

## 1. Begriffe

**Polling** Polling ist eine Möglichkeit, wie man die Kommunikation zwischen mehreren Stationen kontrollieren kann. Hierbei ist ein Teilnehmer als Master eingestuft, der zyklisch alle Slaves fragt, ob sie was zu senden haben und nur nach Anfrage darf der Slave dann antworten.

**Star Topology** Ist eine sternförmige Anordnung der Netzwerkteilnehmer. Dabei ist der im Zentrum liegende Teilnehmer eine Art Master, der die Anfragen weiterleitet.

**Open Shortest Path First** Das ist ein sehr häufiges Routing-Protokoll, das nach dem Dijkstra-Algorithmus den kürzesten Pfad berechnet.

**Backward Learning** Ist ein Algorithmus, wie z.B. ein Switch lernt, an welchem Port und mit welcher Adresse ein Teilnehmer hängt. Dabei wird bei einer Anfrage von jemandem, dessen Daten unbekannt sind, erstmal die Daten gespeichert und dann entweder an alle Teilnehmer geschickt oder nur an einen, wenn der Empfänger schon bekannt ist.

**Bridge** Eine Bridge verbindet verschiedene Netzwerke, die auch von unterschiedlichen Typ sein können (z.B. LAN und WLAN). Das ganze geschieht auf Layer 2.

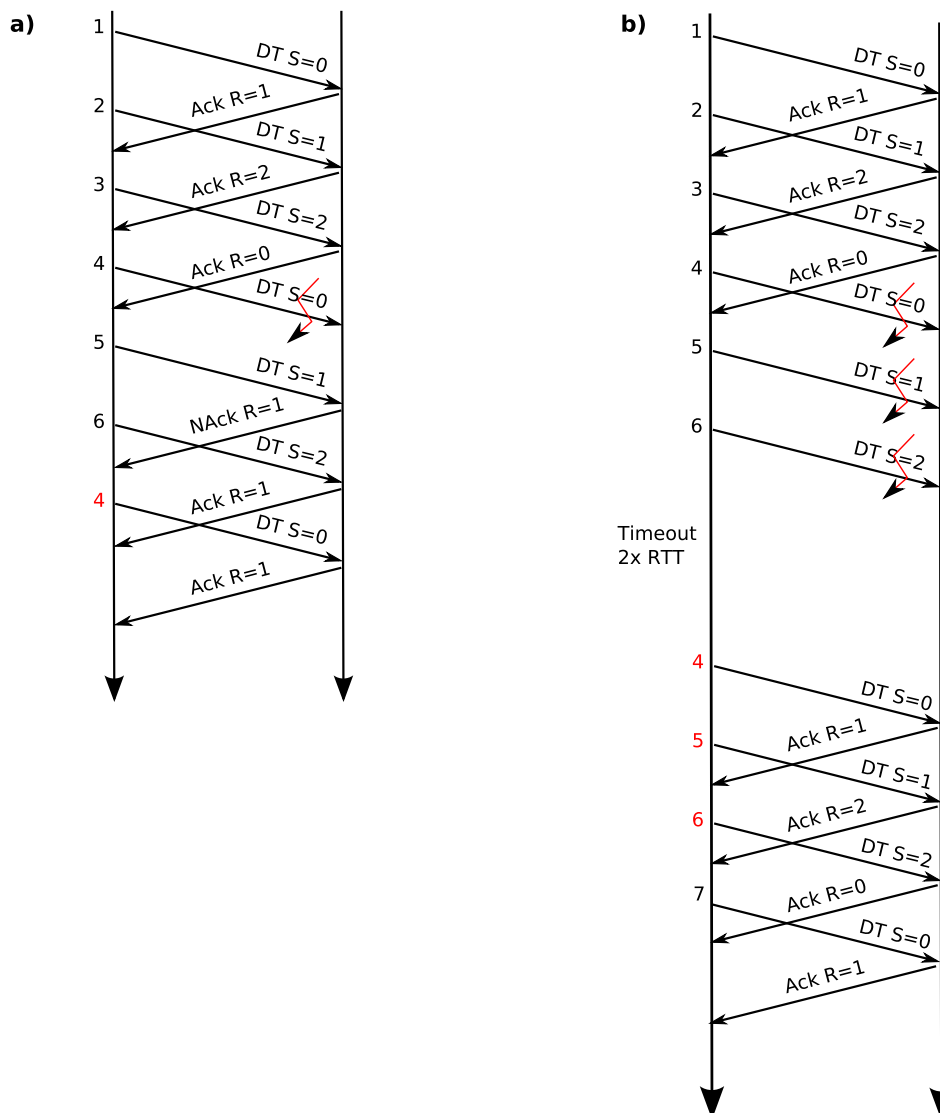
**Gateway** Ein Gateway verbindet verschiedene Teilnehmer auf der Anwendungsebene bzw. Layer 4 und höher. So können von einem Webserver z.B. per Gateway proprietäre Systeme wie Datenbanken erreicht werden.

**VLAN** Das ist ein virtuelles Netzwerk. Und zwar können Netzwerke nicht immer nach ihrer logischen Struktur aufgebaut werden, um trotzdem eine logische Struktur zu haben, legt man dann ein logisches drüber; dabei müssen sich Switches bzw. Bridges merken, welcher Port zu welchem VLAN gehört.

**ARP** Das ist das "Address Resolution Protocol". Mit diesem Protokoll findet der Router raus, welche MAC-Adresse er welcher IP zuordnen muss.

## 2. Dienste

### 3. Sliding Window Algorithmus



#### 4. Fifty/Fifty

- 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.  
Nein. Es wird nicht das ganze IP-Paket gesendet, sondern nur die IP wird gebroadcastet. Dann kann der Empfänger mit seiner MAC antworten und erst dann wird das Paket übermittelt.
- Ob in einem konkreten Protokoll Vorwärtsfehlerkorrektur zum Einsatz kommt, hängt auch vom verwendeten physikalischen Medium auf der unteren Schicht ab.

Nein. FEC ist nicht abhängig von dem physikalischen Medium. Um der Information Redundanz hinzuzufügen bedarf es keine speziellen physikalischen Eigenschaften.

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

Nein. Es ist ja nicht von der Auslastung abhängig, ob ein Paket verloren geht oder fehlerhaft wird. Also ist die Prüfsumme in jedem Fall sinnvoll.

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

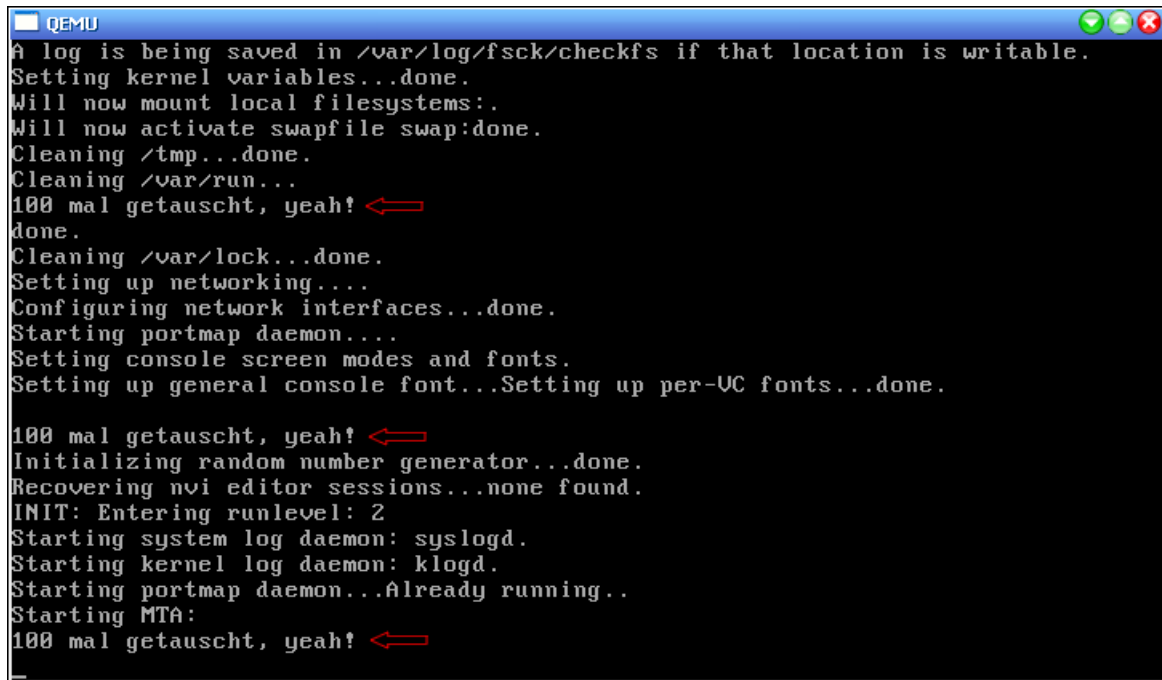
Nein. Multiplexing ist nicht geeignet um Bursty Traffic entgegenzuwirken.

## 5. Scheduler II

Die *sched.c* muss wie folgt in der *schedule()* Funktion in der *if* Abfrage ab Zeile 3380 geändert werden:

```
1 if (unlikely(!array->nr_active)) {
2     /*
3      * Switch the active and expired arrays.
4      */
5     static int tauschzaehler = 0;
6
7     schedstat_inc(rq, sched_switch);
8     rq->active = rq->expired;
9     rq->expired = array;
10    array = rq->active;
11    rq->expired_timestamp = 0;
12    rq->best_expired_prio = MAX_PRIO;
13    tauschzaehler++;
14    if(tauschzaehler == 100) {
15        printk("\n100 mal getauscht, yeah!\n");
16        tauschzaehler = 0;
17    }
```

Nach erfolgreichem Compilieren und Neustart des veränderten Kernels ist erst nichts ungewöhnliches festzustellen, doch dann wird nach einer gewissen Zeitspanne, wenn nicht nur der init-Prozess mehr läuft, alle paar Sekunden die Ausgabe "100 mal getauscht, yeah" ausgegeben. Mehr passiert auch nicht.



```
QEMU
A log is being saved in /var/log/fsck/checkfs if that location is writable.
Setting kernel variables...done.
Will now mount local filesystems:.
Will now activate swapfile swap:done.
Cleaning /tmp...done.
Cleaning /var/run...
100 mal getauscht, yeah! <=>
done.
Cleaning /var/lock...done.
Setting up networking....
Configuring network interfaces...done.
Starting portmap daemon....
Setting console screen modes and fonts.
Setting up general console font...Setting up per-VC fonts...done.

100 mal getauscht, yeah! <=>
Initializing random number generator...done.
Recovering nvi editor sessions...none found.
INIT: Entering runlevel: 2
Starting system log daemon: syslogd.
Starting kernel log daemon: klogd.
Starting portmap daemon...Already running..
Starting MTA:
100 mal getauscht, yeah! <=>
```

## 6. Webserver II

**Variierung der GET Anfrage** Es wurde auf mehreren Arten versucht, die vom Client gesendete GET abfrage abzuändern.

- Es wurde GET ohne jeglichen weiteren Parameter gesendet. Der Webserver meldet einen "Fehler mit GET:Bad file descriptor". Dabei wird die Verbindung zum Client geschlossen, aber der Server stürzt nicht ab, da die Meldung per perror() ausgegeben wird.
- Es wurde POST / HTTP/1.1 gesendet. Auch hier wird wieder der "Fehler mit GET" gemeldet. Auch hier wird die aktuelle Verbindung beendet, aber der Server bleibt am laufen.
- Als letztes wurde nur ein \0 übertragen. Auch hier tritt das gleiche Verhalten wie in den vorherigen Fällen auf.

Es kommt bei einer fehlerhaften GET-Anfrage daher nicht zu einem Fehler, da bei der Eingabe per strtok( " )" nach dem ersten Leerzeichen getrennt wird und dann der vordere Teil mit GET abgeglichen wird. Ist es hier kein GET, dann kommt der besagte "Fehler mit GET". Zudem ist der ganze Lesen und Schreiben Prozess in eine eigene Funktion eingebettet, so dass dann nur diese Funktion fehlerhaft terminiert, aber nicht der eigentliche Webserver.

**Fehlerhafte / nicht erlaubte Anfragen** Es wird versucht auf verschiedene Dateien und Pfade zugegriffen.

- Es wird eine nicht vorhandene Datei angefragt. Hier passiert auch nix, wie erwartet wird eine 404-Meldung zurückgegeben und als Webseite im Browser angezeigt. In einer if-Abfrage wird natürlich getestet, ob der Pfad vorhanden ist bzw. lesbar (per `access(pfad,R_OK)`), ansonsten wird immer die 404-Meldung geworfen.
- Es wird versucht per `GET ../index.html` auf das nächsthöhere Verzeichnis zuzugreifen (die `index.html` ist dort vorhanden). Leider leider wird sie auch zurückgegeben. Ohne Fehler und Murren. Man kann diese Schwachstelle aber beheben, in dem man in der Funktion `lesenSchreiben(int sockRem, char *verz)` den Bereich ab Zeile 46 wie folgt um eine zusätzliche if-Bedingung erweitert:

```
1 datei[0] = strtok(bufRead, "_"); //Bei richtiger
    Anfrage GET
2 datei[1] = strtok(NULL, "_"); // Anfrage Datei
3 if(strcmp(datei[0], "GET") != 0) {
4     perror("Fehler mit GET");
5     return -1;
6 }
7 if(strstr(datei[1], "..") != 0) {
8     perror("Höheres Verzeichnis verboten!");
9     return -1;
10 }
11 printf("Die GET Anfrage wurde gelesen, Datei %s wurde
    angefragt.\n", datei[1]);
```

Jetzt wird in dem extrahierten Dateinamen bzw. Pfad nach dem Substring ".." gesucht, wenn er vorhanden sein sollte, wird ein Fehler ausgegeben.

**Überlauf von Variablen** Es wird versucht durch zu lange Anfragen die Variablen zum Überlaufen zu zwingen. Der Buffer für den InputReader liegt bei 512, also wird versucht durch längere Anfragen den Server zum Absturz zu bringen.

- Als erstes ein zu langer Dateiname: `GET /aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...`  
Auch hier bleibt der Webserver cool. Alles, was zu viel ist wird ignoriert, es kommt ein Fehler nicht gefunden Fehler und das wars.
- Jetzt das ganze mit dem GET: `GETaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...`  
Auch hier passiert nix. Gibt wieder einen Fehler bei der GET-Anfrage.
- Und auch wenn man hinter das Verzeichnis noch versucht einen Überlauf zu produzieren (`GET / aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...`), klappt das nicht.

Auch hier also keine Probleme. Das kommt daher, dass wir schön mit `setvbuf()` die Ausgabe puffere. Die Puffergröße beträgt wie gesagt 512. Zudem ist die Variable `datei[]` in dem dann u.a. der Pfad abgelegt wird, ein Array von Pointern, so dass die Größe variable allokiert wird.