
Technische Informatik III:
Betriebssysteme und Rechnernetze WS 2007/08
Musterlösung zum Übungsblatt Nr. 6

Aufgabe 1: Begriffe

8 Punkte

Beschreiben Sie jeden der folgenden Begriffe durch maximal zwei Sätze: Polling, Star Topology, Open Shortest Path First, Backward Learning, Bridge, Gateway, VLAN, ARP

- Polling: Ein Verfahren um ein geteiltes Medium zu verwalten. Ein Master fragt zyklisch alle Teilnehmer ab, ob sie etwas zu senden haben.
- Star Topology: Netzwerktopologie, in der alle Teilnehmer mit einem zentralen Teilnehmer (z. B. Server oder auch Switch) direkt verbunden sind.
- Open Shortest Path First: Link-State Routing-Protocol, die das gesamte Netzwerk nachzubilden versucht.
- Backward Learning: Routing-Algorithmus bei dem jeder Knoten eine Tabelle der Adressen mit der Anzahl der Hops gespeichert wird. Jeder Knoten entscheidet selbst, welcher Weg der kürzste ist.
- Bridge: Verbindet zwei Netzwerksegmente miteinander. Es arbeitet auf Schicht 2.
- Gateway: Verbindet Netzwerk miteinander die unterschiedliche Protokolle sprechen. Entspricht der OSI-Schichten 4-7.
- VLAN: Virtual Local Area Network, logisches Netzwerk innerhalb des physikalischen.
- ARP: Address Resolution Protocol arbeitet auf Schicht 2. Es dient dazu zu einer IP-Adresse die zugehörige MAC-Adresse herauszufinden.

Aufgabe 2: Dienste

4 Punkte

Geben Sie vier Dienste der Sicherungsschicht (Schicht 2) an, und erläutern Sie in wenigen Stichpunkten deren Aufgabe und Funktionsweise.

- Rahmenbildung:
 - Bildung von Rahmen fester Größe, aus dem von der Vermittlungsschicht übergebenen Bitstrom.
 - Mit Hilfe von definierten Rahmenbegrenzern oder durch Angabe der Nutzdatenlänge wird der Anfang und das Ende von einem Rahmen signalisiert.
- Flusskontrolle:
 - Kontrolle der übertragenen Datenmengen, um den Empfänger vor Überlastung zu schützen.
 - Mit Hilfe von ACK Paketen, diese bestätigen korrekt empfangene Pakete und zeigen somit auch an, dass der Empfänger bereit für neue Pakete ist (z.B. Schiebefensterprotokolle, Alternating-Bit-Protokoll).
- Medienzugriff:
 - Kontrolle des Zugriffs auf das geteilte Medium zur Vermeidung von Kollisionen und damit verbundenem Paketverlust.
 - Verschiedene Protokolle stehen, je nach Anwendungsmedium und –umgebung, zur Verfügung (z. B. Aloha, CSMA/CD, CSMA/CA).
- Fehlererkennung und –korrektur:
 - Überprüfung der übertragenen Rahmen auf Übertragungsfehler und ggf. Korrektur dieser Fehler.
 - Prüfsummen (Parität, CRC) bzw. Vorwärtsfehlerkorrektur (z.B. Mehrfachübertragung mit XOR).

Aufgabe 3: Sliding Window Algorithmus

6 Punkte

Der Sliding-Window-Algorithmus wird für die Flusssteuerung und sichere Datenübertragung eingesetzt. Bei der Kommunikation verwaltet der Sender eine Senderfenstergröße (Send Window Size, SWS) und der Empfänger eine Empfängerfenstergröße (Receive Window Size, RWS).

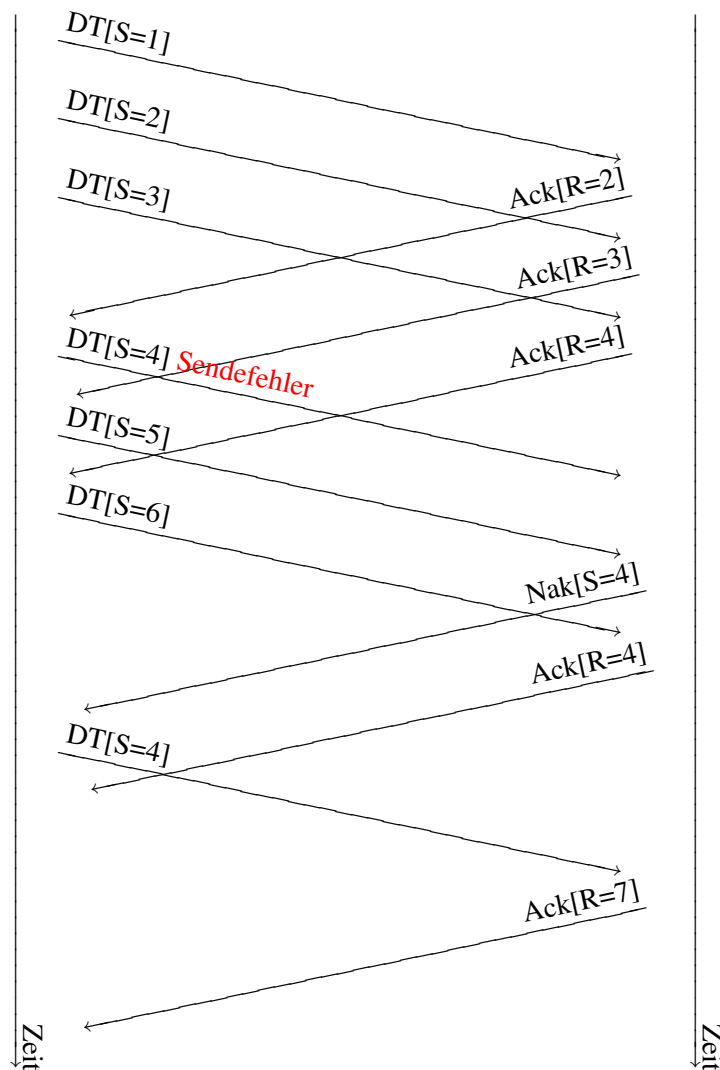
Zeichnen Sie ein Zeitstrahldiagramm für den Sliding-Window-Algorithmus mit $SWS=RWS=3$ Frames für die beiden folgenden Situationen. Verwenden Sie ein Timeout-Intervall von $2 \times RTT$ (Round-Trip-Time).

- Es wird 6 Frames gesendet und Frame 4 geht verloren.

Selective Repeat:

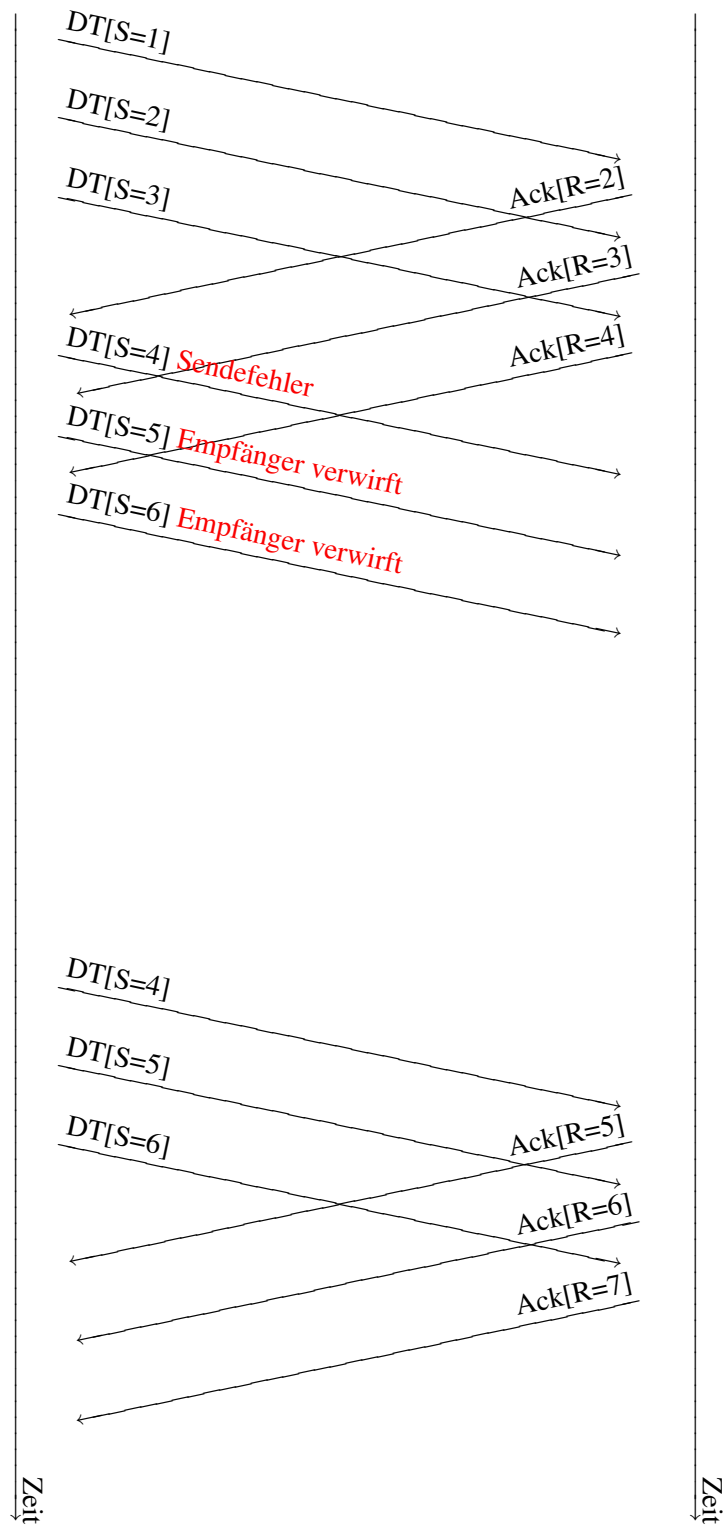
Sender

Empfänger



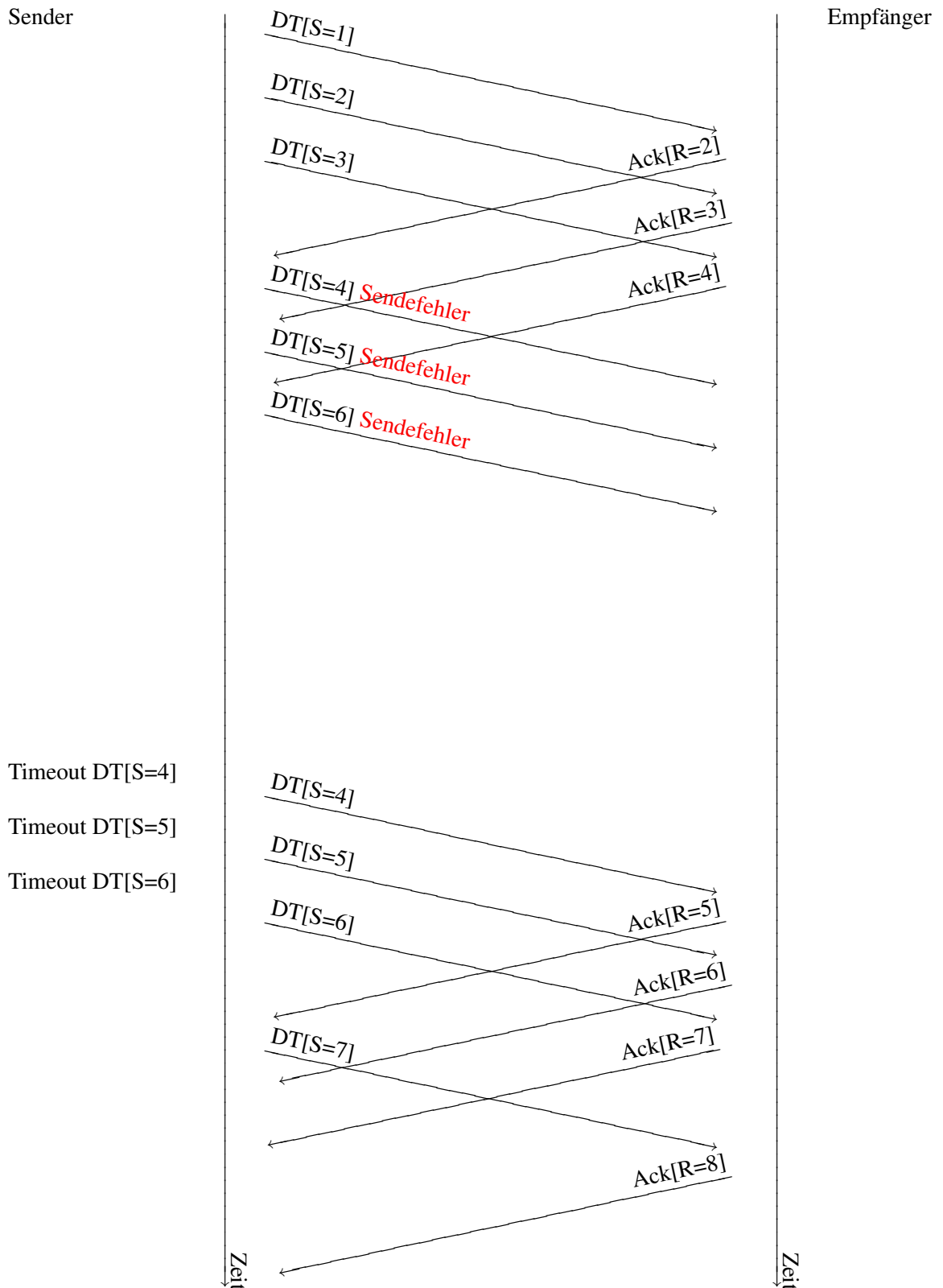
Go-Back-N:
Sender

Empfänger



- Es wird 7 Frames gesendet und Frames 4 bis 6 gehen verloren.

Hier sehen *Go-Back-N* und *Selective Repeat* gleich aus:



Aufgabe 4: Fifty/Fifty

8 Punkte

Entscheiden Sie, ob folgenden Aussagen zutreffend oder nicht zutreffend sind, und begründen Sie Ihre Entscheidung:

- Wenn ein IP-Paket beim Router des Heimnetzes des Empfängers ankommt, so sendet der Router dieses mit einem Broadcast ins LAN, wo es alle angeschlossene Rechner auf die zutreffende IP-Adresse hin untersuchen.

Falsch, der Router weiß, oder fragt per ARP nach, welcher Rechner die zutreffende IP-Adresse hat und sendet das IP-Paket direkt an diesen weiter.

- **Ob in einem konkreten Protokoll Vorwärtsfehlerkorrektur zum Einsatz kommt, hängt auch vom verwendeten physikalischen Medium auf der unteren Schicht ab.**

Richtig, man kann die Anzahl der Übertragungswiederholungen durch Bitfehlern reduzieren, indem man bei störungsanfälligen Medium (z.B. Luft bei WLAN) Vorwärtsfehlerkorrektur auf Schicht 2 einsetzt.

- **Gerade bei niedriger Auslastung kann auf den unnötigen Overhead der Prüfsumme im Paketkopf verzichtet werden.**

Falsch, gerade bei niedriger Auslastung macht der Overhead der Prüfsumme wenig aus.

- **Bursty Traffic kann über die vielen Kanäle des Multiplexing gut entgegengewirkt werden.**

Falsch, intensives Datenaufkommen hat nichts mit dem Bündeln mehrerer Datenkanäle zu tun.

Aufgabe 5: Scheduler II

14 Punkte

Passen Sie den Scheduler des Linux-Kernels so an, dass immer, wenn das aktive und das abgelaufene Prioritätsarray zum 100 Mal vertauscht werden, mit der Funktion *printk()* eine Nachricht auf der Konsole ausgegeben wird.

kernel/sched.c basicstyle

```

1  [...]
2  [...]
3
4  /*Zeile 412: Deklaration und Initialisierung einer globalen
   Zählvariable */
5  int array_switch_count = 0;
6
7  [...]
8
9  /*Zeile 3294: Anfang der schedule()-Funktion */
10 /*
11  * schedule() is the main scheduler function.
12  */
13 asmlinkage void __sched schedule(void)
14 {
15
16  [...]
17
18  /* Zeile 3379: Die Queues active und expired werden vertauscht */
19  if (unlikely(!array->nr_active)) {
20      /*
21       * Switch the active and expired arrays.
22       */
23      schedstat_inc(rq, sched_switch);
24      rq->active = rq->expired;
25      rq->expired = array;
26      array = rq->active;
27      rq->expired_timestamp = 0;
28      rq->best_expired_prio = MAX_PRIO;
29      /* Zeile 3389: Inkrementieren der globalen Zählvariable */
30      ++array_switch_count;
31  }
32
33  [...]
34
35  if (unlikely(test_thread_flag(TIF_NEED_RESCHED)))
36      goto need_resched;
37  /* Zeile 3455: Überprüfung, ob die globalen Zählvariable ausgegeben
   werden muss */

```

```

38     if(array_switch_count >= 0)
39     {
40         printk("Array was switched a 100 times!\n");
41         array_switch_count = 0;    /* Zurücksetzen */
42     }
43 } /* Ende von schedule() */
44 EXPORT_SYMBOL(schedule);
45
46 [...]

```

Testlauf, Booten des Systems:

```

Starting up general console font...Array was switched a 100 times!
Setting up per-VC fonts...done

```

Aufgabe 6: Webserver II

10 Punkte

Testen Sie Ihre Implementierung eines einfachen Webserver aus dem vorhergehenden Übungsblatt auf sicherheitsrelevante Schwachstellen:

1. ASCII-Zeichen:

```

chris@firechris-nb:~$ nc localhost 8080
GET/\\\\\\HH&&& HTTP/1.0
HTTP/0.9 404 Not Found
chris@firechris-nb:~$ nc localhost 8080
GET /makefile\0
HTTP/0.9 404 Not Found
chris@firechris-nb:~$ nc localhost 8080
GET /makefile\n\n\n HTTP/1.0
HTTP/0.9 404 Not Found
chris@firechris-nb:~$

```

Die ASCII-Zeichen beeinflussen den Server nicht, der sendet nur 404, da er so eine Datei nicht findet.

2. Pfade außerhalb des Document Root:

```

chris@firechris-nb:~$ nc localhost 8080
GET ../../../../../../etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh

[...]

chris@firechris-nb:~$ nc localhost 8080
GET ../simpleserv.c
/**
 * @file simpleserv.c
 *
 * TI III: Betriebssysteme und Rechnernetze
 * WS 2007/08
 * Übungsblatt Nr. 5,
 * Aufgabe 6: Webserver
 *
 * Dieses Programm implementiert einen HTTP/0.9 - Server.
 * Es sendet auf GET Anfragen angefragte Dateien oder einen
 * HTTP-Fehlercode.

```

```
chris@firechris-nb:~$ nc localhost 8080
GET ../01_Uebung/makefile
CC = gcc      # Kompiler
FLAGS = -Wall -g # Compileroptionen
```

Da der Server einfach Zeichen an den Doc-root anhängt, kann man beliebige Dateien auslesen. Da kann man die Funktion

```
1 #include <sys/param.h>
2 #include <unistd.h>
3
4 char *realpath(char *path, char resolved_path[]);
```

```

1  [...]
2  [...]
3
4  /* Absoluter Pfad */
5  realpath(file_path , rpath);
6  /* Prüfe Anfang auf Doc-root */
7  start = strstr(rpath , path);
8
9  if(start != rpath)
10 {
11     /* Anfang ist nicht Doc-root */
12     printf("%s:%d: File %s is not in Doc-root \n",
13           cl_addr , cl_port , rpath);
14     rpath[0] = '\0';
15 }
16
17 [...]

```

```
chris@firechris-nb:~$ nc localhost 8080
GET /makefile
CC = gcc          # Kompiler
FLAGS = -Wall -g -pedantic      # Compileroptionen
```

```
chris@firechris-nb:~$ nc localhost 8080
GET ../simpleserv.c
HTTP/0.9 404 Not Found
```

```
chris@firechris-nb:~$ nc localhost 8080
GET aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
HTTP/0.9 404 Not Found
chris@firechris-nb:~$ nc localhost 8080
```

Die langen Request beeinflussen den Server nicht, da der Buffer nur eine Maximale anzahl von Zeichen einliest.