

# R-Tron nRF24L01+ Network

## Designer Information

Ver 0.0

### Vorwort

In diesem Dokument wird die Funktionsweise verschiedener Komponenten des R-Tron nRF24L01+ Network beschrieben. Es ermöglicht die Konfiguration und Inbetriebnahme eines solchen Netzwerks.

### Inhalt

Überblick	3
Quickstart	4
Hardwareaufbau	4
Firmware	4
Linuxsystem als Server	4
Der erste Kontakt	6
Pseudo-Kommando F1.....	7
Pseudo-Kommando F0.....	9
Formate am Serialadapter	9
Serialadapter zu Server	9
Server zu Serialadapter	10
Format zwischen Serialadapter und Nods	10
Aufbau der Frames	10
Frame Server zu Serialadapter	11
Tabelle 1: Frame Server zu Serialadapter.....	11
Alle anderen Frames	11
Tabelle 2: andere Frames.....	11
Nodeadresse	11
Zuordnung der EEPROM Speicherzellen	12
Byte 0 bis B	12
Tabelle 3: Byte 0 bis B.....	12
Tabelle 4: Beispiel SetId (Byte 0, Wert DF).....	12
<i>Trägerfrequenzkanalindex:</i> .....	12
<i>Router:</i> .....	12
Tabelle 5: Routingbyte.....	12
<i>Sendepiegel:</i> .....	13
<i>MyAdr:</i> .....	13
<i>Main:</i> .....	13
<i>Spare:</i> .....	13
<i>Systemadresse:</i> .....	13
<i>Routingtafel:</i> .....	13
Byte C bis 1B (Systempasswort)	13

Byte 1C (TasterBlock)	14
Tabelle 6: TasterBlock.....	14
Byte 1D (PrePORTC)	14
Tabelle 7: PrePORTC.....	14
Byte 1E (PreDDRD)	14
Tabelle 8: PreDDRD.....	14
Byte 1F (PreACSR)	14
Tabelle 9: PreACSR.....	14
Byte 20 (PreADCSRA)	15
Byte 21 (PreADMUX)	15
Byte 22 bis 25	15
Tabelle 10: PreADLimits.....	15
Bytepaare ab 26 (direkt Aktion)	15
<b>Die Kommandocods</b>	<b>15</b>
00 (IO_Status) Server > Node	16
01 (IO_Tasten) Node > Server (Event)	16
Tabelle 11: Events.....	16
Tabelle 12: Taster Events.....	16
02 (IO_PortSet) Server > Node	16
Tabelle 13: PortSetCods.....	16
0E (IO_TStatus) Server > Node	17
0F (IO_ReadSet) Server > Node	17
Tabelle 14: IO_ReadSet Adressen.....	17
10 (IO_ADCSetup) Server > Node	17
Tabelle 15: IO_ADCSetup.....	17
11 (IO_ADCLimits) Server > Node	17
Tabelle 16: IO_ADCLimits.....	17
14 (IO_ACSetup) Server > Node	18
20 (IO_PWMSetup) Server > Node	18
21 (IO_PWMOC1A) Server > Node	18
22 (IO_PWMOC1B) Server > Node	18
24 (IO_PortPulse) Server > Node	18
94 (CG_ReadSet) Server > Node	19
98 (CG_SetSet) Server > Node	19
9C (NN_DefaultPass) Server > Node	19
9D (NN_UserPass) Server > Node	19
9E (NN_Addnode) Server > Node	19
AE (AU_RootKey) Server > Node (ab jetzt wird 4Byte Security verwendet)	19
F4 (NN_ReadInit) Server > Node (ab jetzt wird keine Security verwendet)	20
F8 (NN_SetInit) Server > Node	20
FC (AU_Reject) Node > Server	20
FE (AU_SendBackKey) Node > Server	20
FF (AU_RequestKey) Server > Node	20
<b>Topologie Beispiel</b>	<b>21</b>
Der Weg vom Server zum Node	22
Der Weg vom Node zum Server	25
<b>Impressum</b>	<b>25</b>

# Überblick

Adressierbarkeit von ca. 16 Millionen Nodes  
Nodes als Repeater oder Router konfigurierbar  
Routing über Haupt und Schutzwege ist möglich  
Integrierte Kommando Autorisierung sowohl Master->Node als auch Node->Master  
128-Bit Passwort.  
Das Passwort wird nur einmal durch das Netz bei der Ersteinrichtung übertragen  
Über jeden beliebigen Node kann ein neuer Node ein gebucht werden  
Neue Nodes müssen nicht vorkonfiguriert werden

Kleinster Repeater/Router Node läuft auf einem ATmega8  
Integrierte IO Funktionen:

- 5x Taster direkt anschließbar
- 26x Taster weitere Taster über Diodenmatrix
- Auf allen Tastern bis zu 7xKlick möglich
- 2x Fixe Outputs, auch als PWM verwendbar
- 6x frei einstellbare I/O
- Frei Einstellbarer Timer1
- Frei Einstellbarer ADC
- Frei Einstellbarer Analog Komparator
- Events für 31 Tasten, ADC und Analog Komparator
- Abfrage aller I/O über Polling vom Master möglich
- Master DDR und PORT force möglich, (High, Low, Toggel)
- Portpreset aus EEPROM möglich
- Direktzuweisung Event zu Output möglich (EEPROM gespeichert)

Serialadapter um das Netzwerk mit einen Steuerserver zu verbinden.

- Interface Serial 115,2 kBit x-on/x-off Adapter->Server
- Format HEX und ASCII Server->Adapter, nur HEX Adapter->Server

# Quickstart

## Hardwareaufbau

<https://arduino-info.wikispaces.com/Nrf24L01-2.4GHz-HowTo>

von der Beschreibung abweichend sind die folgenden Verbindungen:

- nRF24L01+ CE → Arduino D8 (ATMega8 PB0)
- nRF24L01+ CSN → Arduino D4 (ATMega8 PD4)
- nRF24L01+ IRQ → Arduino D3 (ATMega8 PD3)

Gaaaanz wichtig der 100µF Kondensator nahe bei der 3,3V Versorgung für den nRF24L01+ nicht vergessen.

Die meisten Arduinobords haben einen Serial2USB Wandler mit drauf. Der wird nur für den Serialadapter benötigt. Ich benutze einen CH340G, der von Linux auch wunderbar erkannt wird. Benutzt man z.B. einen Raspberry-PI als Server kann man dessen Serialport auch direkt mit dem Serialport des ATMega8 verbinden.

## Firmware

Dann einfach die Software aufspielen. Entweder die HEX-Files auf je einen ATMega8 Progen, LowFuse 0xDF HighFuse 0xCA. Da muss sonst nix gemacht werden.

- nRF24L01RootingNode.ATMega8.hex (Node Firmware)
- nRF24L01\_Serial.ino.ATMega8.hex (Serialadapter Firmware)

Oder selbst in Arduino erstellen und z.B. auf einen Arduino Nano Progen.

- nRF24L01RootingNode.ino (Node Firmware)
- nRF24L01\_Serial.ino (Serialadapter Firmware)

Es gehen alle Arduinos mit 16MHz und ATMega168 oder ATMega328. Beim Serialadapter muss der Puffer für die Serielle Schnittstelle im Arduinosystem verdoppelt werden.

z.B. C:\Program Files (x86)\Arduino\hardware\arduino\avr\cores\arduino\HardwareSerial.h

```
#define SERIAL_RX_BUFFER_SIZE 64
```

ändern in

```
#define SERIAL_RX_BUFFER_SIZE 128
```

Arduino muss danach neu gestartet werden.

## Linuxsystem als Server

Letztlich benötigt man einen Server, der Tag und Nacht läuft. Ein Windowssystem dafür zu verwenden ist wohl nicht die erste Wahl. Viele Bastler haben eh eine Linuxmaschine laufen, auf dem sich oft ein Fileserver befindet. Da kann man den Manager gut dazu packen.



## Der erste Kontakt

Nach dem alles eingerichtet und wie oben beschrieben angeschlossen wurde kann es los gehen. In K2 geben wir nun das folgende ein und beenden die Eingabe mit der Entertaste.

```
>nRF24L01Net: 0, ff, ff, f4
```

Auf K1 erhalten wir die folgende Antwort:

```
0, ff, ff, f4  
CF:00, FF, 0D, F4, FF, FF
```

Mit `0, ff, ff, f4` haben wir die Konfigurationsbytes aus den Serialadapter gelesen. `0, ff, ff` ist die Node Adresse für den Serialadapter. Auf die Systematik der Adresse wird später noch eingegangen. `f4` der Kommandocode zum Lesen der Konfiguration des Serialadapter.

Als Antwort bekommen wir `00, FF, 0D, F4, FF, FF`.

`00, FF` zeigt an, das es eine Antwort an den Server ist. `0D` ist die Anzahl der nun folgenden Bytes `F4` zeigt den Antworttyp, in diesem Fall die Antwort auf F4.

Die nachfolgenden 12xFF zeigen an, dass der Serialadapter noch unkonfiguriert ist. Also mal konfigurieren:

```
>nRF24L01Net: 0, FF, FF, F8, 0, DF, FF, FF, FF, B0, 0B, 1E, 50, 42, 41, 02, 03
```

`0, FF, FF` ist wieder die Node Adresse, `F8` das Kommando um den EEPROM des Serialadapter zu beschreiben. `0` die EEPROM-Startadresse, dann kommen die Daten.

`DF` Sendereinstellung

Bit 0-1: RF24\_PA; Bit 2:1Router/0Repeater Bit 3-7 Channel in 4-Step + 3

mehr dazu später.

`FF` LSB Adressteil des Serialadapter der Serialadapter am Server muss immer FF haben.

`FF` LSB Adressteil des Nachbarnode der meine Antworten zum Serverbringt (Main)

`FF` LSB Adressteil des Nachbarnode der meine Antworten zum Serverbringt (Spare)

Da unser Serialadapter direkt am Server hängt bleiben die unkonfiguriert.

`B0, 0B, 1E, 50` ist die Adresse des Netzwerks, die darf frei gewählt werden. Nur in ASCII gesehen darf sie nicht „INIT“ also `48, 4D, 48, 54` sein. Alle zum Netz gehörenden Nodes müssen hier die gleiche Adresse haben.

`42, 41, 02, 03` ist die Routingtabelle des Nodes. Dazu später mehr.

Die Antwort im K1 ist:

```
UD
```

Geben wir nun in K2 wieder

```
>nRF24L01Net: 0, ff, ff, f4
```

ein, so erhalten wir in K1 nun

```
0, ff, ff, f4
```

CF:00,FF,0D,F4,DF,FF,FF,FF,B0,0B,1E,50,42,41,02,03

Jetzt buchen wir unseren ersten jungfräulichen Node dazu  
Aufgebauten Node mit Strom versorgen. A0 bis A5 am Mega müssen nun High sein, da  
die PullUp aktiviert werden. Hält man den A4 wären eines Resets gegen GND so geht  
der Node auch in den Initialisierungsmodus selbst wenn er schon mal Initialisiert wurde.

Nun geben wir das Einbuchkommando 9E an den Serialmanager, Unser neuer Node  
soll die Adresse 40 bekommen. Das Kommando sieht dann so aus

>nRF24L01Net: 0,FF,FF,9E,40

Die Antwort im K1 ist:

FFX:00,FF,0A,F8,00,DF,40,FF,FF,B0,0B,1E,50

FFX: Besagt das an einen benachbarten Node mit der LSB Adresse FF (unkonfigurierter  
Node) das Nachfolgende erfolgreich (X) gesendet wurde. FFL: würde bedeuten der  
Sendeversuch war nicht erfolgreich.

Wie oben taucht auch in der Datensequenz F8 auf, sprich der EEPROM des neuen  
Nodes wurde beschrieben. Damit ist der Knoten ins Netz mit der LSB Adresse 40 ein  
gebucht.

Jetzt kommt die Security ins Spiel. Das Systempasswort eines frisch initialisierten  
Knotens ist immer 16xFF. Das Systempasswort steht im EEPROM ab Adresse 0C. Im  
Serialmanager ist das Systempasswort auf 4x3+12xFF voreingestellt. Das kann man  
gerne jetzt noch schnell mit einem Editor ändern. Dazu aber ihn erst mit stop anhalten.  
Ich benutze aber im folgenden das genannte Passwort.

## Pseudo-Kommando F1

Ich sende nun an meinen neuen Knoten 40 das Kommando F1. Mit dem Kommando F1  
macht der neue Node gar nichts aber der Serialmanager kennt damit den neuen Knoten  
und bekommt auch gesagt, dass das Passwort noch auf 16xFF steht. Warum die LSB  
Node Adresse 40 zweimal in der Nodeadresse auf taucht erkläre ich später.

>nRF24L01Net: 0,40,40,F1

Die Antwort im K1 ist dann:

0,40,40,F1

0,40,40,FF,8

T40X:00,40,01,F1

T40X:00,40,02,FF,08

RX:00,FF,12,FE,08,A7,49,ED,C9,05,C4,04,6E,FF,47,B6,42,23,95,0E,FF

Key,46,62,E4,9A,ED,70,C8,2E,9C,D0,CA,D0,F9,5B,90,46

0,40,40,AE,FD,AE,94,89,8E,A4,39,74,F3,D7,B0,74,D0,22,EB,ED,9,46,62,E4,9A

Key: 7EEBC4829D30CF0B046AA5D4BE79D03F

T40X:00,40,16,AE,FD,AE,94,89,8E,A4,39,74,F3,D7,B0,74,D0,22,EB,ED,09,46,62,E4,9  
A

T40X:00,40,02,FF,0A

RX:00,FF,12,FE,0A,D7,D5,85,52,2C,FF,B0,8E,34,F5,66,A6,38,02,CA,C8

Key,55,E5,8E,D0,F9,C4,BF,1C,B0,05,2F,08,BE,5E,E8,91

Autsch! Wie gut das der Serialmanager das für uns macht.

0,40,40,F1 War ja unser Kommando. Der Serialmanager fordert nun von Neuen Node

einen Parolekey an.

0,40,40,FF, 8

FF ist das Anforderungskommando 8 ist die *Anfrage ID*. Die muss nicht 8 sein. Der Node Antwortet mit

RX:00,FF,12,FE,08,A7,49,ED,C9,05,C4,04,6E,FF,47,B6,42,23,95,0E,FF

RX:00,FF Empfangene Daten für den Server, 12 Länge, FE Kenner für Parolekey 08 die ID der ursprünglichen Anfrage. A7,49,ED,C9,05,C4,04,6E,FF,47,B6,42,23,95,0E,FF der angeforderte Key.

Key, 46,62,E4,9A,ED,70,C8,2E,9C,D0,CA,D0,F9,5B,90,46

Das ist nun der aus dem Passwort 16xFF und dem Parolekey berechnete Autorisierungskey. Den hat der Serialmanager berechnet. Er wird für die Autorisierung von Kommandos Server->Node (S>N) benötigt. Nun muss noch ein Autorisierungskey für die Übertragung Node->Server (N>S) erzeugt werden. AE ist das Kommando dafür.

0,40,40,AE, FD,AE,94,89,8E,A4,39,74,F3,D7,B0,74,D0,22,EB,ED,9,46,62,E4,9A

Der Teil FD,AE,94,89,8E,A4,39,74,F3,D7,B0,74,D0,22,EB,ED ist der Parolekey dafür 9 ist die *Anfrage ID*. 46,62,E4,9A ist ein Teil des S>N Autorisierungskey. Einige Kommandos benutzen nur 4 Bytes des Autorisierungskey so eben auch dieses.

Key: 7EEBC4829D30CF0B046AA5D4BE79D03F

Das ist nun der aus dem Passwort 16xFF und dem Parolekey berechnete N>S Autorisierungskey. Er dient gleichzeitig als temporäre Nodeidentifikation.

T40X:00,40,16,AE,FD,AE,94,89,8E,A4,39,74,F3,D7,B0,74,D0,22,EB,ED,09,46,62,E4,9A

T40X: Zeigt wieder an, dass das Kommando erfolgreich übertragen wurde.

T40X:00,40,02,FF,0A fordert nun wieder einen Parolekey an, da ja Teile vom vorigen S>N Autorisierungskey verbraucht wurden.

RX:00,FF,12,FE,0A,D7,D5,85,52,2C,FF,B0,8E,34,F5,66,A6,38,02,CA,C8

Key, 55,E5,8E,D0,F9,C4,BF,1C,B0,05,2F,08,BE,5E,E8,91

Empfang von Parole und Berechnung des neuen S>N Autorisierungskey.

So nun setzen wir für den Node das Systempasswort. Da ja nur die ersten 4xFF zu 4x3 gemacht werden müssen sieht das dann so aus.

>nRF24L01Net: 0,40,40,98, C,3,3,3,3

Für die Initialisierten Nodes ist das Kommando zum EEPROM beschreiben nicht F8 sondern 98, sonst wird einfach ab EEPROM Adresse C 3,3,3,3 geschrieben, Damit ist das gesamte Passwort jetzt 4x3,12xFF.

Die Antwort sieht so aus

0,40,40,98,C,3,3,3,3#b,55,E5,8E,D0,F9,C4,BF,1C,B0,05,2F,08,BE,5E,E8,91

T40X:00,40,17,98,0C,03,03,03,03,0B,55,E5,8E,D0,F9,C4,BF,1C,B0,05,2F,08,BE,5E,E8,91

T40X:00,40,02,FF,0C

RX:00,FF,12,FE,0C,FE,A8,B8,85,90,BB,FA,79,53,01,13,64,ED,E7,80,04

Key, 7F,B4,74,94,6A,02,71,DE,90,A4,F2,88,4E,89,32,30

Schön zu sehen das ab #b die Autorisierung angehängt wird, b ist wieder die *Anfrage ID*, der Rest der zuvor erzeugte S>N Autorisierungskey. Darauf folgt wieder die Erzeugung eines neuen Key.

Mit dem Kommando 9D wird nun das neue Passwort aktiviert.

>nRF24L01Net: 0,40,40,9D

Die Antwort lautet.

```
0,40,40,9D#d,7F,B4,74,94,6A,02,71,DE,90,A4,F2,88,4E,89,32,30
T40X:00,40,12,9D,0D,7F,B4,74,94,6A,02,71,DE,90,A4,F2,88,4E,89,32,30
T40X:00,40,02,FF,0E
RX:00,FF,12,FE,0E,E3,4C,01,8C,43,1F,F8,AA,77,8C,FC,5B,5E,33,57,3F
Key,C0,5B,4D,8A,CC,8C,84,E9,EC,4B,1B,54,16,CB,70,1F
```

wer die Antwort nicht versteht bitte nochmal oben lesen.

## Pseudo-Kommando F0

Mit dem Kommando F0 wird nun dem Serialmanager mitgeteilt, dass er nun für den Node 0,40,40 das Systempasswort verwenden soll. Das Kommando kann auch für die Resynchronisation zwischen Server und Node verwendet werden. F0 ist auch ein Pseudokommando mit dem der Node nichts macht.

```
>nRF24L01Net: 0,40,40,F0
```

Die Antwort lautet.

```
0,40,40,F0
0,40,40,FF,f
T40X:00,40,01,F0
T40X:00,40,02,FF,0F
RX:00,FF,12,FE,0F,4C,87,5C,8C,F8,B1,F4,F4,CF,28,9D,6B,BA,FC,22,E0
Key,C0,68,F8,68,9D,8D,98,C0,E4,EE,C6,01,94,62,EB,46
0,40,40,AE,84,98,68,24,5E,39,BC,22,11,25,25,80,8C,D4,72,35,10,C0,68,F8,68
Key: E35461346FF083B3CB9941C814AD0E54
T40X:00,40,16,AE,84,98,68,24,5E,39,BC,22,11,25,25,80,8C,D4,72,35,10,C0,68,F8,68
T40X:00,40,02,FF,11
RX:00,FF,12,FE,11,68,8B,19,09,5E,77,CA,A8,FD,01,49,BB,04,50,96,A9
Key,39,B2,C2,55,DC,20,D5,06,02,E3,78,16,81,A8,4E,EB
```

Die Antwort solltet ihr euch nun selbst auf drösel können.

Nun fix eine LED an PB1 Arduino D9 mit Vorwiderstand an GND.

```
>nRF24L01Net: 0,40,40,24,7D,3D,10,7C,3C
```

Die Antwort lautet.

```
0,40,40,24,7D,3D,10,7C,3C#12,39,B2,C2,55,DC,20,D5,06,02,E3,78,16,81,A8,4E,EB
T40X:00,40,17,24,7D,3D,10,7C,3C,12,39,B2,C2,55,DC,20,D5,06,02,E3,78,16,81,A8,4E,EB
T40X:00,40,02,FF,13
RX:00,FF,12,FE,13,A2,10,44,F7,F3,EF,37,17,3D,E8,14,63,CF,C9,14,EF
Key,AB,CA,A7,BB,34,FF,24,4F,A1,13,AB,22,95,82,F6,D7
```

Wenn alles passt leuchtet die LED für 2 Sekunden auf.

## Formate am Serialadapter

### Serialadapter zu Server

Der Serialadapter sendet die Daten im Hexadezimalformat an den Server. Die Buchstaben ziffern sind immer groß geschrieben. Ein Wert ist immer 2 stellig und die

Werte sind durch Kommas getrennt. Das erleichtert das Mitlesen beim Debugging.

## Server zu Serialadapter

Vorweg, diese Datenrichtung besitzt keine Plausibilitätsprüfung. Egal was da eingegeben wird, es werden daraus Bytes berechnet.

In Grundstellung wird eine durch Kommas getrennte hexadezimale Zeichenfolge erwartet. Die Hexadezimalzahlen dürfen ein oder zweistellig sein, Großkleinschreibung ist beliebig möglich. Bei Werten mit mehr als 2 Stellen bleiben nur die letzten beiden Stellen gültig.

0,40,40,FF ist z.B. eine korrekt verarbeitbare Eingabe.

Mit einem : wird auf ASCII Eingabe umgeschaltet. Das ermöglicht z.B. komfortabel Daten an ein Display zu übertragen. Das kann dann so aussehen.

0,40,40,d0,0,0:Hallo Welt!

Mit einem # wird wieder in die Grundstellung geschaltet, will man im ASCII Modus ein # senden so gibt man ## ein. Das # kann man auch anstatt des Komma in der Grundstellung verwenden. Das @ hat ja den ASCII Code 0x40. Damit ist das folgende in der Eingabe gleich:

0:@@#fF ist gleich 0,40,40,FF

Jede Zeile wird mit Line Feed beendet. Das entspricht einem Byte auf der seriellen Schnittstelle mit dem Wert 0xA.

## Format zwischen Serialadapter und Nods

Zwischen Serialadapter und Nods und auch zwischen Node und Node werden die Daten Binär übertragen, also 0,40,1,FF benötigt vier Bytes.

## Aufbau der Frames

Die ersten 3 Bytes sind der Framehaeder.

Im 4. Byte steht immer das Kommando. Es maßgeblich vorgibt welche Länge die Autorisierung hat.

[Kommandos von 00 bis 9F](#) benötigen eine 16 Byte Autorisierung.

[Kommandos von A0 bis DF](#) benötigen eine 4 Byte Autorisierung.

[Kommandos von E0 bis FF](#) benötigen keine Autorisierung.

Jeder Frame vom Server zum Node enthält direkt nach der Kommandosequenz noch zusätzlich eine, ein Byte lange, Anfrage ID. Frames vom Node zum Server besitzen diese nur bei besonderen Kommandos.

Darauf folgt die Autorisierung.

Frames vom Node zum Server haben in der Regel eine 16 Byte Autorisierung die zugleich die temporär ID des Nodes ist.

Ein Frame darf insgesamt niemals länger als 32 Byte werden.

## Frame Server zu Serialadapter

Tabelle 1: Frame Server zu Serialadapter

1.Byte	2.Byte	3.Byte	4.Byte	(5.Byte)	...
Routingbyte	LSBAdr	Sendover	Kommando	(Daten...)	ID (& Aut)
0	40	41	xx	(xx)	...

Wenn im Sendover nicht die *MyAdr* (LSBAdresse) des Serialadapters steht, sendet der Serialadater den gesamten Frame an den Node mit der LSBAdresse in *Sendover*, hier an den Node mit der 41. Zuvor wird aber *Sendover* durch *Länge Daten* ersetzt, die der Serialadapter selbst berechnet.

## Alle anderen Frames

Tabelle 2: andere Frames

1.Byte	2.Byte	3.Byte	4.Byte	(5.Byte)	...
Routingbyte	LSBAdr	Länge Daten	Kommando	(Daten...)	((ID) & (Aut))
0	40	1A	xx	(xx)	...

*Länge Daten* ist die Anzahl der nach *Länge Daten* folgenden Bytes, wenn vorhanden, inklusive der *Anfrage ID* und der Autorisierung.

Nur wenn das *Routingbyte* 0 ist und *LSBAdr* = *MyAdr*, dann wird der Frame als eigener erkannt. In jedem anderen Fall wird der Frame repeatet oder geroutet.

## Nodeadresse

Bei Frames vom Server zum Serialadapter enthält der Framehaeder quasi nur die Adresse des Zielnodes. Der erste Ordner der Adresse ist logisch betrachtet *Sendover*, da *Sendover* den ersten Knoten nach dem Serialadapter bestimmt. Danach folgt das *Routingbyte*, es bestimmt wie das Paket durch das Netzwerk läuft. Erst wenn das *Routingbyte* durch die Verarbeitung der routenden Nodes zu 0 wurde, wird die *LSBAdr* berücksichtigt. Beginnt also ein solcher Frame mit 0,40,40 so wird der Node mit der logischen Adresse 40040 adressiert. Alle Nodes die Adressenmässig direkt nach dem Serialadapter kommen haben als ihre Adresse *MyAdr:0:MyAdr* bzw. *im Frame 0,MyAdr,MyAdr*.

# Zuordnung der EEPROM Speicherzellen

## Byte 0 bis B

Dies Speicherzellen werden sowohl vom Serialadapter als auch von den Nods verarbeitet.

Tabelle 3: Byte 0 bis B

0	1	2	3	4	5	6	7	8	9	A	B
SetId	MyAdr	Main	Spare	Systemadresse				Routingtabelle			
DF	40	FF	FF	B0	0B	1E	50	41	46	48	40

Tabelle 4: Beispiel SetId (Byte 0, Wert DF)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trägerfrequenzkanalindex					Router	Sendepiegel	
Dezimal 27					1	3	
1	1	0	1	1	1	1	1

### Trägerfrequenzkanalindex:

Der Kanal berechnet sich aus:

$$\text{Kanal} = \text{Trägerfrequenzkanalindex} \times 4 + 3 \quad (27 \times 4 + 3 = 111)$$

### Router:

Ist der *Router* = 1 dann wird das *Routingbyte* (Erstes Byte in jedem Frame) verarbeitet.

Tabelle 5: Routingbyte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Index 4. Router		Index 3. Router		Index 2. Router		Index 1. Router	
1		0		3		2	
0	1	0	0	1	1	1	0

Wenn das *Routingbyte* nicht 0 ist, dann nimmt er den Wert in *Index 1. Router* vom *Routingbyte*. Index 2. wird dann zu Index 1., 3 zu 2, 4 zu 3.

Den so veränderten Frame sendet er dann an den entsprechenden Node aus der Routingtabelle. In unserem Fall ist der Index 2, das wäre dann an den Node 48.

Wird in der Routigtabelle auf einen Wert gezeigt, der gleich *MyAdr* ist wird der Frame an

den Node in *LSBAdr* aus dem Frame gesendet. (EndPointSpeider)  
Ist das *Routingbyte* = 0 und *MyAdr* nicht gleich *LSBAdr* wird der Frame an den Node im Index 0 gesendet.  
Ist der *Router* = 0 (Repeater) wird der Frame nicht verändert und einfach an den Node im Index 0 gesendet. Hier dann Node 41.

### **Sendepiegel:**

Beim nRF24L01+ kann man 4 Sendepiegel einstellen 3 ist maximal, 0 ist minimal

### **MyAdr:**

*MyAdr* ist die LSBAdresse des Nodes, ist das *Routingbyte* gleich 0 und *MyAdr* = *LSBAdr* aus dem Frame, dann nimmt der Node die Daten für sich.

### **Main:**

Alle Frames die mit 00 FF beginnen werden an den in *Main* eingetragenen Node gesendet. Diese Frames müssen alle den Weg zum Server finden.

### **Spare:**

Scheitert das Senden eines 00 FF Frames an *Main* so wird dieser an den unter *Spare* eingetragenen Node gesendet.

### **Systemadresse:**

Diese 4 Bytes müssen auf allen Nodes gleich sein. Die Kodierung 484D4854 darf nicht verwendet werden. Sie ist die Grundeinstellung wenn ein Knoten nicht initialisiert ist.

### **Routingtabelle:**

[Siehe Oben.](#)

## **Byte C bis 1B (Systempasswort)**

Der Serialadapter nutzt diese und alle nachfolgenden Bytes nicht.  
In diesen 16 Bytes steht das *Systempasswort*.

## Byte 1C (TasterBlock)

Tabelle 6: TasterBlock

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(A7)	(A6)	A5	A4	A3	A2	A1	A0
1	1	1	1	1	1	1	1

Unkonfiguriert steht auf allen 1 und sind damit gesperrt. Wird ein Bit auf 0 gesetzt so kann der Node ein Event an den Server senden, wenn eine angeschlossene Taste gedrückt wird.

## Byte 1D (PrePORTC)

Tabelle 7: PrePORTC

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(A7)	(A6)	A5	A4	A3	A2	A1	A0
1	1	1	1	1	1	1	1

Unkonfiguriert sind alle PullUps eingeschaltet.

## Byte 1E (PreDDRD)

Tabelle 8: PreDDRD

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PD7	PD6	PD5	na	na	PD2	PD1	PD0
1	1	1	X	X	1	1	1

Unkonfiguriert sind alle Pins als Eingang geschaltet. Sonst ist dort wo eine 1 ist der Port ein Ausgang. FF macht das Gleiche wie 00.

## Byte 1F (PreACSR)

Tabelle 9: PreACSR

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ACD	ACBG	na	ACI	!Event	ACIC	ACIS1	ACIS0
1	1	X	1	0	1	1	1

Bitte die Beschreibung der Dokumentation zum ATmega8 entnehmen.

*!Event:*

*!Event* lässt sich nicht aktivieren. Kann aber vom Server aus gesetzt werden. Ist der Wert 1 so wird kein Event an den Server gesendet. ACD ist auch erst mal 1, damit wird auch kein Event ausgelöst.

## Byte 20 (PreADCSRA)

Ist der Wert gleich FF dann ist der AD-Wandler abgeschaltet. Sonst entspricht es der ADCSRA Dokumentation vom ATmega8, [ATmega8 Datenblatt](#), nur Bit 3 kann nicht gesetzt werden.

## Byte 21 (PreADMUX)

Entspricht der ADMUX Dokumentation vom ATmega8.

## Byte 22 bis 25

*Tabelle 10: PreADLimits*

Byte 22	Byte 23	Byte 24	Byte 25
PreADLowLimitH	PreADLowLimitL	PreADHighLimitH	PreADHighLimitL

PreADLowLimit:

Wird der Wert unterschritten so wird ein Event gesendet.

PreADHighLimit:

Wird der Wert überschritten so wird ein Event gesendet.

## Bytepaare ab 26 (direkt Aktion)

Es gehören immer 2 Bytes zusammen. Gelesen wird, bis das erste Byte den Wert FF hat.

Das Erst Byte enthält den [direkt Aktion Code](#) das Zweite den [PortSetCode](#).

Damit kann ein Event, ohne Zutun des Servers, den Status eines PortPin verändern.

## Die Kommandocods

Ein Kommandocode steht immer an auf den 4. Byte des Frames

## 00 (IO\_Status) Server > Node

Der Adressierte Node sendet folgendes zurück:

0,FF,1D,00,PINB, PORTB, DDRB, PINC, PORTC, DDRC, PIND, PORTD, DDRD, ACValue, ADValueH, ADValueL

## 01 (IO\_Tasten) Node > Server (Event)

Nach einem Event sendet der Node folgendes zurück:

0,FF,15,01,EventCode, Counter, ADValueH, ADValueL

Table 11: Events

EventCode	Counter	direkt Aktion	
02	0	10	Unterschreiten von ADLowLimit
03	0	18	Überschreiten vom ADHighLimit
04	0	20	AC wurde negativ
05	0	28	AC wurde positiv
Taster	Klicks	Taster x 8 + Kicks	

Table 12: Taster Events

Taster an	EventCode
A0	1E
A1	1D
A2	1B
A3	17
A4	0F

## 02 (IO\_PortSet) Server > Node

Keine Antwort vom adressierten Node

Nach dem Kommando können bis zu 11 PortSetCods angegeben werden.

Table 13: PortSetCods

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Toggel	DDR	Port (1=B,2=C,3=D)		Pin des Ports 0-7			Status

Wenn *Toggel* gestzt bitte *Status* immer 0 setzen. *DDR* = 0 schreibt auf *PORT*

## 0E (IO\_TStatus) Server > Node

Der adressierte Node sendet folgendes zurück:  
0,FF,13,0E,ICR1L,ICR1H (siehe ATmega8 Datenblatt)

## 0F (IO\_ReadSet) Server > Node

Nach dem Kommando gibt man die EEPROM Adresse an, ab der gelesen werden soll.

*Tabelle 14: IO\_ReadSet Adressen*

EEPROM Adresse	/// 00 - 1B ///	1C - FF	100 - 11B
Wert nach Kommando	/// ///	1C - FF	00 - 1B

Der adressierte Node sendet folgendes zurück:  
0,FF,1D,0F,[12 Bytes startend ab der angegebenen Adresse]

## 10 (IO\_ADCSetup) Server > Node

*Keine Antwort vom adressierten Node*

Nach dem Kommando müssen 3 Bytes angegeben werden.

*Tabelle 15: IO\_ADCSetup*

1.Byte	2.Byte	3.Byte
ADCSRA bevor ADMUX eingestellt wird	ADMUX	ADCSRA nachdem ADMUX eingestellt wurde

ADCSRA und ADMUX entsprechen der Beschreibung in der Dokumentation zum ATmega8. Bei ADCSRA kann das Bit 3 nicht gesetzt werden.

## 11 (IO\_ADCLimits) Server > Node

*Keine Antwort vom adressierten Node*

Nach dem Kommando müssen 4 Bytes angegeben werden.

*Tabelle 16: IO\_ADCLimits*

1.Byte	2.Byte	3.Byte	4.Byte
ADLowLimitH	ADLowLimitL	ADHighLimitH	ADHighLimitL

Wird vom AD-Wandler der Wert *ADLowLimitH:ADLowLimitL* unterschritten so tritt das Event 02:00 ein. Der [EventCode](#) dazu ist 10.

Wird vom AD-Wandler der Wert *ADHighLimitH:ADHighLimitL* überschritten so tritt das Event 03:00 ein. Der *EventCode* dazu ist 18.

## 14 (IO\_ACSetup) Server > Node

*Keine Antwort vom adressierten Node*

Nach dem Kommando muss ein Byte angegeben werden.

Das Byte entspricht der Dokumentation zum ATmega8 für das Register ACSR.

In ACSR kann das Bit 5 und das Bit 3 nicht gesetzt werden.

Das Bit 3 hat aber die Funktion von *!Event*.

Setzt man das Bit *!Event* auf 1. So werden keine Events vom Analog Komparator erzeugt. Ist das Bit 0, so werden folgende Events erzeugt.

Der Komparatorvergleich wird negativ so tritt das Event 04:00 ein. Der [EventCode](#) dazu ist 20.

Der Komparatorvergleich wird positiv so tritt das Event 05:00 ein. Der *EventCode* dazu ist 28.

## 20 (IO\_PWMSetup) Server > Node

*Keine Antwort vom adressierten Node*

Nach dem Kommando müssen 4 Bytes angegeben werden.

TCCR1A, TCCR1B, ICR1H, ICR1L (ICR wird vor TCCR geschrieben)

Die Bytes entsprechen der Dokumentation zum ATmega8 für die genannten Register.

## 21 (IO\_PWMOC1A) Server > Node

*Keine Antwort vom adressierten Node*

Nach dem Kommando müssen 2 Bytes angegeben werden.

OCR1AH, OCR1AL

Die Bytes entsprechen der Dokumentation zum ATmega8 für die genannten Register.

## 22 (IO\_PWMOC1B) Server > Node

*Keine Antwort vom adressierten Node*

Nach dem Kommando müssen 2 Bytes angegeben werden.

OCR1BH, OCR1BL

Die Bytes entsprechen der Dokumentation zum ATmega8 für die genannten Register.

## 24 (IO\_PortPulse) Server > Node

*Keine Antwort vom adressierten Node*

Nach dem Kommando müssen 5 Bytes angegeben werden.

*PortSetCode, PortSetCode, Wait, PortSetCode, PortSetCode*

Das Kommando führt die beiden ersten [PortSetCode](#) aus und führt dann *nach*

Wait x 0,128 Sekunden die letzten beiden *PortSetCode* aus. Will man einen *PortSetCode* nicht nutzen, so trägt man einen Code ohne Wirkung ein, z.B. 0.

## **94 (CG\_ReadSet) Server > Node**

Der adressierte Node sendet folgendes zurück:  
0,FF,1D,94,[Die 12 Bytes der Nodekonfiguration]

## **98 (CG\_SetSet) Server > Node**

*Keine Antwort vom adressierten Node*

Nach dem Kommando muss die Adresse der ersten zu beschreibenden EEPROM Zelle folgen danach folgt mindestens ein Datebyte höchstens 10. Damit kann im EEPROM Zelle 0 bis 109 beschrieben werden.

## **9C (NN\_DefaultPass) Server > Node**

*Keine Antwort vom adressierten Node*

Veranlasst den adressierten Node das Defaultpasswort (FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF) zu verwenden.

## **9D (NN\_UserPass) Server > Node**

*Keine Antwort vom adressierten Node*

Veranlasst den adressierten Node das Passwort welches im EEPROM 0C – 1B steht zu verwenden.

## **9E (NN\_Addnode) Server > Node**

*Keine Antwort vom adressierten Node*

Nach dem Kommando muss die zukünftige *MyAdr* des zu initialisierenden Nodes angegeben werden. Der adressierte Node sendet auf der Initialeinstellungseinstellung einen NN\_SetInit Frame, der den zu initialisierenden Knoten zu seinem Nachbarn macht und vergibt ihm die angegebene *MyAdr*.

## **AE (AU\_RootKey) Server > Node** (ab jetzt wird 4Byte Security verwendet)

*Keine Antwort vom adressierten Node*

Nach dem Kommando müssen die 16 Bytes der Server ParoleKeys folgen.  
Es wird der N>SAutorisierungskey synchronisiert.

## **F4 (NN\_ReadInit) Server > Node** (ab jetzt wird keine Security verwendet)

Ausschließlich nur zum lesen der Konfigurationsbytes des Serialadapters verwendbar.  
Der Serialadapter antwortet mit  
CF:00,FF,0D,F4,[Die 12 Bytes der Konfiguration des Serialadapters]

## **F8 (NN\_SetInit) Server > Node**

Wie Kommando 98, gibt es nur für den Serialadapter auf der Serielschnittstelle, oder nur netzintern zur Initialisierung eines Nodes.  
Der Serialadapter antwortet mit UD.

## **FC (AU\_Reject) Node > Server**

Ein Node meldet das ein Kommando wg. fehlerhaften Autorisierung abgewiesen wurde.  
Der Node sendet 0,FF,2,FC,Anfrage ID an den Server.

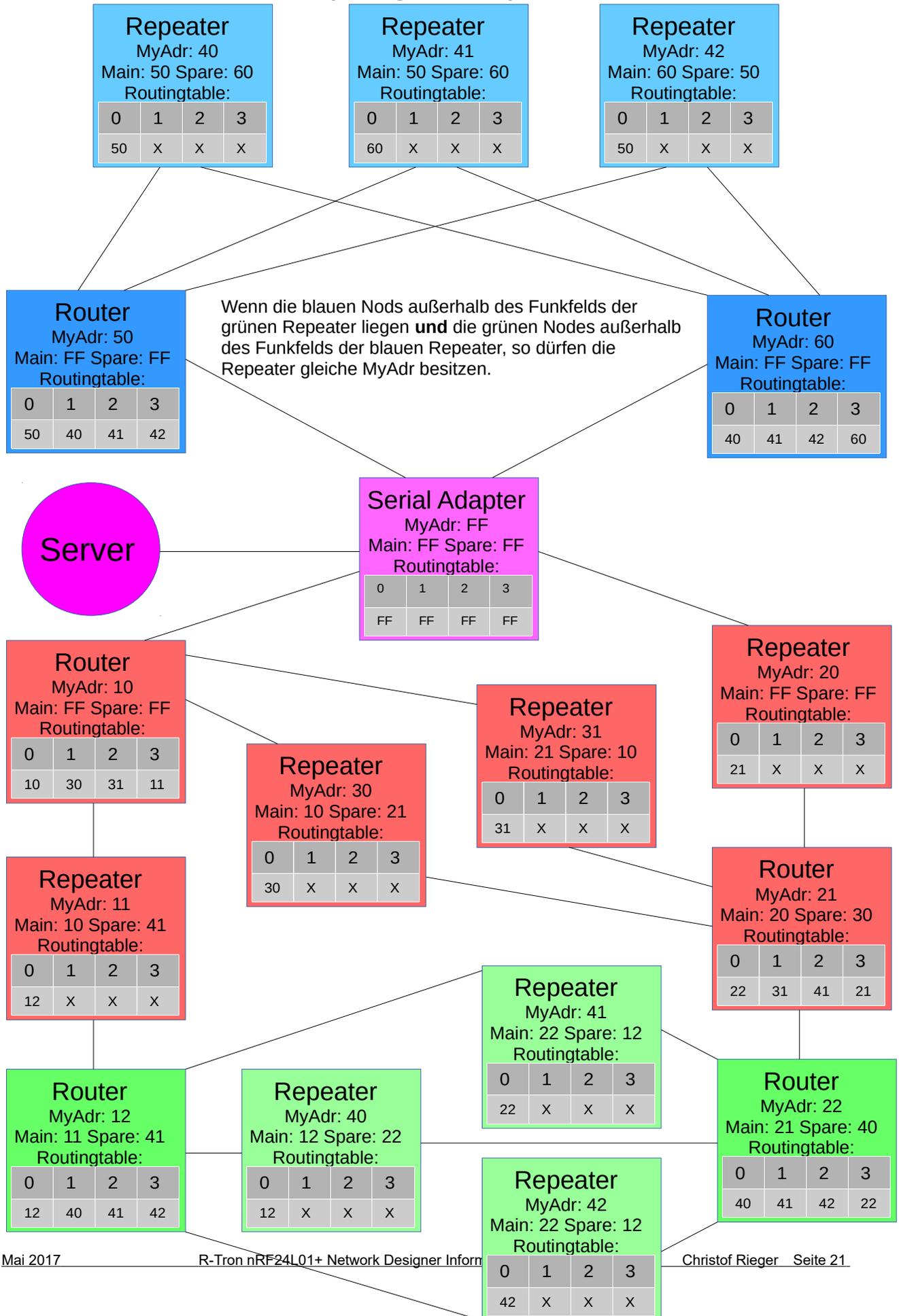
## **FE (AU\_SendBackKey) Node > Server**

Ein Node Antwortet damit auf das Kommando FF und sendet den NodeParoleKey  
0,FF,12,FE,Anfrage ID,16 Bytes des NodeParoleKeys  
Der Server synchronisieren den S>N Autorisierungskey

## **FF (AU\_RequestKey) Server > Node**

Der Server fordert vom Node ein NodeParoleKey an.  
Der adressierte Node synchronisieren den S>N Autorisierungskey und antwortet mit Kommando FE.

# Topologie Beispiel



Ich möchte nun an ein paar Beispielen die Funktionen des Routings und Repeating's erläutern. Dazu druckt ihr euch am besten das Topologie Beispiel aus.

## Der Weg vom Server zum Node

Wie ich schon beschrieben habe gibt es die logischen Adressen der Nodes. Die sich ja aus 3 Bytes zusammen setzt.

*ErsterNode\_nach\_Serialadapter:RoutingByte:MyAdr*

Im Frame ist die Reihenfolge etwas anders

*RoutingByte,MyAdr, ErsterNode\_nach\_SerialAdapter*

Das liegt daran, dass der *ErsterNode\_nach\_SerialAdapter* nach dem Serial Adapter nicht mehr gebraucht wird und diese Adressangabe durch die Längeninformation ersetzt wird.

Um es etwas anschaulicher zu machen wird das erste Byte (Routing Byte) nun nicht im Hexadezimalformat notier sondern in 4 2 Bit Gruppen also in DiBit Format. 9A wird also als 2122 dargestellt. Der Wert muss aber im Hexadezimalformat an den Serial Adapter übergeben werden.

Wir senden nun 0020,42,20,FF vom Server zum Serial Adapter.

Server → 0020,42,20,FF → Serial Adapter

Der Serial Adapter tauscht im Frame die 20 gegen die Längeninformation aus, in diesem Fall 01 und sendet den Frame an den roten Repeater mit der MyAdr 20.

Serial Adapter → 0020,42,01,FF → Repeater 20

Der Repeater erkennt, dass das Routingbyte nicht 0 ist und sendet somit den ganzen Frame an den Node der bei ihm in der Routingtabelle unter dem Index 0 eingetragen ist. Das ist hier der Router mit der MyAdr 21

Repeater 20 → 0020,42,01,FF → Router 21

Der Router erkennt ebenfalls, dass das Routingbyte nicht 0 ist, nur als Router nimmt er LSDiBit vom Routing Byte. Der Wert ist hier 0. Den so geänderten Frame sendet er damit an den Node er eben in der Routingtabelle beim Index 0 eingetragen ist. Also hier an den Router mit der MyAdr 22.

Router 21 → 0002,42,01,FF → Router 22

Im Router 22 passiert nun das Gleiche, nur dass er sich jetzt den Index 2 aus dem Routingbyte holt und damit nun den geänderten Frame an den Node mit der MyAdr 42 sendet.

Router 22 → 0000,42,01,FF → Repeater 42

Im Frame ist nun das Routingbyte 0 ab jetzt wird die Adresse 42 betrachtet. Der Repeater hat die MyAdr 42. Damit nimmt der Repeater 42 nun den Frame in die eigene Verarbeitung. Der Frame hat sein Ziel erreicht.

Wir senden nun mal 0030,42,20,FF

Server → 0030,42,20,FF → Serial Adapter  
Serial Adapter → 0030,42,01,FF → Repeater 20  
Repeater 20 → 0030,42,01,FF → Router 21  
Router 21 → 0003,42,01,FF → Router 22

Bis dahin läuft alles gleich. Der Router 22 benutzt nun die Adresse im Index 3. Der Router erkennt, dass dies seine eigene MyAdr ist. Das aktiviert den EndpointSpieder. Das bedeutet das der Router die LSBAdresse aus dem Frame zum Senden verwendet. Damit sendet er wiederum an den Node mit der MyAdr 42.

Router 22 → 0000,42,01,FF → Repeater 42

Wir sehen, der Repeater 42 erhält wieder den gleichen Frame, obwohl die Adresse anders war.

Nun bauen wir mal ein Tippfehler ein und schreiben ins Routingbyte 0010 anstatt 0020.

Server → 0010,42,20,FF → Serial Adapter  
Serial Adapter → 0010,42,01,FF → Repeater 20  
Repeater 20 → 0010,42,01,FF → Router 21  
Router 21 → 0001,42,01,FF → Router 22  
Router 22 → 0000,42,01,FF → Repeater 41

Da das Routingbyte 0 ist prüft der Repeater 41 ob die LSBAdresse gleich seiner MyAdr ist. Das ist aber nicht der Fall. So sendet er den Frame unverändert an den Node mit der MyAdr aus dem Index 0. In dem Fall wieder zum Router 22.

Repeater 41 → 0000,42,01,FF → Router 22

Eleganter Weiße hat sich aber seit dem letzten mal das Routingbyte verändert. Wir merken gerade, dass es keine gute Idee ist, in den Index 0 an einem Router einen Node einzutragen, der uns den Frame, wenn er nicht an ihn adressiert ist gerade wieder zurück sendet. Der Frame würde ständig hin und her geschickt werden und den

Funkbereich blockieren. Der Router sendet nun seinerseits den Frame an den Node mit der MyAdr aus dem Index 0. Also an den Repeater 40. Der Mechanismus für das entnehmen des Index vom Routingbyte wird auch hier angewendet. Nur aus 0 wird eben wieder 0.

Router 22 → 0000,42,01,FF → Repeater 40

Der Repeater 40 erkennt eben auch, dass es nicht sein Paket ist. Wie gehabt gemäß Index 0 geht das Paket zum Router 12.

Repeater 40 → 0000,42,01,FF → Router 12

Nun die Erlösung, die MyAdr im Index 0 ist die gleiche wie der Router 12 hat. Damit benutzt der Router 12 den EndpointSpider. Also sendet den Frame an den Node mit der MyAdr die gleich der LSBAdresse aus dem Frame ist.

Router 12 → 0000,42,01,FF → Repeater 42

Nach einer ziemlich Odysee ist der Frame tatsächlich noch am Ziel angekommen. Wir lernen, dass es Sinn macht, an einem Router auf dem Index 0 einen EndpointSpider einzurichten. Stößt ein falsch adressierter Frame mit einem abgearbeitetem Routingbyte auf solch einen Router, so wird der Frame entweder einem ZielNode zugestellt oder er geht ins Nirevana.

Jetzt noch eine Übung zum selbst nach vollziehen.

Server → 0033,42,10,FF → Serial Adapter

Serial Adapter → 0033,42,01,FF → Router 10

Router 10 → 0003,42,01,FF → Repeater 41

Repeater 41 → 0003,42,01,FF → Router 12

Router 12 → 0000,42,01,FF → Repeater 42

Das war jetzt der kürzeste 2.Weg zum grünen Repeater 42.

Noch ein Beispiel.

Server → 0323,21,10,FF → Serial Adapter

Serial Adapter → 0323,21,01,FF → Router 10

Router 10 → 0032,21,01,FF → Repeater 41

Repeater 41 → 0032,21,01,FF → Router 12

Router 12 → 0003,21,01,FF → Repeater 41

Repeater 41 → 0003,21,01,FF → Router 22

Router 22 → 0000,21,01,FF → Router 21

Über den EndpointSpider am Router 22 war der Router 21 erreichbar obwohl er nicht in der Routingtabelle vom Router 22 aufgeführt ist.

Wir sehen also, dass in einem so vermaschten Netz ein Node eine ganze Menge Adressen haben kann.

Wir haben ja das Kommando FF gesendet. Dies erwartet ja eine Antwort. Der Antwortframe eines Nods beginnt immer mit 0,FF. Jeder Node der hier 0,FF liest wird den Frame unverändert an den unter Main eingetragenen Node senden. Scheitert das, sendet er ihn an den unter Spare eingetragenen Node. Scheitert das auch, so wartet er eine Sekunde und versucht es erneut beim Main. Das macht er solange bis er selbst einen Frame bekommt oder ein Event ausgelöst wird, oder das senden eben erfolgreich war.

## Der Weg vom Node zum Server

Einige Beispiele, Repeater grün 42 sendet einen Frame.

42 → 22 → 21 → 20 → SerialAdaprer → Server

Node 22 ist nun ausgefallen.

42 → 22 scheitert !!!

42 → 12 → 11 → 10 → SerialAdaprer → Server

Node 21 ist nun ausgefallen.

42 → 22 → 21 scheitert !!!

22 → 40 → 12 → 11 → 10 → SerialAdaprer → Server

Node 20 ist nun ausgefallen.

42 → 22 → 21 → 20 scheitert !!!

21 → 30 → 10 → SerialAdaprer → Server

Der grüne Repeater 41 hat als Main 22 und als Spare 12 eingetragen. Das ist gar nicht so optimal, wenn die Funkfelder es zulassen wäre Main 21 und Spare 30 besser. Die Wege Server → Node und Node → Sever sind somit komplett unabhängig von einander planbar. In beiden Fällen ist darauf zu achten, dass keine Schleifen entstehen können.

## Impressum

Author:

Christof Rieger

Warthweg 6L

64823 Groß-Umstadt