Evaluating possible use cases of the decentralised packet transfer protocol with unmanned aircraft

Research Project Report
in the degree program Computer Science & Engineering
at the Faculty of Information, Media and Electrical Engineering
of the TH Köln - University of Applied Sciences

Submitted by: Janis Neumann 11448330
Submitted to: Prof. Dr. Grebe

Cologne, 30.04.2026

Abstract

Connecting devices in systems that can not utilize standard networking protocols is a significant challenge. To address the challenge, this research project presents and evaluates a protocol designed to connect devices that do not have a constant data link. The protocol enables data exchange by using intermediate devices as forwarding nodes and by temporarily storing packets. Furthermore, it incorporates methods to handle network formation, packet acknowledgment and security aspects. To display the protocol's capabilites and limitations, two experiments were conducted using multiple stationary ground nodes and a mobile node mounted on an unmanned aircraft. The results show that the protocol successfully connects devices without a constant data link and provides a realiable throughput, even over longer distances. However, the protocol also introduces significant packet delays.

Key words/key phrases: mesh networking, delay-tolerant networking, networking of unmanned aircarfts, decentralized packet transfer protocol

Contents

List of Figures **IV**

Acronyms **V**

1 Introduction **1**

2 Protocol Description **2**
2.1 Detailed Protocol Workflow . . . 3
2.1.1 Network Establishment . . . 3
2.1.2 Data Exchange . . . 4
2.1.3 Ending an Exchange . . . 5
2.1.4 Placing DPTP in the OSI model . . . 6
2.2 Example Workflow . . . 6
2.2.1 System Parameters . . . 7
2.2.2 Network Establishment . . . 7
2.2.3 Data Exchange . . . 9

3 Related Work **12**
3.1 Delay-tolerant Networks . . . 12
3.2 Mesh Networks . . . 12
3.2.1 B.A.T.M.A.N Protocol . . . 13

4 Protocol Implementation **16**
4.1 Transmitter Hardware . . . 16
4.1.1 Considered Options . . . 16
4.1.2 The nRF24l01 Modul . . . 17
4.2 Software Implementation . . . 18
4.2.1 Hardware Interface . . . 19
4.2.2 Packet Storage . . . 19
4.2.3 Device Storage . . . 20
4.2.4 Known Connections . . . 21
4.2.5 Cryptography . . . 21
4.2.6 Application Interface . . . 21

5 Performed Test **23**
5.1 Test Description . . . 23
5.1.1 Test Case 1 . . . 23
5.1.2 Test Case 2 . . . 23
5.2 Evaluating the Test Results . . . 24
5.2.1 Evaluation of Test Case 1 . . . 24
5.2.2 Evaluation of Test Case 2 . . . 26
5.2.3 Overall Evaluation . . . 28

6 Conclusion **29**
6.1 Possible Future Improvements . . . 29

Bibliography **30**

List of Figures

2.1 Nodes Positioning 2
2.2 Nodes Positioning with moving Node 2
2.3 Introduction Packet Structure 3
2.4 Data Packet Structure 4
2.5 Answer Packet Structure 5
2.6 Connection Ending Packet Structure 5
2.7 DPTP in the OSI Model 6
2.8 Node Positioning Example Workflow 6
2.9 Hexdump of ITN-PCK from node1 8
2.10 Hexdump of first ITN-PCK from node2 8
2.11 Hexdump of second ITN-PCK from node2 9
2.12 Hexdump of DAT-PCK from node1 9
2.13 Example Worklow Situation 1 10
2.14 Hexdump of AWR-PCK from node3 10
2.15 Example Worklow Situation 2 11
2.16 Hexdump of CED-PCK from node1 11

3.1 B.A.T.M.A.N Originator Messages Structure 13

4.1 Texas Instruments CC1190 16
4.2 HC-12-Modul 17
4.3 Hoperf Electronic RFM69HW 17
4.4 Nordic Semiconductor nRF24l01 18
4.5 nRF24l01 Packet Structure[1] 18
4.6 Packet Storage Structs 20
4.7 Device Storage Structs 20
4.8 Known Connections Structs 21
4.9 Application Interface fucntions 22
4.10 Alpha Implementation Flow Chart 22

5.1 Positions of the Nodes during the Tests 23
5.2 Test 1 results extract 24
5.3 Measured round-trip delay with a single recording for Test Case 1 25
5.4 Measured round-trip delay across three experimental runs for Test Case 1 25
5.5 Test 1 median delay 26
5.6 Test 1 median delay without DPTP 26
5.7 Test 2 results extract 26
5.8 Measured throughput with a single recording for Test Case 2 27
5.9 Measured throughput across three experimental runs for Test Case 2 28
5.10 Test 2 results numerical 28

Acronyms

AWR-PCK AnsWeR Packet. 4, 5, 10, 11, 21, 23, 24, 27, 29, IV

B.A.T.M.A.N Better Approach to Mobile Adhoc Networks. 13, 14, IV

CED-PCK Connection EnDing Packet. 5, 11, 21, 29, IV

DAT-PCK DATa Packet. 4, 5, 9–11, 29, IV

DPTP Decentralized Packet Transfer Protocol. 1–3, 6, 7, 12–14, 16, 18, 19, 21–24, 26–29, II, IV

DTN Delay tolerant networking. 12, IV

ITN-PCK InTroductioN Packet. 3, 4, 7–9, 20, 21, 29, IV

OLSR Optimized Link State Routing. 13, IV

PN Packet Number. 19, IV

RTT Round Trip Time. 23, IV

SPB Shortest Path Bridging. 13, IV

TCP Transmission Control Protocol. 27, 28, IV

TRILL Transparent Interconnection of Lots of Links. 13, IV

UDP User Datagram Protocol. 27, IV

1 Introduction

Networking protocols are essential for almost every modern IT system today. While many protocols for standard applications exist, some applications have highly specific requirements for their networking infrastructure and therefore can not rely on conventional solutions. Instead, specialized protocols are required to provide connectivity in such environments. One of these protocols is the Decentralized Packet Transfer Protocol (DPTP). In this project that protocol was further developed based on its initial version and implemented on wireless transceivers.

The implementation is available at: https://github.com/Janbehere1/DPTP

Design Goals of the DPTP The DPTP is designed to provide connectivity for nodes that rapidly and unpredictably transition between an active and inactive data link. It also should include mechanisms for dynamic network formation. To achieve this nodes should introduce themselves to other nearby nodes periodically and also inform about other nodes they currently know. The original purpose of the DPTP was to create connectivity between unmanned aircraft that move independently and frequently enter and leave the transmission range of another. It should also be able to communicate over longer distances than the transmission range by utilizing other aircraft in between. This connection should be able to transmit high level commands and transport the responses back to a controller or other nodes. For example, the DPTP should enable an unmanned aircraft to receive a command like "fly to these coordinates, take a picture and report back" by transmitting the packet with the command to another aircraft which then physically transport it into the range of the intended receiver and transports the response data back. However, the protocol should also be able to support similar systems through multiple configurable parameters depending on system requirements. Finally, it should not rely on pre-exchanged information between the nodes other than the protocol and its settings (this is important for address determination and security).

Testing DPTP To verify if the design goals are met by the implemented version of the DPTP, two experiments were conducted. The experiments were conducted to evaluate whether the protocol is able to perform in a real-world wireless environment. In addition they were used to measure how the protocol impacts latency and throughput. The results of these experiments provide insight into the practical capabilities and limitations of the DPTP, particularly in scenarios characterized by intermittent connectivity and dynamic network topology. They provide an overview about how the protocols works to help developers chosing if it is suitable for their application.

2 Protocol Description

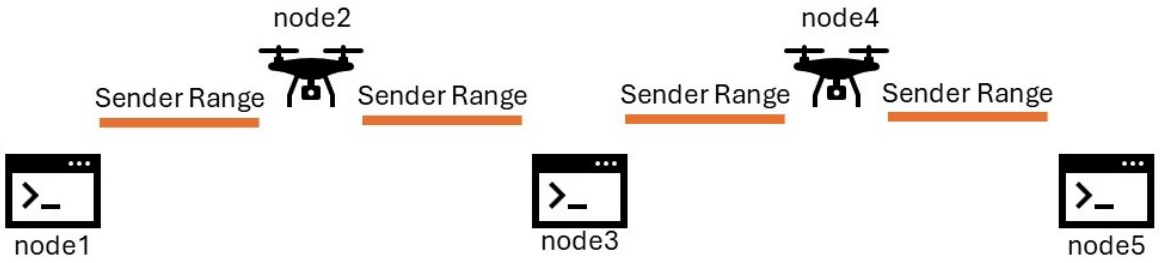

Figure 2.1: Nodes Positioning

The DPTP is a mesh network protocol that has the ability to store packets temporarily. Its main purpose is to provide connectivity between nodes that cannot communicate directly due to limited transmission range or a lack of compatible or overlapping network interfaces. For example, if one node has only wireless transmitting options while the other node only has wire-bound options, a third node with both interfaces could act as a bridge if all three deploy the DPTP. The figure 2.1 illustrates an example where all nodes have the same wireless communication option, but not all nodes are within range of each other. If `node1` intends to exchange packets with `node4`, it could use `node2` and `node3` as relay nodes as in other mesh networks.
A key feature of the DPTP that separates it from other mesh protocols is that it still operates if the nodes are not all in range of each other during the transmission of a packet.

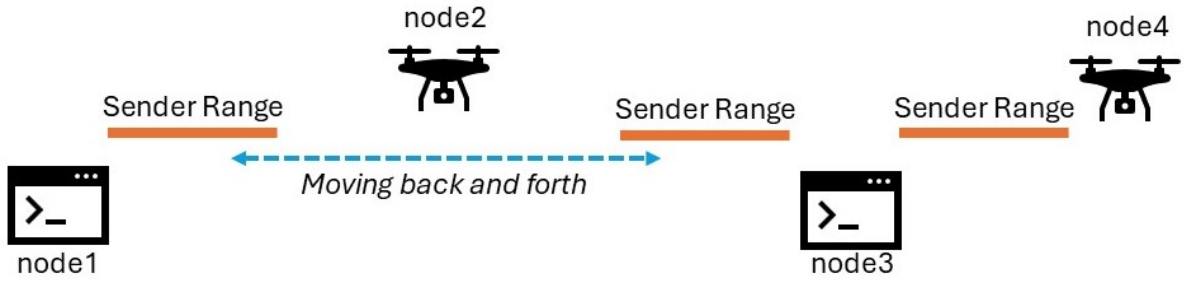

Figure 2.2: Nodes Positioning with moving Node

This can be achieved because the nodes that are not the receiver of a packet, store that packet temporarily and resend it after a fixed period of time. In the example of 2.2 `node2` receives the packet from `node1` while in range of it and retransmits the packet repeatedly. Once it comes within range of `node3`, `node3` receives one of these retransmissions and can then proceed to take the packet over to `node4`.

2.1 Detailed Protocol Workflow

When using the DPTP in an application or system, first a few setting specific parameters need to be defined:

- `Address Size:` How many bytes are used for the address of a node. The standard is eight, so the address space is 2^{8*8}, resulting in a negligible probability of collisions.
- `Packet ID Size:` How many bytes are used for a packet id which is (almost) unique for each packet in the network. The standard is also eight.
- `Standard Hop Number:` How often a packet should "jump" from one node to another until it is discarded. This can differ between the nodes in a network, but the standard is eight.
- `Max Time of Death:` A maximum for the duration how long a packet is stored before being deleted. This can differ between the nodes in a network, but the standard is 60 seconds.
- `Packet Repeat Interval:` A time delay between the retransmission of packets. This can differ between the nodes in a network, but the standard is 5 seconds.
- `Public Key Size:` The number of bytes used for the public key field of a node. The standard only implements ed25519 elliptic curves, therefore the standard public key length is 32 bytes.
- `Signature Length:` The number of bytes used for the signature of a packet. The standard only implements ed25519 elliptic curves, therefore the standard signature length is 64 bytes.

2.1.1 Network Establishment

Before exchanging data the nodes first need to establish a network. For this, each node generates a private/public key and a random address for itself. Depending on the expected network size the address size needs to be chosen large enough to reduce the chance of two nodes with the same address to almost zero. After that they transmit a so called InTroductioN Packet (ITN-PCK) which includes a list of all nodes currently known to the sender in the network (at the very least it knows itself). For each node, the following information is transmitted: The address, the public key, the standard hop number, a 32 byte long human readable node name and a timestamp when the last packet of the node was received.

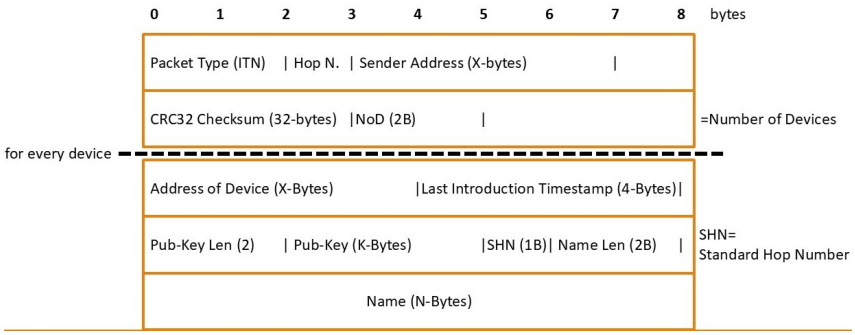

Figure 2.3: Introduction Packet Structure

All other nodes that receive such a ITN-PCK should also transmit one ITN-PCK with the nodes they currently know including the one they received the packet from. Through this a node that was not in range for the first node's ITN-PCK still learns about the existence of the network through the ITN-PCK of the second node. To make sure that ITN-PCKs don't repeat each other

infinitely, a small delay should be included that stops the sending of a ITN-PCK when another ITN-PCK was send a short amount of time before.

2.1.2 Data Exchange

After the network is established, a node can initiate a packet exchange by sending a so called DATa Packet (DAT-PCK). This packet includes among other fields a `receiver address` of the destination node, the `sender address` and a randomly chosen `packet ID` in addition to the application data. Receiving nodes use the packet ID to efficiently determine whether the packet has already been stored. If not, they store it and repeat it within their individual `Packet Repeat Interval` cycle. The sending node also stores and retransmits the packet.

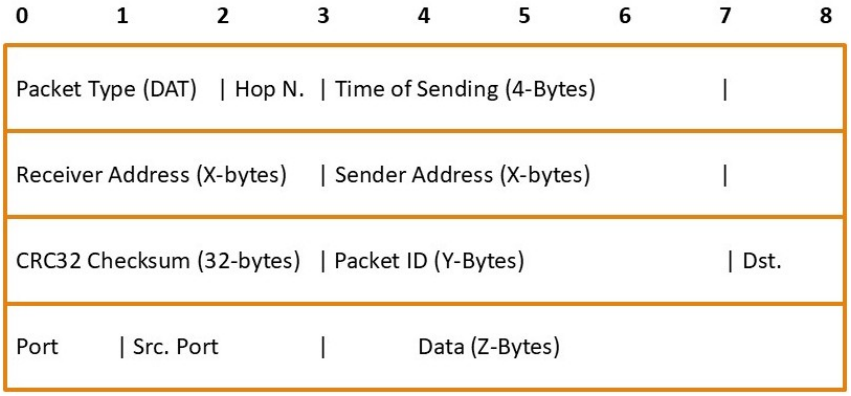

Figure 2.4: Data Packet Structure

Once a DAT-PCK arrives at the node with the `receiver address`, the receiver can process the data and store the packet ID in a list of already processed packets to ensure that if it receives a retransmission of this packet, it does not process it again. Once the receiver is ready to transmit application data back to the sender, it crafts a AnsWeR Packet (AWR-PCK). This is identical to the DAT-PCK, but with an additional so called `answer ID` field. In this field the packet ID of the previous received packet from that sender is written. The AWR-PCK is then transmited and retransmissioned just like the DAT-PCK before. The same goes for all nodes in between the sender and receiver that receive the packet. In addition to that, the nodes that still have the DAT-PCK stored know due to the packet ID in the answer ID field that the DAT-PCK packet has arrived at its destination and can delete it from their storage (this also includes the original sender). All further packets in this `connection` between the nodes is performed with AWR-PCKs that always include the packet ID of the previous received packet in the answer ID field.

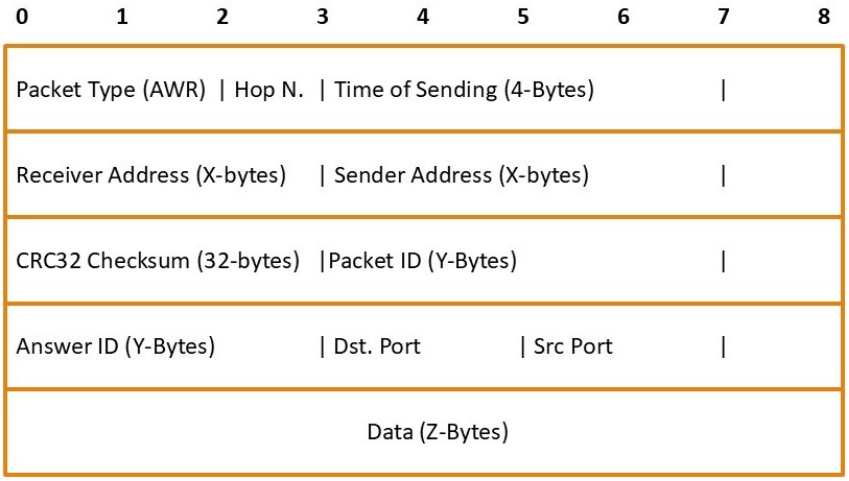

Figure 2.5: Answer Packet Structure

2.1.3 Ending an Exchange

Once one of the nodes decides that they don't need to exchange data anymore at the moment, it can issue a Connection EnDing Packet (CED-PCK). This packet includes the packet ID of the last received packet (like the AWR-PCK) and the ID of the last packet the sender of the CED-PCK transmitted to the other node. For CED-PCKs, each node retransmits it once immediately after receiving it and then stores it like a DAT-PCK or AWR-PCK. Through this the other nodes in range of sender/receiver and around the path the packets took between them know to delete the two packet IDs from their storage.

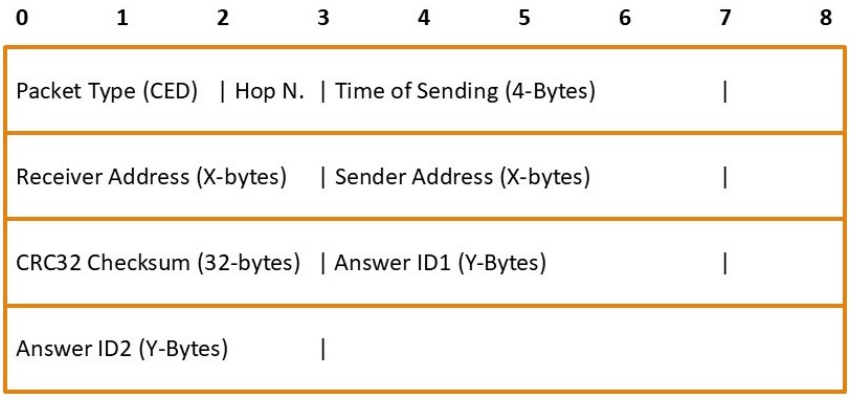

Figure 2.6: Connection Ending Packet Structure

2.1.4 Placing DPTP in the OSI model

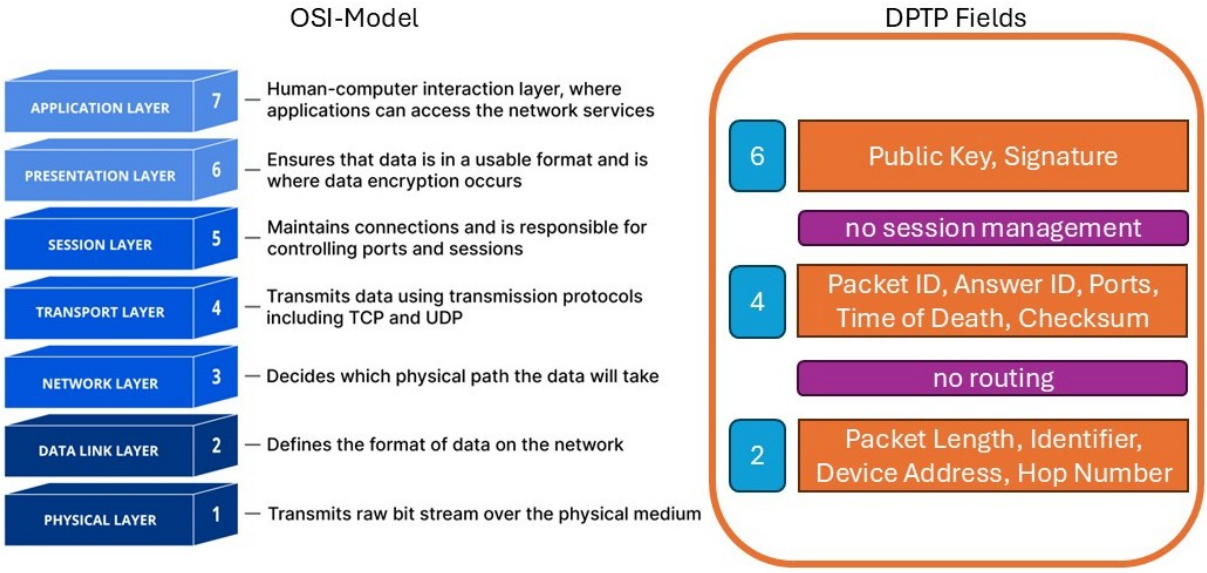

Figure 2.7: DPTP in the OSI Model

The DPTP covers multiple layers of the OSI Model. While separation of layers is essential for compatibility and interoperability in applications used by many people/nodes from different manufacturers, it introduces redundancy. For example in a standard Ethernet/IPv4/UDP packet there are at least two fields for the packet length and three checksum fields. Because the DPTP is a more edge case protocol, combining all required layers into a single protocol saves a couple of bytes overhead and reduces the CPU time by reducing the number of checksum calculations.

2.2 Example Workflow

To better illustrate the workflow of the DPTP an example application that uses this protocol is described in this section together with possible parameters for it. For this a stripped version of the situation used at the beginning of the chapter is used, where `node1` wants to exchange data with `node3`.

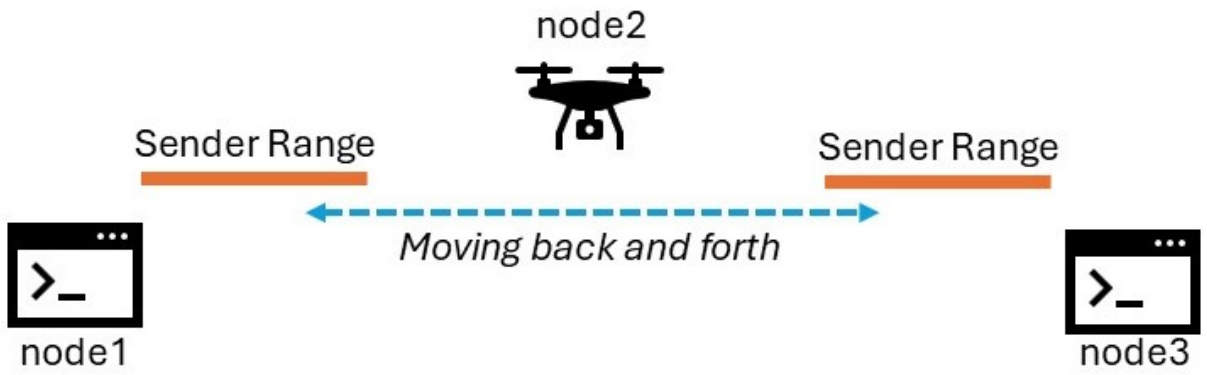

Figure 2.8: Node Positioning Example Workflow

2.2.1 System Parameters

In this example the following settings are chosen:

- `Address Size:` 1 byte
- `Packet ID Size:` 2 bytes
- `Standard Hop Number:` 3
- `Max Time of Death:` 60 seconds
- `Packet Repeat Interval:` 5 seconds
- `Public Key Size:` 32 bytes (ed25519)
- `Signature Length:` 64 bytes (ed25519)

The settings are chosen like this to simplify some aspects in the explanation below. Especially the `address size` and `packet ID size` are very small for a real-life application.
For the rest of the example the following random generated values are assumed:

node1

- `Address:` 0xa1
- `Human Readable Name:` node 1
- `Public Key:` e4 b1 80 8e e6 14 51 cc 1c b8 e4 c2 71 c5 a6 91 a9 44 76 7a 1a 6b 41 dc dd 16 3b 41 f9 39 5b 01

node1

- `Address:` 0xa2
- `Human Readable Name:` node 2
- `Public Key:` 8c a3 8c 1b 6d 79 a0 b6 d2 da 92 5a 01 0c bb 0e f3 b9 9e a3 90 13 1a 35 53 f4 61 52 90 7e 71 2c

node1

- `Address:` 0xa3
- `Human Readable Name:` node 3
- `Public Key:` 04 41 ad 8e 40 78 07 fd 88 ab 29 02 8e 87 10 dc aa 62 59 45 c8 33 90 0e 98 73 1e f3 1c 4c 0d 24

2.2.2 Network Establishment

The first step to exchange data with the DPTP is to establish a network. For that `node1` and `node3` start emitting an ITN-PCK every few seconds. The ITN-PCK of `node1` looks like this:

```
00000000: 0003 03a1 3a56 5b4d 0001 a169 db7d 8f20  ....:V[M...i.}.
00000010: 6559 627c b1a5 b5f4 036b 9bcd 616f dd1e  eYb|.....k..ao..
00000020: dd21 b867 f4ab 6177 92cb 9bbc 5098 bc77  .!.g..aw....P..w
00000030: 0306 6e6f 6465 2031                      ..node 1
```

Figure 2.9: Hexdump of ITN-PCK from node1

- First two Bytes {00 03}: Define the packet type for a parser, in this case as ITN-PCK
- Third Byte {03}: Hop Number of the packet
- Forth Byte {a1}: Address of the sender, for node 1 = 0xa1
- Fifth to Ninth Byte {1e e2 6a 50}: CRC32 Checksum of whole packet
- Ninth to Eleventh Byte {00 01}: Number of nodes in this packet
- Eleventh Byte till the end: Description of each node (in this case only one node, `node 1` itself)
  - Eleventh Byte {a1}: Address of known node
  - Twelfth to 16. Bytes {69 da e2 b0}: Timestamp when the last packet from that node was received
  - 16. Byte {20}: Length of the public key, here 0x20 = 32 bytes
  - 17. Byte to 49. bytes {e4 ... 01}: Public Key of that node
  - 49. Byte {03}: Standard Hop Number of that node
  - 50. Byte {06}: Length of human readable name, here 0x06 = 6 bytes
  - 51. to 57. Bytes {6e 6f 64 65 20 31}: Name of the node, here "node 1"

Once `node2` moves within range of `node1`, it receives this packet and responds with the following packet:

```
00000000: 0003 03a2 09c0 c53b 0002 a269 db7d 8f20  .......;...i.}.
00000010: ef86 b734 5077 2f21 dad3 12bf 4858 b003  ...4Pw/!....HX..
00000020: 85ca 31b3 5a99 d110 f417 73e0 74ef fa93  ..1.Z.....s.t...
00000030: 0306 6e6f 6465 2032 a169 db7d 8f20 6559  ..node 2.i.}. eY
00000040: 627c b1a5 b5f4 036b 9bcd 616f dd1e dd21  b|.....k..ao...!
00000050: b867 f4ab 6177 92cb 9bbc 5098 bc77 0306  .g..aw....P..w..
00000060: 6e6f 6465 2031                           node 1
```

Figure 2.10: Hexdump of first ITN-PCK from node2

Here the packet structure is the same as in figure 2.9, but in addition to the information about itself, `node2` adds the extracted information about `node1` into the ITN-PCK. With this packet `node1` now also knows `node2`.
After moving away from `node1` and in the range of `node3`, `node2` receives an ITN-PCK from it, adds the information about `node3` into its storage and emits this packet:

```
00000000: 0003 03a2 8691 e780 0003 a269 db7d 8f20  ...........i.}.
00000010: ef86 b734 5077 2f21 dad3 12bf 4858 b003  ...4Pw/!....HX..
00000020: 85ca 31b3 5a99 d110 f417 73e0 74ef fa93  ..1.Z.....s.t...
00000030: 0306 6e6f 6465 2032 a169 db7d 8f20 6559  ..node 2.i.}. eY
00000040: 627c b1a5 b5f4 036b 9bcd 616f dd1e dd21  b|.....k..ao...!
00000050: b867 f4ab 6177 92cb 9bbc 5098 bc77 0306  .g..aw....P..w..
00000060: 6e6f 6465 2031 a369 db7d 8f20 7006 abb4  node 1.i.}. p...
00000070: 89ae f381 e42b e427 e55a 1a40 f43f 4744  .....+.'.Z.@.?GD
00000080: da91 0574 30e7 1b04 d92b 9652 0306 6e6f  ...t0....+.R..no
00000090: 6465 2033                                de 3
```

Figure 2.11: Hexdump of second ITN-PCK from node2

With that `node3` learns about the existence of `node2` and `node1` together with their parameters (public key, name, ...). When `node2` then moves back into the range of `node1` and they exchange their ITN-PCKs, `node1` also learns about `node3` and the network is established with all nodes even so `node3` and `node1` never directly communicated.

2.2.3 Data Exchange

In the next step the data exchange of `node1` and `node3` can begin. In this example the assumtion is made that `node1` wants to request sensor data from `node3`. For this `node1` creates a DAT-PCK with the request for sensor data, emits that packet once and then adds it to its own packet storage so it gets retransmissioned. Even if `node2` does not receive the initial transmission of the packet, it receives it once it is in range of `node1` again. The DAT-PCK looks like this:

```
00000000: 0001 0369 db7d cba3 a1c3 885b 8876 0900  ...i.}.....[.v..
00000010: 0100 017b 2272 6571 7565 7374 223a 2022  ...{"request": "
00000020: 7365 6e73 6f72 2064 6174 6122 7db4 c3d9  sensor data"}...
00000030: 7103 dd11 cd98 be8d b1d6 9e2d c5a9 f716  q.........-....
00000040: d903 a687 450d 1806 43ba 5097 7481 2c60  ....E...C.P.t.,‘
00000050: add1 c6a5 d581 febf b5ff 5fb9 0e24 dfd5  ..........._..$..
00000060: c8aa e1b9 6cf3 15b2 e831 a74d 0d         ....l....1.M.
```

Figure 2.12: Hexdump of DAT-PCK from node1

Similar to the ITN-PCKs above it starts with an identifier for the packet type and the hop number. Following that a four byte long time of death (in seconds) is appended. This is the time when the packet was send plus the sender node specific `Max Time of Death` (here 60 seconds). The receiving node can either use this time of death for the packet or use the timestamp upon receiving the packet and adding the own `Max Time of Death` (usually it uses the smaller one). After this the receiver address {a3} and the sender address {a1} are added. The next four bytes are used for a CRC32 checksum followed by the packet ID {76 09} and the destination/source port (both are set to 1 here). At the end only the application data itself and the 64 byte signature of the whole packet are found. The signature is generated from the private key of the sender, here `node1`, and can thus be verified by other nodes with the public key of it. This hinders one node to impersonate other nodes in the network.
Once the packet was received by `node2`, the situation looks like this:

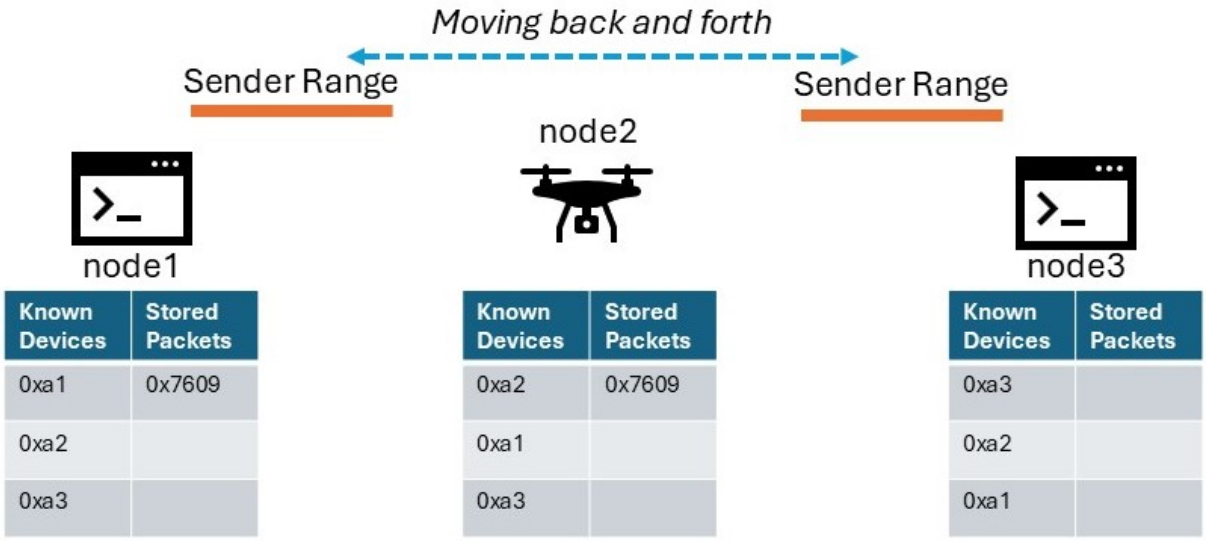

Figure 2.13: Example Worklow Situation 1

Then, after `node2` has moved back into the range of `node3` and retransmissions the DAT-PCK, `node3` can determine based on the receiver address that the packet is intended for it. After processing the received application data, it transmits an AWR-PCK with the sensor data:

```
00000000: 0002 0369 db87 e7a1 a37a fb51 2f54 e376  ...i.....z.Q/T.v
00000010: 0900 0100 017b 2274 696d 6573 7461 6d70  .....{"timestamp
00000020: 223a 2022 3230 3236 2d30 342d 3132 5431  ": "2026-04-12T1
00000030: 303a 3135 3a32 335a 222c 2022 7465 6d70  0:15:23Z", "temp
00000040: 6572 6174 7572 655f 6322 3a20 3231 2e38  erature_c": 21.8
........:........................................
000000b0: 7263 656e 7422 3a20 3738 2c20 2273 7461  rcent": 78, "sta
000000c0: 7475 7322 3a20 226f 6b22 7d77 f8e7 d406  tus": "ok"}w....
000000d0: 7357 330d 8a80 a4cf 1355 a8c0 63a9 03bc  sW3......U..c...
000000e0: 2bea 6d50 b14c cf53 6f0d f6c8 edfa 71db  +.mP.L.So.....q.
000000f0: d7f9 02a5 d1db 5382 0cfe b122 c10a e58d  ......S....".....
00000100: f15f 15ab bb1f 7bba 7934 0c              ._....{.y4.
```

Figure 2.14: Hexdump of AWR-PCK from node3

This packet has the same structure as the DAT-PCK with the addition of the answer ID field after the packet ID field (16th and 17th byte here, {76 09}). This field can be parsed by other nodes that receive the packet to know that the packet with that ID has reached its destination and can be deleted from the packet storage. With this the figure 2.13 changes to this:

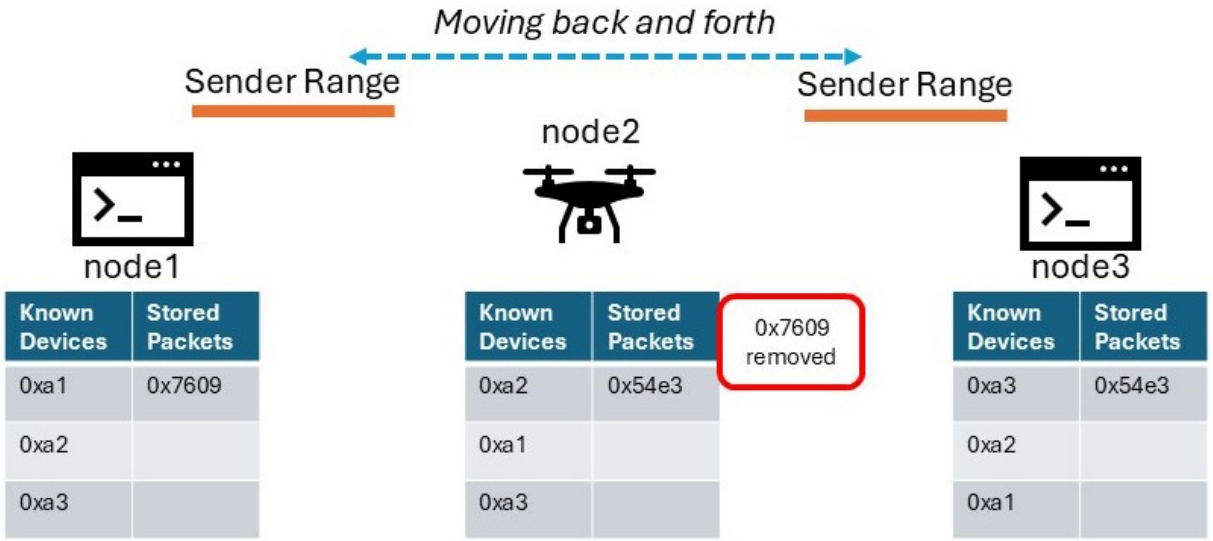

Figure 2.15: Example Worklow Situation 2

Once `node2` moves back into the range of `node1`, this AWR-PCK is transmitted to `node1` together with the requested sensor data. With that packet `node1` receives the requested data and knows that the packet it send reached `node3` due to the answer ID field. If `node1` then decides that it does not need anything more from `node3`, it sents a CED-PCK to end the connection:

```
00000000: 0004 0369 db91 b1a3 a11e 9fad ff76 0954  ...i.........v.T
00000010: e354 2e30 d410 77ce 1eab 9606 aa86 04ca  .T.0..w.........
00000020: 3b11 26f6 1ca8 4a0e b0d5 284d d546 c003  ;.&...J...(M.F..
00000030: 658d 7346 d4eb 3824 4d5a 9196 41cb b886  e.sF..8$MZ..A...
00000040: 2184 f365 bdd3 9549 b0c6 ef0c 0dae 78be  !..e...I......x.
00000050: 04                                       .
```

Figure 2.16: Hexdump of CED-PCK from node1

Here the packet structure with identifier and hop number at the beginning followed by a time of death, receiver address, sender address and CRC32 checksum is the same as for a DAT-PCK/AWR-PCK. The difference is, that in a CED-PCK no field for payload data is provided. Only two answer ID fields for the packet ID of the last received and the last send packet in a connection. Through this all nodes on the "route" are informed to delete those two packets from their storage.
After sending this packet, `node1` signals `node3` that the data exchange has ended and the packet storage of all nodes eventually become empty.

3 Related Work

Non-hierarchical network structures have been well established and are widely used in practice. In addition the principle of networks that store and physically carry data under circumstances where a constant working data layer is not present have also been studied. They are known as Delay tolerant networking (DTN) networks.

3.1 Delay-tolerant Networks

"The DTN architecture is a generalization of the interplanetary Internet defined by NASA."[4] Alongside this definition, NASA introduced the Bundle Protocol which received its seventh version in 2022.[1] The focus of this protocol is to provide connectivity between nodes that do not have a continuously available direct data link. For that it provides a way for nodes to store larger data units, referred to as `bundles`, and transfer them to other nodes which at a later time do the same so the `bundles` may at some point reach their destination.

Details of the Bundle Protocol While this approach introduces significant delays, it enables communication between nodes in urban or even interplanetary regions through carrier nodes that physically move between network regions from the area of one node to the next. In addition to the physical movement the definition of DTN networks encourages that the stored data is held in persistent storage so that nodes can even shut down for some time while being moved. While this approach provides advantages in regard to reliability, it would significantly reduce the processing speed when forwarding data while being connected to both sender and receiver at the same time (as illustrated in Figure 2.1). For this the reference does mention the possibility of mechanisms for routing, but explicitly excludes this from the document: "This document does not address: [...] Mechanisms for populating the routing or forwarding information bases of bundle nodes".[1] For the identification of nodes in the network, the Bundle protocol uses Endpoint IDs that are defined in a registry which all nodes in a network have locally stored or otherwise constant access to. To make the adding of new nodes easier, the Endpoint IDs have a URI scheme with domains and subdomains, so that new nodes can be added as a subdomain to a domain from the shared registry.

Comparison between Bundle Protocol and DPTP While the Bundle protocol and the DPTP have several similarities, a direct comparison is not really meaningful, because they are designed for different purposes: The Bundle protocol is made for applications where the nodes need to transfer data over longer periods of time (hours, days or even years) with travelling nodes that cross the range of other nodes infrequently. In comparison, the DPTP provides a way to transfer data between rapidly moving nodes that often move in and out the range of other nodes and mostly only need to carry the data for at most a few minutes to hours.

3.2 Mesh Networks

Mesh networks are more commonly used than DTN networks. At its core mesh networks are devices connected non-hierarchically that can exchange data directly without the need of a

master/slave principle. They are used in setups where a simple and dynamic configuration is valued as well as for basic connectivity between multiple nodes. Some examples for mesh networks are ethernet hubs, switches and Wi-Fi repeaters/access points.

Mesh Networks techniques For the data exchange there are two basic techniques: Flooding and Routing. With the first technique a node broadcasts a data frame to all reachable nodes and adds a unique identifier (mostly at the beginning of the frame) to allow receivers to determine whether the frame is intended for them or if they can discard the frame. These other nodes also transmit the data frame to all other nodes they can reach except for the one they got it from to avoid an infinite repeating of the same frame. This concept is, for example, used in network hubs. The routing technique on the other hand tracks which nodes are connected to which other nodes and establishes a path from the sender to the receiver (a so-called route). The sender then only sends the data frame to the first node on that route which after that also only forwards it to the next in the route. By this a lot less data frames need to be send/received compared to the flooding technique, but the nodes need to track the network-wise position of each node and edit the routes when they change. For this algorithems like Transparent Interconnection of Lots of Links (TRILL) and Shortest Path Bridging (SPB) are often used, for example in ethernet switches.

Comparison between the principle of Mesh Networks and DPTP With the principle of dynamic configuration with the randomly chosen sender IDs and the repeating of received packets that are not meant for a node itself, the DPTP falls under the category of a flooding based mesh network. But in addition to the data link layer, it also implements the transport and security layer. So in the end the DPTP is a mesh network with additions.

3.2.1 B.A.T.M.A.N Protocol

This protocol was first published in 2008 as an alternative for the previously widely used Optimized Link State Routing (OLSR) protocol.[6][7] It tackled the problem of the high CPU requirements of OLSR in networks with a large number of nodes. Both protocols are used to acquire routes in adhoc networks between devices. While OLSR is based on the idea that every node always knows the best route to every other node in the network, the Better Approach to Mobile Adhoc Networks (B.A.T.M.A.N) protocol only requires each node to identify the neighboring node that provides the best route to every other node.

Originator Message [6][7] To inform the nodes in a network about a new node or to announce changes in the topology of a node, so-called `Originator Messages` are used. These messages have the following packet structure:

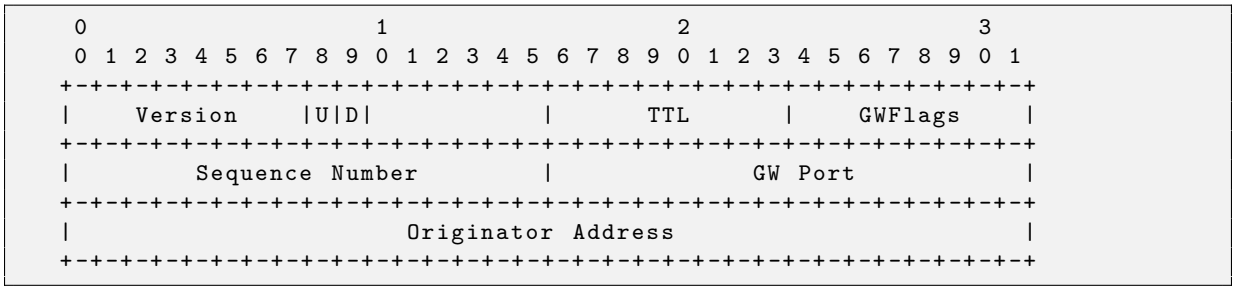

Figure 3.1: B.A.T.M.A.N Originator Messages Structure

- **Version:** Defines the Version of the B.A.T.M.A.N protocol. An unknown number here lets the packet be discarded.
- **Is-direct-link flag (U):** Set to 1 if the neighbor is a direct neighbor.
- **Unidirectional Flag (D):** Indicates if the neighbor can transmit and receive (bidirectional, 1) or not (0).
- **Time to Live (TTL):** The number of hops an Originator Message can make (not related to timestamps).
- **Gateway Flags (GWFlags):** If the Gateway is an uplink to the internet, the bandwidth is represented here. Otherwise the number is 0.
- **Sequence Number:** The number that uniquely identifies the Originator Messages in a given timeframe.
- **Gateway Port (GWPort):** Set to 0 if the node is not an internet gateway; otherwise, it specifies the tunneling port to use.
- **Originator Address:** The IPv4 or MAC address of the interface/node.

Route Calculation These Originator Messages are sent periodically from every node in the network and all other nodes that receive an Originator Message retransmit it to all neighboring nodes except the sender with the exception of the node from which they received the Originator Message. This is the flooding technique mentioned above. Using this all nodes in the network should receive the Originator Message at least once and, if they have multiple neighboring nodes, even multiple times. In this case they can determine the best route to the sender of the Originator Message by counting which neighboring node forwards the same Originator Message most frequently.
The advantage of this method is that not all nodes need to run complex algorithems and calculations to determine the whole, optimal route to a target node. Instead every node stores a neighboring node which is the best gateway for the specific target node and transmits the packets to that neighbor.

Comparison between the B.A.T.M.A.N Protocol and DPTP While routing like with the B.A.T.M.A.N protocol provides a smaller delay and fewer packets in the network for each application data packet, it also requires additional packets that do not carry application data. This can, depending on how many/often nodes move in the network topology, create much extra traffic only to keep the routes up to date. In comparison to that the DPTP does not have any routing mechanisms. Every packet transported with it is distributed with the flooding technique in the network. This means when packets need to be exchanged between nodes that have a few hops distance between them and multiple other nodes in range of these hops or the sender/receiver themselves, the number of packets in the network increases multiplicatively, while the B.A.T.M.A.N protocol only adds one extra packet for each hop independent of other nodes in range.
At the end routing is advantageous when a larger throughput with more packets is needed in a network and the advantages of plain flooding when fewer packets are sufficient or when the topology of the network changes very quickly through a larger number of fast moving mobile devices. In addition to that the B.A.T.M.A.N protocol assumes a constant working data link between the nodes in the network so that in the event a stored route does not work, a new one can be determined through `Originator Messages`. In contrast the DPTP explicitly handles the case that the data link between nodes in the network is interrupted. Due to this a direct

comparison between the two protocols would not be meaningful, because they do not operate in the same enviroment.

4 Protocol Implementation

To conduct the experiments with the DPTP and determine whether it can be used in a real-world application, an alpha version of the protocol was implemented.

4.1 Transmitter Hardware

Because most wireless modules like WLAN or Bluetooth interfaces in computers, mobile devices and even development boards do not allow the transmission of raw bytes, a special hardware module was required. This component had to be capable of transmitting (semi) raw bytes but concurrently handle RF aspects such as frequency selection and timing without much extra work from the developer.

4.1.1 Considered Options

For the project multiple wireless modules were evaluated. The following 4 made the shortlist:

Texas Instruments CC1190 [1] This module is a "low-cost sub-1 GHz transceiver designed for very low-power wireless applications".[5] It can operate on differnet frequencies between 300-348 MHz, 387-464 MHz and 779-928 MHz. One advantage of this module is that it can reach data rates of up to 600 kbps. But the main reason it was not chosen, is that no development board using this module was found within the budget. The module is mainly sold in a QLP 4x4 mm packet to be used in a PCB design, not on a development board.

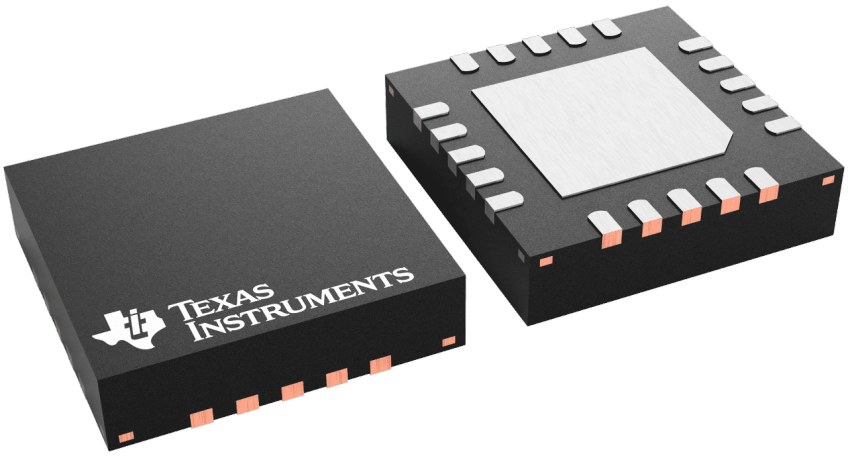

Figure 4.1: Texas Instruments CC1190

Silicon Labs SI4463 [2] The SI4463 module in contrast to the CC1190 was found together with a development board, the HC-12-Module [3]. It also uses a sub GHz frequency, the area 433.4-473

[1] https://www.ti.com/lit/ds/symlink/cc1101.pdf
[2] https://www.silabs.com/documents/public/data-sheets/Si4464-63-61-60.pdf
[3] https://www.elecrow.com/download/HC-12.pdf

MHz. With respect to cost the HC-12-Module was at the upper end of the budget, but the data rate with only up to 10 kbps was considered insufficient given its cost. In addition to that no usable example implementation could be found for the module.

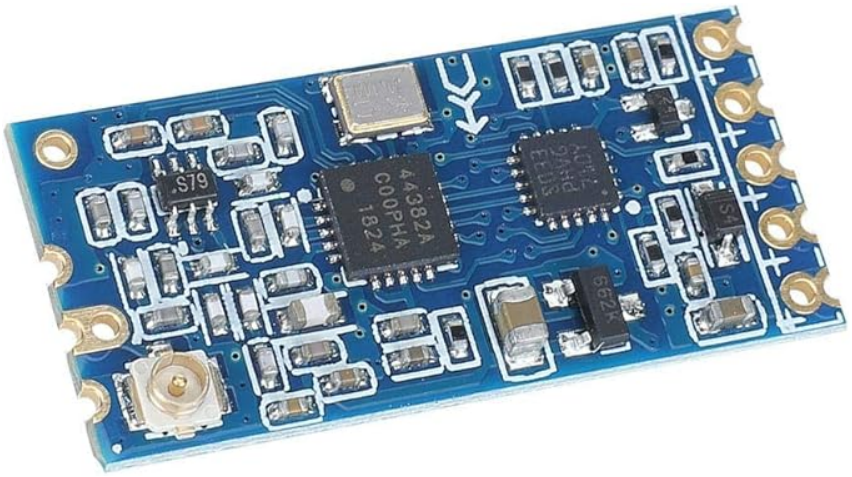

Figure 4.2: HC-12-Modul

Hoperf Electronic RFM69HW [4] This module was directly found as a development board that offers the standard open sub GHz frequencies (315 MHz, 433 MHz, 868 MHz and 915 MHz). With a data rate of up to 300 kbps, the module was not ideal but remained a viable option for this use case. But in the end the module was not used because again no usable example implementation was found.

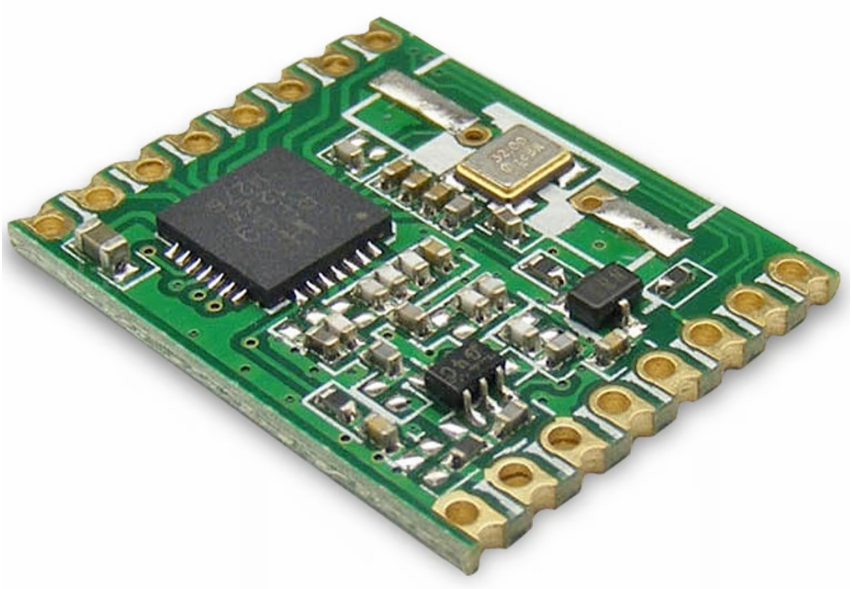

Figure 4.3: Hoperf Electronic RFM69HW

4.1.2 The nRF24l01 Modul

This module was used for the alpha implementation and the experiments in this project. The nRF24l01[5] is a half-duplex transceiver that operates on 2.4 GHz and can provide a data rate

[4] https://www.pollin.de/media/8e/5d/e4/1701728801/D810307-D.pdf
[5] https://cdn.shopify.com/s/files/1/1509/1638/files/NRF24L01_mit_2_4_GHz_Datenblatt.pdf

of 250 kbps, 1 Mbps or 2 Mbps. In addition to this higher data rates compared to the other modules, extensive documentation of wiring and libaries in different programming languages are available. Even examples with Arduinos and Raspberry Pi's are included in that documentation. Also, the price of this module was with around 2€-3€ the most cost-effective.

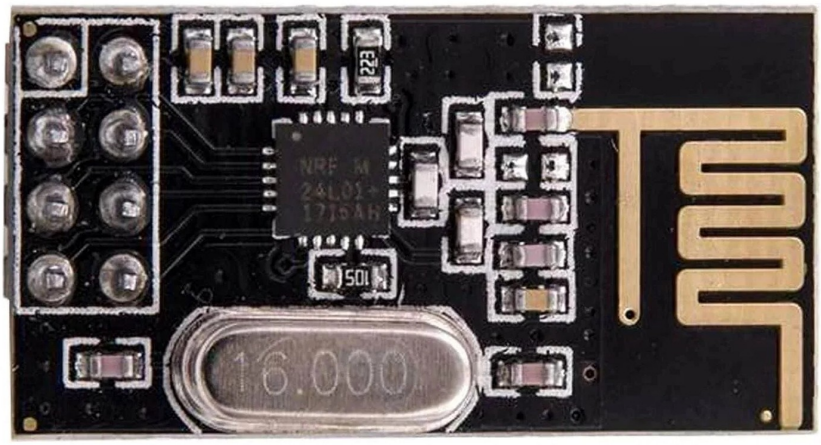

Figure 4.4: Nordic Semiconductor nRF24l01

The module works by sending data frames with 1 to 32 byte payload. In addition to the payload, each packet includes a preamble byte, three to five address bytes, nine bits control fields and a CRC8 or CRC16 checksum.

Figure 4.5: nRF24l01 Packet Structure[6]

The preamble byte is a fixed sequence to discern the start of a real packet from RF noise. The address bytes are used to separate multiple nRF24l01 connections that are within range of another. The sender and receiver must share the same address. The control bits include 6 bits for the payload length, 2 bits for a rolling packet ID and one bit to signal whether auto-ack packets are expected or not. These auto-ack packets can be used to automatically acknowledge received packets by the sender. For this the rolling packet ID is included in the next packet or if three packet IDs are collected before the original receiver wants to transmit a packet back, a separate packet is transmit for the acknowledgment.

4.2 Software Implementation

After choosing the hardware modul, the software implementation of the DPTP for the experiment could begin. As main development language `C` was chosen. The reasons for this were a high compatibility with almost every processor, the precise control of memory usage and the overall processing speed.[2]

[5]https://nrf24.github.io/RF24/
[6]https://github.com/nRF24/RF24/blob/master/datasheets/nRF24L01_datasheet_v2.pdf

4.2.1 Hardware Interface

Because the library for the nRF24l01 was programmed in `C++`, the wrapper for that library was also implemented in `C++`. This wrapper includes an `setupRadio()` function to initiate he modul, a sender function (`sendRawPacket()`) and a listener function (`listenForPackets()`) that accepts a callback to another function as input parameter which then is invoked with each fully received packet.

setupRadio() To activate the nRF24l01 module several parameters need to be set. The most important are `AutoAck`, `Retries` and `AddressWidth`. The first specifies whether the module should transmit auto-ack messages or not. The secound defines, if auto-ack is enabled, how many microseconds a sender should wait before retransmitting a small packet and how many times it should wait and retransmit before dropping the packet. The last parameter is used to define how many bytes are used for the address (between three and five). In the case of this implementation, the auto-ack functions are deactivated, because this does not work if more than two nodes within range of another share the same address (one node would acknowledge a packet and it would not be retransmissioned, even if the other receiver did not receive it correctly). This strongly increased the packet loss.

sendRawPacket() This function can be called with an array of bytes (char pointer) and the number of bytes that should be read from that array. The first four bytes of the array should be the length of the packet in big-endian format. The intention is that these bytes are a whole DPTP packet that needs to be transmitted and the length in the first four bytes is required so that a receiver knows how many bytes to read for the packet. Within the function a temporary 2 byte Packet Number (PN) is randomly generated. Afterwards, the byte sequence is divided into segments of up to 29 bytes and each section is together with the PN transmitted as a small packet. This approach allows the wrapper to handle the fragmentation and reassembly of packets with up to 2^{32} bytes.

listenForPackets() This function is supposed to be started in a separate thread to the main program to ensure that the main program can continue while received packets are reconstructed and processed. For the processing a pointer to a function can be used as input parameter for this function so that each time a packet is reconstructed, the input parameter function is invoked with the bytes of the packet in another thread (to keep the receiver thread free). For the reconstructing the PN and length information in the first 29-byte frame are used.

Besides these three main functions, several auxiliary functions and testing functions are included in the wrapper. by this the wrapper can be used to test and debug problems with the nRF24l01 modules. To combine the wrapper and nRF24l01 library with the rest of the program written in `C`, the wrapper and library is compiled as a shared library so that it can be included in the `C` code.

4.2.2 Packet Storage

Because one of the main design goals of the DPTP is to relay packets for other nodes even if they are not in range when a packet is received, a node needs to store packets. For this during the initiation of the program, a space in the heap is allocated. The number of bytes allocated can be controlled with the global variable `MAX_STORAGE_OF_PACKETS`, which is currently set to 64,000 bytes. For the storage itself these two structs are used:

```
struct Packet {
    uint32_t length;
    int32_t time_of_death;
    char packetID[PACKETID_SIZE];
    char *data;
};

struct StoredPackets {
    uint32_t totalSize;
    uint16_t numberOfPackets;
    struct Packet **packets;
};
```

Figure 4.6: Packet Storage Structs

When adding a new packet to the storage, it is first verified that the packet is valid and that its packet ID is not already within the storage. The reason why the `Time of Death` and `packet ID` are stored an additional time (in the struct and in the data itself) in the `Packet` struct, is so that when checking if a packet is already stored and if its `Time of Death` has been reached, the whole packet does not have to be parsed again. For verifying if the `Time of Death` of a packet has passed and for repeating it in a certain interval, a separate thread executing the `packetSender()` function is responsible. This function runs continuously and checks for each stored packet if the `Time of Death` has passed, then deletes it from the storage, or if not retransmissions it. After looping through all stored packets, the thread sleeps for a time determined by the global variable `PACKET_REPEAT_INTERVAL`.
In addition to the packet storage, a smaller `Processed Packets Storage` is used to ensure that packets for the node itself are not processed multiple times. This storage only stores the IDs of the processed packets in a rolling list with.

4.2.3 Device Storage

Similar to the packet storage, during initialization, memory is allocated for the device storage on the heap (the number of bytes is determined by `MAX_STORAGE_FOR_DEVICES`, currently 32,000 bytes). Also similar, the device storage consists of two structs used in tandem:

```
struct DeviceInfo {
    char deviceID[ADDRESS_SIZE];
    int timestamp;
    unsigned char standard_hop_length;
    char device_name[32];
    char pub_key[PUB_KEY_SIZE];
};

struct StoredDevices {
    uint32_t totalSize;
    uint16_t numberOfDevices;
    struct DeviceInfo **devices;
};
```

Figure 4.7: Device Storage Structs

This device storage also includes the information about the own device. Also, when adding a new device from an ITN-PCK to the storage, first a check is performed whether the new device ID is already used by a device in the storage. If not, it can be added without further checks.

Otherwise, the new public key is compared with the stored one. If they do not match, the new device is not added to the storage. But if they do, the other parameters of the already stored device like `Standard Hop Number` and `Device Name` are updated with the paramter of the new device with the same ID and public key. This is done so that a device can update its parameters if needed.

4.2.4 Known Connections

To track the last transmitted and received packet IDs for different communication partners, a small storage for those connection is implemented similar to the other two storages.

```
struct Connection {
    char device_id[ADDRESS_SIZE];
    char last_recID[PACKETID_SIZE];
    char last_sendID[PACKETID_SIZE];
};

struct Connections {
    uint8_t number_of_connections;
    pthread_mutex_t lock;
    struct Connection **connections;
};
```

Figure 4.8: Known Connections Structs

The information stored in these two structs are used when sending an AWR-PCK and when ending a connection with a CED-PCK. The main difference in the implementation to the other storages is, that for the connections struct a mutex thread lock is necessary. This is because unlike the other two storages, which only get written to by the packet processing thread, the connections storage gets written to by the packet processing thread (when receiving a packet and updating the last received ID) and by the main thread when sending a packet (the last send packet ID) or when ending a connection with a CED-PCK (deleting the connection). The mutex lock is therefore used to ensure that both threads do not write to the variable concurrently.

4.2.5 Cryptography

As described in 2.2.3, the public key of each node is used by the receiving node to verify the signature append to the packet by the sender. In theory almost every asymmetric encryption algorithm can be used for this, but algorithms with short public keys have the advantage that the ITN-PCKs and device storages are not significantly increased by it. For example when using a RSA-4096 key, the number of bytes for a single node in a ITN-PCK is increased by 480 bytes compared with a ED25519 key. The current implementation only supports ED25519 encryption, because it is one of the algorithms with the shortest keys that are considered secure today.[8] As implementation of this encryption mechanism the `Sodium`[7] library, which is a fork of the `NaCl`[8] library with additional usability functions, was used because it is open source and easy to use.

4.2.6 Application Interface

To simplify usage of the DPTP implementation, a shared library `libDPTP` was developed. This library handles almost every aspect of the DPTP and only exposes five functions to the application:

[7] https://github.com/jedisct1/libsodium/tree/master
[8] https://nacl.cr.yp.to/

```
struct ApplicationPacket {
    uint32_t data_length;
    unsigned char hop_number;
    char sender[ADDRESS_SIZE];
    short destination_port;
    short source_port;
    char data[];
};

struct ApplicationPacket* receive();
int send(char *data, uint32_t data_length, char receiverName[32],
    short destination_port, short source_port);
int endConnection(char device_name[32]);
void printKnownDevices();
void initiateProtocol();
```

Figure 4.9: Application Interface fucntions

These functions allow the application to initiate the DPTP, print which nodes are known, send/receive application data and end connections.
The whole process of the implementation is visualised in the following flow chart (a few details are not depicted):

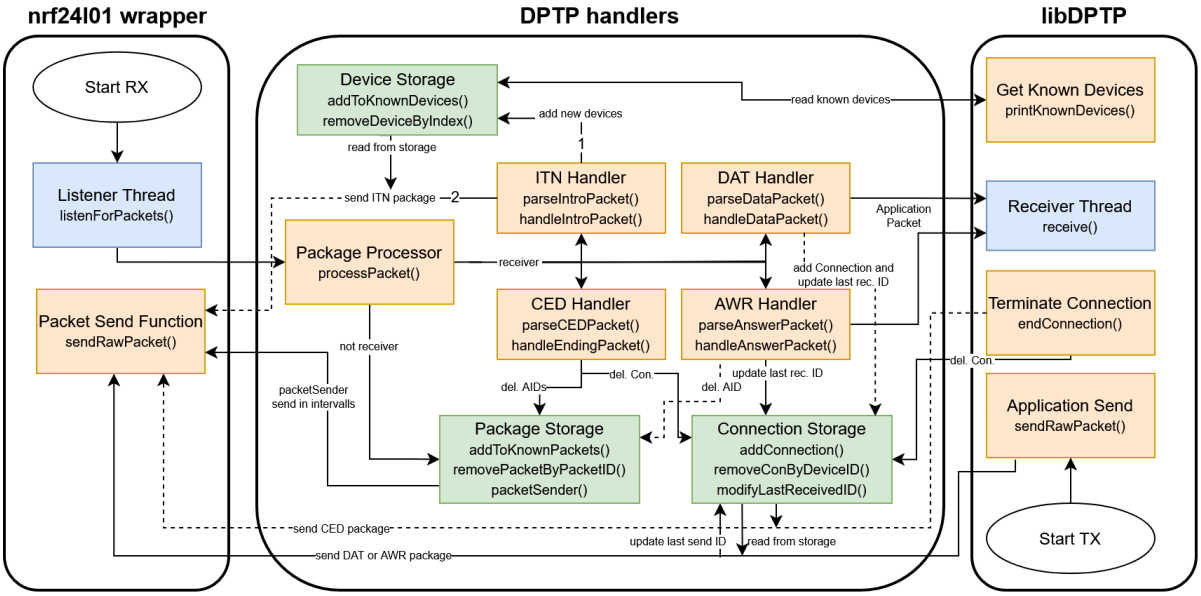

Figure 4.10: Alpha Implementation Flow Chart

5 Performed Test

After implementing an alpha version of the DPTP, tests were conducted to examine whether the protocol fulfills its purpose in a real-life application.

5.1 Test Description

For this two test cases were crafted. Their goal was to measure key network performance parameters with regard to the DPTP. Furthermore they were intended to demonstrate that the protocol can achieve its two main design objectives: Delay-tolerance and decentralised packet transfer (see 3).

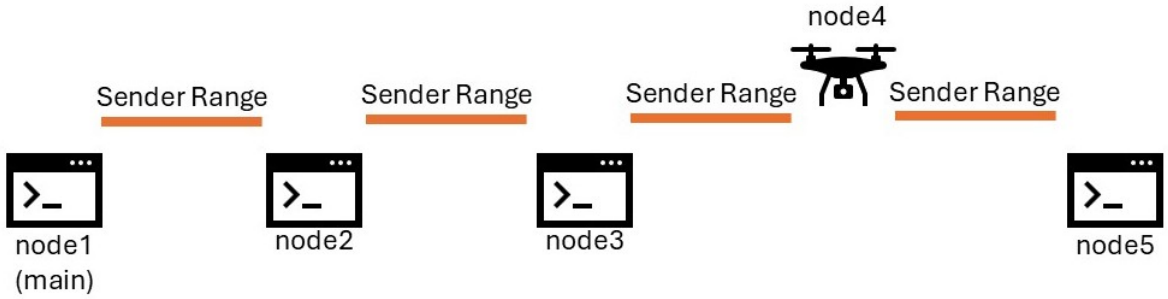

Figure 5.1: Positions of the Nodes during the Tests

5.1.1 Test Case 1

In the first test scenario, five nodes are arranged linearly as shown in Figure 5.1. Four of these nodes deploy the testing software in passive mode, while the fifth (`node1`) deploys it in active mode. The nodes in passive mode are configured to return the received application data unchanged.
Due to all five nodes running the DPTP as a service, all nodes get to know each other once they are powered on. After that `node1` starts to transmit 20 probe messages to each other node with a delay of two seconds between every probe. These messages include the current timestamp of `node1` as well as a counter. Once the packets reach the intended receiver, the receiver sends back an AWR-PCK with the same timestamp and counter. This allows `node1` to compute Round Trip Time (RTT) by comparing timestamps. This value is recorded along with the name of the passive node and the hop number of the received AWR-PCK in a JSON object. With this test the latency of the DPTP network can be tested. The results of the evaluation can be seen in section 5.2.1.

5.1.2 Test Case 2

The basic positioning of the nodes for the second test is the same as for the first one (see 5.1). While in the first test single packets were exchanged, the second test measures throughput that

the nodes can achieve using DPTP how it is effected by more hopping nodes. For this the nodes in passive mode don't transmit the received application data back to the sender, but instead transmit a packet containing eight bytes representing the size of the previous received packet and a counter. `Node1` in this case transmits 10 KB of test data divided into 256-byte blocks with the counter delayed by a two-second interval between transmissions to each node. The results are stored in a JSON object with the hop number, timestamp, numhber of transmitted bytes and counter. This allows measurement of how much time passed between the first and last packet and how many bytes were received and acknowledged.

5.2 Evaluating the Test Results

The tests were conducted in an open field to comply with drone flight regulations and to minimize interference. This include frequency interference by other nodes which use the 2.4 GHz band as well as interference of other people. For a more direct evaluation of the protocols properties and not the capabilities of the transceiver, a comparison of both tests with only two nodes and no add-on from the DPTP is used. In addition to that a software based guard was implemented to ensure that a node only uses packets that it would be able to receive by the design of the experiment. For example even if `node2` receives a packet directly transmitted from `node4`, it discards such packets to ensure that the results are not modified by such anomalies. To implement this, the feature of the `nrf24l01` module (see 4.1.2) to use addresses in the raw packets were used. With them each node got two so called `pipe-addresses` to which it should transmit its packets. Through this only the direct neighbors of a node could use these packets.
A total of three recordings of both tests were made (`2026_04_26-12_14_21`, `2026_04_26-12_27_43` and `2026_04_26-12_41_21`). Each time the first test was performed, then a 20 seconds delay was awaited and then the second test was made.

5.2.1 Evaluation of Test Case 1

```
// timestamp in ns and delay in ms
{"timestamp":1777198464664280956,"sender":"device2","hop_number":8,"id":1,"delay":1244}
{"timestamp":1777198467549538130,"sender":"device2","hop_number":8,"id":0,"delay":6517}
{"timestamp":1777198473017700069,"sender":"device2","hop_number":8,"id":2,"delay":7497}
```

Figure 5.2: Test 1 results extract

The results of the first test demonstrate that delay tolerance (3.1) and mesh network functionality (3.2) of the DPTP is achieved. A packet which was not immediately received by the receiver, was later retransmitted by the `packetSender()` multiple times and with one of these retransmissions reached the next node. Even if a node has been deactivated or out of reach for a short time, it still receives the collection of packets when active again (until the `Time of Death` of the packets is reached). This is also the reason why some packets have a much higher delay than others.
To analyse the results of the first test, a 2D graph was used which depict delay as a function of time when the AWR-PCK was received. In addition to that the different nodes are distinguished by color. Due to the experiment's setup with a line on which all nodes were placed, the hop number directly corresponds to the number of nodes between the main node (`node1`) and the receiving node.

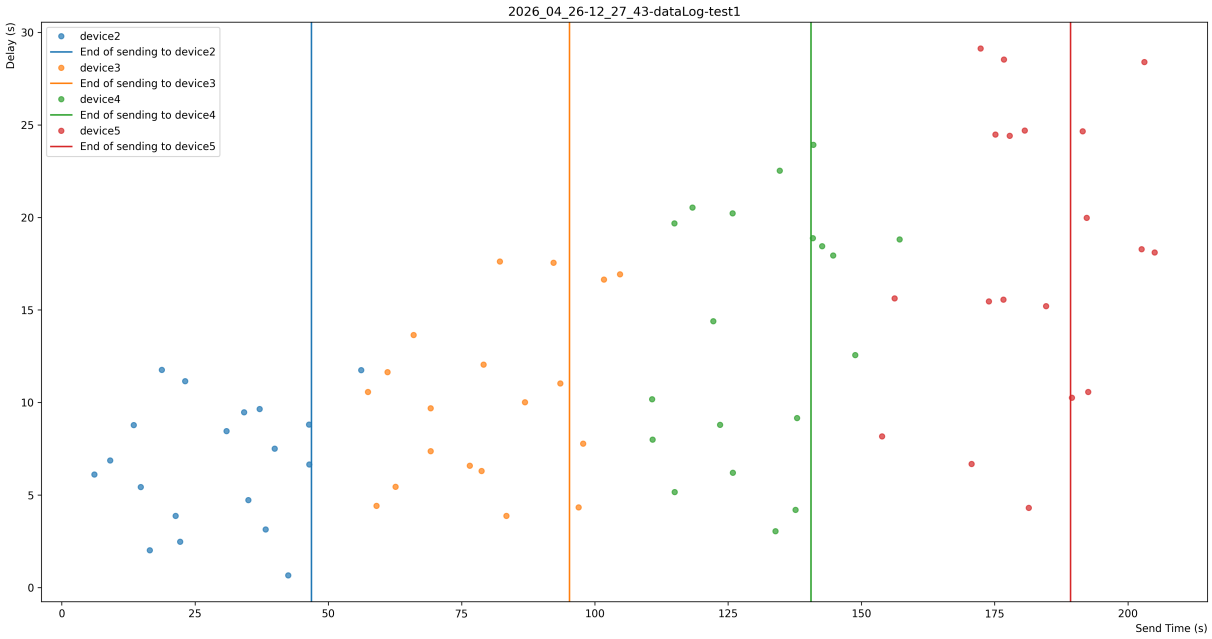

Figure 5.3: Measured round-trip delay with a single recording for Test Case 1

The vertical lines represent the moment when `node1` was finished sending the packets to the node with the same color.

The graph shows that the overall delay increases with increasing distance between nodes to the sender. This is consistent with expectations, because if a packet has to pass through more nodes, for each one it has to wait for the next repeat interval to potentially be transmitted to the next node (see 2.1). Comparing this recording to the others supports this phenomenon.

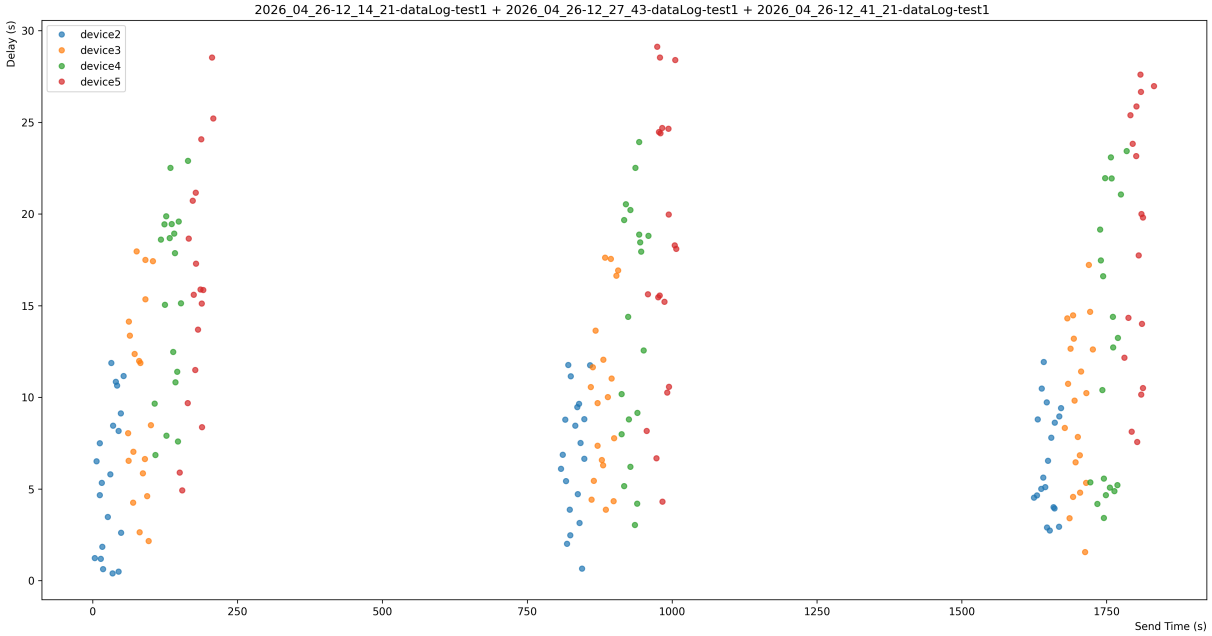

Figure 5.4: Measured round-trip delay across three experimental runs for Test Case 1

To further support the result of an increasing delay based on the number of hops, the median delay for each node with all recordings was determined.

```
device2: [0.403,0.505,0.637,...,6.111,6.517,6.542,...,11.766,11.879,11.942] => 6.517s
device3: [1.57,2.169,2.644,...,9.831,10.022,10.242,...,17.553,17.622,17.968] => 10.022s
device4: [3.044,3.422,4.19,...,14.402,15.048,15.136,...,23.103,23.439,23.928] => 15.048s
device5: [4.314,4.931,5.896,...,15.893,17.293,17.751,...,28.538,28.541,29.131] => 17.293s
```

Figure 5.5: Test 1 median delay

In addition to the median delays with the DPTP, the median delay when only transmitting directly between two transceivers was measured to compare:

```
direct: [0.142,0.144,0.153,...,0.163,0.165,0.165,...,0.175,0.177,0.181] => 0.165s
```

Figure 5.6: Test 1 median delay without DPTP

These values are smaller than the lowest delays of device2. This can be explained with multiple factors. First, the interference at a node is higher because it receives data frames from two nodes instead of one, resulting in a higher frame loss. Because the `nrf24l01` only supports 1 MHz of channel width, this interference is especially strong. These value also roughly match the lower end of the delays to device2, which was reached without any intermediate nodes.

5.2.2 Evaluation of Test Case 2

```
// timestamp in ns
{"timestamp":1777198673525853009, "sender":"device2", ..., "id":1, "numOfBytes":256}
{"timestamp":1777198674894014583, "sender":"device2", ..., "id":0, "numOfBytes":256}
{"timestamp":1777198678951458860, "sender":"device2", ..., "id":2, "numOfBytes":256}
```

Figure 5.7: Test 2 results extract

The results of the second test further support the statement that the DPTP is working as designed. The packets for a node that was not reachable from the sender were forwarded through other nodes. The following graph shows the cumulated number of bytes the receiver node reported back after getting a packet from `node1`. Each line is made of points representing the cumulative transmitted bytes over time.

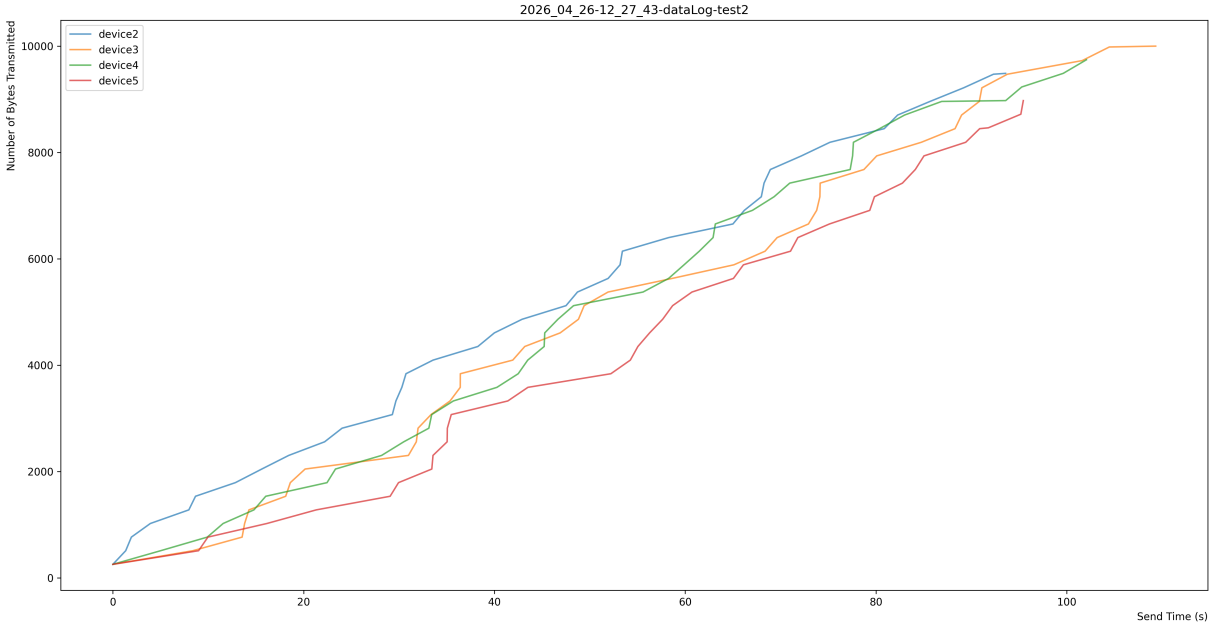

Figure 5.8: Measured throughput with a single recording for Test Case 2

This graph shows two aspects of the DPTP:

Packet Loss Not all four lines reach the same height. This can be explained, because the DPTP does not support the retransmission of lost packets like for example the Transmission Control Protocol (TCP). With the retransmissions of packets through the `packetSender()`, the probability of packet loss is reduced but the protocol does not implement a way to ask for a missing packet. In this regard the DPTP has more similarities with the User Datagram Protocol (UDP). Due to the alignment of the nodes in the experiment, on average the packet loss increased slightly for nodes with higher numbers which can be explained through the increased number of hops a packet has to take to reach those nodes.

Similar Throughput for all nodes Although not all lines reach the same total height and the median delay for nodes further away from `node1` is higher (5.2.1), they exhibit a similar slope that only slightly decreases the further a node is away from `node1`. This can be explained through the effect, that even if the delay of a single packet is higher for nodes further away, this delay is not cumulative but only added once, because the sender does not wait for each AWR-PCK of the receiver. Instead `node1` sends the packets in a static interval with a two seconds sleep netween each packet.

Both of these aspects can also been seen when viewing all three recordings of the second test together[1]:

[1] the x-axis space between the recordings was artificially reduced by the factor ten to improve visibility

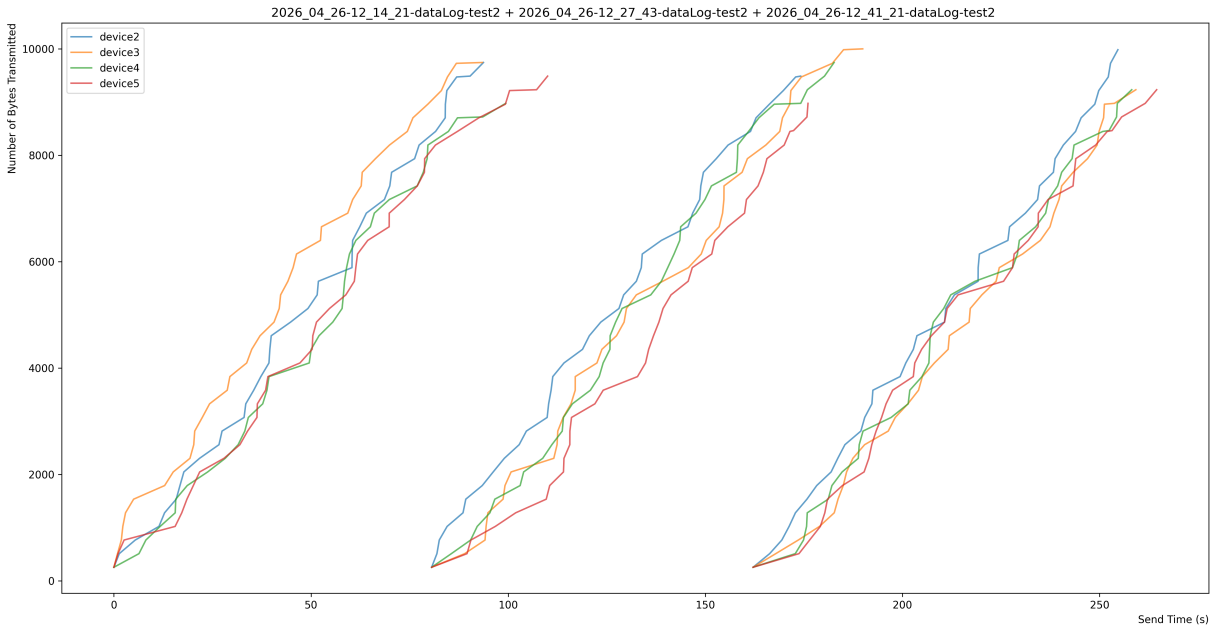

Figure 5.9: Measured throughput across three experimental runs for Test Case 2

As an addition to the visual representation of the data, a numerical analysis was performed which shows the same results:

```
// throughput in B/s
device2: {throughput: [101.220, 98.638, 105.136], packet loss: 4/120}
device3: {throughput: [101.536, 89.123, 92.457], packet loss: 4/120}
device4: {throughput: [87.745, 92.954, 93.467], packet loss: 8/120}
device5: {throughput: [83.922, 91.355, 87.688], packet loss: 9/120}
```

Figure 5.10: Test 2 results numerical

5.2.3 Overall Evaluation

Concluding the results of the two tests it shows that the DPTP has its advantages, but also limits. Especially in the area of delay-tolerant networks where not many protocols have been widely used, the DPTP can be an asset when other protocols could become too slow or do not provide enough features/layers in the OSI model while not being compatible with other protocols. In this regard DPTP has the advantage that it covers the OSI layers 2 to 6 (see 2.7).
On the other hand the DPTP does not support low-latency applications. Theoretically this could be changed if the interval for the `packetSender()` is reduced and if a node retransmits a packet directly after receiving it. But this approach was purposefully not used, because it would also strongly increase the number of frames and therefore interference if multiple nodes are close to each other and all of them always directly repeat a received packet. And by design the DPTP should rather support applications where many nodes are intermittently in proximity and sometimes spatially dispersed than applications that require a low latency. In addition to the high latency, the DPTP can, depending on the OSI layer 1 medium being used, suffer packet losses as shown with test 2 (5.2.2). To tackle this problem controll parameters and functions similar to TCP could be added to the protocol. For this see 6.1.

6 Conclusion

This research project can be summarized as follows: At the beginning, the previously designed DPTP was improved to make it compatible for wireless transmission. In the next step, a wireless half-duplex transceiver was selected and a wrapper for sending and receiving packets with it was implemented. Following that, the implementation for the DPTP was rewritten, features were added/removed and a shared library to be included in other applications to use the DPTP was built.
After the implementation of the DPTP for wireless communication was done, two tests were performed using an unmanned aircraft together and four ground nodes. These tests demonstrated that the implementation works in regard to the two main features of the protocol, delay-tolerance and ad hoc mesh networking. On the other hand, the results show that forwarding packets between nodes does not significantly impact throughput over longer periods, even when multiple intermediate nodes are involved. Taken together, these results indicate that, although the DPTP is primarily suitable for systems with specific layout and high delay tolerance, it offers distinct advantages in such scenarios.

6.1 Possible Future Improvements

As with most network protocols, the DPTP is not yet perfect and can still be improved. Some of these improvements should be:

- `Retransmission of Packets:` Either a new field with a sequence number could be added or the packet/answer ID field could be split into the packet/answer ID and an increasing sequence number for each packets. By this the receiver of a stream of packets knows if a packet is missing and could directly ask the sender to transmit it again after its `Time of Death` has been reached. This request of transmitting the packets again could even be added into the structure of AWR-PCKs.
- `Encryption of application data:` In the current implementation the packets are signed with the private key of a node and can be verified by its public key that the other nodes receive from the ITN-PCK. It could be considered to implement the option to exchange an AES key between two nodes if one of them wants to transmit encrypted data. For the exchange the already known public keys can be utilized. This symmetric key could be bound to the connection and be deleted if a connection is overwritten or a CED-PCK is used to end it.
- `Sending larger amounts of data when nearby:` Currently a node always transmits data from the application immediately. A buffer could be implemented to store larger amounts of data for a single receiver node and hold on to it for a little time to see if the receiver is coming closer to the sender or not. If it does, it might be beneficial to transmit the packets when fewer nodes would be needed to reach the receiver. The number of nodes needed can be measured by the highest hop number from an DAT-PCK or AWR-PCK from that node.

Bibliography

[1] Scott Burleigh, Kevin Fall, and Edward J. Birrane. *Bundle Protocol Version 7*. RFC 9171. Jan. 2022. DOI: 10.17487/RFC9171. URL: https://www.rfc-editor.org/info/rfc9171.

[2] Brethel Chavez, Tedd Angelo, and Darllaine Lincopinis. "C Programming Language: A Review." In: (May 2023), pp. 0-0000. DOI: 10.3897/jucs..

[3] *Commotion Construction Kit: Introduction to Mesh*. https://communitytechnology.github.io/files/cck/networking/2-Introduction_to_Mesh.pdf. Accessed: 2026-03-31.

[4] Jean-Patrick Gelas Ghislain Landry Tsafack Chetsa Laurent Lefevre. "On Applying DTNs to a Delay Constrained Scenario in Wired Networks." In: *International Symposium on Wireless Personal Multimedia Communications (WPMC)*, (2011), p. 6.

[5] Texas Instruments. "B.A.T.M.A.N Unpacked: A Guide to the Protocol's Fundamental Concepts." In: (2009), p. 5. URL: https://www.ti.com/lit/ds/symlink/cc1101.pdf.

[6] Leander Seidlitz José Afonso Gastaldi de Araújo Teixeira. "B.A.T.M.A.N Unpacked: A Guide to the Protocol's Fundamental Concepts." In: (2024), p. 5.

[7] Axel Neumann, Corinna Aichele, Marek Lindner, and Simon Wunderlich. *Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.)* Internet-Draft draft-openmesh-b-a-t-m-a-n-00. Work in Progress. Internet Engineering Task Force, Mar. 2008. 24 pp. URL: https://datatracker.ietf.org/doc/draft-openmesh-b-a-t-m-a-n/00/.

[8] Bundesamt für Sicherheit in der Informationstechnik. "Cryptographic Mechanisms: Recommendations and Key Lengths." In: (2026), p. 92. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=14.

Declaration

We declare that we have written this report independently. We have acknowledged all passages taken verbatim or in spirit from published or unpublished works of others or from the author himself/herself. All sources and aids that we have used for the report are indicated. The report has not been submitted to any other examination authority with the same content or in essential parts.

Note: In some degree programs, the declaration can be found directly after the cover sheet of the report.

30.04.2026

Place, date

Janis Neumann

Signature