Split-Access-Routing und Priorisierung auf Linux-Basis

Oluf Lorenzen

axxeo GmbH

Januar 2010

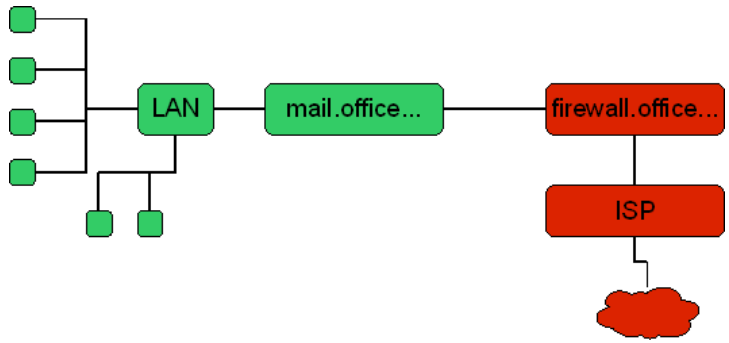
Gliederung

Installationsumfeld

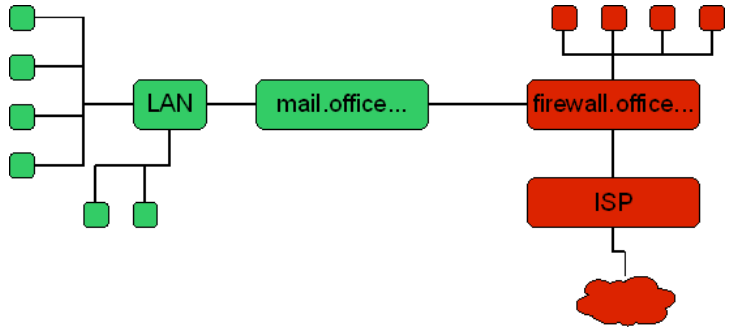
- 4 Mitarbeiter
- Linux-Server für Geschäftskunden
  - als Gateway ins Internet
  - Spam- und Viren-Scan
  - Routing und VPN
- Support/Wartung über SSH (Secure Shell)

Ist-Zustand

- 2Mbit SDSL
- 4 Arbeitsplätze, 2 Server
- Wartung via SSH
- geringer HTTP-Traffic

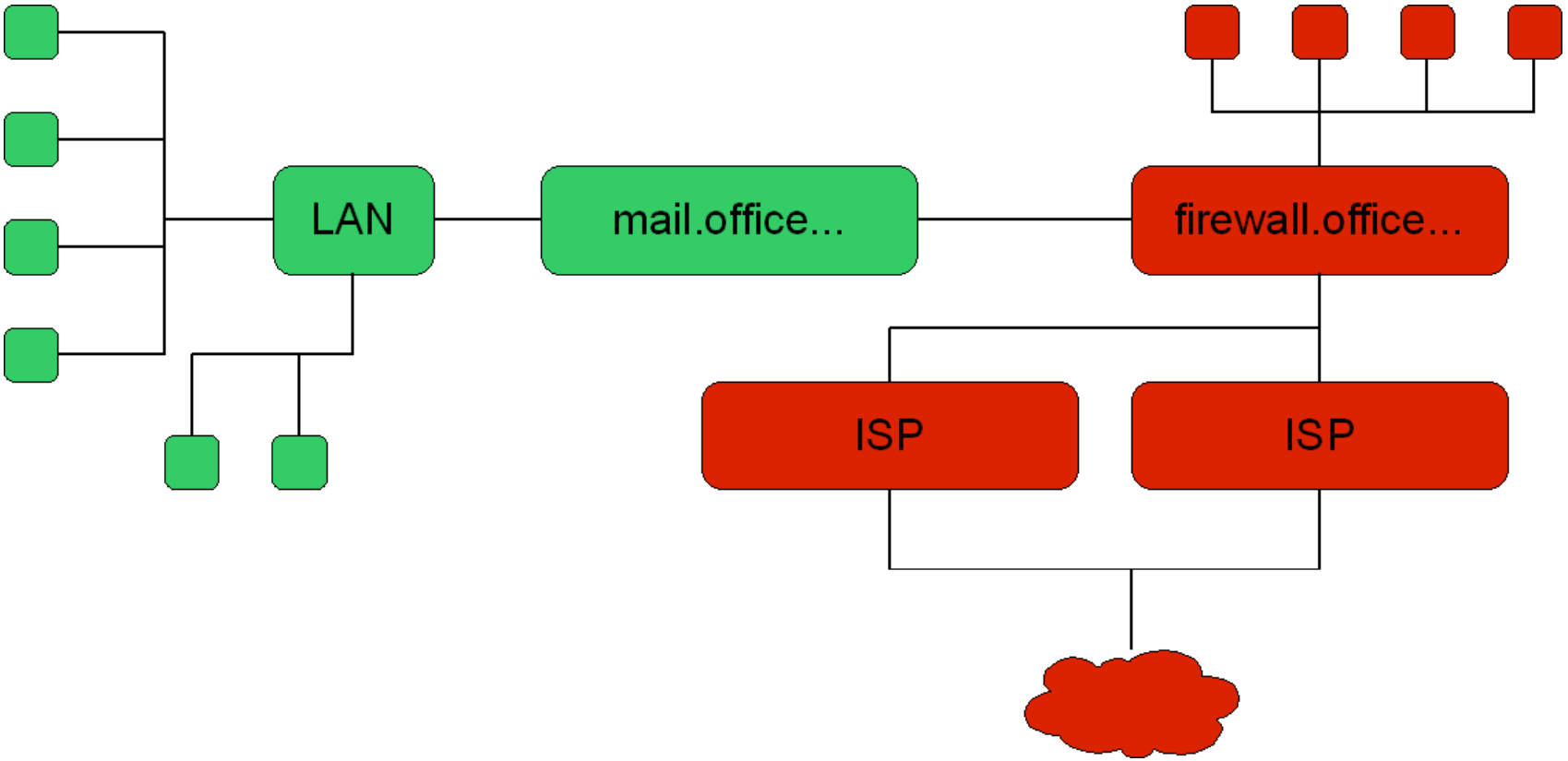


- 2Mbit SDSL
- 5-7 Arbeitsplätze, 2 Server
- Wartung via SSH
- viel (HTTP-)Traffic



Soll-Zustand

- auf Debian basierte Lösung
- LARTC (Linux Advanced Routing & Traffic Control)
- `tc`
- Arbeit bei Netzwerkauslastung möglich
- Nutzung einer zweiten Anbindung


LAN
mail.office...
firewall.office...
ISP
ISP

Split-Access-Routing

- LARTC
- 2 Routing-Tabellen
- eine Default-Route für jede Tabelle
- modifizierte *normale* Default-Route
- Aufteilung des Verkehrs von eins zu eins

Priorisierung

tc-Befehle:

```
tc filter add dev eth1 parent 1:0 protocol all prio 1 handle 4:0:0 u32 divisor 1
tc filter add dev eth1 parent 1:0 protocol all prio 1 u32 match u8 0x11 0xff at
    9 offset at 0 mask 0f00 shift 6 eat link 4:0:0
tc filter add dev eth1 parent 1:0 protocol all prio 1 handle 4:0:1 u32 ht 4:0:0
    match u16 0x35 0xffff at 2 classid 1:1
tc filter add dev eth1 parent 1:0 protocol all prio 1 handle 5:0:0 u32 divisor 1
tc filter add dev eth1 parent 1:0 protocol all prio 1 u32 match u8 0x6 0xff at 9
     match u16 0x0 0xfe00 at 2 offset at 0 mask 0f00 shift 6 eat link 5:0:0
tc filter add dev eth1 parent 1:0 protocol all prio 1 handle 5:0:1 u32 ht 5:0:0
    match u16 0x50 0xffff at 2 classid 1:1
tc filter add dev eth1 parent 1:0 protocol all prio 1 handle 6:0:0 u32 divisor 1
tc filter add dev eth1 parent 1:0 protocol all prio 1 u32 match u8 0x6 0xff at 9
     match u8 0x10 0x10 at 1 offset at 0 mask 0f00 shift 6 eat link 6:0:0
tc filter add dev eth1 parent 1:0 protocol all prio 1 handle 6:0:1 u32 ht 6:0:0
    match u16 0x16 0xffff at 0 classid 1:2
tc filter add dev eth1 parent 1:0 protocol all prio 1 handle 7:0:0 u32 divisor 1
```

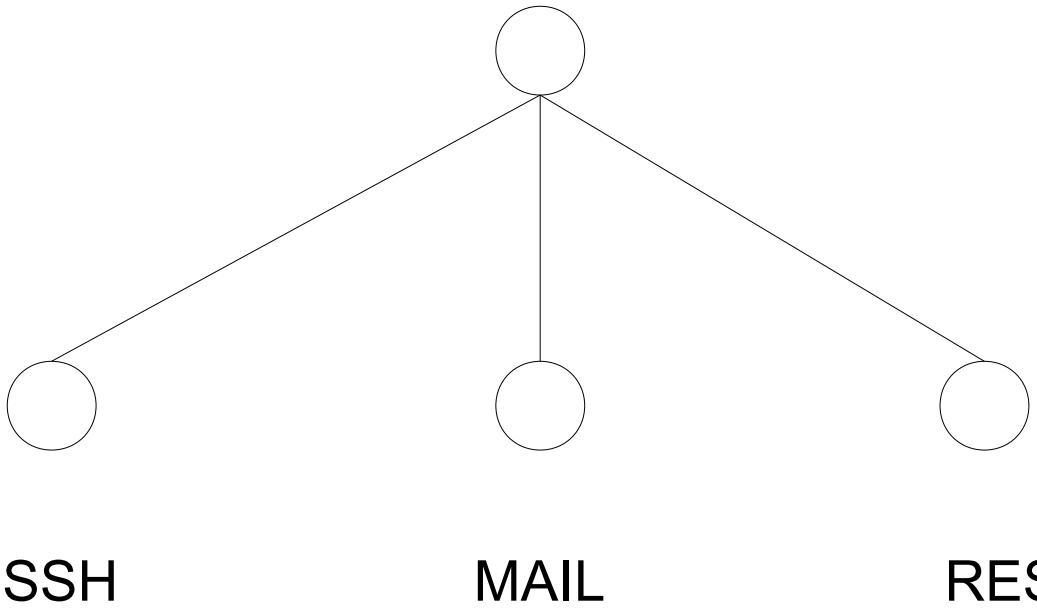
$\Rightarrow$ schwer verständlich, unleserlich ...

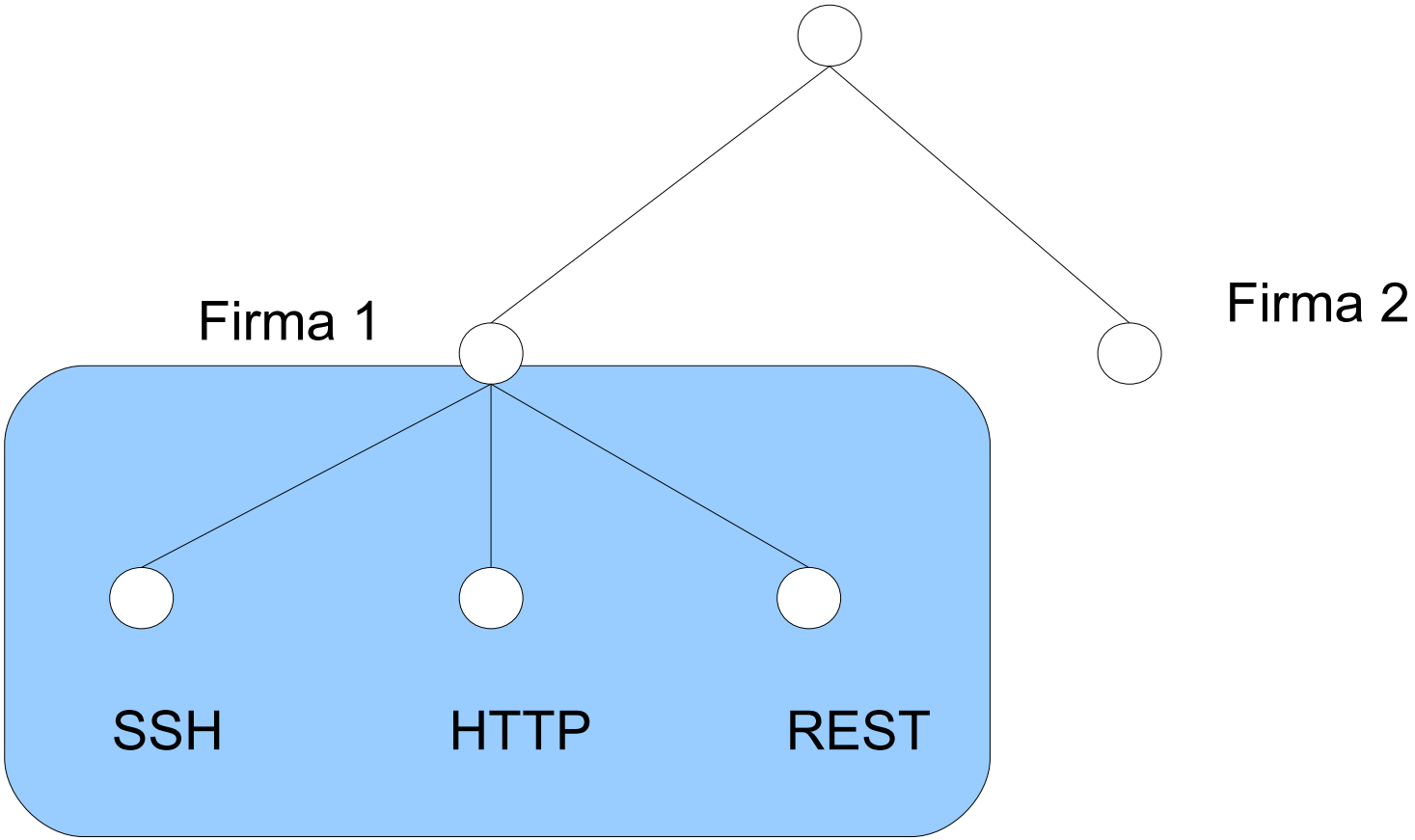
tcng-Script:

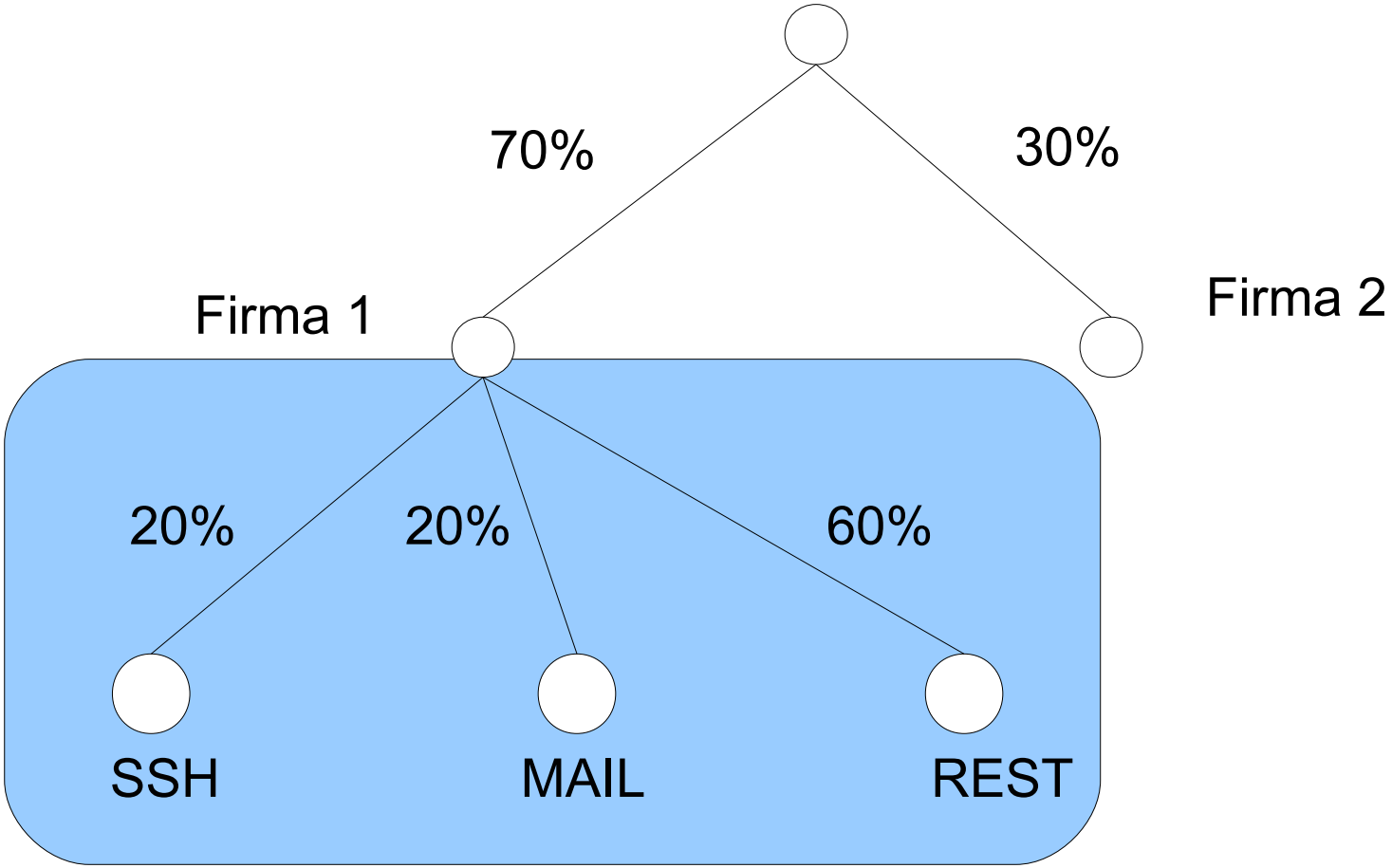
```
dev "eth1"{
    egress {
        class ( <$interactive_in> )
            if ip_tos_delay == 1 && tcp_sport == 22
            if ip_len < 256 && tcp_sport == 22;

        class ( <$other> ) if 1;

        htb () {
            class ( rate 200kbps, ceil 200kbps ) {
                $interactive_in = class ( rate 50kbps, ceil 100kbps ) {sfq;};
                $other = class ( rate 100kbps, ceil 140kbps ) {sfq;};
}}}}
```

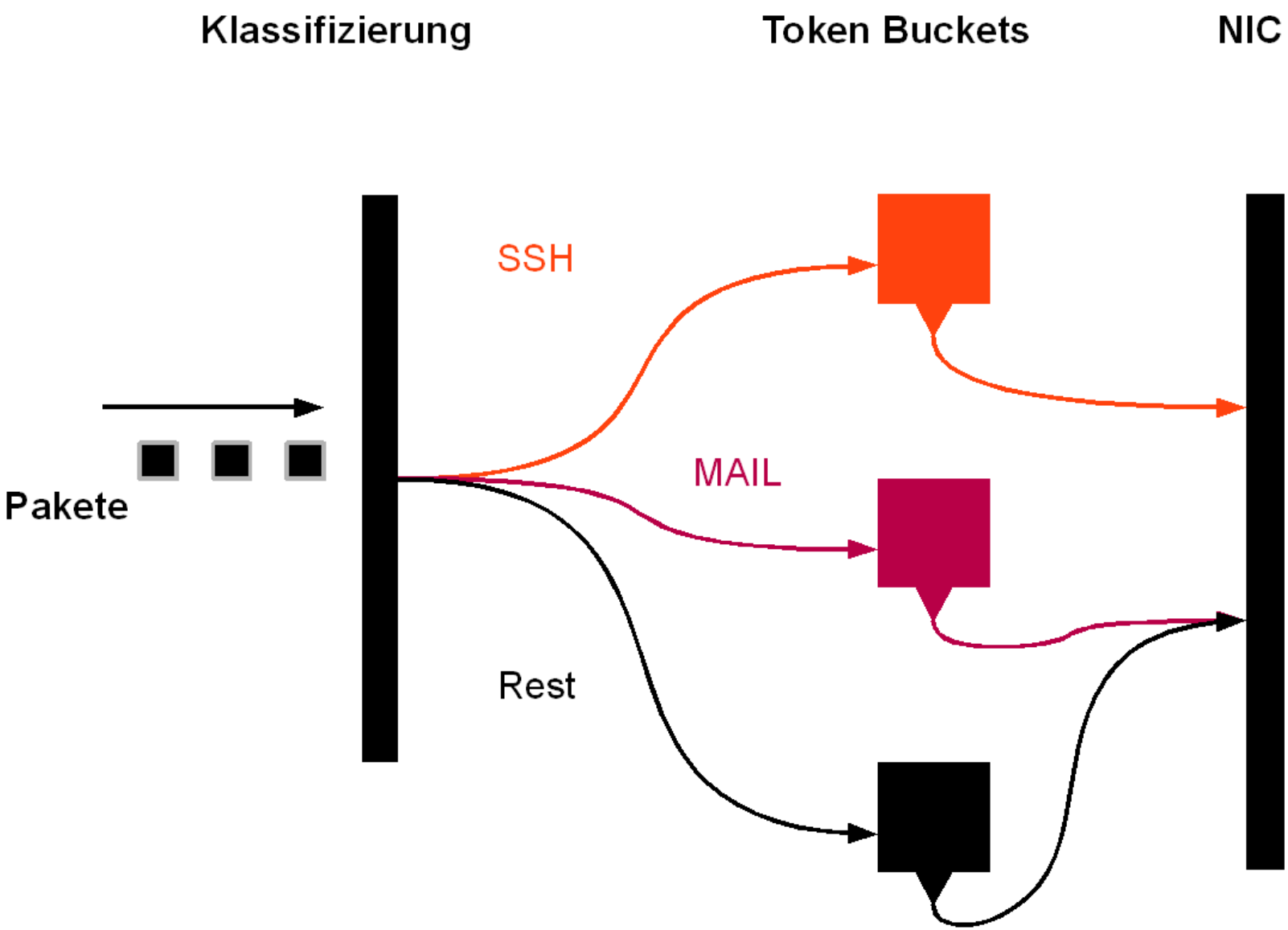
$\Rightarrow$ C-ähnliche Syntax (Syntaxhighlighting!), lesbar, schnell erweiterbar ...

Traffic-Kategorisierung

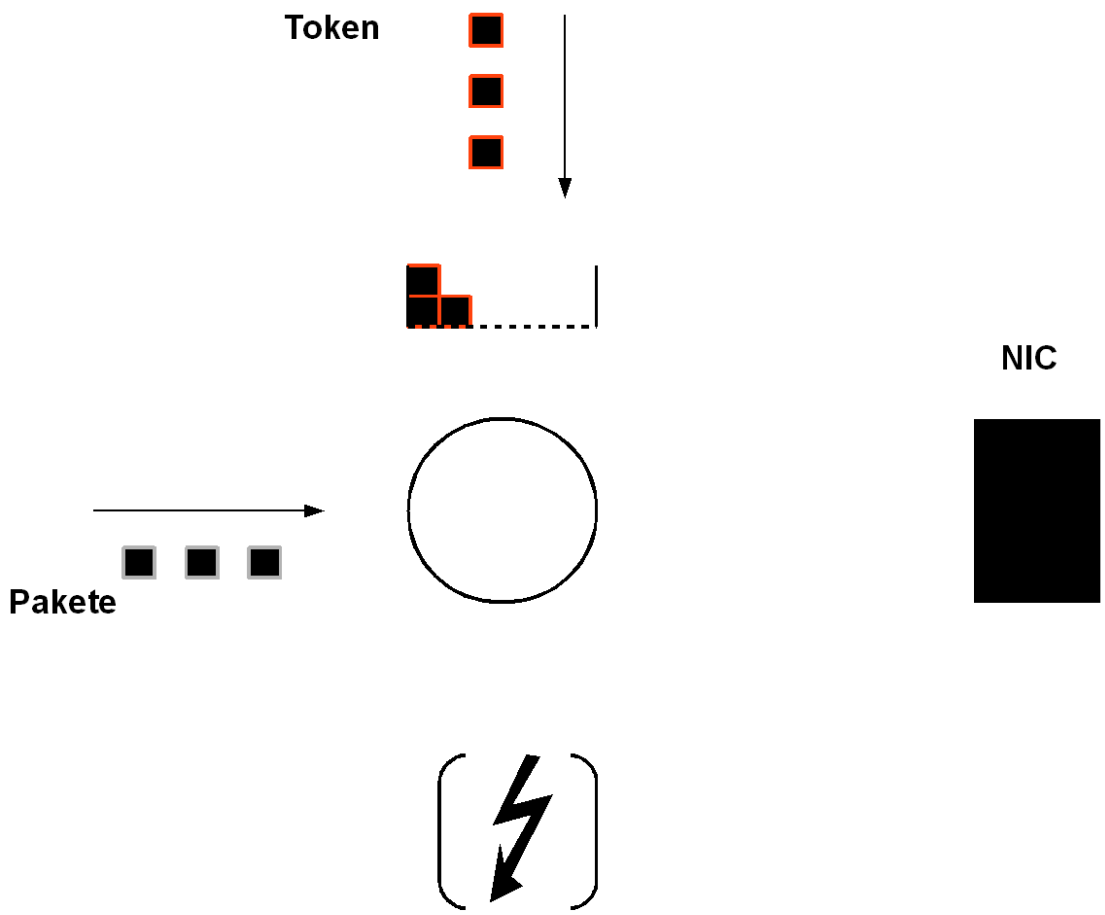


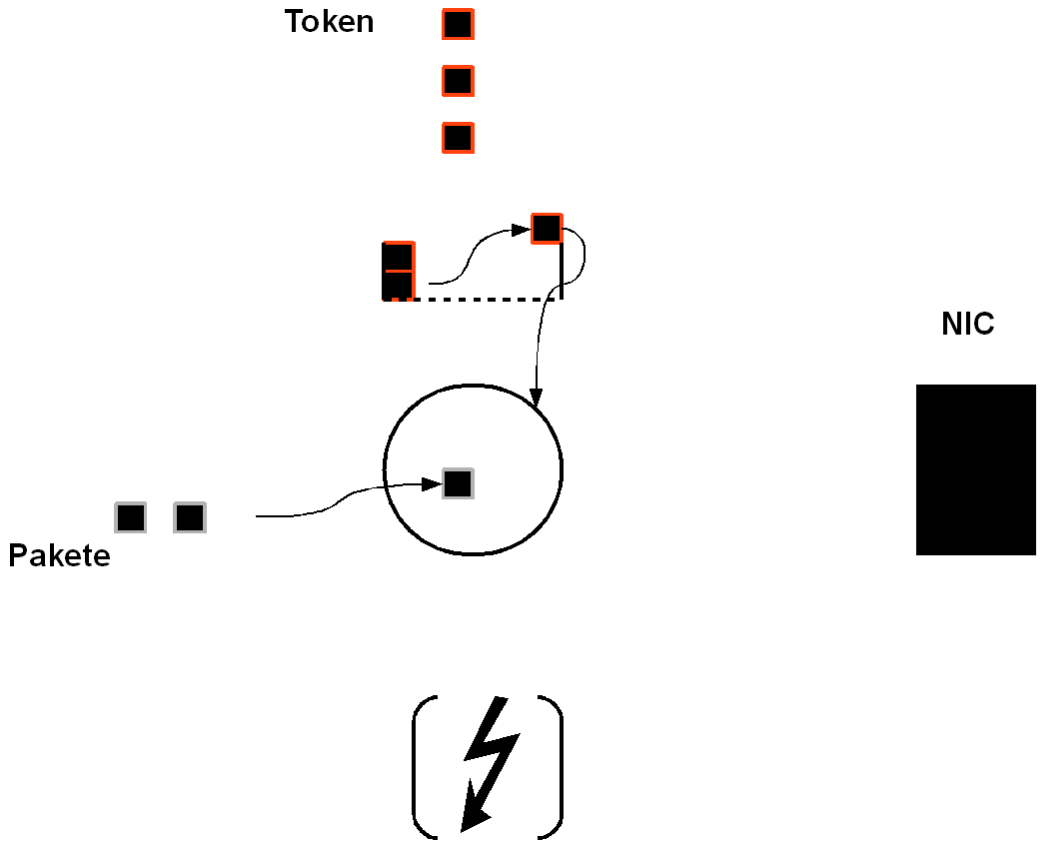

Firma 1
Firma 2
SSH
HTTP
REST

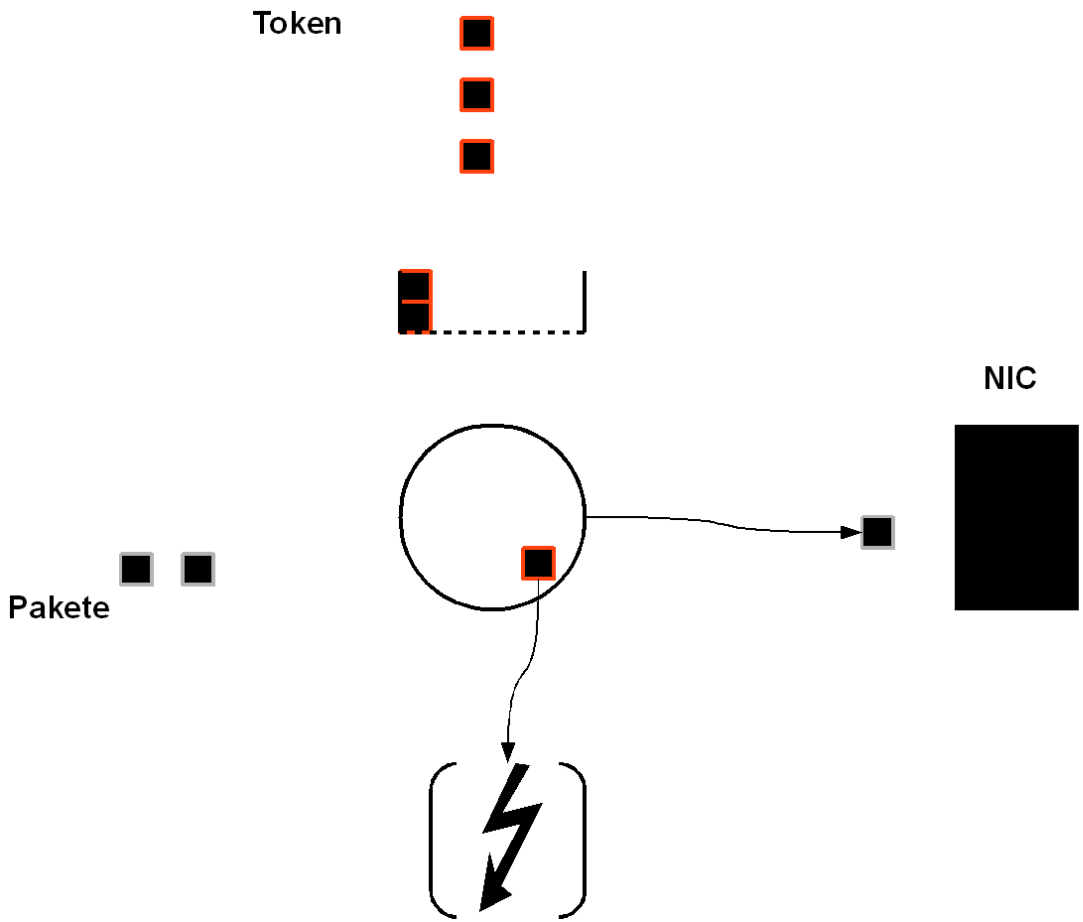

70%
30%
Firma 1
Firma 2
20%
20%
60%
SSH
MAIL
REST


Klassifizierung
Token Buckets
NIC
SSH
MAIL
Rest
Pakete

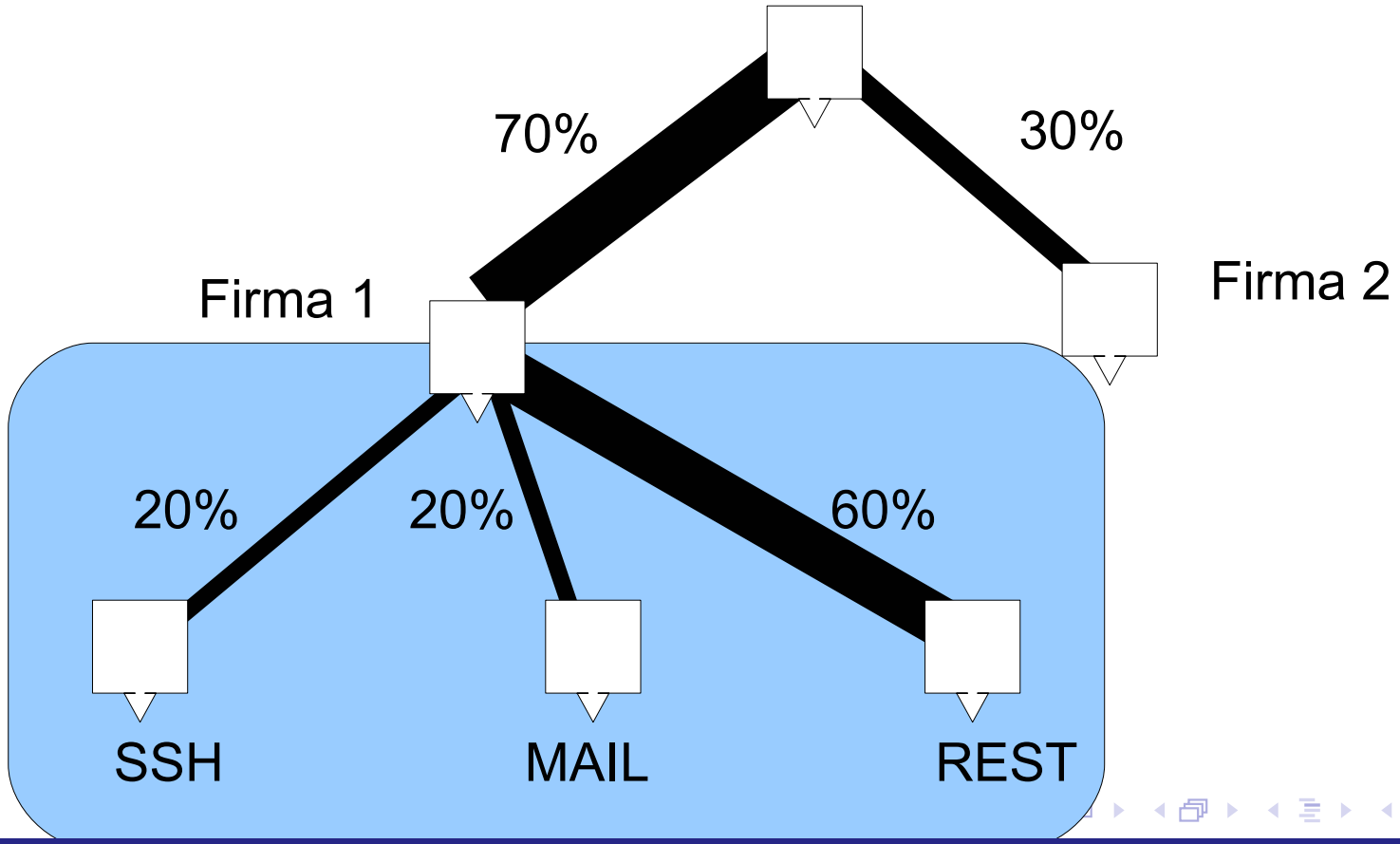
Token-Bucket




Token
NIC
Pakete


Token
NIC
Pakete

Token Buckets & Selektierung



Rückblick

- Keine automatische Umstellung der Leitung
- Verschiebung der Zeitplanung
  - lange Informationssuche
  - Wegfall der automatischen Routing-Änderung
- volle Nutzung der Pufferzeit
- Management via SSH ist auch bei starker Auslastung möglich

Erweiterung

- Priorisierung der Token Buckets
  → stärkere Auslastung der Leitung insgesamt
- Management über Webinterface
  → direktes Management durch Kunden möglich

Vielen Dank für Ihre Aufmerksamkeit!
Haben Sie noch Fragen?