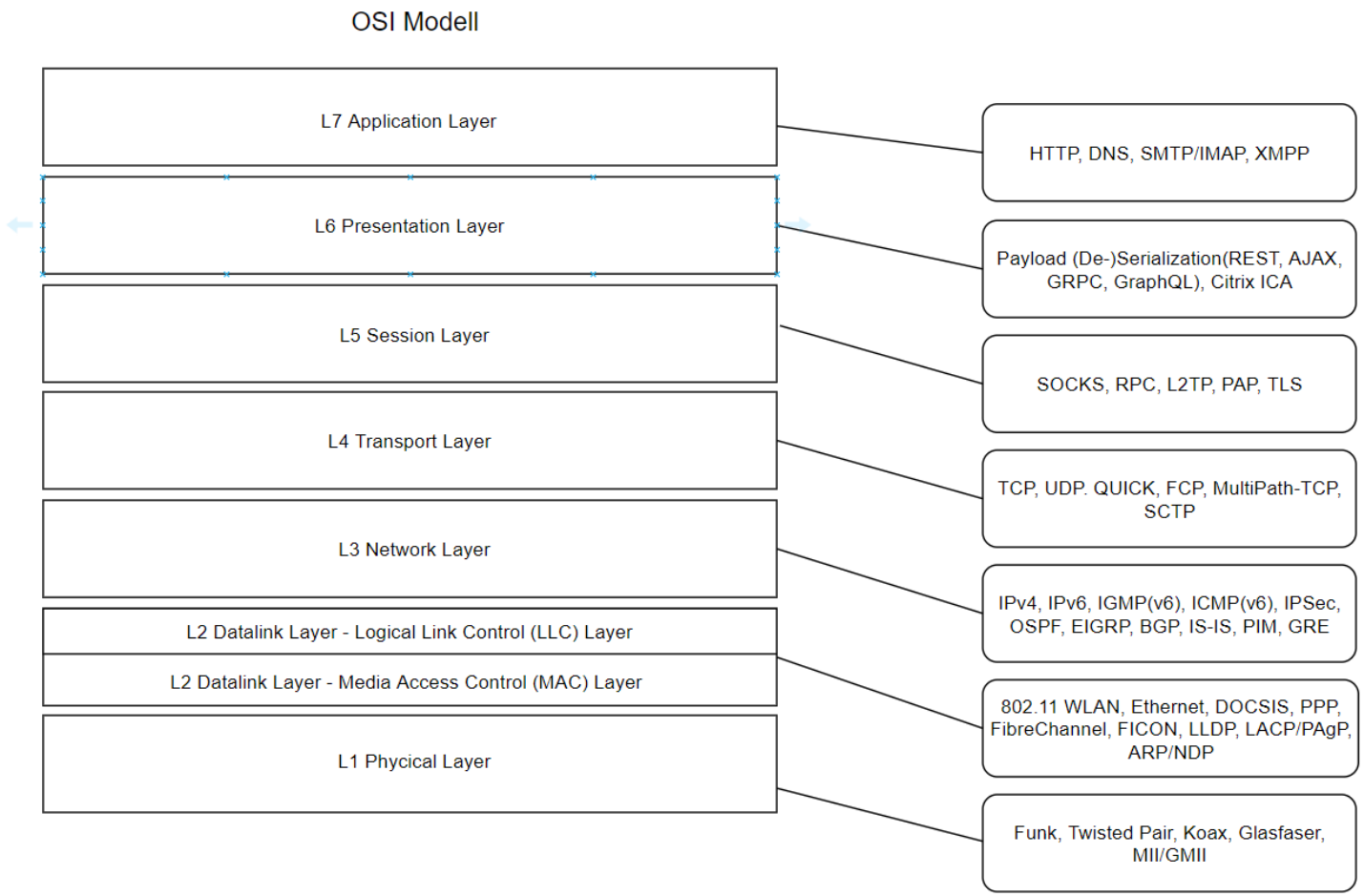
Firewalling unter Linux

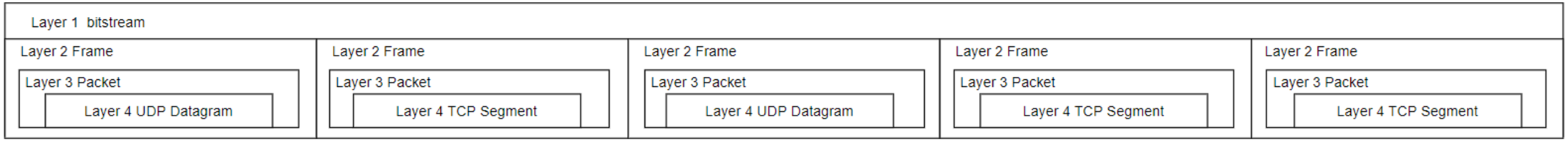
RECAP

OSI Network Layer Model



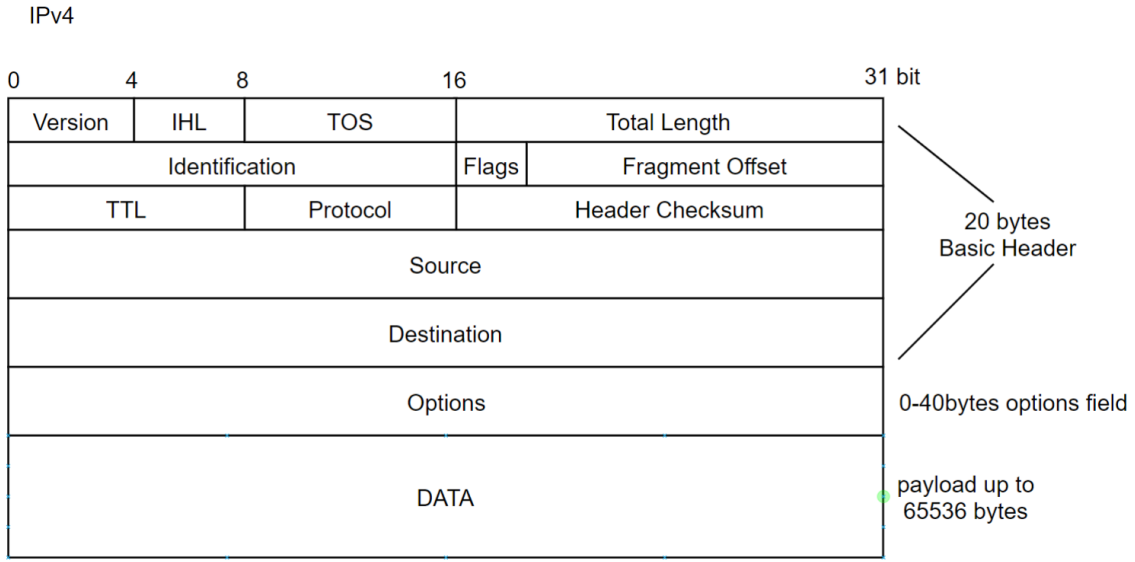
Encapsulation

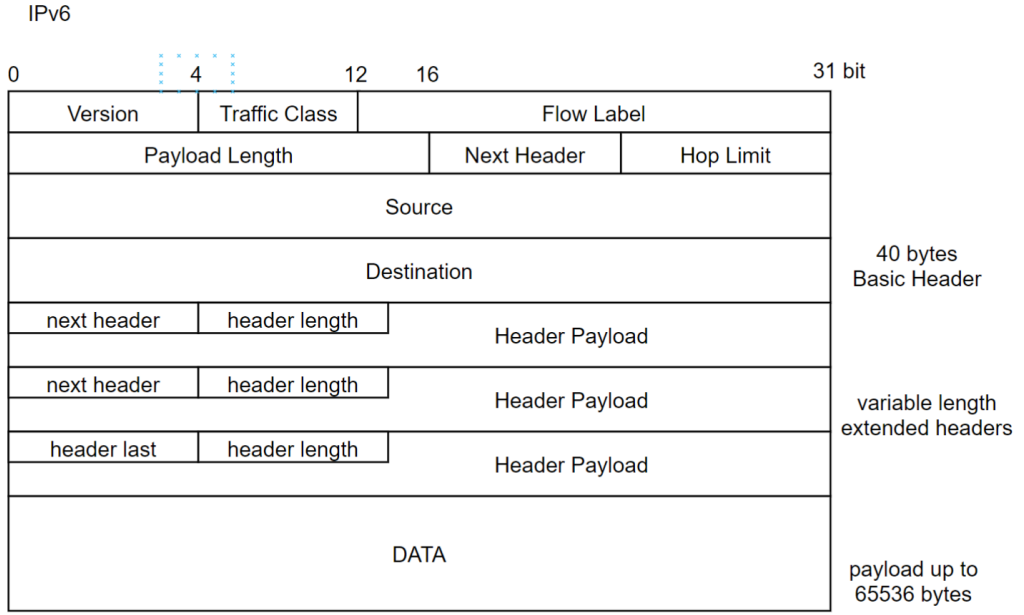
- Layer(Schichten) bieten saubere logische Trennung
- Höhere Layer werden in niedrigeren Layern gekapselt



Internet Layer IPv4/IPv6

- PDUs auf Layer 3 werden Pakete genannt.
- Sie enthalten Header und Payload
- unidirektional





ICMP

- Das Internet Control Message Protocol generiert Fehler falls es bei der Vermittlung von Paketen zu Fehlern kommt und wird zum debuggen von Layer3 Problemen verwendet.
- Bekannteste Vertreter “ping“ und Traceroute
- Verwendet sogenannte Datagramme
- 8byte header und variable Datenlänge
- Wird meist auf Firewalls gefiltert.

IPv4 RFC 1918 - Private Addresses

- Da die Anzahl an IPv4 begrenzt ist hat man recht früh bestimmte Addressräume für private/nicht-öffentliche Nutzung reserviert.
- Diese Netze sollen nicht ins public Internet geroutet werden.
- für IPv4 umfasst das die folgenden Bereiche
  10.0.0.0/8 - 10.0.0.0 - 10.255.255.255
  172.16.0.0/12 - 172.16.0.0 - 172.31.255.255
  192.168.0.0/16 - 192.168.0.0 - 192.168.255.255
- Traffic von und zu diesen Addressen wird via Address Translation ins Internet vermittelt.

IPv4 Dynamische Addresszuweisung Zeroconf/APIPA/Bonjour

- Je nach Konfiguration wird versucht eine link local Addresse festzulegen wenn weder ein DHCP zur Verfügung steht noch eine statische Addresse konfiguriert wurde. Dies dient dem Zweck der Kommunikation im lokalen Netz.

- Für diesen Zweck wurde laut RFC3927 das Subnetz 169.254.x.x/16 für link local Adressen zugewiesen. Dieses Netz ist nicht routable.

IPv6

- Im Gegensatz zu IPv4 sind IPv6 Adressen 128bit lang und werden hexadezimal notiert.
- verkürzte Darstellung möglich.
- Interfaces haben immer zwei IP Adressen. Eine Link Global und eine Link Local Addresse.
- Die Link Local Adresse wird für den NDP Sublayer benötigt, selbst wenn Link Global Adressen konfiguriert sind. (NDP Neighbor Discovery Protocol -> ARP Nachfolger)
- NDP kann verwendet werden um Routing Prefixe zu an einzelne Teilnehmer zu übermitteln.

IPv6 Link Global und Link Local (RFC4193)

- Analog zu RFC3927 wurden in RFC4193 link local Adressen zugewiesen.
- Im Gegensatz zu IPv4 erfüllen IPv6 LL Adressen zusätzliche Zwecke.
- der prefix für link local Adressen ist fe80::/10
- die Default Route wird normalerweise via link local zugewiesen.

L3 - Static Routing

- Jede IP Adress Konfiguration besteht aus Adresse und Netzmaske
  Angegeben wird das im Format Adresse/<Netmask|CIDR>
- Bsp: 192.168.1.1/24 ist eine IPv4 Adresse und hat 32bit.
  Der Netprefix ist 24bit lang. Alle Adressen im Hostanteil(blau) sind lokal erreichbar und brauchen keinen Router.
- Adressen die eine Änderung im grünen Bereich(Netz) erfordern müssen geroutet werden.

11111111.11111111.11111111.00000000
11000000.10101000.00000001.00000001

L4 - Transport Protocol

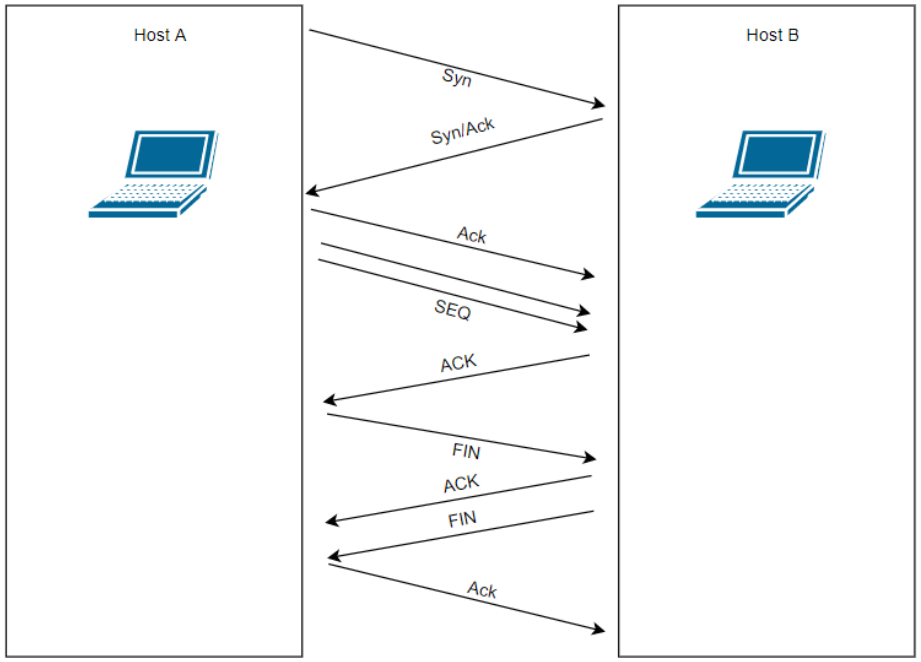
- Auf L4 erfolgt der eigentliche Datentransfer.
- L4 ist normalerweise Portbasiert.
- Ein Service auf einem Host lauscht auf einem Port auf eingehenden Traffic. Der Client öffnet einen Port in seiner ephemeral Range kommuniziert auf diesem Port mit dem Server und dessen Port um entsprechende Daten auszutauschen.
- Ggfs. werden im Laufe der Kommunikation Seitenkanäle auf weiteren Ports geöffnet.

TCP - Segments

- Stateful Connection mit Fehlerkorrektur und Flußkontrolle
- Viel Overhead

TCP Segment

Source Port								Destination Port
Sequence Number								
Ackknowledgement Number								
offset		U	A	P	R	S	F	Window Size
Checksum								Urgent Pointer
Options								
Payload								



UDP - DATAGRAMS

- UDP ist unzuverlässig by Design (Keine Handshakes, kein retransmit - Fire and Forget)
- UDP kennt keine Flußkontrolle(muss auf Applikationsebene implementiert werden)
- geringerer overhead, geringere Latenz.
- präferiert für Protokolle die keine 100% Datenkonsistenz benötigen wie voice oder gaming-traffic.

Source Port	Destination Port
length	checksum
Data	

FIREWALLING

host access control files

- filtert Traffic auf Basis von Prozessen und Source-Addressen/DNS-Lookups
- Bestandteil des TCPWrappers, ersetzt keinen Paketfilter

```
FILES

/etc/hosts.allow, (daemon,client) pairs that are granted access.
/etc/hosts.deny, (daemon,client) pairs that are denied access.


/etc/hosts.allow:

in.httpd: 172.16.21.0/24
sshd: ALL

/etc/hosts.deny:
ALL: ALL
```

Hostbased and Perimeter Firewalling

- Hostbased Firewalls:
  - frontends:
    - firewalld/ufw
    - nftables
    - iptables(veraltet)
  - Verwaltungsaufwand und Komplexität steigt mit Anzahl der Hosts signifikant.

- Perimeter Firewalls:
  - frontends:
    - nftables
    - iptables (veraltet)
  - weniger management-overhead durch Zentralisierung

LAYER 3&4 - STRATEGIEN

Policy White/Hybridlist Approach

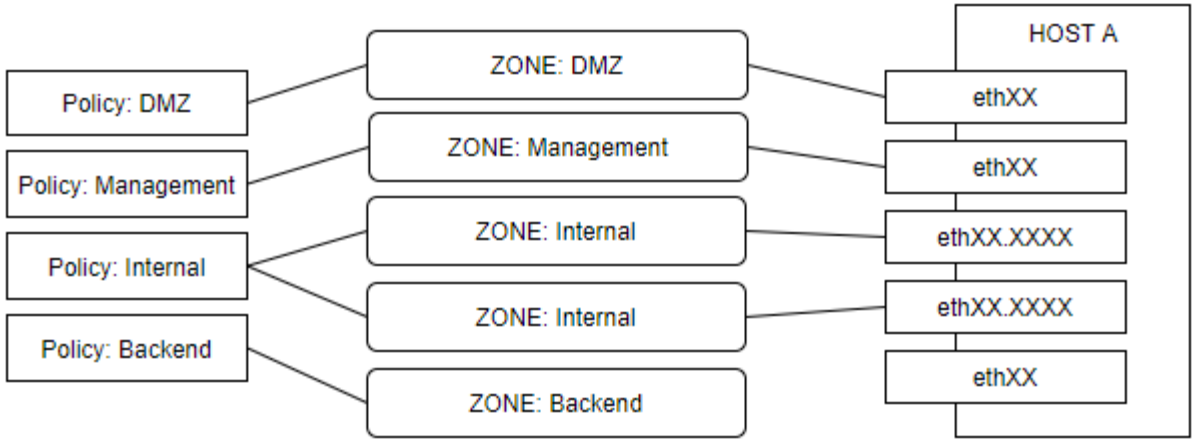
- Blacklists sind nicht handhabbar, da zu im Allgemeinen lang.
- jeder Traffic der nicht erlaubt ist wird verworfen
- Traffic der erlaubt ist kann nach weiteren Kriterien geprüft werden.
- Discriminator-basiert (Protocol, Adressen, Ports, Connection State)

Always DROP, Never REJECT

- Warum DROP:
  - reduziert Last, da keine Antworten verschickt werden sondern Pakete still verworfen werden
  - verhindert Policy Cartography durch Port-Scanning(Perimeter Firewall only)
  - verhindert Missbrauch durch Amplification Attacks auf reject Basis durch Address Spoofing.

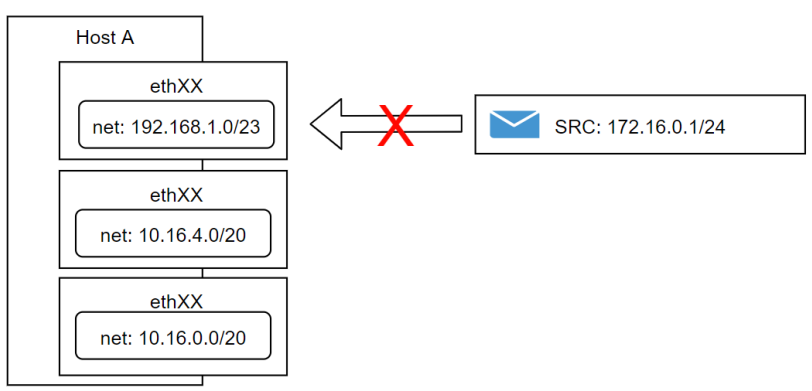
Zone-/Chainbased Approach

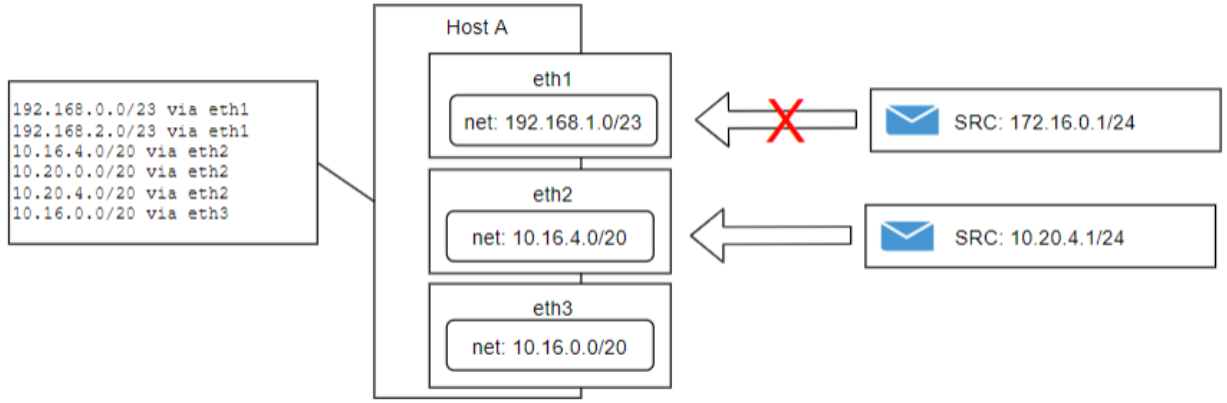
- Warum Zonen-/Chainbasiert:
  - reduziert Last
  - reduziert Policy-Komplexität



IP-Address-Antispoofing

- Interface/Routing-basiert(rp_filter)
- prüft Gültigkeit einer IP-Addresse im Bezug auf den Reverse-Path





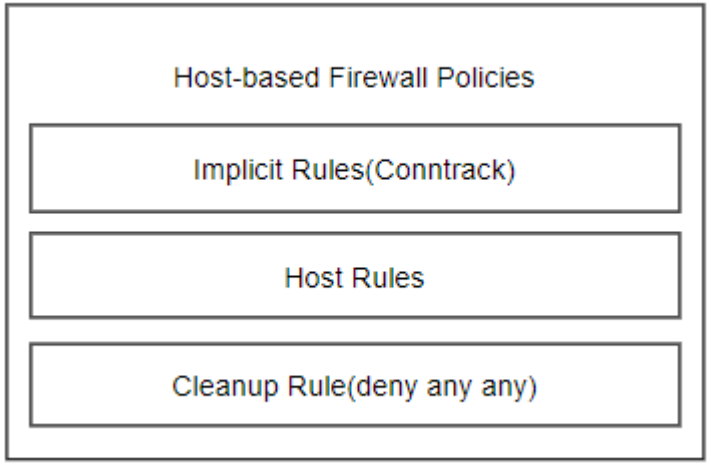
Stateful Firewalls
Connection Tracking (nf.mod_conntrack)

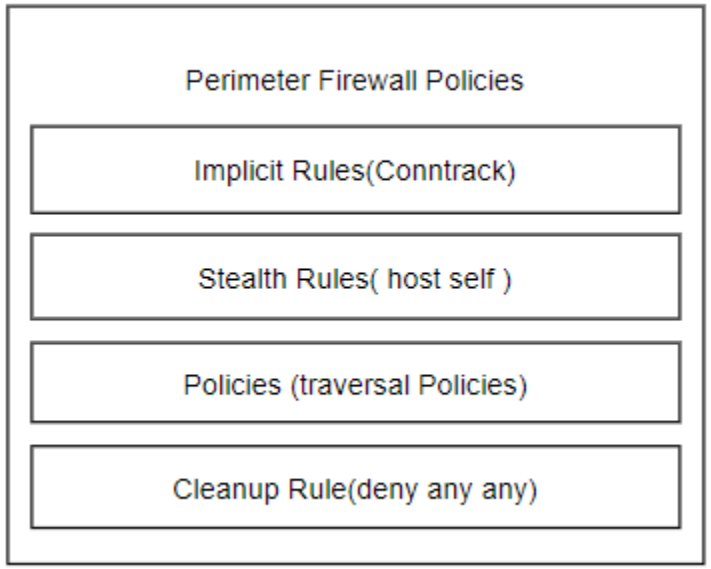
- Verbindungen müssen je nach Protokoll(TCP/QUIC) per Handshake in der Connectiontable registriert werden.
- Verbindungen ohne handshake sind „Out-of-State“ und werden verworfen.
- Verbindungen mit Side-Channels erfordern zusätzliche Trackingkonfiguration(ftp, X11, NFS etc.)
- Erleichtert das Connectionmonitoring
- Garantiert Protokoll Compliance (wenn entsprechend konfiguriert)
  - Schutz gegen ungültige Flagkombinationen(Christmas Packets etc.)

Policy Best Practices

- Implicit Rules implementieren Stateful Checks
- Host- & Stealthrules implementieren:
  - Regeln die für das Device selbst gelten
    (Portfreigaben für Management Verbindungen etc.)
- Traversal Policies implementieren Regeln für weitergeleiteten Traffic
- Toptalker zuerst
- wenn möglich fasse zusammen. (Network Summarization, Service Grouping)
- Im Zweifel früher droppen.

Policy Best Practices
Structure





Policy Best Practices
Performance Killer - Obvious Wisdoms

- Toptalker zuerst
- viele kleine Pakete erzeugen mehr Last als wenige Große
- halte Policies kurz (summarization), First Match Counts
- Verwende Logging mit Bedacht.
- Vermeide Fragmentierung und Packet-Reassembly(MTU, Window-Size)
- Optimiere RX/TX Buffer wenn nötig. (Buffer Underflow/Overflow)
- Benutze Multiqueuing und CPU-Affinity wenn nötig.
- Achte darauf das deine Connection Tracking Table ausreichend dimensioniert ist und entsprechend RAM und CPU vorhanden ist
- Benutze Rate-Limiter zur Flood Protection

HIGH AVAILABILITY

statetable synchronization conntrack-tools conntrackd

- userspace daemon für die Synchronisierung von Connection-Tracking Information.
- Unterstützt Active/Backup und Active/Active Setups (active/active recommended)
- Kümmert sich nicht um Policy Synchronization
- Kümmert sich nicht um L3 HA(IP) Failover
- Failover durch Tracking auf Link-Basis

```
FILES

/etc/conntrackd/conntrackd.conf
```

conntrack-tools conntrackd.conf

- Aufgeteilt in:
  - Synchronization Mode(FTFW, ALARM, NOTRACK - FTFW empfohlen)
  - Synchronization Transport Mechanism (TCP or UDP Unicast, Multicast)
  - General Options:
    - Buffersizes(default size : /proc/sys/net/core/rmem_default)
    - conntrack Hashlimits
    - logfiles & lockfiles(pidfile)
    - optional polling mode
- Filtermode: Filter für Protokolle oder IP Addressen

conntrack-tools useful commands

- Verbindungen anzeigen:

  ```
  conntrackd -L
  ```

- spezifische Verbindungen anzeigen

  ```
  conntrackd -L -p <tcp|udp> --<sport|dport|src|dst> xxx
  ```

- spezifische Verbindung aus der Connectiontable entfernen

  ```
  conntrackd -D -p <tcp|udp> --<sport|dport|src|dst> xxx
  eg.
  conntrackd -D -p tcp --src 192.168.10.2 --dst 192.168.11.2 --dport 80
  ```

- conntrackd statistiken anzeigen:

  ```
  conntrackd -s
  ```

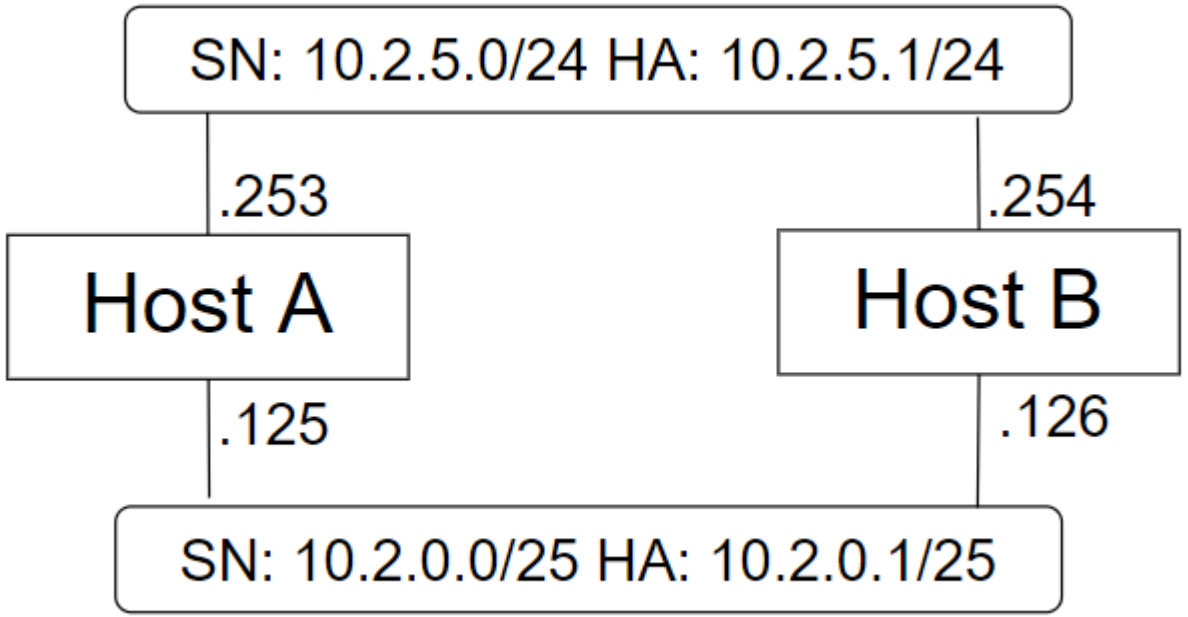
keepalived - VRRP L3 HA

- keepalived implementiert VRRP(Virtual Router Redundancy Protocol) auf Linux Maschinen
- Stellt Layer 3 Failover bereit
- Failover Implementierung durch Scriptmonitoring sowie Linktracking
- sendet Gratuitous ARP für MAC Renewal
- benötigt Adressen für Hosts und HA-IP

```
FILES

/etc/keepalived/keepalived.conf
```

keepalived - VRRP L3 HA

```
vrrp_instance nic_eth0 {
    interface eth0
    virtual_router_id 50
    priority 100
    advert_int 1
    track_interface {
        eth0 weight 1
    }
    virtual_ipaddress {
        10.2.5.1/24
    }

}

vrrp_instance nic_eth1 {
      state MASTER
      interface eth1
      virtual_router_id 50
      advert_int 1
      authentication {
         auth_type PASS
         auth_pass <pwd>
      }
      virtual_ipaddress {
         10.2.0.1/25
      }
      track_file {
         track_file weight 1
      }
}
```



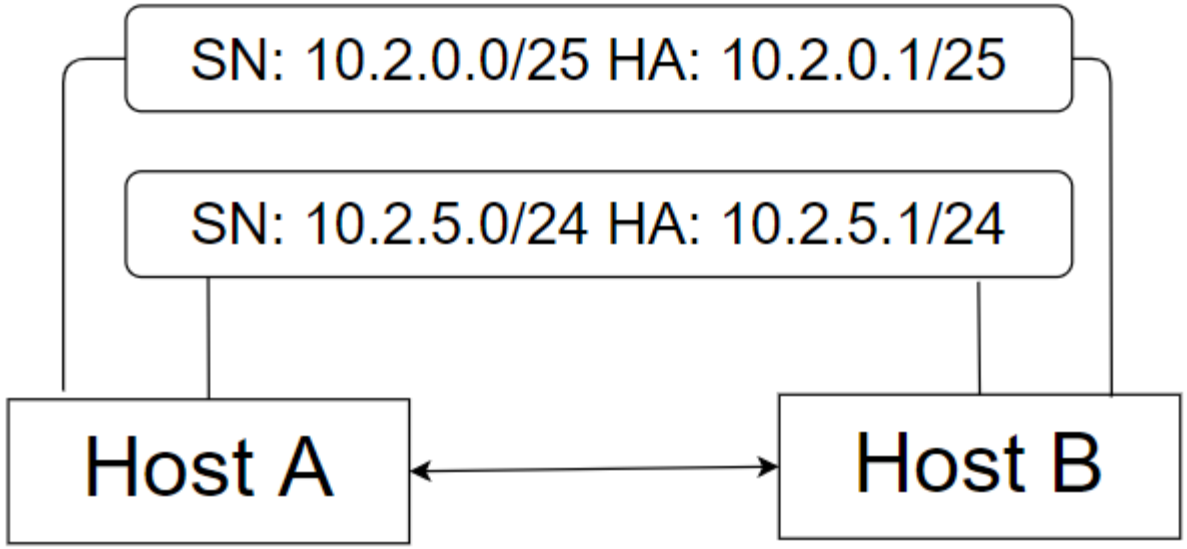

```
vrrp_instance nic_eth0 {
    interface eth0
    virtual_router_id 50
    priority 95
    advert_int 1
    track_interface {
        eth0 weight 1
    }
    virtual_ipaddress {
        10.2.5.1/24
    }

}

vrrp_instance nic_eth1 {
      state SLAVE
      interface eth1
      virtual_router_id 50
      advert_int 1
      authentication {
         auth_type PASS
         auth_pass <pwd>
      }
      virtual_ipaddress {
         10.2.0.1/25
      }
      track_file {
         track_file weight 1
      }
}
```

Pacemaker/Corosync - L3 HA

- PACEMAKER/Corosync implementiert L3 auf Linux Maschinen
- Stellt Layer 3 Failover bereit
- Failover Implementierung durch Corosync Prozess- und Resourcetracking
- arbeitet mit Floating IP only
- sendet Gratuitous ARP für MAC Renewal

Pacemaker/Corosync - L3 HA

```
passwd hacluster
pcs cluster auth node1 node2
pcs cluster setup --name mycluster node1 node2
pcs resource create cluster_ip ocf:heartbeat:IPaddr2 ip=10.2.0.1 cidr_netmask=25 op monitor interval=1s nic=eth1
pcs resource create cluster_ip ocf:heartbeat:IPaddr2 ip=10.2.5.1 cidr_netmask=24 op monitor interval=1s nic=eth0
pcs cluster start --all
pcs cluster enable --all
```



POLICY MANAGEMENT

hostbased Policy Control mit firewalld

- Zielgruppe Server und Desktops
- möglich (aber nicht empfohlen) für Perimeter-Firewalls
- zonenbasiertes konzept
- Integration mit dbus & systemd

hostbased Policy Control mit firewalld

- list zones:
  ```
  firewall-cmd --get-zones
  ```
- list settings of a zone
  ```
  firewall-cmd --zone=external --list-all
  ```
- list services allowed in a zone
  ```
  firewall-cmd --zone=external --list-services
  ```
- allow a service for a zone
  ```
  firewall-cmd --permanent --zone=external --add-port=443/tcp
  ```
- Allow an IP for a Zone
  ```
  firewall-cmd --permanent --zone=external --add-source=10.24.96.0/20
  firewall-cmd --permanent --zone=external --add-source=10.24.96.0/20
  ```

hostbased Policy Control mit firewalld „rich“ rules

- firewalld standard Regeln sind oversimplified.

  e.g.

  ```
  permit tcp 10.0.0.0/8 192.168.10.1/24:443
  ```

  der schlechte Weg - simple zoning:

  ```
  firewall-cmd --new-zone=https_zone --permanent
  firewall-cmd --zone=https_zone --add-source=10.0.0.0/8 --permanent
  firewall-cmd --zone=https_zone --add-destination=192.168.10.1/24 --permanent
  firewall-cmd --zone=https_zone --add-port=443/tcp --permanent
  ```

hostbased Policy Control mit firewalld „rich“ rules

- der bessere Weg - rich rules

```
rule [family="rule family"]
[ source [NOT] [address="address"] [mac="mac-address"] [ipset="ipset"] ]
[ destination [NOT] address="address" ]
[ element ]
[ log [prefix="prefix text"] [level="log level"] [limit value="rate/duration"] ]
[ audit ]
[ action ]
```

e.g.

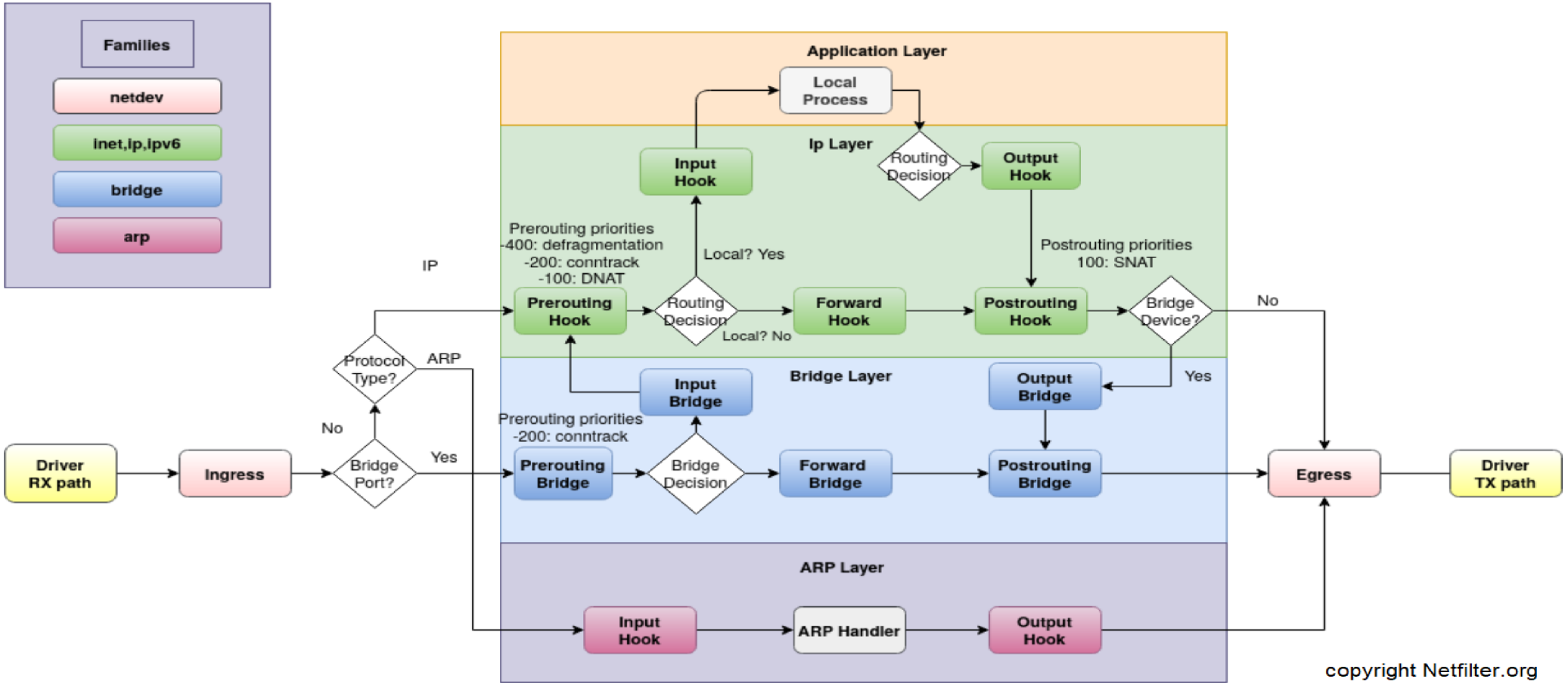
```
firewall-cmd --zone=external --add-rich-rule='rule family=ipv4 source address=10.0.0.0/8
destination address=192.168.10.1/24 service name=443/tcp accept' --permanent
```

Policy Control mit nftables

- <ip|ip6|eb|arp>tables Nachfolger
- benutzt Chaining, Sets und Tables an Stelle von Zones
- empfohlen für Perimeter-Firewalls
- dockt an kernel-netlink hooks an

```
table ip filter {
    chain input {
       type filter hook input priority 0; policy drop;
       iifname "lo" ip saddr != 127.0.0.0/8 drop
       iifname "lo" accept
       iifname eth0 ip saddr 10.0.0.0/8 ip daddr 192.168.10.1/24 tcp dport 443 ct state new,established accept
    }

    chain output {
       type filter hook output priority 0; policy drop;
       oifname "lo" ip daddr != 127.0.0.0/8 drop
       oifname "lo" accept
       oifname eth0 ip daddr 10.0.0.0/8 ip saddr 192.168.10.1/24 ct state established accept
    }

    chain forward {
       type filter hook forward priority 0; policy accept;
    }
}
```

Policy Control mit nftables



Policy Control mit nftables

Chain type	Hooks						
	ingress	prerouting	forward	input	output	postrouting	egress
inet family							
filter	0.9.7 / 5.10	Yes	Yes	Yes	Yes	Yes	No
nat	No	Yes	No	Yes	Yes	Yes	No
route	No	No	No	No	Yes	No	No
ip6 family							
filter	No	Yes	Yes	Yes	Yes	Yes	No
nat	No	Yes	No	Yes	Yes	Yes	No
route	No	No	No	No	Yes	No	No
ip family							
filter	No	Yes	Yes	Yes	Yes	Yes	No
nat	No	Yes	No	Yes	Yes	Yes	No
route	No	No	No	No	Yes	No	No

Chain type	Hooks						
	ingress	prerouting	forward	input	output	postrouting	egress
arp family							
filter	No	No	No	Yes	Yes	No	No
nat	No	No	No	No	No	No	No
route	No	No	No	No	No	No	No
bridge family							
filter	No	Yes	Yes	Yes	Yes	Yes	No
nat	No	No	No	No	No	No	No
route	No	No	No	No	No	No	No
netdev family							
filter	0.6 / 4.2	No	No	No	No	No	- / 5.7
nat	No	No	No	No	No	No	No
route	No	No	No	No	No	No	No

Policy Control mit nftables
useful commands

- Tables/Chains anzeigen
  ```
  nft list <tables|chains>
  ```
- Tables/Chains löschen
  ```
  nft delete table <protofam> <table>
  nft delete chain <protofam> <table> <chain>
  ```
- neue Table anlegen
  ```
  nft add table <protofam> <tablename>
  ```
- neue Chain anlegen
  ```
  nft add chain <protofam> <table> <chainname> { type filter hook <hook> priority
  0\; }
  ```
- Rule anlegen
  ```
  nft add rule inet <table> <chain> ip saddr 192.168.0.0/24 ip daddr 10.0.0.0/8 tcp
  dport {22, 80, 443} ct state new,established counter accept
  ```

Policy Control mit nftables useful commands

- Rules mit handle anzeigen

  `nft -n -a list ruleset`

```
root@debian:~# nft -n -a list ruleset
table ip fw { # handle 1
        chain testchain { # handle 1
                iifname "ens33" ip saddr 192.168.10.0/24 counter packets 0 bytes 0 accept # handle 4
                ip saddr 192.168.10.0/24 ip daddr 10.0.0.0 tcp dport { 22, 80, 443 } ct state 0x2,0x
8 counter packets 0 bytes 0 accept # handle 12
        }
}
```

- Rule Replace

  `nft replace rule <protofam> <table> <chain> handle <num>  <rule-elems>`

- Rule insert

  `nft insert rule <protofam> <table> <chain> position <num> <rul-elems>`

Policy Control mit nftables useful commands

- Sets werden zur Gruppierung genutzt
- Set anlegen

```
nft add set <protofam> <table> <setname> { type ipv4_addr\; comment \"group of hosts|ranges \" \;
```

- Adressen zu sets hinzufügen

```
nft add element <protofam> <table> <setname> { 192.168.3.4 }
nft add element <protofam> <table> <setname> { 192.168.4.0/26, 192.168.5.1 }
```

- Sets in Rules einbinden

```
nft add rule <protofam> <table> <chain> ip saddr @<setname> ip daddr 10.0.0.0/8 tcp dport {22, 80, 443} ct state new,established counter accept
```

Danke!