Routing Protokoll

Table of Contents

1. Einführung.....2
2. Senden von Paketen.....2
3. Paketaufbau.....2
  3.1 RREQ.....2
  3.2 RREP.....3
  3.3 RERR.....3
  3.4 MSG.....4
  3.5 ACK.....4
4. Adressen.....5
5. Modulkonfiguration.....5
6. Beispielablauf des Route-Aufbaus.....5
7. Routes-Tabelle.....8
8. Reverse Routing Table.....9
9. Timers.....9
  9.1 Zufällige Wartezeit.....10
  9.2 Hash-Time.....10
10. Acknowledgements und Timeouts.....10
  10.1 Acknowledgements.....10
  10.2 Timeouts.....10
11. Sequenznummern.....11
12. Metric/Hop Counts.....12
13. Route Errors.....12
14. Flags.....13

Von Oliver Büttner, Bennett Hollstein

1. Einführung

Das Dokument beschreibt das Routing Protokoll für Technik mobiler Systeme im WiSe 2021/2022.

In dem Netzwerk gibt es keine zentrale Kontrollinstanz, sondern jede Node sorgt eigenständig dafür, dass Nachrichten korrekt weitergesendet werden. Es wird davon ausgegangen, dass in dem Testnetzwerk keine böswilligen Nodes vorhanden sind, die z.B. Routes manipulieren.

2. Senden von Paketen

Das Senden von Paketen geschieht über LoRa Module. Hierbei wird ein Großteil der Pakete vollständig in Base64 kodiert.

Gesendet wird auf der Schicht der LoRa Module immer auf die Broadcast-Adresse (0xFFFF) des Moduls, da das Adressieren von Geräten durch das Protokoll selbst geregelt wird.

Soll z.B. folgendes folgendes ACK-Paket (siehe Kapitel 3.) übertragen werden:

```
Type Flags   Hop Addr.   Prev. Hop
0000 0000    3           67
```

Wird dies als base64-encodierter String versendet:

```
QAND
```

Der Aufbau der Pakete wird in dem Dokument **Packets.pdf** beschrieben. Die genauere Nutzung der Pakete und weitere Beispielpakete werden in den darauf folgenden Kapiteln genauer beschrieben.

4. Adressen

Für die Adressen werden 8 Bit Unsigned Integer (0-254) genutzt. Jeder Teilnehmer kann hierbei seine Adresse selbstständig wählen, es existiert keine automatische Adressvergebung wie DHCP, um Netzwerktraffic einzusparen.

Eine Adresse darf trotzdem immer nur von einem Teilnehmer verwendet werden und sollte in der Lebensdauer des Netzwerks nicht an andere vergeben werden.

Die Adressen werden daher im Kurs fest vergeben.

Die Adresse 255 (0b11111111) ist als Broadcast-Adresse reserviert und alle Nodes sollten darauf reagieren.

5. Modulkonfiguration

Die LoRa Module werden über folgenden Befehl konfiguriert:

`AT+CFG=433000000,5,6,12,4,1,0,0,0,0,3000,8,8`

Hierdurch wird sichergestellt, dass die Module ausreichend Fehlerkorrektur benutzen.

6. Beispielablauf des Route-Aufbaus

Das nachfolgende Beispiel soll einen allgemeinen Überblick über den Verbindungsaufbau im Netzwerk geben. Die einzelnen Teile werden in darauf folgenden Schritten genauer erläutert.

In diesem Beispiel wird folgender Netzwerkaufbau genutzt:

```
N1 ↔ N2 ↔ N3 ↔ N4
     ↕
     N5
```

Möchte N1 die Verbindung zur Node N4 aufbauen und hat noch keinen existieren Eintrag in der Routes-Tabelle (siehe Kapitel 7.), sendet dieses Modul ein Route Request (RREQ) Paket an die Broadcast-Adresse.

Dieses Paket sieht hierbei wie folgt aus:

Type	Flags	Hop Addr.	Prev. Hop	Req. Id	Dest. Addr.
0000	0000	255	1 (N1)	1	4 (N4)

Hop Count	Originator Addr.	Originator Seq.
0	1 (N1)	1

In diesem Fall wird N2 die Nachricht empfangen (siehe Netzwerkaufbau). Auch Node N2 besitzt N4 nicht in der Routing Tabelle.

N2 trägt die Request daher in die eigene Reverse-Route Tabelle (siehe Kapitel 8.) ein und broadcastet das RREQ weiter.

Die von N2 weitergeleitete Nachricht sieht wie folgt aus:

Type	Flags	Hop Addr.	Prev. Hop	Req. Id	Dest. Addr.
0000	0000	255	2 (N2)	1	4 (N4)

Hop Count	Originator Addr.	Originator Seq.
1	1 (N1)	1

N2 hat also die Previous Hop Address und den Hop Count (siehe Kapitel 12.) aktualisiert.

N3, N5 und N1 empfangen die Nachricht, wobei N3 und N5 die Nachricht weiter broadcasten, während N1 die Nachricht als Echo wegwirft (siehe Kapitel X).

Schließlich empfängt N4 die Route Request. Da N4 selber der Empfänger ist, sendet N4 eine Route Response zu N1 über N3 zurück.

Die von N4 gesendete RREP hat folgenden Aufbau:

Type	Flags	Hop Addr.	Prev. Hop	Req. Id	Dest. Addr.
0001	0000	3 (N3)	4 (N4)	1 (RREQ)	1 (N1)

Dest. Seq.	Hop Count	Originator Addr.
1	0	4 (N4)

N3 empfängt daraufhin das RREP Paket.

In seiner eigenen Reverse Routing Table sieht N3, dass das Paket im nächsten Schritt an N2 weitergeleitet werden muss.

Zusätzlich erstellt N3 auch selber einen Routing-Table Eintrag für N4, damit N3 keinen zusätzlichen Route Request machen muss, sollte diese Node eine Nachricht zu N4 senden wollen. Bei dem Routing-Table Eintrag wird zusätzlich N2 zu den „Precursors“ eingetragen – dies ist für Route Errors (siehe Kapitel 13.) wichtig.

N3 sendet die RREP also weiter an N2, welches die gleichen Schritte wie N3 durchführt (Reverse Routing Table lesen, Routing Table Eintrag erstellen, Hop Count um 1 erhöhen, Reply weitersenden an Hop aus Rev Routing Table).

Schließlich kommt die Route Reply beim Ausgangspunkt N1 ein.

N1 trägt die gefunden Route in die eigene Routing-Tabelle ein und kann daraufhin beispielsweise eine Message (MSG-Paket) an N4 mit der neuen Route gesendet werden.

Wenn nun z.B. N5 ebenfalls eine Route Request für N4 sendet, empfängt N2 dies. Da N2 nun jedoch bereits einen Eintrag in seiner Routing Table für N4 hat – aus dem vorherigen Request – kann N2 direkt eine Route Response an N5 zurücksenden.

Diese Response hat dabei folgenden Aufbau:

Type Flags	Hop Addr.	Prev. Addr.	Req. Id	Dest. Addr.
0001 0000	5 (N5)	2 (N2)	1	5 (N5)

Dest. Seq.	Hop Count	Originator Addr.
1	2 (RT)	2 (N2)

7. Routes-Tabelle

Jede Node im Netzwerk hält eine eigene Routes-Tabelle, welche Informationen über die Erreichbarkeit von anderen Nodes enthält.

Wichtig ist hierbei, dass jede Node nur jeweils den nächsten Hop speichert und nicht den kompletten Routing-Pfad. Das Paket wird nur zum nächsten Hop geleitet. Der nächste Hop kann dann in seiner Routing-Tabelle den nächsten Hop herausfinden und so weiter.

Die Tabelle hat folgenden Aufbau:

Name	Datentyp	Beschreibung
Destination	NetworkAddress (Byte)	Adresse der Node, die final erreicht werden soll
NextHop	NetworkAddress (Byte)	Nächster Hop, welcher genutzt werden soll
Precursors	Array<NetworkAddress>	Nodes, die über die aktuelle Node an diese Destination senden. Dies ist wichtig für RERR Nachrichten
Metric	Byte	Hop Count bis zum Ziel
Sequence Number	Byte	Letzte bekannte Sequenznummerer Destination. Dies wird benutzt, um herauszufinden, ob die Route veraltet ist
Is Valid (Eine Möglichkeit)	Boolean	Gibt an ob die Route noch genutzt werden soll. Ist die Route nicht mehr valide, sollten RERR versendet werden

Bei Programmstart sollte sich die Node zunächst selber in die Routing Tabelle eintragen. Hierbei ist Destination die eigene Adresse, NextHop ist die eigene Adresse, Precursors ist leer, Metric ist 0, Sequence Number ist 0 und Is Valid ist true.

Dieser Eintrag vereinfacht die weitere Implementierung des Protokolls, da kein extra Sonderfall für die eigene Node programmiert werden muss, sondern immer die Routing-Tabelle genutzt werden kann.

Wird eine neue mögliche Route empfangen, wird geprüft ob die Route noch nicht gespeichert ist, ob die Metric kleiner ist oder die Sequenznummer neuer ist. Ist dies der Fall, wird die alte Route (falls vorhanden) invalidiert und die neue Route gespeichert.

8. Reverse Routing Table

Bei empfangenen Route Requests wird der Request in die Reverse Routing Tabelle eingetragen. Der Eintrag wird später dafür genutzt, um Antworten auf die Route Requests verarbeiten zu können.

Name	Datentyp	Beschreibung
Destination	NetworkAdress (Byte)	Adresse der Node, die durch den RREQ gesucht wird
Source	NetworkAdress (Byte)	Node, von welchem die RREQ kommt. Originator Address aus dem RREQ Paket
RREQ ID	Integer	Request ID des Pakets
Metric	Integer	Hop Count bis zum bis zur aktuellen Node
Previous Hop	NetworkAdress	Address welche die RREQ zuletzt weitergeleitet hat

Werden mehrere Route Requests mit der gleichen Source und gleichen Route Request ID empfangen, wird diejenige gespeichert, bei welchem die Metric (Hop Count) am niedrigsten ist. Schlechtere Einträge werden überschrieben, falls ein besserer gefunden wird.

Route Request IDs werden dazu genutzt, RREQ Pakete zu identifizieren. Die Request ID ist immer nur im Zusammenhang mit der Originator Adress eindeutig, da die Nodes unabhängig voneinander ihre Request IDs hochzählen.

Besitzt eine Node bereits einen Reverse Routing Table Eintrag für eine RREQ ID und eine Originator Adress, wird der Eintrag evtl. aktualisiert, falls die neuere Nachricht eine geringere Metric hat, die Request wird jedoch nicht erneut weitergeleitet, um Loops zu verhindern.

9. Timers

Wenn mehrere Nodes gleichzeitig versuchen zu senden (z.B. da beide Nodes dieselbe RREQ broadcasten wollen), kommt keine der beiden Nachrichten an. Da beide Nodes im Normalfall die gleiche Zeit für den Retry warten, kommt es wieder zur Kollision.

Timer veringern das Problem, indem diese zufällige Wartezeiten, anhand eines bestimmten Verfahrens generieren.

9.1 Zufällige Wartezeit

Bei der zufälligen Wartezeit wird eine Zufallszahl aus der Menge {0, 1, 2, 3} generiert.

9.2 Hash-Time

Der Hash-Time Algorithmus generiert anhand der Addresse eine möglichst eindeutige Sequenz der Menge {0, 1, 2, 3} und Länge 4. Mit jedem „Next“ wird die Zahl am Index aus der Liste genommen und der Index um eins verschoben.

```
private readonly int[] values = new int[4];
private int index = 0;
private int address;

public int Next(byte address) {
    if (address != this.address) {
        Generate(address);
        this.address = address;
    }
    int value = values[index];
    index = (index + 1) % 4;
    return value;
}

private void Generate(byte address) {
    int n = Hash(address);

    for (int i = 0; i < 4; i++) {
        int bits = (n >> (i * 2));
        values[i] = bits & 0b11;
    }
}

private int Hash(int n) {
    int p = 11;
    int hash = 13;
    hash = (hash ^ n) * p;
    return hash & 0xFF;
}
```

10. Acknowledgements und Timeouts

Das Protokoll will möglichst genau sicherstellen, dass Nachrichten angekommen sind, ohne zu viel zusätzlichen Traffic zu verursachen.

Hierzu werden ACK-Nachrichten und Timeouts verwendet.

10.1 Acknowledgements

Wird eine MSG Nachricht empfangen muss in jedem Fall direkt eine ACK Nachricht an den Previous Hop (Sender) gesendet werden – noch bevor die MSG weitergesendet wird.

Wenn die sendende Node nicht innerhalb von 120 Sekunden (2 Minuten) eine ACK Nachricht vom Empfänger bekommt, sendet die Node die gleiche Nachricht erneut.

Dies wird 3 Mal wiederholt, wonach eine Fehlermeldung gezeigt wird. Die Fehlermeldung sollte hierbei jedoch nicht die komplette Node abstürzen lassen, sondern nur als Informationsmeldung für den Operator dienen.

Bei Message (MSG) Paketen wird zusätzlich eine Route Error (RERR) Meldung an alle Precursors der Routing-Tabelle gesendet, um diese über die fehlerhafte Verbindung zu informieren (siehe Kapitel 13.).

10.2 Timeouts

Bei RREQ werden absichtlich keine ACK gesendet, da sonst zu viel zusätzlicher Traffic entstehen kann.

Stattdessen hält die ursprüngliche Node, die den RREQ versendet hat einen Timeout von 5 min. Wenn dieses ausläuft wird die originale RREQ Nachricht mit einer neuen Request Id und neuen Sequenznummer erneut gesendet. Nodes, welche die RREQ nur weiterleiten halten jedoch keinen Timeout.

Dies wird 3 Mal wiederholt, wonach eine Fehlermeldung gezeigt wird, dass keine Route aufgebaut werden kann.

11. Sequenznummern

Sequenznummern werden dazu genutzt, um zu überprüfen, ob eine Nachricht oder Route älter oder neuer als ein andere ist. Hierdurch können veraltete Routing-Informationen durch neuere ersetzt werden.

Jede Node zählt eine individuelle Sequenznummer hoch, es existiert also keine gemeinsame Sequenznummer für das gesamte Netzwerk.

Sequenznummern werden als unsigned Byte hochgezählt, bevor eine Nachricht erzeugt wird. Versendet die Node also eine neue RREQ Nachricht, wird davor die gespeicherte Sequenznummer erhöht und die neue Sequenznummer mit in dem Paket versendet.

Das reine Weiterleiten von Nachrichten (also RREQ, RREP oder MSG von anderen Nodes weitergeben) erhöht die eigene Sequenznummer nicht – erhöht wird nur, wenn ein eigenes Paket erzeugt wird.

Erreicht die Sequenznummer ihren Maximalwert (255 bei 8 Bit), findet ein Bit-Overflow statt – die nächste Sequenznummer ist also 0.

Um zu prüfen, ob eine Sequenznummer neuer oder älter ist, wird die Sequenznummer **als Zweierkomplement behandelt** und subtrahiert. Hierdurch ist es möglich, den relativen statt den absoluten Unterschied zu berechnen.

Nachfolgend ein Beispiel für die Implementierung des Sequenznummer-Vergleichs in TypeScript:

```
1. function twosComplementSubtract(a: number, b: number): number {
2.     return a + (~b + 1);
3. }
4.
5. export function withBitOverflow(number: number) {
6.   // Int8Array to achieve overflow
7.   return new Int8Array([ number ])[0]
8. }
9.
10. export function isSeqNumNewer(incomming: number, current: number): boolean {
11.   return withBitOverflow(twosComplementSubtract(incomming, current)) > 0;
12. }
```

Mit „isSeqNumNewer“ kann geprüft werden, ob die einkommende Nachricht („incomming“) neuer als die aktuell gespeicherte Sequenznummer („current“) ist. Meldet die Funktion „true“ zurück ist die einkommende Nachricht neuer und soll verwendet werden, wird „false“ zurückgegeben ist die gespeicherte Sequenznummer bereits gleich alt oder neuer und die einkommende Nachricht sollte ignoriert werden.

12. Metric/Hop Counts

Die Hop Count bzw. Metric gibt an, wie viele „Hops“ das Paket genommen hat.

Eine Node zählt jeweils direkt nach **Empfangen** die Metric um 1 hoch.

Wird eine Nachricht von N1 → N2 → N3 gesendet, sendet N1 eine Nachricht mit der Metric 0. Direkt nach empfangen zählt N2 diese Metric um 1 auf 1 hoch, verarbeitet die Nachricht evtl. und leitet sie an N3 weiter. N3 wiederholt den Vorgang und zählt direkt die Metric um 1 auf 2 hoch. Hierdurch besitzt das Paket am Ende den korrekten Hop Count von 2.

13. Route Errors

Sollte eine existierende Route nicht mehr genutzt werden können, findet die Node dies heraus, da keine Acknowledgements mehr empfangen werden (siehe Kapitel 10.1).

In diesem Fall sollte die Node Route Error Meldung an Nodes gesendet werden, bei denen bekannt ist, dass sie die Route nutzen.

Hierzu werden die „Precursors“ in der Routing Tabelle genutzt.

Existiert beispielsweise folgender Netzwerkaufbau:

```
N1 → N2 → N3 → N7 → N8 → N9 → N10

N4 → N5 → N6 →
```

Mit folgender Routing-Tabelle bei N8:

Destination	NextHop	Precursors	Metric	SeqNum	Is Valid
N9	N9	[N7]	1	1	True
N10	N9	[N7]	2	1	True
N1	N7	[N9]	4	1	True

Wird nun die Verbindung N8 → N9 getrennt, erkennt N8 dies, da keine ACK-Meldungen von N9 mehr empfangen werden.

N8 geht nun folgende Schritte durch:

- Aussuchen aller Routes in der Routing-Tabelle, bei denen der „NextHop“ die ausgefallene Node – also N9 – ist. Hieraus ergeben sich die Zeilen 1 und 2 der Tabelle

- N8 setzt „Is Valid“ bei allen diesen Routes auf „false“, damit diese nicht länger genutzt werden

- N8 sendet an alle „Precursors“ in den Listen eine RERR-Nachricht, dass diese Routes nicht mehr verfügbar sind.

In diesem Fall sendet N8 die folgende Nachricht an N7:

```
Type Flags   Hop Addr.   Prev. Hop
0010 0000    7 (N7)      8 (N8)

Path Count   Dest. Addr.   Dest. Seq.
2            9 (N9)        1

Dest. Addr.   Dest. Seq.   Padding
10 (N10)      1            00000000
```

N7 empfängt dieses Paket und geht die folgenden Schritte durch:

- Senden einer ACK-Nachricht an N8, um das Paket zu bestätigen

- N7 setzt in seiner eigenen Routing-Tabelle die Routes für N9 und N10 (beide Nodes, die in der RERR-Nachricht angegeben wurden) auf invalide

- Auch N7 sendet nun auch eine RERR Nachricht an alle Precursors der Routing-Table Einträge, die invalidiert wurden - in diesem Fall sind dies N6 und N3. N7 sendet also eine RERR-Nachricht an N6 und N3, mit der Info, dass N9 und N10 nicht mehr erreichbar sind

N3 und N6 empfangen diese RERR-Nachricht und gehen die selben Schritte durch.

Will später einer der Nodes eine Nachricht zu N9 senden, erkennt diese Node, dass keine valide Route mehr zu N9 existiert. Die Node sendet daher eine neue Route Request für N9, da sich evtl. eine neue Route ermöglicht hat, so dass N9 und weiterhin erreicht wird.

14. Flags

Die rechten 4 Bits des ersten Bytes sind „Flag Bits“. Bis jetzt gibt es keinen Anwendungsfall für die Flags, weshalb diese immer auf 0000 gesetzt werden.