecutive packets. Some RDC implementations also synchronize their wake-up frequencies in order to increase the chance of receiving a packet. For outgoing packets the hardware is enabled on demand. Contiki provides a few implementations for RDC, such as nullrdc, where the duty cycling is disabled, sicslowmac which is the default RDC mechanism for 6LoWPAN and contikimac, the default contiki RDC implementation. Beside the actual duty cycle this layer is also responsible for delivering the data to the so called framer, which is an optional part in Contiki and handles the parsing and filling of the packetbuf. Each network stack can define its own modular framer (like IPv6), or handle the filling and parsing of the packetbuf themselves.

- **Radio Driver**
  Contiki provides an interface with a fixed set of functions which should be supported by the network hardware. The implementation of this interface for different radio hardware is a radio driver. A lot of drivers for different radio hardware is already provided by Contiki itself but it is also modular enough to provide own implementations for radio drivers.

Packetbuf

The whole stack is centered around the concept of a unique packet buffer which is internally called “packetbuf”. In order to save memory, all layers of the stack operate on this buffer, as stated in [RS13, pp. 6 sq.]. This design approach has some inherent disadvantages such as potential packet loss when accessing the buffer while a packet arrives or the disability to properly handle queues.

Besides the actual buffer which stores and manages the header and data contents of the payload the packet buffer also provides an interface for storing attributes and addresses. Both of them are handled with key-value pairs which are defined at compile time. Keys are defined in an enum and values are of the type `packetbuf_attr_t` which is a two byte `unsigned int`. They are used in order to attach information to the package which can be read by different layers of the stack.

Addresses are defined within the same enum from the bottom to the top. Their values are of the type `linkaddr_t` which are explained in detail in Section 4.3.2. Contiki's Rime stack uses the additional addresses extensively to store multiple addresses, for example recepient address, receiver address or additional addresses within mesh networks.

2.4.2 RIOT

RIOT is a more recent alternative operating system designed for Wireless Sensor Networks (WSNs). It aims at bridging the gap between operating systems designed for WSNs and traditional, full-fledged ones, as stated in [Bac+13, p. 2]. Historically, RIOT is based upon FireKernel from which it inherited its modular microkernel architecture which allows the use of standard multi threading with standard Application Programming Interfaces (APIs). Advanced features like real time scheduling and the support for C++ are included while having a minimum RAM usage of about 1.5 kB and a minimum Read Only Memory (ROM) usage of about 5 kB. Since RIOT is rather young (its rebranding happened in 2013) the support for newer MCUs with 16- or 32-bit architectures is great, but the support for older 8-bit architectures is lacking. Using the Lesser GNU General Public License (LGPL) it also advertises itself as a free and open operating system where everybody is invited to contribute to.

2.4.3 Zephyr

Another alternative is the Zephyr kernel which is backed by the Linux foundation and tailored at wearable and IoT devices. A special concept of Zephyr is the use of a microkernel with underlying nanokernel. Application developers have the choice to use either only the nanokernel or both, the micro and nanokernel. The former is a high-performance execution environment, uses only a very limited amount of RAM and is capable of scheduling multiple threads. On top of that, the optional microkernel adds supplementary functionality like operating on multiple tasks which is not possible with the nanokernel only.

Zephyr differentiates between tasks and fibers. Tasks should be used to perform lengthy and complex calculation operations, while fibers are designed to be light-weight and should be used for device drivers or performance critical work. Fibers do support priority based multithreading and are generally preferred over tasks. Zephyr is free and open source software and uses the Apache License 2.0.

2.4.4 TinyOS

TinyOS is an operating system for embedded devices with a history dating back to the 1999s. Since then it has gotten quite popular. It features nesC, a dialect of C which is specifically customized for building component based and event driven programs as stated in [McI09]. The philosophy of TinyOS is fueled by the idea of components which interact with each other and with the underlying base system. Programs are a composition of modules and configurations. Modules are for implementing specific functionality and configurations provide a connection between different components. In nesC each component has one or more interfaces which are composed of commands and events. Commands are used for requesting that a component should perform some service. Events can have two meanings, either it informs a component that an activity has been completed, or that an external phenomenon has occurred, like an interrupt. A component can either be an interface provider or an interface user and has to implement the required functionality for this interface. The configuration component then wires interface providers with interface users. A simple task scheduler maintains a First In First Out (FIFO) queue which is executed if the processor is not occupied (e.g. by an interrupt). Synchronous tasks in TinyOS run in a non-preemtive manner that run tasks until completion, therefore tasks are atomic in respect to each other. Using interrupts can trigger unsynchronous events, which is why nesC also provides atomic code blocks. For this reason it can be difficult to make TinyOS applications reliable, because developers using multiple tasks with interrupts must consider all possible interleavings of concurrent activities. A recommended way of testing the application before shipping is either via using a simulator, which is covered in Section 2.5, or via actual testing laboratories.

2.5 Simulators for Wireless Sensor Networks

Real world scenarios of Wireless Sensor Networks (WSNs) are often complex and difficult to test. There are a lot of different circumstances which can occur. In building automation there can be thick ferroconcrete walls degrading the signal quality of wireless actors, or large distances between actors within large buildings. Other areas have to deal with problems like moving actors or sensors which frequently leave and enter the communication range. It would be a huge and costly effort to build a real world testing area to test all real world problems. This is why many embedded operating system developers also provide simulation tools for their product. The following sections discuss a few popular embedded IoT simulators and shed some light on their internal workings.

2.5.1 Cooja

Within the Contiki project there are a few useful tools that help with simulating complex scenarios. Most of these tools are centered around the Cooja network simulator, which has been developed as part of Contiki but can be run independent of Contiki and therefore allows for broader use as described in [RSZ16, p. 2]. This means that also RIOT binaries can effortlessly be simulated in Cooja. It provides a wide range of functionality for simulating WSNs. Figure 2.8 shows the network plugin, where nodes can be placed in a two dimensional graphical plane. In this specific case node 9 is sending out a message, and nodes {8,10,11,12} are receiving it, as they are in transfer range. The transfer ranges for node 2 are displayed as the green (transfer range) and gray (interference range) circles centered around it. Configuration options like disturbers and a different radio mediums with corresponding random seeds do exist in order to simulate a realistic radio medium.

At its core Cooja is providing a graphical user interface, simulation of a radio medium and an extensible framework, with which it can be extended to simulate very different scenarios. The simulators are their own products and are integrated as plugins within the Cooja simulator. There are currently two simulators available, Avrora and MSPSim.

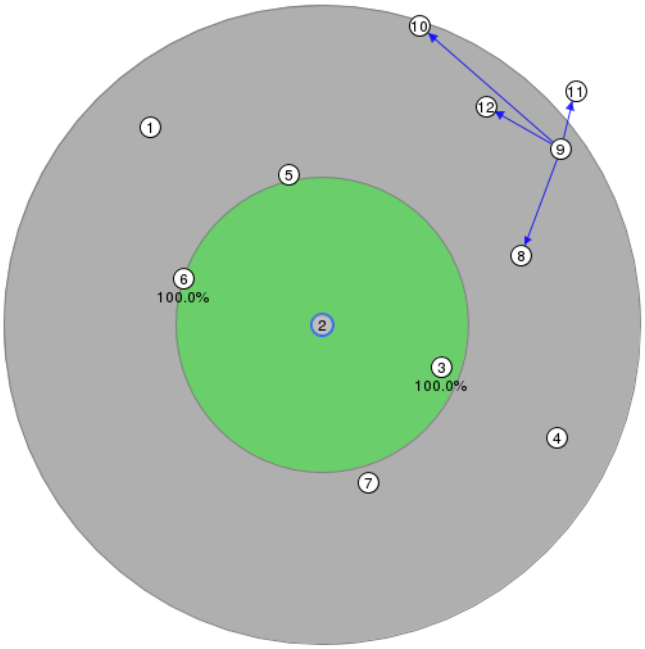

Figure 2.8: Cooja network visualisation

As a more advanced feature Cooja has a built-in javascript editor that can directly affect simulation. This is very handy when trying to create complex simulation scenarios with specific events happening at defined times. For this thesis such scripts have also been used in order to compare different protocols by automatically sending

frames over certain routes and measuring the response time. After measuring mean values and standard deviation were also calculated using javascript.

Avrora

Responsible for handling the simulation of the AVR-MCU families produced by Atmel and Micaz is the Avrora Cooja interface. Avrora provides a java interface for experimentation, profiling and analysis without having to flash a real MCU. This greatly increases productivity when developing for certain MCUs. Its feature set includes, but is not limited to, the following properties:

- **Simulation**
  AVR-MCU programs can be simulated with cycle-accurate execution times.
- **Monitoring**
  In order to better understand the execution behavior of the program and to detect possible performance bottlenecks the provided monitoring infrastructure can be utilized.
- **Debugging**
  The GNU Project Debugger (GDB) tool can be used to stop the execution of a program at runtime, monitor its state and step by step browse through the execution routine. Error-prone behavior in programs can easily be located using this method.

MSPSim

The second simulation interface for Cooja is the MSPSim emulator for the MSP430 microprocessor series. Suitable firmware files can be loaded directly and its feature set also includes simulation, monitoring and debugging as the Avrora simulator.

2.5.2 TOSSIM

TinyOS (see Section 2.4.4) is providing a simulator called TOSSIM. It provides, as discussed in [Asi+09, p. 2], a discrete event queue where simulated events can be placed in a specific order. Ticks are used in order to emulate the passing of time within TOSSIM and those can be modified as needed. Python and C++ can be used as programming interfaces in order to control the simulation dynamically. In order to debug an error-prone program, debugging messages have to be written in the

original source code as there is no debugging functionality. A downside compared to Cooja is that executable files have to be compiled separately for simulation and that the simulator currently only supports the micaz platform. Without the usage of plugins the simulation is text-based only and described using python in combination with a topology text file. Fake sensor data can be input as text files which can then be periodically read out and manual insertion of packets in the network traffic is also possible. A large downside of using TOSSIM is that all nodes are required to run with the exact same executables.

Chapter 3

Design

"Strive not to be a success, but rather to be of value."[1]

3.1 Requirements

The application domain for connected devices is growing day by day. Applications which previously did not have any kind of networking in mind are now continuously integrated within the world wide web. It is difficult to find one specification which fits all needs perfectly as the requirements as such networks differ substantially. In order to provide a network specification which is well suited for the building automation use case, this chapter focuses on determining and defining the properties of components used within this application domain.

3.1.1 Network sizes

The largest computer network on earth is the internet, often called the "network of networks". It is a connection of a multitude of Wide Area Networks (WANs) and uses the internet protocol suite to link billions of devices worldwide. Those WANs can also be separated into smaller subdivisions like Metropolitan Area Networks (MANs), Local Area Networks (LANs) or Personal Area Networks (PANs). Each of these network size categories differ in requirements. The desired protocol shall handle data transmission wirelessly over rather small areas, like the area occupied by single-family houses, hotels or small companies. Therefore it can be categorized as a solution for LAN or PAN sized networks.

[1] Albert Einstein

3.1.2 Conflicting intercorporate networking standards

At the time of this writing manufacturers of different connected automation products do not generally share a common networking standard. This situation is not ideal as there needs to eventually be a standard to let products communicate with each other in order to provide unified and simple user experience. But until a common standard is defined which is approved by most manufacturers, new protocol specifications will arise and existing specifications will be improved. All these protocols have their respective strengths and weaknesses, depending on their use case. It would be convenient to use an already defined standard but the requirements regarding transmission performance were not met when testing diverse existing protocols. As there is nearly no compatibility value to be gained in using an already existing protocol the decision was made to design a new protocol with this thesis.

3.1.3 Internet gateways

Accessing the actors and sensors from outside the own network happens with special gateways, which routes traffic from the internet to the WSN and vice versa. Complementary to the conflicting networking standards is also the fact that manufacturers often produce their own gateway which is only able to support their chosen standard. Some projects are providing a multi protocol gateway where the needed wireless transceivers can be added or removed as necessary. Conceptually this sounds like a good solution but practically the setup and maintenance of such devices can be cumbersome. Also problematic for this approach is that the different gateways expect the information processing in different locations. There are two solutions:

- **Cloud processes information**
  This approach keeps the gateway as minimalist as possible. It only transfers the information from the sensors and actors to a central server within the internet. The server then processes the information and performs the needed tasks. While the production cost of such a minimalist gateway are very low there are some disadvantages with this solution. Customers have to register themselves within a cloud platform and therefore have to offer the company some private information. If the user forgets its registration data he might not be able to control its own devices anymore. Offline operation is not possible, which implies that internet failure causes all sensors and actors to be inoperative, as they are very dependent on the online service.

- **Gateway processes information**
  The second method is to let the gateway do all the processing itself. Doing so needs a few additional components like storage and also a faster processor in

order to provide the needed functionality, which increases retail cost. However, there is no need to register to a service which better respects the users privacy. While this method doesn't have to have a central server it certainly can. Such server enables the possibility to access the system from outside the local network and can also be used to store configuration backups or provide automatic updates. In case the internet connection is terminated the system still remains fully functional within the local network. So it is the more expensive but therefore also more robust and flexible solution.

The desired protocol shall contain a special component called coordinator (See Section 3.2.2), which is expected to have a good computational performance. So it is advisable to combine the task of the gateway with the task of the coordinator in practical use which reduces complexity and cost.

3.1.4 Identifying properties of building automation components

First and foremost the protocol should be specifically designed for building automation. The variety of automation potential in buildings makes this topic rather difficult. In order to easily upgrade existing buildings the protocol is designed for wireless networks, as no wires have to be laid out subsequently. Analysing a chosen set of home automation products reveals some interesting information:

- **Intelligent shading**
  The goal is to optimally distribute the light in and around the building, depending on wind, outdoor light intensity, time and astronomical occurrences.

- **Climate control**
  Climate control shall give the user the possibility to always be informed about weather conditions and set the right temperature throughout the house.

- **Automatic light dimming**
  The building should be aware if there are people in or around the house to provide ideal lightning conditions and save power if no light is needed.

- **Security and access control**
  The building should be able to ask the owner for permission to let people enter the house and inform the police in case an intruder is detected.

- **Fire alarm sensors**
  Alarms the user and the fire department in case of a fire.

All these products have in common that they are statically bound to their location once they are mounted. This means that it is not important to focus on moving objects when thinking about a protocol for building automation. It is not expected that network participants frequently change their neighbors, which is a core information when designing the specification.

3.1.5 Confidentiality

The topic of this thesis does not directly cover message encryption, as this is a rather large topic in itself. Data privacy and security, however, plays an increasingly important role throughout all network specifications. So confidentiality, authenticity and integrity are very important requirements when designing a network specification and therefore should be discussed. There are several things to consider when implementing an encryption mechanism within an embedded mesh network.

Encryption algorithm

A huge variety of algorithms have been developed to keep networks secure, but all algorithms can be categorized as either symmetrical or asymmetrical. Symmetrical algorithms are much faster but the key exchange is a major point of vulnerability because the key has to be delivered safely to the recipient before any communication can occur. Asymmetrical algorithms provide a secure key exchange method. There are two keys, one public and one private - the public key is used to encrypt- and the private key is used to decrypt data. The public key can easily be distributed as it is not necessary to hide it from possible burglars. One significant downside to asymmetric encryption is that longer messages are computationally very expensive to encrypt. Very common are hybrid algorithms which use asymmetrical encryption to exchange the key for the faster symmetrical encryption. Each of these encryption categories has a multitude of actual implementations with different mathematical concepts backing them.

Hardware or software encryption

Especially for hardware constrained environments, like embedded devices, software encryption can be very expensive and delay the actual operation significantly. So the choice might be to use special purpose integrated circuits which provide hardware accelerated encryption like Advanced Encryption Standard (AES) out of the box. This is a rather large commitment, as the encryption can only be changed by replacing hardware components which might be very costly. Many radio chip hardware manufacturers do however already include state of the art

encryption algorithms in their products. Software encryption, on the other hand has the advantage of being more flexible but is only feasible when using performant processors and optimized compilers.

Key distribution

Within WSNs the symmetrical approach of encryption is the most popular choice. Asymmetrical as well as hybrid approaches have the drawback of being computationally and message count-wise too expensive. The most prominent problem when using symmetrical approaches is key distribution, as the key has to be delivered to the participants in a secure way. But without using some kind of encryption key distribution is unsafe as an attacker could also gain knowledge of the key by sniffing the data traffic. Some companies approach the problem using a single master key for encryption, so the key does not have to be delivered to the participants at all as they already expect the master key. This method works as long as no untrusted third party has knowledge about the key. If someone somehow determines the master key, all encryption efforts are in vain as messages could be decrypted easily.

Key rotation and replay attack safety

Another problem of using a single master key are replay attacks. Simply recording and re-transmitting of messages could be used in order to gain gradual control over a system. One way of solving this problem could be to add systematic "salt" to the end of the message before encrypting, and the node only accepts messages with the correct salt number. An additional possibility would be to rotate the used key periodically. The same problems as within the initial key distribution apply here. Exchanging the keys, however, additionally increases security as replay attacks are even harder to execute.

An alternative to peridocially exchanging keys are one-time passwords. Lamport describes in [Shi+15, p. 3] an iterative one-time function approach for authenticity. Prover A choses a one way function h with a limit t and sends $h^t(k)$ to a verifier B. Here k is the chosen key and a one way hash function is applied t times to k as shown in equation (3.1). The i-th identification from A to B is then calculated using the function $y = h^{t-i}(k)$. B then varifies that $h(y) = h_{i-1}$ is true. If this is the case B accepts the message and stores y as the new $h_i = y$. This way the key is never actually exchanged while the verifier can still be assured that the prover does know the key.

$$h^t(k) = \underbrace{h(h(\cdots h(k)))}_{\text{t times}} \tag{3.1}$$

Resetting participants

Functionality-wise resetting the keys of participants should be possible, although only if physical access to the nodes exists. This is needed so that participants can still be reused in other systems. There are different ideas how to reset a node. A dedicated hardware button for resetting could be used. Other vendors sometimes also use the power supply for resetting nodes. In this case the key can be resetted within the first few minutes after the participant is powered on.

3.1.6 Data integrity

In order to ensure correct data delivery a Cyclic Redundancy Check (CRC) function should be used to assure data integrity. If corrupt data is accepted on nodes this can lead to incorrect behavior of the product and potentially, depending on the use case, also be dangerous to surrounding people. So every data frame should be verified assuring its contents integrity.

3.1.7 Platform independence

Protocol specifications should always be platform independent. This is of special importance if the design goals for the protocol include to be of value for different kinds of applications as well as being future proof. In Chapter 5 a reference platform is used for comparison purposes. This reference platform is as of this writing the most important platform for this protocol, as it might be the first real world use case, and should therefore be discussed here.

Reference implementation platform

The reference platform is the system ONYX which is developed by HELLA as described in Section 1.3.1. In more detail the ONYX.NODE, ONYX.WEATHER, ONYX.CONNECTOR and ONYX.CENTER hardware family is predestinated in being the first products to use this protocol specification. Hardware-wise these products use the components described in the following sections for wireless communication.

- **Main compontents**
  The heart of the mainboard is an Atmel ATmega 1284p microcontroller. This chip features 128 kb of flash memory, 16 kb RAM and an internal Electrically Erasable Programmable Read-Only Memory (EEPROM) with 4 kb. So the protocol has to be able to run on resource-constrained systems. In order to reliably provide over the air software updates an external EEPROM is used in combination with a bootloader section within the flash memory.

- **Radio chip**
  The used Radio chip is an Atmel AT86RF212b low power 800 Mhz transceiver. It features an AES 128 bit hardware accelerator in order to speed up the encryption process. Data buffering of up to 128 byte is supported using the built in FIFO Static random-access memory (SRAM) storage. IEEE 802.15.4 support is built in, but optional. The radio chip can be configured to disable IEEE 802.15.4 support by using an undefined frame control field within the header of a frame. Using this technique enables implementing custom protocols.

ARM processors

Computing hardware continuously improves since its invention. One more recent driving factor behind this advancement is the great success of mobile phones which are mostly powered by Advanced RISC Machines (ARM) processors. This led to ARM processors being available cheap and therefore gaining widespread adoption. The advantage of ARM processors is that they come with an efficient instruction set which optimizes execution speed and power consumption. As of the time of this writing, cheaper ARM processors can be acquired at similar prices as some microcontrollers while having much more computational power. So ARM processors are also gradually entering the market of IoT devices.

3.2 Introducing the core Pikrit protocol

As the thesis title implies, this document introduces a routing protocol named *Pikrit*. The core ideas are drafted in this chapter which shall represent the protocols initial design specification. Pikrit is directly tailored to match the requirements defined in Section 3.1 and can generally be seen as a source routing protocol. Source routing means that each address, from the sender to the receiver, is stored within the header of each transferred message. In order to achieve this, a lot of knowledge about the network has to be determined beforehand, which is primarily done by the coordinator (See Section 3.2.2). The design provides different kinds of addressing

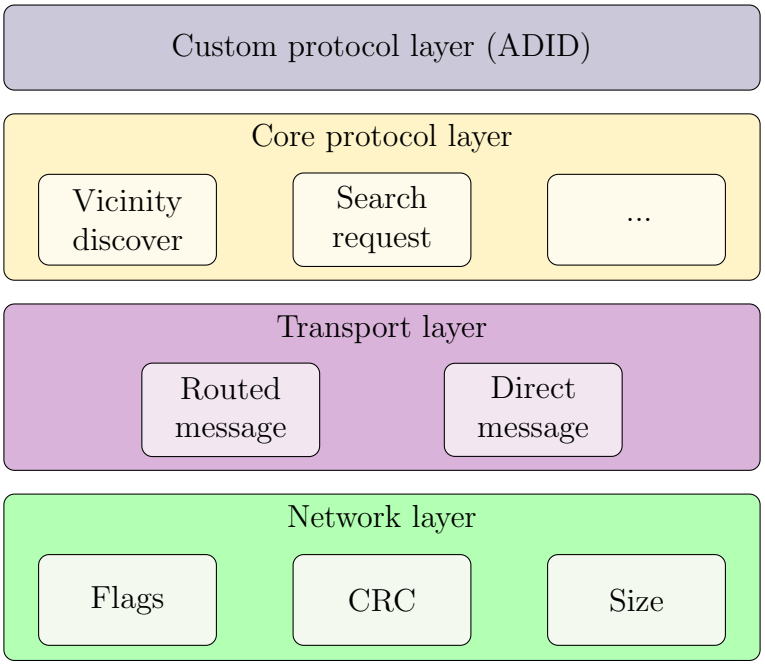

Figure 3.1: Conceptual model of different layers within Pikrit

for a variety of supported network participants. In general the route is restricted to a statically bounded number of hops which is defined at compile time. Currently, a hop count of four is considered enough for the target network size. Figure 3.1 shows a conceptual model of the whole stack to get a better overview of the protocol internals. On the bottom there is the network layer which wraps the content in basic functionality and sets the necessary flags. This layer is the closest layer to the actual hardware. Building on top of this foundation is the transport layer which defines the style of addressing within Pikrit. Once addressing is handled, the core protocols, as described in Section 3.4.4, define the context and application which is supposed to receive the data. If the core protocol layer is not enough, a custom layer can added by application developers.

3.2.1 Definition of network participants

Not all network participants are expected to work the same. Pikrit's design therefore includes a few special use cases. Coordinators, sometimes also called supervisors or orchestrator throughout this thesis, are central for enabling sensible routing of messages. Active participants can route packets through the network and can be coupled with so called passive participants. Passive participants are primarily designed for energy constrained environments. Those three fundamentally different network attenders are supported in the initial design draft.

3.2.2 Coordinators

The component which maintains the networks routing information is the coordinator. Each Pikrit network is allowed to have exactly one of them. They are expected to have enough computing power to solve the Single-source shortest path (SSSP) problem with 253 vertices and their respective positively weighted edges. Often coordinators are also the gateway into the internet, receiving commands and routing them to the necessary active network participant. When adding new components to the network, the coordinator is the main actor as it knows all routing addresses of all current network participants and generates new addresses or recycles old addresses in case an active participant has been added or removed. It has to be told which devices are expected in the network via direct addresses (See Section 3.3.1) and keeps a table with a mapping from direct addresses to routing addresses after it has set the nodes routing address successfully. Pikrit will work without the use of a coordinator, however figuring out how to route a message through the network is facilitated substantially when using using one.

3.2.3 Active participants

Most commonly small actors or sensors in networks are called nodes or motes. Their responsibilities within the network can be seen as a subset of the ones the coordinator. In order to deliver a message to an active participant two different kinds of addresses can be used: Routed addressing and Direct addressing (See Section 3.3). Upon receival of a routed message it checks within the message header if it is the target destination. In case it is, a callback is triggered delivering information about the addresses and Application Domain IDs (ADIDs). Otherwise the message header is updated in order to route the packet to the next active participant in the source routing queue. If an active participant receives a routed message or its routing address is set, it stores the reverse route in its EEPROM. This is important if the node wants to send notifications to the coordinator in case of occurring events, like the push of a button coupled via a passive participant or value updates of a sensor component. Active participants are intended to be the most common component of a Pikrit network and they are optimized for the powered sensor and actor use case.

3.2.4 Passive participants

Special actors with even smaller responsibilities than active participants are the passive participants. It is assumed that they are very power constrained or even autark systems. They do not directly communicate with the coordinator as they are coupled with active participants. The targeted use case for this category of

devices are sensors or switches, which are in a permanent deep sleep mode except if they detect an action which the network should be notified about. Then they send a message to their coupled partner which forwards it to the desired target participant, usually being the coordinator.

3.2.5 Adding network participants

There is deliberately no automatic procedure to discover unknown neighbors in Pikrit. Also things like neighbor solicitation are not part of the design. The only component which has knowledge of other components in the network is the coordinator. Only participants that the user explicitly adds via its hardware address are recognized by the coordinator. This is done because the user should be the entity that decides which components are part of the network and which ones are not. After adding a device to the coordinator, it searches for the node throughout the network using search requests (See Section 3.4.4). Upon finding the node the coordinator sets its routing address. Once the routing address is set the node stores the route back to the sender and is then considered a participant within the network.

3.2.6 Participant limit

Address spaces are always finite, bounded by the length of the representing integer number. As there are direct and routed messages, which each have a different size, there are two possible network participant limits. The direct addressing space is calculated by using the 4 byte hardware address. So there are theoretically $2^{(4 \cdot 8)} = 4{,}294{,}967{,}296$ possible devices within the network which can be sent a direct message.

Routed addressing is much more limited, as the address is only 1 byte in size. The theoretical limit for routed messages is therefore $2^8 = 256$ devices within one network. Practically there are a few unusable addresses that are used for network administration purposes, which Section 3.3.2 lists.

3.2.7 System separation

Pikrit is intended to be used in production with encryption enabled. An analysis of encryption properties required within home automation systems can be found in Section 3.1.5. The reason encryption is referred here is that Pikrit intends to solve the system separation problem using the chosen encryption key. System separation means that if one node is added to a coordinator using the search request control

message (see Section 3.4.4), it should not be able to be controlled by another coordinator. Each coordinator provides its own random and ideally periodically changed encryption key K_i. The idea is that K_i is sent with the initial search request and then updated as needed. This process is not defined at the time of this writing but will be once Pikrit is used in a real world environment. As soon as a node knows its encryption key it is bound to the system and will not accept messages from other coordinators. The assigned coordinator can revoke its key so that the node can be re-used within other systems.

3.3 Addressing

A basic design goal of Pikrit is to allow communication without having to set routing addresses. This allows for things like coupling of passive participants, easier debugging and easy communication between nodes, if needed. In order for this to work there are two possible addressing schemes. Each Pikrit frame has to provide information about which kind of addressing to expect within the header, as described in Section 3.4. Addresses have the following properties:

- Size of addresses is chosen at compile time
- All network participants must have been compiled with equal hop count, direct and routed message size
- Then the equation (3.2) allows for the calculation of the size of direct messages:

$$\text{sizeof(routing address)} = \frac{\text{sizeof(direct address)}}{\text{hop count}} \tag{3.2}$$

  Where it is assumed that direct address size and the hop count is divisible without a remainder.

3.3.1 Direct addressing

Nodes are expected to have a 4 byte long hardware specific address, namely *direct address* (also called hardware address), which are not allowed to have duplicates globally. So all manufactured devices are required to have a unique number (e.g. serial number) and the direct address is immediately derived from this number. These addresses can be used by all nodes at any point in time to try and reach a specific node. However, routing is not defined when using these addresses as this would significantly increase the frames header size. It would theoretically be possible to use a custom ADID (See Section 3.4.4) in order to enable routing with direct addresses but this functionality is not part of the core specification.

3.3.2 Routed Addressing

On the other hand there are so called *routing addresses*. Coordinators are responsible for distributing routing addresses within the network. As the name suggests they can be used to forward packets along a route. If a node has not set its routing address it can not participate as part of a route in a Pikrit network. Generally route addresses are smaller than direct addresses, Pikrit specifies them with a size of 1 byte. Table 3.1 shows the reserved addresses of the Pikrit protocol. These addresses are assigned a special meaning and can not be used in order to route addresses.

Address	Reserved for
`0x00`	Unset routing address
`0x01`	Address of coordinator
`0xFF`	Magic number

Table 3.1: Reserved routed addresses

3.4 Pikrit frame specification

A frame is a digital data transmission unit which is bound in size. The bound is also often referred to as the Maximum transmission unit (MTU) which specifies how much information can be fit in a single frame. Table 3.2 showcases the Pikrit frame upon which core and custom protocols rely. The MTU of a frame is often specified through the hardware and its properties. In order to transmit a basic routed frame successfully, Pikrit needs an overhead of merely 9 bytes in the minimal case. Depending on the ADID (See Section 3.4.4) various protocols are specified which might alter the required frame size.

Header					**Payload**	**CRC**
Properties				**Addressing Scheme**		
Front			**Frame Size**			
Flags		**ADID**				
ENC	DM	Application Identifier	Length	Dependent on DM flag	Data	Check
1 bit	1 bit	6 bit	2 byte	6 or 8 byte	Fill	2 byte

Table 3.2: Core Pikrit frame

3.4.1 Header

The header is the first part of each Pikrit frame and consists of two properties and the addressing scheme. The properties can again be split up into the front and the ADID.

Property: Front

The front is the first part of the header and is a sequence of combinations of flags and ADIDs and has a total length of one byte. Two flags precede the ADID which control how the frame is perceived. They are one bit in size each and can have the boolean value `true` or `false`. Following the flag value, the ADID specifies which content is expected within the payload.

- **ENC - Encryption flag**
  In order to allow for identifying encrypted payloads a special flag *ENC* is introduced which is an abbreviation for Encryption. If this flag is set the payload is expected to be encrypted. As encryption is not part of the initial Pikrit specification (1.0) and therefore not part of this thesis, this flag is a precaution for further Pikrit specification updates and will not be discussed in-depth.

- **DM - Direct Message flag**
  The direct Message flag is responsible for determining which addressing scheme (see Section 3.4.1) is used. This is either routed message addressing where routing addresses are used in static source routing to transfer frames over multiple endpoints or direct message addressing where the full hardware address is used to directly communicate with neighbours.

- **ADID - Application Domain ID**
  The ADID field is interpreted as a 6 bit wide unsigned integer with a special application assigned. Using 6 bits, up to $2^6 = 64$ unique applications can be specified. As 64 applications might not be sufficient for all use cases, only the core application areas such as control messages or important general purpose data transfer methods are modeled using these 6 bits. Those 64 core ADIDs are all reserved and should not be used to incorporate custom protocols as subsequent updates to the Pikrit specification will make use of these reserved IDs. However, there is a special ADID which allows for creating own protocols. Section 3.4.4 describes this, as well as the specification of core ADIDs (control messages), in more detail.

Property: Frame size

This frame property summarizes the size of the whole frame including header, payload and CRC sum into a two-byte field.

Addressing Scheme

As the only dynamic field of Pikrit properties, addressing schemes have a huge variety of responsibilities. Functions and size of the addressing scheme is determined by the used flags. They are configured using the *direct message* and *search request* flags within the header part of the properties. The direct message bit within the front switches between the following addressing schemes:

- **Direct message scheme**
  If the *direct message* flag is set to `true` the addressing scheme part in Table 3.2 contains the addresses defined in Table 3.3. This table solely contains the full address of the node sending the message as well as the full address of the desired destination. If the destination address is not available or not in range, the frame can not be delivered, as frames with the *direct message* flag set cannot be routed.

Full source address	Full destination address
4 byte	4 byte

Table 3.3: Direct message addressing scheme

- **Routed message scheme**
  The second and more important addressing scheme of Pikrit is the routed message scheme which is enabled when setting the *direct message* flag to `false`. Routing is enabled in this scheme and the header is as small as possible in order to allow for a large payload. Table 3.4 shows the fields which are added to the addressing scheme part of Table 3.2. All addresses in this scheme are routing addresses. If a node detects that it should forward a message to

Sender address	Route				
	Origin address	Destination route addresses			
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte

Table 3.4: Routed message addressing scheme

  another node it writes its own routing address in the *Sender address* field.

Contrary to this approach the *origin address* field always contains the routing address of the first, original sender. These pieces of information allows a node to determine if the frame has to be routed to another node or if the receiving node should consume the frame. *Destination route addresses* are four static fields which contain all the nodes that are part of the route. Unnecessary destination route addresses are filled with the reserved address `0x00`, as described in Table 3.1.

3.4.2 Payload

The remaining bytes of a frame are free to be filled with a payload. This can be data filled in by application developers, information of higher level protocols or any other kind of data. More specifically the ADID is the defining property upon which data to expect within the payload field.

3.4.3 CRC

CRCs provide integrity by detecting transmission errors and they are two bytes in size. If an error is detected, the Pikrit protocol is not responsible for re-transmission of the frame. Protocols which are higher in the stack can do this if reliable message transfer is needed.

3.4.4 Custom and core Application Domain ID

The use of ADIDs is what makes the Pikrit protocol very versatile. This identifier allows endpoints to recognize incoming data, similar to a "port" in the popular Transfer Control Protocol (TCP) and Unified Datagram Protocol (UDP) protocols used within the internet. As Pikrit is trimmed to be as lightweight as possible, by the means of data overhead, the protocol reserves 63 predefined application IDs. These are used for control messages, essential core applications as well as custom definable ADIDs as listed within Table 3.5.

3.5 Core Application Domain ID definitions

The following sections highlight some of the core ADID protocols and reveal the intended value they provide.

ID	Application Protocol	Type
0x00	Search request	Control message
0x01	Node vicinity discovery	Control message
0x02	Ping frame	Control message
0x03	Solid data frame	Core protocol
0x04 - 0x05	Multi-frame bulk data transfer (MFBD)	Core protocol
0x06 - 0x3E	Reserved	Reserved
0x3F	Custom application domain	Custom protocols

Table 3.5: Control Message / core ADIDs

3.5.1 Search request control message

An ADID of 0x00 expects the payload to contain a *search request* control message frame. Essentially, such a frame adds a new node into the network. All necessary information to make a new node part of the network is transmitted. Requests are described in Table 3.6. The appended field is the direct address of a node which is

Payload	
Hardware address	Encryption Key
4 byte	not specified

Table 3.6: Search request content

searched by the coordinator within a network. Important to note is that this control message is initially only specified for routed message usage. A new routing address is generated and inserted as the last valid destination route address. Targets of this control message are identified by the direct address and update their routing address to the last valid entry in the destination route address. In order to provide encryption throughout the network this frame is used to deliver an encryption key to the nodes. As encryption is not part of the initial specification the encryption key field is not used.

3.5.2 Node vicinity discovery request

With a node vicinity discovery frame a network participant can be asked to reveal their local neighborhood. Each node within Pikrit listens for data transfer and if it receives a frame from another node it stores this node's Link Quality Indicator (LQI) into a neighbor lookup table. The tasks of the node vicinity discovery are:

- Let another network participant know all routing addresses that are part of the nodes local neighborhood.

- Retrieve information about how good the link quality is to this node.

A node vicinity discovery frame is defined with an ADID of `0x01`. Node vicinity discoveries are only defined with *direct message* flag set to **false**. Otherwise the behavior is currently undefined which might change with future versions of Pikrit. The last routing address within a node vicinity discovery frame's destination route addresses defines the node which shall respond with information about its local neighborhood.

Table 3.7 shows the payload structure. The offset and length fields represent a range of routing addresses for which the coordinator wants to know the LQIs. After a node receives a request for a node vicinity discovery it responds with sending out all LQIs values it knows for the given range of routing addresses. If the node never received a message for a routing address it fills those gaps with the magic number `0x00`.

Payload	
Offset	Length
1 byte	1 byte

Table 3.7: Node vicinity discovery response content

Node vicinity discovery response

Responses to the node vicinity discovery message contain either a snippet or the full neighbor list, depending on the length value of the request. If the amount of LQI values exceeds the MTU of a frame, the coordinator has to repeat the request with adjusted offset and length values. Node vicinity discovery responses look as described in Table 3.8. An example is with parameters length = 5 and offset = 7 is

Payload		
Neighbor with routing address offset	...	Neighbor with routing address offset + length
Size: 1 byte each		

Table 3.8: Node vicinity discovery response

given in Table 3.9.

Payload						
Routing address	7	8	9	10	11	12
LQI	0	0	0	85	91	77

Table 3.9: Node vicinity discovery response example

3.5.3 Ping frame

A ping frame is a convenience frame for detecting whether a given address is reachable. At first a request frame is sent to the address in question. If the node using this address is reachable and receives the ping request, it sends out a ping response using the inverse route. Once the response is received by the original sender the reachability test was successful. The request and response frames in detail:

- **Request**
  The payload of a ping request starts with the four American Standard Code for Information Interchange (ASCII)-characters "`ping`" and is then followed by a random ASCII character sequence of length 60. So the overall payload has a size of 64bytes.

- **Response**
  Similar to the request payload the response payload starts with the four ASCII-characters "`echo`" and is then followed by a random ASCII character sequence of length 60.

Within the reference implementation discussed in Chapter 4 the random ASCII character sequence is starting with the character '`a`' and is then followed by subsequent characters within the ASCII standard up until 60 characters are reached.

3.5.4 Multi-frame bulk data transfer core protocol

A lot of use cases for building automation, especially the control of smart devices, can be covered with quite small payloads so that the MTU size is not exceeded. But many applications do need functions where the size of the data to transmit exceeds the MTU bounds quite substantially. For example, the upload of a firmware file in order to provide software updates will usually exceed the MTU size. The MFBD protocol is introduced as a connection oriented protocol to cover such use cases. It has its own header which can be seen in Figure 3.2. Three ADIDs are reserved for using MFBD. A transfer operation is always bound to a specific ADID which

cannot be used otherwise during transmission. In case a recepient of a MFBD has already bound an active data transfer to an ADID, an error frame is returned indicating that the communication cannot be established.

<table>
<tr><th colspan="4">Pikrit payload</th></tr>
<tr><td colspan="3">MFBD header</td><td rowspan="2">Data</td></tr>
<tr><td>Stage ID</td><td>Transfer to check (TTC) ratio</td><td>Data regulation</td></tr>
<tr><td>3 bit</td><td>5 bit</td><td rowspan="2">up to 4 byte</td><td rowspan="2">fill</td></tr>
<tr><td colspan="2">1 byte</td></tr>
</table>

Figure 3.2: Header of a MFBD frame

Stage ID description

There are four different types of frames within MFBD. To differentiate these frames the stage ID is used. The four stages are:

- **Initiation Frame**
  The first frame when trying to send bulk data is the initiation frame. When this flag is set the data regulation section has a size of 4 byte and contains the total length of the data to transfer. This results in a maximum size of 4 Gb. The receiver of this frame remembers the frame size and expects to receive frames on this ADID until all data is transmitted.

- **Transfer Frame**
  Actual data is delivered by transfer frames which also have a data regulation section size of 4 bytes. This section is filled with the offset of the currently transmitted data. The data section does contain the actual data to transfer and fills the frame until the MTU size is reached. A *transfer unit* in this context is referred to as a block of TTC consecutive transfer frames.

- **Checksum Frame**
  In order to provide data integrity checksum frames are periodically sent back to the originator after TTC transfer frames. The data regulation section in this case is 2 bytes in size and contains the CRC sum over the previous transfer frames until the last checksum frame. If the checksum for one transfer unit is not valid, the whole transfer unit has to be re-transmitted. This should prevent errors due to the following scenarios:

  – **Data corruption while transferring**
    Depending on the signal strength and quality, radio wave information can be misinterpreted by the recipient. If there are no checks in place for data integrity the consequences might be unacceptable. For example when

receiving a corrupt firmware image for microcontrollers and flashing it, the microcontroller might be unusable afterwards.

  - **Packet loss**
    In case a frame is not detected by the radio chip the data transmission can get out of sync and could corrupt data this way. While this is already handled with the offset field of a transfer frame, the checksum frame further improves communication reliability.

- **End Frame**
  The end frame is a special case of a checksum frame. After all transfer frames have been sent and the offset is equal to the size of the received data, an end frame is sent to the originator. The end frame contains the checksum for the bulk data within the data regulation section. Here a well suited CRC function shall be used to calculate a checksum over the entirety of the transferred file. Well suited in this context means that the underlying hash function has very good collision resistance.

- **Error Frame**
  In case of a fatal situation, the response is an error frame. The data regulation size and data sections do not matter in this case. A fatal situation occurs if an initiation frame is sent while a MFBD transmission is already executed on the given ADID. Upon receiving an error frame a good idea might be to try to transfer data over a different ADID.

Transfer to check rate setting

The sender of a MFBD frame chooses how many transfer frames are delivered before the recipient should respond with a checksum frame. For varying sizes of data this setting can be adjusted to achieve a good balance between integrity and transmission speed. As an example, a TTC setting of 1 would mean there are equally as many transfer frames as there are checksum frames. A TTC setting of 0 disables intermediate checksum frames.

Data regulation section

The data regulation section within the MFBD header is dependent on the stage ID.

- **Initiation (Size:** 4 bytes**)**
  Contains the whole length of the data transfer.
- **Transfer (Size:** 4 bytes**)**
  Contains the offset value for the currently transmitted frame.
- **Checksum (Size:** 2 bytes**)**
  Contains the CRC sum of the previous transmission unit.
- **End (Size:** 4 bytes**)**
  Contains a suitable CRC checksum over the whole data block.
- **Error**
  Not defined.

Retransmissions

Given a transmission problem of either one of the following types:

- **Timeout**
  The frame has been sent but the expected response has not been received within an estimated window of opportunity. A good time estimation algorithm is presented in Section 3.5.4 Round Trip Time.
- **Invalid Checksum**
  The checksum contained within the checksum frame does not match the expected value.

If one of these situations occurs the previous transmission unit has to be re-transmitted. A recipient can determine from the transmission frame's data regulation section, which is filled with the data offset, whether a frame is re-transmitted or not. In case a single re-transmission is repeated 5 times the whole file transfer is cancelled.

Round Trip Time

The Round Trip Time (RTT) is the time it takes for a node to receive a response to a request over a given route. Pikrit does not support congestion control and therefore only participants with the same transmission speed are considered. Using this knowledge in combination with the source routing nature of the protocol the RTT can easily and accurately be calculated before sending a frame. Equation (3.3) shows how to calculate the RTT in dependence of the data length d. The variables and their respective meaning are:

d	$\cdots$	Frame length
$t_{rt}(x)$	$\cdots$	RTT in dependence of data length x
N	$\cdots$	Hop count
$t_{st}(x)$	$\cdots$	Single transmission time in dependence of data length x
r	$\cdots$	Reference data length
r_{MTU}	$\cdots$	MTU data length
t_c	$\cdots$	Central Processing Unit (CPU) processing time

$$t_{rt}(d) = N\frac{t_{st}(r)}{r}(d + r_{\text{MTU}}) + (2N - 1) \cdot t_c \tag{3.3}$$

Consider Figure 3.3 for a short explanation on how Equation 3.3 comes about. Imagine sending a frame of data length d over a route with $N = 2$ hops. The initial sending time from the origin takes $t_{st}(d)$. This time is followed by the computing time t_c of Node 1 to figure out that it should route the just received frame to Node 2. So the time $t_{st}(d)$ is needed again, which is also followed by the processing time t_c. As we do not know how large the response frame from node 2 will be, we assume the worst case time $t_{sc}(d_{\text{MTU}})$, which is the time for a frame with the size of the MTU. In order for the message to return to its originator, it has to pass by Node 1 again and t_c is needed for the third time. When counting and generalizing the just discovered times the $t_{st}(d)$ and $t_{st}(d_{\text{MTU}})$ times are used N times while the time t_c is used $N - 1$ times.

To get the time $t_{st}(x)$, a node has to empirically evaluate the duration of a ping frame. Therefore it has to send a ping message with a reference length r, where it is known that the response message is of the same length. The amount of time this procedure takes is measured and assumed to be $t_{\text{measured}} = 2 \cdot t_{sc}(r) + t_c$. Under the assumption of a constant t_c, which has to be estimated, $t_{st}(r)$ can be calculated using equation (3.4). Assuming the transmission time is proportional to the data length, the time for all data lengths d can be linearly interpolated.

$$t_{st}(r) = \frac{t_{\text{measured}} - t_c}{2} \tag{3.4}$$

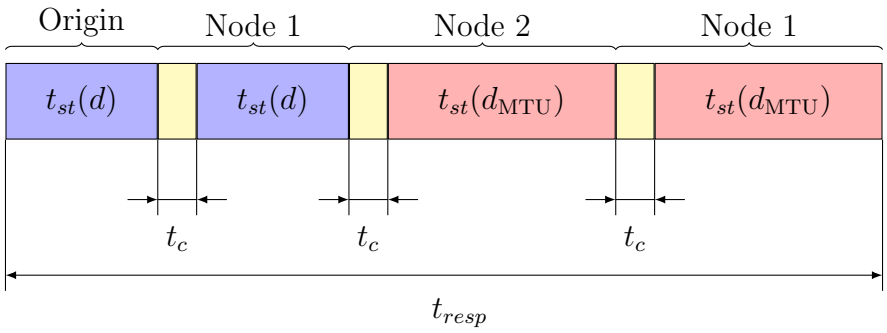

Figure 3.3: Quintessential time diagram of a 2-hop data transmission

3.5.5 Solid data frame

Solid is a control protocol for HELLA automation products. One ADID is reserved for sending such frames.

3.5.6 Custom application domain

Pikrit allows application developers to introduce their own application domains. Their IDs are not directly part of the header like it is the case with core Pikrit ADIDs. Instead, the ADID is set to 0x3F. If this is the case, the first two bytes of the payload are used to create up to $2^{16} = 65535$ custom application domains.

3.6 Coordinator Protocol Extensions

The core Pikrit protocol specification does define neither a way of retrieving information about the link states within a network nor a method for calculating shortest paths within the network. Every Pikrit protocol coordinator implementation can handle these features as desired. Recommended solutions for these problems, however, are provided via what is called coordinator protocol extensions. There are two protocol extensions attached to the core protocol specification, called Peridoic Link-State Retrieval (PLSR) and Weighted undirected shortest path (WUSP).

Routing Addresse	1	2	3	4	⋯
1		0	0	0	0
2			0	0	0
3				0	0
4					0
⋮					⋱

Figure 3.4: Adjacency matrix representing link states

3.6.1 PLSR

Coordinators within Pikrit networks are usually responsible for finding routes to a node within a network. But in order to transfer such a frame, the coordinator needs information about the interconnections between single nodes. As coordinators know the direct address of all network participants (see Section 3.2.2), they can use this information as input for an algorithm which searches for nodes that can only be contacted indirectly. An example algorithm for this is called Periodic Link-State retrieval. It is responsible for filling a data structure in order to easily apply searching for shortest paths, as seen in Section 3.6.2. The data structure of choice is a symmetrical adjacency matrix which is filled with a reliability metric called link rank as seen in Figure 3.4. Being a symmetrical data structure only one half of the matrix has to be stored as a data transfer can be assumed to be direction independent. All links to a network participant itself are also not stored as sending messages to oneself will not increase efficiency of a routing algorithm.

Link rank

The link rank lr is determined via the LQI and the packet loss ratio p_{lr}. An initial version can be seen in Equation (3.5). It is a non-negative combination of the signal quality weighted by the success ratio of the last N_R frames which have been sent over this exact route R. As long as the amount of frames sent over this exact route has not reached N_R, p_{lr} is always 1. Otherwise p_{lr} is calculated as the ratio of successful frames transferred via the route R in dependence of the total number of frames transferred via R. A good value for N_R has to be determined by a longer testing phase, an initial value of $N_R = 5$ is being used.

$$lr(n) = \begin{cases} \text{lqi} \cdot p_{lr} & \text{If } n \geq N_R \\ \text{lqi} & \text{otherwise} \end{cases} \tag{3.5}$$

lqi	$\cdots$	Link Quality Indicator
p_{lr}	$\cdots$	Packet loss ratio
lr	$\cdots$	Link rank
N_R	$\cdots$	Frames sent over route R

Algorithm

Listing 3.1 shows a java oriented pseudo-code for the PLSR function `plsr()`. Its signature consists of a list with direct addresses which have to be found, an adjacency matrix as discussed above, the actual route to search (initially `[0, 0, 0, 0]`) as well as the current depth, starting with 1. The function `appendNextFreeRouteAddress(...)` takes the route and current depth as argument. It gets an unused node address if available and appends it at the current depth within the route and returns it for convenience. Next step is iterating over all direct addresses, as this algorithm makes use of breadth-first search, because it can be expected that a lot of nodes are already found at depth 1. Excluding the directly approachable nodes reduces time complexity for building the adjacency matrix significantly. Each direct address is searched for over the given route which is implied via the function `vicinityDiscover(...)`. In order to provide a simple explanation of the algorithm this method is assumed to be blocking until a response occurs. Real world implementations should avoid this blocking or at least handle it in its own thread. If a response is retrieved successfully, the link rank of all neighbors are written to the adjacency matrix applied via the `addLink(from, to, rank)` function. Directly after that, the direct address is removed from the field of searched addresses as we have found it. The `routeAddress` is from now on in use and cannot be reused by the coordinator until explicitly invoked otherwise. So a new route address for the next hardware address has to be acquired as shown in line 26.

Once the addresses array is empty we are done and can return. Otherwise, if we have no routes at given depth available anymore, we need to increase the depth and call the algorithm recursively to search at a deeper level (lines $34 - 38$). Note that this is just pseudo-code delivering the conceptual idea of this algorithm.

3.6.2 WUSP path calculation

As a last step to achieve a self-organizing routing mechanism the link state information is used to calculate the shortest path to a desired node. Shortest, in this context, is due to respect of the link rank as described in Section 3.6.1. This resembles the classical SSSP with positive weights problem within graph theory which has been widely discussed and therefore is a rather large topic at hand. The currently fastest algorithm with respect to time complexity was presented in 1999

```
static void plsr(Collection<HardwareAddress> addresses,
    AdjacencyMatrix adjMatrix, int[] route, int depth) {

  if (route == null)
    return;
  int routeAddress = appendNextFreeRouteAddress(route, depth);

  for (HardwareAddress hwAddress : addresses) {
    //For simplicity assume vicinityDiscover is blocking.
    Response response = vicinityDiscover(hwAddress, route);
    if(response.isSuccess()) {
      adjMatrix.addLink(
        lastNonZeroAddress(route),
        routeAddress,
        response.getRank()
      );
      Collection<Neighbour> neighbours = response.getNeighbours();
      for (Neighbour neighbour : neighbours) {
        adjMatrix.addLink(
          routeAddress,
          neighbour.getRouteAddress(),
          neighbour.getRank()
        );
      }
      addresses.remove(hwAddress);
      setRouteAddressInUse(routeAddress);
      routeAddress = appendNextFreeRouteAddress(route, depth);
    }
  }

  if (addresses.isEmpty()) {
    return;
  }
  else {
    boolean routesAvailable = getUnusedRoute(route, depth);
    if (!routesAvailable) {
      depth++;
    }
    plsr(addresses, adjMatrix, route, depth);
  }
}
```

Listing 3.1: PLSR Pseudocode

and is described in detail in [Tho99], accomplishing the task within linear time boundaries $\mathcal{O}(n)$. However, it needs to store a lot of additional information to do so. Other viable options for calculating shortest paths with non-negative weights are the popular Dijkstra algorithm or the A* algorithm.

3.7 Examples of important protocol messages

In this section the most essential Pikrit control messages are demonstrated in detail. Important to note is that all numbers and addresses presented in this section are hexadecimal numbers. Some parts of the message is composed of wildcards (Represented via **x**) as they are dependent on the actual payload.

3.7.1 Direct Message

Beginning with the simplest example, a direct message is illustrated in Figure 3.5. This example sends a payload from the node with hardware address `0x0D` to the node with hardware address `0x0C`.

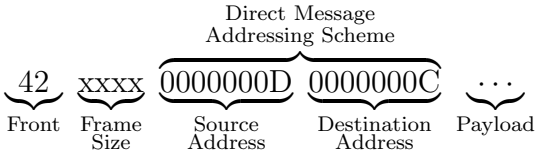

Figure 3.5: Frame of a direct message in Pikrit

3.7.2 Routed Message

A bit more interesting is the routed message example. Figure 3.6 shows a message from node 04 to node 01 via a route, indicating the current sender node via the "Frame" label. The routing addresses of the involved nodes are, in order of execution, 04 as the origin of the example frame, 03 as routing node, 08 as the current sender, 05 as routing node and the destination 01. Sending a message along this route is done with the Pikrit header described in Figure 3.7.

The first element is the Front which is defined with the ADID of a Solid data frame (02). As this message is currently sent by node 08 and was originally sent by node 04 they are the sender and origin address respectively. Next up is the four byte route, which describes along which path the payload shall be sent.

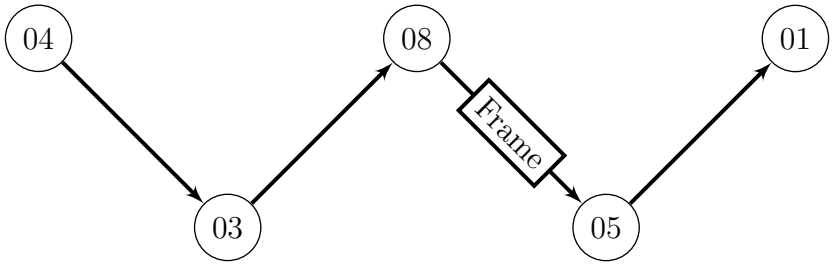

Figure 3.6: Sending some payload via a route

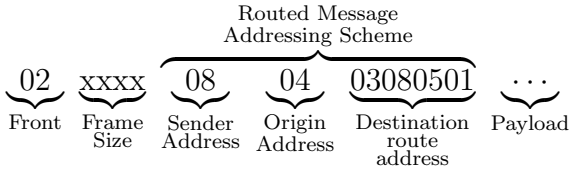

Figure 3.7: Example of a routed message in Pikrit

3.7.3 Search request control Message

Search requests were already defined in 3.4.4 and its purpose is discussed there. This section provides an example for how to set the routing address of a node with a given hardware address. Figure 3.9 shows a small setup with a few nodes, represented by circles which have their already set routing address imprinted. One node, however, is missing its routing address, indicated by the **?** sign. Its hardware address is 8 and its desired routing address is AB. Assuming that node 0D is a coordinator, it decides to update the unset routing address using a search request as illustrated in Figure 3.8.

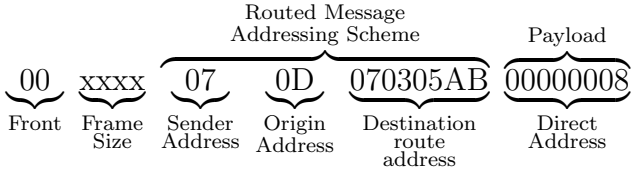

Figure 3.8: Example for setting a routing address in Pikrit

The addressing scheme is very similar to the addressing scheme of a routed message with one exception. Only the last valid entry in the destination route address does not actually describe the destination but the new routing address which shall be set by the receiving node. In order to figure out which node should update its routing address, the hardware address of that node is appended within the payload after the addressing scheme, in our case it is the node with hardware address 8.

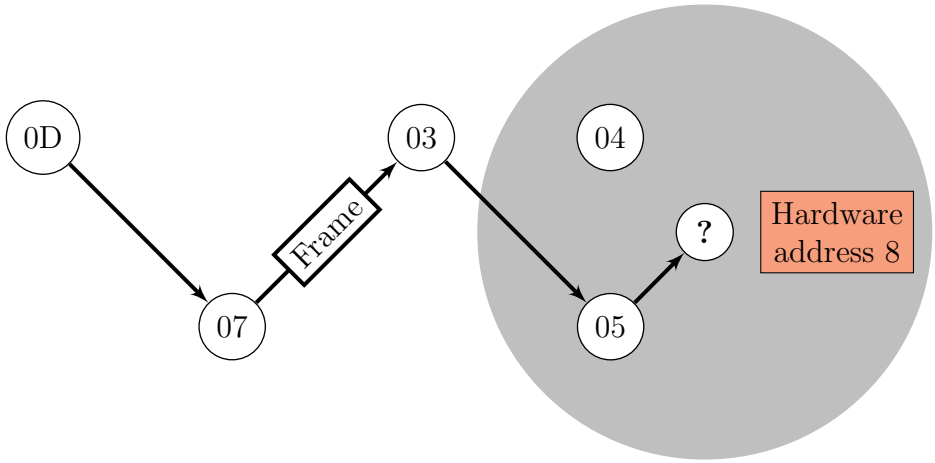

Figure 3.9: Example for search requests and vicinity discoveries

3.7.4 Node vicinity discovery

Figure 3.10 showcases a request for a vicinity discovery based on example Figure 3.9. It is assumed that the node marked with the question mark has already received the routing address *AB* as described in Section 3.7.3. The front includes the ADID 01, which is the vicinity discovery. Followed by the frame size and addressing scheme, which are very standard within the header. As before, the example message is sent from node 07. Specified by the node vicinity discovery application ID, the payload is filled with the two fields offset and length which request a specific part of the neighbor lookup table for node *AB*.

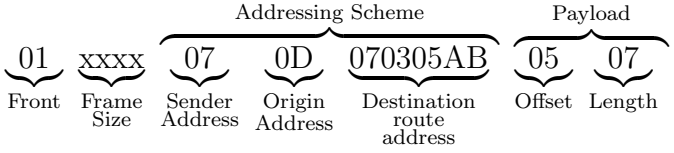

Figure 3.10: Example request for a node vicinity discovery

The payload of the response contains a list of LQIs values for each neighbor address which the node has already recognized, or 0 otherwise, as Figure 3.11 shows.

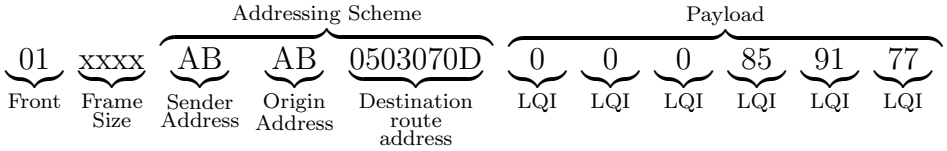

Figure 3.11: Example response for a node vicinity discovery

Chapter 4

Implementation

"When you translate a dream into reality, it's never a full implementation. It is easier to dream than to do."[1]

4.1 Pikrit integration into Contiki

Designing a protocol without continuously testing the credibility based on real world implementations likely leads to unforeseen problems. To avoid such teething problems Pikrit has been implemented as a network layer within Contiki. The process of testing ideas before finalizing them within a specification can be very revealing of unexpected design flaws. Important to note is that Pikrit has been designed with hardware and platform independence in mind. The Pikrit driver consists of three logically separated units, shown in Figure 4.1, which will be explained in the order of an outgoing frames point of view.

At first there is the *application* section that consists of the Pikrit functions which can be invoked as application developer with their respective callbacks. After invoking such a function, the Pikrit *driver* processes the frame which is then sent down the Contiki network stack (see Section 2.4.1). Lastly, within the RDC-layer of the Contiki stack, there is the *Pikrit Framer*, which handles creating and parsing frames as well as calculating header and data sizes. The primary programming language used in Contiki is C and therefore the reference implementation of Pikrit is also written in C.

[1] Shai Agassi

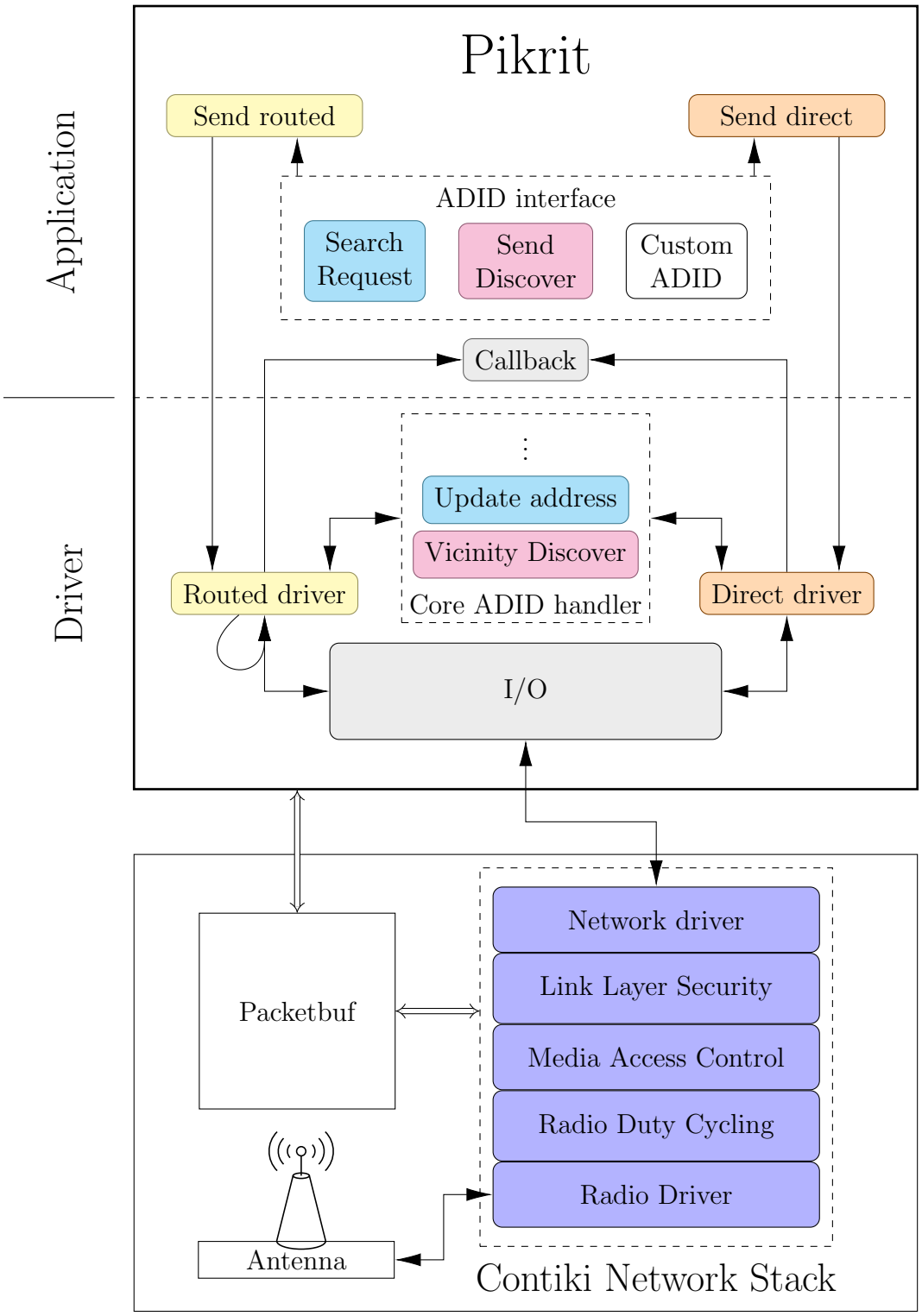

Figure 4.1: High level overview of the Pikrit integration within Contiki

```
void send_direct_message(
  linkaddr_t addr,
  const pkrt_payload payload,
  uint8_t adid
);
```

Listing 4.1: Direct message signature

4.2 Developer interface

As a Pikrit application developer the following functions and the corresponding callbacks are essential. This section describes their signatures, explains how to use them and sheds some light on their implementation within the Pikrit driver unit.

4.2.1 Sending direct messages

Listing 4.1 shows the function signature of the function which is responsible for sending a direct message. The first parameter of the signature is the address to which the message shall be sent. It is of type `linkaddr_t`, which is a union that is shared with the routed messages and its definition can be found in Section 4.3.2. Direct messages use the four-byte interpretation of the union. As a second parameter, the payload has to be specified. Payloads in Pikrit are generally of type `pkrt_payload`. The reason for this is explained in Section 4.3.3. Lastly, the `uint8_t adid` parameter within the direct message signature specifies the used ADID as described in Section 3.4.4.

Implementation

As Listing 4.2 shows, the `packetbuf` is a core mechanism in Contiki, which is loosely explained in Section 2.4.1. The function `packetbuf_copyfrom(...)` copies the contents of the `payload.payload` pointer with length `payload.length` to the `packetbuf`, so the payload is retained in memory even if the current stack returns. Secondly, the packet is attached the necessary front using a `packetbuf_attr_t`, which will be evaluated by the framer to create the correct header. To generate the correct front part of the header, the function `create_pkrt_front()` is used. It takes as parameters whether the frame shall be encrypted, if it is a direct or routed message and the corresponding ADID. Before sending the data frame, the address is also assigned to the `packetbuf`, in order to create the proper header within the framer, as explained in Section 4.3.4. The last function `pikrit_output()` just initiates the data transfer within the Contiki network stack and activates a callback upon successful invocation if a callback is defined.

```
void send_direct_message(linkaddr_t addr, const pkrt_payload
    payload, uint8_t adid) {
  packetbuf_copyfrom(payload.payload, payload.length);

  uint8_t front = create_pkrt_front(pkrt_encrypted, 1, adid);
  packetbuf_set_attr(PACKETBUF_ATTR_FRONT, front);

  packetbuf_set_addr(PACKETBUF_ADDR_PKRT_RECEIVER, &addr);
  packetbuf_set_addr(PACKETBUF_ADDR_PKRT_SENDER,
                     &linkaddr_node_addr);

  pikrit_output();
}
```

Listing 4.2: Direct message framework implemenatation

```
void send_routed_msg(
    linkaddr_t addr,
    const pkrt_payload payload,
    uint8_t adid
);
```

Listing 4.3: Routed message signature

4.2.2 Sending routed messages

Similar to sending a direct message routed messages can be sent using a single function. Its signature is shown in Listing 4.3. While also containing an address of type `linkaddr_t` this address is interpreted differently than the address when sending direct messages. As it is a union which defines an array of length `LINKADDR_SIZE` it can be accessed byte wise. Each of these bytes represents a routing address.

So the address should be filled with up to `LINKADDR_SIZE` many routing addresses. The `pkrt_payload payload` and `uint8_t adid` parameters are used the same way as in direct messages, see Section 4.2.1.

Implementation

The implementation for sending routed messages is more complex, as it is responsible for various application domains. Listing 4.4 shows that there are two function calls, `override_origin(...)` and `send_routed_msg_origin(...)`. Overriding the origin address means that this node is the original sender of the frame and it has not forwarded this frame from somewhere else. This function is mainly responsible for copying the payload into the `packetbuf` and deletes previous `packetbuf` data in the

```
void send_routed_msg(linkaddr_t addr, const pkrt_payload payload,
    uint8_t adid) {
2    override_origin(payload);
    uint8_t front = create_pkrt_front(pkrt_encrypted, 0, adid);
4    send_routed_msg_origin(addr, front);
}
```

Listing 4.4: Routed message framework implemenatation

process.

Sending the message while keeping the current `packetbuf` origin address is invoked with the function `send_routed_msg_origin(...)`. If the origin would not be overridden before calling this function the current origin stored in the `packetbuf` would be reused. This mechanism is used for routing packets throughout the network. When receiving a packet the current nodes address is checked and if it does not match the last routing address in the received `packetbuf` address the `send_routed_msg_origin (...)` function would be invoked after just updating the sender address. So this function is used for forwarding and sending messages, depending on whether the origin has been overridden or not. The implementation of this function is very similar to the direct message `send_direct_message(...)` implementation, except it sets the flags differently.

4.2.3 Search Request: Update a nodes routing address

The control message to update a nodes routing address relies heavily upon the routed message implementation. To use this control message Listing 4.5 shows the function signature. Two parameters need to be passed. At first the route upon which the new node should be searched is provided. Important to note here is that the last valid routing address within that route is already the new routing address for the node. Secondly, the parameter `hardware_address` is the hardware address of the searched node. Upon invoking this function the hardware address is searched upon the path which is specified by `route`. If a node with the corresponding hardware address receives this frame it responds with an empty frame. This will change with future specifications of Pikrit as encryption will be introduced. In case no response is received, another path within the network might be used to search for the given hardware address.

Implementation

To signalize that the current node is the actual sender, the origin address within the `packetbuf` is overridden. The searched hardware address is set as `packetbuf`

```
void search_request(
    linkaddr_t route,
    linkaddr_t hardware_address
);
```

Listing 4.5: Update node routing address signature

```
void search_request(linkaddr_t route, linkaddr_t hardware_address)
    {
    pkrt_payload payload = {.payload = (char*)& hardware_address,
    .length = sizeof(linkaddr_t)};
    send_routed_msg(route, payload, SEARCH_REQUEST);
}
```

Listing 4.6: Update node routing address implementation

address which is then used in lower network layers to create the corresponding frame. Lastly, the shared `send_routed_msg_origin(...)` function is used to send out the control message.

4.2.4 Callback structure

Upon initialization of Pikrit within Contiki an optional callback parameter can be defined. It is implemented as a `struct` with three function pointers, as demonstrated in Listing 4.7. Once the callback interface is implemented and registered using the Pikrit initialization the network stack will invoke callback actions depending on what happens within the network stack. The `(*sent)()` function is called by the framework if a frame has been sent successfully, regardless of whether it is a routed message or a direct message. If an incoming direct message is detected, the function `(*recv_dm)(pkrt_dm_hdr header)` is called, which provides the header in order to get additional metadata. Similarly, if a routed message is detected the function `(*recv_rt)(pkrt_rt_hdr header)` with the corresponding header is invoked. This is not the case for core ADIDs that are handled by the driver. Access to the data is always provided by the `packetbuf_dataptr()` and `packetbuf_datalen()` functions.

```
typedef struct {
  void (*sent)();
  void (*recv_dm)(pkrt_dm_hdr header);
  void (*recv_rt)(pkrt_rt_hdr header);
} pkrt_callback;
```

Listing 4.7: Pikrit callback definition

```
struct network_driver {
  char *name;

  /** Initialize the network driver */
  void (* init)(void);

  /** Callback for getting notified of incoming packet. */
  void (* input)(void);
};
```

Listing 4.8: Contiki Network driver interface

```
static void input() {
  const pkrt_flags flags = (pkrt_flags)packetbuf_attr(
   PACKETBUF_ATTR_FLAGS) & 0xF0;

  if (pkrt_flag_direct_message(flags) &&
      !pkrt_flag_search_request(flags)) {
    input_direct_message();
  }
  else if (!(pkrt_flag_direct_message(flags) &&
     pkrt_flag_search_request(flags))) {
    input_routed_message();
  }
}
```

Listing 4.9: Input implementation

4.2.5 Common I/O operations

As Figure 4.1 illustrates, all Pikrit operations share a common input/output unit. This unit is mainly implementing the Contiki `network_driver` (see Listing 4.8) interface. Besides the initialization (`(* init)(void)`), which is called when Contiki starts up, the network driver interface is only responsible for handling incoming packets.

The Pikrit implementation of the input is shown in Listing 4.9. It parses the flags in order to determine which code path to choose. In the reference implementation these code paths are either `input_direct_message()` for handling direct messages or `input_routed_message()` for handling routed messages. `pkrt_flag_direct_message()` and `pkrt_flag_search_request` are C preprocessor specific defines which access the corresponding bits within the flags.

A main part of input output operations is also sending packets. After filling the packetbuf with all necessary information, sending packets can be uniformly done throughout all of Pikrit by the function `pikirt_output()`, which is explained in Listing 4.10. The `NETSTACK_LLSEC` stands for Netstack Link Layer Security and is

```
int pikrit_output() {
  char routed = pkrt_flag_direct_message(packetbuf_attr(
   PACKETBUF_ATTR_FLAGS) & 0xF0);

  NETSTACK_LLSEC.send(packet_sent, &routed);

  return 0;
}
```

Listing 4.10: Output of a Pikrit frame

an exchangeable interface itself. An implementation can be chosen by defining the implementation as C preprocessor instruction in the project configuration. For the reference implementation `nullsec` (no software encryption) is chosen. When sending a packet through the Contiki netstack an optional `void*` parameter can be attached. Pikrit uses this pointer to determine whether the send callback should be triggered or if it is only forwarding routed packets.

4.3 Embedding Pikrit into the Contiki network stack

All networking implementations within Contiki are written with modularity in mind. Not all areas are perfectly separated in order to be as memory-efficient as possible. This is why some important Contiki network files have to be modified in order to integrate a new network stack. Most importantly, to allow packetbuf usage, the `linkaddr_t` union has to be used. The packetuf is used throughout the whole network stack so it is advisable to be compatible with this concept.

4.3.1 Packetbuf usage

Almost all network layers are accessing the packetbuf, as already described in Section 2.4.1. This section describes how the packetbuf had to be extended in the process of integrating Pikrit into Contiki. The most notable part of the packetbuf is an `enum` which enumerates all possible attribute and address keys. Using a few C preprocessor instructions, those key values are only used for compiling if the Pikrit make flag is set.

```
typedef union {
  unsigned char u8[LINKADDR_SIZE];
#if LINKADDR_SIZE == 2
  uint16_t u16;
#endif
#if LINKADDR_SIZE == 4
  uint32_t u32;
#endif
} linkaddr_t;
```

Listing 4.11: linkaddr_t definition

```
typedef struct {
    char* payload;
    uint16_t length;
} __attribute__((__packed__)) pkrt_payload;
```

Listing 4.12: Pikrit payload definition

4.3.2 Link address type

Contiki's Rime addresses use a union called `linkaddr_t`. Pikrit borrows some of the properties of Rime's `linkaddr_t` and extends them slightly. This is done to ensure compatibility with the rest of the network stack. The definition of the `linkaddr_t` is a union which is explained in detail in Listing 4.11. Unions are a way to store different kinds of data very efficiently with the advantage of being flexible as the data contents can be semantically interpreted in different ways. So a `linkaddr_t` can either be interpreted as an array of size `LINKADDR_SIZE` with one byte fields or as a variable of the size `LINKADDR_SIZE` itself. Pikrit primarily uses a `LINKADDR_SIZE` of four bytes.

4.3.3 Pikrit payload

Payloads in C are usually handled with pointers that point to allocated spaces within memory. One fundamental problem to solve when dealing with pointers in C is to determine the size of the content the pointer describes. In case of a C-style `char[]` string the last byte within the string is generally a `'\0'` byte, so the string can be iterated until this end of string character is reached. When dealing with an unspecified payload it is advisable to separately remember the length of the payload. The structure called `pkrt_payload` is introduced in listing 4.12 to deal with this exact problem. It consists of a pointer to unspecified, character-sized payload data as well as the corresponding length of the data. The size of a single data frame is not expected to exceed a size of $2^{16} = 65\,536$ byte.

```
struct framer {

  int (* length)(void);
  int (* create)(void);
  int (* parse)(void);

};
```

Listing 4.13: Framer interface definition

4.3.4 Pikrit framer

The framer is an optional part of Contiki which is responsible for constructing and parsing the header of frames. Listing 4.13 shows the interface for a framer. A framer has three common operations:

- Calculating the length of the message header
- Creating the corresponding data structure on outgoing frames
- Parsing the header data structure on incoming frames

In order to sanity check headers which are received, Pikrit has its own framer which is called by the RDC-Layer of the Contiki network stack. If the framer detects wrong or incomplete headers, it automatically cancels the network operation.

Chapter 5

Evaluation

"Are you living in a simulation?"[1]

5.1 Platform introduction

In order to theoretically compare the performance of Pikrit with other protocols, a series of tests using RPL and Rime on the same virtual hardware has been executed. Simulator of choice was Contikis Cooja simulator, described in more detail in Section 2.5.1. All simulation cycles were performed on virtual devices, which resemble the properties of the zolertia z1 wireless module for IoT and WSN. This choice was made mainly because Cooja provides a reasonably good simulation environment for this platform out of the box. It features a 16 MHz, 16 bit Reduced Instruction Set Computer (RISC) CPU with 8 kb of RAM, 92 kb of flash memory and a 2.4 GHz transceiver.

The main criteria of performance is the time it takes to send out and receive a response to a small payload (e.g. ping). Each of a total of five consecutive pings will be measured and their response times are logged. Of these five pings an average time to reach a node in the network and its corresponding standard deviation will be calculated. To simplify the simulation a node either has 0% packet loss rate when performing a ping if it is in range of the receiver or 100% packet loss rate otherwise.

[1] Nick Bostrom

5.2 Contiki Rime mesh

The Rime stack is a basic network stack for systems with limited resources that are not able to use the Contiki μIP stack. It is designed with modularity in mind and can therefore be used for various different tasks as stated in [Pre+14, p. 223]. In this simulation the Rime mesh implementation is tested, which sends packets using multi-hop routing to a specified receiver. Important to note here is that the Rime mesh implementation follows a very passive approach because it is not pre-calculating anything when the network is idle. If the Rime stack receives an order to send out a message to a given address it first simply broadcasts this message to its local neighbors. Given that the searched address is not one of its local neighbors it then acquires two additional channels which are responsible for searching the desired address within the network. "Channel" does not refer to the physical radio medium in this instance. A channel is an abstraction within the Rime network stack in order to handle data transmissions with multiple recipients.

5.2.1 Participants and Simulation Script

Ten z1 nodes were placed randomly into the radio environment, starting with the Rime `linkaddr_t`-address 1.0 numbered ascending up to address 10.0. The node with the address 1.0 is sending out five pings each to all other nodes sequentially.

Node one is programmed to send out a ping message to node two upon pressing its virtual button initially. After the virtual button has been pressed five times, it increases the target address by one, so the node with address 3.0 will be pinged. When node 10.0 has been pinged five times, the simulation script ends pressing the button.

Cooja comes with a javascript-based scripting interface. This can be used to automatically simulate node input and analyze node output. Triggering a button press is simulated repeatedly by the script and it waits until the node prints out that it has received a response. Both event times, when sending and when receiving a message, are recognized by the script and are used to calculate data like the average ping time and its standard deviation.

5.2.2 Topology

Figure 5.1 shows where the nodes are placed within the radio medium. All nodes which are within the green highlighted area around node one are within transmission range and have 0% packet loss ratio. Matrix 5.1 clarifies which nodes are in range of each other as showing all ranges from all nodes in the figure would be quite

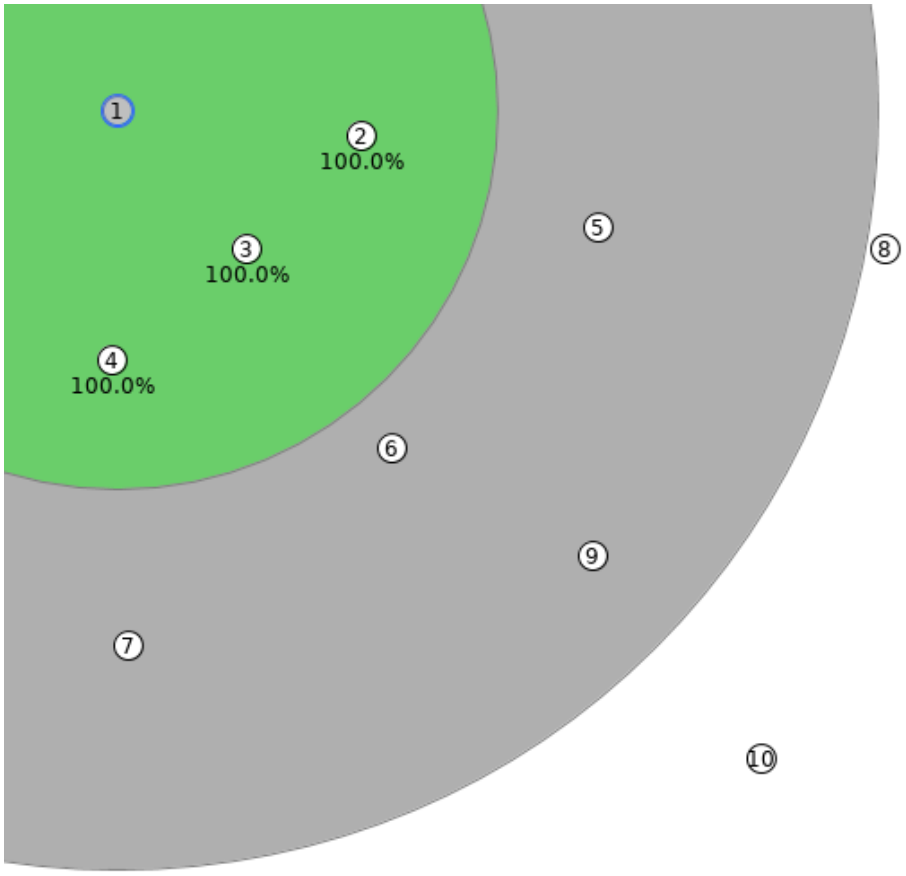

Figure 5.1: Topology for simulating Contikis Rime mesh network implementation

confusing. The topology is held as similar as possible throughout all the different simulations in order to receive comparable data.

5.2.3 Measuring response time

Table 5.2 shows the measured response times and whether the ping was successful or not. Each cell entry shows two different metrics written one above another. The upper entry in each table cell represents the average time, over the course of five pings, it took until a response was received and the lower entry is the standard deviation of the measured times.

Even transmissions without any hops are quite slow, in the best case it took 158.4 ms to receive the response ranging up to 337.5 ms. Results are getting worse when routing a message. For three hops the average time was 1248.2 ms and the standard deviation is quite large with 1836.5 ms. The main reason for the huge variance lies within the fact that there is no layout information within the network. So when pinging a device which is out of local neighbor range, two searching channels are

2	✓								
3	✓	✓							
4	✓	✓	✓						
5	×	✓	✓	×					
6	×	✓	✓	✓	✓				
7	×	×	×	✓	×	✓			
8	×	×	×	×	✓	×	×		
9	×	×	×	×	✓	✓	×	×	
10	×	×	×	×	×	×	×	×	✓
ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Table 5.1: Matrix describing reachability between Rime nodes

ID \ Hops	0	1	2	3	Success
2	190.1 ms 129.5 ms				5/5
3	174.7 ms 99.9 ms				5/5
4	175.0 ms 100.0 ms				5/5
5		549.9 ms 540.4 ms			5/5
6		575.0 ms 650.8 ms			5/5
7		717.2 ms 809.3 ms			4/5
8			875.0 ms 999.6 ms		5/5
9			1049.8 ms 1099.8 ms		5/5
10				2747.1 ms 3002.4 ms	3/5

Table 5.2: Average response times and standard deviations for Rime

acquired which are searching for the node, what takes time. After a route has been found the procedure is sped up, but for the first ping it can take a rather long time until a response is received.

5.3 RPL

RPL is already introduced in Section 2.3.1 as a suggested de-facto standard by the IETF for routing packets in WSNs. Contiki provides an implementation of this protocol which shall be simulated in order to compare it to the Pikrit protocol. Noteworthy is that the simulation here is performed after the DODAG has been completely established. This prerequisite is necessary in order to properly compare it to the static Pikrit protocol which already knows its routes when sending out packages.

5.3.1 Participants

This time, 11 z1 nodes are placed within the network. Ten nodes are running as UDP clients (Node ID 2 to 11) and one node is running as UDP server (Node ID 1). This is needed as the UDP server implementation is the root node for generating the DODAG, as described in 2.3.1. The other UDP clients are used for replying to ping frames. Addresses share the local prefix `fe80::c30c:0:?`, where the `?` character is replaced with the node ID shown within Figure 5.2. For this simple simulation example the IPv6 prefix is hidden as the node ID is sufficient to identify where a frame is sent. Node 1 is not used for simulation as its main use is repairing and building the DODAG.

5.3.2 Topology

The nodes are not positioned exactly the same as within the Rime simulation as seen in Figure 5.2 but this does not affect the simulation result as the time it takes for a frame to travel from one node to another can be neglected. More important than exact positioning is the amount of hops it takes to reach other nodes within the network as this is where the routing performance can be compared sensically.

5.3.3 Measuring response time

The achieved response times using the RPL protocol are shown in Table 5.4. An important improvement is that no packet loss has occurred, all packets are successfully transferred throughout the network. Also the response times are considerably lower with a drastically reduced standard deviation. So the data transfer when using RPL is faster and more reliable over Contiki Rime Mesh.

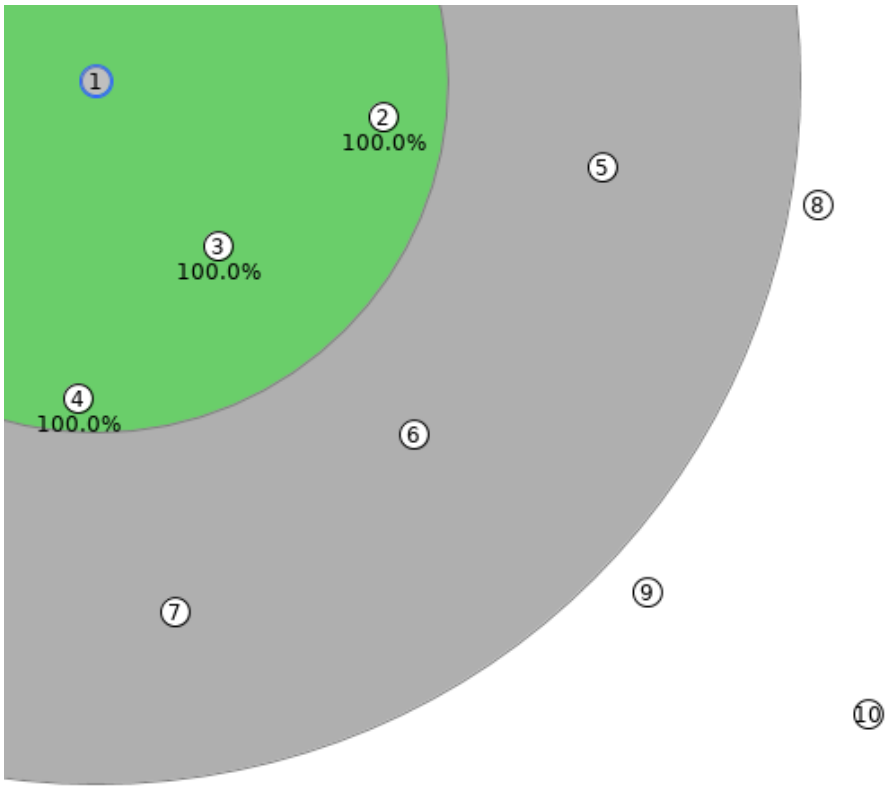

Figure 5.2: Topology for simulating a Contikis RPL implementation

2	✓								
3	✓	✓							
4	✓	×	✓						
5	×	✓	×	×					
6	×	✓	✓	✓	✓				
7	×	×	×	✓	×	✓			
8	×	×	×	×	✓	×	×		
9	×	×	×	×	×	✓	×	×	
10	×	×	×	×	×	×	×	×	✓
ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Table 5.3: Matrix describing reachability between RPL nodes

ID \ Hops	0	1	2	3	Success
2	24.7 ms 1.8 ms				5/5
3	23.5 ms 0.2 ms				5/5
4	24.3 ms 1.8 ms				5/5
5		42.5 ms 2.4 ms			5/5
6		51.1 ms 16.3 ms			5/5
7		42.7 ms 1.7 ms			5/5
8			61.1 ms 2.2 ms		5/5
9			60.6 ms 1.7 ms		5/5
10				77.0 ms 1.6 ms	5/5

Table 5.4: Average response times and standard deviations for RPL

5.4 Pikrit

The third candidate for simulation is the Pikrit protocol which is introduced with this thesis. Important to note here is that the routes to the given nodes must already be determined, similar to the DODAG creation in RPL. To achieve this it is recommended to use the neighbor route discovery protocol extension explained in Section 3.6.

5.4.1 Participants

Very similar to the previous two simulations there are 10 nodes as shown in Figure 5.3. All nodes are pre-configured to have the same route address as hardware address. The addresses are the same as the node ID's shown in the Figure.

5.4.2 Topology

The nodes are not positioned exactly the same as within the previous two simulations as seen in Figure 5.3 but this does not affect the simulation result as the time it takes for a frame to travel from one node to another can be neglected. More important than exact positioning is the amount of hops it takes to reach other nodes within the network as this is where the routing performance can be compared sensically which should by very similar to the previous two simulations. Table 5.5 additionally shows the reachability between nodes.

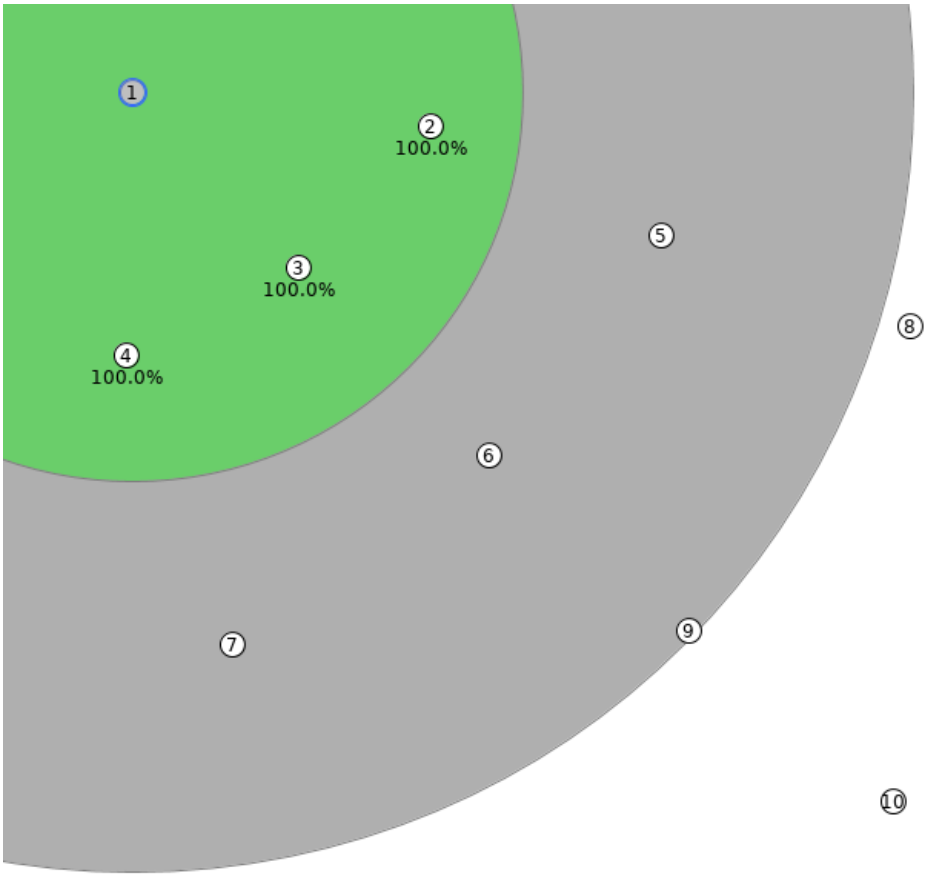

Figure 5.3: Topology for simulating the Pikrit network stack

5.4.3 Measuring response time

The achieved response times using the Pikrit protocol are shown in Table 5.6. Like RPL all responses were received successfully which implies 0% packet loss ratio. The time it takes to route messages is further reduced by using the Pikrit protocol with an almost negligible standard deviation.

2	✓								
3	✓	✓							
4	✓	✓	✓						
5	×	✓	✓	×					
6	×	✓	✓	✓	✓				
7	×	×	✓	✓	×	✓			
8	×	×	×	×	✓	×	×		
9	×	×	×	×	×	✓	×	✓	
10	×	×	×	×	×	×	×	×	✓
ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Table 5.5: Matrix describing reachability between Pikrit nodes

Hops / ID	0	1	2	3	Success
2	11.3 ms < 0.1 ms				5/5
3	11.3 ms < 0.1 ms				5/5
4	11.3 ms < 0.1 ms				5/5
5		16.5 ms < 0.1 ms			5/5
6		16.5 ms < 0.1 ms			5/5
7		16.5 ms < 0.1 ms			5/5
8			21.6 ms < 0.1 ms		5/5
9			21.6 ms < 0.1 ms		5/5
10				27.0 ms < 0.2 ms	5/5

Table 5.6: Average response times and standard deviations for Pikrit

5.5 Pikrit design

Developing a protocol is a challenging task where good design is in the eye of the beholder. There are always compromises to make in one way or another. General purpose protocols try to find a good balance between the most common use cases while special purpose ones, lean in favor of managing certain tasks especially well at the cost of some restrictions in other areas. While not all building automation tasks are time-critical, some of them can certainly be. One example would be wind

sensors measuring wind speeds that are well beyond the theoretical maximum that is allowed for fragile surface mounted objects like awnings. Automatic retraction of awnings as fast as possible is necessary to circumvent physical damage to the product and more importantly any harm that could be imposed to humans by an awning breaking off its mounted surface. Communication speed is very important in such cases and is part of the reason why Pikrit is focusing on timely data transmission. Table 5.7 shows other advantages and disadvantages of Pikrit and compares it with selected protocols discussed in this thesis.

5.5.1 Compared to RPL

RPL and Pikrit differ greatly in design, scope and target application. RPL assumes more powerful network participants as it uses IPv6 addresses and therefore supports a vast amount of participants within a single network. Upward routing uses a clever DODAG based algorithm as described in Section 2.3.1 that achieves very good routing results. When using downward routing, however, RPLs approach has its weaknesses. Application developers have two options on how to counteract these. Firstly, the non-string mode uses source routing like Pikrit. But IPv6 addresses have a huge overhead in comparison to Pikrit routing addresses and therefore this mode reduces payload sizes due to large headers. Secondly, the storing mode assumes that nodes have a lot of memory to remember routes and directions similarly as described in Section 2.2.2. This means RPL focuses on being a sensor-friendly network as upward routes are handled elegantly. Downward routes on the other hand have their own separate mechanism which might imply organizational, computational, as well as data overhead and therefore the protocol is not very attractive when creating mostly actor-based networks.

5.5.2 Compared to Z-Wave

Pikrit and Z-Wave both share a common basis in terms of design. They are both source routing protocols and statically limit the maximum hop count. This is where the similarities end, though. While Pikrit provides up to 65535 custom ADIDs the Z-Wave protocol specification has a large collection of command classes for different device types. The primary controller of Z-Wave might also be a portable device. Exploration frames are sent in the emergency case that a node is not found which takes a lot of time exactly when the user is expecting an action to occur. For this reason, mobile coordinators are not supported by Pikrit. Also, Pikrit focuses more on having reliably fast connections for urgent, safety related control commands to be dealt with in a timely manner. Network separation is another differentiator. Z-Wave uses a separate unique home ID to identify which network a node is part of, while Pikrit will handle separation with the randomly generated encryption key.

5.5.3 Compared to LOADng

LOADng is a project which tries to modernize and improve the AODV protocol to enable its use in resource-constrained environments. For a generic way to handle data communication this protocol is very potent. But the potentially lengthy time in order to broadcast a RREQ when trying to route frames makes it unsuitable for time critical tasks.

5.5.4 Compared to Contiki Rime

While Contiki's Rime protocol is an easy to-use and flexible protocol built into the Contiki OS its main disadvantage clearly is the timely delivery of routed frames. As shown in Section 5.2, the time it takes to reach a certain node when using the Rime mesh implementations is rather long and varies greatly. On the other hand, no configuration is required for Rime in order to provide a routing algorithm. Therefore in cases where the timeliness of actions is not tight, it can be an easy applicable solution to extend the range of a wireless network.

5.5.5 Source routing protocols

Usually source routing protocols have a footprint which is growing linearly with the hop count needed. As it does not have a high priority to route over distances greater than 400 m within home building automation, the distance is restricted to 4 hops. Nodes can be designed with very low cost components on both, the storage and computation front, because they just route the frame according to the static route which takes constant ($\mathcal{O}(1)$) for bounded hop counts. The small design of the routing address of 1 byte allows for a small footprint while supporting up to 254 devices within one network. Compatability with widely used protocols like TCP/IP is not directly given within Pikrit, but if the coordinator is chosen sufficiently powerful, it can act as a gateway for TCP/IP.

	Pikrit	**RPL (ZigBee IP)**	**Z-Wave**	**Loadng**	**Contiki rime**
Supports moving parts	no	partially	partially	yes	yes
Footprint size	constant, small (hop limit)	const. downward, pot. lin. upwards	constant, medium (hop limit)	large	constant, small
Routing speed	very fast	fast	very fast (not for moving parts)	slow (on demand)	very slow
Deterministic response time	yes	no	not for moving parts	no	no
Self-Organizing	with Coordinator	with root node	with primary controller	yes	yes
Hop count	statically defined	unlimited (in theory)	statically defined	unlimited (in theory)	unlimited (in theory)
Data Integrity	built in	built-in or custom	built in	not handled directly	not handled directly
Confidentiality	To be determined	built-in or custom	built in	not handled directly	not handled directly

Table 5.7: Feature matrix for comparison between popular protocols

Chapter 6

Conclusion and future work

"Ich vermute, dass wir nur sehen, was wir kennen."[1]
("I suspect that we only see what we know.")

6.1 Future of Pikrit

The successful conclusion of this thesis is the starting point for further investigation in including Pikrit into real world devices. Figure 6.1 shows an estimated timeline of the inclusion of Pikrit within the home automation product ONYX. Ultimately the goal is to finish this process within a timeframe of one year from the time this thesis is done. This would result in a first finished product at the beginning of the year 2018. Most of the steps which are required to release a finished product are already fully or in partially defined with this thesis. Of course details of their specification could change during the process of integration within ONYX, but the fundamental ideas are finished. The MFBDs specifications for example, can be seen can be seen in Section 3.5.4 and the node integration, which uses Contiki, is schematically described in Chapter 4.

6.1.1 Pikrit confidentiality plans

Further time has to be invested to achieve a sensible, lightweight and attack-resistent encryption mechanism. Confidentiality is intended to be an integral property of Pikrit. Foundations for this inclusion have already been laid out. There is, for example, the encrypted flag within the header of the protocol which indicates to participants whether the frame has to be decrypted. While this flag is intended to not

[1] [Nie+84]

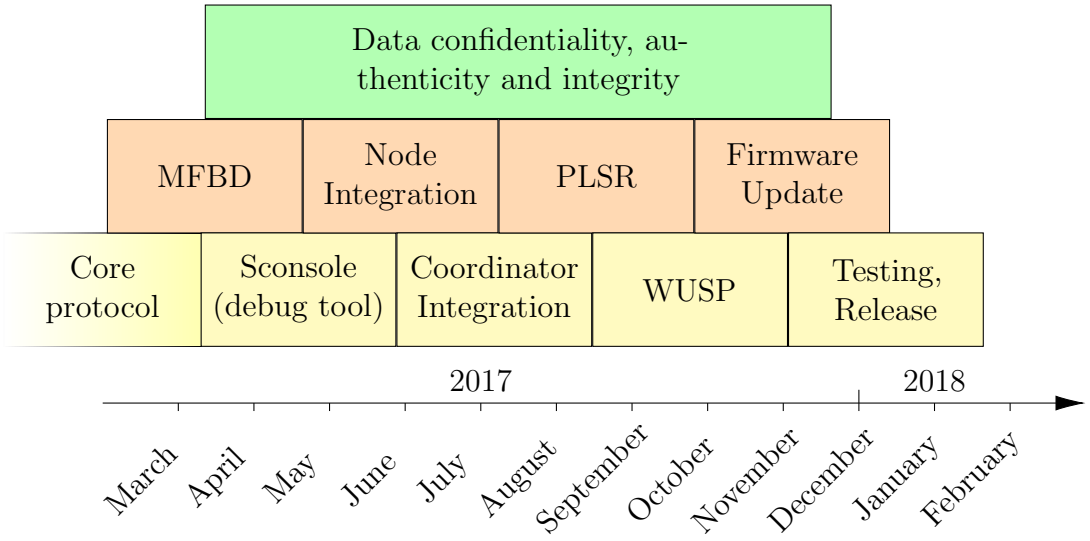

Figure 6.1: Timeline for Pikrit adoption within ONYX

be encrypted, the remainder of the message frame will be. Search requests currently do not have a defined payload. It is intended to include encryption information within this core Pikrit function. This initial key exchange is a particularly crucial part and has to be well thought out. But this alone is not sufficient because there are many stages in data transmission which can be abused to extract information.

6.1.2 Protocol enhancements

In order to widen the protocol to suite for a larger audience there are still a lot of areas to improve. The following list shall give a vague overview of possible improvements.

- **Broadcasting**
  In some circumstances it is convenient to send out a single frame which will be consumed by every participant in the network. Especially within wireless networks this should only be used for routing information in a single direction as responses could cause a lot of conflicts.

- **Multicasting**
  Similar to broadcasting, but with the ability to select a range of recipients to which information is broadcast.

- **Stream transport**
  A stream transport protocol allows an unspecified amount of data to be delivered over a route. Such a connection-oriented protocol only indicates with a single frame whether a channel is established or closed. In case of

connection problems a timeout might be necessary in order to help with cleaning up allocated resources.

6.2 Conclusion

Specifying a protocol is an effortful task which requires thoughtful iteration in order to achieve a usable implementation. It should be weighted carefully whether creating a custom protocol is worth the effort and the loss of compatibility. This thesis shows that the large amount of protocols currently available within the wireless embedded IoT device sector and the fragmentation they produce, justifies the creation of a new application-specific protocol. Focusing on being lightweight and reliably fast widens the ability to use Pikrit in more time critical applications. There is more to a protocol than being fast and reliable. Like many protocols in this area, Pikrit needs a coordinator in order to manage and maintain routes within a network. During maintenance work (PLSR), a huge amount of data transmission can occur, hindering regular data transfer. Choosing the right time to do this maintenance work is crucial for providing a seamless experience for users and safety applications alike. Other protocols, like RPLs trickle algorithm, spread out this maintenance work continuously throughout lifetime of the protocol. Maximizing the actual real world performance of Pikrit is, like most software implementations, an act of precise, astute and balanced tuning of the protocol's properties and its maintenance algorithms.

The future of IoT and embedded wireless devices looks very promising. Things are changing at a fast pace and innovations come and go day by day. Sooner or later developers will agree upon a common de-facto standard for wireless embedded routing. Such unification will simplify the life of end users, distributors and developers as the focus can be shifted more towards the actual communication of devices and not how they communicate. Only time can tell which protocol this will be, but challenging the existing proposals and implementations is a thrilling and enlightening undertaking. Generally speaking, this thesis highlights that the creation of a custom protocol in this area is a sensible choice for the time being.

Acronyms

6LoWPAN IPv6 over Low power Wireless Personal Area Networks 25
ADID Application Domain ID 47
AES Advanced Encryption Standard 42
AODV Ad Hoc on demand Distance Vector 26
API Application Programming Interface 34
ARM Advanced RISC Machines 45
ASCII American Standard Code for Information Interchange 56
CPU Central Processing Unit 60
CRC Cyclic Redundancy Check 44
CSMA Carrier Sense Multiple Access 32
DAO Destination Advertisement Object 23
DIO DODAG Information Object 23
DNS-SD Domain Name System - Service Discovery 25
DODAG Destination Oriented Directed Acyclic Graph 23
EEPROM Electrically Erasable Programmable Read-Only Memory 45
EULA End-User License Agreement 11
FIFO First In First Out 35
GDB GNU Project Debugger 37
GmbH Gesellschaft mit beschränkter Haftung 13
IEEE Institute of Electrical and Electronics Engineers 24
IETF Internet Engineering Task Force 23
IPv4 Internet Protocol version 4 31
IPv6 Internet Protocol version 6 23
IoT Internet of Things 29
LAN Local Area Network 39

List of Figures

List of Tables

List of Equations

Listings

Bibliography

[AK06] A. Anand and R. K. Kiran. "Optimized and adaptive link state routing strategy." In: *GCC Conference (GCC), 2006 IEEE*. Mar. 2006, pp. 1–5. DOI: 10.1109/IEEEGCC.2006.5686178 (cit. on p. 22).

[All14] ZigBee Alliance. *ZigBee IP Specification*. 095023. Revision 34. ZigBee Alliance. 2014 (cit. on p. 24).

[Asi+09] M. Asikainen, K. Haataja, R. Honkanen, and P. Toivanen. "Designing and Simulating a Sensor Network of a Virtual Intelligent Home Using TOSSIM Simulator." In: *Wireless and Mobile Communications, 2009. ICWMC '09. Fifth International Conference on*. Aug. 2009, pp. 58–63 (cit. on p. 37).

[Bac+13] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt. "RIOT OS: Towards an OS for the Internet of Things." In: *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*. Apr. 2013, pp. 79–80 (cit. on p. 34).

[Bel80] A. G. Bell. "Upon the production and reproduction of sound by light." In: *Telegraph Engineers, Journal of the Society of* 9.34 (1880), pp. 404–426. DOI: 10.1049/jste-1.1880.0046 (cit. on p. 19).

[CD11] James M Conrad and Alexander G Dean. *Embedded Systems: An Introduction Using the Renesas Rx62n Microcontroller*. Micrium Press, 2011 (cit. on p. 18).

[CHP11] T. Clausen, U. Herberg, and M. Philipp. "A critical evaluation of the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL)." In: *2011 IEEE 7th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. Oct. 2011, pp. 365–372. DOI: 10.1109/WiMOB.2011.6085374 (cit. on p. 23).

[CYV12] T. Clausen, J. Yi, and A. C. de Verdiere. "LOADng: Towards AODV Version 2." In: *Vehicular Technology Conference (VTC Fall), 2012 IEEE*. Sept. 2012, pp. 1–5. DOI: 10.1109/VTCFall.2012.6399334 (cit. on pp. 27, 28).

[KWL16] Srijan Kumar, Robert West, and Jure Leskovec. "Disinformation on the Web: Impact, Characteristics, and Detection of Wikipedia Hoaxes." In: *Proceedings of the 25th International Conference on World Wide Web*. WWW '16. Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee, 2016, pp. 591–602. ISBN: 978-1-4503-4143-1 (cit. on p. 13).

[Lel14] Adam Lella. *The U.S. Mobile App Report*. Tech. rep. 2014 (cit. on p. 19).

[Lu+03] Yi Lu, Weichao Wang, Yuhui Zhong, and B. Bhargava. "Study of distance vector routing protocols for mobile ad hoc networks." In: *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*. Mar. 2003, pp. 187–194. DOI: 10.1109/PERCOM.2003.1192741 (cit. on p. 21).

[McI09] A. I. McInnes. "Using CSP to Model and Analyze TinyOS Applications." In: *Engineering of Computer Based Systems, 2009. ECBS 2009. 16th Annual IEEE International Conference and Workshop on the*. Apr. 2009, pp. 79–88 (cit. on p. 35).

[Nie+84] F. Nietzsche, G. Colli, V. Gerhardt, M. Montinari, and M.L. Haase. *Nachgelassene Fragmente Juli 1882 - Winter 1883/84*. Walter De Gruyter Incorporated, 1984. ISBN: 9783110079050 (cit. on p. 90).

[PK09] T. Paul and G. S. Kumar. "Safe Contiki OS: Type and Memory Safety for Contiki OS." In: *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom '09. International Conference on*. Oct. 2009, pp. 169–171. DOI: 10.1109/ARTCom.2009.126 (cit. on p. 29).

[PR99] C. E. Perkins and E. M. Royer. "Ad-hoc on-demand distance vector routing." In: *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*. Feb. 1999, pp. 90–100. DOI: 10.1109/MCSA.1999.749281 (cit. on p. 26).

[Pre+14] T. Preiss, M. Sherburne, R. Marchany, and J. Tront. "Implementing dynamic address changes in ContikiOS." In: *Information Society (i-Society), 2014 International Conference on*. Nov. 2014, pp. 222–227 (cit. on p. 79).

[PS16] J. Pick and A. Sarkar. "Theories of the Digital Divide: Critical Comparison." In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. Jan. 2016, pp. 3888–3897 (cit. on p. 12).

[RS13] Kévin Roussel and Ye-Qiong Song. *A critical analysis of Contiki's network stack for integrating new MAC protocols*. Research Report RR-8776. INRIA Nancy, Dec. 2013, p. 13 (cit. on p. 33).

[RSZ16] Kévin Roussel, Ye-Qiong Song, and Olivier Zendra. "Using Cooja for WSN Simulations: Some New Uses and Limits." In: *EWSN 2016* -

NextMote workshop. Ed. by Kay Roemer. ACM. Graz, Austria: Junction Publishing, Feb. 2016, pp. 319–324 (cit. on p. 36).

[Shi+15] V. L. Shivraj, M. A. Rajan, M. Singh, and P. Balamuralidhar. "One time password authentication scheme based on elliptic curves for Internet of Things (IoT)." In: *2015 5th National Symposium on Information Technology: Towards New Smart World (NSITNSW)*. Feb. 2015, pp. 1–6. DOI: 10.1109/NSITNSW.2015.7176384 (cit. on p. 43).

[Ste+12] K. D. Stephan, K. Michael, M. G. Michael, L. Jacob, and E. P. Anesta. "Social Implications of Technology: The Past, the Present, and the Future." In: *Proceedings of the IEEE* 100.Special Centennial Issue (May 2012), pp. 1752–1781. ISSN: 0018-9219 (cit. on p. 10).

[Tho99] Mikkel Thorup. "Undirected Single-source Shortest Paths with Positive Integer Weights in Linear Time." In: *J. ACM* 46.3 (May 1999), pp. 362–394. ISSN: 0004-5411 (cit. on p. 65).

Index