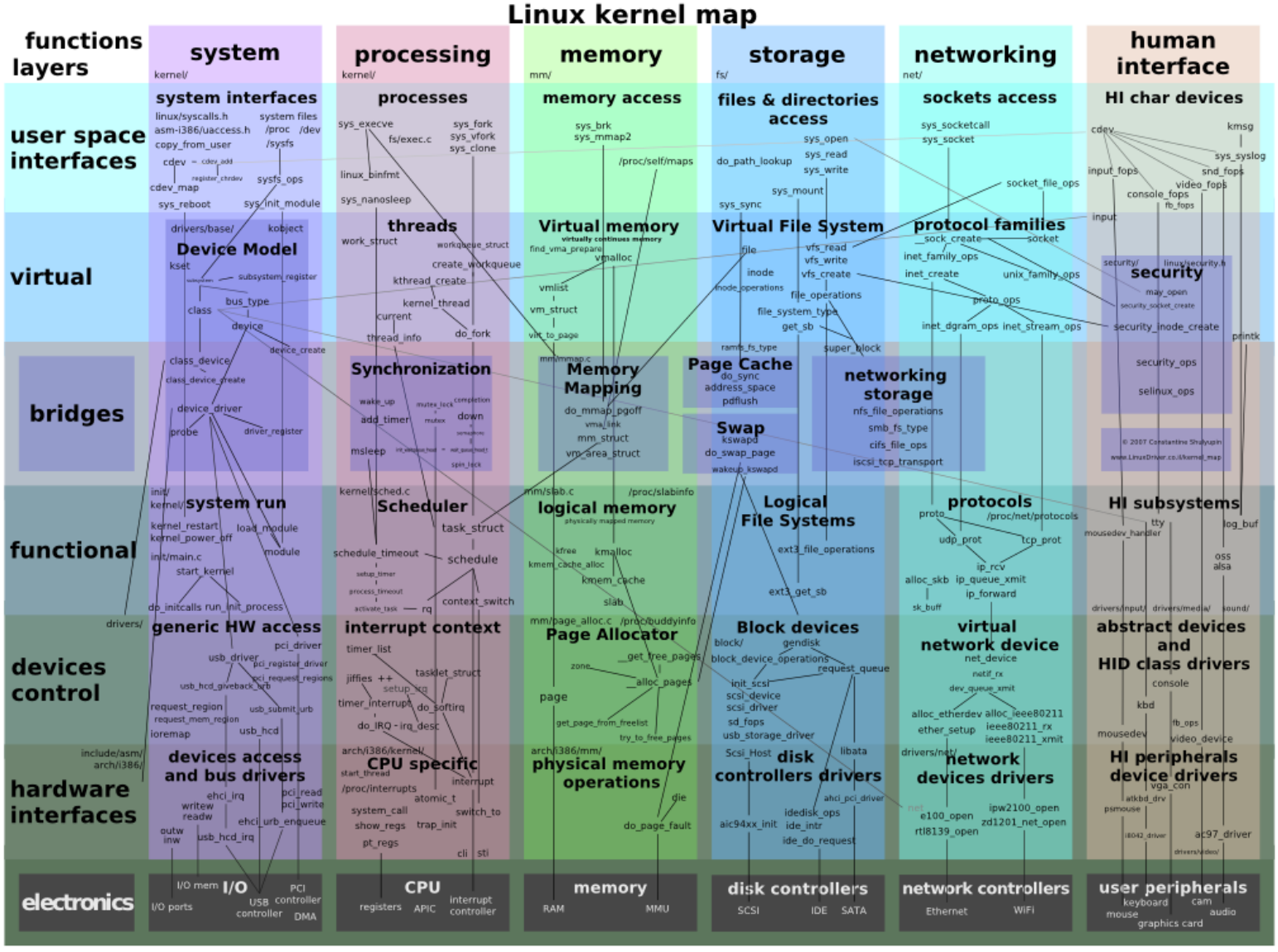
# Linux Kernel
# (deel2)




ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

# Linux Kernel

**Opbouw**

Process Scheduler
Memory Manager
File System
Device Driver
***Network Services***
I/O Scheduler
IPC → Inter Process Communications

# Linux Kernel

## Network Services

Linux heeft een zeer uitgebreide set
van network services…

→ Linux = volwaardige Router + Firewall !!

- ✔ installeer meerdere netwerk kaarten in een pc
- ✔ installeer linux
- ✔ configureer

*… en je hebt een router !*

# Linux Kernel

**Network Services**

… en een uitgebreide set aan
Bash netwerking commando's.

**arp, ip, (ifconfig), ping,**
**iptables, route, ss, (netstat)**
**traceroute, nslookup,**
**Hostname, dig,**
**ifup ifdown,**
**ethtool, iwconfig**
**tcpdump, nmap, ...**

# Linux Kernel

## Kernel Network Stack

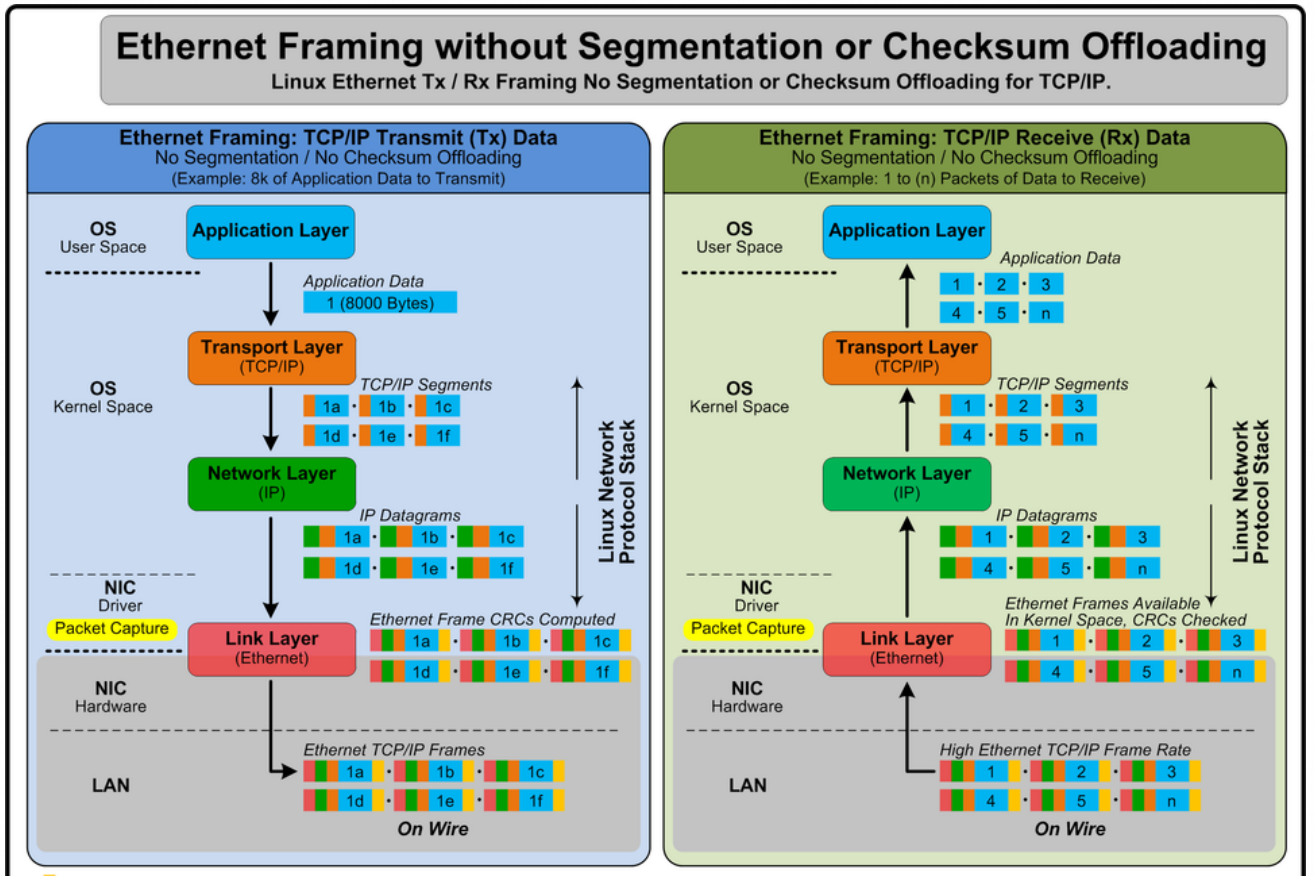
- Stack = “stapel” → opeenstapeling
  van de verschillende protocol lagen

- Network-Stack → verzameling van software
  implementaties van netwerk protocollen
  → IP, TCP, UDP, SNMP, RTP ...

- Bij linux zit deze code in de Kernel.

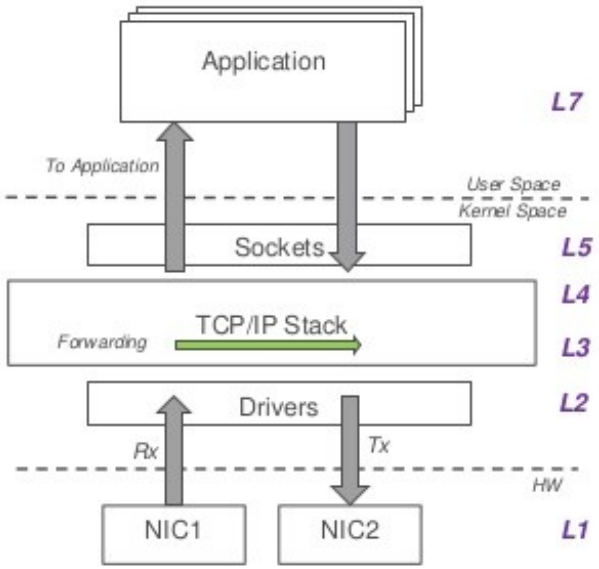
# Linux Kernel

## Kernel Network Stack



# Linux Kernel

## Linux kernel data path

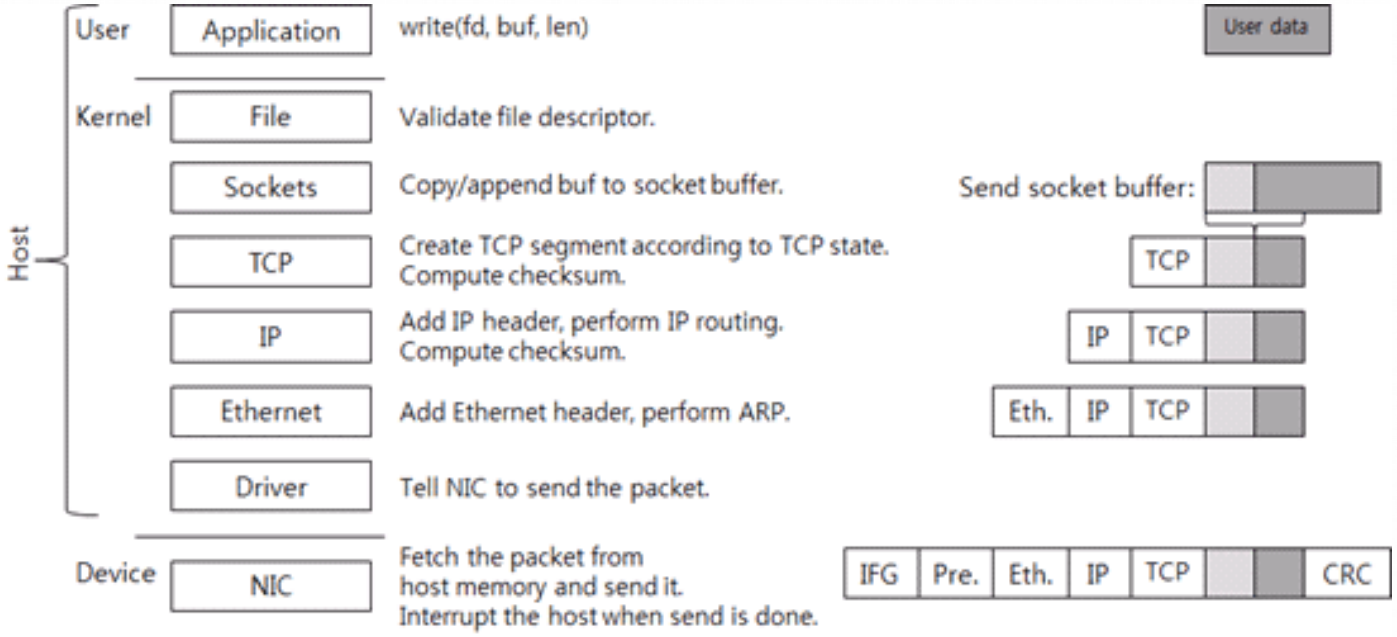



- Design goals or why stack is in the kernel?
  - Linux is designed as an Internet Host (RFC1122) or an "End-System" OS
  - Need to service multiple applications
  - Separate user applications from sensitive kernel code
  - Make application as simple as possible
  - Receive direct access to HW drivers
- Cost
  - Not optimized for Forwarding
  - Every change requires new kernel version
  - Code is too generic
  - Networking stack today is a huge part of the kernel

- Reference: Kernel Data Path

# Linux Kernel

## Kernel Network Stack



# Linux Kernel

## Network Services: Configuratie

Hoe ?

→ bash commando’s
→ configuratie files

# Linux Kernel

## Network Services: Configuratie

- iproute commando's:
  - ifconfig, netstat, arp, route, btctl
  - worden nog steeds veel gebruikt
- iproute2 commando's:
  - ip, ss, tc, arpd, etc
  - nieuwe manier
- NetworkManager → via GUI
  - "netwerking zonder zorgen"

# Linux Kernel

## Network Services: Configuratie

Configuratie files in:

*/etc/network*

Netwerk adapter (NIC) configuratie:

*/etc/network/interfaces = tekstfile*

# Linux Kernel

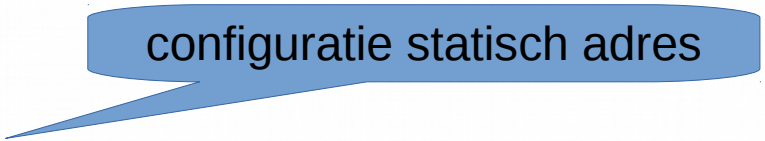
## Network Services: Configuratie

*/etc/network/interfaces*

```
# interfaces(5) file used by ifup(8) and ifdown(8)

auto lo
iface lo inet loopback

iface eth0 inet static
address 192.168.1.5
netmask 255.255.255.0
gateway 192.168.1.254
```



# Linux Kernel

## Network Services: Configuratie

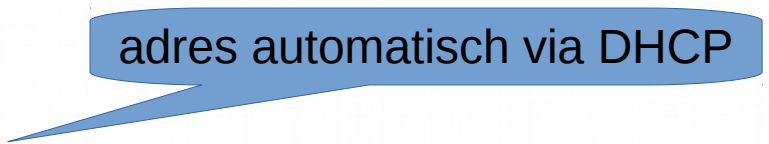
*/etc/network/interfaces*

```
# interfaces(5) file used by ifup(8) and ifdown(8)

auto lo

iface lo inet loopback

auto eth0
iface eth0 inet dhcp
```



# Linux Kernel

## Network Services: Configuratie

*/etc/network/interfaces*

***MAAR…***

*interfaces worden automatisch*
*beheerd door “NetworkManager”*
*tenzij anders opgegeven*
*in de NetworkManager.conf file !!!*

# Linux Kernel

## Network Services: Configuratie

*/etc/NetworkManager/NetworkManager.conf*

*[main]*
*plugins=ifupdown, keyfile*

*[ifupdown]*
***managed=false***

*[keyfile]*
***unmanaged-devices=mac:xx:xx:xx:xx:xx:xx***

*of met de interfacenaam i.p.v het mac-adres:*
***unmanaged-devices=interface-name:em4***

# Linux Kernel

## Network Services: Configuratie

NetworkManager

”netwerking zonder zorgen”

*Wat niet expliciet manueel geconfigureerd is*
*probeert NetworkManager automatisch te configuren*

→ oorsprong & hoofddoel:

*vlekkeloze overgang tussen wifi*
*en bekabeld netwerk bij laptops*

# Linux Kernel

**ip commando**

ip link

add, delete, show
set, help

# Linux Kernel

**ip commando**

ip link set [interface]

up
down

ip link show [interface]

# Linux Kernel

## ip commando

*$> ip link show*

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueu mode DEFAULT group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: enp2s1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
                                                    DEFAULT group default qlen 1000
    link/ether 08:60:6e:6c:a3:2e brd ff:ff:ff:ff:ff:ff
```

# Linux Kernel

**ip commando**

ip link show

1. *lo → loopback interface*
2. *enp2s1 → ethernet interface*
3. *wlp1s2 → wifi interface*

# Linux Kernel

**ip commando**

(nieuwe) netwerk interface namen

<u>eerste 2 letters:</u>

Type interface

en = ethernet
sl = serial line ip (slip)
wl = wlan
ww = wwan

# Linux Kernel

**ip commando**

(nieuwe) netwerk interface namen

volgende letters & cijfers:

p<bus>s<slot>

Dit is op welke bus en welk slot de netwerk kaart fysisch zit op de PCI bus v/h moederbord.

# Linux Kernel

**ip commando**

(nieuwe) netwerk interface namen

<u>Voorbeeld:</u>

enp2s1 → ethernetkaart op bus 2 slot 1

wlp1s0 → wifikaart op bus 1 slot 0

# Linux Kernel

## ip commando

Oude netwerk interface namen

Voorbeeld:

eth0, eth1, wlan1, wlan2

- worden nog steeds gebruikt
- gemakkelijker te lezen dan nieuwe maar...
- minder duidelijk over welke hardware het gaat en waar deze zich bevindt.

# Linux Kernel

**ip commando**

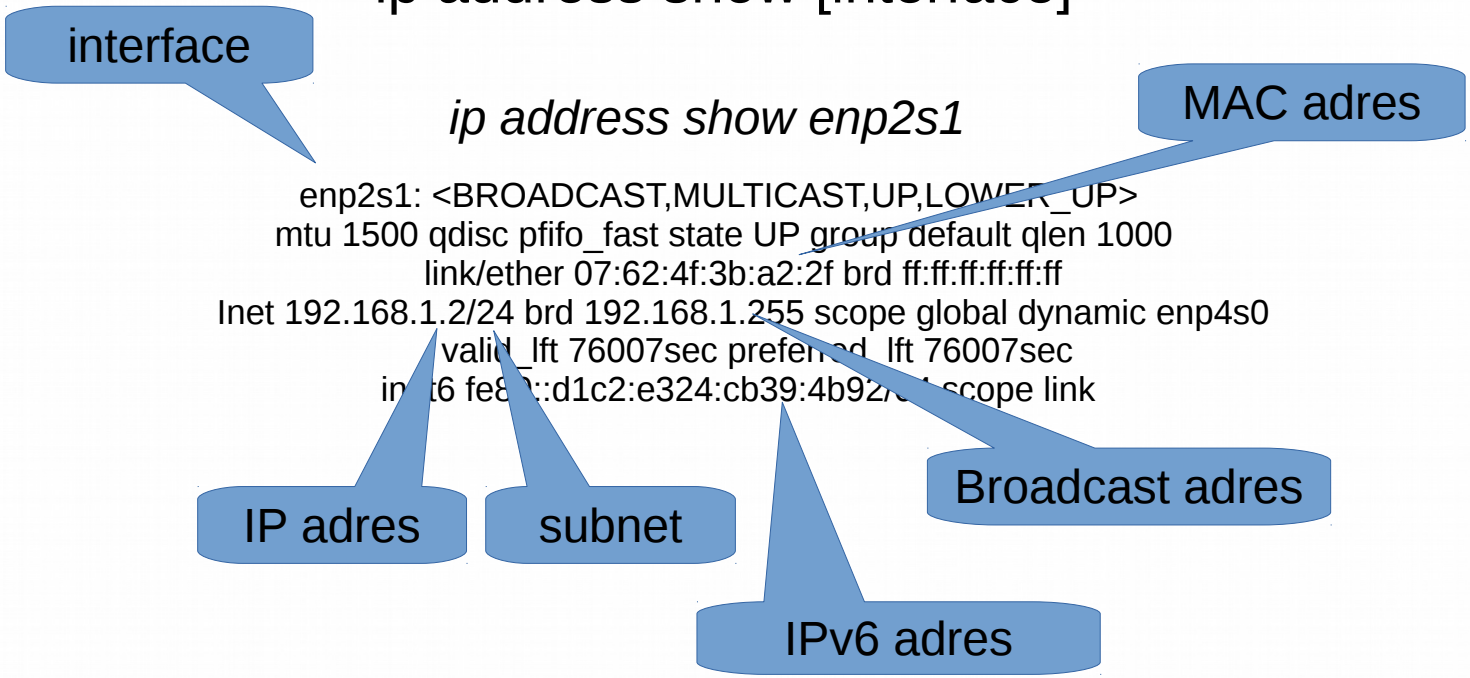
ip address

add, del, change, replace
show, save, flush
showdump, restore
help

# Linux Kernel

## ip commando

ip address show [interface]



# Linux Kernel

## ip commando

ip -4 a show [interface]

a → afkorting “address”
-4 → toon enkel ip v4 adres

*ip -4 a show enp2s1*

enp2s1: <BROADCAST,MULTICAST,UP,LOWER_UP>
mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
Inet 192.168.1.2/24 brd 192.168.1.255 scope global dynamic enp4s0
valid_lft 76007sec preferred_lft 76007sec

# Linux Kernel

## ip commando

-s → “statistics”

*ip –s a show enp2s1*

enp2s1: <BROADCAST,MULTICAST,UP,LOWER_UP>
mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
link/ether 07:62:4f:3b:a2:2f brd ff:ff:ff:ff:ff:ff
Inet 192.168.1.2/24 brd 192.168.1.255 scope global dynamic enp4s0
valid_lft 76007sec preferred_lft 76007sec
inet6 fe80::d1c2:e324:cb39:4b92/64 scope link
**RX: bytes packets errors dropped overrun mcast**
**540453979 393167 0 0 0 6127**
**TX: bytes packets errors dropped carrier collsns**
**34704467 182383 0 0 0 0**

# Linux Kernel

## ifconfig commando

voorloper van het ip commando

*ifconfig = ip link show*

```
eth0   Link encap:Ethernet  HWaddr 09:00:12:90:e3:e5
       inet addr:192.168.1.29 Bcast:192.168.1.255  Mask:255.255.255.0
       inet6 addr: fe80::a00:27ff:fe70:e3f5/64 Scope:Link
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:54071 errors:1 dropped:0 overruns:0 frame:0
       TX packets:48515 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:22009423 (20.9 MiB)  TX bytes:25690847 (24.5 MiB)
       Interrupt:10 Base address:0xd020

lo     Link encap:Local Loopback
       inet addr:127.0.0.1  Mask:255.0.0.0
       inet6 addr: ::1/128 Scope:Host
       UP LOOPBACK RUNNING  MTU:16436  Metric:1
       ...
```

# Linux Kernel

## Network services: routing

- Basis routing → weg van de pc naar buiten

maar ook:

- Ingebakken full-router functionaliteiten

# Linux Kernel

**ip commando**

ip routes

add, append
change, del, flush
get
list, monitor replace
help

# Linux Kernel

## ip commando

ip route list

Voorbeeld:

***ip route list***

default via 192.168.1.1 dev wlp1s0  proto static  metric 100
192.168.1.0/24 dev wlp1s0  proto kernel  scope link  src 192.168.1.2  metric 600
169.254.0.0/16 dev wlp1s0  scope link  metric 1000

# Linux Kernel

## ip commando

ip route get

Voorbeeld:

***ip route get 172.217.19.206***

*172.217.19.206 via 192.168.1.1 dev enp4s0  src 192.168.1.2*

*( 192.168.1.1 = router )*

# Linux Kernel

## ip commando

ip route add

Voorbeeld:

***ip route add 20.0.0.0/8 via 192.168.1.1***

***ip route list***

*20.0.0.0/8 via 192.168.1.1 dev wlp3s1*
…

*( 192.168.1.1 = router )*

# Linux Kernel

**ip commando**

ip neighbour

add, del, change, replace
show,  help

*ip neighbour →  arp*

# Linux Kernel

## ip commando

ip neighbour

Voorbeeld:

***ip neighbour show***

*192.168.1.1 dev enp2s1 lladdr 1c:0a:ff:03:7d:14 REACHABLE*
*192.168.1.7 dev enp2s1 lladdr 11:df:8f:18:fc:0c  REACHABLE*

# Linux Kernel

## ip commando

= zeer uitgebreidt commando:

- ip tunnel → netwerk tunneling
- ip l2tp → ethernet over IP tunnel
- ip maddress → multicast address
- ip mouter → multicast route
- ip xfrm → manage IPSec policies

enz ...

# Linux Kernel

## route commando

oude versie van ip route commando

*Voorbeeld:*

*route add default gw 192.168.1.254 eth1*
=
*ip route add default via 192.168.1.254 dev enp1s1*

gebruik altijd iproute2 commando's als het kan

# Linux Kernel

**ss commando**

ss = “socket statistics”

Geeft informatie over de verschillende netwerk/socket verbindingen

# Linux Kernel

**ss commando**

**ss [opties]**

-t → info over TCP verbindingen
-u → info over UDP verbindingen
-4 → enkel ipv4 sockets
-6 → enkel ipv6 sockets
-d → info over DHCP messages

# Linux Kernel

## ss commando

## ss [opties]

-p → info over process v/d verbinding
-a → all : alle verbindingen
src :[port] → source poort nummer
dst :[port] → destination poort nummer

# Linux Kernel

## ss commando

Voorbeeld:

***ss -t4p***

→ *tcp, v4, process*

| *State* | *Recv-Q* | *Send-Q* | *Local Address:Port* | *Peer Address:Port* |
|---|---|---|---|---|
| *ESTAB* | *0* | *0* | *192.168.1.2:42131* | *172.213.11.122:https* |

*Users:*
*(("firefox-browser",pid=2152,fd=205))*

# Linux Kernel

## ss commando

Voorbeeld:

***ss dst :443***

| *Netid* | *State* | *Recv-Q* | *Send-Q* | *Local Address:Port* | *Peer Address:Port* |
|---|---|---|---|---|---|
| *tcp* | *ESTAB* | *0* | *0* | *192.168.1.2:12987* | *172.213.21.123:https* |

# Linux Kernel

**netstat commando**

netstat = voorloper van ss

- toont eveneens de verbindingen
- Netstat is trager en toont minder info
- Gebruik als het kan iproute2 commando

# Linux Kernel

## ping commando

ping [ipadres][options]

Voorbeeld:

*ping 8.8.8.8 -c 3*

*PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.*
*64 bytes from 8.8.8.8: icmp_seq=1 ttl=59 time=14.3 ms*
*64 bytes from 8.8.8.8: icmp_seq=2 ttl=59 time=12.6 ms*
*64 bytes from 8.8.8.8: icmp_seq=3 ttl=59 time=13.6 ms*

*--- 8.8.8.8 ping statistics ---*
*3 packets transmitted, 3 received, 0% packet loss, time 2003ms*
*rtt min/avg/max/mdev = 12.652/13.532/14.332/0.688 ms*

# Linux Kernel

**tcpdump commando**

tcpdump is een soort “mini wireshark”
ingebouwd in de linux kernel

Gebruik:

tcpdump [opties][protocol]

# Linux Kernel

## tcpdump commando

-c → count: capture ‘n’ aantal packets
-i → interface: luister enkel naar deze interface
-v → verbose: geef meer details
-vv -vvv -vvvv (nog meer details)
-w → write: schrijft de output weg naar een bestand

src port → luister specifiek naar de opgegeven poort
dst port → src = source en dst = destination

# Linux Kernel

## tcpdump commando

Voorbeeld:

tcpdump -i enp2s1 -c 5

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp2s1, link-type EN10MB (Ethernet), capture size 262144 bytes

11:12:43.247830 ARP, Request who-has 192.168.1.2 tell 192.168.1.1, length 46
11:12:43.248857 IP pc1.loc.45141 > 192.168.1.1.domain: 54855+ PTR? 2.1.168.192.in-addr.arpa.
11:12:43.249968 IP 192.168.200.254.domain > pc1.loc.45141: 54855 NXDomain 0/0/0
11:12:43.250500 IP pc1.loc.45141 > 192.168.1.1.domain: 20674+ PTR? 1.1.168.192.in-addr.arpa.
11:12:43.251533 IP 192.168.200.254.domain > pc1.loc.45141: 20674 NXDomain 0/0/0

5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

# Linux Kernel

## tcpdump commando

Voorbeeld:

tcpdump –v icmp -c 3

```
tcpdump: listening on wlp2s1, link-type EN10MB (Ethernet), capture size 262144 bytes

11:29:32.312444 IP (tos 0x0, ttl 255, id 54342, offset 0, flags [none], proto ICMP (1), length 50)
    192.168.1.1 > pc1.loc: ICMP echo request, id 1917, seq 1, length 30

11:29:32.312505 IP (tos 0x0, ttl 64, id 14226, offset 0, flags [none], proto ICMP (1), length 50)
    pc1.loc > 192.168.1.1: ICMP echo reply, id 1917, seq 1, length 30

11:29:33.318584 IP (tos 0x0, ttl 255, id 54343, offset 0, flags [none], proto ICMP (1), length 50)
    192.168.1.1 > pc1.loc: ICMP echo request, id 1917, seq 2, length 30

3 packets captured
4 packets received by filter
0 packets dropped by kernel
```

# Linux Kernel

**tcpdump commando**

FYI:

… voor wie nog meer mogelijkheden wil dan tcpdump
er bestaat ook een échte (command-line) wireshark

***tshark***

*sudo apt-get install tshark*

# Linux Kernel

## tcpdump commando

Voorbeeld:

***$> tshark -f “host 192.168.99.66” -c 3***

```
Capturing on 'wlp36s1'

   1 0.000000000 172.16.4.154 → 172.217.169.48 TCP 66 32902 → 80 [ACK] Seq=1 Ack=1
     Win=40 Len=0 TSval=157888 TSecr=3538294185
   2 0.012146150 172.217.169.48 → 172.16.4.154 TCP 66 [TCP ACKed unseen segment] 80 →
     32902 [ACK] Seq=1 Ack=2 Win=240 Len=0 TSval=3538339242 TSecr=124083
   3 4.791328308 172.16.4.154 → 172.217.169.48 TCP 66 [TCP Previous segment not
     captured] 32902 → 80 [FIN, ACK] Seq=2 Ack=1 Win=40 Len=0 TSval=159085

3 packets captured
```

# Linux Kernel

**traceroute commando**

- Bekijken traject van een packet
- van bron tot destinatie adres
- niet altijd geïnstalleerd → apt-get install traceroute
- Gebruik:

traceroute [ip-address]

# Linux Kernel

## traceroute commando

Voorbeeld:

*traceroute 74.125.236.132*

```
traceroute to 74.125.236.132, 30 hops max, 60 byte packets
1  220.224.141.129 (220.224.141.129)  89.174 ms  89.094 ms  89.054 ms
2  115.255.239.65 (115.255.239.65)  109.037 ms  108.994 ms  108.963 ms
3  124.124.251.245 (124.124.251.245)  108.937 ms  121.322 ms  121.300 ms
4  * 115.255.239.45 (115.255.239.45)  113.754 ms  113.692 ms
5  72.14.212.118 (72.14.212.118)  123.585 ms  123.558 ms  123.527 ms
6  72.14.232.202 (72.14.232.202)  123.499 ms  123.475 ms  143.523 ms
7  216.239.48.179 (216.239.48.179)  143.503 ms  95.106 ms  95.026 ms
8  bom03s02-in-f4.1e100.net (74.125.236.132)  94.980 ms  104.989 ms
```

# Linux Kernel

## nslookup commando

- domain name resolving
- checken of DNS werkt + instellingen juist zijn
- niet altijd geïnstalleerd → apt-get install nslookup
- Gebruik:

nslookup [url]

# Linux Kernel

## nslookup commando

Voorbeeld:

*nslookup www.ap.be*

```
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:           www.ap.be
Address:        193.191.187.241
```

# Linux Kernel

## nmap commando

- nmap = network mapper
- Testen van netwerken
- Security scan’s
- Netwerk audits
- niet altijd geïnstalleerd → apt-get install nmap
- Gebruik:

nmap [scan type] [opties] [target]

# Linux Kernel

## nmap commando

Voorbeeld:

*nmap localhost* → testen welke poorten open/dicht zijn

*Starting Nmap 7.01*
*Nmap scan report for localhost (127.0.0.1)*
*Host is up (0.000076s latency).*
*Not shown: 998 closed ports*
*PORT STATE SERVICE*
*22/tcp open ssh*
*631/tcp open ipp*

*Nmap done: 1 IP address (1 host up) scanned in 0.05 seconds*

# Linux Kernel

## nmap commando

**Opgepast!**

nmap kan ook subnets en volledige netwerken scannen

**het port-scannen van publieke ip-adressen**
**wordt in heel wat landen wettelijk gezien**
**als een illegale cyber aanval !**

*Gebruik dit commando met de nodige voorzichtigheid!*

# Linux Kernel

**NetworkManager**

# Linux Kernel

## NetworkManager

- Bedoelt voor de gewone thuis gebruikers
- “plug-and-play” functionaliteit

- Instellingen via de Desktop-Environment (GUI)
- Instellingen kunnen ook via terminal:

**$> nmcli**

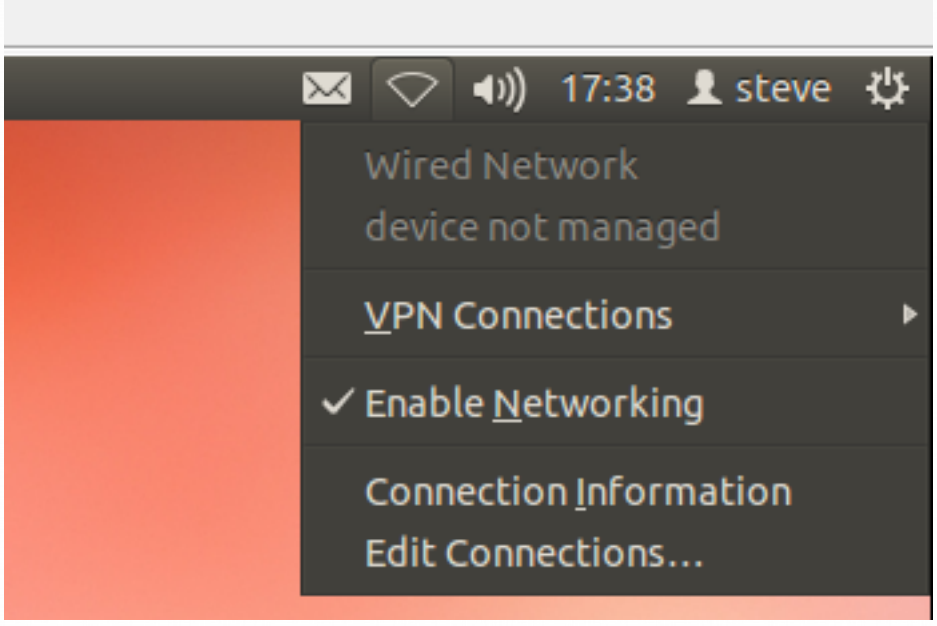
# Linux Kernel

## NetworkManager

- Networkmanager is een deamon:
  *systemctl status NetworkManager.service*
- Configuratie bestanden in:
  */etc/NetworkManager*
- Soms afwezig op embedded boards, servers, etc
  *( werkt niet goed samen met cPanel )*

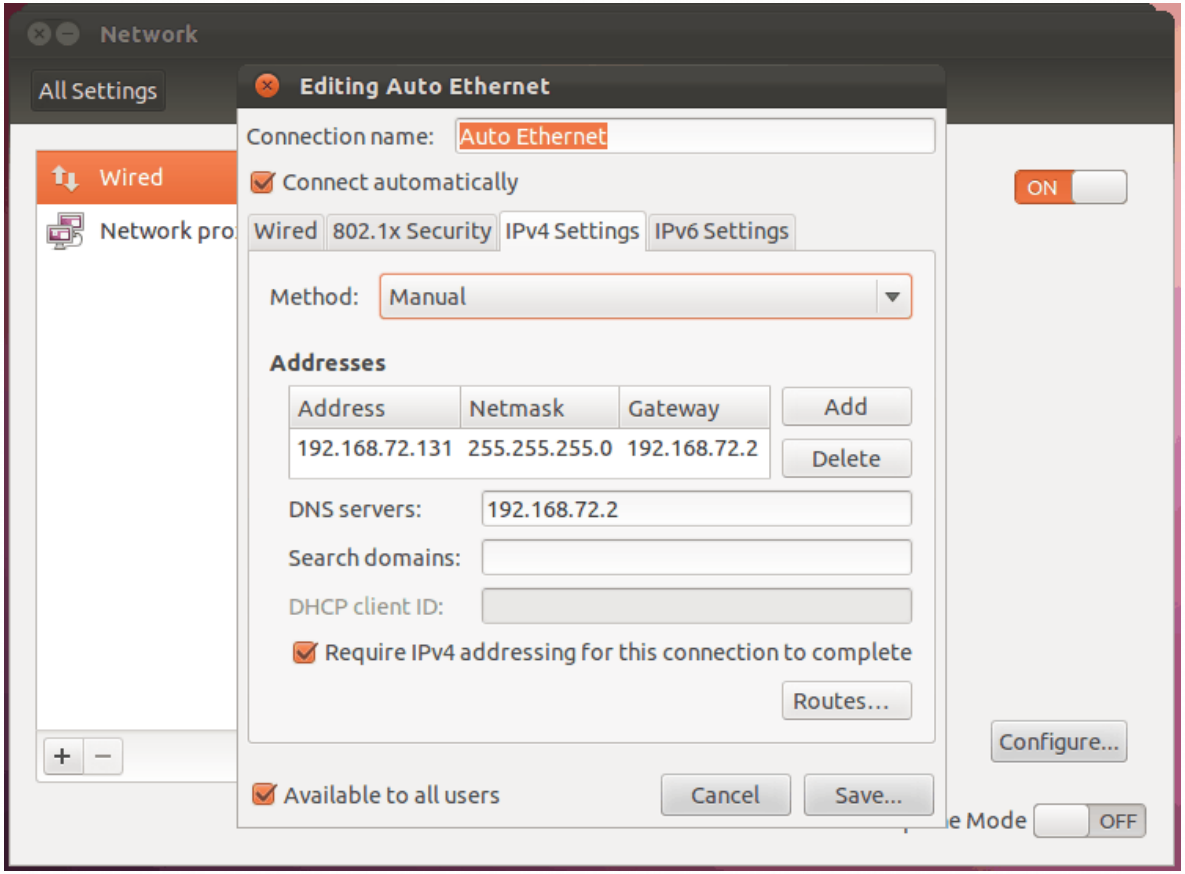
# Linux Kernel

## NetworkManager



# Linux Kernel

## NetworkManager



# Linux Kernel

## Network Services: firewall

- volwaardige firewall in de linux-kernel
- Firewall werkt met “rules”
  → regels wat er moet gebeuren met bepaalde packets
- Firewall werkt met “chains” → schakels v/e ketting
- configureerbaar via bash
- commando:

iptables [options][chains][rules]

# Linux Kernel

## Network Services: firewall

Voorbeeld:

*iptables --list*

```
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0           state INVALID
  394 43586 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
   93 17292 ACCEPT     all  --  br0    *       0.0.0.0/0            0.0.0.0/0
    1   142 ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     all  --  br0    br0     0.0.0.0/0            0.0.0.0/0
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0           state INVALID
    0     0 TCPMSS     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp flags:0x06/0x02 TCPMSS

Chain OUTPUT (policy ACCEPT 425 packets, 113K bytes)
 pkts bytes target     prot opt in     out     source               destination
```

# Linux Kernel

**Network Services: firewall**

alle opties van iptables en
mogelijkheden van de firewall

=

genoeg stof voor een
volledig apparte cursus...

# Linux Kernel

**Opbouw**

Process Scheduler
Memory Manager
File System
Device Driver
Network Services
***I/O Scheduler***
IPC → Inter Process Communications

# Linux Kernel

## I/O Scheduler

- regelt het input-output verkeer naar devices
- devices maken vaak gebruik van dezelfde bus ( bus → collectie van verbindingen bvb PCI)
- devices hebben echter ook verschillende respons snelheden (bvb: ssd sneller dan hdd)
- sommige i/o operaties hebben een deadline, ze moeten klaar zijn vóór een ander event

# Linux Kernel

## I/O Scheduler

- De I/O scheduler neemt al deze taken op zich
- zeer bepalend voor de performatie v/h systeem
- verschillende versies bestaan
- Linux default = “**Completely Fair Queuing**”

  → **CFQ**

# Linux Kernel

**Opbouw**

Process Scheduler
Memory Manager
File System
Device Driver
Network Services
I/O Scheduler
***IPC → Inter Process Communications***

# Linux Kernel

## Inter Process Communications

- Afkorting → IPC
- processen zijn vaak van elkaar afhankelijk doch kunnen niet aan elkaars geheugen stukjes
- ze moeten ook wachten op data, resources, etc
- processen moeten ook vaak met elkaar communiceren:
  → service kan bestaan uit meerdere processen

# Linux Kernel

## Inter Process Communications

- IPC regelt al dit verkeer onder andere d.m.v ‘signals’
- IPC regelt dus de communicatie tussen de processen

  → *“jij mag nu spreken”, “ik heb een bericht voor jou”, …*

- Als we het pipe commando gebruiken is het ook IPC die zorgt voor de communicatie tussen de commando’s

  → *ls | more*

# Linux Kernel

## Kernel Logging

# Linux Kernel

## Kernel Logging

- de kernel logt ook alle events die gebeuren
- Nodig voor debuggen, indien er iets fout gaat
- Meerdere log bestanden
- te vinden in: */var/log*

# Linux Kernel

## Kernel Logging

- btmp → volledige status van het systeem
- utmp →failed login pogingen
- etc...

- belangrijkste kernel logging in “dmesg” bestand

- Bash commando: ***dmesg*** → drukt deze log af.

# Linux Kernel

## Kernel Logging

## dmesg [options]

Voorbeeld:

*dmesg | grep usb | tail 8*

```
[ 0.131874] ACPI: bus type usb registered
[ 0.131910] usbcore: registered new interface driver usbfs
[ 0.131921] usbcore: registered new interface driver hub
[ 0.131955] usbcore: registered new device driver usb
[ 1.091722] usb usb1: New USB device found, idVendor=1d6b, idProduct=0001
[ 1.091727] usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[ 1.091730] usb usb1: Product: OHCI Host Controller
[ 1.091731] usb usb1: Manufacturer: Linux 3.8.0-31-generic ohci_hcd
```

# Linux Kernel

## Kernel Logging

**demo**
**dmesg**