Praktikum Kommunikationsnetze

Protokoll Block A

Gruppe 2

Termin Montag 8 - 12 Uhr

Shan Lin (208523) <lintanja@gmx.de>
Di Wang (208083) <didiwang@web.de>
Christian Richter (192548) <uni@ch-r.de>
Matthias Fleckenstein (215274) <magicmaze@t-online.de>

16. Mai 2004

Inhaltsverzeichnis

Zusammenfassung

Im Rahmen von ***Block A*** des Praktikums Kommunikationsnetze sollen die Eigenschaften eines realen Übertragungskanals am Beispiel eines Funkkanals analysiert werden. Hierbei geht es insbesondere um die untersten zwei Schichten des ISO-OSI-Referenzmodells.

Im Rahmen des ersten Termins steht die Bitübertragungsschicht (*Physical Layer*) im Mittelpunkt. Es wird u.a. die Abhängigkeit der Bitfehlerrate von verschiedenen Sendeleistungen und Modulationstechniken untersucht. Weiterhin werden unterschiedliche Möglichkeiten zur Leitungscodierung, sowie das Scrambling betrachtet.

Im zweiten Termin geht es dann vor allem um die Sicherungsschicht (*Data Link-Layer*). Es werden Methoden der Bit- und Framesynchronisation zwischen Sender und Empfänger analysiert und ihre Auswirkungen auf die Frameverlustrate diskutiert. In diesem Zusammenhang werden auch die Auswirkungen von Fehlerschutzmechanismen betrachtet.

Schließlich wird im dritten Termin das Aloha-Zugriffsverfahren exemplarisch für ein Mehrkanalzugriffsverfahren des MAC-Sublayers betrachtet. Ziel ist es, eine Aussage über die Effizienz des Aloha-Algorithmus zu treffen, indem ein Zusammenhang zwischen Last und Durchsatz hergestellt wird.

Kapitel 1

Erster Termin

1.1 Versuchsaufbau

Alle in diesem Protokoll beschriebenen Versuche haben folgenden Aufbau als Grundlage:

Als Sende- bzw. Empfangsstationen dienen Tranceiver der Firma CHIPCON, vom Typ **CC1020**. Diese sind über eine parallele Schnittstelle mit Laptops verbunden und werden über diese angesteuert. Die Stromversorgung erfolgt über ein ebenfalls angeschlossenes USB-Kabel.

Auf den Laptops ist als Betriebssystem Debian-Linux installiert und die Ansteuerung erfolgt über das eigens zur Verfügung gestellte Kommandozeilen-Tool ***ccconfig***. Vereinfacht lässt sich der Tranceiver auch über ein QT-basiertes grafisches User-Interface (***ccconfig-gui***) steuern.

Nach dem Start der GUI wird der Tranceiver durch den Button "*Init 4.8*" initialisiert. Dies wird vor jedem neuen Versuch wiederholt.

1.2 Frequenzscanner

Ziel dieses Versuches ist es, die Sendefrequenz eines im Versuchslabor stehenden Senders zu ermitteln. Dieser sendet ein relativ starkes Signal auf einer Frequenz im Bereich zwischen 870 und 878 MHz, so dass es sich anbietet, in diesem Frequenzbereich einen Scan durchzuführen.

Um diesen Frequenzbereich einzustellen, muss zuerst eine Mittelfrequenz f_m gesetzt werden. Alle weiteren Operationen beziehen sich dann auf diese Frequenz.

Der Befehl

```
ccconfig -sfrx A 874
```

setzt f_m für dem Empfänger (RX) auf 874 MHz.

Anschließend kann ein Scan um diese Mittelfrequenz herum durchgeführt werden. Die Schrittweite des Scans kann dabei frei gewählt werden. Da der Frequenzbereich relativ groß ist, bietet sich zunächst ein grober Scan des Bereiches an, um die Sendefrequenz ungefähr festzustellen.

Der Aufruf von

```
ccconfig -SC 9000 50 1
```

startet den Scan mit einer Schrittweite von 50 in einem Bereich von 9000 symmetrisch um f_m herum. Die Angaben 9000 und 50 beziehen sich dabei nicht auf reale physikalische Größen, sondern auf sogenannte FIELD-Werte, die direkt in die Register des Chips geschrieben werden. Die 1 steht für eine Testdauer von 10 ms pro Frequenzschritt.

Der Scan liefert das in Abbildung 1.1 (etwas gekürzt) gezeigte Ergebnis. Es ist ein deutlicher Ausschlag der Sendeleistung bei 871,25 MHz zu erkennen. Es liegt also nahe, dass der Sender in diesem Bereich sendet.

```
ARX[MHz]:871.064636, [F]:1910440, FOff[MHz]: -0.0015,  RSSI: 22  ##########
                         BER(modelB):0.532995, totalBE:     105, totalB:      197
...
ARX[MHz]:871.253601, [F]:1910860, FOff[MHz]: -0.0024, RSSI: 41 ####################
                         BER(modelB):0.065657, totalBE: 13, totalB: 198
...
ARX[MHz]:872.357483, [F]:1913313, FOff[MHz]: -0.0009,  RSSI: 14  #######
                         BER(modelB):0.500000, totalBE:       25, totalB:       50
ARX[MHz]:872.379944, [F]:1913363, FOff[MHz]: -0.0012,  RSSI: 15  ########
                         BER(modelB):0.500000, totalBE:       27, totalB:       54
ARX[MHz]:872.402466, [F]:1913413, FOff[MHz]:  0.0012,  RSSI: 15  ########
                         BER(modelB):0.574074, totalBE:       31, totalB:       54
ARX[MHz]:872.424988, [F]:1913463, FOff[MHz]: -0.0012,  RSSI: 14  #######
                         BER(modelB):0.444444, totalBE:       24, totalB:       54
...
ARX[MHz]:880.344971, [F]:1931063, FOff[MHz]:  0.0003,  RSSI: 15  ########
                         BER(modelB):0.481481, totalBE:       26, totalB:       54
ARX[MHz]:880.367432, [F]:1931113, FOff[MHz]: -0.0006,  RSSI: 13  #######
                         BER(modelB):0.482143, totalBE:       27, totalB:       56
ARX[MHz]:880.389954, [F]:1931163, FOff[MHz]:  0.0003,  RSSI: 14  #######
                         BER(modelB):0.444444, totalBE:       24, totalB:       54
ARX[MHz]:880.412476, [F]:1931213, FOff[MHz]: -0.0012,  RSSI: 14  #######
                         BER(modelB):0.518519, totalBE:       28, totalB:       54
```

Abbildung 1.1: Ergebnis Grobscan

Um das genauer zu überprüfen, setzen wir die Mittelfrequenz auf diesen Wert

```
ccconfig -sfrx A 871.25
```

und führen einen feiner aufgelösten Scan in einem kleinen Bereich um die neue Mittelfrequenz herum durch.

```
ccconfig -SC 500 2 4
```

Das (gekürzte) Ergebnis in Abbildung 1.2 zeigt deutlich, dass die Sendefrequenz in einem Bereich um **871,22882MHz** maximal ist. Der Sender scheint also auf dieser Frequenz zu arbeiten.

```
...
ARX[MHz]:871.220337, [F]:1910786, FOff[MHz]:  0.0042,  RSSI: 61  ##############################
         BER(modelB):0.076142, totalBE:       15, totalB:     197
ARX[MHz]:871.221191, [F]:1910788, FOff[MHz]:  0.0036,  RSSI: 62  ##############################
         BER(modelB):0.076142, totalBE:       15, totalB:     197
ARX[MHz]:871.222107, [F]:1910790, FOff[MHz]:  0.0030,  RSSI: 63  ##############################
         BER(modelB):0.111675, totalBE:       22, totalB:     197
ARX[MHz]:871.223022, [F]:1910792, FOff[MHz]:  0.0021,  RSSI: 63  ##############################
         BER(modelB):0.060914, totalBE:       12, totalB:     197
ARX[MHz]:871.223938, [F]:1910794, FOff[MHz]:  0.0009,  RSSI: 63  ##############################
         BER(modelB):0.055838, totalBE:       11, totalB:     197
ARX[MHz]:871.224792, [F]:1910796, FOff[MHz]: -0.0003,  RSSI: 63  ##############################
         BER(modelB):0.060914, totalBE:       12, totalB:     197
ARX[MHz]:871.225708, [F]:1910798, FOff[MHz]: -0.0003,  RSSI: 63  ##############################
         BER(modelB):0.060914, totalBE:       12, totalB:     197
...
```

Abbildung 1.2: Ergebnis Feinscan

1.3 Scrambling

Ein Scrambler dient dazu, die Synchronisation bei der Übertragung zwischen Sender und Empfänger zu erleichtern. Er vermeidet konstante Pegel, indem er die Bit-Folge durch “Verwürfeln” in ein Pseudo Noise-Signal (PN-Signal) umwandelt.

Ziel dieses Versuches ist es, herauszufinden, was passiert, wenn nur einseitig (Sender oder Empfänger) gescrambelt wird.

Zunächst wird auf beiden Seiten ohne aktivierten Scrambler gesendet bzw. empfangen. Anschließend wird senderseitig der Scrambler eingeschaltet, während der Empfänger weiterhin ohne aktivierten Scrambler arbeitet. Als nächstes wird nun auch beim Empfänger der Scrambler eingeschaltet.

Während der Durchführung wird ständig die Bitfehlerrate (BER) überwacht, um Änderungen des Übertragungsverhaltens festzustellen. Da der Sender ständig Nullen emittiert, ist jede am Empfänger auftretende Eins ein Bit-Fehler. Die ausgegebene BER ist daher in diesem Fall das Verhältnis von Einsen zu den insgesamt übertragenen Zeichen (Nullen und Einsen).

1.3.1 Vorbereitung

Bei diesem Versuch werden die Transceiver in einem Abstand von ca. 1 m aufgestellt. Nachdem die Transceiver initialisiert und kalibriert sind, werden auf beiden Modulen die Mittelfrequenzen gesetzt

```
ccconfig -sfrx A 865
ccconfig -sftx A 865
```

Bei den 865 MHz handelt es sich um eine unserer Gruppe fest zugeteilte Frequenz. Da jede Gruppe eine eigene Mittelfrequenz zugeteilt bekommen hat, wird durch Frequenzmultiplex eine gegenseitige Störung ausgeschlossen.

Die beiden `ccconfig`-Aufrufe enthalten verschiedene Parameter (`sftx`, `sfrx`). Auf der Senderseite wird durch Aufruf mit `-sftx` die Sendefrequenz und auf der Empfängerseite durch `-sfrx` die Empfangsfrequenz gesetzt.

1.3.2 Durchführung

Nachdem nun Sender und Empfänger für die Übertragung eingerichtet sind, wird die Bit Error Rate (BER) durch den Befehl

```
ccconfig -ls 10
```

gemessen. Da die Übertragungsstrecke mit 1 m relativ kurz ist, liegt sie erwartungsgemäß nahe Null.

Nach dem Einschalten des Scrambler auf Senderseite steigt die BER auf ca. 0,5 an. Das bedeutet, dass Nullen und Einsen praktisch gleich häufig auftreten. Da der Scrambler auf der Empfängerseite ausgeschaltet ist, wird die "Verwürfelung" nicht rückgängig gemacht, so dass wir den Datenstrom sehen, so wie er im Kanal übertragen wird. Dies entspricht dem erwarteten Ergebnis, da der Scrambler, seiner Funktionsweise gemäß, eine Pseudo-Noise Sequenz erzeugen soll.

Wird nun auch beim Empfänger der Scrambler aktiviert, so fällt die BER wieder auf einen Wert nahe Null ab, da nun die PN-Sequenz wieder zurücktransformiert werden kann.

Schlussfolgernd lässt sich sagen, dass es *sehr* wenig Sinn macht, den Scrambler nur an einem Gerät zu aktivieren. Soll tatsächlich eine Datenübertragung stattfinden, so müssen Sender und Empfänger identisch konfiguriert sein (Scrambler ein oder aus).

1.4 BER bei verschiedenen Übertragungsraten

Bei einer Übertragung gibt es etliche Parameter, welche man am Sender bzw. Empfänger variieren kann, und die Einfluss auf das Übertragungsverhalten haben. Zwei dieser Parameter sind Sendeleistung und Übertragungsrate (Baudrate).

Die Sendeleistung – angegeben in dBm – regelt den HF-Verstärker des Transmitters so, dass dieser eine festgelegte HF-Leistung als Ausgangsleistung abstrahlt. Für unsere Messungen, verwenden wir Ausgangsleistungen im Bereich von -40 dBm bis +5 dBm.

Die Baudrate gibt die Anzahl der Signalpegelwechsel pro Sekunde an. Bei einer binären NRZ-Übertragung entspricht sie gleichzeitig der Bitrate (Bit/s), bei Manchester-Codierung (2 Baud/Bit) jedoch nur der halben Bitrate. Sie muss bei Sender und Empfänger identisch eingestellt sein.

In diesem Experiment soll nun versucht werden, den Einfluss der Baudrate auf die Bitfehlerrate der Übertragung zu bestimmen. Dazu werden mehrere Übertragungen mit jeweils unterschiedlichen Baudraten initiiert und die Bitfehlerrate, sowie der empfangene Eingangsleistungspegel (*RSSI, **R**eceived **S**ignal **S**trength **I**ndicator*) in Abhängigkeit von der Sendeleistung bestimmt.

1.4.1 Vorbereitung

Sender und Empfänge werden in einem Abstand von etwa 2 m voneinander aufgestellt. Um bei dieser geringen Entfernung trotzdem genügend Bitfehler für eine aussagekräftige Messung zu erhalten, werden die Antennen der Geräte vorsichtig demontiert.

Nachdem beide Geräte initialisiert wurden, können die Mittelfrequenzen für Sender und Empfänger wie in Abschnitt 1.3.1 beschrieben eingestellt werden. Anschließend wird in der GUI die zur Übertragung verwendete Modulation für beide Stationen auf FSK (*Frequency Shift Keying*, siehe Abschnitt 1.5) gesetzt und eine Kalibrierung durchgeführt.

1.4.2 Durchführung

Zunächst werden Sender und Empfänger mittels GUI auf eine Übertragungsrate von 4,8 kBaud gesetzt. Nun wird die Sendeleistung des Transmitters schrittweise von -40 dBm auf 5 dBm erhöht. Dies kann zwar ebenfalls über die GUI erledigt werden, bei derartig vielen Messungen ist es jedoch komfortabler und schneller, das Kommando

```
ccconfig -spr dBm
```

zu benutzen. `dBm` steht für die gewünschte Leistung. Nicht alle dBm-Werte sind direkt einstellbar. Wir verwenden folgende Werte:

```
-40, -35, -25, -18, -16, -14, -12, -11, -10, -9,
-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5
```

Um die Bitfehlerrate (BER) und die dazugehörige Eingangsleistung (RSSI) am Empfänger zu messen werden, wie in Abschnitt 1.3 beschrieben, die Übertragungsparameter mittels

```
ccconfig -ls 10
```

ausgegeben. Hier lassen sich beide Werte ablesen.

1.4.3 Auswertung

Die Ergebnisse der Messreihen sind in Abbildung 1.3 und 1.4 zu sehen[1]. In beiden Diagrammen ist die BER bei verschiedenen Übertragungsraten angegeben. Einmal im Vergleich zur gesendeten, und einmal zur tatsächlich empfangenen Leistung.

BER im Vergleich zu gesendeten Leistung

Bild 1.3 stellt die Fehlerrate bei ansteigender Leistung des Senders dar. Wie zu erwarten war, fällt die Bitfehlerrate mit zunehmender Sendeleistung ab. Das bedeutet, dass sich die Qualität der Übertragung verbessert, je stärker das Signal ist, welches der Sender emittiert.

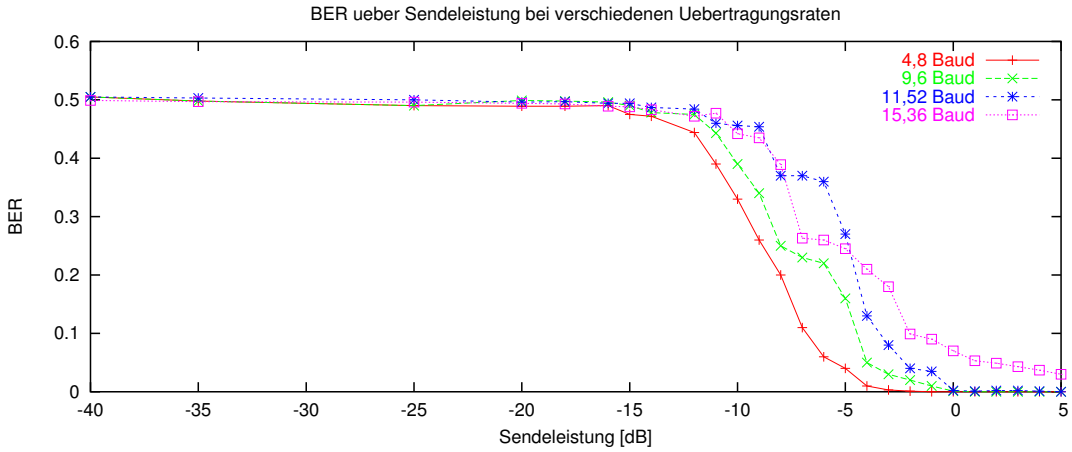


Abbildung 1.3: BER über Sendeleistung bei verschiedenen Übertragungsraten

Bei näherer Betrachtung scheint dies durchaus logisch. Je geringer die Sendeleistung ist, desto kleiner ist die Amplitude des Signals und desto stärker wirken sich Störungen auf das Signal aus. Nun ist die FSK-Modulation gegenüber Amplitudenveränderungen wesentlich robuster als beispielsweise das ASK (siehe auch Abschnitt 1.5.3) aber

[1] zugrundeliegende Messwerte im Anhang (Tabelle A.1)

trotzdem kann das Signal derart verfälscht werden, dass der Entscheider im Empfänger eine falsche Zuordnung trifft und damit einen Bitfehler produziert. Bei starken Signalen fällt die Veränderung durch Störungen viel geringer aus, so dass die Bitfehlerrate sinkt.

Weiterhin fällt auf, dass die BER um so schneller abfällt, je geringer die Übertragungsrate ist. Bei 4,8 kBaud ist ein signifikanter Abfall schon ab -15 dBm zu erkennen, während die Bitfehlerrate bei einer Übertragung mit 15,36 kBaud erst bei -10 dBm deutlich zu sinken beginnt. Auch ist die BER bei 4,8 kBaud bereits ab -3 dBm praktisch Null, während die 15,36 kBaud-Übertragung bis über 5 dBm hinaus noch deutlich darüber liegt.

Eine leichte Abweichung von der Theorie ist lediglich bei der 15,36 kBaud-Übertragung zu erkennen. Dort liegt die BER bei Sendeleistungen um -5 dBm herum untypisch niedrig, was wahrscheinlich auf Messungenauigkeiten zurückzuführen ist.

BER im Vergleich zur empfangenen Leistung

In Bild 1.4 ist die Bitfehlerrate über dem am Empfänger ermittelten RSSI-Wert aufgetragen. Dieser Wert ist (bis auf eine Konstante) näherungsweise proportional zu der am Empfänger detektierten HF-Leistung. Es gilt

$$P_{HF} \approx 1,5 \cdot RSSI - 3k_{VGA}$$

wobei k_{VGA} die in dBm gemessene Verstärkung des Variable-Gain-Amplifiers (VGA) des Empfängers ist. Bei konstanter Verstärkung ergibt sich also ein linearer Zusammenhang zwischen HF-Eingangsleistung und RSSI-Wert.

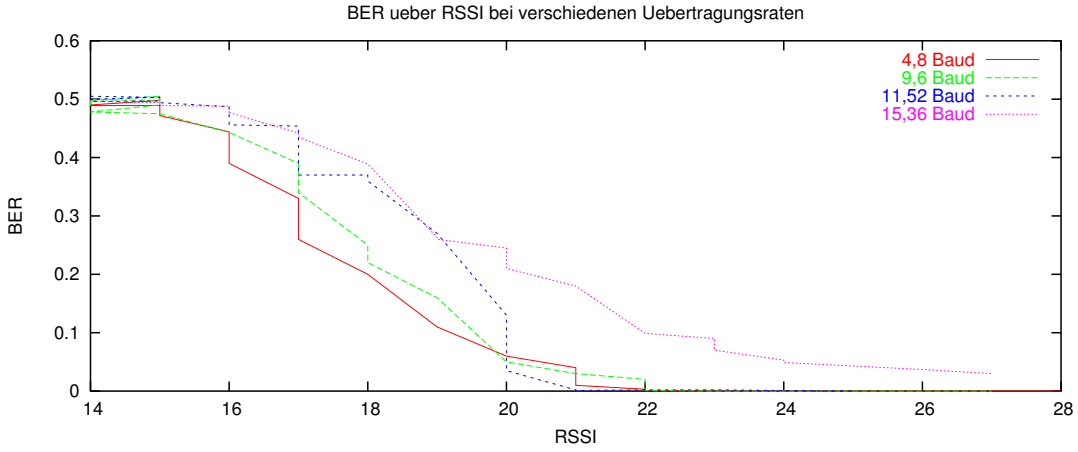


Abbildung 1.4: BER über Empfangsleistung bei verschiedenen Übertragungsraten

Auch hier ist ein Absinken der BER mit steigendem RSSI (und damit auch mit steigender Eingangsleistung) sichtbar. Wie in Abbildung 1.3 fällt die Fehlerrate um so

schneller, je geringer die Baudrate der Übertragung ist. Bei schnellen 15,36 kBaud liegt sie selbst bei einem RSSI-Wert von 27 noch deutlich über Null, während bei vergleichsweise langsamen 4,8 kBaud schon ab RSSI=22 praktisch keine Bitfehler mehr zu verzeichnen sind.

Das Beziehen der Fehlerrate auf die tatsächlich empfangene Leistung hat gegenüber der reinen Sendeleistung als Bezugsmaß den Vorteil, dass weniger Einflußfaktoren zu berücksichtigen sind. Empfangene und gesendete Leistung sind (beim idealen Rundstrahler) über die Gleichung

$$P_r = P_s \cdot \frac{\lambda^2}{(4\pi)^2 \cdot d^\alpha}$$

miteinander verknüpft. Das bedeutet, dass sowohl die Wellenlänge λ, als auch die Dämpfung der Übertragungsstrecke ($\alpha = 2$ im Freiraum) Einfluss auf die empfangene Leistung haben. Bei nicht idealen Rundstrahlern sind zusätzlich noch Form und Standort der Antenne zu berücksichtigen. All das fällt bei der Betrachtung der reinen Empfangsleistung weg. Hier zählt nur die Leistung, die dem Empfänger (und damit der Entscheiderstufe) auch wirklich zur Verfügung stehen und damit maßgeblich für die Bitfehlerrate sind.

1.5 BER bei verschiedenen Modulationsverfahren

Es gibt grundsätzlich mehrere Möglichkeiten, digitale Nachrichtensignale auf einen Träger zu modulieren. Die beiden hier betrachteten Verfahren sind ASK (*Amplitude Shift Keying*) und FSK (*Frequency Shift Keying*).

Bei ASK handelt es sich um die Amplitudenmodulation eines Trägersignals mit einem digitalen modulierenden Signal (Basisbandsignal). Die Trägerfrequenz bleibt bei diesem Verfahren konstant, die Trägeramplitude ändert sich.

FSK ist eine Frequenzmodulation mit mehrerer Frequenzen. Die Frequenz des Trägers wird entsprechend dem Datensignal zwischen verschiedenen diskreten Frequenzen verändert. Bei einem Binärsignal beispielsweise repräsentiert eine Frequenz die digitale "Eins", die andere die digitale "Null".

Um die Güte eines Modulationsverfahrens zu beurteilen, bietet sich die Messung der Bitfehlerrate an. In diesem Versuch soll untersucht werden, wie sich ASK und FSK hinsichtlich des Gütekriteriums BER verhalten.

1.5.1 Vorbereitung

Sender und Empfänger werden zunächst wieder durch Aufrufen von "Init 4.8" initialisiert. Wie in Abschnitt 1.3.1 beschrieben, wird an Sender und Empfänger die zugewiesene Trägerfrequenz gesetzt und anschließend werden beide in den Sende- bzw.

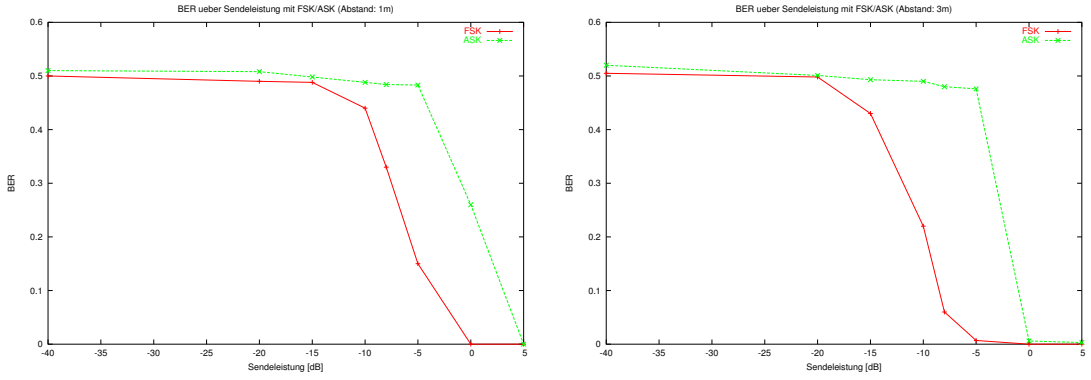
Empfangsmodus geschaltet. Zu beachten ist, dass die Antenne des Senders vor dem Versuch vorsichtig demontiert wird.

1.5.2 Durchführung

Die Messreihen werden für Sender-Empfänger-Abstände von 1 m und 3 m aufgenommen. Zuerst werden beide Stationen in den ASK-Modus geschaltet. Wieder werden Messreihen für Sendeleistungen von -40 dBm bis 5 dBm aufgenommen. Für jede eingestellten Sendeleistung wird die BER gemessen. Dies geschieht, wie in Abschnitt 1.4.2 beschrieben durch den Aufruf von `ccconfig -ls 100`. Anschließend werden Sender und Empfänger in den FSK Modus geschaltet und Messreihen für die gleichen Sendeleistungen wie unter ASK aufgenommen. Analog zur ASK-Messung wird auch hier zu jeder Sendeleistung die BER beobachtet.

1.5.3 Auswertung

Aus den Diagrammen[2] in Bild 1.5wird ersichtlich, dass FSK ab einer Sendeleistung von ca. -15 dBm bei 1 m Abstand deutlich weniger Bitfehler erzeugt als ASK. Beim ASK erreicht die gemessene BER erst bei einer Sendeleistung von ca. 5dBm einen Wert von annähernd Null. Beim FSK wird dieser Wert bereits bei einer Sendeleistung von ca. 0 dBm erreicht.



(a) 1m Abstand

(b) 3m Abstand

Abbildung 1.5: BER über Sendeleistung bei FSK- und ASK Modulation

Das bei einem Abstand von 3m die gemessene BER bei gleicher Sendeleistung geringer ist als bei 1m Abstand ist höchstwahrscheinlich auf Ungenauigkeiten in den Messungen, hervorgerufen durch äußere Störungen, zu erklären. Interessant ist aber, dass

[2] zugrundeliegende Messwerte im Anhang (Tabelle A.2)

bei beiden Grafiken, FSK gegenüber ASK die geringeren Fehlerraten bei gleicher Sendeleistung aufweist. Das wird damit zusammenhängen, dass beim ASK die *Amplitude* der Trägerschwingung im Rhythmus des Bitstroms verändert wird. Am Empfänger müssen diese Amplitudenänderungen erkannt werden, um den Bitstrom zu rekonstruieren. Da wir es hier mit einem Funkkanal zu tun haben, wird das übertragene Signal durch Störungen überlagert, die die Amplitude des übertragenen modulierten Signals beeinflussen. Durch diese Störungen wird die Unterscheidung zwischen den einzelnen Amplituden am Empfänger erschwert. Treten sehr starke Störungen auf, so kann die Amplitude sogar so stark verändert werden, dass das falsche Bit am Empfänger rekonstruiert wird.

Die FSK-Modulation arbeitet wie oben erwähnt durch Änderung der Momentanfrequenz des Signals, um den Bitstrom am Empfänger zu rekonstruieren. Störungen der Amplitude wirken sich (solange diese nicht Null wird) somit nicht auf den rekonstruierten Bitstrom aus. Außerdem sind Überlagerungen im Frequenzbereich weniger problematisch.

1.6 NRZ- und Manchester-Codierung

Es gibt grundsätzlich viele Möglichkeiten Bits, die über eine Leitung übertragen werden, zu codieren. Bei der einfachen NRZ-Codierung werden "Nullen" und "Einsen" lediglich durch unterschiedliche Spannungspegel auf der Leitung codiert. Dies kann zu Problemen bei der Synchronisation zwischen Sender und Empfänger führen, wenn lange konstante Pegel übertragen werden. Eine Lösung dieses Problems bietet der Manchester-Code. Bei diesem Schema ist gewährleistet, dass jede Bitperiode eine Transition in der Mitte hat, was dem Empfänger die Synchronisation mit dem Sender erleichtert.

Ziel dieses Versuchs ist es, das Funktionsprinzip von NRZ- und Manchester-Codierung zu veranschaulichen.

1.6.1 Vorbereitung

Sender und Empfänger werden durch Aufrufen von "Init 4.8" initialisiert. Wie oben beschrieben wird an Sender und Empfänger die zugewiesene Trägerfrequenz gesetzt und anschließend werden Sender und Empfänger in den Sende- bzw. Empfangsmodus geschaltet. Zusätzlich wird hier der Scrambler auf der Empfänger- und Senderseite deaktiviert. Die Messungen erfolgen wieder ohne Sendeantenne.

Außerdem wird das Protokoll `RxAutoTx` aktiviert, damit nur dann Ausgaben in die Gerätedatei des Empfängers erfolgen, wenn Daten des Senders empfangen wurden. Es ist meist zusätzlich erforderlich, den CS-Level so zu korrigieren, dass erst dann Daten empfangen werden, wenn der Sender auch wirklich ein Bitmuster sendet.

1.6.2 Durchführung

Für den ersten Messdurchgang wird der Sender auf Manchester-Codierung und der Empfänger auf NRZ-Codierung geschaltet

Am Sender wird durch den Befehl

```
echo [Bitmuster] > /dev/plccx
```

ein Bitmuster gesendet. Am Empfänger wird das empfangene Bitmuster mit

```
cat /dev/plccx
```

angezeigt.

Für die zweite Messung wird der Sender auf NRZ und der Empfänger auf Manchester-Codierung geschaltet.

Es werden jeweils nacheinander folgende 32-Bit-Muster gesendet:

1. `00000000000000000000000000000000`
2. `11111111111111111111111111111111`
3. `10101010101010101010101010101010`
4. `01010101010101010101010101010101`

1.6.3 Auswertung

Messung 1: Sender = Manchester / Empfänger = NRZ

Das empfangene Bitmuster entspricht dem erwarteten Ergebnis. Der Empfänger interpretiert den empfangenen Bitstrom so, wie er vom Sender gesendet wurde, d.h. obwohl der Sender nur 32 Bit gesendet hat, empfängt der Empfänger 64 Bit. Das hängt damit zusammen, dass beim Manchester-Code die Baud-Rate doppelt so hoch ist wie die Bit-Rate. Für den Empfänger ist aber die Baud-Rate gleich der empfangene Bit-Rate weswegen die 32 Bit als 64 Bit interpretiert werden.

Da der Manchester-Code eine 0 als einen High-Low-Übergang in der Mitte des Bit-Intervalls darstellt und eine 1 als einen Low-High-Übergang, entsteht das empfangene Muster.

Wie bei den Mustern `101010...` und `010101...` zu sehen ist, werden die Muster nicht immer richtig empfangen. Das hängt einerseits mit dem fehleranfälligen Funkkanal und andererseits mit der heraufgeregelten Empfangsschwelle zusammen.

Manchester gesendet	NRZ empfangen
000000...	.#
111111...	#.
101010...	##.##..##..##..##..##..##..##..##..##..##..##..##..##..##..##..##.#.
(ideal)	.##..##..## (...) ##.
010101...	#..##..##..##..##..##...#..##..##..##..##..##..##..##..##..##..##.#.
(ideal)	#..##..

Tabelle 1.1: Sender = Manchester / Empfänger = NRZ

Ein weiterer Grund für diese Störungen ist, dass wir ein Bitmuster senden, ohne dass wir Sender und Empfänger vorher synchronisieren. Dies ist besonders problematisch, wenn eine Null am Anfang des Musters steht.

Als Anhaltspunkt sind bei den fehlerhaft empfangenen Bitmustern die theoretisch korrekten Bitmuster angegeben.

Messung 2: Sender = NRZ / Empfänger = Manchester

Die Bitmuster `0000...` und `1111...` stellen für den Empfänger, der ein Manchester-Signal erwartet, ein Problem dar. Diese Bitströme stellen konstante Pegel über die gesamte Bitübertragungszeit dar. Bei Manchester ist ein solcher Zustand nicht definiert, da stets eine Transition in der Mitte des Intervalls auftritt. Demnach kann der Empfänger mit den gesendeten Signalen nichts anfangen und erzeugt zufällige Bitsequenzen.

NRZ Gesendet	Manchester Empfangen
000000...	......#...#...#.#
111111...	......##.###....###..#
101010...	#...##.#########.##.#.##.#####.##..
(ideal)	.....
010101...	##################

Tabelle 1.2: Messung 2: Sender = NRZ / Empfänger = Manchester

Die Bitmuster `101010...` und `010101...` stellen für den Manchester-Empfänger dagegen (zumindest theoretisch) kein Problem dar. Zu beachten ist wiederum, dass bei Manchester, die Bitrate der halben Baudrate entspricht. Daher werden nicht 32 Bits empfangen, sondern nur 16.

Das Bitmuster `101010...` sollte eigentlich eine Folge von 16 Nullen erzeugen und `010101...` eine Folge von 16 Einsen. Allerdings wird auch hier ein idealer Empfang aus den oben genannten Gründen nicht erreicht.

Kapitel 2

Zweiter Termin

Die Sicherungsschicht (*Data Link Layer*) bietet der Vermittlungsschicht (*Network Layer*) Dienste an, indem sie Dienste von der Bitübertragungsschicht (*Physical Layer*) in Anspruch nimmt. Die Bitübertragungsschicht nimmt einen rohen Bitstrom auf und leitet ihn an den Empfänger weiter. Bei der Übertragung kann es zu Fehler kommen. Dies ist in besonderem Maße in einem Funkkanal der Fall, da hier Interferenzen viel häufiger auftreten als z.B. in einem kabelgebundenen Übertragungsmedium. Der empfangene Bitstrom ist also nicht unbedingt fehlerfrei. Die Aufgabe der Sicherungsschicht (Data Link Layer) ist es, diese Fehler aufzudecken und falls möglich zu korrigieren.

Die Sicherungsschicht arbeitet dabei nach folgendem Prinzip: Der Bitstrom wird in diskrete Rahmen (*Frames*) unterteilt. Im Rahmen dieses Versuchstermins besteht ein Frame aus

- einer Präambel,
- einem SFD-Feld,
- einem Payload-Feld mit fester Länge
- und einer Prüfsumme.

Die *Präambel* dient zu Bitsynchronisation, auf die zu Beginn von Abschnitt 2.1 näher eingegangen wird.

Das *SFD-Feld* (***S**tart of **F**rame **D**elimiter*) ist eine feste, definierte Bitfolge, die den Beginn des Frames kennzeichnet. Damit wird dem Empfänger zu verstehen gegeben, dass es sich bei den folgenden Bits um Nutzbits (*Payload*) handelt. Um die Payload und den Einfluss ihrer Länge geht es im Abschnitt 2.2.

Es existieren verschiedene Algorithmen zur Fehlererkennung. Bei der in diesem Versuch verwendeten CRC-Fehlererkennung (***C**yclic **R**edundancy **C**heck*) wird für jeden

Rahmen eine Prüfsumme berechnet. Am Empfänger wird die *Prüfsumme* erneut berechnet. Stimmt diese nicht mit der Prüfsumme im Frameheader überein, so weiß die Sicherungsschicht, dass ein Fehler aufgetreten sein muss.

2.1 Frameverluste in Abhängigkeit der Präambellänge

Digitale Daten können nicht direkt über einen Kanal (sei es nun ein Kabel oder eine Funkstrecke) übertragen werden. Sie müssen zunächst in analoge d.h. zeit- und wertkontinuierliche Signale umgewandelt werden. Durch sogenannte Signalformung werden dabei für den spezifischen Kanal angepasste, analoge Repräsentationen der Symbole (bei binärer Übertragung, der “Nullen” und “Einsen”) gesendet. Dies können relativ einfache Signalverläufe wie z.B. Rechteckimpulse, als auch entsprechend modulierte Trägersignale sein.

In jedem Fall hat der Empfänger das Problem, dass er (gegebenenfalls nach einer Demodulation) das empfangene Signal abtasten muss, um die digitalen Daten wieder zu erhalten. Die Zeitpunkte der Abtastung sollten dabei möglichst genau in der Mitte des Symbolintervalls liegen. Bei zweiwertiger Übertragung ist so ein Symbol ein Bit, man bezeichnet daher die Synchronisation der korrekten Abtastzeitpunkte als *Bitsynchronisation*. Sie sicherzustellen gibt es mehrere Verfahren. Das Senden von bekannten Bitmustern - einer *Präambel* - zu Beginn jeder Übertragung ist in diesem Zusammenhang besonders wichtig.

Die Länge der Präambel wirkt sich auf die Qualität der Bitsynchronisation und damit auf die Verlustrate der aus den übertragenen Bits gebildeten Frames aus. Ist die Bitsynchronisation ungenügend, werden die Grenzen der Frames nicht bzw. falsch erkannt. Der genaue Zusammenhang zwischen Präambellänge und Frameverlustrate soll im folgenden Versuch genauer untersucht werden.

Dazu werden mehrere Pakete mit einer bestimmten Präambellänge gesendet und gezählt, wieviele davon fehlerfrei (siehe dazu Abschnitt 2.3) ankommen. Diese Prozedur wird mit verschiedenen Längen und, um eine Aussage über den Einfluss der Übertragungsgeschwindigkeit zu bekommen, jeweils mit verschiedenen Baudraten durchgeführt.

2.1.1 Vorbereitung

Sender und Empfänger werden in einem Abstand von einem Meter aufgestellt, initialisiert, und auf die uns zugewiesene Mittelfrequenz eingestellt (siehe Abschnitt 1.3.1). Nach dem Kalibrieren kann die Übertragung durch Ein- und Ausschalten des Scramblers getestet werden.

Prinzipiell ist es egal, ob man die folgenden Versuche mit oder ohne aktiviertem Scrambler durchführt, solange man konsequent dabei bleibt. Wir haben uns daher entschieden, den drei folgenden Versuchen ohne Scrambler zu arbeiten.

Da wir in diesem zweiten Termin nicht mehr auf der reinen Bitebene arbeiten, muss das verwendete Protokoll mit der GUI von "`Manuell`" auf "`Framing C`" umgestellt werden. Es findet nun eine automatische Aufteilung der zu sendenden Bits in Frames (mit Präambel, SOF-Kennung und Prüfsumme) statt.

Für das Senden der Frames mit den entsprechenden Parametern wird ein vom Fachbereich TKN zur Verfügung gestelltes Script – `sendframe` – benutzt. Dieses muss zunächst von [2] geladen

```
wget http://www.tkn.tu-berlin.de/curricula/ss04/pr/sendframe
```

und mit

```
chmod ug+x ./sendframe
```

ausführbar gemacht werden.

2.1.2 Durchführung

Bei diesem Versuch geht es darum, die Präambellänge im Bereich zwischen 0 und 100 Bit zu variieren. Als erstes wird also eine Länge von 0 an beiden Stationen eingestellt.

Es werden nun 1000 Pakete mit einer Nutzlast von jeweils 10 Byte verschickt.

```
sendframe 1000 10
```

Dabei ist zu beachten, dass die 10 Byte nicht nur als Parameter in `sendframe` auftauchen, sondern auch (mittels GUI) an Sender und Empfänger eingestellt werden müssen.

Am Receiver wird jeder korrekt empfangene Frame registriert. Der Befehl

```
cat /dev/plccx > messung_v1_4.8_pr0.dat
```

schreibt für jeden dieser Frames eine Zeile in die angegebene Datei, die dann in etwa folgendermaßen aussieht:

```
...
8_XXXXX_8
10_XXX_10
13_XXX_13
14_XXX_14
18_XXX_18
...
```

Die Anzahl der korrekt empfangenen Frames (von 1000) kann nun ganz einfach durch zählen der Zeilen mit

```
wc -l messung_v1_4.8_pr0.dat
```

festgestellt und nach $p_{Verlust} = (1000 - n_{OK})/10$ in einen prozentualen Frameverlust umgerechnet werden.

Diese Prozedur wird nun mehrmals wiederholt, wobei die Präambellänge in Zehnerschritten bis auf den Wert von 100 Bit erhöht wird. Zwischen 0 und 10 Bit ändern sich die Frameverluste sehr stark, so dass zusätzlich noch eine Übertragung mit 5 Bit eingefügt wird.

Anschließend wird die gesamte Messung noch mit höheren Übertragungsraten durchgeführt. Die Baudrate wird dazu an beiden Stationen nacheinander auf die Werte `4.8`, `9.6`, `14.4` und `15.36` kBaud gesetzt. Zu beachten ist hierbei, dass der Wert für die `RX_Deviation` jedes Mal wieder ungefähr auf `4.8` gebracht wird.

2.1.3 Auswertung

Das Diagramm in Abbildung 2.1 stellt das Ergebnis all dieser Messungen dar[1]. Es zeigt sich, dass die Frameverluste zunehmen, je kürzer die verwendete Präambel wird. Besonders unterhalb einer bestimmten Wertes, der sich je nach Baudrate unterscheidet steigen sie rapide an.

Offenbar benötigt der Empfänger eine bestimmte Mindestlänge der Präambel, um erfolgreich übertragen zu können. Unterhalb dieser Länge ist eine Bitsynchronisation nicht oder nur sehr schlecht möglich, so dass ein großer Teil der Frames verloren geht. Ist diese Länge jedoch einmal erreicht, bringt eine weitere Verlängerung kaum noch Verbesserungen der Übertragungsqualität.

Besonders gut ist das bei 4,8 kBaud, aber auch bei 15,36 kBaud zu sehen. Die Verlustrate pegelt sich auf ein bestimmtes Niveau ein und fällt dann nur noch sehr langsam ab. Sie wird ab da weniger von der Präambellänge, als vielmehr von der Übertragungsgeschwindigkeit bestimmt. Bei 4,8 kBaud geht sie fast auf 0 zurück, während sie bei

[1] zugrundeliegende Messwerte im Anhang (Tabelle A.3)

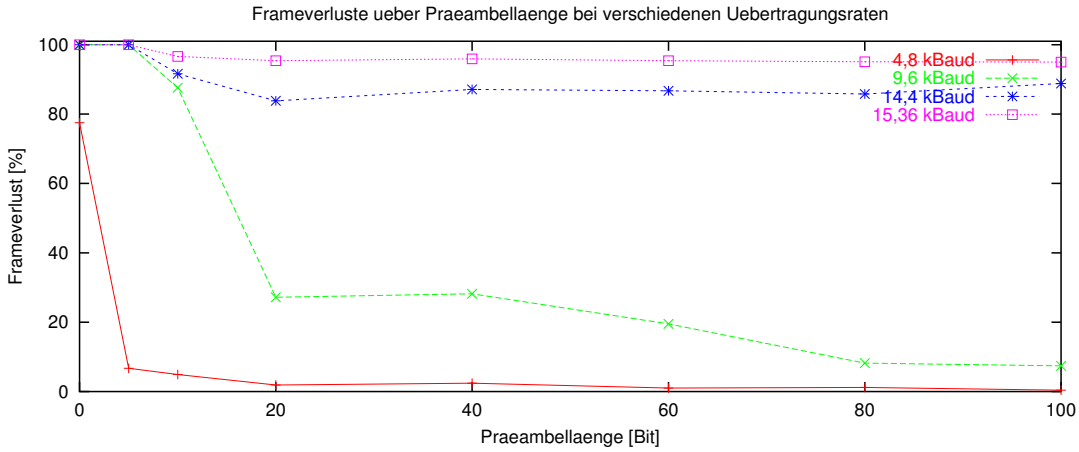


Abbildung 2.1: Frameverluste in Abhängigkeit von der Präambellänge

15,36 kBaud bei etwa 95% stagniert. Selbst mit den längsten Präambeln wird hier wohl nie eine fehlerfreie Übertragung zu erzielen sein.

Das der Anstieg der Verlustrate beim Unterschreiten der Mindestlänge um so steiler ausfällt, je höher die Übertragungsrate ist, lässt sich leicht begründen. Je höher die Baudrate, desto mehr Symbole (hier Bits) werden pro Sekunde übertragen und desto kürzer ist jedes einzelne davon. Kürzere Symbole erfordern jedoch höhere Präzision bei der Wahl der Abtastzeitpunkte und damit eine bessere Bitsynchronisation. Die Auswirkungen einer zu kurzen Präambel sind also bei hohen Baudraten schneller spürbar, da hier eine langsam ungenauer werdende Bitsynchronisation stärker auffällt.

2.2 Frameverluste in Abhängigkeit der Paketlänge

Bei dem in diesem Versuch verwendeten sendframe-Protokoll wird jeder Frame verworfen, der nicht fehlerfrei erkannt wurde, sei es durch fehlende Bitsynchronisation, wie im vorhergehenden Versuch, durch einen fehlerhaften SFD oder durch eine falsche Prüfsumme.

Da im Labor eine feste Paketlänge verwendet wird, entfällt die Notwendigkeit, dafür zu sorgen, dass das SFD-Feld nicht zufällig in den Bits der Payload kodiert ist. Aus dem gleichen Grund ist keine Frameenderkennung notwendig.

Ziel des zweiten Versuchs ist es, die Frameverlustrate in Abhängigkeit der Paketlänge darzustellen. Es ist also zu untersuchen wie sich die Verlustrate verändert, wenn die Paketlänge variiert wird.

2.2.1 Vorbereitung

Die Vorbereitung des Experiments entspricht genau dem in Abschnitt 2.1.1 beschriebenen Vorgehen. Sender und Empfänger werden initialisiert, auf die Mittenfrequenz eingestellt, kalibriert und es wird sichergestellt, dass das sendframe-Script zur Verfügung steht.

Analog zu Versuch 2.1 wird der Scrambler ausgeschaltet und das C-Framing in der GUI aktiviert.

2.2.2 Durchführung

Um die Abhängigkeit zwischen der Paketlänge und der Frameverlustrate zu analysieren, wird die Paketlänge zwischen 10 und 1000 Byte verändert. Dabei wird die Präambel konstant bei einer Länge von 100 Bit gehalten. Wie oben schon kurz erwähnt, wird zur Fehlererkennung ein CRC16-Code verwendet. Die Länge der Payload wird vor jeder Messung in der graphischen Oberfläche eingestellt.

Am Sender werden durch den Befehl

```
sendframe 1000 PAYLOAD_SIZE
```

1000 Frames gesendet. Am Empfänger wird der Paketstrom durch den Befehl

```
/dev/plccx > messung.txt
```

in einer Datei abgespeichert. Anschließend wird am Empfänger durch den Befehl

```
wc -l messung.txt
```

wieder die Anzahl der empfangenen Frames ermittelt. Die verloren gegangenen Frames ergeben sich dann, wie im ersten Versuch, aus der Differenz zu den gesendeten 1000 Frames.

Die Messungen werden analog zu Versuch 1 bei den Baud-Raten `4.8`, `9.6`, `14.4` und `15.36` kBaud durchgeführt. Zu jeder eingestellten Baud-Rate wird die Payload-Size zwischen 10 und 1000 Byte variiert und die Verlustrate aufgezeichnet.

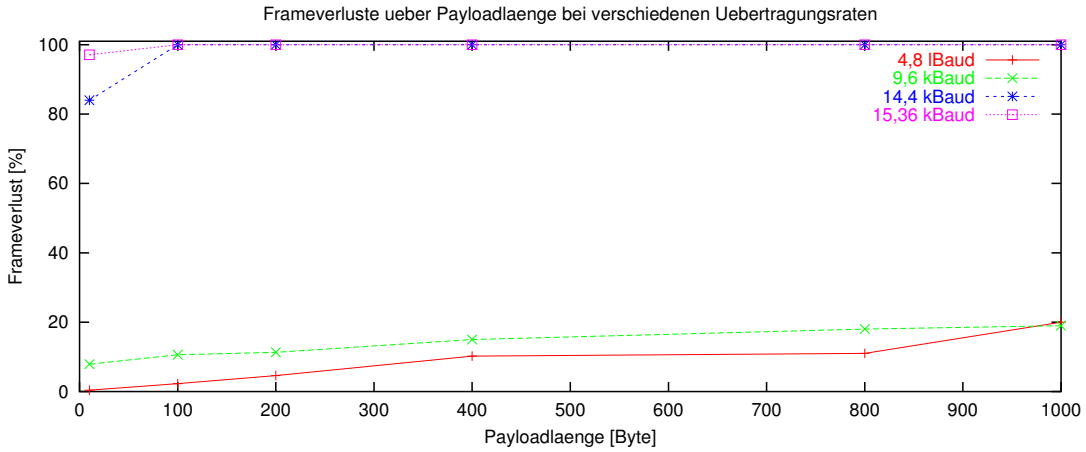


Abbildung 2.2: Frameverluste in Abhängigkeit von der Payloadlänge

2.2.3 Auswertung

Dem Diagramm[2] in Abbildung 2.2 ist zu entnehmen, dass die Frameverlustrate mit der Payload-Größe steigt. Außerdem ist deutlich zu erkennen, dass sie stark von der Baudrate abhängt. So konnten bei einer Payloadlänge von 100 Byte und einer Baudrate von 15,36 kBaud keine Frames mehr korrekt empfangen werden. Bei gleicher Payloadlänge beträgt die Verlustrate bei 9,6 kBaud dagegen nur ca. 10%.

Besonders drastisch ist der Einfluss der Payloadlänge bei einer Baudrate von 14,4 kBaud. Während bei einer Payloadlänge von 0 Bytes Frames noch korrekt empfangen werden, liegt die Frameverlustrate mit 100 Byte bereits bei 100%.

Dieses Ergebnis ist damit zu erklären, dass mit steigender Payloadlänge auch die Wahrscheinlichkeit steigt, dass eines der Bits im Frame fehlerhaft ist. Bei dem verwendeten Protokoll hat dies zur Folge, dass der Fehler durch den CRC-Code erkannt, und der Frame verworfen wird.

Die Frameverlustrate steigt außerdem mit der Baudrate, da die Bits schneller übertragen werden und damit burstartige Störungen nicht mehr nur wenige Bits betreffen, wie bei niedrigeren Baudraten, sondern infolge der Tatsache, dass mehr Bits pro Zeiteinheit übertragen werden, mehrere Bits verfälschen können (siehe dazu auch Abschnitt 2.4.2). In einem Funkkanal treten burstartige Fehler noch häufiger auf als in einem kabelgebundenen Übertragungsmedium.

[2] zugrundeliegende Messwerte im Anhang (Tabelle A.4)

2.3 Auswirkungen von Fehlerschutzmechanismen

Wir haben es in der Praxis mit unzuverlässigen Übertragungskanälen zu tun. Folglich kommen die gesendeten Bits nicht immer fehlerfrei beim Empfänger an. Um es trotzdem zu ermöglichen, dass zwischen Sender und Empfänger sinnvoll Informationen ausgetauscht werden können, muss dafür gesorgt werden, dass Übertragungsfehler am Empfänger erkannt werden. Der Empfänger könnte dann beispielsweise eine Neuübertragung der fehlerhaften Bits vom Empfänger fordern (ARQ-Verfahren).

Es existieren verschiedene Mechanismen zur Fehlererkennung und -korrektur. Im Rahmen dieses Versuches werden zwei Fehlererkennungscodes miteinander vergleichen. Es handelt sich dabei einerseits um den sogenannten *Parity-Check* und andererseits um den bereits erwähnten *Cyclic Redundancy Check* (CRC).

Der Parity-Check ist ein sehr einfacher Fehlererkennungscode. Hierbei hängt der Empfänger ein sog. Parity-Bit an den Datenstrom an. Dieses Parity-Bit wird so berechnet, dass der Bitstrom entweder eine gerade (even parity) oder eine ungerade Anzahl an Einsen (odd parity) hat. Wird beispielsweise die Bitsequenz `1110001` gesendet und verwendet der Sender odd parity, so wird an die Bitsequenz eine Eins angefügt, so dass sich insgesamt eine ungerade Anzahl an Einsen ergibt.

Der Empfänger untersucht den empfangenen Bitstrom und berechnet die Parität. Entspricht die Anzahl der empfangenen Einsen der Parität, also gerade oder ungerade, so geht er davon aus, dass kein Übertragungsfehler vorlag und somit der Bitstrom korrekt empfangen wurde. Dass diese Annahme keinesfalls zutreffen muss, wird weiter unter beleuchtet. Wird bei der Übertragung der Bitsequenz ein Bit invertiert, so ist die Parität auf Empfängerseite nicht korrekt und der Übertragungsfehler wird erkannt.

Ein anderer, weit verbreiteter Fehlererkennungscode ist der Cyclic Redundancy Check (CRC). Dieser arbeitet wie folgt: für einen gegebenen Block von k Bits generiert der Sender eine $n - k$ Bit lange Sequenz, auch *Frame Check Sequence* (FCS) genannt, so dass der resultierende Block der Länge n exakt durch eine vorgegebene Bitsequenz teilbar ist. Der Empfänger teilt den empfangenen Bitstrom durch diese Sequenz (auch Generatorpolynom genannt) und wenn die Division keinen Rest ergibt, wird der Frame als korrekt akzeptiert.

2.3.1 Vorbereitung

2.3.2 Durchführung

Ziel dieses Versuches ist es, die beiden beschriebenen Fehlererkennungsverfahren zu vergleichen. Dazu werden analog zum Versuch in Abschnitt 2.1 zwei Messreihen mit CRC und Parity-Check als Fehlererkennungsverfahren bei 9,6 kBaud aufgenommen, wobei wieder die Präambellänge zwischen 0 und 100 Bit variiert wird. Das verwendete Verfahren wird vor der Messung über die GUI aktiviert.

2.3.3 Auswertung

Das fertige Diagramm[3] (Abbildung 2.3) zeigt, dass die Frameverlustrate bei gleicher Präambellänge bei CRC höher ist als bei Parity-Check.

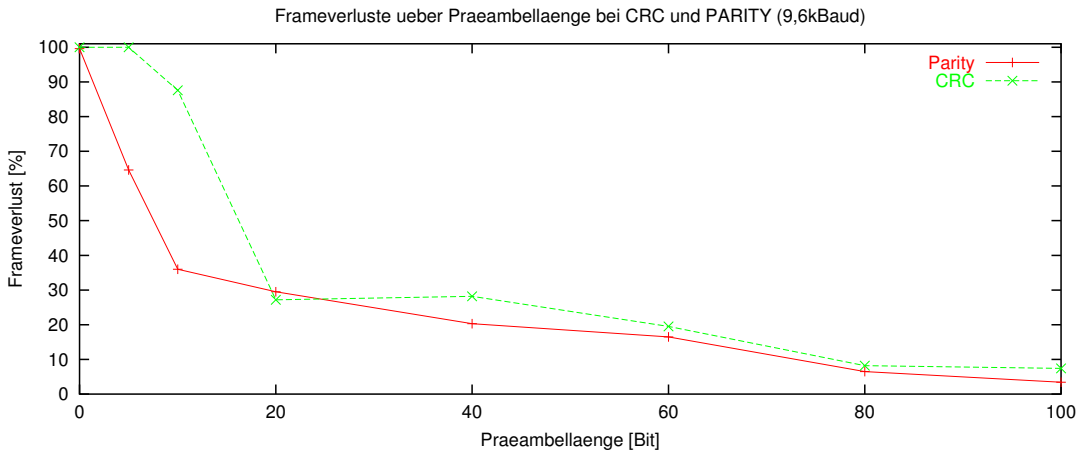


Abbildung 2.3: Frameverluste in Abhängigkeit von der Präambellänge

Dieses Ergebnis lässt sich damit erklären, dass CRC eine genauere Fehlererkennung ermöglicht als Parity-Check. Der Parity-Check erkennt beispielsweise nicht, wenn zwei oder eine gerade Anzahl an Übertragungsfehler vorliegen, weil hierbei die Parität nicht verändert wird.

CRC dagegen erkennt alle Einzelfehler und die meisten Doppelfehler. Ist $(x + 1)$ ein Linearfaktor des Generatorpolynoms, so werden auch alle Fehler erkannt, die aus einer ungeraden Zahl von invertierten Bits bestehen. Am wichtigsten ist jedoch, dass man mit einem Polynomcode mit r Prüfbits alle Burst-Fehler der Länge $\leq r$ erkennt. Burst-Fehler werden bei Parity-Check nur dann erkannt, wenn sie nicht aus einer geraden Anzahl fehlerhafter Bits bestehen. Gerade bei einem Funkkanal, in dem Fehler häufig in Bursts auftreten, ist Parity zur Fehlererkennung nicht ausreichend.

Diese kurze Ausführung zeigt, dass CRC wesentlich besser zu Fehlererkennung geeignet ist als Parity-Check, da mehr Übertragungsfehler erkannt werden. Daraus ergibt sich auch, dass beim CRC die Frameverlustrate höher ist als beim Parity-Check, da zusätzliche Frames als fehlerhaft eingestuft und verworfen werden.

2.4 Zusatzaufgabe: Burstiness der Fehler

Mittels eines von [2] heruntergeladenen Perl-Scripts – `errorstat.pl` – ist es möglich, die in Abschnitt 2.1 und 2.2 gewonnenen Messdaten hinsichtlich der Verteilung

[3] zugrundeliegende Messwerte im Anhang (Tabelle A.5)

der Fehler (Frameverluste) auszuwerten. Hieraus lassen sich Schlüsse über die sog. *burstiness* der Fehler – also ihre Neigung in Bündeln aufzutreten – ziehen.

Besonders interessant ist hierbei die Dauer solcher Fehlerbursts und die durchschnittliche Zeit, die zwischen ihrem Auftreten vergeht.

Aus Zeitgründen wurde diese Untersuchung nur für die Daten aus Abschnitt 2.1, also nur für Frameverluste bei unterschiedlichen Präambellängen durchgeführt. Für den zweiten Versuch müsste man analog vorgehen

2.4.1 Durchführung

Ein Blick in den Quelltext des Perl-Scripts[4] zeigt, dass zwei Parameter, der Dateiname und die Anzahl der erwarteten Frames, übergeben werden müssen. Ein Aufruf liefert als erste Zahl die Anzahl der verloren gegangenen Frames. Danach folgt die durchschnittliche Länge der Bursts und schließlich die mittlere Zeit zwischen zwei Bursts.

Um Arbeit zu sparen empfiehlt sich der Aufruf in einer Schleife. Der Befehl

```
for name in ‘ls messung_v1*.dat‘ ; do \
   echo “# $name” ; \
   perl errorstat.pl $name 1000 ; \
done >> burstiness_v1.dat
```

erzeugt eine Datei mit den berechneten Werten aus allen Messdateien des ersten Versuchs. Diese müssen dann nur noch nach Baudraten und Präambellängen sortiert, und per Gnuplot dargestellt werden.

2.4.2 Auswertung

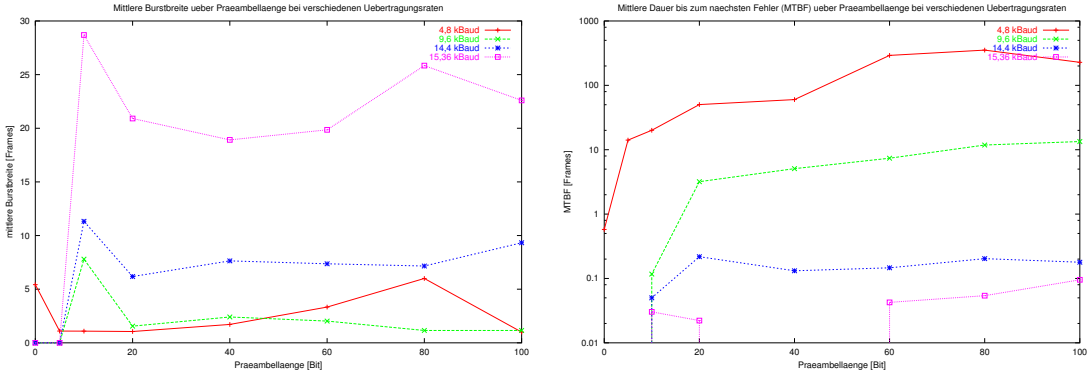
Die errechnete Burstiness für unterschiedliche Präambellängen sind in Abbildung 2.4 dargestellt[5]. Aus 2.4a ist zu erkennen, dass die mittlere Breite des Fehlerbursts mit der Baudrate ansteigt.

Dies entspricht dem erwarteten Ergebnis, da mit zunehmender Baudrate mehr Symbole (hier Bits) pro Sekunde übertragen werden, und ein Burst mit einer festen Zeitdauer bei höheren Baudraten mehr Symbole stört als bei niedrigeren.

In Abbildung 2.4b ist weiterhin erkennbar, dass die mittlere Zeit zwischen zwei Fehlern (MTBF) mit steigender Baudrate abnimmt. Diese Abnahme ist sogar ausgesprochen stark. Man muss sich hierbei vor Augen halten, dass es sich bei dem Diagramm

[4] **Hinweis:** Etwas Dokumentation wäre in diesem Fall hilfreich gewesen. :-)

[5] zugrundeliegende Messwerte im Anhang (Tabelle A.6)



(a) Mittlere Breite der Fehlerbursts

(b) Mittlere Dauer bis zum nächsten Fehler

Abbildung 2.4: Burstiness bei variabler Präambellänge

um eine logarithmische Darstellung der Fehlerabstände handelt. Die Zeiten unterscheiden sich also gleich um ganze Zehnerpotenzen. So tritt bei einer Präambellänge von 60 Bit bei 14,4 kBaud durchschnittlich alle 0,15 Frames ein Fehler auf, während bei 4,8 kBaud nur alle 292 Frames mit einem solchen Burst zu rechnen ist.

Die Abhängigkeit der Burstiness von der Präambellänge ist ebenfalls gut zu sehen. Die mittlere Burstbreite bleibt nahezu konstant, während die MTBF mit zunehmender Präambellänge ansteigt, Fehler also immer seltener auftreten, je besser die Framesynchronisation funktioniert.

Kapitel 3

Dritter Termin

Das Aloha-Verfahren wurde um 1970 von NORMAN ABRAMSON an der Universität von Hawaii entwickelt. Aloha ist ein Verfahren, um den Zugriff von mehreren nicht koordinierten, miteinander konkurrierenden Benutzern auf einen einzelnen, gemeinsam benutzten Kommunikationskanal zu steuern.

Dem Aloha-Verfahren liegt ein einfaches Konzept zugrunde: Benutzer dürfen jederzeit übertragen, wenn sie Daten senden wollen. Folglich kommt es zu Kollisionen und die kollidierenden Frames werden beschädigt.

3.1 Durchsatz über Last beim Aloha-Verfahren

Im Rahmen dieses Versuches haben wir es mit einem Funkkanal zu tun. Daher ist es dem Sender nicht möglich, durch Abhören des Kanals (wie z.B. bei Ethernet) zu erfahren, ob eine Kollision stattgefunden hat. Aus diesem Grund wird der Datenverkehr über eine Basisstation geleitet, die in einem separaten Funkkanal auf einer anderen Frequenz Bestätigungen verschickt. Durch Abhören dieses Rückkanals kann die sendende Station feststellen, ob das gesendete Paket bei der Basisstation angekommen ist oder nicht. Dazu wird nach dem Senden ein Timer gestartet. Läuft dieser ab, bevor der Sender ein Acknowledgement von der Basisstation erhalten hat, wartet er eine zufällige Zeitspanne und versucht dann erneut, das Paket zu übertragen. Die Zeitspanne muss zufällig sein, ansonsten würden die gleichen Pakete immer wieder kollidieren. Durch die Verwendung eines separaten Kanals für die Acknowledgements wird verhindert, dass Bestätigungen kollidieren.

3.1.1 Vorbereitung

Im Unterschied zu den bisherigen Versuchen, werden jetzt vier Transceiver verwendet, wobei ein Transceiver als Basisstation fungiert und die anderen drei als Sender

verwendet werden.

Die drei Sender schicken Pakete an die Basisstation, welche alle von ihr korrekt empfangenen Pakete auf dem separaten Rückkanal quittiert. Gehen Pakete durch Kollisionen verloren und erhalten die Sender daher keine Bestätigung, versuchen sie nach einem zufällig gewählten Backoff, das Paket erneut zu übertragen.

Es ist also notwendig, die drei Sender auf eine gemeinsame Sendefrequenz und eine gemeinsame Empfangsfrequenz (Rückkanal) einzustellen. Dies geschieht (nach Aufrufen des Init-4.8 Settings) am Sender durch

```
ccconfig -sfrx A 860
ccconfig -sftx B 865
```

und an der Basisstation entsprechend umgekehrt durch

```
ccconfig -sfrx A 865
ccconfig -sftx B 860
```

Anschließend wird durch Ein- und Ausschalten des Scramblers die Übertragung zwischen den Sendern und der Basisstation getestet. Verläuft der Test erfolgreich, so wird nun als MAC-Zugriffsverfahren das Aloha-Protokoll im der GUI eingestellt. An der Basisstation wird zusätzlich das Senden von Acknowledgements durch

```
ccconfig -ccAck 1
```

aktiviert.

Die korrekte Übertragung wird schließlich erneut durch das Senden einer Testbitfolge von den Sendern zur Basisstation verifiziert. Dabei wird durch

```
echo 'Testbitfolge x' > /dev/plccx
```

von jedem Sender eine andere Bitfolge gesendet, so dass diese an der Basisstation auseinander gehalten werden können. Zuletzt müssen noch die für die Durchführung verwendeten Scripte `alohaT` und `alohaR` von [2] geladen und ausführbar gemacht werden.

3.1.2 Durchführung

Es werden nun bei allen drei Sendern gleichzeitig Übertragungen zur Basisstation initiiert. Dies geschieht durch den Aufruf von

```
alohaT gruppe2 SENDERID PAKETANZAHL WARTEZEIT 30 SENDERID
```

Dies startet die Übertragung von `PAKETANZAHL` Paketen mit einer mittleren Wartezeit von `WARTEZEIT` und mit einer Größe von 30 Bytes. Jeder Sender erhält eine eigene `SENDERID`, so dass die an der Basisstation ankommenden Datenpakete eindeutig identifiziert werden können. Jede Messung wird in einer Datei abgespeichert, deren Name mit dem Präfix `gruppe2` beginnt und sich aus den oben angegebenen Parametern zusammensetzt.

An der Basisstation werden durch den Befehl

```
alohaR MESSUNG WARTEZEIT
```

die an `/dev/plccx` ankommenden Daten aufgenommen. Die `WARTEZEIT` bezeichnet die Zeit, die ohne Aktivität an `/dev/plccx` verstreichen darf, bevor die Messung abgebrochen wird.

Die Messung wurde unter den vier Gruppen des Praktikums aufgeteilt, wobei unsere Gruppe für die in Tabelle 3.1 angegebenen Parameter zuständig war.

Paketanzahl	mittlere Wartezeit [10ms]
100	380
500	319
750	280
1500	245

Tabelle 3.1: Übertragungsparameter der von uns durchgeführten Messungen

Als Ergebnis einer Messung erhält man zwei Dateien. Die Datei `gruppe2*.kernnel`[1] enthält hierbei die Ausgaben des Kernels während der Messung. Für jedes empfangene Paket bzw. gesendete ACK ist dort ein Eintrag mit genauem Zeitstempel (in Jiffies, 100 Jiffies = 1 Sekunde) und der ID des Senders zu finden.

Zunächst muss ermittelt werden, wie lange die Stationen tatsächlich in Konkurrenz zueinander gesendet haben. Dazu wird aus der `*.kernnel`-Datei des Empfängers der Zeitpunkt des letzten gesendeten Konkurrenzpaketes bestimmt und die Startzeit

[1] eigentlich Ker**n**el, das Script legt aber tatsächlich eine `*.ker`**`nn`**`el` Datei an

des ersten empfangenen Paketes davon abgezogen. $t_{konk} = \frac{t_{end,konk} - t_{start}}{100}$ liefert dann die Dauer der Konkurenzperiode in Sekunden.

Als nächstes werden alle Übertragungen, die nicht innerhalb der Konkurrenzperiode stattgefunden haben aus den *.kernnel-Dateien aller Stationen entfernt.

Die Anzahl der Übertragungsversuche n_{gesamt} lässt sich nun aus den Dateien der einzelnen Sender durch

```
grep "send Pl." s?/gruppe2_id?_n???_t???_s30_p? | wc -l
```

bestimmen. Dieser Aufruf liefert die Anzahl aller Send-Meldungen.

Die Anzahl der von der Basisstation bestätigten Pakete ermittelt man ebenfalls aus den Dateien der Sender durch

```
grep "rcvd Ack-Frame." s?/gruppe2_id?_n???_t???_s30_p? | wc -l
```

Dadurch werden alle empfangenen Acknowledgements gezählt und dadurch die Anzahl der korrekt übertragenen Pakete $n_{korrekt}$ bestimmt. Diese Werte werden anschließend gemittelt.

Die Last errechnet sich nunmehr nach der Anzahl der Übertragungsversuche n_{gesamt} bezogen auf die Dauer der Konkurrenzperiode t_{konk}.

Der Durchsatz gibt die Anzahl der erfolgreich übertragenen Pakete pro Sekunde an und muss daher ebenfalls auf die Dauer der Konkurrenzperiode normiert werden. Möchte man diesen Wert als prozentualen Anteil der Gesamtkapazität darstellen, muss man diese zunächst errechnen. Die theoretisch maximal mögliche Übertragungskapazität des Kanals beträgt

$$C_{max} = \frac{R}{l} = \frac{2400\frac{Bit}{s}}{(30 \cdot 8)\frac{Bit}{Paket}} = 10\frac{Pakete}{s}$$

Da sich 3 Stationen diese Kapazität teilen müssen verbleiben (bei Annahme einer fairen Verteilung) $C_{max}/3 = 3,33\frac{Pakete}{s}$ für jede Station.

Der prozentuale Durchsatz ergibt sich damit zu

$$S = \frac{\frac{\bar{n}_{korrekt}}{t_{konk}}}{\frac{C_{max}}{3}} \cdot 100 = \frac{30 \cdot \bar{n}_{korrekt}}{t_{konk}}$$

3.1.3 Auswertung

In Abbildung 3.1 wird der Durchsatz über der Last dargestellt[2]. Es ist zu erkennen, dass der Durchsatz zunächst mit der Last ansteigt. Bei halber Auslastung des Kanals (Last bei ca. 0.5) ist das Maximum erreicht, das hier bei etwa 18% liegt. Tatsächlich liegt der Wert sogar etwas darüber, was wahrscheinlich auf eine Meßungenauigkeit zurückzuführen ist. Steigt die Last über einen Wert von ca. 0.5, so nimmt der Durchsatz rapide ab.

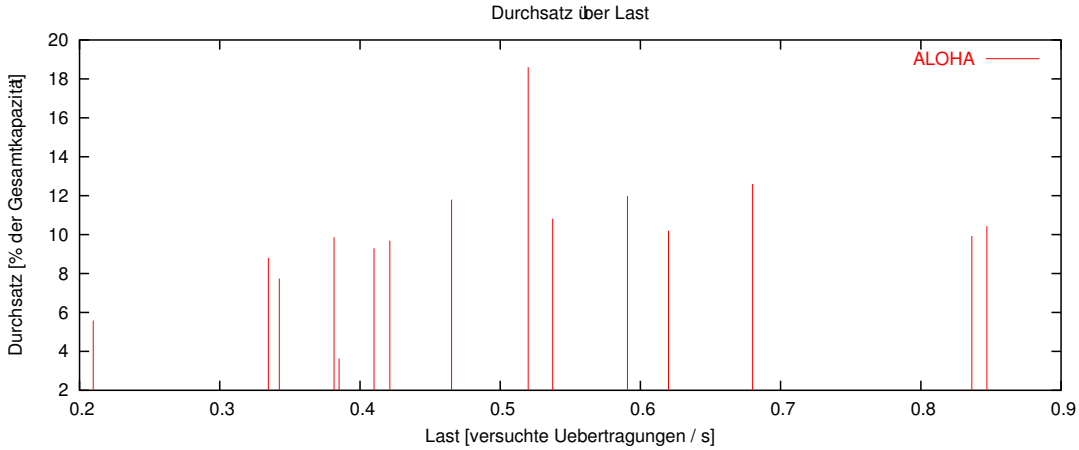


Abbildung 3.1: Durchsatz über Last bei Aloha

Dies entspricht in etwa dem aus der Theorie zu erwartenden Ergebnis. Danach wird bei einer Last von $G = 0,5$ ein maximal möglicher Durchsatz von $\frac{1}{2e} = 0,1839$ erreicht, ca. 18,4% entspricht.

Um dies zu verstehen, wird im Folgenden der theoretisch maximale Durchsatz kurz hergeleitet.

Im Modell wird von einer unbegrenzten Anzahl von Benutzern ausgegangen, die neue Frame der Länge l nach einer Poisson-Verteilung mit einer mittleren Anzahl von N Frames pro Frameübertragungszeit $t_g = \frac{l}{R}$ senden. Damit die Datenübertragung zur Basisstation noch erfolgreich verlaufen kann, wird vorausgesetzt, dass

- überhaupt Frames gesendet werden ($N < 0$) und gleichzeitig
- nicht mehr Frames als maximal möglich ($N < 1$) übertragen werden.

Es muß also gelten: $0 < N < 1$.

[2] zugrundeliegende Messwerte im Anhang (Tabelle A.7)

Da die Stationen zusätzlich zu den neuen Frames auch noch alte Frames übertragen, falls vorher Kollisionen aufgetreten sind (sog. Retransmission), sei die gesamte angebotene Last ebenfalls poissonverteilt, mit einem mittleren Wert von G pro Frameübertragungszeit. Dieser Wert entspricht der im Versuch ausgerechneten Last.

Der Durchsatz S ist die gegebene Last multipliziert mit der Wahrscheinlichkeit P_0, dass eine Übertragung erfolgreich ist, d.h. keine Kollisionen aufgetreten sind. Demnach gilt

$$S = G \cdot P_0 \tag{3.1}$$

Die Periode in der ein weiterer Übertragungsversuch zu einer Kollision führen, und damit die gegenwärtige Übertragung stören würde (vulnerability period) entspricht $2t_g$. Die Wahrscheinlichkeit, dass während t_g keine weiteren Rahmen übertragen werden, ergibt sich aus der Poisson-Verteilung zu

$$P(k = 0) = \frac{G^k e^{-G}}{k!} = e^{-G} \tag{3.2}$$

Für eine Periode von $2t_g$wäre die die Wahrscheinlichkeit dementsprechend e^{-2G}, so dass sich eine korrekte Übertragung mit der Wahrscheinlichkeit von

$$P_0 = e^{-2G} \tag{3.3}$$

bewerkstelligen lässt. Mit Gleichung 3.1 und 3.3 ergibt sich für Aloha ein theoretischer Durchsatz von

$$S = G \cdot e^{-2G} \tag{3.4}$$

Ein einfacher Optimierungsansatz liefert dann einen maximalen Durchsatz von $\frac{1}{e} = 0,1835$, also etwas über 18% bei einer Last von $G = 0,5$.

Anhang A

Messwerte

A.1 Erster Termin

#	BER	RSSI	BER	RSSI	BER	RSSI	BER	RSSI
# RATE:	4.8	4.8	9.6	9.6	11.52	11.52	15.68	15.68
-40	0.505	15	0.505	15	0.505	14	0.499	14
-35	0.498	15	0.498	14	0.503	15	0.497	14
-25	0.490	14	0.490	14	0.500	14	0.496	14
-20	0.489	14	0.499	14	0.496	14	0.494	15
-18	0.489	15	0.498	15	0.497	15	0.493	15
-16	0.490	15	0.496	15	0.494	15	0.489	15
-15	0.475	15	0.490	15	0.494	15	0.488	16
-14	0.472	15	0.478	14	0.487	16	0.483	16
-12	0.444	16	0.475	15	0.484	16	0.472	16
-11	0.390	16	0.443	16	0.460	16	0.477	16
-10	0.330	17	0.390	17	0.456	16	0.442	17
-9	0.260	17	0.340	17	0.454	17	0.435	17
-8	0.200	18	0.250	18	0.370	17	0.389	18
-7	0.110	19	0.230	18	0.370	18	0.263	19
-6	0.060	20	0.220	18	0.360	18	0.260	19
-5	0.040	21	0.160	19	0.270	19	0.245	20
-4	0.010	21	0.050	20	0.130	20	0.210	20
-3	0.003	22	0.030	21	0.080	20	0.180	21
-2	0.001	22	0.020	22	0.040	20	0.099	22
-1	0.000	23	0.010	22	0.035	20	0.090	23
0	0.000	24	0.001	22	0.002	21	0.070	23
1	0.000	25	0.000	23	0.001	21	0.053	24
2	0.000	25	0.000	24	0.002	22	0.049	24
3	0.000	26	0.000	25	0.002	23	0.043	25
4	0.000	26	0.000	26	0.001	24	0.037	26
5	0.000	28	0.000	27	0.000	25	0.030	27

Tabelle A.1: BER bei verschiedenen Übertragungsraten

```
#       1m Abstand              3m Abstand
# dB    BER FSK BER ASK         BER FSK BER ASK
-40     0.500   0.510           0.505   0.520
-20     0.490   0.508           0.498   0.501
-15     0.488   0.498           0.430   0.493
-10     0.440   0.488           0.220   0.490
-8      0.330   0.484           0.060   0.480
-5      0.150   0.483           0.007   0.476
0       0.000   0.260           0.000   0.006
5       0.000   0.000           0.000   0.003
```

Tabelle A.2: BER bei FSK und ASK

A.2 Zweiter Termin

```
# Payloadsize 10, 1000 Frames (Präambel variiert 0..100)
#
#Präam.            Empfangene Frames (von 1000)
#       4,8             9,6             14,4    15,36   kBAUD
0       225             0               0       0
5       933             0               0       0
10      951             124             84      34
20      981             728             162     46
40      976             718             129     41
60      990             805             133     46
80      988             918             142     49
100     996             926             112     50
```

Tabelle A.3: Frameverluste bei verschiedenen Präambellängen

```
# C-Framing (Scrambler aus), Prüfung: CRC
# Präambel 100, 1000 Frames (Payload variiert 10..1000)
#
#PL.    Empf. Frames (von 1000, ab 400Byte nur noch 100 Pakete)
#               4,8             9,6             14,4    15,36   kBAUD
10              996             921             160     29
100             977             894             0       0
200             954             887             0       0
400             898             850             0       0
800             890             820             0       0
1000            800             810             0       0
```

Tabelle A.4: Frameverluste bei verschiedenen Payloadlängen

#Prä.	Empf. Frames (von 1000) bei 9,6kBAUD	
#	PAR	CRC
0	4	0
5	354	0
10	640	124
20	705	728
40	797	718
60	835	805
80	935	918
100	966	926

Tabelle A.5: Frameverluste über Präambellänge für CRC und PARITY

#	4,8		9,6		14,4		15,36	kBAUD
#Prä.	MG	MTBF	MG	MTBF	MG	MTBF	MG	MTBF
0	5.4225	0.5774	0	0	0	0	0	0
5	1.0983	14.098	0	0	0	0	0	0
10	1.0888	20.044	7.8018	0.1171	11.325	0.05	28.696	0.0303
20	1.0555	50.333	1.5574	3.1839	6.1804	0.2180	20.911	0.0222
40	1.7142	60.142	2.4102	5.0854	7.6403	0.1315	18.92	0
60	3.3333	292	2.0312	7.3854	7.3706	0.1465	19.851	0.0425
80	6	353	1.1549	11.830	7.1610	0.2033	25.837	0.0540
100	1	228	1.1562	13.421	9.3263	0.1789	22.595	0.0952

Tabelle A.6: Burstiness der Frameverluste (über Präambellänge)

A.3 Dritter Termin

Pakete	***Wait***	***tatsächlich gesendete Frames****				***Bestätigte Frames (am Sender)***				***Konk.t***	***Last***	***Durchsatz***
		1	***2***	***3***	***mittel***	***1***	***2***	***3***	***mittel***	***[sec]***	***]1/sec]***	***[1/sec]***
100	400	102	100	139	113.667	77	100	80	85.6667	331.95	0.34242	0.2581
100	**380**	**114**	**111**	**77**	**100.67**	**93**	**104**	**71**	**89.33**	**479.91**	**0.2098**	**0.1861**
100	360	108	127	116	117.00	83	76	100	86.33	282.88	0.41	0.31
100	340	243	286	168	232.333	100	34	85	73	603.3	0.38510	0.121001
500	320	633	615	698	648.667	505	482	504	497	1539.57	0.42133	0.3228
500	**310**	**618**	**539**	**474**	**543.67**	**501**	**479**	**451**	**477.00**	**1624.58**	**0.3347**	**0.2936**
500	300	781	1027	840	882.67	474	500	468	480.67	1432.33	0.62	0.34
500	290	1285	1436	778	1166.33	500	484	452	478.667	1376.84	0.84711	0.347656
750	285	1037	1017	1217	1090.33	771	709	716	732	2028.77	0.53743	0.3608
750	**280**	**891**	**859**	**798**	**849.33**	**720**	**725**	**752**	**732.33**	**2226.06**	**0.3815**	**0.3289**
750	275	1032	1258	1144	1144.67	694	1428	1497	1206.33	1953.52	0.52	0.62
750	265	2032	2411	1110	1851	714	733	750	732.333	2213.29	0.83631	0.330879
1000	255	1389	1381	1643	1471	996	1020	967	994.333	2490	0.59076	0.3993
1500	**245**	**1262**	**1161**	**1089**	**1170.67**	**999**	**967**	**1002**	**989.33**	**2516.10**	**0.4653**	**0.3932**
1000	225	1492	1698	1585	1591.67	1000	977	1003	993.33	2352.42	0.68	0.42
1000	225	5811	5311	1356	4159.33	995	935	1000	976.667	2540.74	1.63706	0.384402

(Fett gekennzeichnete Einträge wurden von unserer Gruppe gemessen)

Tabelle A.7: Last und Durchsatz

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Aufgabenblatt dieses Praktikums, http://www.tkn.tu-berlin.de/curricula/Block_A1-3.pdf

[2] Verwendete Scripte, zu finden unter http://www.tkn.tu-berlin.de/curricula/ss04/pr/

[3] Peter Noll, Vorlesungsskript "Nachrichtenübertragung II", TU-Berlin, 2004

[4] Andrew S. Tanenbaum, "Computer Networks", 4th edition, Prentice Hall, 2003