Projekt-Dokumentation

Luc Heitz «luc.heitz@stud.unibas.ch»
Manuel Schmidt «manu.schmidt@stud.unibas.ch»
Raphael Waltenspül «raphael.waltenspuel@stud.unibas.ch

18. Juli 2021
Universität Basel

1. Einleitung

Nach der Einführung in die Idee hinter dem BACnet und Secure Scuttlebut war für uns schnell klar, dass die Abhängigkeit vom normalen Internet im beschriebenen Szenario nicht optimal ist. In einer Situation, in der potenziell das gesamte Internet abgestellt werden könnte, ist unserer Meinung nach das Wichtigste, dass die Bevölkerung über Distanz immer noch über eine Art Messenger verfügt. Dabei gefiel uns die Idee, einen simplen Messenger über die LoRa Technologie zu entwickeln. Unser primäres Ziel war, ein funktionierendes System zu entwickeln, statt uns komplett korrekt an die Struktur vom BACnet oder Secure Scuttlebut zu halten. Nachdem wir das Projekt des LoRaLink des Vorjahres gesehen hatten, war uns bewusst, dass wir dafür stärkere Hardware als die LoPy Geräte benötigen. Die Wahl fiel auf das Raspberry Pi der 3. Generation und das LoRaShield von Dragino. Wir erhofften uns durch das Verwenden stärkerer Hardware, die Problematik des Vorjahres umgehen zu können. Um dies umzusetzen, teilten wir das Projekt in mehrere Teile auf: Den Treiber/Linklayer konnten wir teils von einem anderen Projekt übernehmen welche zuständig für das Aufteilen der Pakete in Byteströme sind, die Transportlayer ist zuständig für das Senden/Empfangen/Weiterleiten von Nachrichten und die Applicationlayer die die zu sendenden Daten entgegennimmt und die empfangenen Nachrichten ausdruckt.

2. Verwendete Hardware

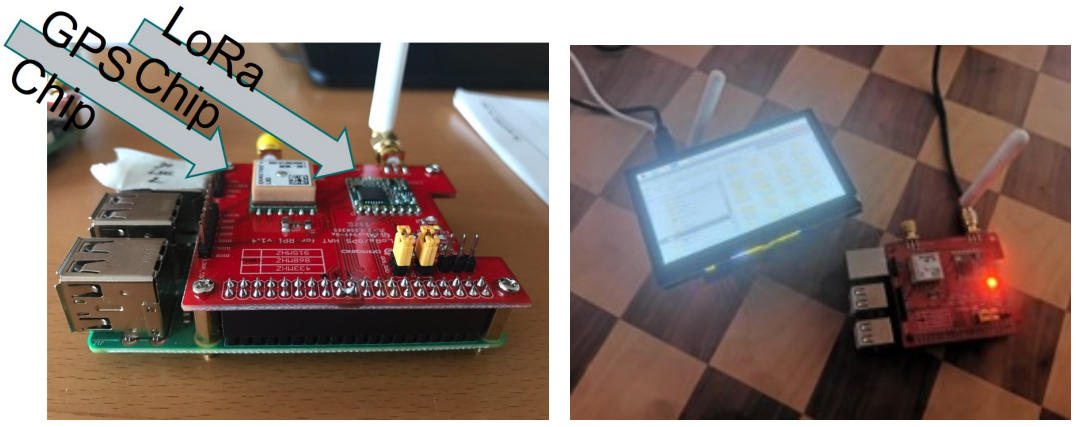
2.1. Hardwareaufbau

Für den Hardwareaufbau wurden folgende Komponenten gewählt:

- Raspberry Pi 3 Modell B
- Dragino Lora/GPS Hat (Chip SX1276)
- 1 Raspberry Pi mit Touchdisplay als Mobiler Empfänger

Zusätzlich wurden Benötigt:

- Stromquellen für Aussen Tests (Powerbanks)



(a) Raspi LoRa Hat (b) Version mit LCD display

Abbildung 1: Aufbau der Raspberry Pi LoRa Stationen

Die Kommunikation mit dem Chip SX1276 findet über einen SPI Bus statt. Das detaillierte Schalt-Schema ist im Anhang A.1 angefügt. Weiter ist unter https://wiki.dragino.com/index.php?title=Lora/GPS_HAT eine Dokumentation des Lora Hats als Wiki zu finden.

2.2. Inbetriebnahme

Das erste Ziel war das Raspberry Pi und Lora Hat in Betrieb zu nehmen und über LoRa P2P Nachrichten an ein anderes Modul zu senden. Die Inbetriebnahme des Raspberry Pi zusammen mit dem LoRa Hat gestaltete sich jedoch schwieriger als geplant.

Es traten folgende Hauptprobleme auf:

Fehler 1: Keine Antwort auf dem SPI bus, das Signal wurde nicht durch den Chip SX1276 verarbeitet.

Ursache 1: Der Chip Enable des LoRa SPI war auf einem anderen Pin als der CE des Raspberry Pi's

Lösung 1: Es wurde eine Lötbrücke zwischen Pin 22 und 24 gesetzt.

Nun konnte eine Nachricht zwischen zwei Modulen gesendet werden.

Fehler 2: Nach kurzer Zeit stürzte der SPI bus ab und es konnten keine Nachrichten mehr empfangen oder gesendet werden.

Ursache 2: Auf dem Bus wurden Störungen verursacht, welche durch den nicht angeschlossen Pin erfolgten.

Lösung 2: Einerseits wurde ein Hearbeat bzw. Watchdog implementiert, welcher das Board bei einem Spi Fehler neu startet, andererseits wurden die Fehler verursachenden Pins hardwareseitig entfernt, um die Störungen auf dem Bus zu reduzieren.

Danach konnten über einen P2P Chat Nachrichten zwischen den Modulen versendet werden.

3. Grundidee

Unser Netzwerk besteht aus drei Schichten. Diese Schnittstellen werden in unserem Netzwerk mit Hilfe von Queues verbunden. Jede Schicht besitzt zwei Queues die jeweils die Schicht mit der oberen oder unteren Schicht verbindet. Die oberste Schicht ist die Application Layer, bei welcher es mehr oder weniger, um das Einlesen von den zu versendenden Nachrichten (Messages) und das Darstellen von den zu erhaltenden Nachrichten geht. Die darunter liegende Schicht ist die Transport Layer, welche für die Erstellung und Bearbeitung des Appendonly Logs zuständig ist. Die Transport Layer ist auch zuständig für das Gossip Protokoll, das bedeutet, welche Pakete durch unser Netzwerk hindurch kommen und wie sie verteilt werden. Die unterste Schicht unseres Netzwerkes bildet die Link Layer, die für die physische Datenübertragung zuständig ist.

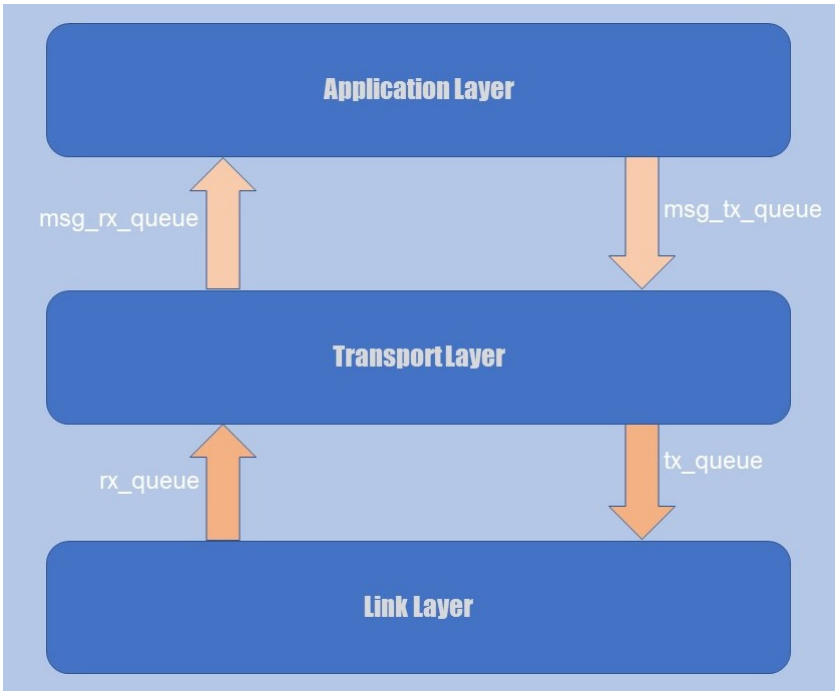


Abbildung 2: Schnittstellen des Netzwerks

3.1. Treiber / Link Layer

Um die Hardware in das Gesamtsystem einzubinden, wurde die Lora Link Layer entwickelt. Die Lora Link Layer implementiert den SX1276 Treiber. Es bietet als Schnittstelle zwei Queues an, eine Rx-Queue und eine Tx-Queue.

Solange in der Tx-Queue kein Paket zum versenden ansteht, horcht die Link Layer und füllt alle über LoRa empfangenen Pakete in die Rx-Queue. Will man etwas versenden, füllt man das Paket in die Tx-Queue. Die Link Layer schaltet den Modus automatisch um bis alle Pakte in der Tx-Queue versendet wurden.

4. Transportlayer

4.1. Gossip Protokoll

Das Gossip Protokoll baut auf dem Broadcast der Link Layer auf. Da wir auf der physischen Layer nur diese Möglichkeit besitzen um eine Nachricht weiterzuleiten, mussten wir das «spreading rumors» Gossip Protokoll auf der Transport Layer implementieren. Es gibt vier wichtige Funktionalitäten, die das Protokoll bieten muss:

1. Jede Nachricht wird durch Broadcast weitergeleitet und jeder Empfänger in Reichweite erhält diese Nachricht.
2. Jeder Knoten ist Empfänger/Sender/Transmitter in unserem Netzwerk.
3. Überprüfung ob eine Nachricht für sich gedacht ist.
4. Überprüfung ob man diese Nachricht schon mal empfangen/weitergeleitet hat.

Punkte 3) und 4) werden im Folgenden noch genauer beschrieben. Sobald eine Nachricht empfangen wird, überprüft die Transport Layer, ob diese Nachricht für sich selbst gedacht ist. Falls dies der Fall wäre, wird diese Nachricht zur Application Layer weitergeleitet, wo

sie dann ausgedruckt wird. Wenn sie aber für einen anderen User, im Netzwerk gedacht ist, muss als erstes überprüft werden, ob man diese Nachricht schonmal empfangen oder weitergeleitet hat. Dies haben wir mit einem Hashwert (mit MD5) gelöst, welchen wir in einem Dictonary (duplicates) gespeichert haben. Der Hashwert wird bei jeder empfangenen Nachricht anhand des angefügten Zeitstempels der Application Layer und der dazugehörigen Nachricht berechnet. Dabei sollte jeder dieser Hashwerte einzigartig für jede empfangene Nachricht sein. Falls man die Nachricht noch nicht empfangen oder weitergesendet hat, befindet sich der Hashwert noch nicht in duplicates und daher wird diese Nachricht zur Link Layer übertragen und mit Hilfe des Broadcast weitergesendet. Bei mehrfachem Empfangen der gleichen Nachricht, erkennt duplicates den Hashwert und darauf wird diese nicht mehr weitergeleitet, sodass man nicht eine Nachricht konstant hin und her sendet.

4.2. Arten von Nachrichten

Um Nachrichten versenden/empfangen zu können und gleichzeitig den Appendonly Log auf dem aktuellen Stand halten zu können werden drei Arten von Nachrichten verwendet:

- **Normale Nachrichten:** Diese sind mit einer 0 (Null) im Flagfeld gekennzeichnet. Sie werden von jedem der erreichbar ist empfangen und/oder weitergeleitet und versucht an den Appendonly Log anzuhängen.
- **Request Nachrichten:** Diese sind mit einer 1 im Flagfeld gekennzeichnet. Sie werden versendet, wenn der Empfänger aufgrund von nicht konsekutiven Sequenznummern nicht in der Lage ist, die Nachricht an den Appendonly Log anzuhängen. Die Request Nachricht wird an den Sender, der nicht angehängten Nachricht adressiert. Als Daten wird die Sequenznummer der letzten korrekt angehängten Nachricht mitgesendet.
- **Response Nachrichten:** Diese sind mit einer 2 im Flagfeld gekennzeichnet. Sie werden nur versendet, wenn davor eine Request Nachricht empfangen wurde. Als Daten werden alle notwendigen Informationen versendet, die für den Appendonly Log benötigt werden.

4.3. Appendonly Log

Der Appendonly Log liegt als persistentes log.txt auf jedem Gerät und wird entweder passiv oder aktiv aktualisiert. Beim Starten des Programmes wird überprüft, ob diese Datei bereits existiert. Falls nicht wird eine neue Datei mit einem Initialisierungseintrag für das Gerät angelegt. Technisch ist das mit Hilfe von einem Dictionary in Python implementiert, wobei der Identifier des Gerätes der Key und eine Liste von seinen Nachrichten der Value ist. Die Nachrichten sind in einer JSON Struktur aufgebaut und beinhalten folgende Felder: Sequenznummer, Empfänger, Zeitstempel und Nachrichteninhalt. Die Sequenznummer entspricht der Position im Array. Jedes Gerät ist für seine eigene Liste verantwortlich aber nimmt in separaten Dictionary Einträgen Nachrichten auf, die von anderen versendet wurden. Dies funktioniert primär passiv, während dem die Nachrichten empfangen oder weitergeleitet werden. Dabei wird jedes Mal überprüft von wem die empfangene Nachricht stammt und ob die Sequenznummer der letzten Nachricht im entsprechenden Log um eins kleiner ist als die der empfangenen Nachricht. Ist dies der Fall, wird die neue Nachricht angehängt. Falls nicht wird eine Request Nachricht mit der letzten Sequenznummer der Liste des Senders an den Sender gesendet. Bei Erhalt einer Request Nachricht wird, die in den Daten übermittelte Sequenznummer mit der eigenen

aktuellen Sequenznummer subtrahiert. Da man mit Python mit negativen Arrayzugriffen arbeiten kann, nutzen wir dies aus und versenden von der negativen Differenz aus bis und mit dem Eintrag an der Stelle -1 (letzter Eintrag) einzelne Response Nachrichten. Response Nachrichten können von jedem Transmitter auch verwendet werden, um ihre Logs zu aktualisieren. Mithilfe der JSON Library wird zur Laufzeit, wenn gebraucht, die log.txt Datei als Dictionary geladen. Wenn die gewünschten Änderungen vorgenommen wurden, wird es im JSON Format in das log.txt exportiert.

5. Application Layer

Die Hauptaufgabe der Application Layer ist es Eingaben aus der Konsole einzulesen und zu überprüfen, ob diese das richtige Format und die richtige Länge besitzen. Nach vielen Tests mussten wir die Datenlänge auf unter 100 Bytes beschränken, ansonsten kamen die Nachrichten über unsere Hardware nicht mehr an. Die Konsoleneingabe ist auch beschränkt auf ein bestimmtes Eingabemuster («Receiver;Data»). Falls beide dieser Fälle korrekt sind, hängt die Application Layer für den späteren Gebrauch noch einen Zeitstempel an die Nachricht und leitet diese verpackte Nachricht über die Queue als Schnittstelle weiter zur Transport Layer. Des Weiteren ist die Application Layer zuständig, um die erhaltenen Nachrichten des Benutzers auf der Konsole darzustellen. Auch hier wird auf einer Queue zugehört bis ein Element von der Transport Layer empfangen und weitergeleitet wird. Dies wird dann auf der Konsole in dem richtigen Format dargestellt. («Received From: Sender: Data») Das Einlesen wie das Darstellen laufen beide auf einem separaten Thread.

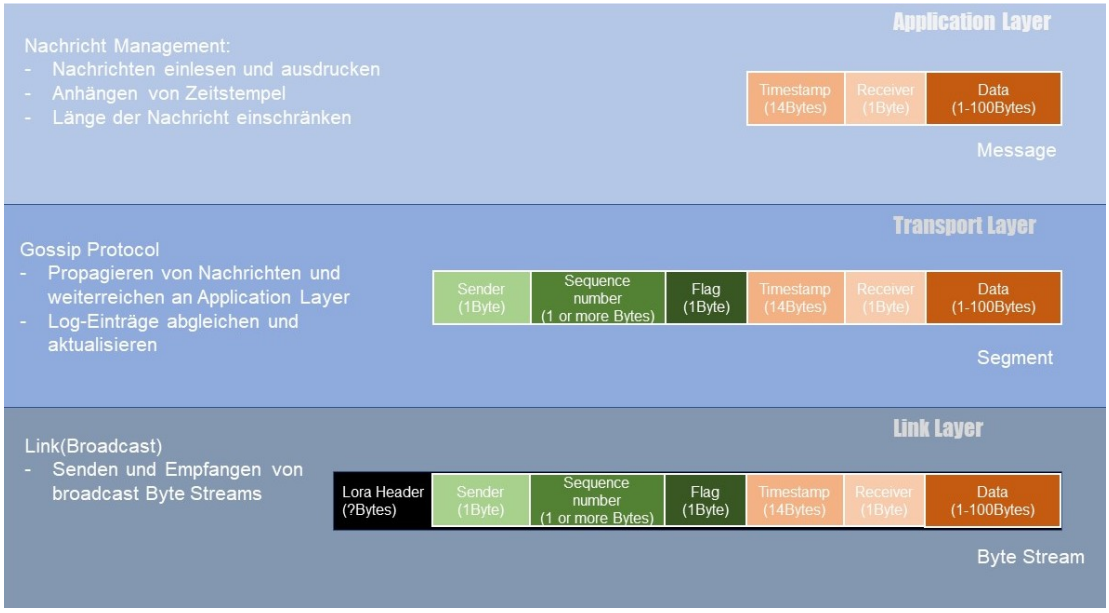


Abbildung 3: Aufbau der drei Netzwerkschichten

6. Diskussion

Aufgrund der oben beschriebenen Hardwareprobleme wurde bereits die meiste Zeit, die für das Projekt gedacht war, aufgebraucht. Grundsätzlich haben wir unser Ziel erreicht, nämlich Nachrichten gezielt und in kurzen Intervallen hin und her zu senden. Damit wäre aus unserer Sicht das Projekt abgeschlossen gewesen. Nach einem weiteren Gespräch

machten wir uns noch an die Arbeit einen einfachen und persistenten Appendonly Log zu implementieren. Generell wurden alle Komponenten einfach gehalten, da wie bereits erwähnt, die Funktionalität vor der Konformität zu BACnet oder Secure Scuttlebut stand. Bei unseren Tests funktionierte das Senden und Empfangen von Nachrichten mit vier Geräten gleichzeitig. Ein Problem dabei ist, dass Kollisionen nicht vermieden werden, dies müsste auf der Linklayer und im Treiber implementiert werden. Besonders problematisch war das vom Hersteller eigentlich kaum Dokumentation vorhanden war. Auch im Internet war kaum etwas zur verwendeten Hardware zu finden. Eine weitere Verbesserung wäre eine Verschlüsselung der Nachrichten einzubauen. Auch der Appendonly Log könnte optimiert werden. Grundsätzlich wird nur passiv angehängt oder reaktiv nach fehlenden Einträgen angefragt. Es wäre sinnvoll auch ein initiatives Anfragen, unabhängig von Fehlern beim Anhängen von Einträgen, nach Informationen zu implementieren.

A. Appendix

A.1. Schalt-Schema LoRa Hat

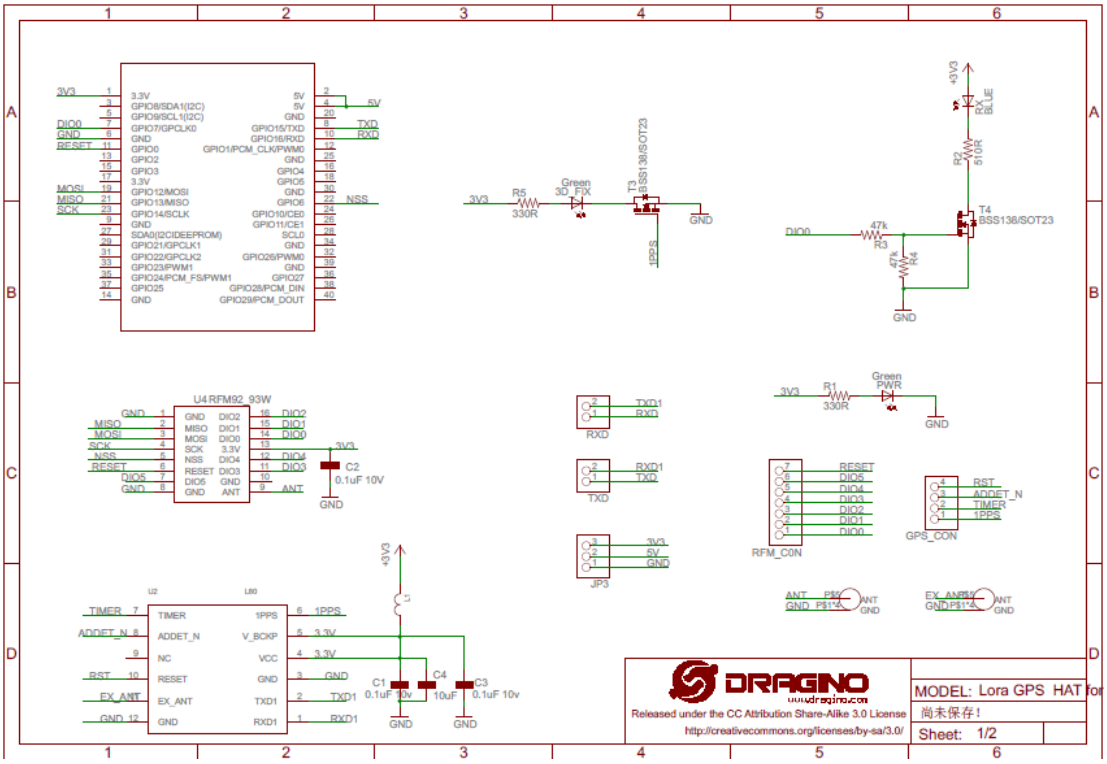


Abbildung 4: Schema LoRa Hat