# Übung Mobilkommunikation – Praktische Aufgaben

## *Paketformat*

Die Datenpakete haben ein fest vordefiniertes Format. Als Ethernet-Typ wird **0x5507** festgelegt. Alle Elemente werden in **Network Byte Order** übertragen und müssen mittels hton*() und ntoh*() umgewandelt werden. Das Format ist wie folgt:

| **Feld** | **Format** | **Beschreibung** |
|---|---|---|
| header | struct ether_header | Ethernet-Header |
| seqno | uint16_t | Sequenznummer zur Duplikat-Erkennung |
| type | uint16_t | Typ des Pakets |
| size_param | uint32_t | Parameter: Größe / Offset(*) |
| datalen | uint16_t | Anzahl der Bytes in data[] (max. 1024!) |
| data | uint8_t[datalen] | Daten(*) |

(*) diese Felder sind vom Pakettyp abhängig.

Die Sequenznummer dient dazu, doppelt empfangene Pakete auszufiltern – dazu wird beim Empfang überprüft, ob das seqno-Feld größer ist als im zuletzt von dieser Station empfangenen Paket. Überläufe des 16-Bit-Wertes müssen beachtet werden.

Es sind unterschiedliche Paket-Typen möglich, die Unterscheidung erfolgt anhand des type-Feldes. Vom Pakettyp hängt der weitere Inhalt, und damit die Bedeutung der Felder size_param und data ab:

| **type** | **Paket-Typ** | **size_param** | **data** |
|---|---|---|---|
| 1 | HELLO | reserviert | Benutzername |
| 2 | MSG | reserviert | zu übertragende Text-Nachricht |
| 3 | OFFER | Dateigröße | Dateiname |
| 4 | DATAREQ | Offset in Datei | Dateiname |
| 5 | DATA | Offset in Datei | Inhalt der Datei ab dem angegebenen Offset |

Das size_param-Feld ist für den Versand von Dateien vorbehalten und besitzt für HELLO- und MSG-Nachrichten keine bedeutung.

Der eigentliche Inhalt der Nachricht steckt im data-Feld. Das davor kommende datalen-Feld legt hierbei fest, wie viele Datenbytes im data-Feld stehen.

### HELLO-Nachrichten

Nachrichten vom Typ HELLO muss jeder Rechner periodisch im 3-Sekunden-Takt versenden, um von seinen Nachbarn entdeckt zu werden. Sie werden an die Broadcast-Adresse (ff:ff:ff:ff:ff:ff) gesendet. Im data-Feld steht der Name des Benutzers, der die Applikation gestartet hat (oder alternativ der Rechnername).

### MSG-Nachrichten

Dieser Nachrichtentyp wird verwendet, um Text-Nachrichten an andere Benutzer zu schicken. Dazu kann entweder die MAC-Adresse eines Rechners (für „private“ Kommunikation) oder die Broadcast-Adresse (für Nachrichten an alle) als Ziel verwendet werden.

### OFFER-Nachrichten

Eine OFFER-Nachricht wird verwendet, um jemandem eine Datei anzubieten. Sie enthält den Dateinamen (ohne Verzeichnis-Elemente, also ohne „/“-Zeichen) in data und die Dateigröße im size_param-Feld. Der Empfänger einer OFFER-Nachricht sollte dem Benutzer die Möglichkeit geben, die Datei anzunehmen oder abzulehnen. Wenn der Benutzer sie annimmt, kann der Inhalt der Datei mit einer Abfolge von DATAREQ-Nachrichten abgerufen werden.

### DATAREQ-Nachrichten

Da sich große Dateien nicht mit einem einzelnen Netzwerkpaket übertragen lassen, müssen sie in einzelne Pakete aufgeteilt werden. Versendet man die Pakete mit dem Inhalt einer Datei direkt hintereinander, so tritt ein Überlauf in der Netzwerkkarte auf und die meisten Daten werden verworfen. Um das zu vermeiden, werden die Datenpakete vom Empfänger sequenziell angefordert, indem er DATAREQ-Pakete beginnend ab dem Datei-Offset 0 generiert, bis er alle Fragmente der Datei korrekt empfangen hat. Der Offset wird dabei im size_param-Feld und der Dateiname im data-Feld gesendet. Wird ein DATAREQ nicht in angemessener Zeit (300ms) beantwortet, so muss die Anfrage wiederholt werden.

### DATA-Nachrichten

Eine DATAREQ-Nachricht wird mit einem DATA-Paket beantwortet. Das Format ist darauf ausgelegt, dass der Sender sich keine Verbindungsdaten merken muss – er bekommt den Dateinamen und die Dateiposition im DATAREQ-Paket, und kann die Datei daraufhin öffnen, die erforderlichen Daten einlesen und ein DATA-Paket mit dem Inhalt zurücksenden. Dabei muss das Datenpaket mit 1024 Bytes gefüllt werden, sofern die Datei noch nicht abgeschlossen wurde.

### Zu Beachten!

- Bei struct für Paketformat „`__attribute__((packed))`“ verwenden
- `htons()` beim Vesenden von 16-Bit-Zahlen(Host-to-Network short)
- `htonl()` beim Vesenden von 32-Bit-Zahlen(Host-to-Network long)
- beim Empfangen analog `ntohs()` und `ntohl()` (Network-to-Host)
- OFFER- und DATAREQ-Nachrichten dürfen kein „/“-Zeichen im Dateinamen enthalten! (Überprüfung mittels `strchr(data, '/')`)
- Alle Dateinamen sollten sich auf ein festgelegtes Verzeichnis beziehen, um das Überschreiben oder Auslesen kritischer Daten zu vermeiden (z.B. „datenspeicher/“).

## *Berechtigungen zum Arbeiten mit RAW Sockets*

Zum öffnen von RAW-Sockets auf den Laborrechnern müssen dem kompilierten eigenen Programm zusätzliche Rechte eingeräumt werden. Dazu müssen nach jedem Kompilieren die folgenden Shell-Befehle ausgeführt werden (Das Binärprogramm wird hier im Beispiel „mobkom-programm“ genannt). Daraufhin kann das Programm von der Scratch-Partition ausgeführt werden:

```
echo $USER # sollte den aktuellen Benutzernamen ausgeben
glukas
mkdir -p /scratch/$USER
cp mobkom-programm /scratch/$USER
sudo mobkom.sh mobkom-programm
Allowing RAW socket access to /scratch/glukas/mobkom-programm
/scratch/$USER/mobkom-programm
...eigene Ausgaben...
```

# *Öffnen eines RAW-Sockets*

Um einen RAW-Socket zu erzeugen, an die WLAN-Schnittstelle zu binden und verwendbar zu machen, müssen mehrere Schritte befolgt werden. Diese werden im folgenden dargestellt. Die Abfrage aufgetretener Fehler wird dabei aus didaktischen Gründen ausgelassen.

## Erforderliche Include-Dateien

```
#include <netpacket/packet.h>
#include <net/ethernet.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
```

## Öffnen des Sockets

Mit dem folgenden Befehl wird ein RAW-Socket geöffnet, der mit dem Pakettyp für die Datenpakete der Übung verknüpft ist:

```
int rawsock;

rawsock = socket(PF_PACKET, SOCK_RAW, htons(0x5507));
```

Der Rückgabewert muss darauf überprüft werden, ob ein Fehler aufgetreten ist, ansonsten kann der Socket weiter verwendet werden.

## Binden an die WLAN-Schnittstelle

Damit man über den Socket Pakete über das WLAN-Interface versenden und empfangen kann, muss er an die Schnittstelle gebunden werden. Dazu wird erst mittels ioctl() die Nummer der Schnittstelle ermittelt (der übergebene Name muss an das lokale System angepasst werden!), und dann mit bind() die Bindung angefordert (in addr wird dabei der Pakettyp und die Nummer der Schnittstelle übergeben):

```
struct ifreq req;
struct sockaddr_ll addr;

strncpy(req.ifr_name, "eth1", IFNAMSIZ);
ioctl(rawsock, SIOCGIFINDEX, &req); /* Fehlerabfrage! */

addr.sll_family = AF_PACKET;
addr.sll_protocol = htons(0x5507);
addr.sll_ifindex = req.ifr_ifindex;
bind(rawsock, (struct sockaddr*)&addr, sizeof(addr)); /* Fehlerabfrage! */
```

## Ermitteln der MAC-Adresse

Um die eigene MAC-Adresse zu erfahren (die für den Versand von Paketen erforderlich ist), kann ein weiterer ioctl()-Befehl eingesetzt werden – auch hier muss der Name der Netzwerkkarte angepasst werden:

```
char meine_mac[ETHER_ADDR_LEN];

strncpy(req.ifr_name, "eth1", IFNAMSIZ);
ioctl(rawsock, SIOCGIFHWADDR, &req); /* Fehlerabfrage! */
memcpy(meine_mac, req.ifr_hwaddr.sa_data, ETHER_ADDR_LEN);
```

## Arbeiten mit Zeitstempeln

Um eine Uhrzeit (absoluter Wert) oder eine Zeitdifferenz (relativer Wert) abzubilden, wird die folgende Struktur verwendet (sie ist bereits in den genannten Headern deklariert):

```
#define _BSD_SOURCE 1
#include <sys/time.h>
#include <time.h>

struct timeval {
      time_t            tv_sec;     /* seconds */
      suseconds_t tv_usec;    /* microseconds */
};
```

Elementare Manipulationsmöglichkeiten sind:

```
struct timeval TIMEOUT = { 3, 0 }        /* relativ: 3 Sekunden */
struct timeval now, abs_to, rel_to;      /* Variablen */

gettimeofday(&now, NULL);                /* absolut: Zeit seit 1.1.1970 */
timeradd(&now, &TIMEOUT, &abs_to);       /* absolut: jetzt + 3 Sekunden */
timersub(&abs_to, &now, &rel_to);        /* relativ: Timeout */
if (timercmp(&now, &abs_to, <=)) {
      printf("Timeout noch nicht erreicht\n");
}
```

## Verwendung der poll()-Funktion

Die poll()-Funktion dient dazu, gleichzeitig Daten von einem oder mehreren Desktriptoren (d.h. Sockets, Dateien oder Standardeingabe) zu erwarten, und periodisch eigene Handlungen (z.B. HELLO-Nachrichtenversand) durchführen zu können. Dazu müssen die zu überwachenden Deskriptoren in einem Array hinterlegt und an poll() übergeben werden.

```
// Beispiel steht aus
```

## Ablauf einer Dateiübertragung

Hier ein Beispiel für die Übertragung einer 1600 Byte großen Testdatei:

| | |
|---|---|
| Sender schickt | [OFFER, size_param=1600, datalen=8, data="test.txt"] |
| Empfänger schickt | [DATAREQ, size_param=0, datalen=8, data="test.txt"] |
| Sender schickt | [DATA, size_param=0, datalen=1024, data=<Dateibytes 0..1023>] |
| | (Paket kommt nicht an, Timeout verstreicht) |
| Empfänger schickt | [DATAREQ, size_param=0, datalen=8, data="test.txt"] |
| Sender schickt | [DATA, size_param=0, datalen=1024, data=<Dateibytes 0..1023>] |
| Empfänger schickt | [DATAREQ, size_param=1024, datalen=8, data="test.txt"] |
| Sender schickt | [DATA, size_param=1024, datalen=576, data=<Dateibytes 1024..1599>] |

Die Übertragung ist abgeschlossen.

## Verwendung der Python-GTK-GUI

Statt einer eigenen Programmoberfläche besteht die Möglichkeit, auf eine Python-GTK-Schnittstelle zurückzugreifen. Diese kann unter der folgenden URL heruntergeladen werden:

http://ivs.cs.uni-magdeburg.de/EuK/lehre/lehrveranstaltungen/ss10/mobkom/frontend.py

Die Anbindung erfolgt über einen TCP-Socket: die MobKom-Anwendung arbeitet dabei als TCP-Server und die GUI als Client. Die Kommunikation erfolgt in Form von Textnachrichten. Dabei kann das Frontend benutzt werden, um Chat-Nachrichten an andere Teilnehmer zu versenden,

Nachrichten anderer Benutzer anzuzeigen sowie um der Applikation mitzuteilen welche Dateien sie von anderen empfangen bzw. an andere senden soll.

## *Verwendung eines TCP-Server-Sockets*

Die GUI verbindet sich über TCP auf den Port 5507 des lokalen Systems („localhost“ bzw. 127.0.0.1). Dazu muss von der MobKom-Anwendung der entsprechende Server-Port eingerichtet werden:

```
int lst_sock, frontend;
struct sockaddr_in sa;

lst_sock = socket(PF_INET, SOCK_STREAM, 0); /* Fehlerabfrage! */
memset(&sa, 0, sizeof(sa));
sa.sin_family = AF_INET;
sa.sin_port = htons(5507);
bind(lst_sock, (struct sockaddr*)&sa, sizeof(sa)); /* Fehlerabfrage! */
listen(lst_sock, 1); /* Fehlerabfrage! */
```

Danach ist auf dem System der TCP-Port 5507 offen und eine ankommende Verbindung kann akzeptiert werden. Der accept-Befehl liefert dabei einen TCP-Socket für die eigentliche Verbindung zurück, auf dem mit read() und write() Nachrichten empfangen / versendet werden können, und der in eine select()-Umgebung eingebunden werden kann:

```
frontend = accept(lst_sock, NULL, 0); /* Fehlerabfrage! */
```

## *Kommunikation mit der GUI*

Die Kommunikation mit der Oberfläche erfolgt nachrichtenbasiert. Jede Nachricht bildet eine Textzeile (beendet durch das Newline-Zeichen '\n'). MAC-Adressen werden immer als String übertragen. Die Anwendung muss ankommende Netzwerkpakete auswerten und entsprechende Nachrichten an die GUI weiterleiten. Die GUI verarbeitet Benutzereingaben und generiert daraufhin Nachrichten, die an die Anwendung gesendet werden.

### Nachrichten von der Anwendung zur GUI

Bei ankommenden Netzwerkpaketen muss die Anwendung der GUI die aufgetretenen Ereignisse über Textnachrichten mitteilen. Die Felder einer Nachricht werden durch Leerzeichen getrennt. Es gibt die folgenden Nachrichtenformate:

| Typ | Feld 2 | Feld 3 | Feld 4 | Feld 5 | Feld 6 |
|---|---|---|---|---|---|
| START | <eigene MAC> | <eigener Name> | | | |
| JOIN | <Sender-MAC> | <Sender-Name> | | | |
| PART | <Sender-MAC> | <Sender-Name> | | | |
| MSG | <Empf.-MAC> | <Sender-Name> | Seq.-Nr | x | <Nachricht> |
| FOFFER | <Empf.-MAC> | <Sender-Name> | x | Dateigröße | <Dateiname> |
| FRECV | <Empf.-MAC> | <Sender-Name> | x | Empf. Größe | <Statustext> |
| FDONE | <Empf.-MAC> | <Sender-Name> | x | Dateigröße | <Statustext> |
| FABORT | <Empf.-MAC> | <Sender-Name> | x | x | <Statustext> |

Eine Beispielnachricht wäre also:

„MSG ff:ff:ff:ff:ff:ff Georg 16577 0 Hallo Welt!\n“

Damit wird der GUI mitgeteilt, dass eine an alle gerichtete Nachricht von Benutzer Georg empfangen wurde.

JOIN- und PART-Nachrichten werden zur Pflege der Benutzerliste in der GUI genutzt, MSG-Nachrichten als Text im entsprechenden Fenster angezeigt. Den Dateiempfang muss die Anwendung selbst implementieren, lediglich die Steuerung erfolgt dabei über das Frontend. Dafür gibt es die F-Nachrichten FOFFER, FRECV, FDONE und FABORT.

Empfängt die Anwendung ein OFFER-Paket, muss dieses als FOFFER an das Frontend gemeldet werden. Hier wird dem Benutzer die Option gegeben, diese Datei mittels GET anzufordern.

Während des Dateitransfers kann die Applikation regelmäßig mittels FRECV den aktuellen Fortschritt des Transfers melden. Im Statustext kann z.B. die aktuelle Transferrate und die geschätzte verbleibende Zeit gemeldet werden. Nach Abschluss des Transfers muss eine FDONE-Nachricht gesendet werden.

Wird der Transfer vorzeitig abgebrochen, ist eine FABORT-Nachricht mit der Fehlerbeschreibung zu versenden.

## Kommunikation von der GUI zur Anwendung

Die GUI versendet bei bestimmten Benutzer-Aktionen Nachrichten an die Anwendung. Die Nachrichten haben den folgenden Aufbau:

„<Typ> <Empfänger> <Daten>“

| Typ | Empfänger | Daten |
|---|---|---|
| MSG | <Empf.-MAC> | Zu versendende Chat-Nachricht |
| GET | <Empf.-MAC> | Name der zu empfangenden Datei (aus FOFFER-Nachricht) |
| SEND | <Empf.-MAC> | Dateiname, der einem anderen Teilnehmer per OFFER angeboten werden soll |