Flurfunk – ein LoRa-basierter Hausflohmarkt

Bericht zum Projekt für «Distributed Programming and Internet Architecture»

Universität Basel, Sommersemester 2025

vorgelegt von Andrea Seehuber, 05.07.2025

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Digitale Kommunikation ist heutzutage fast immer auf zentrale Infrastrukturen angewiesen: Server, Mobilfunkmasten, Internetprovider. In Krisensituationen, abgelegenen Regionen oder autoritär regierten Staaten kann diese Abhängigkeit jedoch problematisch werden. Zentrale Server lassen sich zensieren, kontrollieren oder schlicht abschalten. Die permanente Internetverbindung ist zudem energieintensiv und schliesst Menschen ohne Zugang aktiv aus. Dezentrale Netzwerke bieten hier eine alternative Kommunikationsform: Sie funktionieren ohne zentrale Steuerung, sind robuster gegen Ausfälle und ermöglichen einen direkten, selbstbestimmten Austausch von Informationen. Projekte wie «Meshtastic» zeigen, wie durch den Einsatz von LoRa-Funkmodulen einfache Peer-to-Peer-Netzwerke aufgebaut werden können, die Nachrichten energieeffizient über mehrere hundert Meter oder sogar Kilometer übertragen – ganz ohne Mobilfunk oder WLAN (Meshtastic LLC, unbekannt).

LoRa (Long Rang) ist ein proprietäres Modulationsverfahren auf Basis von Chirp Spread (CSS), das auf lizenzfreien ISM-Bändern (z. B. 868 MHz in Europa) arbeitet. Es ermöglicht extrem stromsparende Kommunikation bei sehr geringer Bandbreite, typischerweise im Bereich von wenigen hundert Bit pro Sekunde. Dadurch eignet es sich ideal für batteriebetriebene Geräte und einfache Datenübertragung über grosse Entfernungen bei gleichzeitig hoher Störfestigkeit (Union schweizerischer Kurzwellen-Amateure, 2024).

1.2 Zielsetzung

Das Projekt «Flurfunk» setzt genau auf diese Idee: Eine App, die Bewohner eines Hauses über ein LoRa-basiertes Mesh-Netzwerk miteinander verbindet, um gebrauchte Gegenstände auszutauschen. Dabei erfolgt sämtliche Datenübertragung ohne Anbindung an das Internet und lokal. Beim ersten Start der App gibt der Benutzer seine Adressdaten ein, die zur eindeutigen Identifizierung im Netzwerk dienen. Dann erlaubt die Benutzeroberfläche die Erstellung neuer Angebote in verschiedenen Kategorien. Die eigenen Angebote können deaktiviert werden, falls sich eine andere Person dafür interessiert und auch wieder reaktiviert werden. Im deaktivierten Zustand werden sie auf dem GUI der Netz-Peers nicht mehr angezeigt. Angebote können vom eigenen GUI auch komplett ausgeblendet werden, wenn sie physisch nicht mehr vorhanden sind. Alle Daten werden periodisch und verschlüsselt innerhalb des eigenen Netzwerks per Broadcast übertragen.

2 Umsetzung des Projekts

2.1 Rahmenbedingungen

Für die Kommunikation über das LoRa-Mesh kamen zwei Waveshare SX1261 LoRa USB-Adapter (850 – 920 MHz) zum Einsatz (Digitec Galaxus AG, 2025). Diese Geräte bieten eine einfache Möglichkeit, LoRa-Kommunikation per USB-UART-Schnittstelle mit einem beliebigen Host-System aufzubauen. Sie unterstützen einen Stream- und einen Packet-Modus und lassen sich über

AT-Befehle steuern. Um eine zuverlässige Kommunikation im europäischen ISM-Band zu gewährleisten, wurde auf den Geräten vor Inbetriebnahme nach Anleitung auf der Website des Herstellers ein Firmware-Update durchgeführt (Waveshare, unbekannt). Die Konfiguration erfolgte zunächst über die Android-App «Serial USB Terminal» und ein Austausch von Nachrichten konnte erfolgreich getestet werden.

Die beiden USB-Dongles wurden jeweils über einen USB-OTG-Adapter direkt mit zwei Android-Smartphones verbunden:

- Sony Xperia 5 II (Android 12)
- Samsung Galaxy A16 (Android 15)

Beide Geräte unterstützen USB-OTG und ermöglichen so eine direkte serielle Kommunikation zwischen App und LoRa-Dongle. Der Zugriff auf die USB-Schnittstelle erfolgt über das Android-USB-Host-API in Verbindung mit einer eigenen USB-Abstraktionsschicht innerhalb der App.

Die App wurde in Java mit Android Studio entwickelt. Dabei kamen die offiziellen Android-Entwicklungswerkzeuge sowie das USB-Host-API und AndroidX-Bibliotheken zum Einsatz. Für die serielle Kommunikation mit dem LoRa-Dongle wurde die Open-Source-Bibliothek «usb-serial-for-android» verwendet (Kai Morich, 2025). Die App ist modular aufgebaut und unterscheidet klar zwischen der hardwarenahen Steuerung des LoRa-Dongles, dem Netzwerkprotokoll und der verteilten Datenhaltung.

2.2 Implementierung

2.2.1 Hardwarenahe Kommunikation

Die App kommuniziert direkt mit dem angeschlossenen LoRa-Dongle über die serielle USB-Schnittstelle. Die Umsetzung orientiert sich dabei an der Funktionsweise der Android-App «USB Serial Terminal» und ist zweistufig aufgebaut (Kai Morich, 2024):

- Die Klasse `LoRaService` ist ein persistenter Android-Service, der von der Netzwerkkomponente `ATLoRaManager` gebunden werden kann. Er kapselt die gesamte Lebensdauer der Verbindung zum Dongle und delegiert die eigentliche I/O-Kommunikation an ein internes `LoRaSocket`-Objekt. Die Methoden `connect`, `write` und `disconnect` dienen als API für die Anwendung.
- Die Klasse `LoRaSocket` verwaltet die konkrete serielle Verbindung. Sie öffnet den USB-Port, konfiguriert ihn mit Standardparametern und startet dann einen Hintergrundthread über die Klasse `SerialInputOutputManager`, der kontinuierlich auf eingehende Daten wartet. Diese werden bei Empfang an eine `LoRaListener`-Instanz weitergegeben, die als Callback-Schnittstelle dient. Falls der Lese-Thread unerwartet abbricht, versucht `LoRaSocket` bis zu drei Wiederverbindungen, bevor die Verbindung endgültig aufgegeben wird. Dadurch bleibt die App auch bei instabiler Verbindung funktionsfähig.

Der verwendete LoRa-Dongle übernimmt intern die Fragmentierung und Pufferung von Datenpaketen. Nutzdaten bis 960 Byte werden automatisch in Pakete à 240 Byte aufgeteilt und eine zuverlässige Übertragung der Datenpakete wird durch eine integrierte CRC-Prüfung sichergestellt. Fehlerhafte Pakete werden vom Dongle automatisch verworfen (Waveshare, unbekannt).

2.2.2 Nachrichtenrouting und Netzwerkprotokoll

Die Klasse `ATLoRaManager` bildet die Schnittstelle zwischen der LoRa-Hardware und dem Anwendungscode. Nach dem Start übernimmt sie die Initialisierung des Dongles per AT-Befehlen (z. B. AT+MODE=1 für den Stream-Modus) und setzt alle relevanten Übertragungsparameter. Da serielle Daten über USB nicht paketorientiert, sondern in beliebig grossen Chunks eintreffen, ist ein robuster Parser erforderlich. Hierfür wird ein interner `SerialFrameReader` eingesetzt, der alle eingehenden Bytes puffert und nach einem definierten Nachrichtenrahmen durchsucht: Gemäss Netzwerkprotokoll beginnt eine Nachricht mit einem `#` und endet mit der Zeichenkette `;EOM`. Um sicherzustellen, dass der Empfangspuffer korrekt mit Chunks arbeitet und keine Überläufe oder Fehlinterpretationen auftreten, wird die Parserlogik thread-sicher umgesetzt. Für den ausgehenden Nachrichtenfluss wird eine Sende-Queue verwendet, die in einem dedizierten Thread verarbeitet wird. Um Kollisionen zu vermeiden, wird vor jedem Sendevorgang überprüft, ob aktuell ein Empfang stattfindet, und ein Mindestabstand zwischen zwei Übertragungen erzwungen.

Vollständig rekonstruierte Nachrichten werden über eine Callback-Schnittstellt im Stil eines `Consumer<String>` an die registrierte Anwendungsschicht weitergereicht. In der `MainActivity` der App wird hierfür der `MessageDispatcher` registriert. Der Dispatcher übernimmt daraufhin folgende Aufgaben:

- **Protokoll-Parsing.** Die Nachricht wird mithilfe der Klasse `Protocol` analysiert und in ihre Bestandteile zerlegt.
- **Mesh-ID-Filterung.** Nur Nachrichten mit passender `MID` (Mesh ID) werden weiterverarbeitet. Die `MID` wird beim Erstellen des Benutzerprofils aus der eingegebenen Adresse erzeugt.
- **Duplikatprüfung.** Bereits verarbeitete Nachrichten-IDs werden zwischengespeichert und ignoriert, um redundante Verarbeitung zu verhindern.
- **Befehlsbasiertes Routing.** Je nach Protokollkomando (`SYNOF`, `REQOF`, `PRDAT`, etc.) wird die Nachricht an eine zuständige Komponente delegiert:
    - `OfferSyncManager` für Angebotsdaten und
    - `PeerSyncManager` für Benutzerprofile.

Dieses Routingmodell ist modular aufgebaut und erlaubt eine einfache Erweiterung des Protokolls um neue Nachrichtentypen.

2.2.3 Synchronisationslogik

Die App basiert auf einem vollständig dezentralen, rein broadcast-basierten Synchronisationsmechanismus. Um den Zustand aller Angebote und Benutzerprofile innerhalb eines gemeinsamen LoRa-Meshs abzugleichen, kommen zwei Synchronisationsprozesse zum Einsatz, die in der `MainActivity` alle 90 Sekunden und mit einem zeitlich versetzten Abstand von 5 Sekunden aufgerufen werden:

- Peer-Synchronisation zum Abgleich und Import von Benutzerprofilen – `SYNPR`
- Angebots-Synchronisation zum Abgleich und Import von Flohmarktangeboten – `SYNOF`

Die Klasse `PeerSyncManager` verwaltet alle Vorgänge zum Austausch von Benutzerprofilen und basiert auf drei Protokollnachrichten:

- `SYNPR` enthält eine Liste aller bekannten Peer-IDs samt Änderungszeitpunkten.
- `REQPR` fordert anhand dieser Liste veraltete oder fehlende Profile gezielt an.
- `PRDAT` überträgt vollständige Profilinformationen verschlüsselt.

Die Klasse `OfferSyncManager` ist für den Austausch und Import von Angebotsdaten zuständig. Sie arbeitet nach einem ähnlichen Mechanismus:

- `SYNOF` enthält eine Liste der bekannten Angebote samt Änderungszeitpunkten.
- `REQOF` fordert anhand dieser Liste gezielt veraltete oder fehlende Angebote an.
- `OFDAT` überträgt die vollständigen Angebotsdaten verschlüsselt.

Alle empfangenen Daten werden lokal gespeichert, mit bestehenden Einträgen abgeglichen und aktualisiert, wenn der übertragene Zeitstempel höher ist als der vorhandene.

Sowohl Benutzer- als auch Angebotsdaten werden als JSONObjects verschickt. Eine Angebots-Objekt hat z. B. folgendes Format:

```
{"category":"ELECTRONICS",
"createdAt":1751717068181,
"creatorId":"b40d774ae4dea2df",
"deleted":false,
"description":"Samsung Galaxy A16, nicht 5G fähig",
"lastModified":1751717068181,
"offerId":"c0e4d418134fca5c",
"status":"ACTIVE",
"title":"Smartphone"}
```

Jedem Attribut des Objekts ist ein eindeutiger Protokollschlüssel zugeordnet, mit dessen Hilfe die Informationen für die Darstellung auf dem GUI extrahiert werden können.

Die maximale Grösse einer LoRa-Nachricht ist durch die Hardware auf 960 Bytes limitiert. Um grössere Datenmengen trotzdem senden zu können, werden sowohl Peer- als auch Angebotsdaten beim Versand in mehrere Blöcke (Chunks) aufgeteilt. Dies geschieht explizit im Code durch manuelle Aufteilung der Daten in JSONArray-Fragmente.

2.2.4 Konsistenzmodell und CRDT-Kompatibilität

Die Synchronisation der Daten erfolgt in der App nach dem Prinzip eines asynchronen, eventual consistent Systems, bei dem jeder Peer jederzeit aktualisierte Informationen senden oder empfangen kann. Dieses Modell orientiert sich an der Idee von CRDTs, wobei in diesem Projekt zwei relevante CRDT-Typen zur Anwendung kommen (Shapiro, Preguiça, Baquero, & Zawirski, 2011):

- **Das Benutzerprofil wird als Last-Write-Wins Register (LWW) angelegt.** Jedem Benutzer wird einmal eine eindeutig identifizierbare `UID` zugewiesen sowie ein Zeitstempel `TS`, der bei Änderungen am Profil aktualisiert wird. Dabei gilt: Wenn ein empfangenes Profil einen neueren `TS` hat als das lokal gespeicherte, werden nur die betroffenen Felder überschrieben.

- **Angebote werden als LWW-Element-Set angelegt.** Jedes neu erstellte Angebot erhält wiederum eine eindeutig identifizierbare `OID` sowie einen TS, der bei Änderung des Angebots aktualisiert wird. Der Status eines Angebots kann entweder «aktiv» oder «inaktiv» sein, Angebotsdetails können nicht verändert werden.

Jeder Benutzer hält neben seinem eigenen Profil noch ein LWW-Element-Set mit allen Profilen, inklusive seinem eigenen, welches per Broadcast periodisch synchronisiert wird. Einmal synchronisierte Profile bzw. Angebote werden nie gelöscht, können aber auf «inaktiv» gestellt werden, um sie vom GUI auszublenden. Dabei können Angebote vom Besitzer selbst inaktiviert werden oder durch ein automatisches Timeout (3 Wochen). Löscht ein Benutzer sein Profil, wird einmalig eine letzte Broadcast-Nachricht versendet, mit der allen anderen Peers im Netz signalisiert wird, dass die Angebote dieses Benutzers inaktiviert werden können. Zusätzlich gibt es für jeden Benutzer einen `lastSeen`-Zeitstempel `LS`, der bei jeder Aktivität aktualisiert wird. Wenn `LS` durch Abgleich mit der lokalen Zeit einen bestimmten Grenzwert überschritten hat, werden ebenfalls alle Angebote des fraglichen Benutzers inaktiviert.

Damit bleibt die Systemkonsistenz erhalten: Die zugrundeliegenden CRDT-Strukturen bleiben vollständig, während einzelne Daten im GUI situationsabhängig gefiltert werden. Nachfolgend ist der Algorithmus zur Aktualisierung der lokalen Angebotsliste dargestellt:

```
public static void updateOrAdd(List<Offer> offers, Offer newOffer) {
    for (int i = 0; i < offers.size(); i++) {
        Offer existing = offers.get(i);
        if (existing.getOfferId().equals(newOffer.getOfferId())) {
            existing.merge(newOffer);
            return;
        }
    }
    offers.add(newOffer);
}

public void merge(Offer other) {
    if (!this.offerId.equals(other.offerId)) return;
    if (other.lastModified > this.lastModified) {
        this.title = other.title;
        this.description = other.description;
        this.category = other.category;
        this.status = other.status;
        this.deleted = other.deleted;
        this.lastModified = other.lastModified;
    }
}

private void updateTimestamp() {
    this.lastModified = System.currentTimeMillis();
}
```

Da der `TS` im `updateTimestamp`-Aufruf lokal über `System.currentTimeMillis` gesetzt wird, basiert die Konfliktauflösung implizit auf der Systemzeit. Dies kann in seltenen Fällen zu Inkonsistenzen führen, wenn Geräte abweichende Uhren verwenden.

2.2.5 Verschlüsselung und Datenschutz

Um die Vertraulichkeit der übertragenen Daten zu gewährleisten, verschlüsselt die App alle Nachrichteninhalte mit Klartext vor dem Senden mit dem symmetrischen Verschlüsselungsverfahren AES im CTR-Modus. Der Schlüssel wird deterministisch aus der Mesh-ID der jeweiligen Hausgemeinschaft abgeleitet. Dazu wird die MID mittels SHA-256 gehasht und die ersten 16 Bytes als AES-Schlüssel verwendet. Die MID wird aus den eingegebenen Adressdaten berechnet, indem alle Leer- und Sonderzeichen entfernt, der Text in Kleinbuchstaben umgewandelt und anschliessend per SHA-256 gehasht wird. Aus dem Hash wird ein kompaktes, 6 Byte langes Präfix extrahiert und Base64-kodiert, um eine kurze, aber stabile MID zu erzeugen. Auf diese Weise besitzen alle Benutzerprofile mit derselben Adresse denselben Schlüssel, ohne dass dieser explizit gespeichert werden muss. Jede Nachricht wird zusätzlich mit einem zufällig generierten Initialisierungsvektor `IV` versehen, der gemeinsam mit dem verschlüsselten Text im Netzwerk übertragen wird. Die Kombination aus deterministischem Schlüssel und zufälligem `IV` garantiert, dass identische Nachrichten bei jeder Übertragung zu unterschiedlichen Chiffren führen. Die Entschlüsselung erfolgt analog, indem der Schlüssel erneut aus der `MID` abgeleitet und der Klartext aus dem verschlüsselten Inhalt der Nachricht und dem `IV` rekonstruiert wird.

3 Test und Evaluation

Die App wurde während der Entwicklung kontinuierlich getestet – primär direkt auf den Android-Smartphones unter realen Bedingungen mit verbundenem LoRa-Dongle. Die Überprüfung erfolgte sowohl visuell über die grafische Oberfläche als auch durch gezielte Analyse der Logcat-Ausgabe. Für die beiden Netzwerkklassen `PeerSyncManager` und `OfferSyncManger` wurden automatisierte Unit-Tests implementiert, um die korrekte Verarbeitung und CRDT-konforme Konfliktauflösung der synchronisierten Daten zu prüfen.

Ein wiederkehrendes Problem im Testprozess war der Umgang mit den USB-Datenpaketen, die vom Dongle in fragmentierter Form übertragen werden. Die Daten mussten korrekt zusammengesetzt werden, bevor daraus vollständige JSONObjects rekonstruiert werden konnten. Besonders bei grossen Nachrichten (z. B. Listen mit vollständigen Angebotsdaten) kam es wiederholt zu korrupten Datenchunks, weil der Dongle zu früh wieder in den Sendemodus wechselte, bevor ein vollständiger Empfang abgeschlossen war. Dies führte zu JSON-Parsing-Fehlern. Um mit diesen Fehlern umzugehen, wurde in der Angebots-Synchronisation `handleOfferData` ein Robustheitsmechanismus implementiert, der unvollständige oder fehlerhafte JSON-Elemente extrahiert und ignoriert, damit zumindest die korrekten Angebote verarbeitet werden können. In anderen Modulen wurde auf solche Mechanismen verzichtet, was weiterhin zu Problemen führen kann.

Insgesamt zeigt die App trotz dieser Herausforderungen ein stabiles Verhalten in typischen Nutzungsszenarien. Die Synchronisation erfolgt zuverlässig, solange gültige Nachrichten empfangen werden können, und alle Benutzeraktionen auf dem GUI werden korrekt im Mesh verarbeitet. Screenshots der graphischen Benutzeroberfläche sowie von relevanten Logging-Ereignissen finden sich im Anhang des Berichts.

4 Ausblick

Mit der Flurfunk-App konnte ein funktionierender Prototyp für ein vollständig dezentrales Kommunikationssystem auf Basis von LoRa implementiert werden. Im Fokus standen dabei Robustheit, Energieeffizienz und ein CRDT-kompatibles Datenmodell zur Sicherstellung der Systemkonsistenz. Für den produktiven Einsatz ergeben sich jedoch verschiedene Erweiterungs- und Verbesserungsmöglichkeiten:

- **Nachrichtenweiterleitung.** Ein ursprünglich implementierter Mechanismus zur paketbasierten Weiterleitung über mehrere Peers (sogenanntes «Hopping» wie z. B. in Meshtastic realisiert) wurde wieder entfernt, da sich damit in der Praxis keine stabile Zustellung realisieren liess. Zukünftig könnte eine robustere Hopping-Logik mit TTL-Feldern, Duplikaterkennung und explizitem Routing wieder aufgenommen werden.
- **Resilienz gegenüber fragmentierten und fehlerhaften Nachrichten.** Die systematische Behandlung beschädigter oder unvollständiger Datenpakete ist bisher nur teilweise implementiert. Ein generisches, fehlertolerantes Paket-Parsing wäre hier sinnvoll.
- **Sicherheit und Datenschutz.** Die aktuelle symmetrische Verschlüsselung auf Basis der Mesh-ID ist pragmatisch, aber unsicher gegen gezielte Angriffe. Eine Erweiterung um asymmetrische Schlüsselpaare und Signaturen wäre sinnvoll, um Manipulation und Identitätsdiebstahl zu verhindern.
- **GUI- und UX-Verbesserungen.** Viele Funktionen (z. B. Inaktivität von Angeboten oder Peers) werden nur logisch verarbeitet, aber nicht explizit im Frontend dargestellt. Eine bessere visuelle Rückmeldung über den Zustand des Netzwerks würde die Transparenz und Nutzbarkeit verbessern.

Trotz aller Limitierungen zeigt die App exemplarisch, wie sich mit einfachen Mitteln ein robustes, selbstorganisiertes Kommunikationssystem aufbauen lässt. Der gesamte Code steht auf GitHub zur Verfügung: https://github.com/Antimon80/Flurfunk

5 Literaturverzeichnis

Digitec Galaxus AG. (2025). *Digitec - Development Boards and Kits*. Abgerufen am 04. Juli 2025 von Digitec: https://www.digitec.ch/en/s1/product/waveshare-sx1262-lora-tcxo-usb-adapter-850-930mhz-development-boards-kits-53127368

Kai Morich. (2024). *GitHub Repository*. Abgerufen am 04. Juli 2025 von SimpleUsbTerminal: https://github.com/kai-morich/SimpleUsbTerminal

Kai Morich. (2025). *usb-serial-for-android*. Abgerufen am 04. Juli 2025 von GitHub Repository: https://github.com/mik3y/usb-serial-for-android

Meshtastic LLC. (unbekannt). *Meshtastic*. Abgerufen am 04. Juli 2025 von Meshtastic: https://meshtastic.org/

Shapiro, M., Preguiça, N., Baquero, C., & Zawirski, M. (2011). *A comprehensive studiy of Convergent and Commutative Replicated Data Types*. Paris: Inria - Centre Paris Rocquencourt.

Union schweizerischer Kurzwellen-Amateure. (2024). *Was ist LoRa?* Abgerufen am 04. Juli 2025 von USKA: https://uska.ch/lora/

Waveshare. (unbekannt). *USB-TO-LoRa-xF*. Abgerufen am 04. Juli 2025 von Waveshare Wiki.

Redlichkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte Hilfe angefertigt habe. Es wurden keine weiteren Quellen als jene, die im Literaturverzeichnis ausgewiesen sind, verwendet.

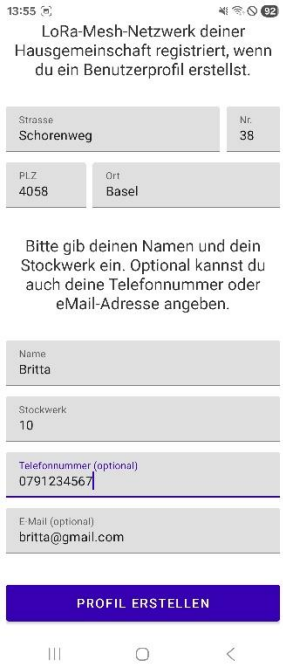
Ich habe im Rahmen der Erstellung dieser Arbeit die Sprach-KI ChatGPT (OpenAI) genutzt, um mir bei der Formulierung von Textabschnitten, der Überprüfung technischer Konzepte sowie bei der Erklärung der verwendeten Datenmodelle und Algorithmen helfen zu lassen. Alle durch ChatGPT generierten Inhalte wurden kritisch geprüft, technisch überarbeitet und in eigener Verantwortung in den Gesamtkontext eingeordnet. Das Logo der App ist KI-generiert.

Basel, 05.07.2025

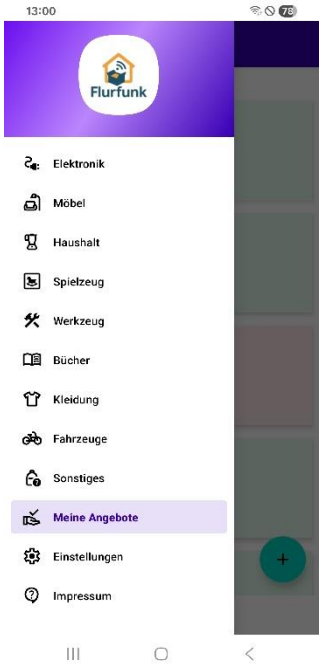
Anhang

Graphische Benutzeroberfläche

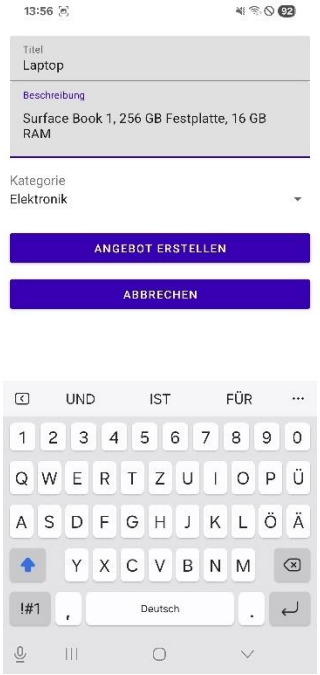
Startbildschirm beim erstmaligen Öffnen der App



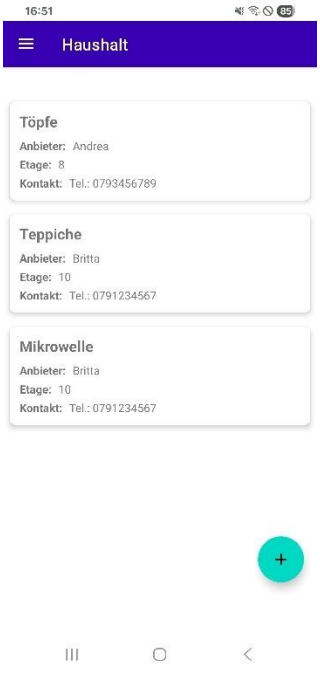
Erstellung des Benutzerprofils



Menüleiste mit Angebotskategorien



Erstellung eines Angebots



Listenansicht einer Angebotskategorie

Detailansicht eines fremden Angebots

Listenansicht der eigenen Angebote (farbig hinterlegt je nach Aktivitätsstatus)

Detailansicht eines eigenen Angebots

Einstellungen zum Ändern der eigenen Kontaktdaten

Impressum

Relevante Logcat-Ereignisse

```
07-02 20:08:03.243 28021 28021 I MainActivity: TX size-probe  seq=8  len=900
07-02 20:08:22.944 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:22.945 28021 28133 D ATLoRaManager: Chunk (64 B): #TST|SEQ=8;LEN=900;DAT=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:22.949 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:22.950 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:22.956 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:22.956 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:22.961 28021 28133 D LoRaSocket: onNewData: received 48 bytes
07-02 20:08:22.961 28021 28133 D ATLoRaManager: Chunk (48 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:23.749 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:23.749 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:23.755 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:23.755 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:23.761 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:23.761 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:23.766 28021 28133 D LoRaSocket: onNewData: received 48 bytes
07-02 20:08:23.766 28021 28133 D ATLoRaManager: Chunk (48 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:24.554 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:24.554 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:24.560 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:24.560 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:24.566 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:24.566 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:24.571 28021 28133 D LoRaSocket: onNewData: received 48 bytes
07-02 20:08:24.571 28021 28133 D ATLoRaManager: Chunk (48 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:25.253 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:25.253 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:25.259 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:25.259 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:25.265 28021 28133 D LoRaSocket: onNewData: received 64 bytes
07-02 20:08:25.265 28021 28133 D ATLoRaManager: Chunk (64 B): AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
07-02 20:08:25.269 28021 28133 D LoRaSocket: onNewData: received 15 bytes
07-02 20:08:25.269 28021 28133 D ATLoRaManager: Chunk (15 B): AAAAAAAAAAA;EOM
07-02 20:08:25.273 28021 28133 I ATLoRaManager: Full message to be parsed: #TST|SEQ=8;LEN=900;DAT=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA;EOM
```

Die maximale Grösse einer vollständig rekonstruierbaren Protokollnachricht liegt gemäss Hersteller bei 960 Bytes, was durch systematisches Testen verifiziert werden konnte.

```
07-04 18:39:39.829   993  1035 D LoRaSocket: onNewData: received 64 bytes
07-04 18:39:39.830   993  1035 D ATLoRaManager: Chunk (64 B): #SYNOF|MID=TEfOejuO;ID=39c12ac3fe98edb8;OFF=[{"OID":"858ac17b0fe
07-04 18:39:39.832   993  1035 D LoRaSocket: onNewData: received 32 bytes
07-04 18:39:39.832   993  1035 D ATLoRaManager: Chunk (32 B): 4469e","TS":1751644398974},{"OID
07-04 18:39:40.091   993  1035 D LoRaSocket: onNewData: received 64 bytes
07-04 18:39:40.092   993  1035 D ATLoRaManager: Chunk (64 B): ":"24cf009835dee9cf","TS":1751644403712},{"OID":"50906dd5d18095b
07-04 18:39:40.895   993  1035 D LoRaSocket: onNewData: received 64 bytes
07-04 18:39:40.896   993  1035 D ATLoRaManager: Chunk (64 B): f","TS":1751646777372},{"OID":"880e74a182e2fa7e","TS":1751644408
07-04 18:39:40.901   993  1035 D LoRaSocket: onNewData: received 64 bytes
07-04 18:39:40.901   993  1035 D ATLoRaManager: Chunk (64 B): 618},{"OID":"19eb0d36e30db3bb","TS":1751644412912},{"OID":"85b50
07-04 18:39:40.907   993  1035 D LoRaSocket: onNewData: received 64 bytes
07-04 18:39:40.908   993  1035 D ATLoRaManager: Chunk (64 B): 34ba49a7157","TS":1751644610741},{"OID":"b0571e013daeae0a","TS":
07-04 18:39:40.912   993  1035 D LoRaSocket: onNewData: received 48 bytes
07-04 18:39:40.912   993  1035 D ATLoRaManager: Chunk (48 B): 1751644613561},{"OID":"50bda9911842378e","TS":17
07-04 18:39:41.648   993  1035 D LoRaSocket: onNewData: received 64 bytes
07-04 18:39:41.649   993  1035 D ATLoRaManager: Chunk (64 B): 51644615868},{"OID":"0695c44fb2fc25b0","TS":1751644618217},{"OID
07-04 18:39:41.652   993  1035 D LoRaSocket: onNewData: received 64 bytes
07-04 18:39:41.652   993  1035 D ATLoRaManager: Chunk (64 B): ":"0a33d30189073b11","TS":1751646295386},{"OID":"9c711e4cdc67a70
07-04 18:39:41.658   993  1035 D LoRaSocket: onNewData: received 64 bytes
07-04 18:39:41.659   993  1035 D ATLoRaManager: Chunk (64 B): c","TS":1751646631571},{"OID":"652eb4d3861c05d2","TS":1751646506
07-04 18:39:41.661   993  1035 D LoRaSocket: onNewData: received 30 bytes
07-04 18:39:41.661   993  1035 D ATLoRaManager: Chunk (30 B): 875}];SID=d84c9430b05b4cf6;EOM
07-04 18:39:41.662   993  1035 I ATLoRaManager: Full message to be parsed: #SYNOF|MID=TEfOejuO;ID=39c12ac3fe98edb8;OFF=[{"OID":"858ac17b0fe4469e","TS":1751644398974},{"
OID":"24cf009835dee9cf","TS":1751644403712},{"OID":"50906dd5d18095bf","TS":1751646777372},{"OID":"880e74a182e2fa7e","TS":1751644408618},{"OID":"19eb0d36e30db3bb","TS":1
751644412912},{"OID":"85b5034ba49a7157","TS":1751644610741},{"OID":"b0571e013daeae0a","TS":1751644613561},{"OID":"50bda9911842378e","TS":1751644615868},{"OID":"0695c44f
b2fc25b0","TS":1751644618217},{"OID":"0a33d30189073b11","TS":1751646295386},{"OID":"9c711e4cdc67a70c","TS":1751646631571},{"OID":"652eb4d3861c05d2","TS":1751646506875}]
;SID=d84c9430b05b4cf6;EOM
```

Eine Broadcastnachricht mit allen beim Peer lokal vorhandenen Angebots-IDs und den zugehörigen Zeitstempeln wurde empfangen und für das Protokollparsing rekonstruiert.

```
07-04 18:17:38.093   993  1035 I ATLoRaManager: Full message to be parsed: #REQOF|MID=TEfOejuO;ID=f642d4d20dc9cd51;SID=d84c9430b05b4cf6;REQ=["85b5034ba49a7157","b0571e0
13daeae0a","50bda9911842378e","0695c44fb2fc25b0","9c711e4cdc67a70c","0a33d30189073b11"];EOM
07-04 18:17:38.110   993  1035 D OfferSyncManager: OFDAT larger than 960 B - #OFDAT|MID=TEfOejuO;ID=25670550d8312eda;OFA=tnXaiRmVDhT2+9V63qDu83T9GfdsVp9RSvk0u5qrPJ9tShT
bSpKM6KMrn2gD1tolc7b/e9Sp85+HmhMfVXTaHdOtUh/tQGSr0XjJxXY960dmlOxQvQYrNY7C90kUWoeoc13ojR2PdbxTcywIrFI3080QZPQMpeFOLa0R/maf7Q/CnWTdYuvAivX2jypwbYEJYLi0TbqAmDhZFhqLZ1niHLX
ssbSE9qSNodbivTHbAM6QC9LtBI34PhXrqJcuog1IjvnFlfdcy+0oRvXCG/KraHrE7gEN2KawqzB8YVscXrz3bU+kiCQdSsYSjz7Me3lVEqR3aByUFEOsc6hBHw0G6PFN/4Yng7RygjYD4+zhUPocsSJ3b5WHcEdc+XJ5OLT
EQ7hM8lJHEOze36A6Un3DbJP0+RnrcHvZCyrNoCENmIFJ8U9lMM7+Nxlj1NVwRX+DOBApeOYj1HIBOAkSqSTMQt0hpityApPQqr2pHWQjKHXkVAg0Gkvej3fcu0oKn2Xij9qY49kLmuPy3mo1lc1h8qYWWU6cZZ4cEx9eM+v
nlg1bY6aRzPv9w+KqO/3YdE5e6cpCt8uwzjlPZO+9LAKcCnRrnyuROJ48emkuaEdotvy1ZySu8PYwJZnXpfLJge40OXO2BEWzJi+hVRR8ZeOqEEOpUFZpd8tmQ6jgRgCvppM+3Gvn0ydVFjILfEV2Gat9OGoSfIfnLa758+W
tIvvLnpIqywYPhWtNBzhMZNbUel1hilWqGGe3iUzazVXh3GtcqHb6cxMl/VP1sN3PTafWALKsww==;IV=xNLjufdSCIRziaTDk6PIhA==;SID=0c05ce2c68f552f0;EOM
07-04 18:17:38.111   993  1035 D OfferSyncManager: OFDAT smaller than 960 B - #OFDAT|MID=TEfOejuO;ID=fe61a31e1f256f2b;OFA=Gb/XZem6ZOBdJyz/5Xj7JNEhtkemIpHpDTHrdHc4crWWHX
UtImnYiLFZtIRs7DhKRDoUX8O0oq6mzNz45xYVBrZ+pZtq+16/Vq2TStdjuNN7dqcB+3eBpQ1znENwIT81uvsmC1QU5sAMD/Ak+ApqcFIGGA7/m9SlX1v/IlgqFWwHJLILuV3xw+wkrRI2+fsnv4sHSUwKmICx;IV=N09W2A
41B0aVcYzY7zh2sw==;SID=0c05ce2c68f552f0;EOM
07-04 18:17:38.398   993  1024 D ATLoRaManager: Message to be sent: #OFDAT|MID=TEfOejuO;ID=25670550d8312eda;OFA=tnXaiRmVDhT2+9V63qDu83T9GfdsVp9RSvk0u5qrPJ9tShTbSpKM6KMr
n2gD1tolc7b/e9Sp85+HmhMfVXTaHdOtUh/tQGSr0XjJxXY960dmlOxQvQYrNY7C90kUWoeoc13ojR2PdbxTcywIrFI3080QZPQMpeFOLa0R/maf7Q/CnWTdYuvAivX2jypwbYEJYLi0TbqAmDhZFhqLZ1niHLXssbSE9qSN
odbivTHbAM6QC9LtBI34PhXrqJcuog1IjvnFlfdcy+0oRvXCG/KraHrE7gEN2KawqzB8YVscXrz3bU+kiCQdSsYSjz7Me3lVEqR3aByUFEOsc6hBHw0G6PFN/4Yng7RygjYD4+zhUPocsSJ3b5WHcEdc+XJ5OLTEQ7hM8lJH
EOze36A6Un3DbJP0+RnrcHvZCyrNoCENmIFJ8U9lMM7+Nxlj1NVwRX+DOBApeOYj1HIBOAkSqSTMQt0hpityApPQqr2pHWQjKHXkVAg0Gkvej3fcu0oKn2Xij9qY49kLmuPy3mo1lc1h8qYWWU6cZZ4cEx9eM+vnlg1bY6aR
zPv9w+KqO/3YdE5e6cpCt8uwzjlPZO+9LAKcCnRrnyuROJ48emkuaEdotvy1ZySu8PYwJZnXpfLJge40OXO2BEWzJi+hVRR8ZeOqEEOpUFZpd8tmQ6jgRgCvppM+3Gvn0ydVFjILfEV2Gat9OGoSfIfnLa758+WtIvvLnpIq
ywYPhWtNBzhMZNbUel1hilWqGGe3iUzazVXh3GtcqHb6cxMl/VP1sN3PTafWALKsww==;IV=xNLjufdSCIRziaTDk6PIhA==;SID=0c05ce2c68f552f0;EOM
07-04 18:17:38.398   993  1024 D ATLoRaManager: TX → 893 B
07-04 18:17:39.991   993  1024 D ATLoRaManager: Message to be sent: #OFDAT|MID=TEfOejuO;ID=fe61a31e1f256f2b;OFA=Gb/XZem6ZOBdJyz/5Xj7JNEhtkemIpHpDTHrdHc4crWWHXUtImnYiLFZ
tIRs7DhKRDoUX8O0oq6mzNz45xYVBrZ+pZtq+16/Vq2TStdjuNN7dqcB+3eBpQ1znENwIT81uvsmC1QU5sAMD/Ak+ApqcFIGGA7/m9SlX1v/IlgqFWwHJLILuV3xw+wkrRI2+fsnv4sHSUwKmICx;IV=N09W2A41B0aVcYzY
7zh2sw==;SID=0c05ce2c68f552f0;EOM
07-04 18:17:39.991   993  1024 D ATLoRaManager: TX → 301 B
```

Datenpakete, die eine Grösse von 960 Byte überschreiten, werden in kleinere Pakete aufgeteilt und jeweils als eigenständige Protokollnachrichten unmittelbar hintereinander versendet.

```
07-04 18:18:43.848   993  1035 I ATLoRaManager: Full message to be parsed: #OFDAT|MID=TEfOejuO;ID=bcfaaf6463832811;OFA=iroIuMtVLftaM+R+ZhL+t/rhFdOWBMtQJ6CAQs4bPpRoB74yS
kXJxW5a3SoDcaKdXvzCa76WklCZf7/Dti9EDDuolP9UNMVnUPQN8SyEIiGRE7HihHMfhn+6coImGqq7zt8S1lWzkhhdn76mtwJB9KNaroYNohNDua9vJp8Wmv5I1cnSpj03ZZ0DJLjQYeFbiZbNCNDS0s+Crv+MkTfnCA7y4
HYW23xVRa+ktwJLzv9b9kSEOue5ENi7p4XMnPUVJXCT7TqlxSpjmLyBWauxRWTWdcrgdf3NDZqNeqZOBlH/rSO6kcYhSl5l9lA3+pCAYXgalGSfkBw+M42jMav0oXPCY7HS85S8UBXCMUY1Ql6Ezz2W3XDfWYiInvkHCBwO8
SwlUdCKoElgKCGGRjrZo8/lJbzWwBEHJqu8FIDVN/ljbD31StvjNJmqIjUfUmrK/p7MqNp06CkEP+LXHAdxVgEDYmTAqcP7URSWNOU9vRh7ZSEE0QZzr1CH8b/UGNZg/CEyWpHRo5BTKi8wWlfR5UfBIlAoDjbf0kRmwT8vi
nZG0mLtOUPp3TJiV5nl91k0rTe6kMFPVjohAW55JEV5onAAHEnuezNoXc33qOWxCDAcV7U9UZY0P4EjxvMCqzO5pnfFJP5EyqWTl0ZCdGkfiLuCX7PgJOid2+PcXWSJU2sIA4ykxYpWcM7dKtYwzaIgs6MWA9ojzavogd9fI
WxRG6CxZxsfrJtcxdN7s8JRzIoOkVUPCFg92M7z8QSnHaZm8Kd2HS37wTFWBsIPVa4pV49Wo7ot+2eMcNu+DwzM613m+//UhgzUbcZJ;IV=qCCcck6HVgZ/VDIr5IMiAw==;SID=d84c9430b05b4cf6;EOM
07-04 18:18:43.888   993  1035 I OfferSyncManager: Saving incoming offer data
```

Eine Liste mit verschlüsselten Angebotsinformationen wurde empfagen, erfolgreich geparst und lokal gespeichert.

```
07-05 15:05:24.253   612   963 I ATLoRaManager: Full message to be parsed: #OFDAT|MID=TEfOejuO;ID=41fe8beecf00dec0;OFA=8KzlzLief/L2ltzlQCDemc7y+ItONUQCRR2deeb5pAy5CcCU6
qQbdjU/vFlzCgDwP7XPFhN9af1FC31i+uVc69GxStY+lcSbdUBPmb+PEaKP2uZm/Qqw5yaDcym4CN5I/U2G/KJXCYPcS/u3Y4K2nsrY4r2a4eF4wmQsKiqOe505dsj9FHE7e5WFkV4PtIPtq3lirpoMS3v1qHbNb2E0teeDb
dSDtuHUhrVZkWVGk8cce3AHLj+aWEMM7pm7/W1V1aR0ZZAidVZU9datQ7bZEOW70sl8HNDRB/3091gH0pX2gSqWH4JtFBzFidLlqjP2YhRJpqVQ6kD1B7P8fbVDuaXhewPuRJIQ/w4RrvbQIWRTk24cI3KEks8Cgmz0MJTuQ
4Pb490kg08aM8N6bMGpb5xV2kQXivhLmhLq2S5vDtRZKL2MOB7RBGT85pIlBolq8ciYg1yd+vgGZezRELhhFJHFaTYhNmo+OYH3Fkj+SA4+LBltYy22lwd4fAr4eWYAbHCAQp3fdNgKuVqfE4t8x3qHdE5HCp8l8gw45ESkX
n4J87v8bTLYW0r8WCId58YHdg==;IV=6VAflsHIx/Ih1l46TTBZhKw==;SID=6e84ad077230fc3c;EOM
07-05 15:05:24.275   612   963 E OfferSyncManager: Failed to handle incoming offer data
07-05 15:05:24.275   612   963 E OfferSyncManager: org.json.JSONException: Unterminated string at character 426 of [{"OID":"1cb1189cf49c1fbf","TS":1751716666260,"TTL":"
Laptop","DSC":"Surface Book 1, 256 GB Festplatte, 16 GB RAM","CAT":"ELC","CID":"6e84ad077230fc3c","STA":"AC"},{"OID":"bc7618197cc3949d","TS":1751716695267,"TTL":"Karten
spiele","DSC":"UNO, Bonanza, Hol's der Geier","CAT":"TOY","CID":"6e84ad077230fc3c","STA":"AC"},{"OID":"216◆◆_X◆c◆H>VoXb$◆◆◆<(◆◆◆ek◆@
Zk◆◆◆mB◆95◆◆◆%◆ {L◆◆◆◆C◆E◆◆◆◆S◆◆◆◆◆'x◆|◆5x◆□◆%[T◆◆◆◆b◆◆$◆j◆◆◆'◆^
07-05 15:05:24.275   612   963 E OfferSyncManager:       at org.json.JSONTokener.syntaxError(JSONTokener.java:469)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at org.json.JSONTokener.nextString(JSONTokener.java:239)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at org.json.JSONTokener.nextValue(JSONTokener.java:111)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at org.json.JSONTokener.readObject(JSONTokener.java:403)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at org.json.JSONTokener.nextValue(JSONTokener.java:104)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at org.json.JSONTokener.readArray(JSONTokener.java:449)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at org.json.JSONTokener.nextValue(JSONTokener.java:107)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at org.json.JSONArray.<init>(JSONArray.java:94)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at org.json.JSONArray.<init>(JSONArray.java:110)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.example.flurfunk.network.OfferSyncManager.handleOfferData(OfferSyncManager.java:321)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.example.flurfunk.network.MessageDispatcher.onMessageReceived(MessageDispatcher.java:128)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.example.flurfunk.MainActivity$$ExternalSyntheticLambda1.accept(D8$$SyntheticClass:0)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.example.flurfunk.network.ATLoRaManager.handleFrame(ATLoRaManager.java:395)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.example.flurfunk.network.ATLoRaManager.$r8$lambda$MfmBxhQUb2fYIW9g8fL6kceQReY(Unknown Source:0)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.example.flurfunk.network.ATLoRaManager$$ExternalSyntheticLambda0.accept(D8$$SyntheticClass:0)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.example.flurfunk.network.ATLoRaManager$SerialFrameReader.onSerialRead(ATLoRaManager.java:477)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.example.flurfunk.network.ATLoRaManager.onSerialRead(ATLoRaManager.java:310)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.example.flurfunk.usb.LoRaSocket.onNewData(LoRaSocket.java:127)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.hoho.android.usbserial.util.SerialInputOutputManager.stepRead(SerialInputOutputManager.java:309)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.hoho.android.usbserial.util.SerialInputOutputManager.runRead(SerialInputOutputManager.java:240)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at com.hoho.android.usbserial.util.SerialInputOutputManager$$ExternalSyntheticLambda0.run(D8$$SyntheticClass:0)
07-05 15:05:24.275   612   963 E OfferSyncManager:       at java.lang.Thread.run(Thread.java:1119)
07-05 15:06:47.733   612   612 D MainActivity: Broadcast peer sync sent.
```

Timing-Probleme führen gelegentlich dazu, dass falsch decodierte Nachrichten nicht geparst werden können.