

# Project BACnet (BAsel Citizen Net): Networking *without* Internet

University of Basel, July 2020

The Internet has become so ubiquitous that we take it for granted, forgetting completely how much its use depends on political goodwill: Access restrictions are quite common worldwide, be it the “Chinese firewall”, India cutting the Internet for Kashmir, or the Spanish Police seizing the DNS name of the Catalan independence movement.

In spring 2020, BSc students of the lecture *Introduction to Internet and Security* at the University of Basel implemented a system that avoids that dependency. The task was to write software that enables social media (a chat application) without having to rely on Internet connectivity and without need for servers that would be too easy a target, if someone wanted to bring down the service. Instead, USB sticks are used to implement a so called “Sneaker Net” where content is propagated among friends. Wireless communication using the unlicensed WiFi frequency band was included for speeding up content dissemination over a distance of more than 10km. Finally, two groups also looked into forwarding sensor data over LoRa, a long-range and low-energy communication technologies.

Technically, BACnet (BAsel Citizen Net) followed concepts developed in ‘Secure Scuttlebutt’, a decentralized approach for implementing server-less distributed applications using replicated append-only logs. The motivation for BACnet was the hypothesis that sensor-resistant communication systems *can* be built from scratch with reasonable effort and time, covering fancy WiFi as well as more low-tech USB stick swapping, yet serving humans with basic inter-person messaging. I’m very pleased to see that the students of the *Internet and Security* lecture proved me right. I would like to acknowledge their effort, dedication and creativity! I was also impressed how they were able to advance the project despite the COVID-19 crisis that unfolded just days after we had the first start seminar.

In this booklet you will find the reports that the students handed in. After having presented their results in a mini-seminar, the students were asked to describe salient aspects of their subproject. I hope that this collection will serve a next cohort of students to learn more about decentralized approaches in distributed system. Students were free to choose German or English for their writeup.

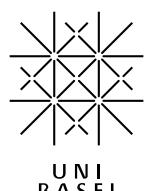
Prof. Dr. Ch. Tschudin, summer 2020  
with the help of Dr. C. Scherb and MSc C. Marxer (assistants)

## Table of Contents

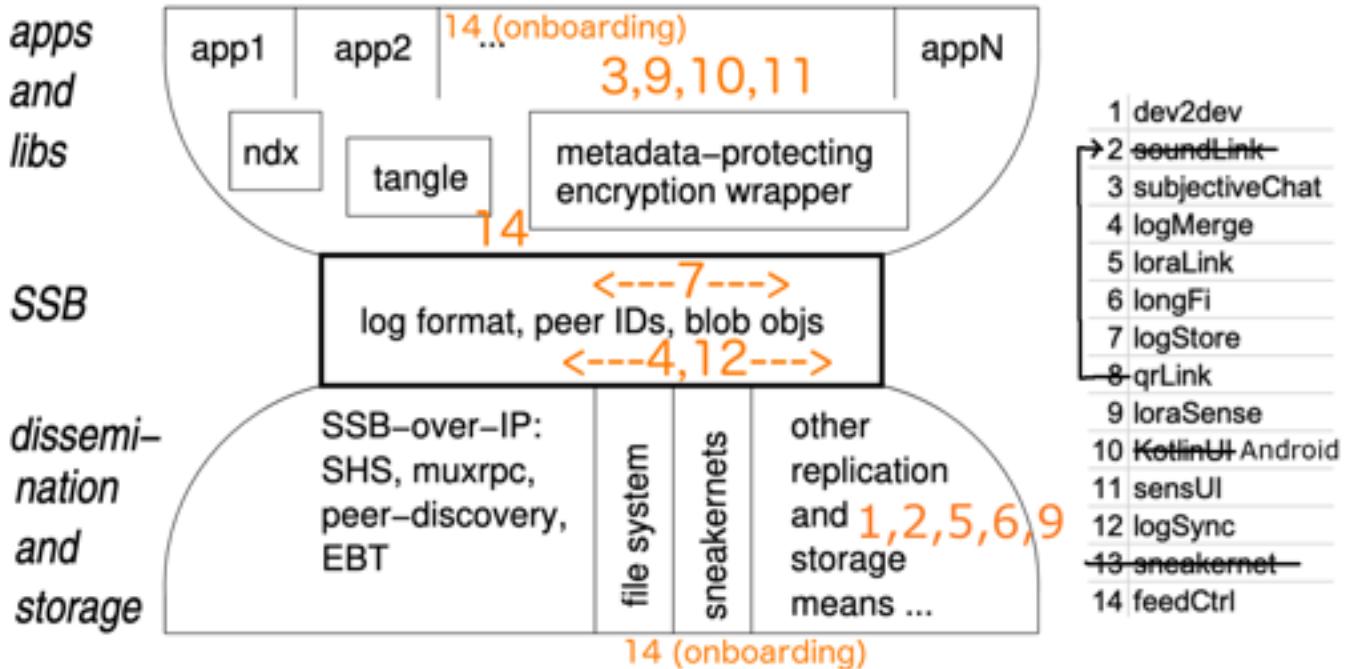
Group	Topic	Authors
1	Dev to dev	Tim Keller, Mateusz Palasz and Oliver Weinmeier
2	QR link	Renato Farruggio and Caroline Steiblin
3	Subjective chat UI	Tunahan Erbay and Ken Rotaris
4	Log merge	Günes Aydin, Nikodem Kernbach and Joey Zgraggen
5	LoRa link	Patricia Heckendorf and Julian Trinkler
6	Long-FI	Faris Ahmetasevic, Marco Banholzer and Nderim Shatri
7	Log store	Viktor Gsteiger and Moritz Würth
9	LoRa sense	Patrice Delley and Leonardo Salsi
10	Android app	Sanja Popovic, Travis Rivera Petit and Nour Shokry
11	Sens UI	Marc Schäfer
12	Log sync	Carlos Tejera and Alexander Oxley
14	Feed ctrl	Wilson A. Eghonghon, Damian Knuchel and Yannick A. Rümmele

## References

- GitHub repo: <https://github.com/cn-uofbasel/BACnet>  
Secure Scuttlebutt: <https://scuttlebutt.nz>  
ICN19 paper: <https://dl.acm.org/doi/10.1145/3357150.3357396>



## Role and Position of the BACnet Subprojects



**Figure 2: Secure Scuttlebutt's protocol stack.**

Subproject	Goal
1 Dev to dev	Synchronization over Bluetooth and WiFi
2 QR link	Bidirectional data exchange via QR codes
3 Subjective chat UI	Python-based UI for BACnet
4 Log merge	Library for feed creation, event validation
5 LoRa link	Sensor data propagation over Long Range radio
6 Long-Fi	Synchronization via long range WiFi
7 Log store	SQLite database for binary and decoded log entries
9 LoRa sense	Sensor devices producing log entries
10 Android app	Chat application displaying BACnet messages
11 Sens UI	Chart generation for LoRa-relayed sensor data
12 Log sync	UPD-based synchronization between two peers
14 Feed ctrl	Trust management: onboarding and master-feed

# Introduction to Internet and Security

Frühjahrsemester 2020

Gruppe 1 - dev2dev

July 19, 2020

Tim Keller  
Mateusz Palasz  
Oliver Weinmeier

## Contents

1	Zusammenfassung . . . . .	2
2	Einführung und Hintergrund . . . . .	2
3	Implementation . . . . .	3
4	Ergebnis . . . . .	4
5	Diskussion . . . . .	4
6	Referenzen . . . . .	5

## 1 Zusammenfassung

Ziel unserer Projektarbeit war die Synchronisierung der Logs. Dabei war gedacht, dass Bluetooth und Wifi als Übertragung Technologie genutzt werden sollte. Die Kopplung wäre dann, per Bluetooth durchgeführt worden und Wifi wäre für die Übertragung verantwortlich gewesen.

Letztendlich ist es uns, durch einige Einschränkungen gelungen, Bluetooth als alleinige Technologie zu integrieren. Das bedeutet wir haben Bluetooth für die Kopplung genutzt, sowie für die Datenübertragung.

Außerdem war angedacht, unserer Technologie für MacOS und Windows zur Verwendung zu stellen. Dies war im Endeffekt nicht realisierbar, wodurch wir nur MacOS als Betriebssystem nutzen konnten.

## 2 Einführung und Hintergrund

Die größten Inspirationen für das Projekt waren bereits existierende Technologien wie AirDrop und verschiedene Derivate, die eine vergleichbar komfortable Benutzung haben. AirDrop ist eine proprietäre Funktionalität in macOS und iOS Geräten, welche den wireless Austausch von Datenobjekten zwischen 2 Geräten ermöglicht, wobei keine vorherige Verbindung der beiden Geräte nötig ist, solange beide WiFi und Bluetooth angeschaltet haben. Wenn man genauer hinschaut, gibt es noch weitere proprietäre Technologien wie Bonjour die für AirDrop wichtig sind, aber das Allgemeine Prinzip baut auf WiFi und Bluetooth auf. Auch auf Android-Geräten gibt es eine ähnliche Funktionalität basierend auf WiFi-Direct, welches ein öffentlicher WiFi-Standard ist.

Vor allem die Einfachheit der Benutzung von AirDrop war letztendlich, was wir mit unserem Projekt für das BacNet replizieren wollten. Im Gegensatz zum Ansatz vom Sneakernet soll man nicht mit USB Sticks manuell Daten austauschen und von Endknoten zu Endknoten in namensgebenden Sneakern sich bewegen müssen sondern bei uns war es das Ziel, dass wie bei AirDrop, zwei Benutzer sich über eine wireless Verbindung die Logs austauschen können.

Die genaue Funktionalität von AirDrop funktioniert über WiFi und Bluetooth. Dabei wird erst mit Bluetooth die Verbindung aufgebaut und die Informationen zu einem neuen ad-hoc Netzwerk zwischen beiden Seiten aufgebaut und der Datenaustausch an sich wird über WiFi. Ähnliche Technologien funktionieren auf einem etwas verschiedenen Weg aber erstellen trotzdem ein ad-hoc Netzwerk und benutzen WiFi für den schnellen Datenaustausch.

Unsere Implementation übergeht WiFi und benutzt Bluetooth sowohl für den Verbindungs austausch und für den Datenaustausch an sich.

Die Hauptplattform für unser Projekt ist macOS. Doch aufgrund der fehlenden Schnittstellen für die Programmierung mit WiFi-Direct auf macOS basiert unser Projekt wie bereits erwähnt ausschließlich auf Bluetooth. Unvermeidbare Fehler bzw. Unannehmlichkeiten die bei der Benutzung von dev2dev auftauchen können, wie lange Discovery-Zeit bis andere Geräte erkannt werden oder die im Vergleich zu Wi-Fi niedrige Geschwindigkeit der Datenübertragung, greifen letztendlich auf die Limitationen von Bluetooth und generell wireless Technologien zurück.

### 3 Implementation

Um dieses Ziel in die Tat umsetzen zu können, haben wir unsere Gruppe in verschiedene Funktionsträger aufgeteilt:

- Projektleiter: Tim Keller
- Technischer Leiter: Oliver Weinmeier
- Schriftführer: Mateusz Palasz

Tim Keller, als Projektleiter, hatte stets den Überblick, über das Gesamte Projekt. Termine und Einhaltung der Deadlines gehörten hauptsächlich zu seinem Aufgabenbereich. Außerdem war die Koordination die größte Herausforderung, die er sich stellen musste.

Oliver Weinmeier, als Technischer Leiter, war für die Integrierung der Technologien verantwortlich. Er hat die Ziele in kleinere Aufgaben unterteilt, und dafür gesorgt, dass diese ordnungsgemäß und sinnvoll bearbeitet wurden.

Mateusz Palasz, als Schriftführer, war für alle Dokumente zuständig. Darunter fallen auch Präsentation, die Notizen wären Meetings, sowie die daraus resultierenden Journals.

Gleichzeitig haben wir unser Projekt in zwei Teile aufgeteilt: der kleinere Teil war die Vorbereitung des Versendens der Dateien und die Verarbeitung der empfangenen Dateien. Der größere Teil war der Kommunikationsaufbau zwischen den zwei Geräten.

Dabei war ursprünglich geplant eine Kommunikation via Bluetooth aufzubauen, um dann die Details für ein Versenden der nötigen Dateien via Wifi-Direct auszutauschen. Allerdings mussten wir feststellen, dass die Geräte, welche uns zur Verfügung (hauptsächlich Mac und iPhone Geräte) stehen, nicht mit Wifi -Direct kompatibel sind. Aus Gründen von Zeit und Ressourcen haben wir uns deshalb dafür entschieden die Dateien nur via Bluetooth zu versenden und dafür ein bisschen an Effizienz zu verlieren.

Dafür haben wir unseren eigenen Bluetooth Dienst erstellt, welcher auf einem bestimmten Port den zweiten Client sucht und anschließend eine Verbindung aufbaut. Bei mehreren möglichen Clients kann ausgesucht werden, mit welchem Gerät die Verbindung aufgebaut werden soll. Anschließend werden Informationen ausgetauscht über den Aktuellen Stand der Dateien. Schlussendlich werden dann über den Dienst die Dateien ausgetauscht, welche den jeweiligen Clients fehlen.

Für den Umgang mit den PCAP Dateien und deren Präparation um versendet oder empfangen zu werden haben wir uns stark an den Demo Prozessen von Professor Tschudin orientiert. Diese konnten so gut eingebunden werden, dass diese dann gut zusammen, dass über den dort vorgeschlagenen Workflow die Brücke bis zum tatsächlichen Senden über Bluetooth nicht mehr allzu groß war.

Bei der Integration mit anderen Gruppen konnten wir dann unser Projekt auch ganz gut anknüpfen. Die Funktionen der API, welche von Gruppe 4 Log-Merge angeboten wurde, haben mit ein bisschen Absprache und Koordination sehr gut in unseren Workflow hineingepasst. Wie in unserer Demo zu sehen war konnten wir mit unserem Tool verschiedene Datenbanken mit PCAP Logs im Gesamtkontext des Projekts vergleichen.

Durch die Aufteilung der Verantwortungsbereiche erzielten wir eine eindeutige Schaffung der verschiedenen Zuständigkeitsbereiche und eine Verbesserung unseres effektiven Arbeitens. Die Implementation, hat so gut wie ausschließlich über Pair Programming stattgefunden, da das Testen durch die Umstände wie, nur MacOS als Betriebssystem und zwei Mac Geräte stark eingeschränkt war.

## 4 Ergebnis

Grundsätzlich haben wir das Projekt begonnen mit dem Ziel unser eigenes abgewandeltes Programm nach dem Vorbild der Apple Software Airdrop zu gestalten. Im Verlauf des Projekts ist uns dadurch klar geworden, welche Facetten die Software hat und in welchem Rahmen es überhaupt möglich ist mit unseren Umständen dies zu verwirklichen. Unser finales Produkt basiert zwar auf einer ähnlichen Funktionalität wie AirDrop, allerdings ist es noch ziemlich verschieden. Besser vergleichbar ist möglicherweise ein Vorgänger von AirDrop, das reine Senden von Dateien via Bluetooth von einem mobilen Telefon zum anderen. Aufgrund dessen basieren AirDrop und unsere Dev2Dev Software auf ähnlichen Grundbausteinen.

Dadurch, dass wir leider von dem Weg abgehe musste ein Adhoc-Netwerk über Wifi-Direct zu benutzen, verloren wir einen elementaren Teil, welcher zur Effizienz und Zuverlässigkeit von AirDrop beiträgt. Durch die Abhängigkeit von Bluetooth als Übertragungstechnologie, wird deren Schwächen eine große Fläche geboten. Der Aufbau einer Verbindung via Bluetooth kann ziemlich schwanken sein, manchmal braucht es mehrere Versuche, um sich mit dem richtigen Gerät zu koppeln. Auch die Verbindung an sich ist anfällig, wobei ein kleiner Verbindungsunterbruch bei einer größeren Übertragung verhängnisvoll sein. Zusätzlich nimmt die Benutzbarkeit bei steigender Übertragungsmenge, durch die niedrige Übertrangsrate, stark ab.

Zusätzlich haben wir festgestellt, dass im Open Source Feld zwar viele Optionen für Bluetooth Technologien zur Verfügung steht, allerdings nur wenig flexibel eingesetzt werden kann. Wir haben das Paket PyBluez als beste Option für unser Projekt abgesondert, allerdings mussten wir viele Bausteine umgehen und eine besondere Version aus einem Fork benutzen, um überhaupt die Software benutzen zu können.

Schlussendlich haben wir nur die Grundbausteine legen können, um wirklich eine zuverlässige und alltagstaugliche Übertragungssoftware nach dem Vorbild von AirDrop zu schaffen. Mit mehr Flexibilität und Ressourcen kann dieses Skelett sicherlich noch zu einem zuverlässig benutzbaren Produkt im Rahmen des Basel Citizen Nets werden.

## 5 Diskussion

Während unserer Arbeit ist uns aufgefallen, dass die Kommunikation eines der wichtigsten Elemente produktiven Arbeitens ist. Die Informationsketten müssen ständig funktionieren und strukturiert sein, um stets effektive Ergebnisse zu erhalten.

Um von außenstehenden Personen gute Informationen zu erhalten ist die direkte Ansprache auf das vorhandene Problem zu bevorzugen und am produktivsten. Hierbei war besonders Prof. Tschudin eine Herausragende Option. Damit Gruppenintern die Leistung und der Fortschritt ersichtlich

sind ist eine ständige Überprüfung und Dokumentation der Termine und Ergebnisse nötig. Die Dokumentation ist nicht nur für das laufende Projekt sinnvoll, sondern auch für folgende Projekte, um aus deren Erfahrungen zu lernen.

Des Weiteren wäre es von Vorteil gewesen, Limits und Einschränkungen frühzeitig zu erkennen, um nicht unnötig Zeit zu vergeuden. Unglücklicherweise haben wir Zeit in Sackgasen investiert, für nichts und wieder nichts, um im Endeffekt festzustellen, dass wir uns auf dem falschen Weg befinden. Letztendlich war dann eine Unterredung mit Professor Tschudin notwendig, damit wir akzeptierten konnten, dass für dieses Problem zurzeit keine Lösung gibt, oder der zu dem Zeitpunkt eingeschlagene Weg, nicht dem entsprach, was wir uns vorgenommen haben.

Rückblick erkennend kann festgestellt werden, dass wir unsere gruppeninterne Ziele, den Umständen entsprechend gut umgesetzt und erreicht konnten. Wir arbeiteten stets effektiv, zielorientiert, motiviert und kommunikativ. Auch mit unserer Ausarbeitung unseres Projektauftrags sind wir zufrieden.

## 6 Referenzen

- <https://en.wikipedia.org/wiki/AirDrop>
- [https://en.wikipedia.org/wiki/Wireless\\_ad\\_hoc\\_network](https://en.wikipedia.org/wiki/Wireless_ad_hoc_network)
- [https://en.wikipedia.org/wiki/Wireless\\_sensor\\_networks](https://en.wikipedia.org/wiki/Wireless_sensor_networks)
- <https://en.wikipedia.org/wiki/Bluetooth>
- [https://en.wikipedia.org/wiki/Bonjour\\_\(software\)](https://en.wikipedia.org/wiki/Bonjour_(software))
- [https://en.wikipedia.org/wiki/Wi-Fi\\_Direct](https://en.wikipedia.org/wiki/Wi-Fi_Direct)
- <http://www.wi-fi.org/discover-and-learn/wi-fi-direct>
- [http://www.ag-nbi.de/wp-content/uploads/2011/10/03\\_Ad\\_hoc\\_network\\_programming2\\_two\\_slidesperpage.pdf](http://www.ag-nbi.de/wp-content/uploads/2011/10/03_Ad_hoc_network_programming2_two_slidesperpage.pdf)
- <https://arxiv.org/abs/1812.06743>
- <https://arxiv.org/abs/1808.07353>
- <https://github.com/tigerking/pybluez/releases>

# **Group 02: qrLink Project Report**

(formerly soundLink)

**Renato Farruggio & Caroline Steiblin**

Introduction to Internet and Security  
Computer Science  
University of Basel  
19.07.2020

## Abstract

The course's programming project aim was to develop and implement a regional network for electronic communication that functions without the use of the Internet. This project's contribution to the overall goal aimed to create a decentralized tool to transfer data safely between two Android devices, using QR codes. This was achieved through the ZXing library, to transfer QR codes across Android devices, Chaquopy, to synchronize databases from the course's BACnet repository, and an integration with the BACnet client application. This QR code data transfer tool was implemented successfully, although inefficiently - around 12 bytes of data could be sent per packet, with a speed of 2-3 packets per second. This 268-word abstract would therefore take around 8-11 seconds to be transferred between two perfectly aligned Android devices, limiting the overall practicality and scalability of this tool.

## 1 Introduction

The Internet was originally developed to be decentralized, but has, over time, become centrally-organized. For example, Apple's iPhone email client and Google's Gmail dominate the market with a combined 54% of total email clients used [Richter, 2019]. In Introduction to Internet & Security, the course's main goal for the projects developed was to create and implement a regional network for electronic communication that functions without the use of the Internet. This overarching goal for a "re-decentralized" network was then divided into sub-projects to be developed in smaller groups, based on the Secure Scuttlebutt protocol stack, seen in Appendix A. The relevant sub-project discussed in this report is concerned with dissemination and storage, specifically with "other replication and storage means."

This project's contribution to the overall goal was to develop and implement an interface that solved the following problem: How can two Android devices send and receive data between each other, through QR codes? As well, the resulting interface would have to synchronize any database from the BACnet GitHub repository through a similar QR mechanism, and be integrated with other relevant projects in other parts of the SSB protocol stack. Relevant here were project groups 10 (KotlinUI, an Android application for BACnet) and 12 (logSync, an API to synchronize two databases). This report will outline the tool tailor-made to solve the problem(s) above, as well as all the relevant technologies and libraries required to enable the functionality.

## 2 Background

To enable the development of an interface that achieves all the requirements defined above - data transfer on two Android devices via QR codes, database synchronization, and external application integration - a few relevant technologies were required.

Firstly, to create an application for Android devices, Android Studio<sup>1</sup> using Java, needed to be used, as it is both the official and most efficient tool for creating Android-compatible interfaces. This allowed the project tool created to function for the devices required.

Transferring data is a key issue in many areas of computer science, and can be done in a large variety of ways - connecting physical cables, WiFi, Bluetooth, etc. QR (Quick Response) codes were developed in 1994 and are types of matrix barcodes that encode data in black and white patterns [Gregersen, 2019]. Although this project was originally looking to transfer data via sound waves, QR codes seemed to be a more feasible alternative after testing. To implement QR codes in the interface created, the library ZXing<sup>2</sup> was used. ZXing is an open-source image processing Java library for 1D and 2D barcodes, including QR codes, supporting both generation and decoding [git, 2020]. This library facilitated the QR code part of the interface functionality.

---

<sup>1</sup><https://developer.android.com/studio>

<sup>2</sup><https://github.com/zxing/zxing>

To enable BACnet database synchronization, for which groups used Python, a Python-Java “translation” needed to take place. The most effective method for this was to use Chaquopy<sup>3</sup>, an Android-Studio-compatible tool that allowed for Python-Java mixing (as well as the use of Python libraries) in the interface code.

Finally, the developed data transferring interface needed to be integrated into the BACnet chat application developed by group 10<sup>4</sup>. This application allows for onboarding processes, as well as for “trusted” users to write posts, see other users’ posts, change username, and add/remove users as “trusted” friends.

## 3 Implementation

### 3.1 Tasks

The process to complete the project could be broken up into key steps: outlining key interface functionality, technical research, implementation, planning integration partners, and integration. By the project’s position in the overall BACnet architecture, interface functionality was dependent mainly on the technical specifications allowed by the Android devices, although it had to be able to fit with the projects developed by other relevant groups. These specifications were found to be packet sizes around 500 bytes per packet with 2-5 packets per second, although these specifications were later restricted further due to the use of front-facing cameras. We implemented a packet splitting mechanism and a dedicated suitable transport protocol to be able to handle larger packets. Technical research was then conducted to match relevant support tools to the functionality needs, as well as to fit in with other projects developed which were done in Python. Implementation of the interface required meetings with relevant stakeholders (Prof. Tschudin, other groups) as well as pair programming and debugging to ensure all the requirements outlined were fulfilled. Finally, frequent communication between relevant integration groups was needed to adjust the interface to be ready for integration, and then the interface needed to be integrated into the BACnet application developed by group 10 (KotlinUI).

### 3.2 Division of Labor

With two group members, the project’s core programming and ancillary tasks were distributed the following way: Renato was responsible for back-end development and integration. Caroline was responsible for documentation, project organization, external group communication, and technical research. Both group members developed key project functions and debugged together.

### 3.3 Conceptual Configuration

The interface relied on the advanced QR protocol seen in Figure 1. A packet with content “123456” is transferred between Alice and Bob’s Android devices. The packet size here is three bytes (compared to 12 bytes in the interface developed) per QR code. The first byte in the QR packet is the header containing just the flag “last.” As Alice’s device splits its packets, Bob’s device needs to make sure it receives the entire packet before it continues working on it. The “last” flag is therefore always 0, except in the event that Alice’s device sends the last “subpacket” (in which the flag at 1). The process is therefore quite rudimentary. Alice’s device displays the first packet, here the first QR code containing “012”, until it recognizes the same QR code on Bob’s device, via the front-facing camera. Then Alice’s device sends the next packet. Bob’s device assembles the packet

---

<sup>3</sup><https://chaquo.com/chaquopy/>

<sup>4</sup><https://github.com/cn-uofbase1/BACnet/tree/master/groups/10-KotlinUI>

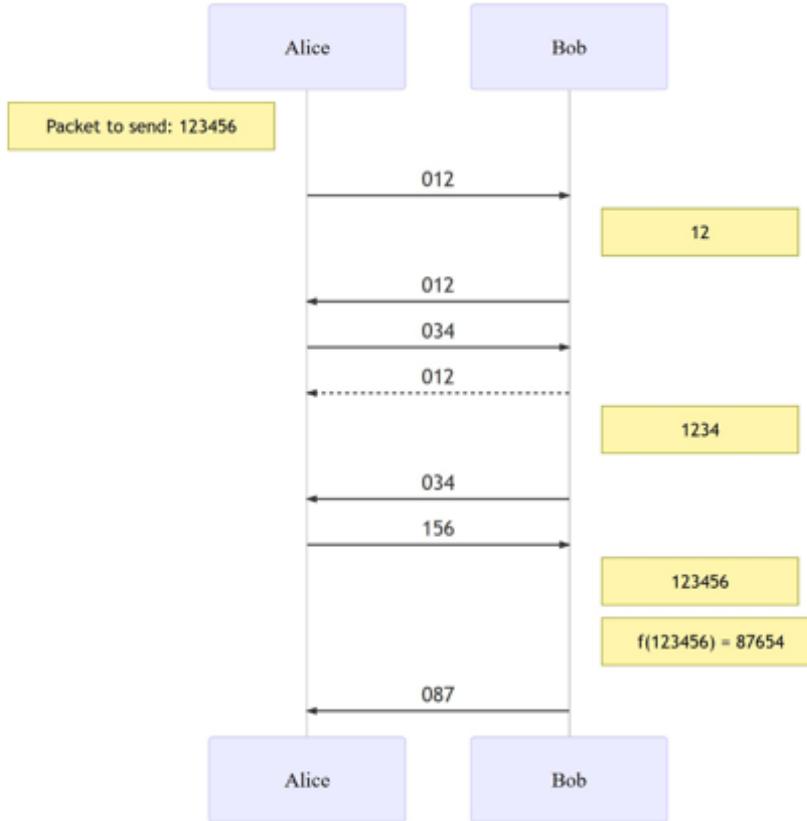


Figure 1: Advanced QR protocol

until it receives the “last” flag. It does not send this packet back, but processes it and sends its first “subpacket” back as a response.

When Alice’s device reads a QR code that has already been read and sent previously, it will be ignored and it will be assumed that the receiver, Bob’s device, has not received the last packet. Sending is therefore not up-to-date, but more of a convention, as only one image is displayed on the device’s screen. Therefore, Alice’s device does not need to send the packet again as it will remain on the device and visible to Bill’s device until Bill’s device acknowledges it and displays it back. This can be seen by the dotted line from Bob’s device to Alice’s with packet “012” in Figure 1. This process was realized through the use of the devices’ front-facing cameras for data transfer, versus the conventional use of back-facing cameras.

Through packet-splitting, sending arbitrarily large packets is possible, allowing this interface to be integrated into the tools developed by other projects. That said, bytes are not able to be encoded through the Zxing library, so Base64 (common binary-to-text encoding scheme) was used. This means that the packets are received as bytes, these bytes are transformed into multiple byte-arrays that are in turn encoded in Base64 and displayed. On the other side, text data in Base64 is read from the QR code, translated back into bytes, and attached to the overall packet until it is complete.

When the overall packet has been transferred between devices, the process is concluded, and the corresponding data is opened.

Using Chaquopy, the following functions from group 12 (logSync), in Python, were transformed

into Java-operable functions: `get_i_want_list` (list indicating required extensions and sequence number), `get_i_have_list` (list containing information on all feeds on one device for database comparison), `get_event_list` (list containing actual feed extensions), and `sync_extensions`, the latter finally enabling database synchronization. The protocol followed here can be viewed in Appendix B.

## 4 Results

Different data transfer interfaces using QR codes exist, although none found works without internet connectivity. The tool developed in this project sends raw data over QR codes directly, instead of saving the data in the internet and linking to it via QR codes. Implementing this functionality into the generalized BACnet application was a solid “proof of concept,” and was shown to perform onboarding (i.e. building trust between devices in a secure way) effectively. That said, currently the tool does not work unless both Android devices are directly facing each other at the right angle and height and cannot share large amounts of data in reasonable lengths of time. This process also proved to work best in a shaded room. A screenshot of the Android device setup to run this tool is seen in Appendix C. In the final presentation, synchronization between two users on the BACnet application took around five minutes - a long time to hold a phone in one specific position without moving. Support stands can be used, although it restricts usability. As well, using the front camera to scan QR codes bounds the speed of the data transfer to the quality of the front camera. Therefore, the effectiveness of the resulting interface is quite low, as the tool is not capable of efficient large data transfer, limiting its practicality.

That said, the onboarding process in the final presentation was implemented successfully and shown to be tested on a Huawei P10 (with an 8-megapixel front camera) and a Huawei P20 (with a 24-megapixel front camera). The transfer rate at 12- and 24-bytes per packet were tested, both of which took 5 minutes and 40 seconds. Even though a shorter time was expected for the 24-byte transfer, the devices took longer to detect and evaluate the larger QR codes.

As it stands on GitHub, the standalone application developed in this project is not adapted to the latest security updates and is thus not functional. As the project’s interface was integrated with group 12’s functionality into the BACnet application, the version found in group 10’s repository<sup>5</sup> is the most up-to-date.

## 5 Conclusion & Learnings

Overall, this project was successful - an interface was created and implemented, which transferred data safely between two Android devices through QR codes, could synchronize with BACnet databases, and could be integrated into the BACnet application developed by another group. That said, the process to this finished product was a difficult one, mainly due to the lack of in-person work due to the COVID-19 crisis. Digital tools like Zoom<sup>6</sup> were helpful as they provide video chat, screen sharing and even screen control functionalities, which helped overcome limitations on pair programming. The change in process meant key collaborative tasks such as problem solving, especially with testing, had to be done mostly alone, making the co-development of the interface very disjointed. Communication between both project members was quite seamless, although communication with other groups was more difficult - initial ideas early on in the project which would have enabled more efficient application integration were rejected by external groups, leading the developed interface to also have its own application that finally was not used. As

<sup>5</sup><https://github.com/cn-uofbasel/BACnet/tree/master/groups/10-KotlinUI>

<sup>6</sup><https://zoom.us/>

well, external groups relied on different ways of communication among themselves, which led to misunderstandings and some frustration.

In addition to developing better communication, this project went through its own complications, as originally the interface developed was to have transferred data over sound, not QR codes. Difficulties working with the quiet library<sup>7</sup> in Android Studio led to many bugs and restrictions in testing. Due to another group dropping out, the project was then allowed to pivot to QR codes. All things considered though, the individual project and larger course BACnet were a positive experience for this group's members.

## 6 Future Work

Recommendations for extending the functionality of this QR data transfer tool would mainly be in the area of capacity and efficiency. For capacity, increasing the amount of data a QR can hold would greatly improve the effectiveness of this interface. It was also found that the front-facing camera quality played a large role in limiting the packet sizing - either this tool could be implemented on better quality front-facing cameras, or a functionality using regular, back-facing, cameras could be developed. The project's source code can be found in the overall BACnet repository<sup>8</sup>.

## References

- (2020). Zxing. [Online; accessed 12-July-2020].
- Gregersen, E. (2019). Qr code. [Online; accessed 16-July-2020].
- Richter, F. (2019). Infographic: The world's most popular email clients. [Online; accessed 16-July-2020].
- Tschudin, C. (2020). Bacnet architecture. [Online; accessed 12-July-2020].

---

<sup>7</sup><https://github.com/quiet/org.quietmodem.Quiet>

<sup>8</sup><https://github.com/cn-uofbasel/BACnet/tree/master/groups/02-soundLink>

## Appendix

### A Secure Scuttlebutt protocol stack

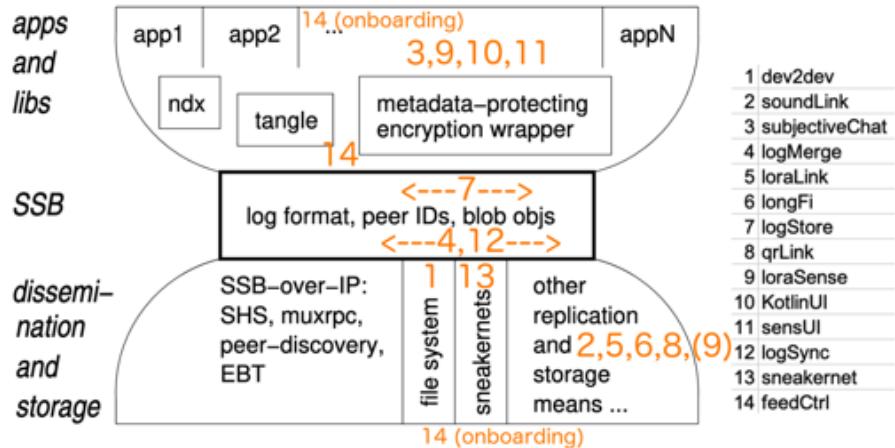


Figure 2: Secure Scuttlebutt's protocol stack.

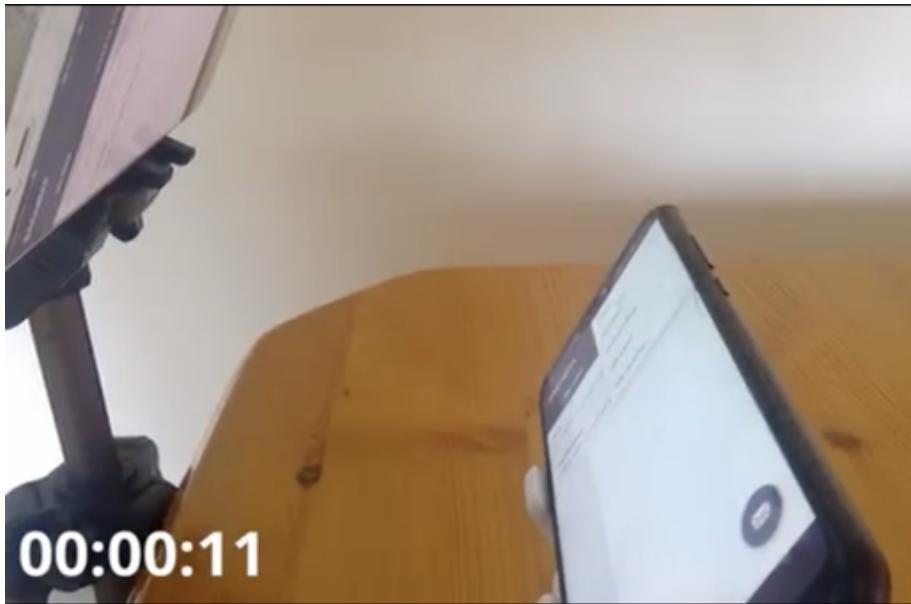
(Tschudin, 2020)

### B logSync Protocol

Three connections have to be established between two devices. The protocol looks like shown below:

	Device A			Device B
1	- Creates "I HAVE"-list			
2	- Sends "I HAVE"-list	-		
		-		
3			-	- Receives "I HAVE"-list
4				- Compares list with own entries
5				- Creates "I WANT"-list
6			-	- Sends "I WANT"-list
		-		
7	- Receives "I WANT"-list	-		
8	- Filters events			
9	- Creates EVENT-list	-		
		-		
10			-	- Receives EVENT-list
11				- Synchronisation

## C Android Device Setup



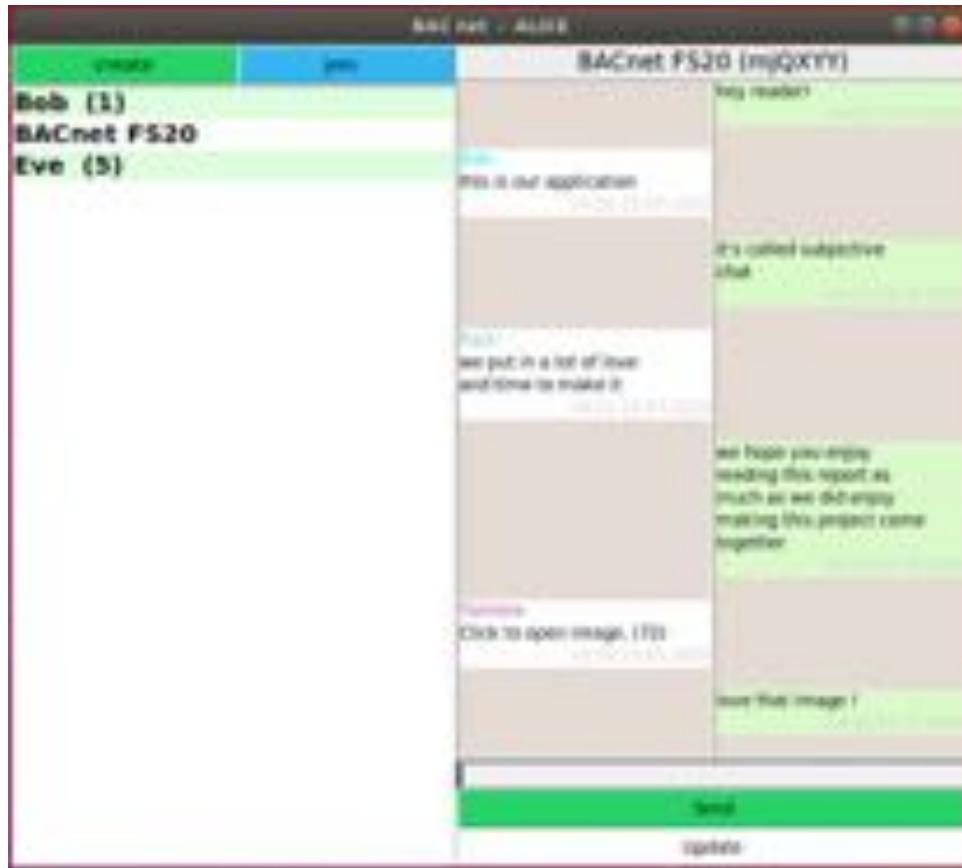


# Subjective Chat

## Ein intuitives User Interface

Computer Science - Universität Basel

fertiggestellt am: 15.07.2020



Autoren: Tunahan Erbay

Ken Rotaris

Dozent: Prof. Christian Tschudin

Assistenten: Christopher Scherb

Claudio Alexander Marxer

---

# Inhaltsverzeichnis

<b>Executive Summary</b>	<b>3</b>
<b>Einleitung</b>	<b>4</b>
Auftrag	4
Ziele	4
Technologien	5
<b>Prozess</b>	<b>6</b>
Arbeitsaufteilung	6
Design	6
Implementation	8
Erste Annäherungsversuche	8
Prototyp	9
Fertigung der grundlegenden Applikation	9
Verschicken von Bildern & PDFs	9
Finale Präsentation	10
<b>Funktionalitäten</b>	<b>10</b>
Allgemein	10
Login	10
Main-Window	11
Kontaktliste (linker Arbeitsbereich)	11
Private Chats (rechter Arbeitsbereich)	11
Gruppenchats (rechter Arbeitsbereich)	12
Updating	13
Logout/Beenden	13
<b>Diskussion</b>	<b>13</b>
<b>Fazit</b>	<b>14</b>
<b>Lessons Learned</b>	<b>14</b>
<b>Appendix</b>	<b>15</b>
Abhängigkeiten	15
Screenshots	16
Meilensteine	18

---

## Executive Summary

Das Projekt “Subjective Chat” stellt ein interaktives und intuitives User Interface dar, mit deren Hilfe der Benutzer Zugang zu den unteren Schichten im BACnet-Stack erhält (siehe Kapitel Abhängigkeiten). Die Applikation stellt sicher, dass der Benutzer die relevanten Daten zum Austauschen von Nachrichten einfach visualisieren kann und auch neue Daten der Datenbank einfach “anhängen” kann (für Näheres siehe Kapitel Auftrag).

Die Anforderungen an unser Produkt sind in diesem Report schriftlich festgehalten (siehe Kapitel Ziele). Anhand dieser Anforderungen haben wir den Fokus auf die grundlegenden Funktionalitäten gelegt und später auch die Applikation mit zusätzlichen Funktionalitäten erweitert.

Mithilfe der Applikation können sich die User (einmalig) ein “Login” erstellen und sich damit einloggen. Nachdem dies geschehen ist können die User Privat- & Gruppenchats erstellen oder joinen. Sie können sich, wie üblich bei Chat-Applikationen, gegenseitig Nachrichten, Bilder & PDFs versenden. Sie haben dabei immer einen Überblick über alle Chats. Die Chats und die Kontaktliste werden “quasi-automatisch” aktualisiert (Siehe Kapitel ‘Funktionalitäten’ für mehr Details).

Bei der Implementation sind wir auf einige interessante Probleme gestossen, die es zu lösen galt. In diesem Bericht erfährt der Leser, was diese Probleme waren, wie sichergestellt wurde, dass die Funktionalitäten der Applikation möglichst intuitiv designt wurden, was dabei unsere Design Kriterien waren und wie weitgehend die implementierten Funktionalitäten reichen.

Die Haupt-Problemstellung war für uns die Applikation intuitiv zu gestalten. Zu diesem Zweck haben wir uns am Anfang Umfragen, Wireframes und der Methoden der Vorlesung “Konzeption und Design von User Interfaces” bedient (siehe Unterkapitel Design). Somit konnten wir erstmals definieren, wie die Applikation in groben Zügen aussehen soll. Die Basis war dabei ein “Whatsapp Klone”, da 100% der befragten Personen Whatsapp benutzen und Whatsapp in unserer ersten Umfrage einen “Intuitiv Score” von Ø 4.91 aufwies.

Eine abschliessende Umfrage, um den “Intuitiv Score” der Applikation zu messen, scheiterte an den Softwarekenntnissen (Installation der Programmierumgebung und Module bei den Befragten). Von Mitstudierenden haben wir immerhin die vollen 5/5 erhalten. Die Intuitivität ist ein Faktor, welches schwer zu messen ist. Wir kommentieren deshalb kurz an dieser Stelle und erachten das Projekt als gelungenes Projekt (siehe Kapitel ‘Fazit’ für Begründung), das unsere eigenen Erwartungen definitiv übertrffen hat.

---

## Einleitung

### Auftrag

Unsere Problemstellung lag darin, aufbauend auf den Applikationen der anderen Gruppen eine Subjective Chat Applikation zu implementieren, die als Virtualisierungsschicht dient um sämtliche Daten, die für den User relevant sind, **visuell darzustellen**. Der Fokus wurde dabei auf die **intuitive und einfache Bedienung** der Applikation gelegt. Es gab viele Fragen, mit denen wir anfangs konfrontiert waren. In welche Richtung soll es gehen? Wie vermeiden wir zu grosse Abhängigkeiten von anderen Gruppen um Wartezeiten zu umgehen? Wie weitreichend gestalten wir die Funktionalitäten der Applikation? Wie implementieren wir die verschiedenen Aktionen auf eine übersichtliche, intuitive Art und Weise? Wir haben möglichst methodisch versucht, zu all diesen Fragen optimale Lösungen zu finden.

### Ziele

Aus dem Auftrag heraus haben wir anfangs die folgenden Grundanforderungen an unsere Applikation gesetzt:

- Sie muss **intuitiv** sein im Sinne einer leicht verständlichen Bedienung.
- Sie soll den Usern erlauben, zu zweit, also privat, miteinander zu chatten (**Privat-Chat**).
- Sie soll den Usern erlauben in Gruppen zu chatten (**Gruppen-Chat**).
- Sie muss **funktionieren im Zusammenspiel mit den anderen Applikationen** im BACnet-Stack.

Als zusätzliche Leistung hatten wir folgende erweiterte Anforderungen an unsere Applikation gesetzt:

- Die Applikation soll fähig sein, **PDFs** zu versenden und anzuzeigen.
- Die Applikation soll fähig sein, **Bilder** in unterschiedlichen Formaten zu versenden und anzuzeigen.
- **Fragmentierung:** Die Applikation soll sich dem “Bottleneck” des BACnets anpassen. Dies waren die Übertragungsgruppen mit einer 3 KB Übertragungsleistung.
- Versendete Dateien sollen auf dem Endgerät im korrekten Format abgespeichert werden damit die übertragene Datei auch ausserhalb der Applikation **als Datei verfügbar** ist (eindeutige Ordnerstruktur und eindeutige Namen damit nichts überschrieben wird).
- Erstellen eines **Logos**

---

## Technologien

Für die Programmierumgebung gab es zwei Optionen: Java und Python. Wir haben zuerst darüber nachgedacht, Java zu wählen, denn die GUI Programmierung im Java-Modul “JavaFX” bietet einiges mehr an Funktionalitäts- und Designmöglichkeiten und wir zusätzlich bereits mit JavaFX vertraut waren. Uns war nicht klar, wie stabil die Verbindung zwischen Java und Python (alle anderen Gruppen benutzten Python) sein wird. Nach einer Recherchearbeit standen wir vor der Wahl zwischen einer Verbindungsstelle, die aus UDP Sockets bestand oder aus einem C++-Verbindungsstück, über das die beiden Programmiersprachen kommunizieren können. Kurz gesagt, wäre es ein unsicherer und unnötiger Umweg zum Ziel. Schlussendlich haben wir uns für **Python** entschieden, um die Integrität mit den anderen Gruppen zu bewahren.

Für das GUI standen für uns “PyQt” und “Tkinter” zur Auswahl. Wir haben uns für das in Python integrierte **Tkinter** entschieden.

Um in Tkinter mit Bildern zu hantieren, benötigt es zusätzlich das Modul **“Pillow”** (Python Imaging Library).

Um den letzten Stand des Programms (nach dem Beenden) zu rekonstruieren, haben wir das Modul **“Pickle”** verwendet. Diese Daten sollen jedoch nicht in der Datenbank selbst abgespeichert werden, sondern nur lokal verfügbar sein. Das Pickle Modul ermöglicht uns genau das verschlüsselt zu tun. Wir können damit Variablen in einem lokalen File abspeichern und wieder rausholen. Die relevanten Daten werden nach dem Beenden der Applikation mit dem Modul in einem “.pickle”-File gesichert und beim Starten der Applikation wieder rausgeholt, um den letzten Zustand wiederherzustellen.

Tkinter unterstützt es nicht, eigene Fonts zu installieren und zu benutzen. Wir sind dies umgangen mithilfe des **“Pyglet”** Moduls. Damit laden wir die Font in das Memory und täuschen so vor, dass die Font bereits installiert ist.

Um das Verschicken von Bilder und PDFs zu ermöglichen, mussten wir diese Kodieren und Dekodieren. Wir sind dabei die konservative Schiene gefahren und haben uns zu diesem Zweck für **“base64”** entschieden. Dies ermöglichte es uns, Bilder und PDFs in Byte Strings zu Konvertieren.

---

## Prozess

### Arbeitsaufteilung

Die Arbeitsteilung fand in enger Zusammenarbeit statt. Das ermöglichte uns das Continuous Integration Prinzip anzuwenden. Wir haben das Projekt in kleine Teilaufgaben unterteilt und daraus haben wir dann ToDo-Listen für die jeweiligen Meilensteine angefertigt und diese dann abgearbeitet. Somit konnte man auch gut parallel arbeiten, ohne sich gegenseitig auf die Füsse zu treten. Denn während einer z.B. die Sprechblasen (Farbe, Zeit, evtl. Name, etc.) implementierte, konnte der andere sich mit den Textfeldern, Buttons, etc. auseinandersetzen. Eine Tabelle, welche zeigt, wer welche Teilaufgaben von wem implementiert wurden, würde schnell überlaufen daher wird an dieser Stelle darauf verzichtet. Wir hatten das Endziel immer vor Augen und haben uns anhand von uns selbst gesetzten Meilensteinen (mehr dazu im Appendix) organisiert. Bei Fragen und Unklarheiten, welche per "einfacher" Kommunikation nicht gelöst werden konnten, wurde Partnerprogramming angewendet.

### Design

Um unsere Applikation für die Benutzer so intuitiv wie möglich gestalten zu können, haben wir uns die Meinungen von aussenstehenden Personen zur Hilfe geholt. Aufgrund von Corona war diese Befragung auf unseren engeren Umkreis (Familie und Freunde) beschränkt. Wir haben sie gefragt<sup>1</sup>, welche Chat-Applikationen sie am häufigsten benutzen und welche für sie am intuitivsten zu bedienen sind:

Man sieht in der Grafik auf der nächsten Seite, dass Discord am intuitivsten wahrgenommen wurde. Jedoch handelte es sich hier nur um 2 Personen weshalb die Aussage, dass Whatsapp am intuitivsten ist, mehr Aussagekraft hat (da Whatsapp sowohl am meisten genutzt, als auch mit 4.91 die zweithöchste Bewertung hat). Aufgrund von dieser ersten Befragung haben wir uns für ein Whatsapp angelehntes Design entschieden. Dabei sind diese Daten selbstverständlich nicht absolut repräsentativ<sup>2</sup>, weil es sich um eine kleine Gruppe von lediglich 27 Personen handelte. Es diente für uns lediglich als verifizierbarer Kontext, mit dem wir unsere Design-Entscheidung besser treffen konnten.

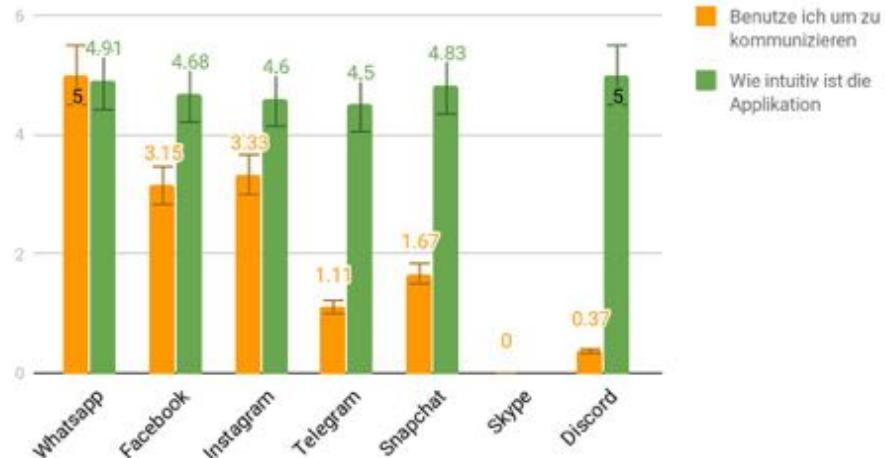
---

<sup>1</sup> Die Befragung fand auf verschiedenen Wegen statt, manche wurden per Whatsapp durchgeführt, manche als Nebensatz in einem persönlichen Gespräch und andere übers Telefon.

<sup>2</sup> Hier sieht man eine aktuelle und räpresentative Verteilung der Nutzung der verschiedenen Chat-Applikationen:

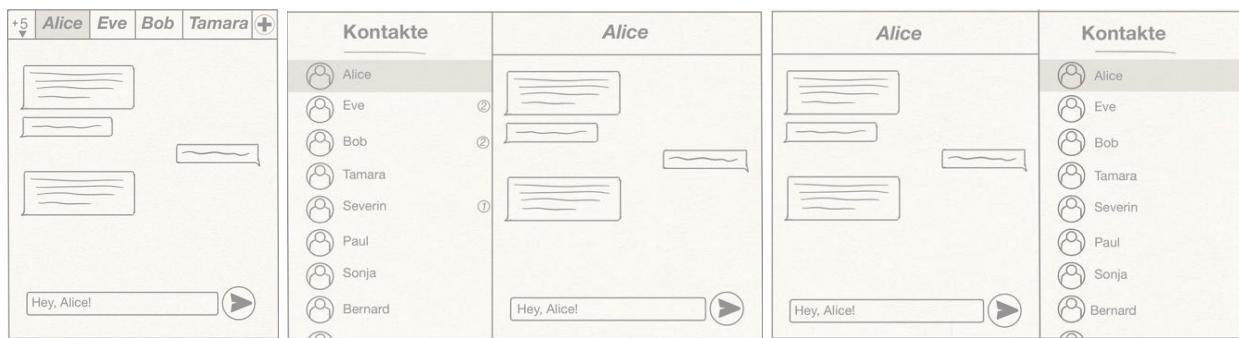
<https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/> von 2020

## Benutzte Chat Applikationen



3

Da aber unsere Applikation eine PC-Applikation sein sollte und die erwähnten Applikationen von oben meist auf dem Mobiltelefon verwendet wurden, haben wir uns dazu entschieden, eine zweite Umfrage zu starten. Zu diesem Zweck haben wir Wireframes angefertigt, in denen die Oberfläche der Applikation auf verschiedene Arten organisiert wurde und ein Feedback dazu verlangt wurde. Diese Umfrage fand im engeren Freundeskreis statt und betraf nicht mehr alle 27 Personen der vorherigen Runde. Folgend sieht man unsere angefertigten Wireframes von denen eines ein Tab-basiertes Design (Wireframe 1) darstellt, wie dies bei Browsern der Fall ist und dazu noch zwei weitere, weniger kompakte Varianten mit der Kontaktliste einmal links und einmal rechts (Wireframes 2 & 3).



1.

2.

3.

Aufgrund der Rückmeldung die wir erhalten haben, wurde uns von den allermeisten angeraten, das Design 2 (Mitte) zu wählen.

<sup>3</sup> Die orangen Daten wurden der Übersicht halber (damit es vergleichbar ist im selben Chart-Prozentual ausgedrückt und dann mit 5 multipliziert (5 entspricht also 100% was 27 Personen entspricht in unserer Umfrage). Hier sind die unverfälschten Daten in der Reihenfolge von links nach rechts: 27, 17, 18, 6, 9, 0, 2

---

Wir hatten damit eine solide Basis, auf der wir unser Applikationsdesign möglichst intuitiv aufbauen konnten. Da wir das Seminar “Konzeption und Design von User Interfaces” besucht hatten, konnten wir die dort verwendeten Methoden bei diesem Projekt gut einsetzen. Basierend darauf haben wir uns dann Stück für Stück weiter herangetastet an eine optimal intuitive Applikations-Oberfläche. Unsere Haupt-Designentscheidungen waren:

- Statt Error und Info Pop-ups, verwenden wir die beiden betreffenden Text Felder als Anzeige der Fehlermeldung oder Aktionsaufforderung. Wird in das Feld geklickt, so wird diese Meldung sofort gelöscht und muss nicht vom User zuerst manuell gelöscht werden.
- Es funktionieren nicht nur die Buttons, sondern immer auch das Anklicken der Enter-Taste für einen möglichst guten Flow in der User Experience.
- Statt vieler einzelner Knöpfe, haben wir die verschiedenen Aktionen innerhalb eines Menüs untergebracht. Wir haben uns dabei entschieden, nicht ein langweiliges normales Menü mit Unterkapiteln zu kreieren, sondern ein kontinuierliches Knopf-Menü (siehe ‘Screenshots’ Bilder 3 - 6) zu implementieren, dass die relevanten Knöpfe und Textfelder aus dem GUI entfernt und hinzufügt beim Gang durchs Menü. Dies soll die Benutzerfreundlichkeit erhöhen und das Verständnis erleichtern. Es handelt sich beim Algorithmus um einen durch viel Trial and Error selbst erzeugten zyklischen Algorithmus der dafür verantwortlich ist, die richtigen und **nur** die relevanten Knöpfe im Menü, zur richtigen Zeit anzuzeigen. Die Methoden “saveTypeAndSwitchState” und “switchState” können gerne im Code aufgesucht werden unter:

BACnet/groups/Demo B/src/subjective\_chat.py (Zeile 522 - 644)

## Implementation

Wie weiter oben bereits angedeutet, haben wir unsere Applikation mittels dem “Continuous Integration” Verfahren implementiert. Untenstehend sind die Phasen und Versionen, die unsere Applikation durchgegangen ist.

### Erste Annäherungsversuche

Anfangs waren wir uns unklar darüber, wie der gewünschte Prototyp aussehen sollte. Uns war nach der Erstbesprechung nicht ganz klar, dass die von Herrn Tschudin erwähnte Datenbank bereits auf Git vorhanden war. Wir hatten keine Möglichkeit, um Nachrichten von einem Client zum anderen zu senden. Deshalb haben wir ganz am Anfang einen Prototyp entwickelt, welcher mit der Aufgabe 3 (vom Übungsblatt 1 der Internet & Security Vorlesung) verknüpft wurde. Dieser Prototyp hat dann zwar auch funktioniert und man konnte damit den Server starten, sich mit den anderen Clients verbinden und die Daten der anderen Clients erhalten. Im Hintergrund wurden Nachrichten dann auch verschickt, aber, da die Aufgabe mit `select()` geschrieben wurde, ist uns das Tkinter User Interface an dieser Stelle trotz Threads immer eingefroren.

---

## Prototyp

Als wir dann die bereitgestellte Demo-Datenbank (unter BACnet/src/demo/lib/) hatten, haben wir unser Programm so umgeschrieben, dass er mit dieser Datenbank funktioniert. Wir haben bis zum ersten “offiziellen Meilenstein” ein Prototyp implementiert, welcher mit seinen elementaren Funktionen läuft (anmelden, Nachrichten schreiben, Nachrichten erhalten).

Der Prototyp diente nur dazu, um die Funktionalität der Applikation zu zeigen. Obwohl Tkinter sehr limitiert ist, liess sich das UI optisch sehen, war jedoch noch lange nicht fertig. Der Prototyp war also ein Annäherungsversuch an einen Whatsapp Clone mit sehr beschränkter Funktionalität.

## Fertigung der grundlegenden Applikation

Nachdem wir die Grundfunktionalitäten implementiert hatten bestand das nächste Ziel darin, das UI schöner zu gestalten und diese mit der Datenbank von Gruppe 7 funktionieren zu lassen, die zu diesem Zeitpunkt noch nicht ganz fertig waren (es waren noch keine Schnittstellen verfügbar).

Das Design des UI stand dank der Befragungen schon einigermassen fest und es gab nur noch kleinere Detail-Veränderungen am geplanten Design, um das Ganze optisch schöner zu machen (z.B. die Buttonpositionen, -ausrichtungen, -grösse, Schrift, etc.). Das Herumspielen & das Nachlesen mit/über Tkinter hat uns viel Zeit gekostet, nicht nur weil es für uns Neuland war. Sondern vor allem auch aufgrund der sehr eingeschränkten Funktionalität. Dank einiger Workarounds gelang uns dies dann auch.

Um unsere Applikation von der Demo-Datenbank abzuhängen und mit der “neuen” Datenbank, (welche von Gruppe 7 zur Verfügung gestellt wurde), zu verknüpfen, mussten wir viele grundlegende Stellen im Code anpassen. Als wir die Funktionen eingefügt und die grundlegende Logik in unserem Code angepasst hatten, waren wir mit der eigentlichen Applikation fertig.

Ausserdem haben wir noch zusätzlich ein Loginfenster implementiert, welche lokal ein “Keyfile” erzeugt, falls noch keines existiert. Um das Ganze auch wirklich eine Applikation nennen zu können, fehlte natürlich noch ein Logo. Wir haben dazu ein Logo kreiert welches die Aufgabe des BACnets grafisch ausdrückt. Dieses Logo wird sowohl im Login Fenster, als auch als Favicon der App angezeigt (siehe Appendix > Screenshots). Hier muss noch gesagt werden, dass dieses BACnet-Logo nur ein Vorschlag von unserer Seite ist. Wir haben es jedem aus BACnet FS20 in unserem Repository zur Verfügung gestellt.

## Verschicken von Bildern & PDFs

Als die Grundfunktionalitäten implementiert waren, wollten wir die Applikation erweitern. Als Studenten versenden wir öfters Bilder und/oder PDFs, deswegen wollten wir diese Funktion ebenfalls implementieren. Die Details sind, falls gewünscht, im Kapitel “Funktionalitäten”

---

nachzulesen. Die Bilder/PDFs konnten also nun verschickt und beim Anklicken angezeigt werden. Am Anfang haben wir nur “.png” unterstützt jedoch wurde dies später das auf die vielen “üblichen” Bildformate erweitert.

## Finale Präsentation

Für die finale Präsentation war unsere Gruppe der Demogruppe B zugeteilt. Wir haben in dieser Gruppe die Organisation übernommen und den anderen Gruppen bei Problemen bezüglich ihrer Applikation in der Demo so gut wie möglich geholfen.

Ausserdem haben wir im Verlaufe der letzten Tage erfahren, dass in der Demogruppe D eine Applikation für die Präsentation der Daten fehlt und haben unsere Applikation deshalb zur Verfügung gestellt. Wir mussten zusätzlichen Aufwand betreiben, damit dies schlussendlich auch erfolgreich über die Bühne geht, aber hatten dadurch eine Absicherung, dass die live Demo in mindestens einer der Demogruppen funktionierte<sup>4</sup>.

In beiden Demo-Gruppen lief alles nach Plan und wir konnten zeigen, dass unsere Applikation im Zusammenspiel mit anderen Gruppen einwandfrei funktioniert.

## Funktionalitäten

### Allgemein

In den Textfeldern stehen Anweisungen wie z.B. “Enter Group-Name”. Bei falschem/ungültigem Input erscheinen Fehler-Messages in den Textfeldern. Die Textfelder werden jedoch beim Anklicken geleert. Der eingegebene Text, kann entweder mit anklicken des jeweiligen Bestätigungsbuttons oder aber auch mit betätigen der Enter-Taste bestätigt/versendet werden. Diese sind kleine Details, die aber sehr stark der Verbesserung der Intuitivität dienen.

### Login

Wenn man die Applikation zum ersten Mal startet öffnet sich zuerst ein Log-in-Fenster (siehe ‘Screenshots’ Bild 1). In diesem kann man einen Usernamen eingeben. Dieser Name kann beliebig sein (beschränkt in der Länge und durch Sonderzeichen). Wenn man einen gültigen Namen eingetippt hat, erzeugt es beim Bestätigen, ein Key-File und speichert es lokal ab.

---

<sup>4</sup> Es war zu jenem Zeitpunkt (zwei Tage vor der Präsentation) unklar ob die Anstrengungen Gruppe 12 und Gruppe 6 zu helfen, sich auszahlen werden und zu einer funktionierenden Demo führen. Dies war abgesprochen mit Herrn Tschudin und ist (wie bereits im persönlichen Gespräch erwähnt) keinesfalls eine Anschwärzung an die betroffenen Gruppen sondern soll aufzeigen, dass wir über unser eigenes Projekt hinaus extra weit gerannt sind, um das Ganze zu einem erfolgreichen live Demo zu bewegen.

---

## Main-Window

Das Main-Window (siehe ‘Screenshots’ Bild 2) ist aufgeteilt in zwei Arbeitsbereiche, einen linken und einen rechten. Im linken Bereich hat man den Überblick über die Kontaktliste und kann Chats erstellen und bestehenden Chats beitreten (siehe README für das How-To). Im rechten Bereich wird immer der aktive (Privat-/Gruppen-) Chat angezeigt.

Kontaktliste (linker Arbeitsbereich):

Beim Erstellen eines neuen Chats erscheint dieser Chat ganz oben und wird geöffnet (→ als aktiver Chat im rechten Bereich angezeigt).

Dabei ist der Partnername von der Art des Chats abhängig: Ist dies ein Privatchat, wird vorübergehend ein Eintrittscode als Partnername gewählt, womit dann der Partner dem Chat joinen kann. Sobald der Partner dann “gejoint” ist, wird dieser Code mit dem echten Partnernamen ersetzt. Bei Gruppen-Chats steht immer der jeweilige Gruppenname.

Die Kontaktliste ist dynamisch gestaltet und wird bei jedem Update aktualisiert, dabei gelangt der aktuellste (der mit der zuletzt erhaltenen Nachricht) Chat zuoberst.

Ausserdem haben wir noch einen Counter eingebaut, welches bei jedem Kontakt, die Anzahl der ungelesenen Nachrichten darstellt. Das hat dazu geführt, dass wir nun auch bei jedem Chat speichern, wann er zuletzt geöffnet wurde. Nebenbemerkung: Die unlesbaren Nachrichten (dazu gleich mehr) werden nicht mitgezählt (siehe ‘Screenshots’ Bild 8).

Private Chats (rechter Arbeitsbereich):

Private Chats (siehe ‘Screenshots’ Bild 7) sind wie es auch der Name schon sagt, privat, also unter 2 Personen. Das haben wir auch so restriktiv: Beim Joinen, wird überprüft, ob da bereits 2 Personen drin sind und wenn das der Fall ist erscheint eine Fehlermeldung, ansonsten wird man in den Chat “reingelassen”. Ausserdem ist es auch verboten, einem Chat zu joinen, in dem man schon drin ist (mit sich selbst chatten).

Wie bei Whatsapp: Der Hintergrund ist ein ganz helles Braun. Ganz oben steht der Name vom Partner. In der linken Spalte stehen die Nachrichten vom Partner mit einem weissen Hintergrund. In der rechten Spalte sind die eigenen Nachrichten mit einem hellgrünen Hintergrund. Diese beiden Spalten/Listboxen sind miteinander synchronisiert worden, damit das Scrollen angenehmer für den User ist.

---

Nachrichten werden mithilfe der von uns implementierten Algorithmen im “TextWrapper.py” in die Spalten passend umgebrochen. Dabei wird auf die Länge der Wörter und auf die Silbentrennung geachtet: Silben getrennt wenn möglich und ansonsten Buchstaben getrennt.

Es gibt zu den normalen Nachrichten auch Nachrichten, die zwar in der Datenbank gespeichert sind, aber für die User nicht sichtbar sind. Dafür haben wir eine “zusätzliche Schicht” auf die Applikationsschicht draufgelegt. Das haben wir eingeführt damit wir Informationen über diesen Chat (Chat-Art, Anzahl-Mitglieder, etc.) auch mit in die Nachricht packen können. Jedes Mal wenn jemand einen Chat erstellt schreibt er automatisch die Chat-Art (private/group) und seinen eigenen Namen als ungelesene Nachricht rein. Und jedes Mal wenn jemand in ein Chat beitritt, schreibt er ebenfalls automatisch eine ungelesene Nachricht mit seinem Namen rein. Somit haben auch beide Seiten sofort die Namen ihrer Partner ohne dass sie “manuell” etwas eintippen müssen. Diese ungelesenen Nachrichten können dann gezählt werden, wenn man wissen will, wie viele User bereits in einem Chat sind und somit können die Restriktionen auch überprüft und eingehalten werden. Anhand der aller ersten Nachricht kann die Chat-Art bestimmt werden und wenn es ein Privater Chat ist, dürfen maximal 2 Personen dem Chat beitreten.

Das haben wir gemacht, indem wir die Nachrichten mit Splits versehen haben. Dabei sagt die Anzahl der Splits aus, ob eine Nachricht im Chatfenster ausgegeben werden sollte oder nicht.

Zusätzlich haben wir zum Schluss noch das Versenden von Bildern und PDFs implementiert. Damit diese Funktion für alle Gruppen funktioniert, mussten wir die Nachrichten (alles in String umgewandelt) in kleine Stücke aufteilen und so versenden. Wir wussten von der LoraSens-Gruppe, dass sie maximal 3 KB pro Nachricht unterstützten. Auch wenn sie nicht mit uns in der Demogruppe waren, wollten wir diesen Limit ebenfalls unterstützen. Um sicherzugehen fragmentieren wir die Dateien in 1 KB Stücke. Wir haben zusätzlich eine maximal übertragbare Dateigröße von 100 KB festgelegt, um die Applikation nicht zu überlasten.

Am besten lässt sich das Ganze anhand eines Beispiels erklären: Ein Bild wird in 20 Nachrichten partitioniert. Die ersten 19 Nachrichten sollen nicht lesbar sein für den User, sondern erst die letzte Nachricht. Denn erst dann sind alle Stücke da und können zusammengesetzt werden. Diese Nachricht besteht dann aus einer Anweisung (siehe ‘Screenshots’ Bild 9), die klar kommuniziert, dass der User auf diese Nachricht klicken soll, um die Datei zu öffnen.

#### Gruppenchats (rechter Arbeitsbereich):

Die Gruppenchats (siehe ‘Screenshots’ Bild 8) besitzen die gleichen Funktionalitäten wie ein Privatchat, jedoch mit kleinen Unterschieden: Hier besteht der Partnername, welcher ganz oben gezeigt wird aus dem Gruppennamen + der Eintrittscode in Klammern. Somit können immer neue Leute der Gruppe beitreten. Die Restriktion mit max. 2 Personen gilt hier nicht. Aber man kann trotzdem nicht in eine Gruppe zwei Mal joinen.

---

Ausserdem müssen im Chat zusätzlich die Namen der Partner auch noch ausgedruckt werden: Wenn aus der Kontaktliste eine Gruppe angeklickt wird, und der Chat geöffnet wird, entnimmt man die Chat-Art der allerersten Nachricht im Chat und wenn es ein Gruppenchat ist, kommt vor jeder Nachricht noch der Absendername dazu, damit jeder weiss, wer was geschrieben hat. Das wollten wir wie bei Gruppenchats in Whatsapp farbig machen. Anstatt zufällige Farben zu wählen, haben wir eine Funktion geschrieben, welcher zu jedem Namen eine eigene Farbe zuweist. Die Methode "Colorize" nimmt als Input einen Namen und gibt einen dafür eindeutigen Hex-Code als Output zurück.

## Updating

Das Ziel war, dass es eine Patchwork-artige Applikation wird. Also dass man einen Update-Button hat, womit man manuell updaten kann. Wir haben die Idee eines automatischen Updatings in Betracht gezogen, jedoch schnell gemerkt, dass das nicht so sinnvoll wäre, da die Übertragung nicht so schnell geschehen würde (z.B. Sneakernet) und automatisches Updaten kein grosses Nutzen bereitstellt. Daher hat auch Herr Tschudin uns davon abgeraten da Zeit zu investieren.

Wir haben aber eine Annäherung an ein automatisches Updating implementiert: Egal wohin der User klickt, es wird bei jedem Klick geupdated. Man muss nicht unbedingt, den Update-Button betätigen. Auch wenn man Nachrichten mittels Send-Button oder mittels der Enter-Taste sendet, wird geupdated. Textfelder, Back-Buttons, Listboxes, etc. alle führen zu einem Updating der Nachrichten. Dadurch wurde die Applikation dynamischer und lebendiger.

## Logout/Beenden

Beim Beenden der Applikation werden alle relevanten Daten (Kontaktliste samt Chat-IDs, Name, etc.) lokal in einem .pickle-File abgespeichert. Diese werden dann beim Neustarten der Applikation wieder geladen. Ohne diesen Schritt würden wir alle Chats nach dem Beenden verlieren und müssten für jede Person, mit der wir schreiben wollen nochmals ein Chat erstellen.

## Diskussion

Wir waren sehr motiviert für das Projekt und haben sehr viel Fleiss und Zeit reingesteckt, aber dafür lässt sich das Resultat unserer Meinung nach zeigen. Alles in allem haben wir unsere eigenen Erwartungen und die selbst gesetzten Anforderungen übertroffen. Unsere Applikation ist offen für weitere Erweiterungen in viele Richtungen. Man könnte sie zum Beispiel um eine Tondateien-Unterstützung erweitern. Dies wäre auch nicht so aufwendig. Aber es ist für uns Studenten (oder auch allgemein) eine uninteressante Erweiterung, da es nicht mehr so gängig ist Audiodateien zu versenden. Jedoch wäre das der Grundstein für die Audioaufnahme-Funktion,

---

die wir in Betracht gezogen hatten, wie es diese auch in Whatsapp und weiteren Chat-Applikation gibt. Erst zusammen mit der Audioaufnahme-Funktion wäre diese Erweiterung interessant genug. Diese würde aber unseren Zeitrahmen sprengen, so gern wir auch diese Funktion implementiert hätten. Deswegen haben wir die Audiodatei-Unterstützung weggelassen. Ausserdem könnte man wenn man mehr Zeit hätte allgemein das Versenden von Files weiter ausbauen und das Limit von 100 MB erhöhen.

## Fazit

Unser Ziel war es, eine Applikation zu implementieren, welche die Kommunikation innerhalb des BACnets für den Endbenutzer erlaubt. Sie sollte gut integrierbar sein mit den anderen Gruppen und am wichtigsten: Sie soll einfach & intuitiv zu bedienen sein. Dies ist nicht wirklich messbar und bedeutet nicht für jeden dasselbe. Wir haben aus einer Umfrage und aus persönlichen Erfahrungen möglichst viel versucht rauszuholen. Unser Produkt scheint relativ intuitiv zu sein. Die Mitstudenten wussten wie sie die Applikation bedienen sollen, ohne das wir gross was sagen mussten. Das mag daran liegen, dass unsere Applikation in Sachen Design und Farbwahl stark an Whatsapp angelehnt ist. Das war aber gewollt, da wir wissen, dass Whatsapp heutzutage fast von allen gebraucht wird oder das jeder zumindest einmal in Kontakt damit kam. Whatsapp hat nun mal einen Standard verschaffen und ist zu einem Vorbild von allen anderen Chat-Applikationen geworden. Von diesem Standard wegzukommen würde die Intuitivität nur erschweren. Daher haben wir uns für einen “Whatsapp-Clone” entschieden, jedoch unterscheidet unsere Applikation sich in vielen Punkten davon. Es war uns sehr wichtig, dass sehr wenige Aktionen zur Verfügung stehen, wie dies z.B. bei Apple-Geräten der Fall ist. Statt sämtliche Menüpunkte als Buttons anzuzeigen, haben wir sehr viel Zeit in ein intuitives Flow-Menü reingesteckt. Im Grossen und Ganzen haben wir, unserer Meinung nach, unsere Ziele erreicht. Die Applikation ist mit den anderen Gruppen gut integriert und intuitiv zu bedienen.

## Lessons Learned

Gelernt haben wir einiges. Wir haben viel Verantwortung in der Demo B übernommen, indem wir die Organisation, Deadlines und Führung auf Anfrage der anderen Gruppen übernommen haben. Dies hat uns einen kleinen Einblick in die Projektleitung gegeben und uns in einer konstruktiven zwischenmenschlichen Kommunikation zum Erreichen eines gemeinsam gesetzten Ziels geschult. Da Tkinter in seinen Möglichkeiten sehr limitiert ist, haben wir mit einigen Workarounds gearbeitet um das zu erreichen, was wir wollten. Dies war mit Trial and Error und mit viel Recherchieren verbunden, aber hat uns die Augen geöffnet und hat einige Grenzen, die wir in der Programmierung bisher gesehen haben, aufgelöst. Alles in Allem wurde das Programmieren für uns beide zu einem fluideren Gewebe, dass man zu seinen eigenen Anforderungen manipulieren und ergänzen kann, wenn man die Recherchezeit und den Fleiss reinsteckt.

---

## Appendix

### Abhängigkeiten

Unsere Lernziele und auch das Arbeitstempo waren sehr stark vom Fortschritt der anderen Gruppen abhängig, da wir auf der Applikationsschicht sind und auf die unterliegenden Schichten aufbauen. Um dies zu mildern, haben wir wie oben beschrieben am Anfang die Aufgabe 3 von Blatt 1 und dann Herr Tschudins Demo Datenbank verwendet. Die Zurverfügungstellung unserer Applikation für Demo Gruppe D hat weitere Abhängigkeiten mit sich gebracht.

Dabei waren wir am meisten von der **Gruppe 7 (logStore)** abhängig. Sie stellte uns auf Anfrage 3 Methoden zur Verfügung, womit wir Daten in die Datenbank ablegen und wieder rausziehen konnten:

- *insert\_chat\_msg(): Einfügen einer neuen Nachricht in die Datenbank*
- *get\_full\_chat(): Herausholen sämtlicher Nachrichten aus der Datenbank*
- *get\_chat\_since(): Herausholen sämtlicher Nachrichten seit einem gewissen Zeitpunkt*

Diese drei Methoden sind alle auf Strings basiert. Dies wurde gegen Schluss zu einer interessanten Herausforderung, da wir uns entschieden haben, Bilder und PDFs zu unterstützen.

Um Nachrichten in die Datenbank ablegen zu können, mussten die Nachrichten als Event in einen CBOR zusammengefasst sein. Bis ca. 5 Tage vor der finalen Abgabe, hatten wir das im Code selbst drin. Doch zu diesem Zeitpunkt hat die **Gruppe 4 (logMerge)** ein “EventcreationTool” zur Verfügung gestellt. Und damit die Integrität mit allen Gruppen erhalten blieb, mussten wir das in unserem Code ebenfalls einbauen. Auch wenn das Umändern wieder Zeit gekostet hatte, hat es sich schlussendlich gelohnt, da dieses Tool das Erstellen und Verwalten von keyfiles erleichtert. Aber auch einheitlich von allen anderen Gruppen benutzt wird, was wiederum das Integrieren vereinfachte.

Die **Gruppe 14 (feedCtrl)** hat uns ihre Applikation zur Verfügung gestellt, womit man den sozialen Radius verwalten kann. Diese Applikation wurde uns erst gegen Ende eingeführt und hat einige Änderungen in unserem Code bezweckt, was uns einige Nachtschichten gekostet hat.

Es gab noch zwei weitere Gruppen, von denen wir zwar nicht bei der Implementation abhängig waren, aber bei der finalen Präsentationen. Da wir in dieselbe Demogruppe (B) zugeteilt waren:

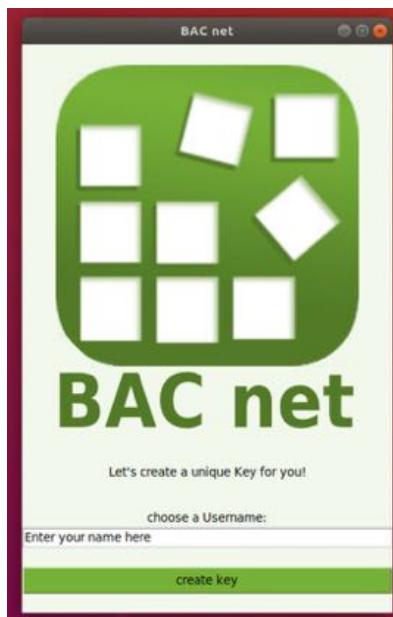
Die **Gruppe 12 (logSync)** war zuständig dafür, dass zwei Computer bzw. Datenbanken synchronisiert wurden. Diese Gruppe war also essentiell für die Präsentation, da ohne sie nicht gezeigt werden konnte, dass die Applikationen nicht nur lokal funktionieren. Es gab einige Probleme bei der Integration dieser Gruppe. Deshalb haben wir bei der Betreuung der Gruppen

besonderen Fokus auf diese Gruppe gelegt, da wir mit Gruppe 7 & 14 bereits gut zusammengearbeitet hatten und da keine Fehlerquellen sahen. Wir konnten zusammen mit Gruppe 14 diesen Fehler beheben und auch sie 12 erfolgreich in unsere Demogruppe einbinden.

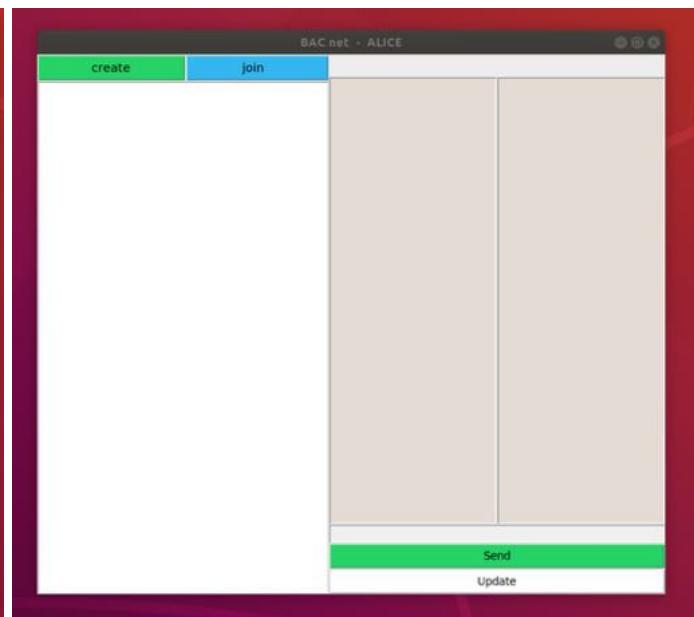
Die **Gruppe 6 (longFi)** sollte die Datenübertragung über Antennen verwirklichen. Jedoch hatte diese Gruppe Probleme, die wir und die anderen Gruppen so nicht beheben konnten. Wir konnten also keine Daten übertragen. In letzter Minute hat die Gruppe 12 uns jedoch ihren UDP Protokoll, mit der Sie ihre Synchronisations-Applikation getestet hatte, zur Verfügung gestellt. Somit hatten wir eine Alternative gefunden, die wir in der Präsentation benutzen konnten.

## Screenshots

Um sich ein Bild von der Applikation zu machen (falls man sie noch nie gesehen/gestartet hat), sind hier einige Ausschnitte aus der Applikation. Man könnte aber die Applikation auch einfach selbst starten. Dazu gibt es ein How-To im README-File in unserem Repository.



1) Login-Fenster



2) Hauptfenster ohne Kontakte / gestarteten Chats



3) Buttons zum erstellen / joinen von Chats



4) Privat- & Gruppenchat-zweig (erscheint nur beim Klicken von create)

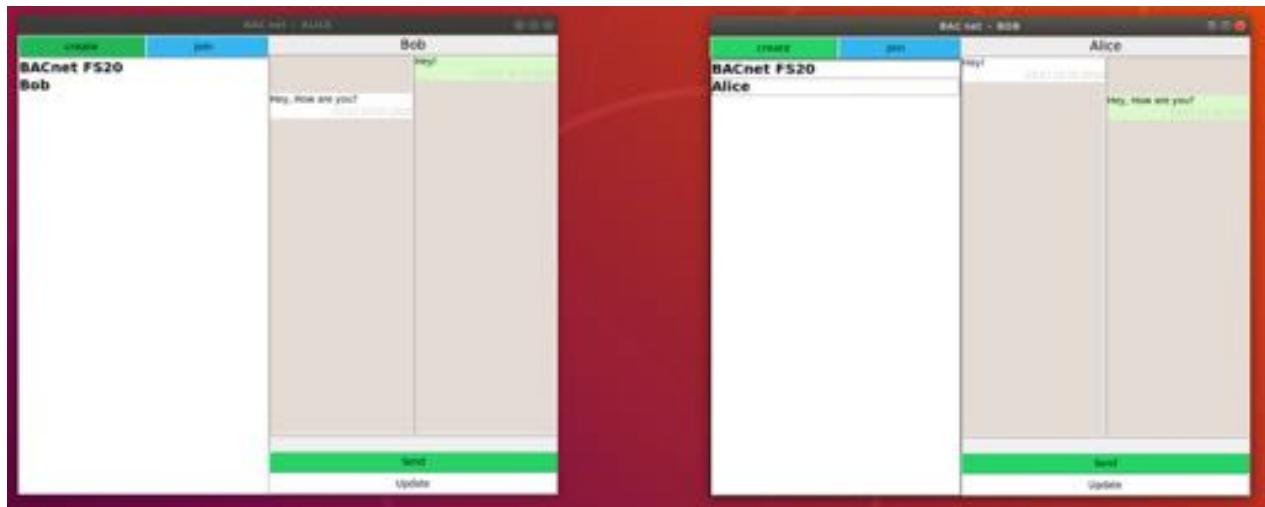
---

Enter Group name here	OK	back
-----------------------	----	------

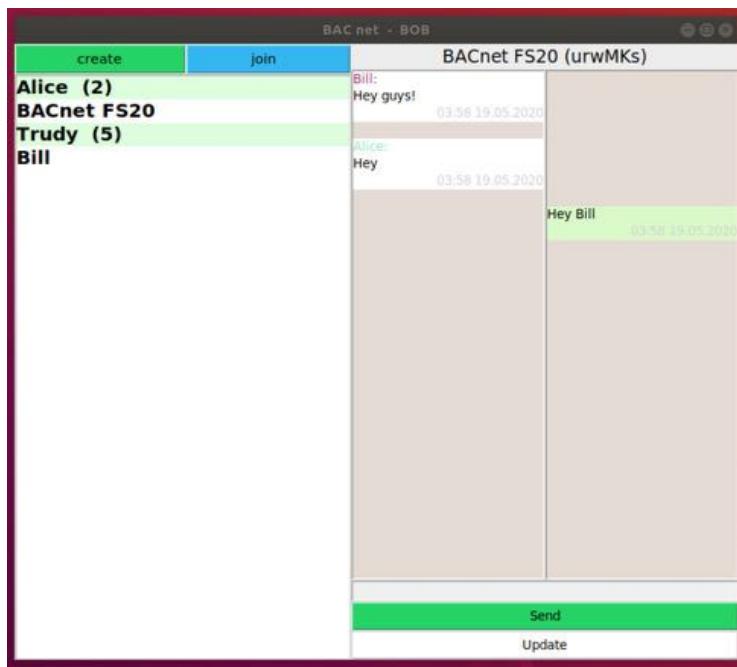
5) Auswählen eines Gruppennamens beim Erstellen von einem Gruppenchat

enter ID	OK	back
----------	----	------

6) Eingeben vom Chat-ID, welcher benötigt wird um in einen Chat zu joinen



7) Das Aussehen von einem Privatchat aus zwei Perspektiven



8) Die dynamische Kontaktliste mit dem Counter (links), Gruppenchat (rechts)

Tamara:  
Click to open image. (72)  
14:50 15.07.2020

9) "Instruktion-nachricht" bei einem erhaltenen (oder gesendeten) Bild.

10) Titelbild: Das Fertige Produkt. Alles in einem.

## Meilensteine

Hier haben wir kurz unsere Meilensteine grob aufgelistet. Der Leser kann daran erkennen, wie wir unser Projekt grob organisiert haben und wir gehen später auf einige Punkte näher ein. Die **fett** markierten Meilensteine waren vorgegeben.

1. Bestimmung der Programmierumgebung und Design - 25.03.2020
2. Erste Implementation (mithilfe von IAS Übung 1) - 01.04.2020
- 3. Vorträge Projektzwischenstand:** Prototyp fertig anhand Demodatenbank von Herrn Tschudin - 16.04.2020
4. GUI fertig (samt ursprünglich geplante Funktionalitäten) - 30.04.2020
- 5. Koordination Integration:** Integration der Applikation mit der Datenbank von Gruppe 7 - 07.05.2020
6. Zusätzliche Funktionalitäten integrieren: - Dies fand etwas verstreut statt aber war soweit fast fertig am 13.05.2020
- 7. Fertige Applikation:** Neu-Integration mit der abgeänderten Datenbank von Gruppe 7 und 14 (zusammen jetzt) - 20.05.2020
- 8. Projektvortrag:** Das gesamte Produkt - 27.05.2020

Introduction to Internet and Security

Frühjahrssemester 2020

Projektarbeit

# BACnet – logMerge (Gruppe 4)

*Günes Aydin, Joey Zgraggen, Nikodem Kernbach*

# Inhaltsverzeichnis

<b>1. Abstract</b>	<b>1</b>
<b>2. Einführung und Rahmenbedingungen</b>	<b>1</b>
2.1. BACnet . . . . .	1
2.2. Feeds/Logs . . . . .	2
2.3. Konzept der Masterfeeds . . . . .	2
2.4. Einordnung unseres Projekts . . . . .	3
<b>3. LogMerge</b>	<b>3</b>
3.1. libcap-Format . . . . .	4
3.2. Funktionsweise . . . . .	4
3.3. Arbeitsteilung LogMerge . . . . .	4
<b>4. EventCreationTool</b>	<b>4</b>
4.1. Eigenschaften . . . . .	5
4.2. Funktionsweise . . . . .	5
4.3. Arbeitsteilung EventCreationTool . . . . .	5
<b>5. Fazit</b>	<b>5</b>
<b>A. Referenzen</b>	<b>6</b>
<b>B. Spezifikation LogMerge (Englisch)</b>	<b>7</b>
<b>C. Spezifikation EventCreationTool (Englisch)</b>	<b>9</b>

## 1. Abstract

Im Rahmen der Vorlesung Introduction to Internet and Security wurde ein dezentrales Netzwerk implementiert, in welchem die Daten lokal auf jedem Gerät in einer Datenbank gespeichert werden. Durch diese Architektur ist gegeben, dass die Daten zwischen Geräten synchronisiert werden müssen, um Informationen auszutauschen. Das Ziel unserer Gruppe war es eine Schnittstelle an Transportgruppen anzubieten, mithilfe derer es möglich ist Daten zwischen lokalen Datenbanken zu synchronisieren. Hierzu kann die Transportschicht von uns Daten anfragen oder den Import neuer Daten einleiten. Ergebnis unserer Arbeit ist eine benutzerfreundliche Schnittstelle, die sich automatisch um Duplikate, Verifizierung des Urhebers, Prüfung der Datenintegrität respektive -kontinuität und die Weiterleitung der Daten an die Datenbank kümmert. Hierbei wenden wir stets einen externen Filter an, der nur vertrauenswürdige Datenströme zulässt.<sup>1</sup> Außerdem entwickelte unsere Gruppe ein Werkzeug, welches das standardisierte Formatieren von Daten ermöglicht, sodass diese in ein für das BACnet bestimmtes Format (Feeds und Events) gebracht werden. Dieses Werkzeug, welches auch von einigen anderen Gruppen verwendet wurde, kümmert sich automatisch um das korrekte Verketten, Signieren und Hashen der Events.

## 2. Einführung und Rahmenbedingungen

Innerhalb der Vorlesung Introduction to Internet and Security wurde an der Implementierung eines dezentralen Netzwerks gearbeitet, das BACnet getauft wurde. BACnet steht hierbei für „Basel Citizen Network“, bei dem die Grundfunktionalität auch ohne Internet verfügbar ist. Daten werden zum Beispiel mithilfe von USB-Sticks oder Direktverbindungen zwischen Geräten übertragen. Dieses Projekt wurde in kleinere Teile zerlegt, die je von einer Gruppe von zwei bis drei Studenten bearbeitet wurden. Wir entwickelten ebenfalls zwei Tools, die in den folgenden Kapiteln erläutert werden. Am Ende wurden alle Teilprojekte zu einem grossen BACnet zusammengeführt. BACnet lehnt sich lose an das bereits existierende dezentrale Netzwerk „Secure Scuttlebutt“ [5] an.

### 2.1. BACnet

Da BACnet ein dezentrales Netzwerk ist, liegen die Daten nicht zentral auf einem Server, sondern auf jedem Gerät in einer lokalen Datenbank. Ein Gerät speichert hierbei nur Daten, die für den Nutzer des Geräts auch relevant sind. Applikationen können außerdem neue Daten in die Datenbank schreiben. Daraufhin müssen alle anderen lokalen Datenbanken auf den aktuellen Stand gebracht werden. Die Datenübertragung kann auf verschiedene

Weisen erfolgen, zum Beispiel mithilfe von USB-Sticks oder Bluetooth.

Man kann das BACnet grob in drei Schichten einteilen:

- **Die Anwendungsschicht** (auch Applikationsschicht genannt), ist die Schicht, auf der die Daten ins Netzwerk kommen. Hier befinden sich alle Programme, die sich an die Endnutzer des BACnet richten. Darunter fallen zum Beispiel Messenger oder interaktive Spiele. Dabei ist es wichtig, dass die Daten von der Applikation in das richtige, vom BACnet verarbeitbare Format gebracht werden. Im BACnet werden alle Daten in Feeds organisiert, die wiederum in Events aufgeteilt sind. Das Konzept von Feeds und Events wird in Kapitel 2.2 näher erläutert.
- **Datenbank und Verwaltung:** Kernstück des BACnet ist die lokale Datenbank, die sich auf jedem Gerät befindet. Diese kümmert sich um das effiziente Speichern der Daten. Außerdem wird auf dieser Schicht durch Verifizierung sichergestellt, dass keine fehlerhaften oder korrupten Events in der lokalen Datenbank landen. Schlussendlich existiert auch eine Schnittstelle an die Transportschicht, welche es ermöglicht den Zustand der lokalen Datenbank abzufragen, Daten abzurufen und die Aufnahme neuer Daten einzuleiten. Das Format, in dem die Daten mit der Transportschicht ausgetauscht werden, ist das .pcap-Dateiformat, auf welches in Kapitel 3 näher eingegangen wird.

- **Die Transportschicht** ist die Schicht, auf der die Daten übertragen werden. Hier befinden sich die Programme, welche Zugang zu mehreren Geräten haben, beziehungsweise einen Kommunikationskanal zwischen zwei Geräten aufbauen können. Hauptaufgabe dieser Schicht ist die Übertragung der .pcap-Dateien, sodass ein Abgleich von lokalen Datenbanken möglich ist.

Dieser sogenannte „BACnet-Stack“ ist auch in Abbildung 1 nochmals zu sehen. Hier findet man auch eine Übersicht über die beteiligten Gruppen und wo diese im Stack einzuordnen sind. Es ist essentiell, dass sich benachbarte Gruppen auf Schnittstellen zwischen ihren Programmen einigen.

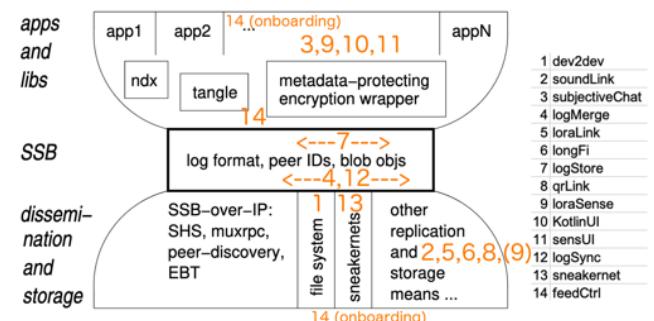


Figure 2: Secure Scuttlebutt's protocol stack.

Abb. 1: BACnet-Stack Quelle: [8]

<sup>1</sup> Diese Funktionalität wird uns von der Verwaltungsgruppe (Gruppe 14 feedCtrl) bereitgestellt

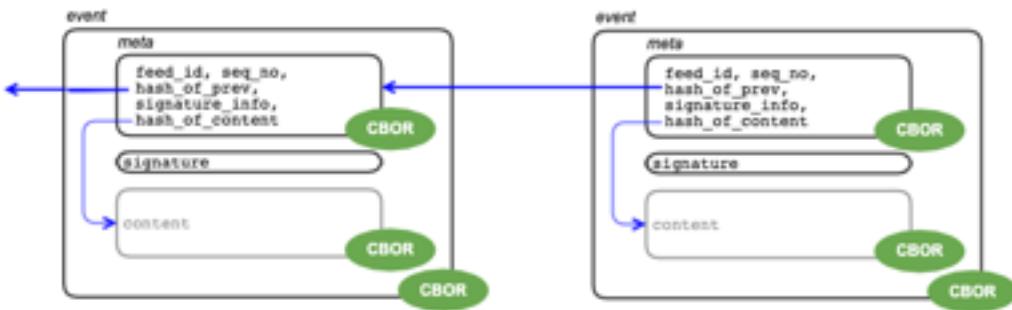


Abb. 2: Mehrere Events bilden ein Feed/Log Quelle: [9]

## 2.2. Feeds/Logs

Die Daten werden im BACnet einheitlich in Feeds beziehungsweise Logs verwaltet. Applikationen können beliebig viele solcher Feeds erstellen, um ihre Daten zu organisieren. Man kann sich einen Feed wie ein Tagebuch vorstellen. Jeder Feed hat eine eindeutige Kennung, die sogenannte Feed-ID. Er besteht aus vielen Einträgen, die Events genannt werden. Bereits getätigte Einträge können nicht mehr gelöscht werden. Der Aufbau eines Events ist genau spezifiziert und kann in Abbildung 2 eingesehen werden.

Auf die einzelnen Komponenten wird im Folgenden nochmals näher eingegangen:

- **Meta-Header:** Der Header ist die im binären cbor-Format<sup>2</sup> kodierte Liste der nachfolgenden Eigenschaften, welche für die Verwaltung und Einordnung des Events wichtig sind:
  - **Feed-ID:** Die ID, mithilfe derer man eindeutig identifizieren kann, zu welchem Feed das Event gehört.
  - **Sequence number:** Eine Ganzzahl, die die Position des Events im Feed beschreibt. Mit diesen Nummern werden fortlaufend alle Events eines Feeds durchnummeriert.
  - **Hash of previous Meta-Header:** Der Hashwert des Meta-Headers des vorangehenden Events. Hiermit kann überprüft werden, ob das Event korrekt in den fortlaufenden Feed angebunden ist.
  - **Signature info:** Kennziffer des verwendeten Signieralgorithmen.
  - **Hash of Content:** Der Hashwert des Inhalts. Hiermit kann verifiziert werden, dass der Inhalt nicht von jemandem drittem ausgetauscht wurde.
- **Signatur:** Eine Signatur über den cbor-kodierten Meta-Header. Mithilfe dieser wird festgestellt, ob das Event wirklich von dem angeblichen Autor stammt. Hierzu werden verschiedene Signieralgorithmen verwendet.

<sup>2</sup> cbor-Format: binäre Variante des beliebten JSON-Formats [10]

- **Content:** Der eigentliche Inhalt des Events, umfasst die Applikationsdaten. Hierin ist immer ein Identifikator enthalten, der aus dem Applikationsnamen und einer weiteren Zeichenfolge besteht, welche zur Identifikation der Art der Daten durch die Applikation verwendet werden kann [4].

Da jedes Event eine Signatur enthält, kann jeder im BACnet verifizieren ob das Event wirklich vom angeblichen Nutzer stammt. Um ein Event korrekt zu signieren, benötigt man einen privaten Schlüssel, den nur der Besitzer des Feeds hat. Zu jedem Feed gehört also ein privater Schlüssel, welcher beim Erstellen des Feeds durch den Besitzer generiert wird.

Es gibt verschiedene Algorithmen, die für das Signieren und Hashen verwendet werden können, hier eine Auswahl:

### • Signieralgorithmen:

- **ED25519:** Ein asymmetrischer Algorithmus, bei dem die Feed-ID gleichzeitig der öffentliche Schlüssel ist. Jeder Nutzer des BACnet kann mithilfe des Meta-Headers und der Feed-Id somit feststellen, ob ein Event vom richtigen Autor stammt.
- **HMAC\_SHA256:** Ein symmetrischer Algorithmus, bei dem es nur einen privaten Schlüssel gibt, den alle Leser eines Feeds brauchen, um ihn zu verifizieren. Die Feed-ID wird hierbei zufällig gewählt. Der Vorteil ist, dass er sich um einiges effizienter berechnen lässt, als zum Beispiel der ED25519. Allerdings muss der private Schlüssel über einen sicheren Kanal geteilt und nur an vertrauenswürdige Personen gesendet werden.

### • Hashingalgorithmen:

- **MD5:** Ein etwas älterer und mittlerweile unsicherer Algorithmus, der sich aber sehr effizient berechnen lässt.
- **SHA256:** Ein sicherer aber aufwendigerer Algorithmus.

## 2.3. Konzept der Masterfeeds

Als Nutzer des BACnet möchte man nicht alle Feeds in seine lokale Datenbank importieren und auch nicht

alle Feeds an jeden exportieren. Deshalb muss innerhalb der Datenbank- und Verwaltungsschicht eine Methode existieren, mithilfe derer man festlegen kann, welchen Feeds man vertraut und welchen nicht. Um dies zu ermöglichen wurde durch die Gruppen dieser Schicht<sup>3</sup> das Konzept der „Masterfeeds“ entwickelt. Zu einem Gerät (beziehungsweise einer Instanz einer lokalen Datenbank) gehört immer ein Masterfeed, der bei Initialisierung der Datenbank erstellt wird. Erstellt eine Applikation auf diesem Gerät einen neuen Feed, so wird dem Masterfeed ein Event zugefügt, das die Zugehörigkeit dieses Applikationsfeeds zum Masterfeed angibt. Damit allerdings kein Masterfeed einen Applikationsfeed zu unrecht als seinen eigenen beanspruchen kann, ist auch das erste Event eines jeden Applikationsfeeds ein Verweis auf den Masterfeed. Der Masterfeed beinhaltet ausserdem noch einen Nutzernamen, der beliebig geändert werden kann.

Nun kommt der wichtige Unterschied zwischen Master- und Applikationsfeeds: Den Masterfeeds wird standardmäßig vertraut, während den Applikationsfeeds misstraut wird. Dadurch werden beim Synchronisieren immer alle Masterfeeds zwischen lokalen Datenbanken abgeglichen. Da es möglich ist aus einem Masterfeed eine Liste von zugehörigen Applikationsfeeds zu extrahieren, kann ein Nutzer einem Applikationsfeed vertrauen ohne ihn explizit zu kennen, sodass dieser ab sofort ebenfalls in die lokale Datenbank importiert wird. Auf diese Weise kann ein Nutzer des BACnet neue Inhalte einsehen. Welchen Feeds ein Nutzer vertraut steht ebenfalls im seinem Masterfeed. Somit kann sogar ein sozialer Radius festgelegt werden, und man kann zum Beispiel automatisch Freunden von Freunden<sup>4</sup> auch vertrauen. An dieser Stelle wollen wir auf das Repository der Gruppe 14 feedCtrl [\[1\]](#) verweisen, die für die Masterfeeds hauptsächlich zuständig war.

## 2.4. Einordnung unseres Projekts

Unser Teilprojekt befindet sich auf der Datenbank- und Verwaltungsschicht, zwischen der Datenbankgruppe<sup>5</sup> deren Funktionalität wir über eine Schnittstelle benutzen, und der Transportschicht, an die wir eine Schnittstelle zur Verfügung stellen. Ausserdem erreichen wir über die Datenbankgruppe auch die Verwaltungsgruppe<sup>6</sup> von der wir abfragen welchen Feeds lokal vertraut wird. Mithilfe unseres Tools können Transportgruppen den Stand der lokalen Datenbank abfragen, Feeds exportieren oder den Import neuer Events einleiten. In Abbildung [3](#) ist die Einordnung von LogMerge in den BACnet-Stack nochmals visualisiert.

Ausserdem entwickelten wir noch ein weiteres Werkzeug, mit dem man Feeds und Events einfach erstellen kann. Da man über das BACnet Daten nur in Feeds übertragen kann, kann unser Tool in der Applikationsschicht

<sup>3</sup> An dieser Stelle möchten wir uns noch einmal bei den Gruppen 14 feedCtrl und 7 logStore für die tolle Zusammenarbeit bedanken!

<sup>4</sup> Freunde: Feeds denen man vertraut

<sup>5</sup> Gruppe 7 logStore

<sup>6</sup> Gruppe 14 feedCtrl

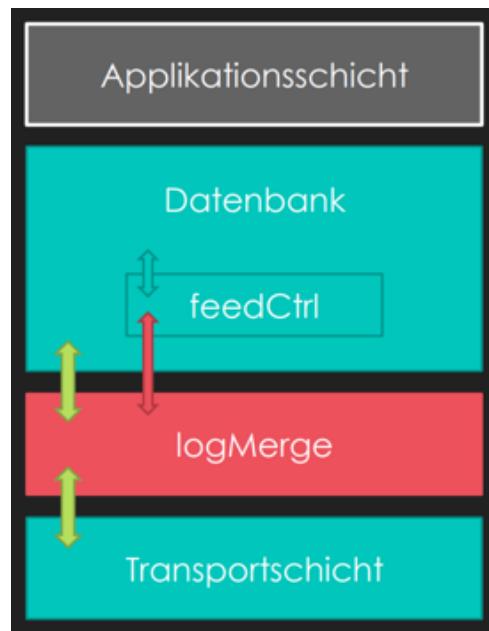


Abb. 3: Position von LogMerge im BACnet-Stack

zum Umwandeln von Daten in BACnet-fähiges Format verwendet werden. Ein anderer Anwendungsbereich ist das Testen der Projekte der Datenbank- und Verwaltungsschicht. Das Tool kann somit nicht eindeutig einer Schicht im BACnet-Stack zugewiesen werden, passt aber am ehesten in die Applikationsschicht.

## 3. LogMerge

Die Software LogMerge stellt einen Service an Gruppen der Transportschicht zur Verfügung, die mithilfe unserer Schnittstelle auf die lokale Datenbank eines gegebenen Geräts zugreifen können. Wir ermöglichen den Transportgruppen drei Funktionalitäten:

1. Statusabfrage der Datenbank
2. Importieren von neuen Feeds und Events in die Datenbank
3. Exportieren von benötigten Feeds und Events

Diese werden in Kapitel [3.2](#) nochmals genauer erläutert.

LogMerge wurde unsererseits auf den Schnittstellen der Datenbank- und Verwaltungsgruppen aufgebaut. Nur so können wir auf die Datenbank zugreifen und das Vertrauensverhältnis zu Feeds korrekt beachten. Deshalb sind die Projekte dieser Gruppen auch in unserem integriert, sodass sich Nutzer unserer Software nicht um die separate Installation der anderen Programme kümmern müssen. Allerdings gibt es einige andere Softwareanforderungen, die in der Spezifikation ab Seite [7](#) zu finden sind. In der Spezifikation finden sie auch einen praktischen „Quick start guide“. Unsere Software und die Spezifikation kann auch im Internet auf Github gefunden werden [\[6\]](#).

Zurzeit werden von unserer Software nur Feeds und Events unterstützt, die den Signieralgorithmus ED25519

und den Hashingalgorithmus SHA256 verwenden. Diese Kombination ist auch die Standardeinstellung beim Erstellen von Events mit unserem EventCreationTool (siehe Kapitel 4). Alle mit uns zusammenarbeitenden Gruppen verwendeten ebenfalls immer nur diese Kombination, da sie relativ sicher ist (siehe Kapitel 2.2). Unsere Softwarearchitektur ermöglicht allerdings die einfache Integration weiterer Algorithmen.

### 3.1. libcap-Format

Um die Arbeit im BACnet einheitlich zu halten, wurde beschlossen, dass die Daten auf der Transportschicht immer in Dateien des libcap-Formats<sup>7</sup> vorliegen. Werden also Events importiert oder exportiert, so erhalten wir diese in Form von .pcap-Dateien oder müssen selber .pcap-Dateien schreiben. Eine solche binäre Datei beinhaltet eine fast beliebige Anzahl von Events. Der Aufbau lässt sich schematisch folgendermassen darstellen<sup>8</sup>:

Globaler Header	Paket-Header	Binäres Event	
Paket-Header	Binäres Event	Paket-Header	...

Hierbei ist:

- **Globaler Header:** Immer 24 Bytes gross. Beinhaltet die Dateiversion und einige weitere Metadaten.
- **Paket-Header:** Immer 16 Bytes gross. Beinhaltet einen Zeitstempel und die Länge des nachfolgenden binären Events.
- **Binäres Event:** In cbor-Format codiertes Event, wie in Kapitel 2.2 erklärt.

Das binäre „zusammenbasteln“ einer solchen Datei erwies sich hierbei als relativ mühsam, da es keine vorgefertigten Tools zum erstellen solcher Dateien gibt. Eine weiterführende Dokumentation des Formats befindet sich im Internet [7].

### 3.2. Funktionsweise

Dieses Kapitel soll die drei Funktionalitäten (siehe Seite 3) und die dazugehörigen erforderlichen Hintergrundprozesse etwas näher beleuchten. Die Schnittstelle wird über die Klasse LogMerge zur Verfügung gestellt, von der man ein Objekt erstellen muss. Dieses bietet drei Methoden, eine pro Funktionalität. Möchte man Events aus .pcap-Dateien in die Datenbank importieren, so muss man den Pfad zum Verzeichnis dieser Dateien der entsprechenden Methode übergeben. LogMerge extrahiert daraufhin alle in den Dateien enthaltenen Events und verifiziert alle Signaturen und Hashes. Werden Unstimmigkeiten erkannt, so werden die entsprechenden Events verworfen. Ebenfalls verworfen werden alle Events von Feeds, denen der Nutzer der lokalen Datenbank nicht vertraut. Diese Information ruft unsere Software, über die Schnittstelle

<sup>7</sup> Dateien des libcap-Formats haben die Endung .pcap

<sup>8</sup> Es existiert eine theoretische Obergrenze, nach derer die Events in mehrere Dateien aufgespalten werden

<sup>9</sup> Darstellung angelehnt an die Quelle [7]

der Datenbankgruppe, bei der Verwaltungsgruppe ab. Schlussendlich werden alle übrig gebliebenen Events an die Datenbankgruppe übergeben und von dieser in der Datenbank hinterlegt.

Fragt man den Status der Datenbank bei uns ab, so geben wir die Feed-IDs aller verfügbaren Feeds zusammen mit ihrer jeweiligen höchsten Sequenznummer (Die maximale Sequenznummer unter allen Events des gegebenen Feeds) zurück. Es werden allerdings nur Feed-IDs weitergegeben, für die von der Verwaltungsgruppe grünes Licht gegeben wurde. So soll verhindert werden, dass private Inhalte an Personen weitergegeben werden, denen man nicht vertraut.

Möchte man schliesslich Events exportieren, so muss man spezifizieren, von welchen Feed-IDs und jeweils ab welcher Sequenznummer man Events benötigt. Auch hier muss zunächst die Verwaltungsgruppe grünes Licht geben. Bei erfolgreicher Vertrauensprüfung werden die zu exportierenden Events je eines Feeds in je eine .pcap-Datei im Wunschverzeichnis abgespeichert. Masterfeeds werden hierbei, auch wenn sie nicht explizit abgefragt werden, immer exportiert, damit sich neue, noch unbekannte Masterfeeds über das BACnet verbreiten können. Ausserdem hat man die Möglichkeit die maximale Anzahl der zu exportierenden Events einzuschränken, wenn man Bandbreite sparen möchte. Nutzt man diese Möglichkeit, so werden die Events mit den niedrigwertigsten Sequenznummern in die .pcap-Dateien geschrieben, sodass keine Lücken entstehen.

Unsere Schnittstelle ist auch vollständig in der Spezifikation im Anhang (ab Seite 7) festgehalten.

### 3.3. Arbeitsteilung LogMerge

Aufgrund des Corona-Virus waren persönliche Treffen unmöglich, weshalb wir über die Onlinedienste Zoom und Discord arbeiteten. Oft nutzten wir auch die Möglichkeit den Bildschirm zu teilen, und arbeiteten stets zusammen am Projekt. Die Verantwortlichkeiten verteilten wir folgenderweise:

- **Joey Zgraggen:** Lesen und schreiben von .pcap-Dateien.
- **Nikodem Kernbach:** Import von Feeds in die lokale Datenbank.
- **Günes Aydin:** Export von Feeds und Statusabfrage der lokalen Datenbank.

## 4. EventCreationTool

Während der Entwicklung unseres Projektes wollten wir unsere Software kontinuierlich auf ihre Funktionalität prüfen. Hierzu benötigten wir allerdings korrekt und inkorrekt erstellte Logs<sup>10</sup> und Events, die eigentlich nur von Gruppen in der Anwendungsschicht erzeugt werden.

<sup>10</sup> Log ist ein anderes Wort für Feed

Da uns zu diesem Zeitpunkt noch keine dieser Gruppen ein valides Log präsentieren konnte, entschieden wir uns eine Software zu schreiben, die Logs und Events korrekt erstellen kann. Daraufhin verwendeten diverse Gruppen unser Tool um ihren Code zu testen, einige Applikationsgruppen sogar standardmäßig, um Logs und Events zu erstellen.

## 4.1. Eigenschaften

Unser Tool unterstützt die Signaturalgorithmen ED25519 und HMAC \_ SHA256 um Events zu signieren, wobei ersterer standardmäßig verwendet wird. Für das Hashen wird zurzeit nur der Algorithmus SHA256 unterstützt. Wir haben ausdrücklich alle Anwendungsgruppen informiert, dass wir bereit sind weitere Algorithmen zur Verfügung zu stellen, erhielten aber keine Sonderwünsche. Die Events, welche mithilfe unseres Tools erstellt werden, kodieren wir binär im cbor-Format. Um dies einfach machen zu können, benutzten wir das Paket *cbor2* [2]. Für das Signieren wird das externe Paket *PyNaCl* [3] benutzt. Will man das Tool verwenden, so muss man den Ordner *eventCreationTool* in sein Projekt kopieren und kann dann die Klasse *EventFactory* aus der Datei *EventCreationTool.py*, wohin man sie benötigt, importieren. Das Tool wirft bei falscher Nutzung von uns definierte Fehler, welche abgefangen werden sollten. Eine Übersicht über die Fehlertypen, und die ganze restliche Schnittstelle, finden sie in der vollständigen Spezifikation des Tools ab Seite [9].

## 4.2. Funktionsweise

Dieses Kapitel soll einen kurzen Überblick über die Möglichkeiten des Tools geben. Wenn sie eine komplette Dokumentation der Schnittstelle suchen, finden sie diese im Anhang ab Seite [9]. Ein Log, beziehungsweise Feed, wird bei unserem Tool durch ein Objekt der Klasse *EventFactory* repräsentiert. Möchte man einen neuen Feed erstellen, so erstellt man ein neues Objekt dieser Klasse, wobei man optional den Signierungs- und Hashingalgorithmus spezifizieren kann. Daraufhin wird ein Schlüsselpaar erstellt und der private Schlüssel in einer *.key*-Datei im aktuellen Verzeichnis gespeichert. Möchte man einen benutzerdefinierten Speicherort verwenden, so kann man dies ebenfalls beim Erstellen des Objektes tun. Nun sollte man zunächst das erste Event erstellen, mithilfe dessen die Zugehörigkeit des Feeds zum jeweiligen Masterfeed gesetzt wird. Hierzu muss man den Namen der zugehörigen Applikation, sowie die Masterfeed-ID des Geräts der entsprechenden Methode übergeben. Zurück bekommt man das Event im cbor-Format, entsprechend der Spezifikation [4] auf dem BACnet-Repository. Um weitere Events zu erstellen, muss man nun jeweils nur noch einen Identifikator und die zu speichernden Daten an eine Methode übergeben. Der Identifikator ist hierbei der Applikationsname und eine weitere Zeichenfolge, welche die im Event enthaltenen Daten beschreibt. Die jeweils zurückgegebenen Events sollte man als Applikationsgruppe unverzüglich der Datenbank hinzufügen.

Möchte man ein *EventFactory*-Objekt zu einem bereits existierenden Feed erstellen, damit man neue Events an diesen anhängen kann, so kann man beim Erstellen des Objekts das neueste Event (bereits erstelltes Event mit höchster Sequenznummer) übergeben. Wichtig hierbei ist, auch das zugehörige Verzeichnis des privaten Schlüssels korrekt anzugeben. Das Tool kümmert sich automatisch um die Verkettung, Signierung und das Hashen der daraufhin im korrekten Format erstellten Events. Ausserdem bietet es noch weitere Möglichkeiten an, wie zum Beispiel das Auslesen der privaten Schlüssel oder Anzeigen aller Feed-IDs zu denen private Schlüssel in einem bestimmten Verzeichnis vorhanden sind. Alle Möglichkeiten sind zusammen mit einem „Quick start guide“ in der Spezifikation ab Seite [9] zu finden.

## 4.3. Arbeitsteilung EventCreationTool

Auch am EventCreationTool arbeiteten wir stets zusammen, legten aber folgende Verantwortlichkeiten fest:

- **Joey Zgraggen:** Das korrekte Signieren und Hashen der Events.
- **Nikodem Kernbach:** Sicherstellung der korrekten Verkettung der Events.
- **Günes Aydin:** Abspeichern und Laden der privaten Schlüssel.

## 5. Fazit

Das Ergebnis unserer Arbeit waren zwei unabhängige Tools: *LogMerge*, welches eine Schnittstelle für Transportgruppen zur Datenbank darstellt, sowie das *EventCreationTool*, mithilfe dessen man einfach Logs und Events erstellen kann. Das Tool *LogMerge* liegt sehr zentral im BACnet-Stack, wodurch es besonders wichtig war die Schnittstellen mit unseren Nachbargruppen auszuhandeln. Vor allem mit den Gruppen 7 (*logStore*) und 14 (*feedCtrl*) standen wir schon früh in Kontakt, wodurch wir gut auf die Integrationsphase am Schluss des Projektes vorbereitet waren. Um *LogMerge* zu testen, entwickelten wir das *EventCreationTool*, das schliesslich auch freiwillig von diversen Anwendungsgruppen benutzt wurde. Dadurch wurde sichergestellt, dass die Logs und Events im BACnet ein einheitliches Format haben, was auch die Zusammenarbeit der betroffenen Gruppen untereinander enorm vereinfacht hat. An der finalen Präsentation präsentierten wir unsere Tools als Teil der Demo D. Teil dieser Demo waren die Gruppen *dev2dev* (Gruppe 1), *subjectiveChat* (Gruppe 3), *logStore* (Gruppe 7), *sneakernet* (Gruppe 13), *feedCtrl* (Gruppe 14) und schlussendlich auch wir, *logMerge* (Gruppe 4). An dieser Stelle möchten wir uns nochmals bei allen Gruppen für die gute Zusammenarbeit bedanken. Die Projekte der anderen Gruppen finden sie im Internet [11]. Die genauen Spezifikationen unserer Schnittstellen (sowie die Funktionalitäten unserer Tools) können sie im Anhang einsehen.

## A. Referenzen

- [1] [https://github.com/cn-uofbasel/BACnet/  
tree/master/groups/14-feedCtrl](https://github.com/cn-uofbasel/BACnet/tree/master/groups/14-feedCtrl) aufgerufen  
am 15. Juli 2020
- [2] <https://pypi.org/project/cbor2/> aufgerufen  
am 15. Juli 2020
- [3] <https://pypi.org/project/PyNaCl/> aufgerufen  
am 15. Juli 2020
- [4] [https://github.com/cn-uofbasel/BACnet/  
blob/master/doc/BACnet-event-structure.md](https://github.com/cn-uofbasel/BACnet/blob/master/doc/BACnet-event-structure.md)  
aufgerufen am 17. Juli 2020
- [5] <https://github.com/ssbc> aufgerufen am 17. Juli  
2020
- [6] [https://github.com/cn-uofbasel/BACnet/  
tree/master/groups/04-logMerge](https://github.com/cn-uofbasel/BACnet/tree/master/groups/04-logMerge) aufgerufen  
am 18. Juli 2020
- [7] [https://wiki.wireshark.org/Development/  
LibpcapFileFormat](https://wiki.wireshark.org/Development/LibpcapFileFormat) aufgerufen am 18. Juli 2020
- [8] Bild auf  
[https://github.com/cn-uofbasel/BACnet/  
blob/master/doc/img/bacnet-stack.png](https://github.com/cn-uofbasel/BACnet/blob/master/doc/img/bacnet-stack.png)  
abgerufen am 19. Juli 2020
- [9] Bild auf [https://github.com/cn-uofbasel/  
BACnet/blob/master/doc/img/hash-chain.png](https://github.com/cn-uofbasel/BACnet/blob/master/doc/img/hash-chain.png)  
abgerufen am 19. Juli 2020
- [10] <https://cbor.io/> aufgerufen am 19. Juli 2020
- [11] [https://github.com/cn-uofbasel/BACnet/  
tree/master/groups](https://github.com/cn-uofbasel/BACnet/tree/master/groups) aufgerufen am 19. Juli 2020

# LogMerge

This is our project that provides an API to BACnet transport layer groups for importing and exporting events from the local BACnet database.

## Content

- Requirements and installation
- Supported signing and hashing algorithms
- Quick start guide
- Full API specification
  - API that we provide
  - API that we use

## Requirements and installation

In order to use the tool, you have to install the following python packages:

- PyNaCl
- cbor2
- sqlalchemy (for dependencies)
- testfixtures (for dependencies)

We recommend to install them using pip:

```
> pip install cbor2
> pip install pynacl
> pip install sqlalchemy
> pip install testfixtures
```

The needed dependencies to use this tool come together with our package. Just make sure you have the packages installed as mentioned above. The API provides 3 basic functions, that are described in the section [API](#). You can simply copy over the folder `logMerge` to your project and there you go!

We are thinking about creating a `pip install` package, but are currently busy. As for now, you can install the dependencies by running `pip install path/to/logMerge` (or simply `pip install .` if your command line is inside the `logMerge` folder).

## Supported signing and hashing algorithms

Currently we support the following signing algorithms:

- ed25519

And the following hashing algorithms:

- sha256

Feel free to contact us if you need different ones!

## Quick start guide

In order to use LogMerge, you need to import this whole folder to your project. Just add this folder to your source tree and follow the installation guide [above](#). The whole API is inside the file `LogMerge.py`.

When you as transport group come to us, you probably want to import the `.pcap` files that you have ready for us into the database. To do so, just do the following:

```
import LogMerge
lm = LogMerge.LogMerge()
lm.import_logs(path_to_folder_with_pcap_files)
```

, where `path_to_folder_with_pcap_files` is the path to the folder wherefrom all `.pcap` files will be imported. (All `.pcap` files from subfolders will be imported too.)

Now, that you imported your files, you probably want to know which feeds are available for export in the local database. Therefore you can use the following code:

```
status_dictionary = lm.get_database_status()
```

The `status_dictionary` will now contain all feed ids that you can get from the database and their highest available sequence numbers as values. I.e. the structure thereof is:

```
status_dictionary = {feed_id_1: max_seq_no, feed_id_2: max_seq_no, feed_id_3: max_seq_no, ...}
```

The feed ids will be of type `bytes` and the sequence numbers will be of type `int`.

Last but not least you might want to export events from the database in order to transport them to the next BACnet user. Therefore, please use the following API:

```
lm.export_logs(path_to_pcap_folder, dict_feed_id_current_seq_no)
```

In this call, the first parameter specifies the path to the folder where our program will write the .pcap files to. The second parameter should be a dictionary with all the feed ids (as bytes) you want to export as keys. The value for each feed should be the highest sequence number that you do not need (or -1 if you want all events). Only events with a higher number will be exported. Thus the dictionary should look somehow like this:

```
dict_feed_id_current_seq_no = {feed_id_1: highest_already_exported_seq_no, feed_id_2: highest_already_exported_seq_no, ...}
```

If you want to limit the events that will be written into the .pcap files, you can set the optional parameter of the `export_logs` method. Please look in the [full api](#) for further information.

## Full API specification

### API that we provide

#### class LogMerge

The class LogMerge is the API of this tool.

```
get_database_status()
```

- Returns: Type: `dict` (keys: `bytes`, values: `int`). A dictionary that contains all feed ids that you can export from this database together with the highest available sequence number as value.

```
export_logs(path_to_pcap_folder, dict_feed_id_current_seq_no, maximum_events_per_feed_id=-1)
```

- Returns: Nothing, but .pcap files are created.
- Parameters:
  - `path_to_pcap_folder`: Type: `str`. The path to the folder where the .pcap files will be saved (if any). For each feed a new file is created. If there are too many events for one .pcap file, multiple files are created.
  - `dict_feed_id_current_seq_no`: Type: `dict` (keys: `bytes`, values: `int`). A dictionary that contains all feed ids that you want to export from this database together with the highest sequence number you already have as value. Only events with higher sequence numbers will be exported. The sequence number should be -1 if you want all events of the specified feed. This must not be null! If you want just the master feeds, please pass an empty dictionary {} instead.
  - optional `maximum_events_per_feed_id`: Type: `int`. The maximum number of events that will be exported per feed. This can be used to reduce the payload of the transport medium. If this value is -1, then there is no limit of events.
- NOTE: .pcap files that are already in the folder will be **OVERWRITTEN** if the names match. Thus it is a good idea to export always to a new folder.

```
import_logs(path_of_pcap_files_folder)
```

- Parameters:
  - `path_of_pcap_files_folder`: Type: `str`. The path to the folder that contains the .pcap files that should be imported. All subdirectories are also searched for .pcap files. All events are read in from the files, verified and eventually imported into the database. (They are also checked by Group 14 first.)

### API that we need from others

We are using the API provided by Group 7 logStore and their adapter for Group 14 feedCtrl. The full specification of the API can be found [here](#).

# EventCreationTool

This is a simple tool for creating BACnet feeds and events. Current version is 1.5

## Content

- Requirements and installation
- Supported signing and hashing algorithms
- Quick start guide
  - Additional important information
- Full API specification
  - Private key file format
  - exception HashingAlgorithmNotFoundException
  - exception SigningAlgorithmNotFoundException
  - exception KeyFileNotFoundException
  - exception IllegalArgumentTypeException
  - class EventFactory
  - class EventCreationTool
- Changelog

## Requirements and installation

In order to use the tool, you have to install the following python packages:

- PyNaCl
- cbor2

We recommend to install them using pip:

```
> pip install cbor2
> pip install pynacl
```

The tool provides the two python classes `PythonCreationTool` and `EventFactory` which you can use as API. To use the tool you need the two python files `PythonCreationTool.py` and `Event.py`. Copy them over to your project (you could drop them in a separate folder) in order to use the tool.

We are thinking about creating a `pip install` package, but are currently busy. As for now, you can install the dependencies by running `pip install path/to/eventCreationTool` (or simply `pip install .` if your command line is inside the `eventCreationTool` folder).

## Supported signing and hashing algorithms

Currently we support the following signing algorithms:

- ed25519
- hmac\_sha256

And the following hashing algorithms:

- sha256

Feel free to contact us if you need different ones!

## Quick start guide

Please also look at the [Additional important information](#) below!

Once you installed the tool, you probably want to start off by creating a new feed. The simplest way to do so is by creating a new object of the class `EventFactory`:

```
import EventCreationTool
ecf = EventCreationTool.EventFactory()
```

This will create a new feed. If you want to create multiple feeds, you can simply create multiple `EventFactory` objects. The first event inside an application generated feed must be the reference to which master feed this application feed belongs. Thus you have to call the function:

```
first_event = ecf.first_event(app_name, master_feed_id)
```

Where `app_name` is the identifier of your app (the first part of your content identifiers, see next code snippet) and `master_feed_id` is the local master feed id which you have to obtain from the database (please refer to their documentation for how to do this). **DO NOT** forget to add this `first_event` to the database, or you will not be able to append new events to your newly generated feed.

You can now create events on that feed by using the `next_event()` method and passing your custom content as follows:

```
new_event = ecf.next_event(content_identifier, content_parameter)
```

When using this, please stick to the conventions for BACnet ([here at the bottom](#)). Thus `content_identifier` should be a string like '`yourapp/yourcommand`' and `content_parameter` could be whatever you want (but using dictionaries is probably most convenient). Our tool does not, however, enforce the convention.

If you restart your application and need the factory for the same feed again, you can simply obtain the last event from your database and pass that event when creating the factory object like this:

```
last_event = # Obtain the last event of the feed you want to append to from the database
ecf = EventCreationTool.EventFactory(last_event)
```

Make sure that the event you pass is really the most recent one. Otherwise the event chain of your feed will diversify. You of course must not call the `first_event` method when using this constructor, as the first event was already created in the past.

If you need the feed ids of the feeds you have the private key for, you can obtain them by using the following static method:

```
list_of_feed_ids = EventCreationTool.EventCreationTool.get_stored_feed_ids()
```

You will get a `list of bytes`, which are the feed ids so you can obtain an event from the database and therewith reinitialize the `EventFactory`ies (as stated above).

## Additional important information

The call `EventCreationTool.EventFactory()` will automatically create a `.key` file in your current working directory (i.e. the directory from which you ran the program that called the `EventCreationTool`). You can also specify another location to store your keys, as specified in the full API specification.

These files contain the private keys to your feeds. **MAKE SURE TO NOT LOSE THEM!** If you loose these files, you will not be able to create new events (and our tool will throw some custom exceptions as specified [below](#))

The tools provides even more functionality as specified in the next section. There you will learn how to set custom private key paths, obtain private keys (you need them if you use hmac signing) or even set different signing and hashing algorithms. Also, if you need a tool that allows more control, you could use the `EventCreationTool` class instead of `EventFactory`. API is specified below.

# Full API specification

## Private key file format

Whenever a feed is created, a private key is generated. This key is binary saved in a file. The naming convention we use is: `feed_id.key` (please note that the feed id and public key is the same thing.) The file will be generated by default in the current working directory but the path can be adjusted as specified below.

### exception HashingAlgorithmNotFoundException

This exception is thrown whenever you try to use a hashing algorithm that is not known to your used version of the tool.

### exception SigningAlgorithmNotFoundException

This exception is thrown whenever you try to use a signing algorithm that is not known to your used version of the tool.

### exception KeyFileNotFoundException

This exception is thrown whenever the tool needs access to a private key but this key is not available at the provided path. (For example thrown if you try to append to a feed that is not yours and you thus have not the private key.)

### exception IllegalArgumentTypeException

This exception is thrown when you pass an argument which is of the wrong type. The exception message sometimes provides a list of accepted types, but you do best if you stick to the API below.

### exception FirstEventWasNotCreatedException

This exception is thrown when you try to use the `EventFactory` to yield a `next_event` without creating a first event using the `first_event()` method first. This is used only in the highly abstract `EventFactory` class, as the `EventCreationTool` class is intended for the more advanced user which we do not protect from himself.

### exception FirstEventWasAlreadyCreatedException

This exception is thrown when you try to create a first event using the `first_event` function, but a first event was already created. This is used only in the highly abstract `EventFactory` class, as the `EventCreationTool` class is intended for the more advanced user which we do not protect from himself.

## class EventFactory

The class `EventFactory` is the recommended API of this tool.

```
__init__(last_event=None, path_to_keys=None, path_to_keys_relative=True,
        signing_algorithm='ed25519', hashing_algorithm='sha256')
```

- Returns: A new object of the class.
- Throws: `SigningAlgorithmException`, `HashingAlgorithmException`
- Parameters:
  - optional `last_event`: Type: `bytes` (cbor encoded as you can get it from the database). Event on which base the object is created. Sets up the created factory to yield following events. If this is not specified, a new feed will be created.
  - optional `path_to_keys`: Type: `str`. If you need a specific path to private keys. Can be a relative or absolute path (See following parameter). Default is the current working directory.
  - optional `path_to_keys_relative`: Type: `bool`. Must be set to `False` if the above specified path was an absolute path.
  - optional `signing_algorithm`: Type: `str`. The signing algorithm you want to use. Refer to [here](#) for supported algorithms.
  - optional `hashing_algorithm`: Type: `str`. The hashing algorithm you want to use.
- NOTE: After creating a new feed (creating `EventFactory` object not using `last_event` parameter), you will have to immediately create a first event using the `first_event()` method and add it to the database. See also in the quick start guide.
- NOTE2: `signing_algorithm` and `hashing_algorithm` will be ignored if you specify a `last_event`. In this case, the algorithms will be chosen to match the previous events of the feed. This is to protect you from misusing a feed.

```
get_feed_id()
```

- Returns: Type: `bytes`. The feed id (i.e. the public key) of the associated feed.

```
get_private_key()
```

- Returns: Type: `bytes`. The private key of the associated feed. If you need to obtain the key for a partner when using the `hmac` signing algorithms.
- Throws: `KeyFileNotFoundException`

```
first_event(app_name, master_feed_id)
```

- Returns: Type: `bytes` (cbor format). The first event of the associated feed with the correct master feed reference as specified in the parameters.
- Throws: `KeyFileNotFoundException`, `IllegalArgumentException`
- Parameters:
  - `app_name`: Type: `str`. The identifier of your application. Please use this same identifier as first part of your event identifier when using the `create_event` methods. Please stick to the conventions [here at the bottom](#). Looks somehow like this: `appname/command` **wherefrom only appname should be passed!**
  - `master_feed_id`: Type: `bytes`. The feed of your local master feed id. Please refer to the documentation of Group 7 logStore on how to obtain it and also on Group 14 feedCtrl on what it is for.

```
next_event(content_identifier, content_parameter=None)
```

- Returns: Type: `bytes` (cbor encoded). The new event that is generated together with your content.
- Throws: `KeyFileNotFoundException` (If you try to append to a feed and do not have the private key at the specified location), `FirstEventWasNotCreatedException` (If you try to create a next event without creating a first event using the `first_event()` method first.)
- Parameters:
  - `content_identifier`: Type: `str`. The identifier of the content you append. Please stick to the conventions [here at the bottom](#). Looks somehow like this: `appname/command`
  - optional `content_parameter`: Type: whatever. The content you want to save. Could be whatever but dictionary is probably most convenient. A event without this parameter(s) is also valid. Also look [here at the bottom](#)

## class EventCreationTool

If you need some more control, you can use this class instead of `EventFactory`. Using this class you do not need a separate object for each feed. It is however recommended to use separate objects for different hashing and signing algorithms (so you don't have to set the algorithm each time). Here comes the API:

```
@classmethod
get_stored_feed_ids(directory_path=None, relative=True, as_strings=False)
```

This is the only static method.

- Returns: Type: `list of bytes` or `str`. The list of those feed ids (i.e. public keys), for which the private key is present at the (specified) keys directory and thus the feeds on which ownership can be claimed and new events can be appended.
- Parameters:
  - optional `directory_path`: Type: `str`. The path to the private keys. Default is the current working directory, i.e. the root directory from which the program that called the tool run. Could be an absolute or relative path.
  - optional `relative`: Type: `bool`. Specifies whether the path passed to the previous argument is a relative or absolute path. Must be set to `False` if an absolute path is passed.
  - optional `as_strings`: Type: `bool`. If set to true, a `list of str` is returned instead of a `list of bytes`. If so, the `str` values are the hexadecimal representations of the `bytes` typed private keys.

```
set_path_to_keys(directory_path, relative=True)
```

- Parameters:
  - `directory_path`: Type: `str`. Set the path at which the private keys can be found. (The default at declaration is the current working directory.) Can be absolute or relative to the current working directory.
  - optional `relative`: Type: `bool`. Must be set to `False` if the previous parameter was an absolute path.

```
get_own_feed_ids(as_strings=False)
```

- Returns: Type: `list of bytes` or `str`. Returns all feed ids of which the private keys are stored at the currently set path to keys (This is the feed ids that belong to the user).
- Parameter:
  - optional `as_strings`: Type: `bool`. When set to `True` the method will return a `list of str`. Else or if not provided it will return a `list of bytes`.

```
set_hashing_algorithm(hashing_algorithm)
```

- Throws: `HashingAlgorithmNotFoundException` if the supplied algorithm is not supported.
- Parameter:
  - `hashing_algorithm`: Type: `str`. One of the supported hashing algorithms (see [here](#)). This algorithm will be used to hash whenever events are created with the associated `EventCreationTool` object.

```
set_signing_algorithm(signing_algorithm)
```

- Throws: `SigningAlgorithmNotFoundException` if the supplied algorithm is not supported.
- Parameter:
  - `signing_algorithm`: Type: `str`. One of the supported signing algorithms (see [here](#)). This algorithm will be used to sign whenever events are created with the associated `EventCreationTool` object.

```
get_supported_hashing_algorithms()
```

- Returns: Type: `list of str`. A list of the names of the supported hashing algorithms.

```
get_supported_signing_algorithms()
```

- Returns: Type: `list of str`. A list of the names of the supported signing algorithms.

```
generate_feed_and_create_first_event(app_name, master_feed_id)
```

- Returns: Type: `bytes` (cbor format). The first event of a newly generated feed with the correct master feed reference as specified in the parameters.
- Throws: `SigningAlgorithmNotFoundException`, `IllegalArgumentException`
- Parameters:
  - `app_name`: Type: `str`. The identifier of your application. Please use this same identifier as first part of your event identifier when using the `create_event` methods. Please stick to the conventions [here at the bottom](#). Looks somehow like this: `appname/command wherefrom only appname should be passed!`
  - `master_feed_id`: Type: `bytes`. The feed of your local master feed id. Please refer to the documentation of Group 7 logStore on how to obtain it and also on Group 14 feedCtrl on what it is for.

```
generate_feed()
```

- Returns: Type: `bytes`. The feed id (i.e. the public key as this is the same) of the newly generated feed.
- Throws: `SigningAlgorithmNotFoundException`

```
create_first_event(feed_id, app_name, master_feed_id)
```

- Returns: Type: `bytes` (cbor format). The first event of the associated feed with the correct master feed reference as specified in the parameters.
- Throws: `KeyFileNotFoundException`, `IllegalArgumentException`
- Parameters:
  - `feed_id`: Type: `bytes` or `str`. The feed id and thus public key of the feed we want to append to. The private key must obviously be in the keys path, else a custom exception is thrown.
  - `app_name`: Type: `str`. The identifier of your application. Please use this same identifier as first part of your event identifier when using the `create_event` methods. Please stick to the conventions [here at the bottom](#). Looks somehow like this: `appname/command wherefrom only appname should be passed!`
  - `master_feed_id`: Type: `bytes`. The feed of your local master feed id. Please refer to the documentation of Group 7 logStore on how to obtain it and also on Group 14 feedCtrl on what it is for.

```
create_event(feed_id, last_sequence_number, hash_of_previous_meta, content_identifier, content_parameter)
```

This method should not be used to create the first event of a feed. Please refer to `create_first_event()` instead.

- Returns: Type: `bytes` (cbor format). The event of the associated feed with content as specified in the parameters.
- Throws: `KeyFileNotFoundException`, `IllegalArgumentException`
- Parameters:
  - `feed_id`: Type: `bytes` or `str`. The feed id and thus public key of the feed we want to append to. The private key must obviously be in the keys path, else a custom exception is thrown.
  - `last_sequence_number`: Type: `int`. The sequence number of the most recent event of that feed. The sequence number of this event will be set to this number plus one.

- `hash_of_previous_meta`: Type: `list` with two elements: `int` (first) and `bytes`. The second one is the hash value for the cbor encoded meta header of the previous event, the first being an indicator for which hashing algorithm was used to calculate that hash. Please refer to [this](#) convention for more information.
- `content_identifier`: Type: `str`. The identifier of the content you append. Please stick to the conventions [here at the bottom](#). Looks somehow like this: `appname/command`
- `content_parameter`: Type: whatever. The content you want to save. Could be whatever but dictionary is probably most convenient. A event without this parameter(s) is also valid. Also look [here at the bottom](#)

```
create_event(previous_event, content_identifier, content_parameter)
```

This is the more convenient variant of the previous `create_event()` method, as you do not need to mine the information about the feed and events, but you can simply pass the last event and the tool will do the magic for you. This method should not be used to create the first event of a feed. Please refer to `create_first_event()` instead.

- Returns: Type: `bytes` (cbor format). The event of the associated feed with content as specified in the parameters.
- Throws: `KeyFileNotFoundException`
- Parameters:
  - `previous_event`: Type: `bytes` (encoded as cbor). The most recent event of the feed we want to append. The returned event will be chained after this event.
  - `content_identifier`: Type: `str`. The identifier of the content you append. Please stick to the conventions [here at the bottom](#). Looks somehow like this: `appname/command`
  - `content_parameter`: Type: whatever. The content you want to save. Could be whatever but dictionary is probably most convenient. A event without this parameter(s) is also valid. Also look [here at the bottom](#)

```
get_private_key_from_feed_id(feed_id)
```

The private key should remain a secret. This method is just visible, because you may need to share the key if you are using hmac signing. If you are not: You are better off not using this method.

- Returns: Type: `bytes`. The private key of the specified feed.
- Throws: `KeyFileNotFoundException` if the private key was not found in the keys path.
- Parameter:
  - `feed_id`: Type: `bytes` or `str`. The feed id and thus the public key of the feed we want the private key of. (In hexadecimal representation when passed as string)

```
get_private_key_from_event(event)
```

The private key should remain a secret. This method is just visible, because you may need to share the key if you are using hmac signing. If you are not: You are better off not using this method.

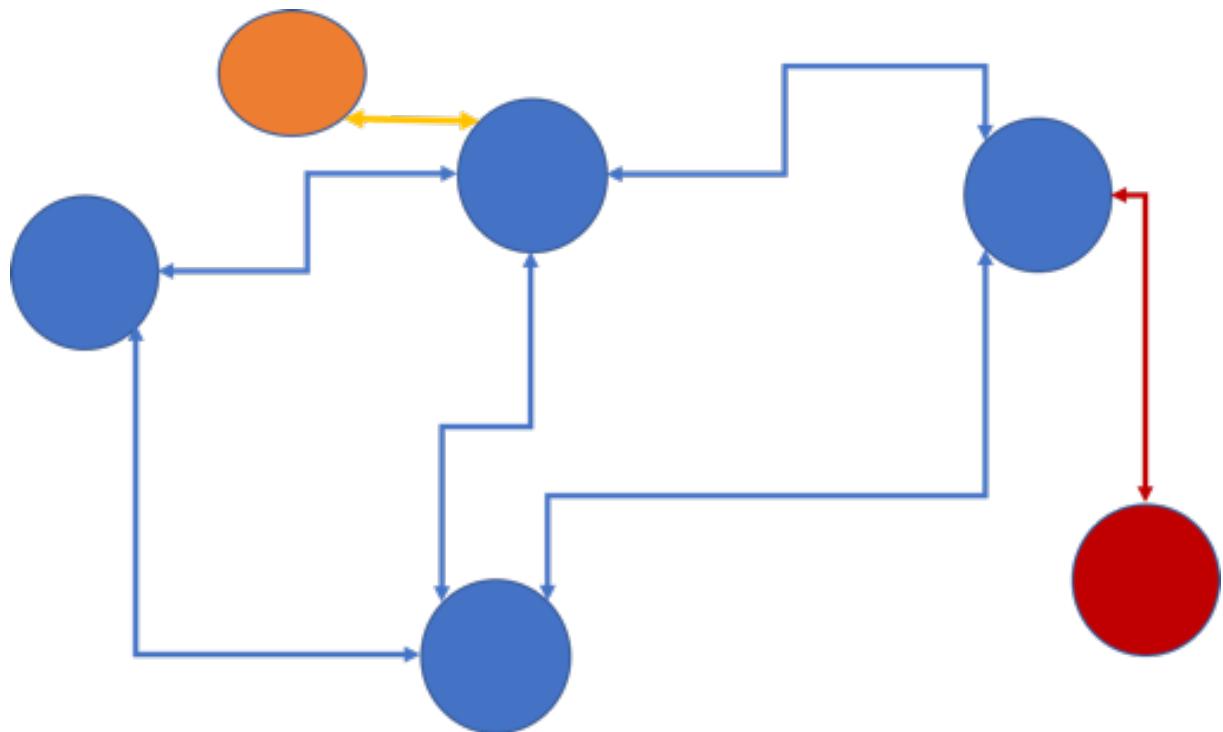
- Returns: Type: `bytes`. The private key of the associated feed.
- Throws: `KeyFileNotFoundException` if the private key was not found in the keys path.
- Parameter:
  - `event`: Type: `bytes` (cbor encoded). A event from the feed we want the private key for. The feed id and thus the public key is thereof obtained and used to find the corresponding private key.

## Changelog

- 
- V1.0: First release for extern use. API as specified [above](#).
  - V1.1: Added EventFactory class for even simpler creation of events. Also some bugfixes and renaming.
  - V1.2: Added possibility to obtain feed id from EventFactory. Added a static method to EventCreationTool that returns the feed ids of the own feeds (i.e. the ones we have private keys for). Also minor convenience changes. First version with complete README.md file.
  - V1.3: Added setup.py for easier install using `pip install ..`
  - V1.4: Added Tests for EventFactory class.
  - V1.5: Changed the tool to create the first event according to Group 14 feedCtrl Feeds specifications.

# LoRa Link

Lora basiertes Netzwerk für das Projekt BACnet



Von Patricia Heckendorf und Julian Trinkler

22. Mai 2020

Zur Vorlesung Introduction to Internet and Security

bei Prof. Dr. C. Tschudin

## 1 Zusammenfassung

Das BACnet-Projekt hat das Ziel elektronische Kommunikation ohne Internet auf verschiedene Arten zu ermöglichen. Zusammen mit der Gruppe LoraSense und der Gruppe SensUI erstellten wir ein dezentrales LoRa (Long Range/Low Power Radio) Netzwerk.

LoraSense misst mittels Sensoren, die an ein LoRa-Gerät (*Pycom, 2020*) angeschlossen sind, meteorologische Größen. Diese werden mittels unserem LoraLink Netzwerk an ein Endgerät (PC) weitergeleitet. Dort können sie mit einer Applikation grafisch dargestellt werden. Via PC kann die Wetterstation gesteuert werden. Die Kommunikation ist also bidirektional.

Das LoraLink Netzwerk besteht aus drei Schichten. Die oberste Schicht (**Feed Layer**) ermöglicht das Erstellen und das Bewirtschaften der Feeds. Via Feed Layer kann ein Knoten einen Feed abonnieren: Neuigkeiten werden automatisch an die darüber liegende Schicht (Applikationsschicht in LoraSense und SensUI) weitergeleitet.

Die **Sync Layer** beinhaltet das Gossip-Protokoll, welches zur Synchronisation der Feeds und Events zwischen den am Netzwerk teilnehmenden Knoten dient. In zufälligen Zeitintervallen teilen die verschiedenen Knoten mit, welche Feeds sie verfolgen und wie viele Events sie bereits gespeichert haben. Umliegende Knoten, die diese Gossips erhalten und aktuellere Events dieser Feeds gespeichert haben, geben das Verschicken dieser Events in Auftrag.

Die **Link Layer** ist dafür zuständig die Daten via LoRa zu senden. Dabei wird CSMA angewandt, optional kann auch CA ausgewählt werden.

## 2 Einführung und Hintergrund

Das BACnet-Projekt (*BACnet, 2020*) entstand als Projekt parallel zur Vorlesung *Introduction to Internet and Security*. Das Ziel war elektronische Kommunikation zu ermöglichen ohne auf das Internet zurückzugreifen. Das Netzwerk sollte ohne Client-Server-Architektur auskommen, sondern dezentral aufgebaut sein. Nachrichten (sogenannte Events) sollen einer Person zugeordnet werden und nicht verändert werden können, deshalb werden alle erstellten Events signiert und mittels append-only-Log (aoL) weitergegeben.

Es haben sich verschiedene Gruppen gebildet, die Teilauspekte davon umsetzen. In einem zweiten Schritt wurden verschiedene Teilgruppen zusammen geschlossen, so dass sich autonom funktionierende Prototypen des BACnets ergaben.

Als Vorbild für dieses dezentralisierte Netzwerk diente uns das Secure Scuttlebutt (*Tarr et al., 2019, Scuttlebutt, 2020*). Dieses funktioniert nach dem Prinzip publish/subscribe im Gegensatz zum Empfänger/Senderprinzip auf dem das Internet beruht.

Unsere Gruppe LoraLink ermöglicht die Umsetzung eines solchen dezentralen Netzwerks mittels der LoRa (Long Range/Low Power Radio) Technologie. Diese Technik hat den Vorteil, dass Signale über längere Distanzen (100 m -10 km) übertragen werden können, als beispielsweise mittels WiFi. Sie brauchen wenig Energie und eignen sich für abgelegene Orte. Der Nachteil dieser Übertragungstechnologie ist, dass nur kleine Datenpakete verschickt werden können. Größere Nachrichten (Chat) oder Bilder können nicht übertragen werden. Einzelne Messdaten einer Wetterstation können jedoch problemlos transportiert werden. Aus diesem Grund haben wir uns mit der Gruppe LoraSense und SensUI zusammengeschlossen um einen Prototyp einer Wetterstation-Kommunikation umzusetzen.

Das Ziel ist eine bidirektionale Verbindung zwischen einer Messstation und einem PC via LoRa-Netzwerk herzustellen. Der Knoten mit der Messstation erstellt einen Sensor Feed, dieser soll anschliessend über das ganze LoRa-Netzwerk verteilt werden. Von einem Knoten werden via WiFi Daten an ein PC weitergeleitet. Darauf sollen mit einer interaktiven Applikation die Wetterdaten dargestellt werden können. In die umgekehrte Richtung, sollen vom PC Kontrolldaten an die Messstation weitergeleitet werden.

## 3 Umsetzung

### 3.1 Grobkonzept

Mit Hilfe von Sensoren an einem LoRa - Gerät wird ein «Sensor Feed» generiert (publish) und ein read-only Log erstellt (siehe Abbildung unten LoraSense oranger Knoten). Die gemessenen meteorologischen Größen werden in gewissen Intervallen als Events an den read-only-log gehängt. In Reichweite befindende LoRa-Knoten können diese Feeds abonnieren (subscribe).

Über das LoRa-Netzwerk wird der Sensor Feed zu den umliegenden LoRa-Knoten mit Hilfe eines Gossip-Protokolls übertragen. Am anderen Ende des Netzwerks werden via WiFi die Daten auf einen PC übertragen. Hier können die Messgrößen mit Hilfe einer grafischen Oberfläche dargestellt werden (SensUI: roter Knoten).

Ausserdem kann von dieser Oberfläche das Messintervall gesteuert werden, durch kreieren von Events im Control Feed. Via das LoRa Netzwerk wird der Control Feed mit der Messstation geteilt.

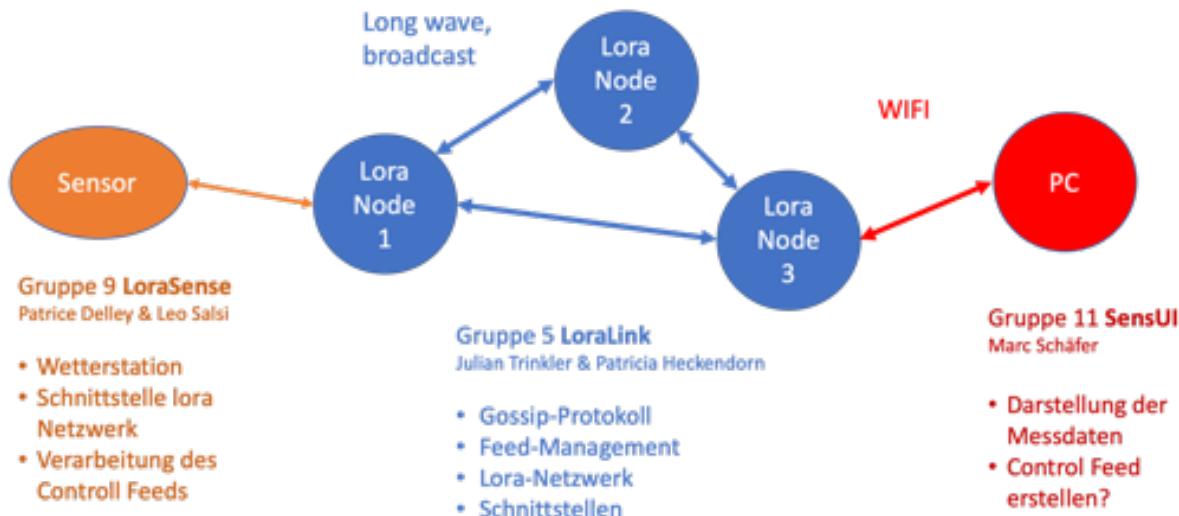


Abbildung 1 Übersicht über die Zusammenarbeit mit LoraSense und SensUI

### 3.2 Hardware und Software

Sowohl die Sensoren der Wetterstation als auch die Knoten des LoRa-Netzwerks werden mit LoPy4 Geräten von Pycom umgesetzt (*Pycom, 2020*). Es wird die für Europa freie Frequenz von 868 MHz verwendet. Für das Debugging wurden auch LoRa 32 Geräte von HELTEC genutzt, welche durch die integrierten Displays praktisch sind. Die Kommunikation zwischen LoPy4 und LoRa32 ist im Anhang 6.2 beschrieben. Innerhalb der Stadt betrug die Reichweite eines LoPy4 ca. 100 m (getestet), auf freiem Feld kann die Reichweite bis zu 10 km betragen. Die Lopy4 Geräte werden mit Micropython (*Micropython, 2020*) programmiert, wir nutzten Atom als Entwicklungsumgebung (den Code zum Projekt findet man auf Github: *LoraLink code, 2020*). Die Heltec Geräte werden mit C++ in Arduinoumgebung programmiert (*Heltec, 2020*).

### 3.3 Aufteilung der Arbeit

Gemeinsam haben wir das Grobkonzept umgesetzt. Die einzelnen Arbeitsschritte haben wir einzeln umgesetzt und dann ausgetauscht. Da die Hardware doppelt vorhanden war, konnten wir getrennt die neuen Versionen testen und verbessern. Die Präsentation und den Bericht haben wir zusammen erstellt, wobei die Zusammenarbeit via Office und Zoom gut über die Distanz funktioniert haben.

## 4 Resultate

### 4.1 LoRaLink Protokolle – eine Übersicht

Unser Netzwerk besteht aus 4 Schichten. Die oberste Schicht, die Applikationsschicht, besteht aus dem GUI um die Messdaten darzustellen und die Messintervalle einzugeben (Gruppe SensUI) sowie der Verarbeitung der Sensors Daten (Gruppe LoraSense). Darunter kommt die Feed-Layer, welche für das Erstellen der Feeds und das Bearbeiten der append-only Logs zuständig ist. Die darunterliegende Schicht Sync Layer verpackt die Events und organisiert mit dem Gossip-Protokoll welche Datagramme über das Netzwerk geschickt werden sollen. Die Link Layer als unterste Schicht ist zuständig die Pakete (Frames) via LoRa und WiFi zu verschicken und zu empfangen.

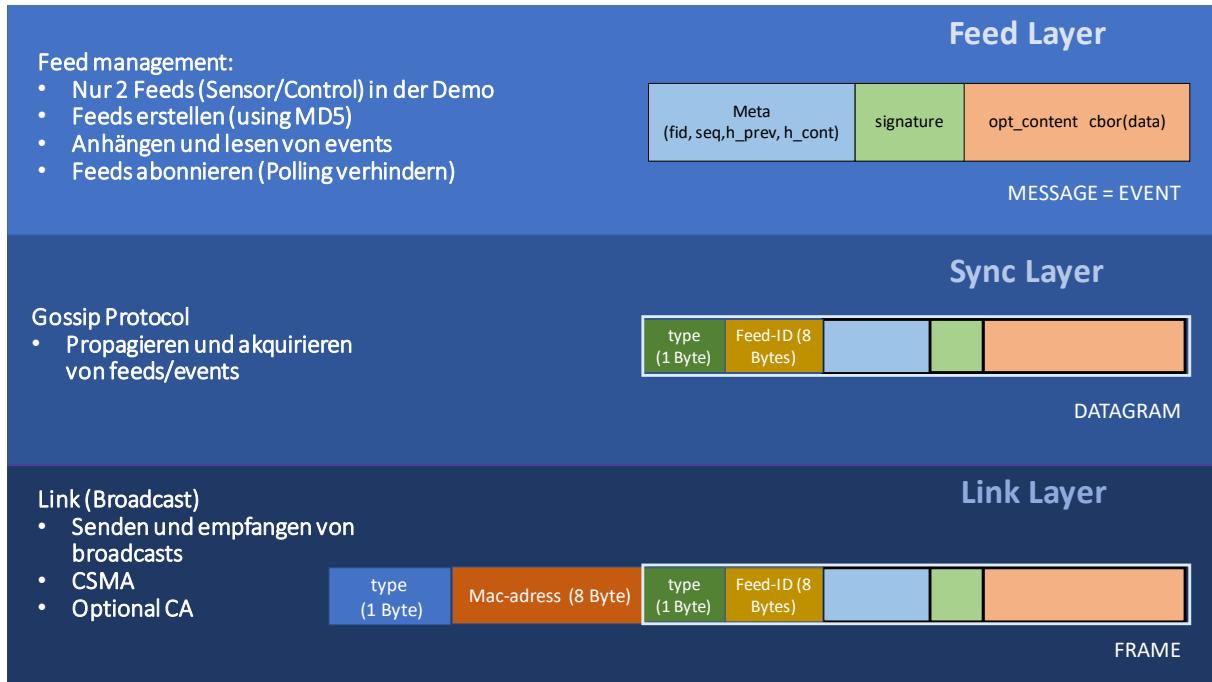


Abbildung 2 Layering des LoraLink Netzwerks

## 4.2 Feed Layer

Die Feed Layer bietet den Applikationen (SensUI und LoraSense) sowie dem Sync Layer eine einfache Schnittstelle zur BACnet Feed Library (*BACnet demo, 2020*). Sie bietet folgende Dienste an (die Funktionen sind in Anhang in Tabelle 1 aufgelistet):

- Erstellen von Feeds:
  - Sensor Feed: mit gemessenen Daten von den Sensoren
  - Control Feed: Steuerdaten um vom PC die Sensoren zu steuern (z.B. die Messintervalle zu ändern)
- Erstellen von Events in einem bestimmten Feed mit Hilfe der Library die von Prof. Tschudin zur Verfügung gestellt wurde (*BACnet demo, 2020*). Ein Event enthält im ersten Teil (Meta) die Feed ID, die Sequenznummer, den Hashwert des vorhergehenden Events so wie den eignen Hashwert (*BACnet event structure, 2020*). Für die Verschlüsselung der Events wurde nicht die gebräuchliche und sichere SHA256 Verschlüsselung, sondern MD5 gebraucht. MD5 gilt heute nicht mehr als sicher, da jedoch via LoPy4 nur kleine Pakete verschickt werden können (ca. 250 Bytes) und der Prozessor gewisse Grenzen setzt, mussten wir auf diese kompaktere Verschlüsselung zurückgreifen.

Meta (fid, seq, h_prev, h_cont)	signature	opt_content cbor(data)
------------------------------------	-----------	------------------------

- Anhängen von empfangenen Events an einen Feed und Prüfung der Gültigkeit. Es wird die Feed ID und die Sequenznummer überprüft.
- Herauslesen einzelner bzw. aller Events aus einem Feed.
- Abonnieren (subscribe) von Feeds: Benachrichtigung an die Applikationsschicht falls ein Event an den entsprechenden Feed angehängt wurde.
- Weitere Funktionen wie das Finden von Feed Ids und löschen von Feeds.

## 4.3 Sync Layer

### 4.3.1 Grundidee

In der Sync Layer werden die Feeds verschiedener Knoten synchronisiert. Dazu verwenden wir ein sogenanntes Gossip-Protokoll. Die Grundidee des Gossip-Protokolls ist, dass Knoten via Broadcast Information über die verfügbaren Feeds versenden. Eine solche Gossip-Nachricht enthält die Feed-ID (z.B. Sensor Feed ID) und die Anzahl verfügbarer Events in einem Feed (höchste Sequenznummer) welche der Knoten zur Zeit enthält. Es werden alle Feeds welche der Knoten besitzt, nach einander via Gossip-Nachricht versendet.

Sobald ein benachbarter Knoten eine Gossip-Nachricht erhält, kontrolliert er, ob er die darauffolgende Sequenznummer des Feeds mit der entsprechenden Feed ID besitzt. Falls ja, so sendet er den entsprechenden

Event. Aus Gründen der Kapazität haben wir uns entschlossen jeweils nur einen Event pro Nachricht zu versenden.

### 4.3.2 Multiple Access Protocol

Da es sich um ein dezentrales Netz mit gleichberechtigten Knoten handelt und Knoten nicht gleichzeitig «hören» und «sprechen» können, wird ein Multiple Access Protocol benötigt, damit Kollisionen einerseits vermieden werden bzw. sich das System von Kollisionen erholen kann.

Um dieses Problem zu lösen wird in LoraLink eine Mischung aus ALOHA und CSMA verwendet. ALOHA wird auf der Sync Layer Ebene umgesetzt. So werden Gossip Nachrichten nicht in immer gleichen Zeitintervallen versendet, sondern das Intervall wird jeweils zufällig verändert. Auch vor dem «Antworten» auf ein Gossip mit dem Senden einer Nachricht mit einem Event, wird wiederum eine Zufallszeit gewartet, damit es, falls zwei Knoten auf das gleiche Gossip antworten möchten, möglichst nicht zu Kollisionen kommt.

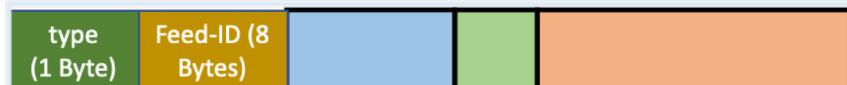
CSMA wurde auf der Link Layer umgesetzt, insofern folgt die Beschreibung auch im Link Layer.

### 4.3.3 Übertragungsrate

Da sehr kurze Wartezeiten zwischen dem Senden bzw. Empfangen von Nachrichten schnell zu einer Überlastung der Geräte geführt hat und da das aktuelle Einsatzgebiet im BACnet nicht allzu hohe Frequenzen benötigt, sind Wartezeiten zwischen 5 und 20 Sekunden verwendet worden.

### 4.3.4 Datenformat

In der Sync Layer wird der Event von der oberen Schicht (feed layer) in ein Datagramm verwandelt. Ein Datagramm enthält an erster Stelle den Typ (gossip = 0, event =1, Länge = 1 Byte), dann die Feed ID (9 Byte) und dann den als CBOR kodierten Event. Die Gossip Nachricht ist ähnlich aufgebaut (1.Byte: 0, 2.-10.Byte Feed ID, 11.-12. Byte beinhaltet die Sequenznummer).



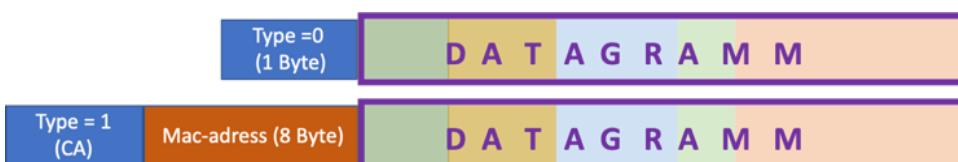
### 4.3.5 Schnittstellen mit dem Feed- und Link Layer

Zusätzlich zum oben beschriebenen Gossip-Dienst, bietet die Sync Layer folgende Dienste an:

- Erhält Events (Message) von der Feed Layer und verpackt sie in Datagramme.
- Leitet Datagramme an die darunterliegende Link Layer zum Versandt weiter.
- Hereinkommende Frames aus der Link Layer werden dekodiert und entsprechende Massnahmen ausgelöst (z.B. Versenden eines Events oder Weiterreichen eines einkommenden Events an die Feed Layer).

## 4.4 Link Layer

Die Link Layer ist verantwortlich die einzelnen Pakete via broadcast an die umliegenden LoRa Knoten zu versenden. Die Datagramme die von der darüberliegenden Sync Layer kommen, werden in ein Frame verpackt. An erster Stelle wird der Frame type angegeben (0 = Datagramm, 1 = Collision Avoidance (CA)), dies nimmt 1 Byte Platz in Anspruch. Anschliessend folgt im Falle eines Datagramms der Inhalt, welcher aus dem Sync Layer weitergegeben wurde und im Falle einer CA-Nachricht der Typ CTS oder RTS und die MAC-Adresse. (Die Verpackung in ein Frame ist nur dann nötig, wenn CA (siehe 1.6.2) gewählt wird, weil dann die MAC-Adresse für das Verschicken von rts und cts Pakete wichtig wird.)



### 4.4.1 CSMA (Carrier sense multiple access)

Einen wichtigen Beitrag zum Vermeiden unnötiger Kollisionen leistet das CSMA, dabei prüft ein Knoten vor dem Senden ob der Kanal frei ist. Falls nicht, wird in kurzen Zeitintervallen wiederholt geprüft, bis der Kanal frei ist und erst dann wird die eigentliche Nachricht versendet. In Abbildung 3 ist das Link Layer Protokoll mit CSMA als finite state machine (FSM) dargestellt.

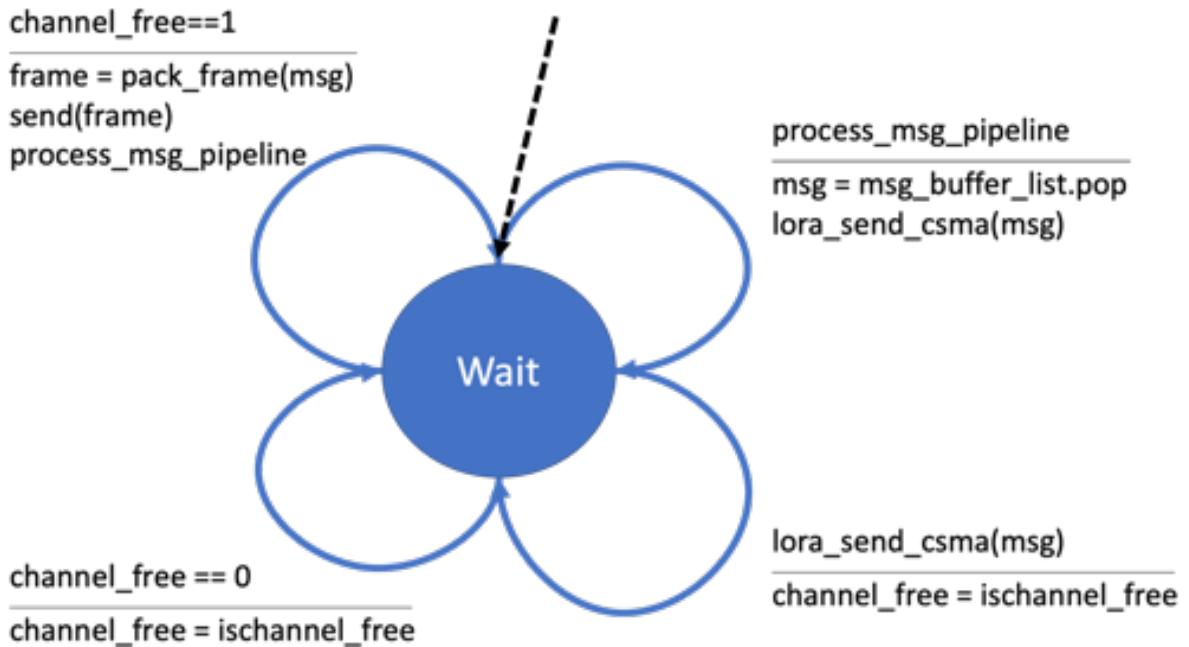


Abbildung 3 Finite state machine des link layer Protokolls

#### 4.4.2 CA (Collision Avoidance), optional

Optional kann beim Versenden von Datagrammen CA ausgewählt werden. In Abbildung 4 wird die Funktionsweise schematisch dargestellt: der rote Knoten sendet ein «request-to-send (RTS)». Sobald ein umliegender Knoten (grün) das rts erhält, sendet er ein «clear-to-send (CTS)» und sendet keine weiteren Daten (weder Gossip, noch Events, grüne Fläche). Sobald ein weiter entfernter Knoten (blau) das CTS vom grünen Knoten erhält, sendet dieser auch keine weiteren Daten, bis er der Sperr-Timer abgelaufen ist. Sobald der rote Knoten ein CTS erhält, sendet er seine Nachricht.

Collision Avoidance ist in einer Broadcast-Umgebung nur von zweifelhaftem Nutzen. Erhalten nämlich zwei Nodes das gleiche RTS einer dritten Node, so ist nicht klar welche Node mit einem CTS darauf antworten soll. Ein Ansatz wäre, dass beim vor dem Antworten mittels eines CTS während einer kleinen zufälligen Zeitspanne gewartet wird. Ausserdem wird die MAC Adresse mitgeschickt um zu identifizieren, von wem das RTS stammt bzw. für wen das CTS gedacht ist. Diesen Ansatz haben wir aber nicht weiter verfolgt, da wir eine entsprechende Lösung im Gossip-Protokoll auf dem Sync Layer umgesetzt haben. Und da die Sendefrequenz im Sync Layer sehr niedrig ist, ist CA wohl gar nicht wirklich nötig.

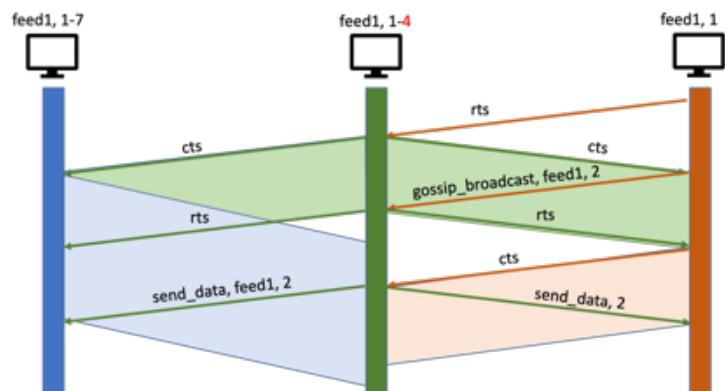


Abbildung 4 Collision Avoidance

#### 4.5 Knoten auf dem PC

Da der Sensor Feed auch auf einem Computer repliziert werden muss und der Control Feed für eine Steuerung der Sensoren auf einem UI von einem Computer aus gesteuert werden muss, mussten wir zusätzlich einen Knoten auf einem PC installieren. Aufgrund des oben beschriebenen Schichtenmodells konnten wir einfach auf der Link Ebene LoRa durch Wifi ersetzen.

Der ganze Code war grundsätzlich problemlos von Micropython auf Python portierbar, einzig bei den Threads waren kleine Änderungen notwendig.

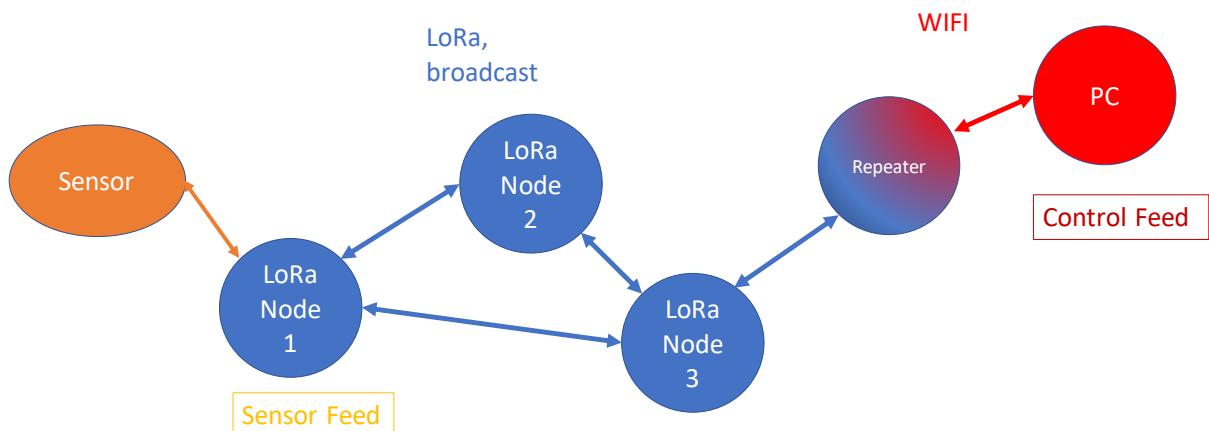


Abbildung 5 Netzwerk mit Repeater

## 4.6 Repeater

Erste Versuche gleichzeitig auf einem LoPy4 parallel eine Link Layer für LoRa und WiFi laufen zu lassen sind leider gescheitert. Core Dumps und Probleme mit der maximum recursion depth beim Arbeiten mit Threads haben eine lauffähige Version verhindert.

Deshalb haben wir auf einem LoPy einen einfachen Repeater implementiert. Dessen Aufgabe ist es hereinkommende Pakete aus dem LoRa Netzwerk zu Empfangen und via WiFi an den PC weiterzuleiten. Sowie hereinkommende Pakete via WiFi vom PC via LoRa an andere LoRa Knoten zu senden. Dieser LoPy agiert somit nicht als Knoten im LoRa-Netz, sondern übernimmt nur die Übersetzung von LoRa zu Wifi und umgekehrt vor.

## 5 Fazit

### 5.1 Probleme und Lösungsansätze

Als Problem stellte sich die Leistungsfähigkeit des LoPy's heraus. Bei eher aggressivem Senden und Empfangen mit kleinen Zeitintervallen bzw. beim Arbeiten mit vielen Threads waren Core Dumps nicht unüblich. Wir haben deshalb, wie im Sync Layer beschrieben, mit Zeitintervallen der Grösse 5 bis 20 Sekunden gearbeitet. Was in diesem Fall für das Sammeln von meteorologischen Messdaten auch ausreichend sein sollte.

Als nächstes Problem stellte sich bei der Integration aller drei Teilprojekte das Threading heraus. Da innerhalb von Threads die «max recursion depth» von 50 auf 10 heruntergesetzt wurde. Das Erstellen von Events und Auslesen von Inhalten aus Events benötigt aber mehr als 10 geschachtelte Funktionsaufrufe. Dieses Problem erkannten wir erst bei der Integration der drei Teilprojekte, erst dann haben wir diese Funktionalität in Threads verschoben. Für die Demo haben wir uns für den pragmatischen Ansatz einer «shared» while-loop entschieden, aus welcher heraus alle Layers aufgerufen werden.

### 5.2 Erreichtes und Ausblick

Grundsätzlich haben wir das Ziel erreicht, ein funktionierendes Feed-basiertes dezentrales Netz mit Hilfe der LoRa Technologie zu erstellen.

Das Layering Konzept funktioniert und bietet eine gewisse Flexibilität beim Einsatz verschiedener Technologien wie LoRa oder Wifi. So könnte auch das Feed Layer bzw. das Sync Layer einfach mit entsprechenden Funktionen ergänzt werden.

Hätten wir die Probleme mit dem Threading eher erkannt, so hätten wir aber den Ablauf des Programms etwas anders steuern können. Wir hätten z.B. ein Control Layer mit einer while loop erstellen können, welche für die Steuerung der anderen Layers zuständig wäre.

Wegen erschwerten Bedingungen durch die Pandemie konnten wir die geplante Testung des LoRa-Netzwerks mit mehreren lopy4 Knoten (nicht nur 2) nicht umsetzen. Dies wäre sicher ein wichtiger Test für die Stabilität des Netzes gewesen.

Nächste Schritte bei einer Weiterführung des Projekts, wäre das automatische Handling mehrerer Feeds (Vorarbeiten dazu haben wird gemacht). Parallel dazu könnte man die Funktion einbauen, Feeds selektiv zu folgen.

## 6 Quellen

BACnet <https://github.com/cn-uofbasel/BACnet> Heruntergeladen am 27. Mai 2020  
BACnet demo: <https://github.com/cn-uofbasel/BACnet/tree/master/src/demo> Heruntergeladen am 27. Mai 2020  
BACnet event strcture: <https://github.com/cn-uofbasel/BACnet/blob/master/doc/BACnet-event-structure.md> Heruntergeladen am 27. Mai 2020  
Heltec: [https://heltec-automation-docs.readthedocs.io/en/latest/esp32/quick\\_start.html](https://heltec-automation-docs.readthedocs.io/en/latest/esp32/quick_start.html), Heruntergeladen am 27. Mai 2020  
LoraLink code:<https://github.com/cn-uofbasel/BACnet/tree/master/groups/05-loraLink> Heruntergeladen am 27. Mai 2020  
Micropython: <https://docs.micropython.org/en/latest/library/uselect.html?highlight=wait> Heruntergeladen am 27. Mai 2020  
Pycom <https://pycom.io/product/lopy4/> Heruntergeladen am 27. Mai 2020  
Scuttlebutt 2020 : [https://www.scuttlebutt.nz\\_](https://www.scuttlebutt.nz_), Heruntergeladen am 27. Mai 2020  
Tarr D., Meyer A., Lavoie E., Tschudin Ch. (2019). Secure Scuttlebutt: An Identity-Centric Protocol for Subjective and Decentralized Applications (2019) ACM ISBN 978-1-4503-6970-1/19/09.  
<https://doi.org/10.1145/3357150.3357396>

## 7 Anhang

### 7.1 Funktionsübersicht

Tabelle 1 Feed Layer

function	description	calling
<b>create_feed</b> (type, keyfile filename, pcapfilename)	creates new feed, returns Feed ID	create_keyfile/ load_keyfile crypto.hmac, feed.FEED
<b>get_sensor_feed_fid()</b> <b>get_control_feed_fid()</b>	Get Feed ID of sensor/control Feed	
<b>get_fid_list()</b>	Get list of all Feeds	Optional: get list from all pcap files
<b>get_wired_event()</b>		event.to_wire
<b>create_event(FID, payload )</b>	creates cbor event,appends to pcap	feed.write
<b>get_event_content(FID, SEQ)</b>	gets content of event (string)	event.get_hash
<b>feed = get_feed_content(FID)</b>	gets feed in cbor format	
<b>Subscribe_sensor/control_feed</b>	Informs if new events arrived	callback
<b>append(fid,seq,e_wired)</b>	Appends event to pcap(check fid, seq)	feed._append

Tabelle 2 Sync Layer

function	description	calling
<b>send_gossip()</b>	broadcasts last event number of all feeds to surrounding loras	feed_layer.get_fid_list feed_layer.get_feed_length lora_feed_layer.get_name

	(1.byte =0, 2th-9th byte= fid, largest sequencenumber)	link_layer.append_msg_to_pipeline
receive_msg_cb	Handling incomming message	decode_msg
decode_msg	Decodes incomming message	handle_incoming_gossip
handle_incoming_gossip	If it's a gossip (first byte = 0), check whether requested events are on node, if yes, send them	send_event
handle_incoming_event	If it's an event (first byte= 1), event is appended to feed pcap	append
send_event	Message given to link layer (1.byte =1, 2th-9th byte= fid, payload)	link_layer.append_msg_to_pipeline

Tabelle 3 Link Layer

function	description	calling
lora_cb(lora)	Receiving message via lora link	LoRa.RX_PACKET_EVENT, lora.recv, lora_link.reveive_msg_cb
append_msg_to_pipeline	Puts message on buffer list	decode_msg
process_msg_pipeline	Loops throug buffer lists, sends messages via lora link (optional with CA and CSMA)	lora_send_csma_ca
lora_send_csma_ca/ lora_send_csma	If CSMA: send rts, wait for cts If CA: wait if channel is free	lora_send_csma, handle_incoming_cts handle_incoming_rts lora.ischannel_free
pack_event / unpack_event	pack/unpack frame: Type & MAC & event	

## 7.2 Kommunikation zwischen Heltec und Pycom Geräten

Damit die Heltec- und Pycomgeräte miteinander kommunizieren konnten:

```
Heltec.begin(true /*DisplayEnable Enable*/, true /*LoRa Enable*/, true /*Serial Enable*/, true /*LoRa
use PABOOST*/, BAND /*LoRa RF working band*/);
LoRa.setTxPower(14,PA_OUTPUT_RFO_PIN);
LoRa.setFrequency(BAND);
LoRa.setSignalBandwidth(125E3);
LoRa.setSpreadingFactor(7); //12
```

Erhalten der Nachrichten:

```
void onReceive(int packetSize)//LoRa receiver interrupt service
{
    packet = "";
    packSize = String(packetSize,DEC);

    while (LoRa.available())
    {
        packet += (char) LoRa.read();
    }

    Serial.println(packet);
    rssi = "RSSI: " + String(LoRa.packetRssi(), DEC);
    receiveflag = true;
}
```

Display:

```
Heltec.display -> drawString(0, 50, "Packet " + (String)(counter-1) + " sent done");
Heltec.display -> drawString(0, 0, "Received Size" + packSize + " packages:");
Heltec.display -> drawString(0, 10, packet);
Heltec.display -> drawString(0, 20, "With " + rssi);
Heltec.display -> display();
```

# Long Range Wifi - BACnet «Backbone»

## Einführung

In der Vorlesung Introduction to Internet and Security ist das diesjährige Projekt das Erstellen des BACnet, indem alle Studierende ein Unterprojekt wählen. BACnet ist ein regionales Netz für elektrische Kommunikation, das bei Bedarf ohne Internet auskommt. Dies bedeutetet, dass BACnet ganz ohne IP-Adresse funktioniert. Unsere Gruppe, 06-longFi, beteiligte sich als Unterprojekt, indem wir Teil der Replication sind. Wir stellen ein Netz mittels Ubiquiti Antennen her, welches eine Distanz von mindestens 10 km abdeckt. Einführend haben wir uns es zur Aufgabe gemacht die nötige Hardware mit der technischen Funktion Etherframes über 10km zu versenden. Die Etherframes werden mittels unserer API als Pakete versendet.

## Methodik

Damit das Netzwerk sichergestellt wird, benötigt man zuverlässige Hardware, sowie eine Software. Die Methodik erklärt die wesentlichen Schritte der Hardwarebeschaffung und der Softwareimplementierung. Wir haben uns für zwei Ubiquiti Powerbeams entschieden.

## Softwareimplementierung

Ziel von uns war es Ethernet Frames zu verschicken und diese dann mit Wireshark zu analysieren.

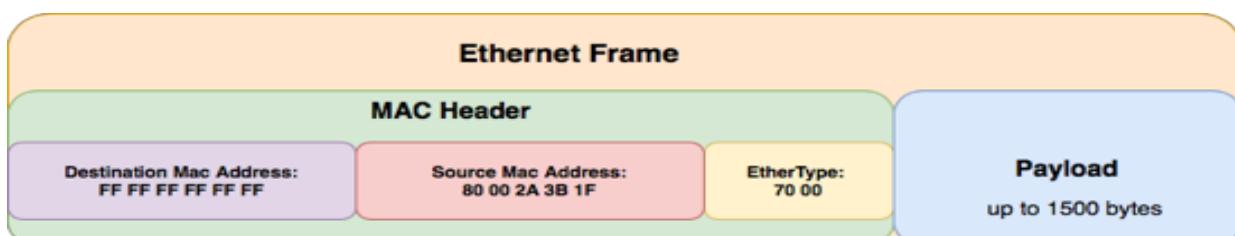


Abb. 1 Aufbau eines Etherframes

Wie man aus der Abbildung 1 entnehmen kann, verwenden wir jeweils MAC Adressen, um die Ethernet Frames mit Hilfe der Powerbeams zu versenden. Den EtherType haben wir bewusst eigen gewählt, damit es nicht zu Verwechslungen mit anderen eingehenden Signalen kommt. Da wir uns im Layer Level 2 des OSI-Referenzmodells, also im Link-Layer, befinden, konnten wir bis zu 1'500 Bytes an Daten, inklusive Header, versenden. Die Destination Mac Address repräsentiert hierbei den Broadcast, wobei die Source Mac Address in dieser Abbildung willkürlich gewählt wurde.

## Implementierung

Die Implementierung unseres Etherframes erfolgte anhand von zwei Python-Skripten, nämlich „sendEther.py“ und „receiveEther.py“. Wir machten uns überwiegend Scapy zu Nutze. Scapy ist ein Python Programm, dass ein mächtiges Werkzeug für die Netzwerkpacket-Manipulation zur Verfügung stellt. Diese Schnittstelle verwendeten wir für das Versenden unserer Etherframes. Im „sendEther.py“ erstellten wir unser Ethernet Frame, wobei wir diesen über das Ethernet Interface des spezifischen Gerätes versendeten. Dieses Ethernet Frame wurde alle 10 Sekunden erneut gesendet, damit wir im Wireshark spezifisch nach diesem EtherType filtern konnten. Das „receiveEther.py“-Skript ist dazu da, die gesendeten Etherframes zu empfangen und auf der Konsole auszugeben. Da alle Frames als Bytes gesendet werden, brauchten wir noch Funktionen, welche die empfangenen und gesendeten Bytes lesbar in der Konsole ausgeben.

Das Etherframes wird mit den Objekten Raw und Ether erstellt. Die Zuweisung erfolgt mit „`ether = Ether(type = 0x7000) / Raw(load = 'data')`“. Mit der Funktion „`sendp()`“ wird dieses Ethernet Frame als broadcast gesendet. Um die empfangenen Frames zu filtern benötigen wir noch „`conf.L2socket`“ aus der Scapy API. Diese zählen als Sockets, die spezifisch mit dem gegebenen Interface im Linklayer verbinden können. Die empfangenen Bytes können dann nach dem Interface und dem Ethertype gefiltert werden, und somit bekamen wir die Bestätigung, dass die gesendeten Frames auch empfangen wurden. An den Abbildungen im Abschnitt Resultate, kann ein Auszug des Wiresharks erkennen, welcher den Inhalt der Etherframes repräsentiert.

## Hardwarebeschaffung

Um eine Verbindung über eine grosse Distanz aufzubauen, benötigten wir, zusätzlich zu den Endgeräten, Hardware. Bei den ersten Recherchen im Internet fiel uns auf, dass Ubiquiti Powerbeams zur Verfügung stellt. Die Auswahl an den Ubiquiti Produkten ist sehr gross, sodass wir gewisse Anforderungen stellten: Preisgünstig, handlich, Mindestdistanz von 10km, Funktion als Access Point und Station, sowie schnelle Lieferungszeiten. Wir konnten so drei Produkte filtern, nämlich PB-M2-400, PB-5AC-620 und PB-5AC-Gen2. Der Entscheid fiel auf PB-M2-400, welches mit 90 Euro am preisgünstigsten war. Der Nachteil einer Bandbreite von 2.4 GHz, kam letztlich zu unseren Gunsten, da die zu sendenden Dateien sehr klein sind und diese in diesem Frequenzbereich heutzutage weniger anfällig auf andere Signale ist. Der Grund hierfür liegt auf der gängigeren Nutzung der 5GHz Bandbreite. So änderte sich der wesentliche Nachteil zur Stärke. Die Installation der Hardware erfolgte über die bereitgestellte Software Air-OS von Ubiquiti. Die PowerBeams werden separat als Access Point und Station initialisiert, um eine direkte Verbindung zueinander aufzubauen zu können. Die IP-Adressen der Geräte werden über DHCP vom Router gewählt. Die Netzwerkkonfiguration wird durch die Nutzung des DHCP vom «Server» an die „Clients“ zugewiesen. In unserem Fall entspricht der Router dem Server, welcher den Powerbeams, also Clients, die IP-Adressen vergibt. Die Verbindung verläuft bidirektional und erstellt dann über die Distanz Subnetze. Die folgende Abbildung 2 erläutert unsere Architektur. Außerdem erkennt man, dass mittels einem Router an der Station ein weiteres Subnetz erzeugt wird, indem mehrere Teile des Gesamt-„Netzes“ sind.

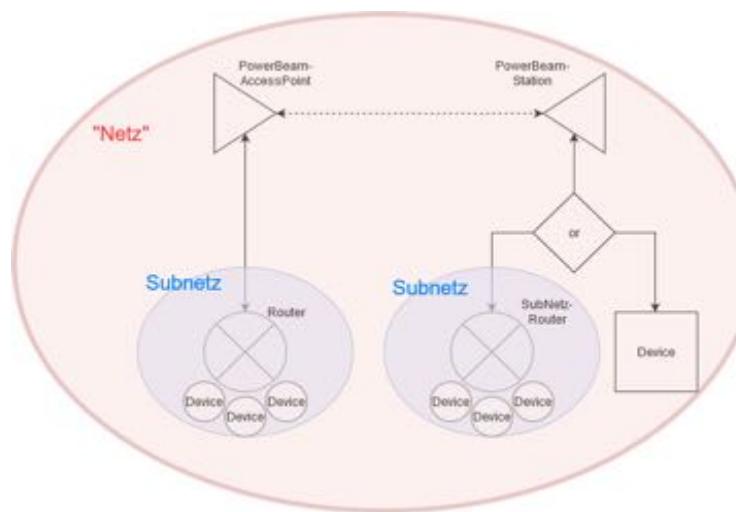
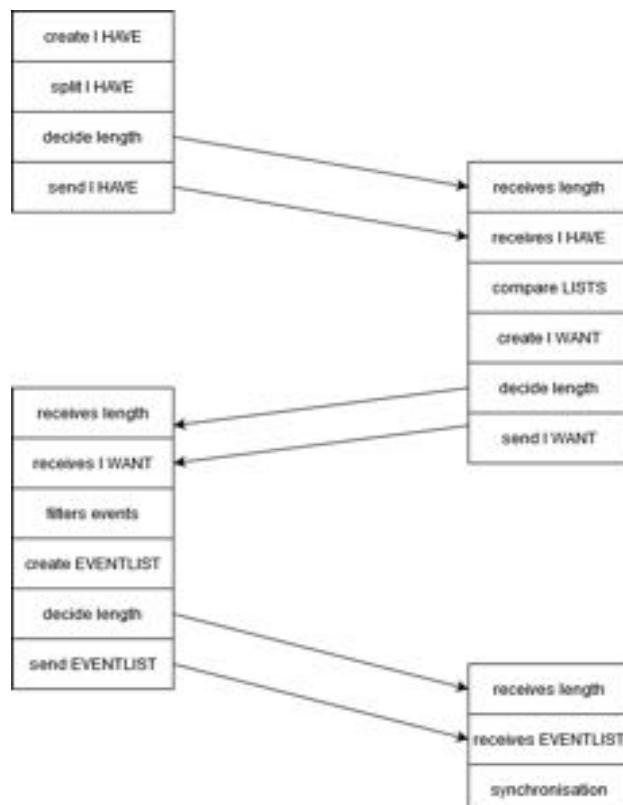


Abb. 2 Schema Netzwerk

Router gewählt. Die Netzwerkkonfiguration wird durch die Nutzung des DHCP vom «Server» an die „Clients“ zugewiesen. In unserem Fall entspricht der Router dem Server, welcher den Powerbeams, also Clients, die IP-Adressen vergibt. Die Verbindung verläuft bidirektional und erstellt dann über die Distanz Subnetze. Die folgende Abbildung 2 erläutert unsere Architektur. Außerdem erkennt man, dass mittels einem Router an der Station ein weiteres Subnetz erzeugt wird, indem mehrere Teile des Gesamt-„Netzes“ sind.

## Integration

Die Integration geschah unter den verschiedenen Gruppen, wobei unser Schwerpunkt die Integration mit der Gruppe „12-logSync“ war. Die Gruppe hatte es zum Ziel eine Datenbanksynchronisation bereit zu stellen, welche über das Prinzip von I want- und I have-Listen basiert. Unsere Aufgabe dabei war es Funktionen zur Verfügung zu stellen, welche die Daten via Broadcast im Netz verschicken und empfangen. Wie bereits bei der Implementierung angesprochen wird dies über den Ethertype gefiltert. Die Bestimmung der Daten, welche aktualisiert werden müssen, geschieht über das Prinzip des Listenaustausches.



Zu Beginn versendeten wir die uns bereitgestellten PCAP-Dateien. Diese können grösser als 1'500 Bytes und mussten somit geteilt werden. Die geteilten Pakete werden einzeln in Reihenfolge versendet, welche die Empfänger zur Originaldatei zusammenfügt. In der folgenden Tabelle sieht man die von uns bereitgestellten Funktionen:

Abb. 3 Prinzip des Listenaustausches

### receiveEither.py

```

def ethernet_frame(data)
This function returns the converted information of the packets.

def get_mac_addr(bytes_addr):
This function returns the converted MAC-addresses.

def get_protocol(bytes_proto):
This function returns the converted protocol.

def append(list):
This function appends the splits (1500 bytes) to the origin data

def receive_len(interface):
This function receives the amount of splits needed for a package.

def receive_list(length, interface):
This function receives the list of the data splits and appends them to the original data
  
```

### sendEither.py

```

def send_len(packet, interface): This function sends the length of the given packet

def send_packet(packet, interface): This function sends the given packet through the interface.
  
```

## Resultate

In den folgenden Abschnitten widmen wir uns die erzielten Resultate. Diese beinhalten Signale der Verbindungen auf verschiedenen Distanzen, sowie Abbildungen der Kosolenausgaben und der gesendeten und empfangenen Etherframes.

### Signale

In der nachfolgenden Tabelle unterteilen wir jeweils die Qualität der Signalübertragung, der Distanz und die Luftlinie der einzelnen abgedeckten Bereiche. Die folgenden Daten entnahmen wir aus der AirOS des Ubiquiti. Die Luftlinien werden mittels „[luftlinie.org](#)“ nachgestellt. Aufgrund der Grenzschiessung der EU und Schweiz sind die Ortschaften jeweils in Deutschland. Bei unseren Versuchen ist die Qualität und Kapazität von dem Accesspoints und der Stationen immer gleich.

airMaxQuality	airMaxCapacity	Transmit-CCQ AP / Station		TX/RX-Rate (Mbps/Mbps)	Distanz (+- 0.5 km)
96%	91%	99.1%	99.1%	144.444/144.444	9 -10 m
87%	69%	98.2%	98%	104.44/104	2 – 3 km
69%	47%	80.3%	81.1%	78/78	15 km

Die Transmit-CCQ stellt die Übertragung der Daten dar, welche direkt beim Senden des Signals übertragen wurde. Der der Transmit-CCQ-Verlust wird wieder übertragen, sodass es zu keinem erheblichen Verlust kommt. Die TX/RX-Rate stellt das Verhältnis der Transmission und des Receive dar. Hierbei bedeutet dies, dass es eher einen Upload/Download-Rate darstellt. In der Tabelle ist die TX/RX-Rate von den Accesspoints dargestellt. Bei den Station Points wäre dies umgekehrt. Die airMaxCapacity und airMaxQuality stellen jeweils die theoretisch maximalen Werte der möglichen Performance dar. Wie man hierbei erkennen kann nehmen die Werte mit weiterer Distanz ab. Wobei hierbei verschiedene Hindernisse, wie Bäume, Häuser, Hügel usw. auftreten, die die Werte beeinflussen. Die folgende Abbildung repräsentiert die Distanz von 14.7km als Fluglinie. Der erreichte Punkt war eine Wirtschaft, die von blossem Auge aus dem Haus von Marco zu sehen war. Dort konnten wir eine Erlaubnis für das Testen der Antennen holen. So konnten wir unsere Implementierung testen.

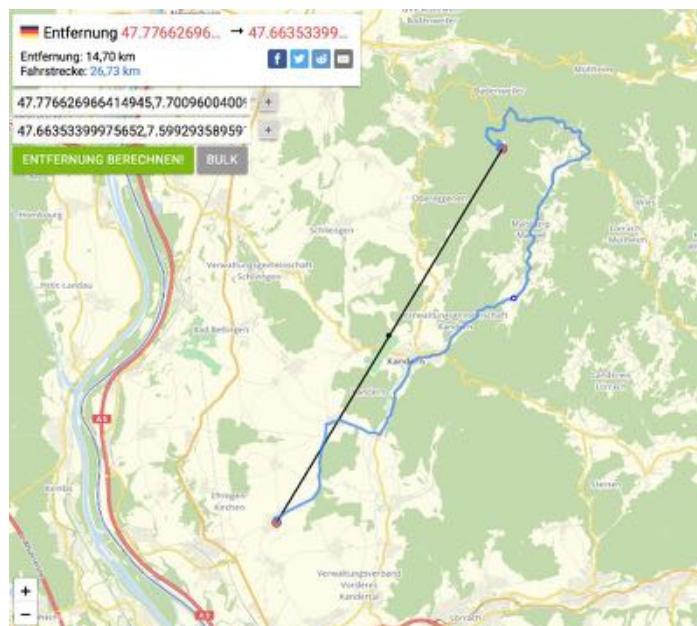


Abb. 4 Luftlinie von maximal erreichter Distanz

## Konsolenausgaben und Integration

Damit man sehen kann, wie die Software die Resultate liefert, sieht man in den folgenden Bildern jeweils die Ausgabe der Konsolen. In der Abbildung 4.a ist die Konsolenausgabe von dem Befehl «sudo main.py hasPackets -en0» zu sehen. Man sieht den Ablauf der versendeten Pakete, die Länge, über welches Protokoll und was im Paket steht. Den Antagonisten dazu sieht man in Abbildung 5.b, wobei man die Funktionsweise des «I-HAVE», «I-WANT» Prinzip sieht. Ausserdem sieht man die Länge des Paketes, wobei Len 1.0 für 1'500 Bytes sieht. An Abbildungen 5.c und 5.d erkennt man die Aktualisierung des Verzeichnisses am Endgerät. Somit konnten wir erfolgreich Datenbanken über Etherframes versenden.

```
LENGTH 0x3e8
.
Sent 1 packets.
.
Sent 1 packets.
Request accepted, list of files sent.
Waiting for request...

Ethernet Frame:
Destination MAC: FF:FF:FF:FF:FF:FF
Source MAC: 8C:85:90:46:2B:48
Protocol: 7000
Packet: b'0x3e8'
LEN 1.0

Ethernet Frame:
Destination MAC: FF:FF:FF:FF:FF:FF
Source MAC: 8C:85:90:46:2B:48
Protocol: 7000
Packet: b'\x80'
LEN 1
The other device is up-to-date.
LENGTH 0x3e8
.
Sent 1 packets.
.
Sent 1 packets.
"I WANT"-list received...
Sending event list...
```

Abb. 5.a: needsPacket

```
Ethernet Frame:
Destination MAC: FF:FF:FF:FF:FF:FF
Source MAC: 8C:85:90:46:2B:48
Protocol: 7000
Packet: b'0x3e8'
LEN 1.0

Ethernet Frame:
Destination MAC: FF:FF:FF:FF:FF:FF
Source MAC: 8C:85:90:46:2B:48
Protocol: 7000
Packet: b'\x80'
LEN 1
"I HAVE"-list received...
LENGTH 0x3e8
.
Sent 1 packets.
.
Sent 1 packets.
Sending "I WANT"-list...

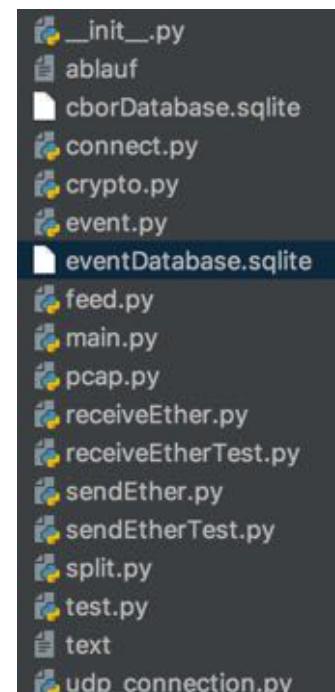
Ethernet Frame:
Destination MAC: FF:FF:FF:FF:FF:FF
Source MAC: 8C:85:90:46:2B:48
Protocol: 7000
Packet: b'0x3e8'
LEN 1.0

Ethernet Frame:
Destination MAC: FF:FF:FF:FF:FF:FF
Source MAC: 8C:85:90:46:2B:48
Protocol: 7000
Packet: b'\x80'
LEN 1
Event: list received...
```

5.b hasPackets



5.c Verzeichnis alt



5.d Verzeichnis aktualisiert

Wichtig dabei ist es, dass die Anzahl der zu sendenden Pakete bereits vor dem Sendevorgang bestimmt werden muss. Die Information wird anschließend an den Empfänger weitergegeben, damit diesem die Menge der empfangenden Pakete bewusst ist. Am Ende des Projekts war das Ziel Datenbanken der anderen Gruppe zu senden und zu empfangen. Da es jedoch zu Problemen mit der Kompatibilität verschiedenen Betriebssystemen kam (siehe Abschnitt Schwierigkeiten), mussten wir ein neues Programm schreiben, bei dem wir unsere Implementierung mit den Datenbanken testen konnten. Um den Test durchzuführen erstellten wir auf einem Linux-Gerät eine Datenbank durch die Chat-Applikation von Gruppe 12. Die Chat-Applikation ist in der nun folgenden Abbildung ersichtlich. Nebenbei erkennt man Wireshark mit der Nachricht, die über unsere Etherframes geschickt werden. Bei der Abbildung 6 erkennt man, wie wir vorgegangen sind, um nach unseren Signalen zu filtern. Aufgrund des eigenen Ethertypes, konnten wir schnell danach filtern. In dieser kurzen Aufnahme kann man die versendeten Frames erkennen. Rot hervorgehoben sieht man die Nachrichten, die der Chatapplikation entstammen.

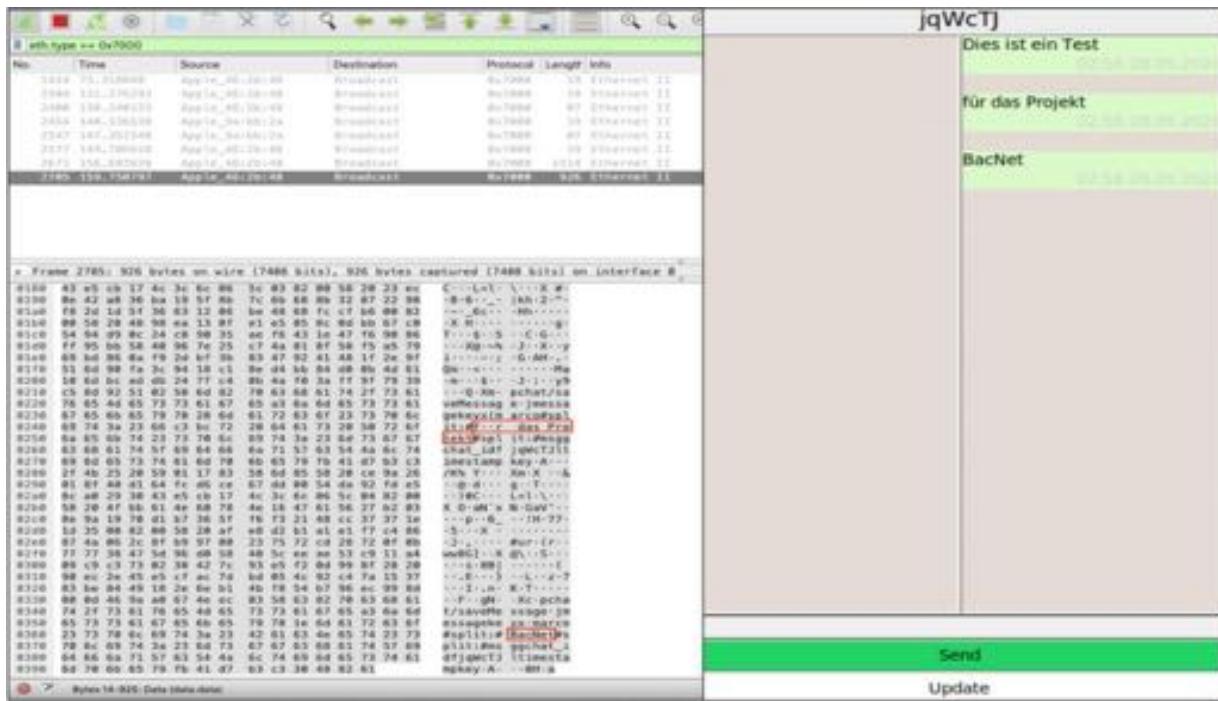


Abb. 6 Wiresharkauszug und Chatapplikation

## Schwierigkeiten

In diesem Semester war die ganze Welt vom Coronavirus betroffen. Dies sorgte auch bei uns für Schwierigkeiten. Insbesondere in der Hardwarebeschaffung und der Installation hat sich das Projekt schwieriger gestaltet. Nachdem der Bundesrat die Grenzen zu der EU und die EU die Grenzen zur Schweiz geschlossen hatte, waren wir dabei die einzelnen Antennen zu bestellen. Wir bestellten die Antennen in Deutschland, da sie dort etwa zum halben Preis angeboten wurden und Marco dort wohnt. Dabei kam es zu Lieferverzögerungen. Die Installation konnten wir nur über Meetings vollziehen. Die Verzollung in der Schweiz war leider finanziell nicht tragbar. Somit war Marco mehr oder weniger allein mit dem physischen Aufbau beschäftigt. Nichtsdestotrotz konnten wir mit Hilfe von Prof. Dr. Tschudin die Satelliten einrichten. Die Meetings unserer Gruppe hielten wir per Zoom und Whatsapp ab. Bei der Software gab es überwiegend Schwierigkeiten mit den Betriebssystemen. Wir waren alle Benutzer von Windows und Mac. Schliesslich wechselten wir auf Ubuntu, indem Nderim noch ein Gerät zur Verfügung stellte, sodass wir zwei Endgeräte mit Ubuntu hatten. Unseren Code haben wir so angepasst, dass er die Raw Ethernet Frames versenden und empfangen kann. Dies funktionierte auch auf Mac. Bei der Integration gab es dort Unstimmigkeiten mit anderen Gruppen, die in der kurzen Zeit nicht zu bewerkstelligen war. Der Code der anderen Gruppe war betriebsspezifisch auf Windows konzipiert. Trotzdem war es uns möglich in unserem Ethernet Frame die Pcap-Dateien und Sqlite-Dateien über 15km Distanz zu versenden. Bei der Antenneninstallation war es insofern eine Schwierigkeit, eine passende störfreie Distanz zu finden. Die Antennen waren in sensibel auf Hindernisse. Die Hindernisse in diesem Fall stellten Wälder, Hügel und andere Häuser. Dies führte unter anderem zum Qualitätsverlust der Signalstärke. Zusätzlich mussten die Antennen zueinander gerichtet sein, was eine ruhige Hand benötigt, da kleinste Winkeländerungen zu viel schwächeren Signale führen. Glücklicherweise konnten wir eine Ortschaft in direkter Umgebung finden.

## Fazit

Insgesamt können wir unsererseits sagen, dass wir unseren Teil des Gesamtprojektes erfüllt haben. Unter verschiedenen Distanzen konnten wir unsere Raw Ethernet Frames, sowie auch Packed Ethernet Frames versenden. Wie man aus den Resultaten sehen kann, konnten wir verschiedene Distanzen überbrücken, wodurch die Qualität des Verbindungssignals sinkt, je weiter man die Antennen zueinander richtet. Dies hängt insbesondere von Hindernissen ab. Wobei die Qualität der Signalstärke sehr gut auf Distanzen von weniger als 5km war. Scapy ist eine wunderbare Python-API, die noch viel mehr in der Netzwerktechnik ermöglicht. Aufgrund dieser API konnten wir unseren Code zu Ende stellen und mussten nicht alles von Beginn an Programmieren. Als Verbesserung wäre klar das Budget von anderen Produkten. Bei den Ubiquiti gibt es verschiedene Produkte, die besser wären. Dort wären auch weitaus höhere Distanzen möglich. Ein wichtiger Punkt ist das Management des recht hardwarelastigen Projektes. Klar war es gesundheitsbedingt nicht möglich das Projekt vor Ort zu errichten. Dennoch fiel uns die Kommunikation über Zoom zu Beginn schwer zu vollziehen, und das unser Projekt von dort aus umzusetzen. Mit der Zeit gewohnten wir uns an die Umstände und deswegen konnten wir auch gut mit den anderen Gruppen kommunizieren. Die Integration zu einem grösseren Projekt war sehr kurzzeitig. Dort fehlte es uns an strikte Aufgaben- und Rollenaufteilungen. Anhand des Gesamtprojektes konnten wir sehr viel lernen. Unser Unterprojekt brachte uns vor allem bei, wie das Internet aufgebaut ist. Wir konnten ein Netzwerk über eine hohe Distanz sicherstellen. Wir hatten einen stetigen Austausch und einen konstanten Lernprozess. Wir haben uns sehr viel Wissen angeeignet, das wir durchaus privat und beruflich verwenden können. Somit können wir mit Freude berichten, dass wir stolz auf unseren Lernfortschritt während dem Projekt sind.

Frühjahressemester 2020

Vorlesung: Introduction to Internet and Security

Prof. Dr. Christian Tschudin

Universität Basel

19. Juli, 2020

# **IAS Bericht 07-logStore**

---

**Verfasser:**

Viktor Gsteiger

[v.gsteiger@unibas.ch](mailto:v.gsteiger@unibas.ch)

Matrikel - Nr.: 18-054-700

&

Moritz Würth

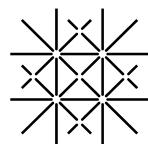
[moritz.wuerth@stud.unibas.ch](mailto:moritz.wuerth@stud.unibas.ch)

Matrikel - Nr.: 18-051-920

**Begutachter:**

Prof. Dr. Christian Tschudin

[christian.tschudin@unibas.ch](mailto:christian.tschudin@unibas.ch)



**Universität  
Basel**

Departement  
Mathematik und Informatik

# Inhaltsverzeichnis

1. Abstract .....	2
2. Einleitung.....	2
2.1 Problemstellung .....	2
2.2 Unsere Position im Projekt.....	3
2.3 Ziele .....	3
3. Produkt.....	4
3.1 Methodik .....	4
3.2 Prozess .....	4
3.3 Resultat.....	5
3.4 Integration .....	7
4. Fazit.....	8
4.1 Produkt.....	8
4.2 Prozess .....	8
4.3 Gesamtprojekt.....	8
5. Abbildungsverzeichnis.....	9
6. Referenzen .....	9
7. Weiterführende Referenzen .....	9

# 1. Abstract

Das Ziel der vorliegenden Projektarbeit war es, dem „Basel Citizen Net“-Projekt im Verlauf der Vorlesung „Introduction to Internet and Security“ eine Funktionalität eines effizienten Speichers zur Verfügung zu stellen. Dieser Speicher und vor allem dessen Schnittstellenfunktionen sollten gleichzeitig auch die Kommunikation zwischen den Applikationen und den Netzwerk-Anbindungen ermöglichen. Beim „Basel Citizen Net“ handelt es sich um ein dezentralisiertes Netzwerk, welches auf der Basis von einem gemeinsamen Protokoll den teilnehmenden Knoten erlaubt, ohne Internetzugang und über binär Dateien, Informationen auszutauschen. Die von der Gruppe logStore entwickelte Speicherfähigkeit sollte das effiziente Abrufen von Informationen von Applikationen mithilfe von Hilfsfunktionen bereitstellen. Gleichzeitig soll den Netzwerkschicht-Gruppen ermöglicht werden, mit anderen Knoten Daten auszutauschen, indem sie auf die Daten in der Datenbank zugreifen können. Dafür hatte die entwickelte Applikation ein effizientes Datenbankmanagement zu bewerkstelligen und auch die Integrität der Nachrichten zu wahren. Um dies effizient umzusetzen, wurde zusammen mit der Gruppe feedCtrl, ein in sich geschlossenes Produkt entwickelt, das durch klar definierte Schnittstellen einfach mit den anderen Projekten integriert werden konnte. Das Produkt erfüllte die am Projektbeginn definierten Anforderungen sehr gut und überzeugte durch eine einfache Anbindung, effiziente Abrufmechanismen und einem klaren Arbeitsablauf. Durch die sehr erfolgreiche Anwendung in drei Vorzeigeprodukten konnte die Applikation ihren Wert bestätigen. Der vorliegende Projektbericht richtet sich an andere Studierende, Assistenten, Interessierte und Dozierende der Fachrichtung Computer Science oder ähnlichem.

# 2. Einleitung

## 2.1 Problemstellung

Speicherkapazitäten sind ein integraler Teil jedes Netzwerkes und ermöglichen es den meisten Netzwerken überhaupt zu funktionieren. Ohne einen Buffer, einem Speicher, der Informationen zwischenspeichert, bevor diese vom Gerät verarbeitet werden kann, würde das Konzept des Routers wenig Sinn ergeben und es würde zu einem sehr hohen Paketverlust kommen. Ein Buffer ermöglicht es einem Router bei einer zu gross eintreffenden Datenmenge potentiell mehr Daten zu verarbeiten als vom Gerät möglich wäre, indem die ankommenden Pakete im Buffer zwischengespeichert werden und erst bei verfügbaren weiterverarbeitungs-Resourcen des Routers abgerufen werden [1].

Die Wichtigkeit eines Buffers trifft in einem Netz mit einer mehr oder weniger konstanten Verbindung zu, ist jedoch umso wichtiger in einem Netzwerk, das ohne ständige Verbindung funktioniert. In einem sogenannten Sneaker-Netz, einem Netz, bei dem erst durch eine unregelmässige physische Übergabe von Daten der Austausch stattfindet, ist eine Speicherfähigkeit ungemein wichtig, da es nicht die Erwartung sein kann, dass jegliche Applikationen selber den Überblick behalten sollen, wie der aktuelle Stand der Übertragung ist. Neben dieser Buffer-Fähigkeit wird es für die Applikationen auch wichtig, einen persistenten Speicher zur Verfügung zu haben, auf diesen sie bei Bedarf jederzeit zugreifen können. Während der Vorlesung „Introduction to Internet and Security“ entwickelten die Studierenden ein dezentrales Netzwerk, welches ohne aktive Verbindung zum herkömmlichen Internet auskommen sollte. Die Idee des daraus entstandenen „Basel Citizen Net“ basiert auf dem „Secure Scuttlebutt“, das dezentralisiertes Netzwerkprotokoll darstellt, welches stark auf das Konzept von Identitäten basierten Applikationen setzt [2]. Das „Basel Citizen Net“ Protokoll und Netzwerk, welches im Verlauf der verschiedenen Projekte entwickelt wurde, ermöglichte unterschiedliche Varianten der Übertragungstechnologien, die in unterschiedlichen Abständen Daten übertragen würden. Die Datenpakete werden hierbei in einer Binärform transferiert und gehören zu einem Feed, der durch das kontinuierliche referenzieren eines Paketes auf das vorherige zustande kommt. Im Falle des „Basel Citizen Net“ gibt es außerdem einen master-Feed, bei dem alle zu einem Nutzer gehörenden Feeds eingetragen werden.

Trotz der Unterschiedlichen Übertragungstechnologien und der verschiedenen Feeds sollten die Nutzer des Netzwerkes das Gefühl haben, über eine mehr oder weniger zuverlässige Übertragungstechnologie Daten zu versenden. Damit soll für die Applikationsschichten, welche den Nutzern die Daten anzeigen, eine Abstraktion zum Netzwerk zur Verfügung gestellt werden, welches gewisse Anforderungen erfüllt, ähnlich wie im „Department of Defense“ (DoD) Schichtenmodell des herkömmlichen Internets [3]. Den Applikationsgruppen muss ermöglicht werden, effizient die aktuell übertragenen Informationen anzufordern und auch neue Informationen zu speichern, die bei der nächsten Übertragungsrunde versendet werden würden. Dabei ist zudem wichtig, dass die Applikationsgruppen nicht ständig aktiv sein müssen, sondern dass der Datenaustausch ohne deren

aktive Beteiligung laufen soll. Abschliessend würde es für das Netzwerk wichtig werden, sich auf eine kontinuierliche und verlässliche Speicherung der Daten-Pakete stützen zu können, um die Netzwerkfunktionalität effizient zu bewerkstelligen.

## 2.2 Unsere Position im Projekt

Wie bereits erwähnt ist die Gruppe logStore aus dem Kontext des DoD-Schichtenmodell auf der Ebene Transportschicht zu platzieren. Damit war von Beginn an klar, dass eine enge Zusammenarbeit mit den Applikationsgruppen und den Netzwerkschicht-Gruppen unerlässlich werden würde. Durch eine klare Schnittstellendefinition zusammen mit den Interessensgruppen würde diese Kommunikation erleichtert werden und es zu dem ermöglichen, bereits früh Tests durchzuführen. Damit wurde von Anfang an eine Test-driven development Methode angestrebt, die eine kontinuierliche Integration mit den anderen Gruppen ermöglichen sollte. Des Weiteren wurde relativ früh in der Konzeptphase des Projektes klar, dass eine enge Zusammenarbeit mit der Gruppe feedCtrl [4] erforderlich war, um die Problemstellung effizient anzugehen.

## 2.3 Ziele

Das Projekt der Gruppe logStore soll einen persistenten Speicher auf jedem Knoten des Netzwerkes zur Verfügung stellen. Dieser Speicher soll den Applikationen auf den jeweiligen Knoten ermöglichen, Daten auf den Speicher zu schreiben und zu lesen. Gleichzeitig soll den Netzwerkschicht-Gruppen, im Projekt die Gruppen logMerge [5] und logSync [6], ermöglicht werden, die letzten Informationen mit der Hilfe des Speichers von den Applikationen zu lesen und neue Informationen aus einem Netzwerkkontakt in den Speicher zu schreiben. Damit soll das Produkt der Gruppe logStore den Applikationen ermöglichen, Daten auszutauschen und soll zusammen mit den Netzwerkschicht-Gruppen den Überblick behalten, welche Daten noch ausgetauscht werden müssen. Wie in der Vorlesung „Introduction to Internet and Security“ ersichtlich wurde, funktioniert damit die Gruppe logStore auf der Ebene der Transportschicht und soll gewisse Dienstleistungen des TCP Protokolls zur Verfügung stellen. Insbesondere soll das Produkt eine Übertragung in falscher Reihenfolge korrigieren und bei fehlerhafter Übertragung die Daten noch einmal anfordern, beziehungsweise klar definieren, wie weit die aktuelle Übertragung ist und was noch fehlt. Dies gleicht einer wiederholten Anforderung, da bei einer nächsten Übertragungsrunde noch einmal der gleiche Stand mittgeteilt wird und somit noch einmal der gleiche Inhalt angefordert wird. Dabei ist es wichtig, dass es für jegliche Anbindungsgruppen keine Möglichkeiten geben wird, Daten aus dem Speicher zu löschen, da dies die Integrität des Feeds, im Projektfall eine append-only Lösung bei der nur ein Element hinzugefügt, jedoch niemals gelöscht werden kann, zerstören würde. Zusätzlich muss es gewisse Kontrollmechanismen geben, die ungültige Informationen ausschliesst und so die Integrität der Datenbank und den Speicher der Knoten schützt. Dieser umfangreiche Anforderungskatalog wird im Verlauf dieser Projektarbeit immer wieder aufgenommen und im Kontext der Arbeit und des Projektes kritisch analysiert und dessen Umsetzung dokumentiert.

## 3. Produkt

### 3.1 Methodik

Um die gesetzten Ziele erreichen zu können war eine exakte Methodik unerlässlich und so wurde schon früh, gemeinsam mit den Stakeholdern, die Schnittstellen und Anforderungen der anderen Gruppen ermittelt. Dies ermöglichte es der Gruppe logStore, nach geeigneten technologischen Lösungen zu suchen und diese früh auf ihre Funktionalität zu testen.

Der wichtigste Punkt der Methodik, welche von der Gruppe logStore verwendet wurde, war das Trennen der Zugriffe und das Einschränken der Rechte. Sowohl die Applikationsgruppen wie auch die Netzwerkschicht-Gruppen sollten nur über klar definierte und abgesprochene Schnittstellen Zugriff auf die Daten erhalten. Dabei sollen beide Seiten nicht eine allgemeine Datenhoheit erhalten und zum Beispiel das Löschen von Daten sollte für beide Seiten unmöglich sein. Auch sollten hiermit illegale Zugriffe von Dritten möglichst eingeschränkt werden, allerdings war der Sicherheitsaspekt in der Gruppe logStore keine direkte Priorität.

Eine der ersten Entscheidungen war es, datenbanktechnisch mit der Lösung SQLite [7] zu arbeiten. Dies ermöglichte es uns, auf den meisten Endgeräten problemlos eine Datenbank zu erstellen, welche der Grösse des Projektes mehr als genügte und auch speicherplatztechnisch gut vertretbar war. Weiter ermöglichte SQLite es, zusammen mit der bekannten Python Bibliothek sqlalchemy [8], mit wenigen Zusatzinstallationen das Datenbank-management effizient und schlank zu gestalten. Zusätzlich konnte durch ein effizientes Session-management der Datenbankkommunikation unnötige Schwachstellen und Prozessorbelastung präventiv verhindert werden. Abschliessend zur Datenbank-Schnittstelle kann noch erwähnt werden, dass durch das effiziente Einsetzen von Anfragen an die Datenbank die Abfragezeiten optimiert werden konnten. In der Methodik bedeutete dies, dass es klare Abläufe bei einem Datenbankzugriff geben soll, welche die Datenbank vor ungültigen Zugriffen schützen und so die Integrität der Daten wahren sollte.

Neben der Integrität der Nachrichten war die Integrität eines Feeds ein weiterer wichtiger Punkt, der schon in der Methodik beachtet werden musste. Zur Erinnerung, ein Feed ist eine Kette von Paketen, bei der jeweils jedes Paket auf das vorherige referenziert. So wurde früh zusammen mit logSync und logMerge eine einheitliche Paketform definiert und diese wurde durch das EventCreationTool der Gruppe logMerge auch bei allen Applikationsgruppen effizient und früh durchgesetzt. Das EventCreationTool ermöglicht es einfach neue Feeds und Pakete zu diesen Feeds zu erstellen. Dies erleichterte es sowohl für die Applikationsgruppen, da diese wussten in welcher Form sie der Schnittstelle von logStore die Daten übergeben konnten sowie auch für die Kontrollmechanismen, da diese von einer einheitlichen Paket Form ausgehen konnten und ungültige Pakete schnell entdeckt und nicht gespeichert und nicht übertragen wurden. Weiter wurde zusammen mit der Gruppe feedCtrl und logMerge ein Protokoll entwickelt, welche die Kontrolle der Feeds an sich ermöglichen sollte. Dieses Protokoll wird im Punkt Resultat noch weiter besprochen.

Damit wurde von Anfang an in der Methodik auf eine klare Form der Daten und auf eine klar definierte Form der Datenspeicherung gesetzt, was im Kontext die Umsetzung der Ziele erleichtern, beziehungsweise die Umsetzung auch einfacher überprüfbar machen sollte.

### 3.2 Prozess

Die Gruppenteilnehmenden Viktor Gsteiger und Moritz Würth hatten bereits mehrere Projekte zusammen durchgeführt, wodurch die Arbeitsabläufe schon routiniert und effizient waren. Ein wichtiger Teil des Prozesses war von Anfang an die Integration des Projektes mit den anderen Gruppen. Moritz Würth kümmerte sich aus diesem Grund um die Kommunikation mit den anderen Gruppen. Dadurch wurde schon relativ früh im Arbeitsprozess die Schnittstellendefinition ein wichtiges Thema, da das Projekt logStore ein wichtiger und hoch integrierter Teil des Projektes ist.

Der erste Schritt im Projekt war es eine standardisierte Datenbank zu erstellen, die wir zu belieben auf die jeweiligen Gruppen anpassen konnten. Dabei wurden die Rollen aufgeteilt in Schnittstellen zu den Applikationen wie subChat [9] und KotlinUI [10] und Schnittstellen zu logSync und logMerge. Viktor Gsteiger übernahm die Schnittstellen zu den Gruppen logSync und logMerge und Moritz Würth die Schnittstellen zu den Gruppen subChat und KotlinUI. Die Datenbankanbindung und Projektstruktur wurde grösstenteils von Viktor Gsteiger entwickelt und umgesetzt. In der Anfangsphase des Prozesses wurde viel Zeit für die Recherche betreffend der Datenbankanbindung und Gestaltung aufgewendet. Als ein erster Entwurf der Datenbank als Prototyp

funktionierte, wurde ein aktiver Kontakt zu den Interessensgruppen hergestellt, um die Datenbank auf ihre Bedürfnisse anzupassen. Die Bedürfnisse und die technischen Anforderungen wurde in zahlreichen Zoom-Meetings abgeklärt, welche es der Gruppe logStore ermöglichte, das Produkt klarer zu definieren und erste Schnittstellenprototypen bereit zu stellen. Im weiteren Verlauf übermittelten die verschiedenen Interessensgruppen detaillierte Dokumente, welche Methoden mit Parametern und Rückgabewerten enthielten. Damit wurde die Datenbank auf die Bedürfnisse der Gruppen angepasst, erweitert und mit den nötigen Aufrufaktionen versehen. Im weiteren Verlauf kam es immer wieder zu kleinen Änderungen an der Datenbank, die dazu führten, dass weitere Gruppen informiert werden mussten. Dies reduzierte sich allerdings nach und nach bis schliesslich die Erwartungen der Gruppen erfüllt wurden.

Im abschliessenden Teil des Prozesses wurde zuerst die Integration mit der Gruppe feedCtrl durchgeführt, da die beiden Produkte die stärksten Bindungen zwischen den Modulen haben. Fortführend wurden die Netzwerkschicht-Gruppen mit ihren Anwendungen integriert. Dabei wurde auch das Ablaufprotokoll, welches die Integrität der Pakete und Feeds überprüft, geprüft und integriert. Dank der guten Zusammenarbeit der Gruppen feedCtrl, logMerge, logSync und logStore war dieser Integrationsprozess zwar langwierig, jedoch im Endeffekt erfolgreich. In einem letzten Schritt mussten die Applikationsgruppen noch die korrekte Anbindung an die bereits integrierten Teile implementieren beziehungsweise testen. Am Schluss des gesamten Integrationsprozesses und damit auch des Arbeitsprozesses der Gruppe logStore konnte das Produkt der Gruppe logStore in allen Projekten, die an das allgemeine Netzwerk angeschlossen waren erfolgreich eingesetzt werden.

### 3.3 Resultat

Das Produkt der Gruppe logStore ist eine schlanke Anwendung, die es den Applikationsgruppen ermöglicht entweder durch allgemeine Schnittstellen oder durch individuell angepasste Schnittstellen mit der Speicherkapazität einer effizienten SQLite Datenbank zu interagieren. Durch eine frühe Kommunikation konnte für die beiden im Projekt beteiligten Applikationsgruppen individuelle Schnittstellen erstellt werden, welche es jenen erlaubt, effizient Daten zu speichern und damit vorsortierte und gefilterte Informationen zu erhalten. Diese Vorgehensweise nimmt den Gruppen eine grosse Last ab. Durch Kontrollmechanismen bei den Datenbankzugriffen und bei der Rückgabe der Daten kann den Gruppen die Feed Integrität sowie auch die Nachrichten Integrität ermöglicht werden.

Weil die Netzwerkschicht-Gruppen von logSync und logMerge und die Applikationsgruppen ziemlich unterschiedliche Informationen brauchen wurde mit unterschiedlichen Tabellen gearbeitet, wobei die Tabelle für die Netzwerkschicht-Gruppen alle Pakete speichert, welche entweder von den Applikationen oder den Netzwerkschicht-Gruppen korrekt eingegeben werden. Dies wahrt die Integrität der Pakete und gewährt, dass diese korrekt in das Netzwerk übertragen werden können. Die Tabelle für die jeweiligen Applikationsgruppen speichert allerdings nur die für die jeweilige Applikation relevanten Informationen und sortiert, beziehungsweise organisiert diese in einer Weise, die die Abfragen auf die relevanten Datenbankinformationen vereinfachen. Da die Abfrageanforderungen der Datenbankzugriffe je nach Applikation unterschiedlich sind, wurde für jede Applikation eigens eine Tabelle und dazugehörige Zugriffe definiert. Im Falle einer Erweiterung könnte dies auch einfach umgangen werden, indem eine Applikationsgruppe direkt auf die Tabelle für die Netzwerkschicht-Gruppen zugreift, dabei müsste diese neue Applikation jedoch die Filter und Sortierungsfunktionalitäten eigens implementieren.

Weiter konnte durch eine frühe und enge Zusammenarbeit mit der Gruppe feedCtrl ein fertiges Produkt angeboten werden, dass schlank und effizient die Transportschicht des Netzwerkes zur Verfügung stellt. So wird den Netzwerkschicht-Gruppen ermöglicht, die Feeds vor dem annehmen beziehungsweise dem versenden zu verifizieren und dadurch die Feed- und Paketintegrität zu wahren.

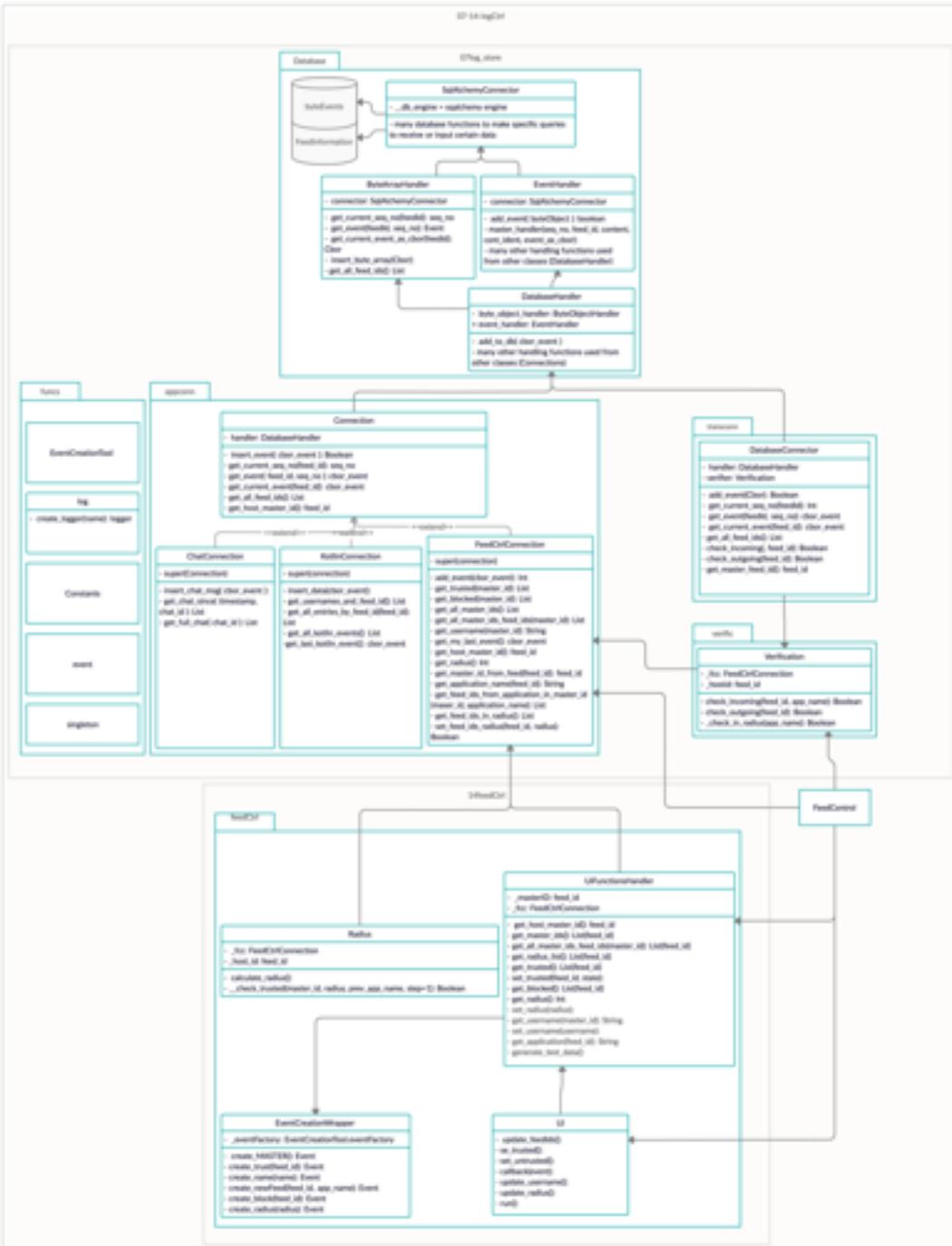


Abbildung 1: UML Klassendiagramm der 07-14-logCtrl Anwendung

Wie man in der Abbildung 1 klar erkennen kann, sind die Applikationen der Gruppen logStore und feedCtrl stark voneinander abhängig. Hierbei ist die Applikation der Gruppe logStore im oberen Teil der Abbildung zu sehen und die Applikation der Gruppe feedCtrl im Unteren. Zusammen bilden sie die Applikation logCtrl [11]. Im unteren Bereich ist die Anwendung der Gruppe feedCtrl aufgelistet, welche vor allem auf die Klasse FeedCtrlConnection zugreift, um mit der Hilfe dieser wichtige Datenbankabfragen zu machen. Die FeedCtrl-Connection ermöglicht es der Gruppe feedCtrl, ein effizientes User Interface darzustellen, welches den Knoten die Feed-Verwaltung ermöglicht. Weiter ist die Klasse Verification ein essentieller Teil des logStore Paketes, da sie den Netzwerkschicht-Gruppen erlaubt, Pakete und Feeds zu verifizieren und so die Sicherheit, Integrität und Effizienz des Systems wahrt.

Abschliessend gilt es im Hinblick auf Abbildung 1 zu erwähnen, dass aus historisch gewachsenen Gründen zwei unterschiedliche SQLite Datenbanken existieren, die im Database Paket zu sehen sind. Sie enthalten entweder die für die Applikationsgruppen relevanten Informationen oder die Pakete für die Netzwerkschicht-

Gruppen. Die Verifikation und die Funktionalitäten der Gruppe feedCtrl wird hierbei zu den Applikationsgruppen gezählt, da die Datenbankabfragen jener Applikationen relativ aufwändig sind und allein mit Paketen der Netzwerkschicht-Gruppen nicht zu stemmen wären.

Die Kontrollmechanismen von feedCtrl wurden, wie zuvor erwähnt, auf Datenbank-Ebene mit einer eigenen Master Tabelle in der Applikationsdatenbank implementiert. Einmal mehr gilt es an dieser Stelle anzuführen, dass diese Kontrollmechanismen als Applikationszugriffe behandelt werden und die gleichen Zugriffsrechte wie die anderen Applikationsgruppen besitzen. Weiter wurde für den Zugriff und die Speicherung von Daten auf der Master Tabelle zusammen mit feedCtrl und logMerge ein Zugriffs- und Ablaufprotokoll definiert. Hierbei ist es entschieden, dass jeder Knoten des Netzwerkes einen eigenen master-Feed hat, der alle Applikations-feeds dieses Knotens bei sich registriert. Jener master-Feed wird vor allem für die Funktionalität von feedCtrl verwendet, weshalb in diesem Bericht nicht näher auf die Thematik eingegangen wird. An dieser Stelle sei allerdings angemerkt, dass die Funktionalität des master-Feeds sehr abhängig von den dazugehörigen Datenbankanbindungen ist und ohne die eigene Tabelle nur sehr ineffizient hätte implementiert werden können.

Weiter wurde durch eine persistente Datenbank verhindert, dass eine logStore Applikation ständig laufen muss. Vielmehr werden die Methoden der Gruppe logStore bei Bedarf aufgerufen und sind so von sich aus sparsam mit der Prozessorleistung der Knoten des Netzwerkes.

Abschliessend lässt sich festhalten, dass das Produkt der Gruppe logStore, zusammen mit feedCtrl, logSync und logMerge die wichtigsten Punkte des TCP Protokolls implementieren. Dazu gehört das Liefern von Daten in der richtigen Reihenfolge, die von den Gruppen logMerge und logSync zusammen mit den wichtigen Datenbank-Anbindungen zur Verfügung gestellt werden, wie auch dass die Integrität der Daten durch das exakte Abspeichern in der Datenbank und anhand der Kontrollmechanismen bei der Speicherung und Abfrage gewahrt werden. Zusätzlich wird eine Übertragung in einem gewissen Massen garantiert, da bei einer fehlerhaften Übertragung die Daten erneut abgefragt werden. Abschliessend wird durch das zur Verfügung stellen eines persistenten Speichers auf den Knoten auch Überlastungs-Korrektur in einem gewissen Umfang umgesetzt, da in einem Falle von einer Überlastung des Übertragungsmediums die Pakete nicht verloren gehen und von den Gruppen logMerge und logSync durch eine Abgleichung nur ein gewisser Umfang von Paketen gleichzeitig übertragen wird. Dies ist jedoch nicht primär die Aufgabe der Anwendung der Gruppe logStore, die in einem Fall von Überlastung viel mehr als effizienter Puffer dient.

### 3.4 Integration

Zu Beginn des Projekts wurde mit allen beteiligten Gruppen ein Zoom-Meeting gemacht, um mit den anderen Projektgruppen unsere ersten Schritte und Ideen betreffend der Schnittstelle zu teilen. Im zweiten Meeting wurden diese Ideen spezifiziert und wir baten außerdem die einzelnen Gruppen uns ein Dokument zukommen zu lassen. In jenem sollten die Schnittstelle zur Datenbank detailliert mit Parametern und Rückgabewert beschrieben werden. Anhand dieses Dokuments konnten wir die Schnittstelle anschliessend implementieren. Im Anschluss wurde bei einem dritten Meeting den Gruppen unsere Schnittstelle inklusive Tests vorgeführt. Hierauf konnten die Gruppen die Datenbank in ihr Projekt übernehmen. Kamen neue Ideen dazu oder war etwas nicht so, wie die Gruppen es sich vorstellten, fing der Ablauf wieder wie beim zweiten Meeting an. Leider kamen solche Zurückversetzungen immer wieder vor, weil zu Beginn die Spezifikationen zu ungenau waren und außerdem Missverständnisse auf beiden Seiten auftraten. Mit zunehmender Genauigkeit der Angaben reduzierten sich die Missverständnisse und folglich auch die Rücksetzer. Neue Ideen wurden implementiert, ohne Bestehendes gross zu ändern.

## 4. Fazit

### 4.1 Produkt

Die Gruppe logStore konnte im Kontext der eigenen Ziele die Anforderungen erfüllen. So ermöglichte, dass das Produkt der Gruppe logStore den anderen Gruppen eine effiziente Anbindung und Schnittstelle nebst einer schlanken Datenbank, welche die Anforderungen der Speicherplatz-Effizienz wie auch einen sinnvollen Umfang von Datensicherheit erfüllt. Weiter konnte durch die erfolgreiche Zusammenarbeit in allen drei Demogruppen die Funktionalität live getestet und vorgezeigt werden.

Das Endprodukt erfüllt die angeforderten Fähigkeiten der Daten-Integrität sowie der Feed-Integrität und ermöglicht wichtige Teile des TCP Protokolls. Weiter wurde durch eine effiziente Dokumentation und durch ein umfangreiches Testen die Funktionalität brauchbar für die anderen Gruppen gemacht und die Funktionalitäten erfüllen einen selbst-festgelegten Standard. Die Gruppe logStore ist durchaus zufrieden mit dem Endprodukt, da die Funktionalitäten des Produktes für die anderen Gruppen in einem sinnvollen Masse garantiert werden können und die verschiedenen Interessensgruppen der Gruppe logStore sich auf das Produkt verlassen können.

Mit einem grösseren Rahmen hätte sich die Gruppe logStore noch zum Ziel gesetzt, die Ausnahme-Handhabung effizienter zu gestalten, da diese noch Verbesserungspotential hat. Konkret werden bei vielen Datenbankzugriffen lediglich die erhaltenen Informationen zurückgegeben und es wird nicht eine Ausnahme geworfen, falls eine Anfrage fehlerhafte Informationen zurückgibt. Weiter würde das Produkt in seiner Paketform noch schöner zusammengeführt und möglichst als eigenständiges Paket mit dem Paket-Manager PIP verbunden werden. Zusätzlich wäre bei einem grösseren Projektumfang die Effizienz einiger Datenbankzugriffe noch analysiert und womöglich verbessert worden. Da jedoch der Hauptfokus des Projektes auf einem funktionierenden und gut integrierten Produkt lag, wurden diese Punkte priorisiert.

### 4.2 Prozess

Der Prozess der Gruppe logStore wurde schon früh, mit adäquaten Zwischenzielen die gut eingehalten werden konnten, geplant. Bei der Integration mit den Interessensgruppen wurde der Aufwand zum Teil von den beteiligten Parteien unterschätzt. Es benötigte viel mehr Treffen, die leider wegen den aktuellen äusseren Umständen nicht immer in gleich effizienter Weise durchgeführt werden konnten. Trotz all dem konnte durch eine aktive Kommunikation und durch den starken Einsatz aller beteiligten Gruppen potenzielle Fehler in der Prozessplanung korrigiert werden.

In einem nächsten Projektzyklus würde vermutlich die Interessensgruppen-Analyse und Anforderungseinholung früher priorisiert werden. Dies würde es allen Beteiligten ermöglichen, eine effizientere Umsetzungs- und Integrationsstrategie durchzuführen. Weiter würde im Prozess das test-orientierte Entwickeln beibehalten werden, da dies die Effizienz der Integration erleichtert und zudem sicherstellt, dass die kommunizierten Schnittstellen die geforderten Anforderungen erfüllen.

### 4.3 Gesamtprojekt

Das Produkt der Gruppe logStore steht im Kontext des gesamten Projekts sowie auch den Zielen des Projekts gut da, da alle Schnittstellen mit den anderen Gruppen funktioniert haben und ebenso die Integration grössenteils sehr erfolgreich war. Zusätzlich konnte durch eine proaktive Zusammenarbeit und mithilfe einer proaktiven Kommunikation mit den verschiedenen Interessensgruppen des Projekts das Ziel eines funktionierenden Netzwerkes stark gefördert werden. Die Gruppe logStore sieht sich deshalb zusammen mit vielen anderen Gruppen als integraler Teil des Netzwerkes. Insgesamt ist die Gruppe logStore stolz auf ihr Produkt, insbesondere, weil das Produkt bei allen drei Gruppendemonstrationen, an denen die Gruppe logStore beteiligt war, funktioniert hat, eingesetzt wurde und auch live vorgezeigt werden konnten. An diesem Punkt sollte auch noch die sehr gute Zusammenarbeit mit den Gruppen logMerge und feedCtrl erwähnt werden, ohne deren starke Zusammenarbeit die erfolgreiche Präsentation in den verschiedenen Gruppen sicherlich nicht möglich gewesen wäre.

## 5. Abbildungsverzeichnis

Abbildung 1: UML Klassendiagramm der 07-14-logCtrl Anwendung

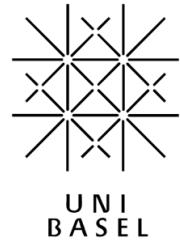
6

## 6. Referenzen

- [1] JAMES F. KUROSE, K. W. (2013). *Computer Networking A Top-Down Approach*. New Jersey: Pearson Education, Inc.
- [2] DOMINIC TARR, ALJOSCHA MEYER, ERICK LAVOIE and CHRISTIAN TSCHUDIN (2019). Secure Scuttlebutt: An Identity-Centric Protocol for Subjective and Decentralized Applications. Macao China: Association for Computing Machinery.
- [3] VINTON G. CERF and EDWARD CAIN (1983). The DoD Internet Architecture Model. North-Holland: Elsevier Science Publishers B.B.
- [4] DAMIAN KNUCHEL, YANNICK RÜMMELE und WILSON AILEN EGHONGHON (2020). Group 14 feedCtrl. Letzter Zugriff: 2020.07.18, 08:55 CEST. <https://github.com/cn-uofbasel/BACnet/tree/master/groups/14-feedCtrl>.
- [5] JOEY ZGRAGGEN, NIKODEM KERNBACH und GÜNES AYDIN (2020). Group 04 logMerge. Letzter Zugriff: 2020.07.18, 09:09 CEST. <https://github.com/cn-uofbasel/BACnet/tree/master/groups/04-logMerge>.
- [6] ALEXANDER OXLEY und CARLOS TEJERA (2020). Group 12 logSync. Letzter Zugriff: 2020.07.18, 09:02 CEST. <https://github.com/cn-uofbasel/BACnet/tree/master/groups/12-logSync>.
- [7] HIPP, WYRICK & COMPANY, Inc. (ohne Datum). *SQLite*. Letzter Zugriff: 2020.07.18, 09:11 CEST. Von SQLite: <https://www.sqlite.org/docs.html> abgerufen.
- [8] MICHAEL BAYER (Juni 2020). *SQLAlchemy*. Letzter Zugriff: 2020.07.18, 09:13 CEST. Von SQLAlchemy: <https://docs.sqlalchemy.org/en/13/> abgerufen.
- [9] KEN ROTARIS und TUNAHAN ERBAY (2020). Group 03 subChat. Letzter Zugriff: 2020.07.18, 09:14 CEST. <https://github.com/cn-uofbasel/BACnet/tree/master/groups/03-subChat>.
- [10] SANJA POPOVIC, NOUR HANY und TRAVIS RIVERA PETIT (2020). Group 10 KotlinUI. Letzter Zugriff: 2020.07.18, 09:16 CEST. <https://github.com/cn-uofbasel/BACnet/tree/master/groups/10-KotlinUI>.
- [11] DAMIAN KNUCHEL, YANNICK RÜMMELE, WILSON AILEN EGHONGHON, VIKTOR GSTEIGER und MORITZ WÜRTH (2020). logStore – feedCtrl: logCtrl. Letzter Zugriff: 2020.07.18, 09:18 CEST. <https://github.com/cn-uofbasel/BACnet/tree/master/groups/07-14-logCtrl/src>.

## 7. Weiterführende Referenzen

- CATHERINE PLAISANT, B. S. (July 1999). Interface and data architecture for query preview in networked information systems. *ACM Transactions on Information Systems*.
- POPOV, S. (2017). *On the timestamps in the tangle*. Berlin: IOTA Foundation.
- SHINJI KIKUCHI, A. M. (2011). *Databases in Networked Information Systems*. Japan: Springer, Berlin, Heidelberg.
- SZALACHOWSKI, P. (2018). Towards More Reliable Bitcoin Timestamps. SUTD.



## PROJECT REPORT IN THE CONTEXT OF INTERNET & SECURITY

# LoRaSense

Gathering and Distribution of Environmental Data

Group 9: Leonardo Salsi & Patrice Delley  
July 19, 2020

## Abstract

Environmental data can be collected in various ways. Within the scope of the project the goal was defined to collect weather data at several locations. Normally this is done at weather stations with the nodes not interacting with each other thus making it hard to collect all data. We created a network where at every so-called node all the information is present. We did this with the help of LoRa. The data from the sensors is read and sent to a central node where it is evaluated. Furthermore, the nodes should be able to receive commands and process them accordingly, such as the time span between the individual measurements making it easy to change settings from afar without needing to reprogram and upload to single nodes. Our project is able to measure temperature, humidity, air pressure and light intensity. The communication between the nodes is stable thanks to LoRa and a very small loss of data was noticed. The environmental data can then be visualized with the help of software "SensUI".

## 1 Introduction

LoRaSense is the collection of nodes that are capable of collecting environmental data and sharing information with each other via a LoRa network. Each node should not only gather data and distribute, but also receive information from other nodes, such that all nodes hold the same set of data. This leads to a decentralized system. Furthermore, LoRaSense must allow to extract said data in order to sort and visualize it without needing to visit or have a direct connection to every node.

The central goal of our contribution can be summarized in

*Can we create a network where environmental data is collected and shown from multiple locations and distributed in order to guarantee data consistency over the entire network?*

This question was too open in its formulation for the beginning, which is why we decided to split it into sub-questions to which we also referred as milestones. By answering one of the following question we made a significant step towards the realization of the project.

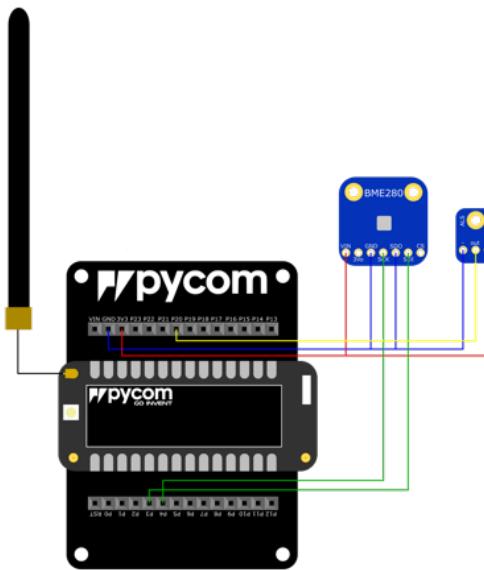
1. What information do we want to collect?
2. What hardware is suitable for our project?
3. How can we represent the data?
4. Is LoRa the suitable transmission technology for us?
5. What are the necessary interfaces in order to allow easier implementation in other projects?
6. How can data consistency be guaranteed?
7. In the case that a node shuts down, what steps should be taken?
8. How scalable is LoRaSense?
9. How can we extract the data to a PC and visualize it?

In the early stage of development our group started collaborating with LoRaLink and with SensUI. LoRaLink focused on the interconnectivity between the nodes while SensUI developed a program that displayed the collected data in a detailed and comprehensive manner.

## 2 Background

### Hardware

- Pycom's LoPy 4 is a microcontroller that is able to run *MicroPython* and has a native interface for LoRa connectivity. Along with the expansion board it provides pins allowing to connect sensor chips to the LoPy. It also included the LoRa antenna.
- The *BME280* is a chip that is able to gather information about temperature, humidity and air pressure. It provides a connection to the board either via *I<sub>2</sub>C* or *SPI*. In our project the first connection variant was chosen.
- The *ALS* is a chip that is capable to read the light intensity and translates it into a voltage that can be directly read from the pin.
- Heltec LoRa 32 is a development kit similar to the LoPy 4 that is controlled by the arduino programming language. By default this module has a small display attached to it. We used this only at the beginning testing how to work with LoRa and grasping the general idea, since it was easy to debug due to the small display.



How we connected the BME280 and the ALS to the LoPy 4's expansion board.

### Software

- The IDE used for development was *Visual Studio Code* since it was compatible with *Pymakr*, a tool that uploads to and runs code on LoPy.
- Since LoPy is not able to understand the entirety of Python, the project was written in an abridged version called *MicroPython* which is an abbreviated version of Python.

## 3 Implementation

### 3.1 Stages of Development

The development of LoRaSense can be split up into 5 phases. Each had a different main focus.

#### Phase 1: Early Stage

The early stage was mainly used for planning. We first needed to set up a concept to decide what kind of parameters we were going to measure. According to our decision, we chose the sensor chips that can be connected with our LoPy-modules. Due to LoPy only being able to work with MicroPython, we had to set up the environment and develop a sense for how it works and write first samples, such that when the parts arrive we could test the hardware. At the end of the early stage, we knew *how* the tools are to be used.

Table of labor: Table 1 in the appendix.

#### Phase 2: Hardware Testing Stage

As soon as all hardware was delivered, we had to make sure that all parts were working according to our expectations. Since we both had to work remotely, one of us had to take upon the first hardware-related tests and implementations. Leonardo mostly worked on the LoPy 4 where Patrice worked with the Heltec LoRa 32. After this phase, Patrice also started working on the LoPy as the Heltec devices were not a promising option aside from debugging.

At the end of the testing stage, we were certain that all components were working as expected.

Table of labor: Table 2 in the appendix.

#### Phase 3: Prototyping

First code samples were written in this stage in order to make measurements of temperature and humidity only for testing purposes. LoRaLink was not implemented in this stage yet, which is why we sent data in raw format from one node to another via LoRa. This is also where we decided what form the data should take on when working with LoRaLink. With that, a protocol was formed that would later prove important for SensUI as well. This included all measurements as well as a time-stamp for each round of measurements. A library was created that serves as the interface for the BME280 sensor. Also, the network we developed was more extensive, since now connecting more than 2 nodes were possible.

At this point, we knew how the chips are connected to the LoPy and how LoPy sends data via LoRa and is able to be read on the console of a connected computer.

Table of labor: Table 3 in the appendix.

#### Phase 4.1: Implementation of LoRaLink

LoRaLink was responsible for distributing data through a network. We were provided an interface allowing us to store data on each node and thereafter it would update the other nodes with said information.

After this stage of development, each node of LoRaSense sent data to each other such that all nodes held the same dataset.

Table of labor: Table 4.1 in the appendix.

#### Phase 4.2: Inclusion of SensUI

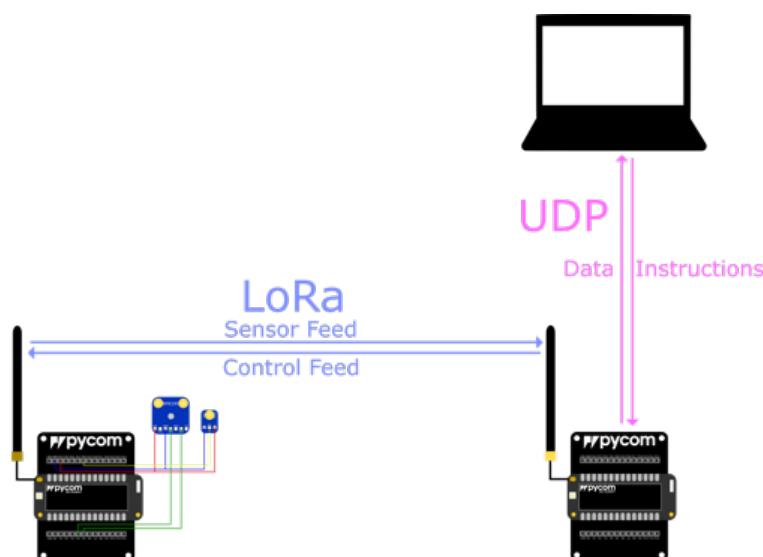
We needed to send the data from the nodes to a computer and chose to use a central node, which resides in the same network as the computer SensUI is running on, that collects the data from all nodes. It also served as a sender such that commands could be sent to said nodes. We also helped with the setup for the UDP connection between LoPy and a computer, since we had some experience with it following our efforts in giving each measurement a time-stamp. The inclusion of SensUI was mostly between SensUI and LoRaLink though it was beneficial for us to be integrated into that workflow as we were able to help.

Hereafter we were able to give commands to the nodes as well as receive and display information gathered on different nodes. It was only ever tested with one node since we did not have more than 2 LoPy at any given place and with a configuration as such everyone was able to test it for themselves.

Table of labor: Table 4.2 in the appendix.

#### 3.2 Set-Up Concept

We differentiate between sensor node and a master node. Sensor nodes collect and hold environmental data and feed them via the *sensor feed* to the master node, which in addition to the LoRa-network is also connected to the WiFi. This connection serves the purpose to forward collected data to a computer running SensUI via UDP. SensUI also allows to change the frequency in which measurements are made. If the user changes said frequency, the master node informs all sensor nodes via the *control feed* that said adjustments must be made; the sensor nodes then act accordingly.



## 4 Results and Discussion

We now can collect the following data:

1. Temperature
2. Humidity
3. Air Pressure
4. Light Intensity

And we decided the suitable hardware for the project would be:

1. LoPy 4
2. BME280
3. ALS

Our first prototype with Heltec modules worked faster than we expected. It did not take a long time to get it working and putting out random numbers generated on the device. The problem with Heltec LoRa 32 was that we could not connect sensors to the device. While not being the most elegant solution, we printed the data onto the serial monitor built into the Arduino IDE.

We tried around with different technologies such as WiFi as well but decided to stay with LoRa since it seemed to be best suited in our project with higher ranges, which would represent the real world.

As we started working with the LoPy 4, testing with random numbers through LoRa went well. Problems arose when we connected the sensors to the board. We created a library tasked with being the interface between the LoPy 4 and the sensors.

- LoRaLink was able to guarantee data consistency and safety through frequent updates of the nodes. It also was able to recover from a node that shut down (when the battery ran out). With only said node needing to be restarted (after swapping battery) would thereafter be functional again.
- SensUI was useful for giving us an interface with which we could issue commands to the second channel of communication. It also provided a graphical representation of the data collected.

The combination of the three individual projects was certainly time-consuming due to interface problems with connection for example. LoRaSense provides the measurements to LoRaLink which transport it to SensUI. Finally, SensUI then represents the transmitted data. We were able to achieve what we set out to do. We created a scalable network of nodes that collects weather data and is graphically represented.

## 5 Lessons Learned

A problem we encountered was a difficult setup of the LoPy 4 including its expansion board. Loading the original firmware onto it was even with a tutorial hard. When Patrice encountered an error not yet documented, he was able to find a solution through going on a forum and asking about it. Forums can be extremely helpful and should be encouraged to be used more.

Another aspect that stood out to us was working in groups. Now this is more or less normal but what made it interesting was multiple groups working together. Each having their vision and working on their own at first. Then coming together and working alongside each other to create something everyone can be proud of.

It has to be mentioned though that it is not always easy! Sometimes you have questions and want answers fast but we were lucky since all groups participating were always easily reachable and were always willing to go into a quick zoom-session.

Organization for the whole project was a little chaotic at first. The COVID-19 situation certainly did not make this any easier. For quite a long time it was unclear with which groups we would be working together. We believe that a complete overview from the beginning would have been helpful. Still given the special circumstances it was a really interesting experience with lots of problems that needed solving.

Not being able to meet up was quite a challenge, since we were unable to link up more nodes than shown in our set up. We ordered more breadboards needed to connect all sensors to the expansion board, but they failed to arrive in time.

## 6 Appendix

Table 1

	Leonardo	Patrice
Conception	x	x
Determining hardware	x	x
Raw connection scheme		x
First code samples	x	

Table 2

	Leonardo	Patrice
Component testing	x	x
First wiring	x	

Table 3

	Leonardo	Patrice
Hardware prototyping	x	
Environmental data formatting		x
Programming	x	x

Table 4.1

	Leonardo	Patrice
Inclusion of LoRaLink	x	x

Table 4.2

	Leonardo	Patrice
Interface programming	x	
Programming interpreter for commands		x

Here a few photos taken of the project.



Figure 1: Heltec LoRa 32 receiver display

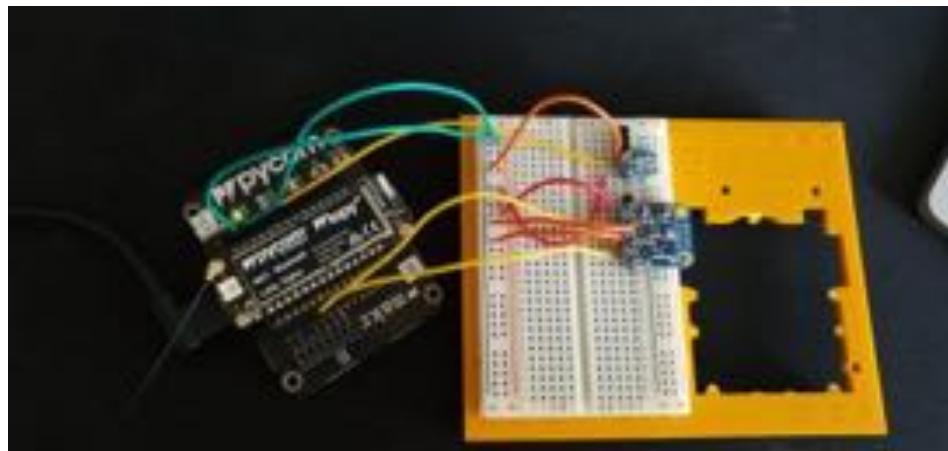


Figure 2: Setup for a measuring node



## BACNET ANDROID APP

COMPUTER SCIENCE DEPARTMENT  
INTERNET AND SECURITY  
FS 2020

---

Sanja Popovic  
Nour Shokry  
Travis Rivera Petit  
— Group 10 —

# Contents

<b>1 ABSTRACT</b>	<b>1</b>
<b>2 INTRODUCTION</b>	<b>1</b>
2.1 BACnet++ . . . . .	2
2.2 Background . . . . .	2
<b>3 IMPLEMENTATION</b>	<b>2</b>
3.1 Setup . . . . .	2
3.2 Functionality requirements . . . . .	2
3.3 Technical requirements . . . . .	3
<b>4 DIVISION OF LABOR</b>	<b>4</b>
<b>5 RESULTS</b>	<b>4</b>
5.1 Full app description summary . . . . .	4
<b>6 FUTURE WORK &amp; COMMON PITFALLS</b>	<b>5</b>
<b>Bibliography</b>	<b>6</b>

---

## 1 ABSTRACT

In recent years, the privacy concerns of centralized communication protocols has come under criticism, giving rise to the interest in non-centralized communication.

The Basel Citizen network (BACnet) is a decentralized log-based communication protocol whose event structure uses feeds which consist of append-only logs that are serialized using CBOR. No entity has full knowledge of the entire communication graph, in other words, messages between peers are sent directly without being processed by server.

This work is part of a course project consisting of several groups that aims at creating fully-fledged BACnet-based applications. Our task is to develop a social media application and couple it with back-end provided by other groups in this project. The social media app, called *BACnet++* enables users to post and exchange messages securely via QR-Codes, change their usernames, follow or unfollow friends, and add new contacts.

In this text we cover the techniques used to develop the front-end of the application. In particular, we cover the integration of features that have been provided by the other groups in this project.

## 2 INTRODUCTION

The Basel Citizen Network (BACnet) is a decentralized communication protocol where messages between users bypass any intermediary server. This is possible due to the nature of decentralized networks, where the nodes connected to it are independent from any server point in the network. This type of non-centralization has become attractive in recent years due to the mistrust in large companies whose applications work like a black box – raising privacy concerns for the users.

Decentralized applications require a new approach in their program architecture since most of the traditional methods known today are based on the use of central servers. For example, topics like device synchronization, and authentication must be done independently on every device, which has no real knowledge of the other users on the platform other than a few.

As part of a course project, students attending the *HS2020 Internet & Security* lecture have been working together with the aim of creating fully developed decentralized applications using the BACnet protocol. The students have been assembled into groups of 2-3 people, where each group has worked on a layer of the BACnet protocol stack. By overlaying the code of the groups together, three applications have been created: Subchat, a Whatsapp-like chat applications that runs on Linux and MacOS [1]. LoraSense and SensUI, weather data fetching applications that sends this information through long range wide area networks [2] [3]. BACnet++, an Android social media application. Our group has worked on the development of BACnet++, where we mostly focused on its front-end features since some lower-level functionality has been provided by other groups. The groups we worked together with are:

- *Group 2: qrLink*. They provide a feature that allows to pair new devices together with the use of QR codes.
- *Group 4: logMerge* They provide us an API for the creation of BACnet events.
- *Groups 7 & 14: logCtrl*. They provide an API tailored to the needs of our project for inserting and retrieving events into local BACnet databases. This includes the back-end functions for generating the master ID, changing usernames and setting contacts to trusted or untrusted.
- *Group 12: logSync*. They provide an API for BACnet database synchronization.

We were also responsible for overlaying the code of these groups and ours to create the fully-fledged application that has come to be BACnet++.

---

## 2.1 BACnet++

Our aim has been to develop a social media Android application using the BACnet protocol, which we called BACnet++. In it, users may post messages on a message board shared by the users' trusted contacts. This message board may vary from user to user, due to the fact that users may have different friends, and due to synchronization matters. Each device stores its own version of the board given that there is no server hosting the messages. In order to keep this social media going, pairs of devices can be used to synchronize with each other, updating the message board of each device with the latest messages of their contacts.

## 2.2 Background

Secure Scuttlebut (SSB) is a perfect example for a decentralized network. In SSB, all users have a local database and they exchange data directly between each other. When a user writes a new message, the data is going to be signed with the user's unique identity to make sure that the data is truly written by the user themselves.<sup>[4, 'Peer Connections']</sup> This is done by assigning each user a private key and a public key, also known as a feed ID. The private key is solely known by the owner, with which a new written message is signed. The public key is known by other users, who then can verify the source with the author's public key. An important feature of SSB is that events of an author are chained. Each event has a reference to the previous event. The whole construction is called an append-only log or feed, where it is only possible to append new events and impossible to modifying older ones. <sup>[4, 'Feeds']</sup>.

The Basel Citizen Network (BACnet) is based on SSB. The aim is to overcome the disadvantages of a centralized network. BACnet extends and also modifies some features of SSB. In BACnet, a new message written by a user is wrapped up in a CBOR event that is a binary format instead of JSON, a text format used in SSB <sup>[5]</sup>. An important conceptional difference is the idea of master ID's. A master ID corresponds to an individual person. Each user has one unique master ID and can own multiple public-private-key pairs according to the amount of feeds they have or devices they use. For instance, a user could create different feeds for different topics. Another extension to SSB is the concept of onboarding. In SSB, a new SSB-user can connect to a pub to meet new people via the Internet, in particular, this allows the user to follow their first contact <sup>[4, 'Pubs']</sup>. Since it is desired for the project to fully work without the Internet, two users can directly follow each other and also synchronize their databases via QR-Link exchange.

# 3 IMPLEMENTATION

## 3.1 Setup

BACnet++ has been developed in the usual way for Android by using the Java programming language. However, the groups 4, 7, 12 and 14 have written their programs using Python, thus in order to couple them with our project, a tool for intermixing Java and Python code is called for. Chaquopy is a Python SDK for Android that enables developers to fuse Java, Kotlin and Python code <sup>[6]</sup>, we have worked with this SDK to ensure compatibility with other groups. Given that we are working on an open-source project, we applied for a licence and obtained it to get full access of Chaquopy.

## 3.2 Functionality requirements

Android SDK's Activities feature provides the window for each functionality that we had to draw its UI. We have been responsible for implementing the following three activities which can be accessed through the menu button.

---

## My/General Feed Activity

This is where the user can post and read messages, possibly accompanied by pictures. We decided to divide this activity into 2 tabs: *My Feed* tab, where the user can see all of their own posts, as well as *General Feed* tab, where the user can see the full history of posts. Attached to each post are timestamp and sender's username. When a user changes their username, a new post is automatically created to inform other users of this action. New posts are then encrypted and afterwards added to the database.

## Account Activity

This is where the user can change their username. When a username is changed, a message of type "username" is shown in the user's feed.

## Friends List Activity

The friends list shows the contacts the user paired with, which are set to untrusted by default. In this activity the user can set these contacts to trusted or untrusted. Only trusted contacts are used for database synchronization.

### 3.3 Technical requirements

Figure 1 shows the groups involved in the development of BACnet++. We have been responsible for the higher-level requirements of the application.



Figure 1: The posts are stored in the database. We used the database insertion and retrieval functions provided by the groups logStore and feedCtrl. logSync is responsible for managing which data has to be exchanged between two databases and qrLink takes care of the actual transmission of data.

## Network data transfer

The group qrLink have enabled us to transfer QR-codes between pairs Android devices, whereas group logSync provided us with the synchronization of BACnet databases between the two devices, determining which events should be transferred. This enabled the user to add new friends and stay updated with their posts.

## Database technology

The group logStore integrated their product early on with group feedCtrl to create logCtrl providing us with the interface needed to store, retrieve and filter data. LogCtrl is a Python API that has functions which are suitable for our functionality requirements such as updating the feed, changing the state of the currently selected feed and changing usernames. The API also enabled us to program a feature that creates a ID as soon as the app starts for the first time.

---

## 4 DIVISION OF LABOR

Each member was responsible for one of the mentioned functionality requirements.

**Travis Rivera Petit:** Responsible for creating My/General Feed, where users can post text and picture messages, as well as see their own posts and their friends' posts.

**Sanja Popovic:** Responsible for creating Account Settings where users can change their usernames and display their public keys, as well as keeping in touch with the other groups to ensure the integration process runs smoothly.

**Nour Shokry:** Responsible for creating Friends List, where users can see their friends' usernames, as well as select or deselect the friends they want to display on their general feed.

The members also worked together two to three times a week via screen sharing, to make sure that the activities worked as intended, as well as implement other aspects of the application such as integrating the technical requirements (storage and transfer of database).

## 5 RESULTS

We managed to build an app from scratch and couple the native Java code with the Python programs provided by other groups by using the Chaquopy SDK. Users communicate and share messages without being connected to the Internet. Our UI's design has proven to be user friendly and intuitive.

### 5.1 Full app description summary

In the resulting app, users can post messages and pictures and read posts from their trusted contacts. There exist two tabs: In the 'My Feed tab', the user can only view their own posts and in the 'General Feed' tab, the posts of the user's friends are displayed. The feed content varies from user to user since each device has its own local database which has to be manually synchronized by pairing it with another device.

Users are identified by an ID and are also given the option of choosing a username that is displayed on top of each of their messages. Users may change their username at any point in time without any restriction.

Adding contacts and synchronizing devices together is done by placing the devices in front of each other in near proximity and by then entering the QR pairing mode. This will cause the two devices to start sending QR codes from one device to the other.

After a user has been discovered, they're automatically added to the friends list and set to untrusted by default. Contacts set to untrusted do not have their messages displayed on the message board and are not used for synchronization. BACnet++ has a feature where contacts can be set to trusted or untrusted.

Finally, the app is divided into five activities: the main activity where users may post messages, the account activity where users may change their username, and the friends list activity where contacts may be set to trusted or untrusted, and the QR-send and QR-receive activities, the latter two of which have been developed by a different group and are not discussed in this work. Figure 2 shows screenshots of each of the app's activities that we worked on.

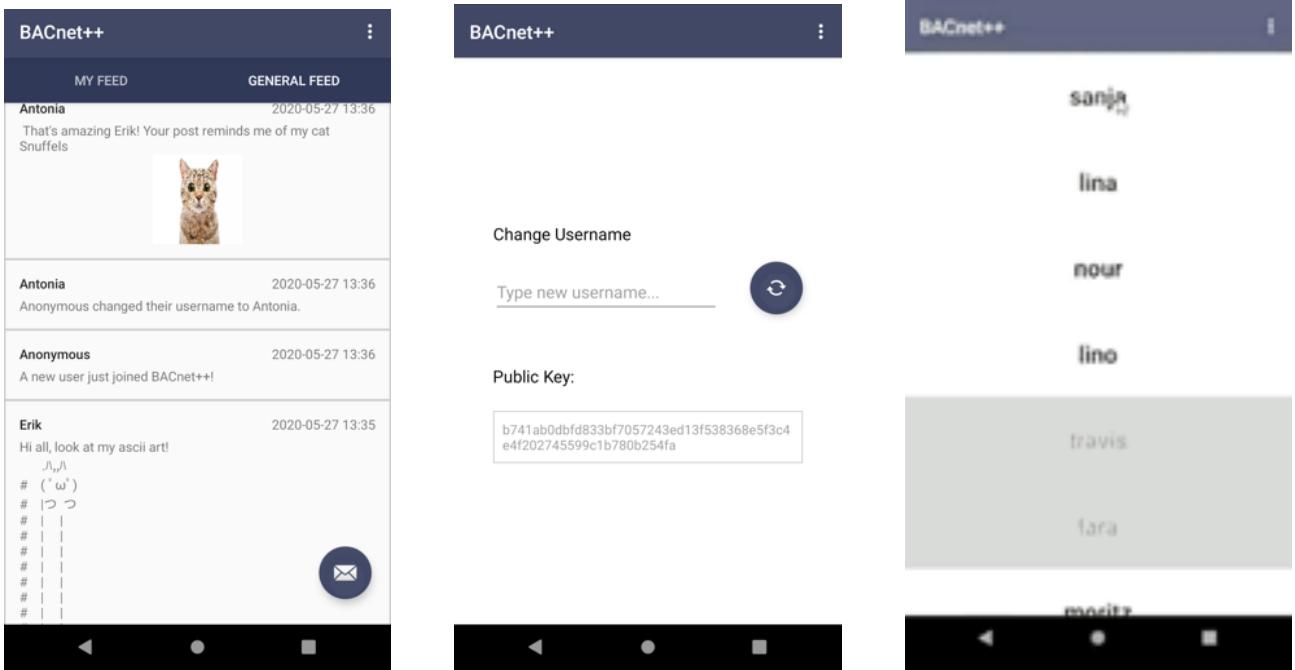


Figure 2: From left to right: the main activity, the account activity and the friends list activity. On top right: menu button. On bottom right of main activity: post button.

## 6 FUTURE WORK & COMMON PITFALLS

There are many ways in which this project can be expanded. Here we list down a few of them.

- *The use of public key identifiers next to usernames on the friends list.* The app could be slightly improved by showing the public key of each user along with its username in the friends list. This would have the benefit of being able to distinguish between any two users with the same username.
- *More social media features.* Profiles, like-buttons and private messages could also be implemented.
- *Surfing* A common challenge with SSB and BACnet is the fact that each user has to store a long feed of data. As users post more and more messages, the storage capacity can get out of hand. A solution for this could be surfing — a feature that automatically deletes old messages in order to prevent a storage overload —.
- *Image compression.* Posted images are resized to a fixed size and saved as a bitmap as part of a message. These bitmaps are long strings that can take much of a device's memory. A string compression algorithm implementation would help to mitigate this and thus improve the app's performance.

We learned a lot in this project. Many groups were involved in the development of an application where aspects like data transfer, GUI, database, synchronization of two databases and coupling them together needed good collaboration. We experienced that effective coordination and communication were essential to build an app, yet not easy to implement, especially given the current extraordinary circumstances.

---

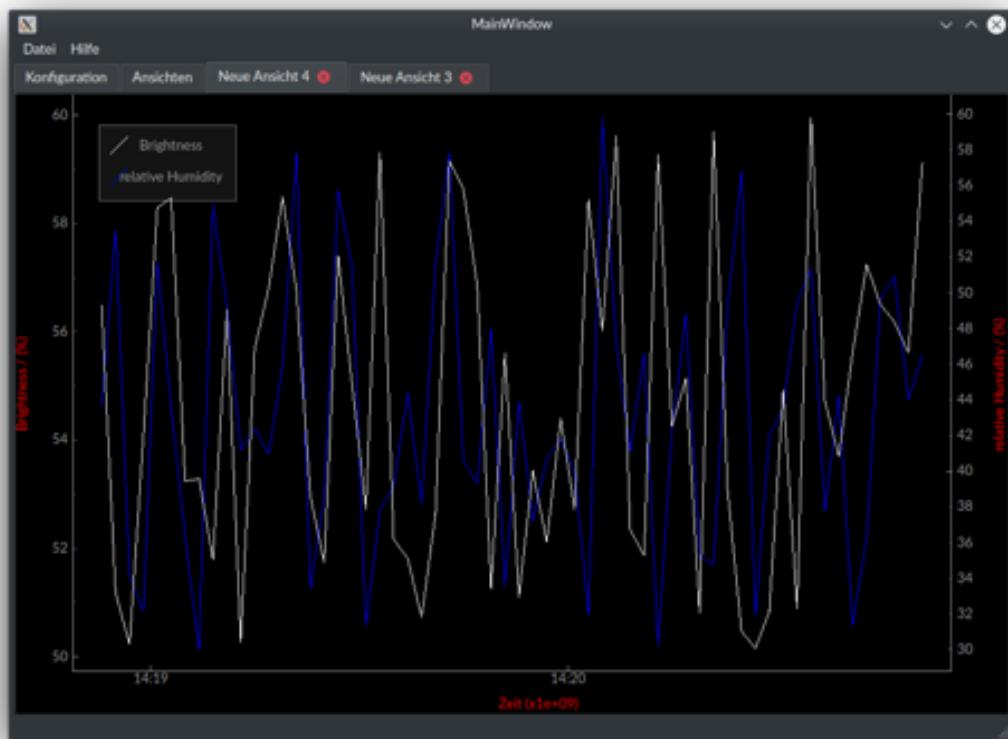
## Bibliography

- [1] Subchat. <https://github.com/cn-uofbasel/BACnet/tree/master/groups/03-subChat>.
- [2] Lorasense. <https://github.com/cn-uofbasel/BACnet/tree/master/groups/09-loraSense>.
- [3] Sensui. <https://github.com/cn-uofbasel/BACnet/tree/master/groups/11-sensUI>.
- [4] Scuttlebut protocol guide. <https://ssbc.github.io/scuttlebutt-protocol-guide/index.html>.
- [5] Bacnet event structure. <https://github.com/cn-uofbasel/BACnet/blob/master/doc/BACnet-event-structure.md>.
- [6] Chaquopy website. <https://chaquo.com/chaquopy/>.

# Project Report BACnet:SensUI

Marc Schäfer

Internet and Security FS2020



Version 1.0

## Contents

<b>1 Abstract</b>	<b>3</b>
<b>2 Introduction</b>	<b>3</b>
<b>3 Background</b>	<b>3</b>
<b>4 Implementation</b>	<b>4</b>
4.1 MainWindow . . . . .	5
4.2 NodeConfigTab . . . . .	5
4.3 ViewConfigTab . . . . .	5
4.4 ViewWidget . . . . .	5
<b>5 Results</b>	<b>6</b>
<b>6 Conclusions</b>	<b>6</b>

## 1 Abstract

SensUI is part of the BACnet<sup>[1]</sup> and provides a graphical user interface for the environmental measuring system LoRaSense<sup>[2]</sup>. The main focus is on displaying measured data live but it is also possible to view stored data and to configure LoRaSense nodes. The communication between SensUI and LoRaSense nodes is based on LoRaLink<sup>[3]</sup>. Qt5<sup>[4]</sup> is used as GUI framework and, as for most of the BACnet projects, the code is written in Python<sup>[5]</sup>. To visualize the data, custom views can be created and it is possible to store information about specific nodes like their name, position and a description. SensUI is fully event-driven, software-polling is not necessary. The combination of SensUI, LoRaSense and LoRaLink was successfully tested multiple times for demonstration purposes. This setup worked well but involved only one LoRaSense node and one SensUI GUI.

## 2 Introduction

One part of the **Basel Citizen Network** (BACnet) is the LoRaWAN<sup>[6]</sup> based sensor network LoRaSense. The LoRaSense project consists of multiple microcontroller-driven sensor nodes connected by LoRaWAN to capture data from environmental sensors. To use the captured data on other systems or configure sensors one can connect to the LoRaSense network via a node using WLAN. This node then redirects all measured data over the WLAN bridge to the client or receives configuration commands from the client. The LoRaWAN and WLAN connection is handled by LoRaLink. On the client side, SensUI is intended to receive this data and send the configuration commands.

SensUI provides a GUI for this purpose. The received data is visualized as XY- or XYY-plot, respectively. It is possible to display live data as well as historical data stored in the pcap<sup>[7]</sup> files of the feed layer. The visualization can be customized via multiples views to meet individual demands. For ease of use the nodes and views can be named and parameterized, the configurations are saved locally. Beside of this, nodes in the LoRaSense network can be configured directly in the GUI.

The plots can be exported in different formats, for example as rendered image or as raw, comma separated values (CSV). SensUI can manage multiple nodes with multiple sensors, currently limited to temperature, relative humidity, brightness and air pressure.

The GUI is platform independent and was tested with Ubuntu (Gnome Desktop), Kubuntu (KDE Plasma Desktop), Windows 10 and macOS 11.0.

## 3 Background

As mentioned before, SensUI is a Qt5 application. Qt5 is a popular, platform-independent GUI toolkit. While developed in C++, there are also a lot of bindings for other languages like PHP, Java and fortunately Python. Qt5 was chosen because of its very detailed and good documentation, included widgets, native platform look-and-feel and its event-based signal/slot system. Beside of this, Qt5 is easy to handle through its widget system and QML modeling language, its widely used and well tested. For example the desktop environment KDE Plasma<sup>[8]</sup>, the media player VLC<sup>[9]</sup> and the hypervisor Oracle VirtualBox<sup>[10]</sup> are using Qt5 for their graphical user interfaces. Another nice feature is the WYSIWYG-editor Qt Designer<sup>[11]</sup>. For the implementation, the most popular

Qt5 binding for Python was chosen, PyQt5<sup>[12]</sup>. Another, almost identical binding is PySide 2<sup>[13]</sup>. The major difference are of legal origin.

There are also many other GUI toolkits for Python. TkInter<sup>[14]</sup> is the standard library included in Python. The downside of this toolkit is its uncomfortable usability, the missing native look-and-feel and its much smaller distribution because it's a exclusive Python toolkit. Another option is the Atlas toolkit<sup>[15]</sup> for Python. The objective of Atlas toolkit is to provide a desktop application as web application and display it within a browser, a dedicated web server isn't necessary. Layouts are written in HTML and accessed by Python via the DOM. While this is a quick and easy way to create a simple GUI, this concept does not fit well for SensUI.

The most important aspect is to display sensor data as graph. The library considered best for this purpose is PyQtGraph<sup>[16]</sup>. Other libraries are for example Matplotlib<sup>[17]</sup>, PyQwt<sup>[18]</sup> or GuiQwt<sup>[19]</sup>. Matplotlib would be a good alternative, but PyQtGraph provides the best compatibility with Qt5, is easy to include into the project and actively maintained. Unfortunately, PyQwt and GuiQwt haven't been updated for a long time and were therefore out of the question.

PyQtGraph is a very versatile scientific plotting library so SensUI is using only the basic functions of it. The sensor data is drawn as XY- or XYY-plot, respectively. PyQtGraph also includes handling for mouse and keyboard actions like zooming or adjusting the viewing area so there is no extra effort necessary.

## 4 Implementation

First of all, a general layout of the GUI according to the following demands was developed:

- Multiple custom views
- Controls to create / modify / delete custom Views
- Configuration options for the nodes / sensors

For a clean and simple look, a tabbed view with the custom views and configuration pages as single tabs was considered best as main layout. Fortunately, Qt includes a "Tab Widget" which provides exactly this functionality. A very comfortable way of creating layouts with Qt is to use the Qt Designer and the QML modeling language. By this one can create its own widgets, stored as \*.ui-file.

In the next step, custom widgets were created for the main window (called MainWindow), the node configuration (called "NodeConfigTab") and the view configuration (called "ViewConfigTab"). The easiest way to do this is to inherit custom widget from other widgets. the MainWindow inherits from PyQt5.QWidgets.QMainWindow, NodeConfigurationTab and ViewConfigTab from PyQt5.QWidgets.QWidget, the basic widget class and one of the most basic Qt5 classes. The custom views are created programmatically and inherit from PyQtGraph.PlotWidget.

Preferences like created custom views, their state (opened or closed) and information on nodes are saved as JSON formatted file and restored by jsonpickle<sup>[20]</sup> at every startup of SensUI. The received sensor data is **not** stored by SensUI because this already happens in the feed layer, developed by LoRaLink. All data are stored as PCAP files and therefore is it possible for SensUI to not only display live data but also old data.

## 4.1 MainWindow

The MainWindow consists of a QML layout file (`MainWindow.ui`) and a controller (`MainWindow.py`). It acts as a framework for the TabWidget and menu bar. The MainWindow controller is also responsible for saving and restoring the preferences, subscribing to a feed via the LoRaLink feed layer and refreshing the views if new data is received.

## 4.2 NodeConfigTab

The NodeConfigTab uses a QML layout (`NodeConfigTab.ui`) and a controller (`NodeConfigTab.py`), too. The NodeConfigTab is used to configure nodes in the network and also save some information locally. It provides a list to select the node and input field to enter information like a name, position, measurement interval. With check boxes one can select the active sensors on this node which can be temperature, relative humidity, brightness and air pressure. The save button on the one hand saves the configuration to file and sends configuration commands to the node on the other hand. For now, only the measurement interval is sent to the nodes but it's pretty easy to further expand this feature.

## 4.3 ViewConfigTab

The ViewConfigTab also uses a QML layout (`ViewConfigTab.ui`) and a corresponding controller (`ViewConfigTab.py`). With the ViewConfigTab, custom views can be created, altered and deleted. A view has a name and can display up to two, independent Y-Axes (left and right). Each Axis can have its own measurement size, descriptor and set of sensors. Only sensors that match the selected measurement size are displayed in the sensor list and only those can be selected. It is possible to select multiple sensors for one Y-Axis.

Like for the node configuration, the view configuration is saved to file when the save button is hit. When a new custom view is created, it is not opened automatically. The view can be opened using the "Open" button.

## 4.4 ViewWidget

The ViewWidget does not have a QML layout, instead the layout is created programmatically when a custom view is opened. The ViewWidget is basically a `PyQtGraph.PlotWidget` with additional data and axis management. The ViewWidget is asked by the ViewManager to redraw when new data is received. The ViewWidget then pulls the new data on its own and refreshes the plot.

## 5 Results

The SensUI was implemented as planned. There was no serious issue, only PyQtGraph was a bit more complicated to use than expected. The Multi-Axes support is not that good but sufficient for the project. Maybe Matplotlib would have been the better option but was not tested with these requirements. Thanks to Qt5 respectively PyQt5, SensUI is running under Linux, Windows and macOS without any additional effort. The integration of the LoRaLink feed layer was also easy and working as intended. For demonstration purposes, only one node with a hard coded id was used but SensUI is designed to query the link layer for available nodes as well ("scan" the LoRaSense network). Because SensUI just subscribes to the sensor- and configuration feed in a general manner, it is also possible to use different feeds. One way to do this is to advise the feed layer to accept and forward data from other feeds. This feature is not implemented yet. The main objective, the display of sensor data in customs views, was fully achieved. In figure 1 one can see the ViewConfigTab which is used to create, alter and delete custom views. In this case, four custom views have been created, the current selected view has the right Y-axis enabled for temperature sensors.

In figure 2, a custom view with both Y-axes is shown. Each Y-axis has a dedicated measurement size and displays data from different sensors (in this case: Left axis is brightness from node 1, colored blue. Right axis is relative humidity from node 1, colored white). PyQtGraph takes care of adjusting the viewing area appropriately. To save resources, the refreshing rate is limited to one update per second but this can be changed easily. If requested, the limit can also be disabled and as soon as the feed layer forwards data to SensUI it is displayed immediately.

## 6 Conclusions

In my opinion, the project was successfully implemented. Of course there are some points that can be further improved and additional functions that can be integrated, for example the possibility to chose the colors for the plots. For one person, the scope of this project was appropriate. The collaboration with the LoRaSense and LoRaLink teams was really productive and enjoyable. In this way i also get a nice insight in their projects as well and learned something about LoRaWAN and MicroPython. The different milestones were helpful as an reference for the own progress and time management.

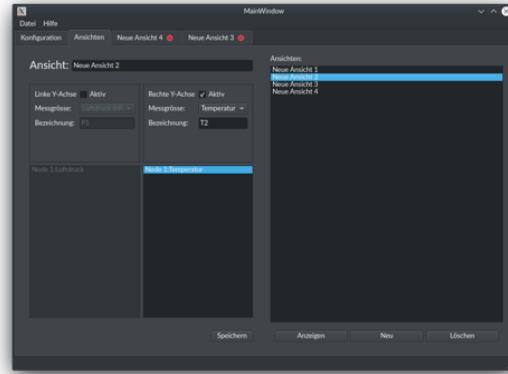


Figure 1: ViewConfigTab of SensUI



Figure 2: Custom view of SensUI

## References

- [1] Tschudin, C., *BACnet*, <https://github.com/cn-uofbasel/BACnet>, **18.07.2020**
- [2] Delley, P., Salsi, L., *LoRaSense*, <https://github.com/cn-uofbasel/BACnet/tree/master/groups/09-loraSense>, **18.07.2020**
- [3] Heckendorf-Kenzelmann, P., Trinkler, J., *LoRaLink*, <https://github.com/cn-uofbasel/BACnet/tree/master/groups/05-loraLink>, **18.07.2020**
- [4] Qt Group, *Qt Framework*, (Version 5, 2020), <https://www.qt.io/>, **18.07.2020**
- [5] Python Software Foundation, *Python* (Version 3.8, 2020), <https://www.python.org>, **18.07.2020**
- [6] LoRa Alliance, *LoRaWAN*, <https://lora-alliance.org/about-lorawan>, **18.07.2020**
- [7] Wikipedia contributors, *Pcap*, Wikipedia, The Free Encyclopedia, <https://en.wikipedia.org/w/index.php?title=Pcap&oldid=967313515>, **18.07.2020**
- [8] KDE e.V., *KDE Plasma Desktop*, (Version 5, 2020), <https://www.videolan.org>, **18.07.2020**
- [9] VideoLAN, *VLC*, (Version 3.0.9, 2020), <https://www.videolan.org/videolan>, **18.07.2020**
- [10] Oracle, *Virtualbox*, (Version 6.1, 2020), <https://www.virtualbox.org>, **18.07.2020**
- [11] Qt Group, *Qt5 Designer*, (Version 5, 2020), <https://doc.qt.io/qt-5/qtdesigner-manual.html>, **18.07.2020**
- [12] Riverbank Computing, *PyQt*, (Version 5.13, 2020), <https://riverbankcomputing.com/software/pyqt/intro>, **18.07.2020**
- [13] Qt Group, *PySide2*, <https://www.qt.io/qt-for-python>, **18.07.2020**
- [14] Python Software Foundation, *TkInter*, <https://wiki.python.org/moin/TkInter>, **18.07.2020**
- [15] Simon, C., *Atlas toolkit* (Version 0.11.1, 2020), <https://atlastk.org>, **18.07.2020**
- [16] Campagnola, L., *PyQtGraph* (Version 0.11.0, 2020), <http://www.pyqtgraph.org>, **18.07.2020**
- [17] The Matplotlib development team, *matplotlib* (Version 3.3.0, 2020), <https://matplotlib.org>, **18.07.2020**
- [18] Colclough, M., Vermeulen, G., *PyQwt* (Version 5.2.0, 2014), <http://pyqwt.sourceforge.net>, **18.07.2020**
- [19] Raybaut, P., *guiqwt* (Version 3.0.3, 2016), <https://github.com/PierreRaybaut/guiqwt>, **18.07.2020**
- [20] jsonpickle GitHub Repository, *jsonpickle* (Version 1.5), <https://github.com/jsonpickle/jsonpickle>, **18.07.2020**.

# Projekt: Introduction to Internet and Security

Gruppe 12 logSync: Carlos Tejera und Alexander Oxley

July 19, 2020

## Abstract

Unsere Aufgabe in diesem Projekt war es die Synchronisation von zwei Datenbanken (auf zwei verschiedenen Geräten) zu ermöglichen. Dies sollte mit verschiedenen Transportschichten funktionieren, in diesem Fall mit der Gruppe qrLink und Gruppe longFi. Dabei waren folgende Punkte wichtig einzuhalten:

- Keine Kopien der Datenbanken, sondern nur Anhänge der neuen Einträge
- Verifizierung der Signaturen, Feed IDs, Sequenznummern und Hash Werten
- Möglichst einfache Integration mit Applikationen der anderen Gruppen

Da jede Gruppe möglichst unabhängig voneinander arbeiten musste, haben wir unser Programm mit einer eigenen UDP Verbindung implementiert. Wir konnten die Synchronisation testen, indem wir eine Verbindung zwischen zwei Geräten im gleichen Netzwerk erstellt haben. Somit konnten wir sicher sein, dass unser Teil funktioniert und unser UDP-Programm diente den anderen Gruppen als Vorlage. Für die Integration mussten wir zusammen mit den Gruppen logStore und feedCtrl arbeiten.

Zum Schluss des Projektes konnten wir alle Punkte erreichen, die wichtig waren für eine korrekte und funktionierende Synchronisation. Die Integration mit logStore und feedCtrl konnten wir erfolgreich abschliessen. Mit der Chat Gruppe konnten wir einen funktionierenden und aktualisierenden Chat erreichen und das Gleiche galt für die Applikation der KotlinUI Gruppe.

## 1 Einführung

Mithilfe von Transportgruppen, Applikationsgruppen und logStore realisieren wir das Device Onboarding. Device Onboarding ist ein Prozess, bei dem der Netzwerzkzugriff auf das Gerät bereitgestellt wird. Das Gerät wird im Netz registriert und kann im Netzwerk teilnehmen. Im grösseren Projekt wird das mit Master Feeds gelöst, siehe FeedCtrl Gruppe. Im dezentralisierten Netzwerk werden Datenbanken synchronisiert, da die Redundanz der Daten auf allen Knoten eine der Säulen des Dezentralisierten Netzwerkes ist. Somit muss viel synchronisiert werden. In unserem Grossen Projekt ist aber die kleine Datenmenge von grosser Bedeutsamkeit, weil in späteren Prozessen mit LORA oder QR gearbeitet wird. Somit war es wichtig, dass nur die Daten, die wirklich übertragen werden müssen, übertragen werden. Durch dieses Prinzip können die Daten auf 3 Stapel abstrahiert werden: Was ich

habe, was ich brauche, und was ich schicke. Das dezentralisierte Netzwerk ist auch anfällig auf Daten, die richtige Daten nachahmen, aber eigentlich keine sind. Da der Transfer von Daten über mehrere Wege stattfinden kann, ist der Transfer selber nicht sicher. Deshalb muss man die selber Daten sichern, nicht den Transport. Wir kontrollieren die Integrität der Feeds, indem wir beim Zeitpunkt des syncen nur solche Logerweiterungen zulassen, welche drei Tests bestehen. Signatur, die Sequenznummer und der Hashpointer.

## 2 Hintergrund

Der Hintergrund unseres Projektes sind ganz klar ‘Sneakernets’ und spezifisch ‘Secure Scuttle Butt’. Sneakernet ist ein informeller Begriff für die Übertragung elektronischer Informationen durch physische Bewegung von Medien wie Magnetbändern, Disketten, optischen Datenträgern, USB-Flash-Laufwerke oder externen Festplatten zwischen Computern, anstelle diese über ein Computernetzwerk zu übertragen und deshalb sind sie dezentralisiert. Der Begriff bezieht sich auf das Gehen in Turnschuhen als Transportmechanismus.

### 2.1 Secure Scuttle Butt

‘Secure Scuttle Butt’ ist ein sicheres Messaging-System. SSB Benutzer speichern nur anhängbare (append-only) kryptografisch signierte Protokolle aller öffentlichen Nachrichten, die sie auf ihren Reisen gesehen haben. Dominic Tarr, der Entwickler von SSB, lebt selbst auf einem Segelboot und seine Benutzung von SSB bietet eine gute Analogie für die Erklärung von SSB. [1] Benannt nach einem Wasserfass (a ‘butt’ – einem ‘Kolben’), welches geschnitten (oder ‘scuttled’ – versenkt) worden war, spülten Seeleute des frühen 19. Jahrhunderts Schmutz, während sie daraus ‘zogen’. Wenn sich zwei Benutzer treffen, synchronisieren sie diese Nachrichten über ihr lokales Netzwerk oder tauschen sogar USB-Sticks mit Dateien aus. Jedes Konto in SSB ist ein ‘Tagebuch’ dessen, was eine Person digital geschrieben hat. Jedes SSB Konto besteht einfach aus zwei Dingen: einem Tagebuch und privaten / öffentlichen asymmetrischen Kryptoschlüsseln. [2] Die Authentizität von Tagebüchern bleibt erhalten, da alle Tagebucheinträge auf die zuvor geschriebene Nachricht verweisen und dann signiert werden. Dies verhindert Manipulation. Am Allerwichtigsten ist aber, dass SSB auch ‘offline’ funktioniert.

## 3 Implementation

### 3.1 Erste Schritte

Bevor wir mit der Implementation beginnen konnten, nahmen wir uns ein Beispiel an der Demo von Prof. Tschudin [3], um zu verstehen wie das Konzept der Feeds und der Keys funktionierte. Da wir noch keine Datenbanken zur Verfügung hatten, haben wir uns entschieden unser Programm mit PCAP files (wie in der Demo) zu implementieren. Dabei bestand unser erster Prototyp aus einer Synchronisation lokaler Directories mit verschiedenen selber generierten PCAP files, jedoch noch ohne Verifizierung. Als dies erfolgreich funktionierte, gingen wir zum nächsten Schritt, nämlich die Synchronisation zweier Directories auf zwei

verschiedenen Geräten. Wir entschieden uns eine UDP Verbindung zu erstellen, um die Synchronisation zu vollführen. Dabei mussten wir uns ein effizientes Protokoll ausdenken, damit dies funktionieren würde und für die anderen Gruppen verwendbar wäre.

## 3.2 Protokoll

	Device A		Device B
1	- Creates "I HAVE"-list		
2	- Sends "I HAVE"-list	-	
		-	
3		-	- Receives "I HAVE"-list
4			- Compares list with own entries
5			- Creates "I WANT"-list
6		-	- Sends "I WANT"-list
		-	
7	- Receives "I WANT"-list	-	
8	- Filters events		
9	- Creates EVENT-list	-	
		-	
10		-	- Receives EVENT-list
11			- Synchronisation

Figure 1: Kommunikation zwischen zwei Geräten

Wir entschieden uns drei Listen zu erstellen:

- ”I HAVE”-list: Beinhaltet alle Feed IDs und deren letzten Sequenznummer (Gerät A).
- ”I WANT”-list: Beinhaltet ein Vergleich der ”I HAVE”-list von Gerät A mit der ”I HAVE”-list von Gerät B und dabei werden die Sequenznummern verglichen. Stimmen diese nicht überein, wird eine ”I WANT”-list auf Gerät B erstellt. Diese beinhaltet die Feed IDs, die ein Update brauchen und ab welcher Sequenznummer die neuen Einträge benötigt werden.
- Event-list: Beinhaltet alle Events aller Feed IDs (ab der gegebenen Sequenznummer) von der ”I WANT”-list.

## 3.3 Integration

Als auch der zweite Prototyp fertig war, haben wir uns bereits mit den Gruppen logStore und feedCtrl ausgetauscht. Wir mussten unser Programm so ändern, dass wir statt mit PCAP files, Datenbanken verwenden konnten.

### 3.4 Verifizierung

Da wir unabhängig von den anderen arbeiten konnten, entschieden wir uns die Verifizierung am Schluss hinzuzufügen. Dabei wurden folgende Punkte verifiziert:

- Feed ID: Stimmen die Feed IDs überein?
- Signatur: Um welche Art von Verschlüsselung handelt es sich?
- Sequenznummer: Ist dies die korrekte Fortsetzung des letzten Events?
- Hash Wert: Handelt es sich um keine Fake-Fortsetzung des letzten Event und stimmt die Verschlüsselung überein?

Schliesslich wurde auch dieser Teil in unserem Code hinzugefügt.

## 4 Resultate

Wir sind nach Oben und Unten stark integriert, um Datenbanken zu synchronisieren brauchen wir FeedCtrl, eine Applikationsgruppe, logStore und eine Transport Gruppe.

Wir sind mit den anderen Gruppen der Mittelschicht, namentlich Gruppe 7 logStore und Gruppe 14 FeedCtrl, stark integriert. Gruppe 7 logStore bietet Applikationsgruppen Datenbanken an, welche wir dann synchronisieren. Diese Integration funktionierte sehr gut, wir konnten ihre Datenbankmethoden in unserem PCAP Code implementieren. Von der Gruppe 14 FeedCtrl haben wir ihre Implementation mit Master Feeds übernehmen müssen. Erst wenn man Feed IDs in den Master Feed eingefügt hat, kann man Events richtig in die Datenbanken einfügen. Wegen der Implementation von Gruppe 14 muss man auch zuerst Master Feeds synchronisieren und dann bei einer zweiten separaten Synchronisation kann man die Event Feeds synchronisieren. Dies muss aber nur beim ersten Treffen von zwei Geräten passieren.

Wir haben mit der Subjective Chat Gruppe 3 zusammengearbeitet. Nach Oben zu den Applikationen sind wir sofern integriert, weil die Applikationen Datenbanken mithilfe von Gruppe 7 logStore kreieren und erweitern. Wir synchronisieren diese Datenbanken dann, und die Datenbanken können dann auf anderen Geräten mit den gleichen Applikationen gelesen werden.

Wir haben mit Gruppe 6 longFi Gruppe und Gruppe 2 QRLink zusammengearbeitet. Mit der longFi Gruppe hat es aus verschiedenen Gründen leider nicht ganz geklappt (Verbindungsprobleme, fehlende OS-Kompatibilität). Wir arbeiteten zusammen bis zur Abgabe daran.

Mit der QRLink Gruppe ging es viel besser. Sie haben unsere Methoden benutzt, um Datenbanken mittels vielen QR Bildern zu schicken. Dies wurde in der Präsentation sehr gut mit einem beschleunigten Video gezeigt.

## 5 Fazit

Wir hatten am Anfang Probleme das Projekt als Ganzes zu verstehen. Die Komplexität kam daher, dass alles sehr theoretisch erklärt wurde. Es war daher schwer die Erwartungen

und Voraussetzungen herauszulesen. Die einzelnen Teilprojekte alleine zu verstehen war etwas kompliziert, aber machte für uns Sinn, als wir es als ganzes Projekt betrachtet haben. Nichtsdestotrotz haben wir aus dem vorigen Grund mehrere Termine mit Prof. Tschudin vereinbart, um sicher zu gehen, dass wir unsere Aufgabe richtig verstanden haben. Es stellte sich heraus, dass die inhärente Komplexität unserer einzelnen Teilprojekte gar nicht so gross war.

Die grössten Probleme kamen in der Integrationsphase, wo es erwartet war, unsere Teilprojekte in das Grosse zu integrieren. Wir wissen jetzt, dass Integration und Kollaboration viel einfacher gehen, wenn man persönlich zusammensitzen kann und die Arbeit ausfeilen kann. Die Integration über Zoom und Discord lieferte viel Lerneffekt.

Ebenfalls war es schwierig gegenseitige Voraussetzungen und Erwartungen festzusetzen, wenn man erst gegen Schluss wusste wie sein eigenes Projekt richtig aussehen würde. Wenn man dieses Projekt einfacher / direkter machen wollen würde, sollte man mehr spezifische Requierements und Erwartungen festsetzen. Diese Schwierigkeit ist aber sicher erwartet und vorgeplant.

Trotz der schwierigen Phasen konnten wir unser Projekt erfolgreich abschliessen.

## References

- [1] *Scuttlebutt: an "off-grid" P2P social network that runs without servers and can fall back to sneakernet*, available at  
<https://boingboing.net/2017/04/07/bug-in-tech-for-antipreppers.html>.
- [2] *SCUTTLEBUTT, A DECENTRALIZED SOCIAL PLATFORM*, available at  
<https://www.inthemesh.com/archive/secure-scuttlebutt-facebook-alternative/>.
- [3] *Demo von Professor Tschudin*, available at  
<https://github.com/cn-uofbasel/BACnet/tree/master/src/demo>

Introduction to Internet and Security

Frühjahrssemester 2020

Projektarbeit

# 14-Feed Control

*Wilson A. Eghonghon, Yannick A. Rümmele, Damian Knuchel*

19. Juli 2020

## Inhaltsverzeichnis

<b>1. Abstract</b>	1
<b>2. Einleitung</b>	1
<b>3. Master-Feed</b>	1
3.1. Events . . . . .	2
<b>4. Onboarding</b>	2
4.1. Kennenlernen . . . . .	2
<b>5. Sozialer Radius</b>	3
<b>6. Feed Control</b>	3
6.1. Verification . . . . .	3
6.2. UI . . . . .	4
<b>7. Fazit</b>	5
<b>A. Package 07-14-LogCtrl</b>	6
<b>B. Honourable Mentions</b>	7

## 1. Abstract

Ziel dieses Projektes ist, das Entwickeln eines Systems, welches die Probleme des Onboardings löst und das Verschicken und Empfangen von Daten kontrolliert (FeedControl). Da es sich beim BACnet um ein unabhängiges Netzwerk handelt, kann nicht, wie im Internet, mit Hilfe von Port und IP-Adresse mit einem anderen Nutzer in Verbindung treten. Mit einem neuen Ansatz für das Onboarding konnte die Gruppe FeedCtrl dieses Problem erfolgreich lösen. Mit Hilfe von Filterfunktionen kann nach dem Erstkontakt entschieden werden, welche Daten von einem Nutzer empfangen werden sollen. In Zusammenarbeit mit der Gruppe LogStore konnte das Onboarding und das Filtern von Paketen entsprechend unseren Anforderungen im Umfang von diesem Projekt implementiert werden.

## 2. Einleitung

Beim BACnet handelt es sich um ein dezentralisiertes Netzwerk, welches ohne IP-Adresse die Kommunikation von Teilnehmern ermöglicht. Die verschiedenen Feeds von den Nutzern sind anhand von einem öffentlichen Schlüssel unterscheidbar. Anhand von diesem öffentlichen Schlüssel müssen wir nun entscheiden, mit wem wir kommunizieren, welche Daten empfangen werden und welche wir dann auch wieder weiter versenden wollen.

Da jede Applikation seinen eigenen Feed erstellt, können wir deren öffentlichen Schlüssel als Port betrachten. Wenn wir den Schlüssel in einer Art von Forwarding Tabelle abspeichern, können wir später feststellen, ob wir diese in die von Gruppe LogStore zu Verfügung gestellten Datenbank abspeichern wollen.

Ein Port alleine reicht aber noch nicht zum Verbindungsaubau. Was jetzt noch fehlte ist eine Alternative zu der IP-Adresse. Hierfür hat die Gruppe FeedCtrl den master-feed entwickelt,

welcher alle Applikations-feeds von einem Nutzer einem master-feed zuordnet. Innerhalb von einem master-feed werden nur Informationen über den Nutzer selbst übertragen.

## 3. Master-Feed

Der master-feed ist eine Methode um das Problem, dass jede Applikation seinen eigenen Feed erstellt zu lösen. Für andere Nutzer wäre es nicht erkennbar, welche Feeds zu einer bestimmten Person gehören.

Innerhalb von einem master-feed wird abgespeichert welche Applikations-feeds er besitzt. In Zusammenarbeit mit der Gruppe LogStore implementierten wir ein System, welches beim Erstellen eines neuen Applikations-feeds direkt einen Event in den master-feed erzeugt, welches den Applikations-feed repräsentiert. Die Gruppe LogMerge ergänzte ihr EventCreationTool<sup>1</sup> so, dass beim Erstellen eines neuen Applikations-feeds als erster Eintrag der master-feed repräsentiert wird. Diese zweiseitige Absicherung dient dazu, dass kein anderer master-feed einen Applikations-feed beanspruchen kann.

Somit hätten wir nun das Problem vom Erkennen eines Nutzers gelöst. Während ein Applikations-feed mit einem Port verglichen werden kann, kann nun ein master-feed mit einer IP-Adresse verglichen werden. Die Kombination aus Applikationsfeed und master-feed ist eindeutig.

---

<sup>1</sup>Das EventCreationTool ist ein von LogMerge entwickeltes Tool, welches das Erstellen von einem neuen Feeds und das Hinzufügen von neuen Events in einfacher Form ermöglicht.

## 4. Onboarding

### 3.1. Events

Es können nur ganz bestimmte Events innerhalb von einem master-feed abgespeichert werden, ansonsten wird dieser von der Datenbank als ungültig angesehen.

- MASTER/MASTER: Identifiziert einen Feed als master-feed
- MASTER/Trust: Vertraue einem Feed von einem anderen Nutzer
- MASTER/Name: Setzt den Nutzernamen vom Besitzer
- MASTER/NewFeed: Hinzufügen eines neuen Feeds
- MASTER/Block: Blockiert einen Feed von einem anderen Nutzer
- MASTER/Radius: Setzt den sozialen Radius

Der Grundaufbau vom master-feed muss immer identisch sein. Als ersten Event enthält er MASTER/MASTER, so dass die Datenbank ihn als solchen identifizieren kann. Der zweite Event ist MASTER/Radius, welches den sozialen Radius bestimmt. Der dritte Event ist MASTER/Name um dem Nutzer einen Nutzernamen zuzuteilen. Da wir am Anfang keine anderen Nutzer kennen, kann derselbe Nutzername von mehreren Nutzern gleichzeitig verwendet werden. Dies führt jedoch zu keinen Problemen, da die Differenzierung anhand der Feeds geschieht und nicht anhand den Nutzernamen. Nachdem der Grundriss steht können die anderen Events willkürlich folgen. Die Datenbank speichert immer den aktuellen Zustand in einer speziell für den master-feed erstellten Tabelle ab.

## 4. Onboarding

Für das Onboarding war Damian Knuchel zuständig. Beim Onboarding handelt es sich um

das Kennenlernen mit anderen Nutzern und den ersten Verbindungsaubau zu denen. Die Art der Verbindung ist dabei irrelevant. Relevant ist nur, dass beim Kontakt die master-feeds ausgetauscht werden. Mit dem Austausch von den master-feeds lernt man neue Nutzer im BACnet kennen und kann sich entscheiden, mit welchen von ihnen man kommunizieren will.

### 4.1. Kennenlernen

In einem früheren Abschnitt bezeichneten wir den master-feed als eine Art von IP-Adresse, also einen Identifikator für einen Nutzer. Um anderen Teilnehmern unsere Existenz im BACnet mitzuteilen muss man ihnen den eigenen master-feed zustellen. Auf Ebene der Netzwerkschicht, in diesem Projekt die Gruppe LogMerge und Log-Sync, wird eine Anfrage an die Datenbank für gewisse Applikations-feeds gestellt, welche sie in die Datenbank abspeichern oder über die Transportschicht weiter versenden wollen. Beim Importieren bekommen alle master-feeds automatisch grünes Licht und werden in die Datenbank übertragen.

Beim Exportieren aus der Datenbank werden automatisch alle master-feeds, welche hinterlegt sind, mitgereicht. Das Ganze gleicht einem greedy-Verfahren, in welchem man versucht, so viele Nutzer wie möglich kennen zu lernen, und anderen Nutzern alle dir bekannten Teilnehmer mitzuteilen. Langsam, aber sicher wird eine Datenbank erstellt, welche Informationen zu allen Nutzern im BACnet beinhaltet.

In grösseren Dimensionen wäre dieser Ansatz nicht umsetzbar. Speicher ist eine wertvolle Ressource. Auch wenn ein einzelner master-feed keine grosse Kapazität benötigt, würden die Daten von Millionen von Nutzern auf den eigenen Rechner zu einer Überflutung des Speichers führen.

## 5. Sozialer Radius

Für das Entwickeln vom sozialen Radius war Wilson A. Eghonghon zuständig. Beim sozialen Radius handelt es sich um ein System, dass das Empfangen von Daten ausserhalb der eigenen Freundesliste, aber innerhalb von einem bestimmten Radius, ermöglicht. Bei diesem Radius handelt es sich nicht um eine physische Distanzangabe wie zum Beispiel in Metern. Es geht darum, mit Personen in Kontakt zu kommen, welche Freunde der eigenen Freunde sind. Um zu bestimmen, welche Nutzer sich innerhalb von einem gegebenen Radius befinden verwenden wir zwei Werte: Wem genau vertraut ein Nutzer aktuell und um welche Applikation handelt es sich. Zur Wiederholung, mit einem Trust-Event im eigenen master-feed vertrauen einem Applikations-feed von einem Nutzer. Ein Nutzer, dem wir vertrauen, ist eigentlich nichts anderes als ein Applikations-feed, welchen wir auf ‘trusted’ gesetzt haben.

Der Ablauf ist wie folgt: Als erstes generieren wir eine Liste der Applikationen/Feeds, welchen ein Nutzer vertraut. Danach wird über diese Menge von Feeds iteriert. Bei jedem Iterationsdurchlauf wird der master-feed von jenem Feed abgefragt. Anhand von den Daten vom gefundenen master-feed überprüfen wir erneut, ob jener einem Feed vertraut, welcher von derselben Applikation stammt, wie der Feed dem der ursprüngliche Nutzer vertraut. Von nun an wiederholt sich dieser Prozess bis zu einer Iterationstiefe, welcher dem aktuell gesetzten Radius entspricht.

Dieses Feature war vor allem für die SubChat Gruppe von Interesse. Innerhalb von ihrer Chat Applikation bieten sie die Funktionalität einen Gruppenchat zu erstellen. Wenn sich innerhalb von diesem Chat auch Nutzer befinden, welchen man nicht folgt, kann man mit Erhöhen vom sozialen Radius deren SubChat-feeds empfangen. Die SubChat Applikation kann diese Daten wie alle anderen Feeds aus der Datenbank auslesen und darstellen.

## 6. Feed Control

Für jeden öffentlichen Schlüssel auf der ‘Need’-Liste<sup>2</sup> wird überprüft ob wir diese Daten, anhand der aktuellen Konfiguration des masterfeeds wirklich übergeben wollen. Dasselbe gilt für das Einlesen von Feeds. Die Netzwerkschicht fragt für jeden Feed an, ob dieser an die Datenbank übergeben werden soll. Um das greedy-Verfahren vom Sammeln von masterfeeds aufrecht zu erhalten, werden diese alle akzeptiert.

### 6.1. Verification

Die Verification-Klasse **6** dient als Schnittstelle für die Netzwerkschichtgruppen LogMerge und LogSync. Bei Verwendung verhindert sie das Importieren und Exportieren von ungewollten Daten.

Die Methode ‘check\_incoming(feed\_id, app\_name)’ überprüft anhand des öffentlichen Schlüssels und dem Namen der Applikation des zu importierenden Feeds, ob dieser importiert werden soll. Der Applikationsname ist nur relevant, falls man einen Radius grösser als eins verwendet.

Die Methode ‘check\_outgoing(feed\_id)’ überprüft anhand des öffentlichen Schlüssels des zu exportierenden Feeds, ob dieser wirklich exportiert werden soll. Dies ist relevant, falls man bestimmte Feeds nicht mit anderen Nutzern teilen will.

Mit der für FeedCtrl erstellten Schnittstelle von LogStore kann die Verification-Klasse direkt auf die relevanten Daten zugreifen und überprüfen, welchen Feeds vertraut wird und welchen nicht. Die Datenbank filtert und erstellt automatisch Listen mit allen angefragten Daten. So wird beim Überprüfen, ob ein Feed auf der Trusted-Liste steht eine Anfrage an die Datenbank gestellt, welche eine

---

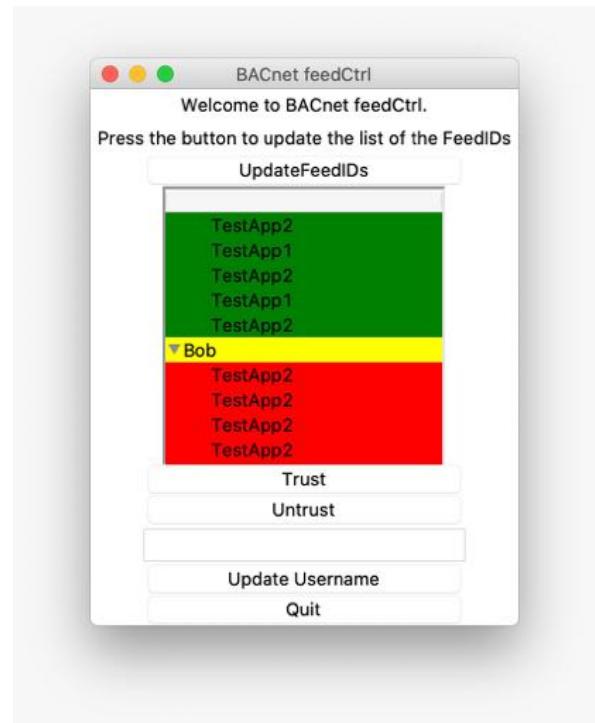
<sup>2</sup>Die ‘Need’-Liste ist ein von anderen Gruppen implementiertes System, welches als Anfrage für gewisse Feeds an eine andere Datenbank gilt.

Liste von Trusted-Feeds zurückgibt. Anhand von dieser kann nun festgestellt werden, ob ein Feed den Check besteht oder nicht.

## 6.2. UI

Die UI wurde von Yannick A. Rümmele unter Verwendung von Tkinter entworfen. Die UI wird verwendet, um den Feed Controller zu konfigurieren. Alle Konfigurationen, ausgenommen des Radius, werden innerhalb vom master-feed abgespeichert. Es gibt die Möglichkeit seinen eigenen Nutzernamen abzuändern und den Radius zu aktualisieren. In einer Tabelle werden die master-feeds von anderen Nutzern aufgelistet. Als Erstes wird der master-feed von den jeweiligen Nutzern mit dessen Nutzernname und dessen öffentlichen Schlüssel dargestellt. In einem Tree werden darunter die zum Nutzer zugehörigen Applikations-feeds dargestellt. Diese können ausgewählt werden und es kann entschieden werden, ob ihm vertraut wird oder nicht. Auf diese Weise kann ein Nutzer selbst konfigurieren, welche Feeds akzeptiert werden sollen. Änderungen werden im Tree mit einem Klick auf den Button 'Update FeedIDs' angezeigt. Die Änderungen in der Datenbank erfolgen aber bereits beim Klicken auf den jeweiligen Button.

Für Geräte, welche keine Tkinter Nutzeroberflächen darstellen können, gibt es auch eine Schnittstelle über die Entwickler auf alle Funktionalitäten von der UI zugreifen können. In Zusammenarbeit mit der KotlinUI Gruppe konnte auf einem Android Device Feed Control ausgeführt werden. Die UI wurde komplett von der KotlinUI Gruppe entworfen.



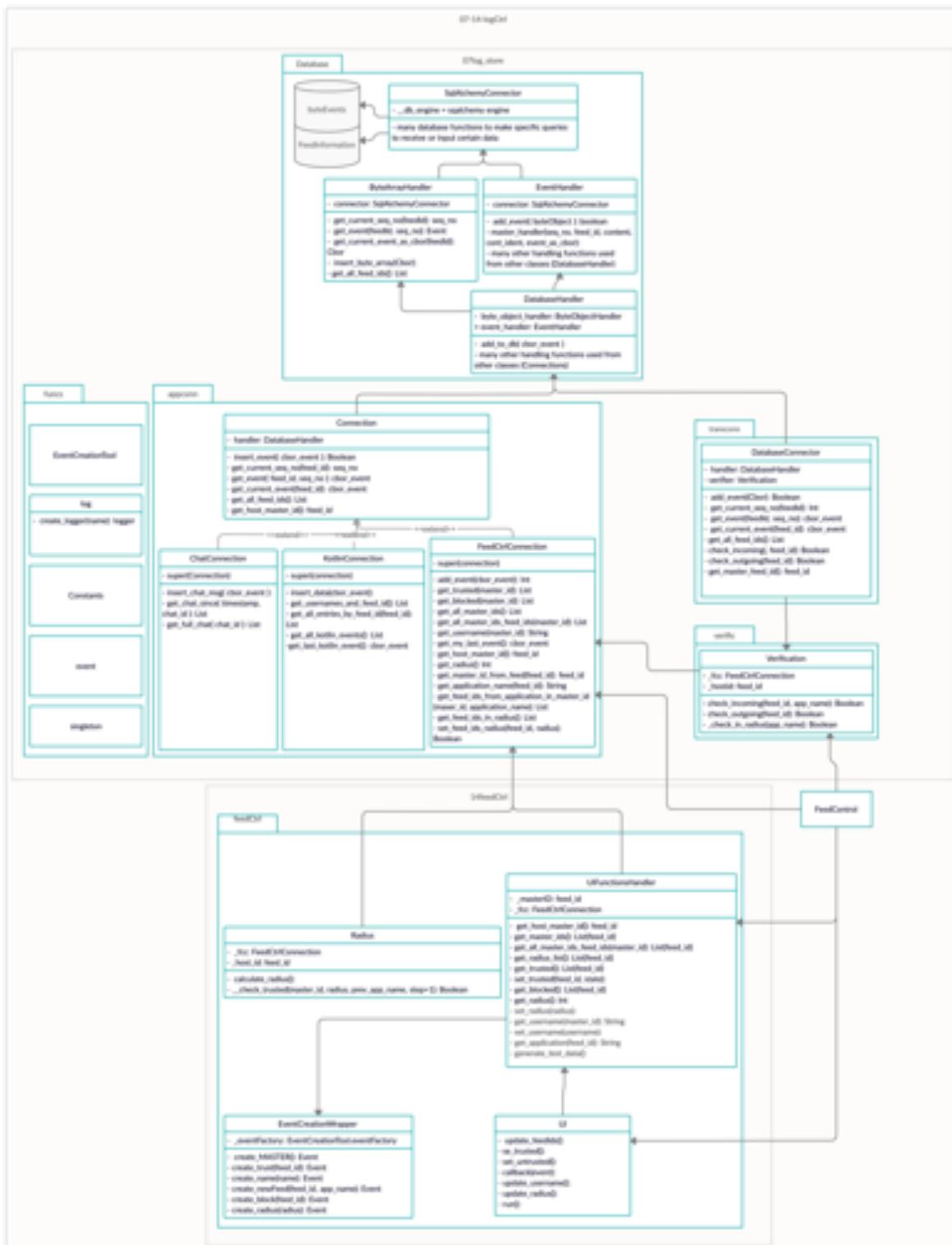
**Abb. 1:** Die UI mit dem master-feed von zwei Nutzern. Gelb: master-feed, Grün: Trusted, Rot: Untrusted  
Unterhalb von einem master-feed werden die dazugehörigen Applikations-feeds dargestellt.

## **7. Fazit**

Zusammenfassend lässt sich sagen, dass Feed Control den zu Beginn des Projekts festgelegten Anforderungen der Gruppe entspricht. Durch den master-feed kann man neue Nutzer innerhalb vom BACnet kennenlernen und mit Hilfe der UI dann einstellen, ob man mit ihnen kommunizieren möchte. Der Filter verhindert erfolgreich das Überfluten der Datenbank mit ungewollten Daten. In Zusammenarbeit mit allen dreien Demogruppen konnte demonstriert werden, dass Feed Control seine Funktion erfüllt.

## A. Package 07-14-LogCtrl

### A. Package 07-14-LogCtrl



## *B. Honourable Mentions*

Das UML-Diagramm stellt den Klassenaufbau und Zusammenhang vom Packet LogCtrl dar. Es wurde so gut wie möglich versucht die Klassen von den beiden Gruppen getrennt zu halten. Schluss und endlich kommt es aber vor, das Code von einer Gruppe auch innerhalb von einem Subpacket von einer anderen Gruppe auftaucht. Z.B. Verification-Klasse, welche sich innerhalb von log\_store befinden ist eigentlich Bestandteil vom Projekt von Gruppe feedCtrl.

## **B. Honourable Mentions**

An dieser Stelle bedanken wir uns bei anderen Gruppen und Personen, welche beim Entwickeln von Feed Control aktiv dazu beigetragen haben.

- Nikodem Kernbach: Zusammen mit Nikodem ist während stundenlangem Überlegen das Konzept des master-feeds entstanden. Dieses Konzept ermöglichte erst die Adressierung von Nutzern im BACNet.
- Gruppe 07-LogStore (Viktor Gsteiger, Moritz Würth): Die Zusammenarbeit mit der Gruppe 7 war super. Das Ganze lief professionell und ohne Probleme ab. Innerst kürzester Zeit haben sie den master-feed zu ihrer Datenbank hinzugefügt und uns eine Schnittstelle bieten können.