CLOUDFLARE

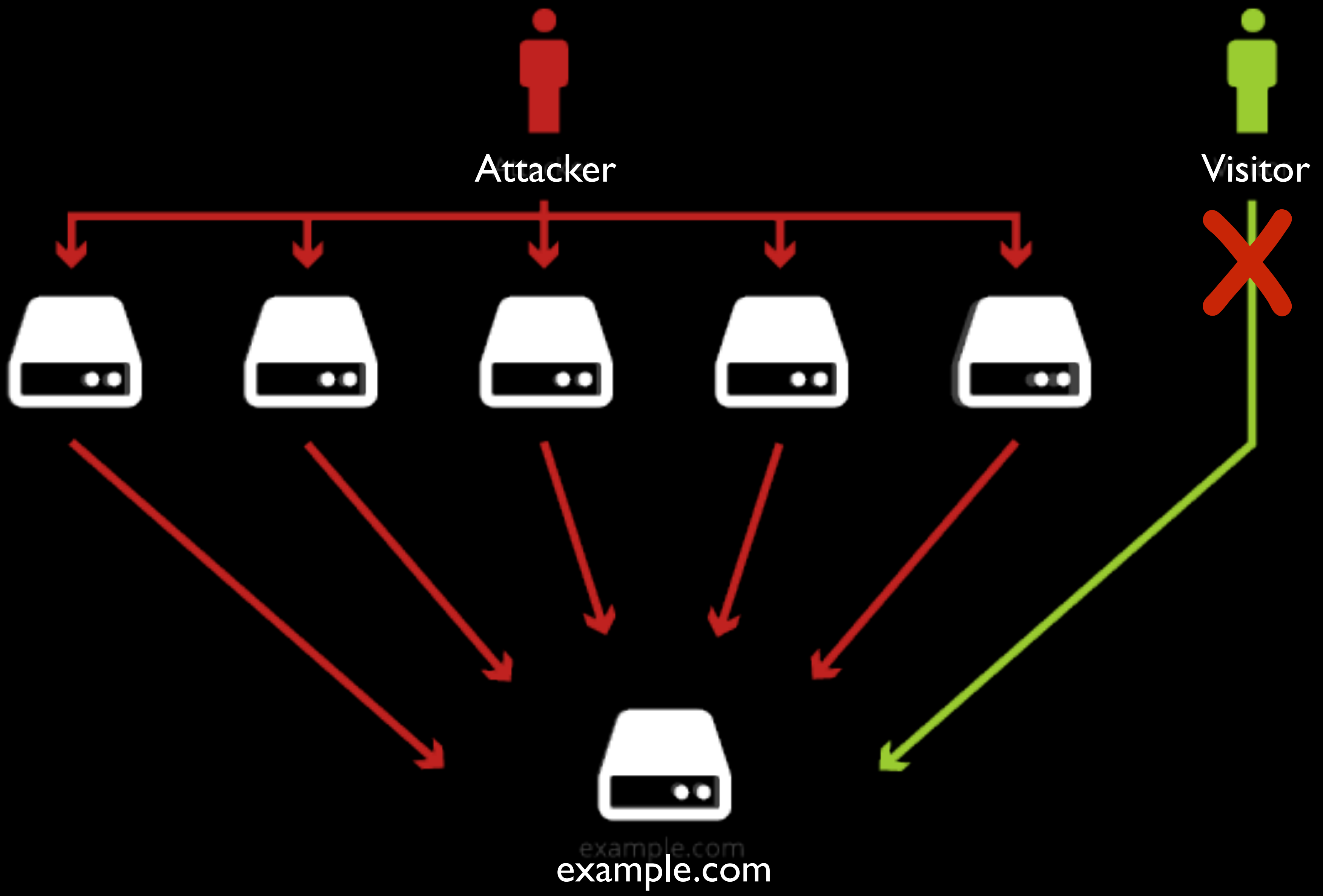
Lessons from defending the indefensible

Marek Majkowski

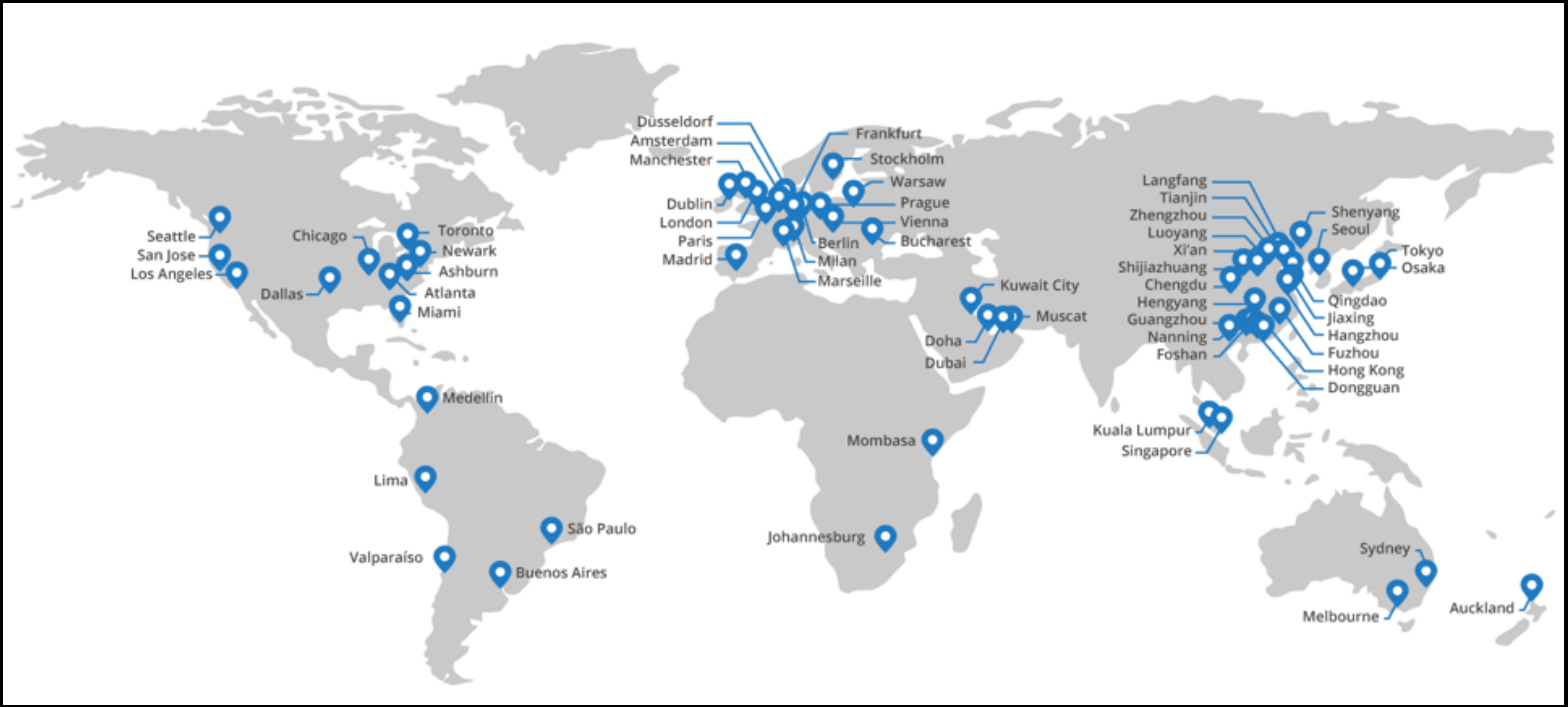
marek@cloudflare.com @majek04

Denial of service (DoS)

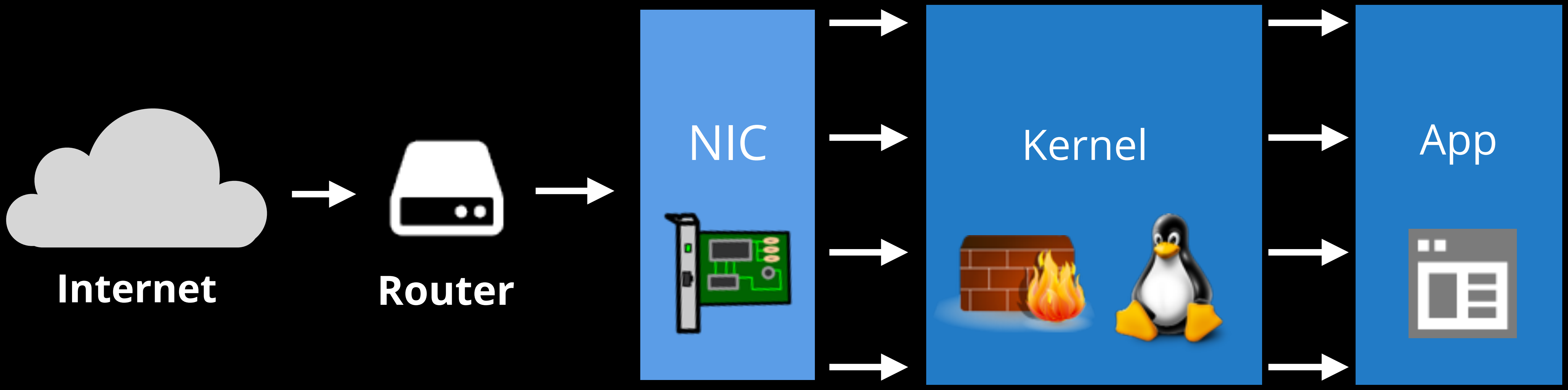
DoS is a hard problem



Unique view


Düsseldorf
Amsterdam
Manchester
Frankfurt
Stockholm
Warsaw
Prague
Vienna
Bucharest
Dublin
London
Paris
Madrid
Berlin
Milan
Marseille
Seattle
San Jose
Los Angeles
Chicago
Toronto
Newark
Ashburn
Atlanta
Miami
Dallas
Medellín
Lima
São Paulo
Valparaíso
Buenos Aires
Kuwait City
Muscat
Doha
Dubai
Mombasa
Johannesburg
Langfang
Tianjin
Zhengzhou
Luoyang
Xi'an
Shijiazhuang
Chengdu
Hengyang
Guangzhou
Nanning
Foshan
Shenyang
Seoul
Tokyo
Osaka
Qingdao
Jiaxing
Hangzhou
Fuzhou
Hong Kong
Dongguan
Kuala Lumpur
Singapore
Sydney
Melbourne
Auckland

Attack surface



Internet

Router

NIC

Kernel

App

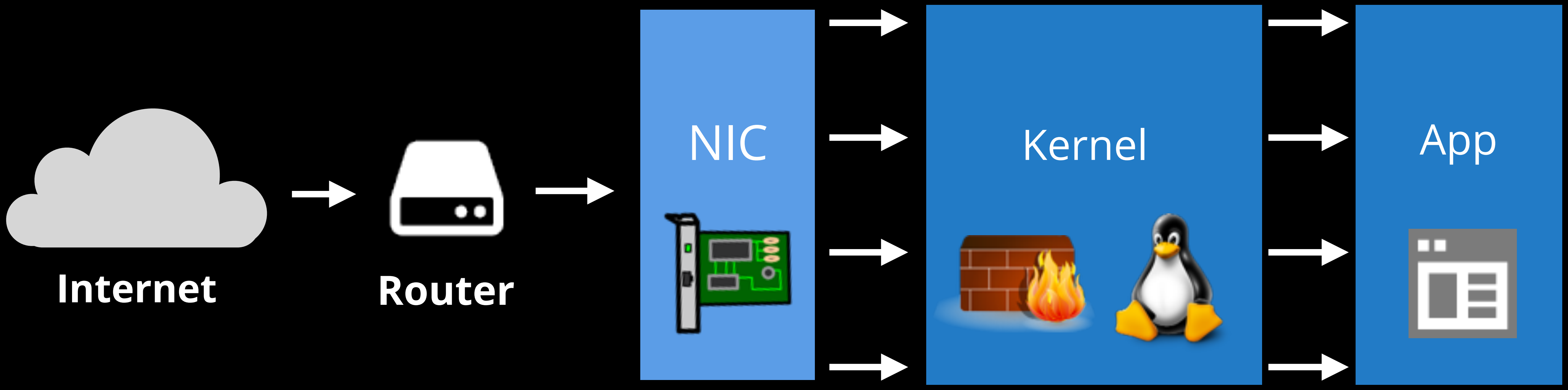
Congestion

Packet handling

Network stack

Slow response

Packets per second



Internet

Router

NIC

Kernel

App

∞

>20M

10M

~1M

0.3M

0.05M

Your Linux needs your help

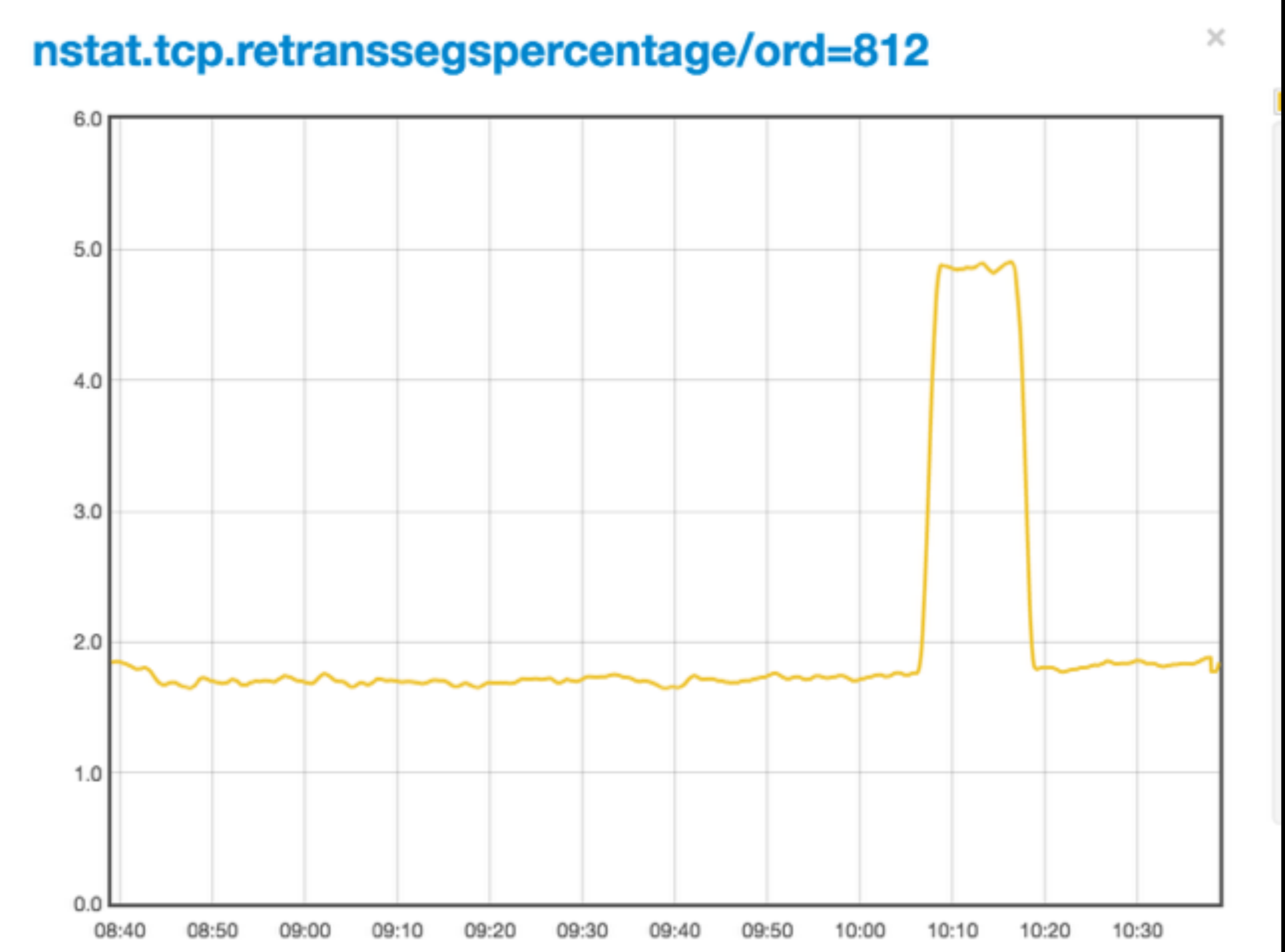
∞ pps

Network congestion

Congestion

```
$ netstat -s
Tcp:
    ...
    2291681363 segments send out
    43887463 segments retransmited
    ...
```

Congestion



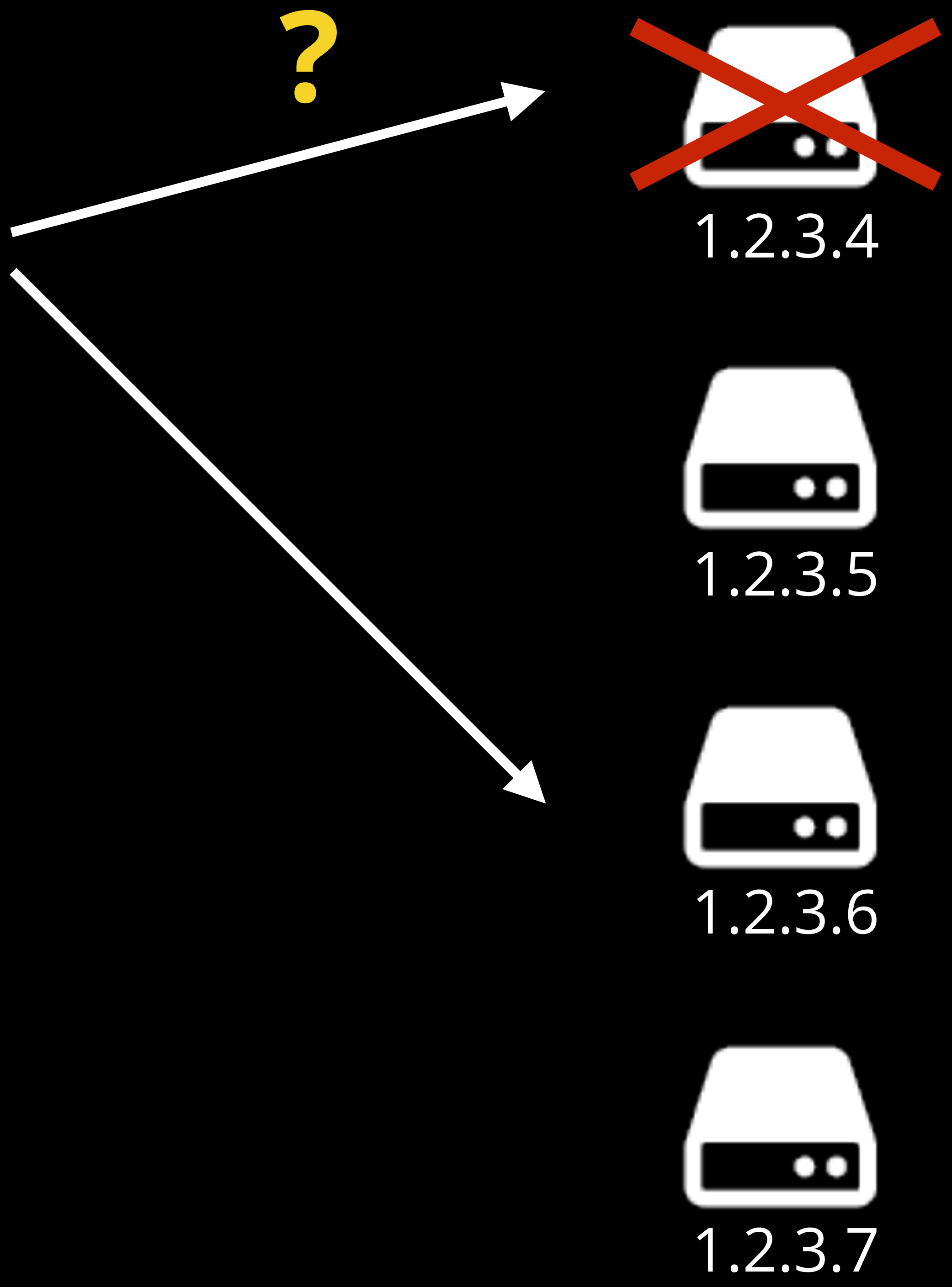
nstat.tcp.retranssegspercentage/ord=812
×
6.0
5.0
4.0
3.0
2.0
1.0
0.0
08:40
08:50
09:00
09:10
09:20
09:30
09:40
09:50
10:00
10:10
10:20
10:30

BGP null routing

```
route 198.41.222.X/32 {
    discard;
    community [ 13335:666 13335:668 13335:36006];
}
```

Application integration

```
dig A example.com
```

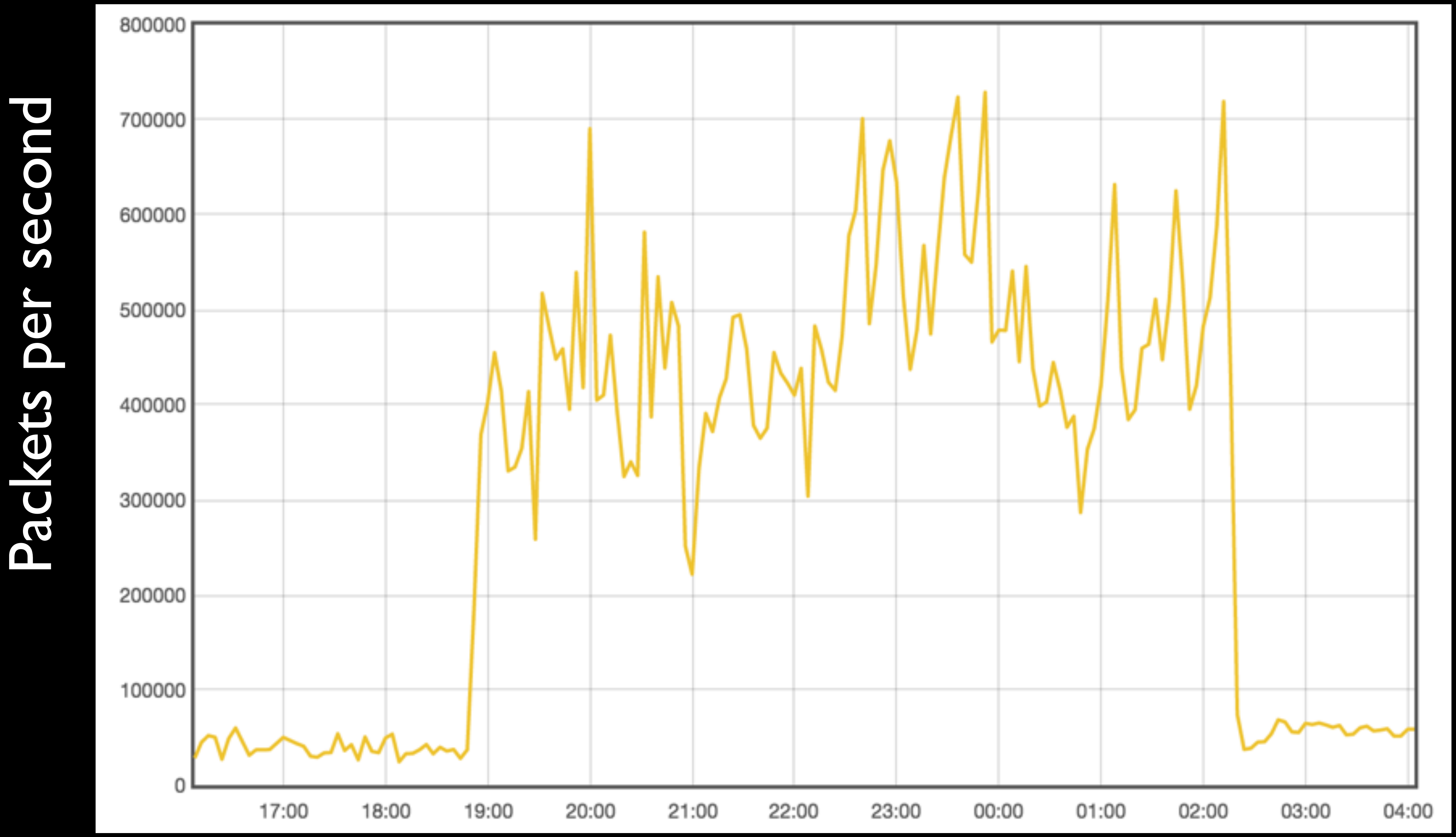


High volume packet floods

Let it flow

Yogendra Joshi

Looks like this



Spoofed?

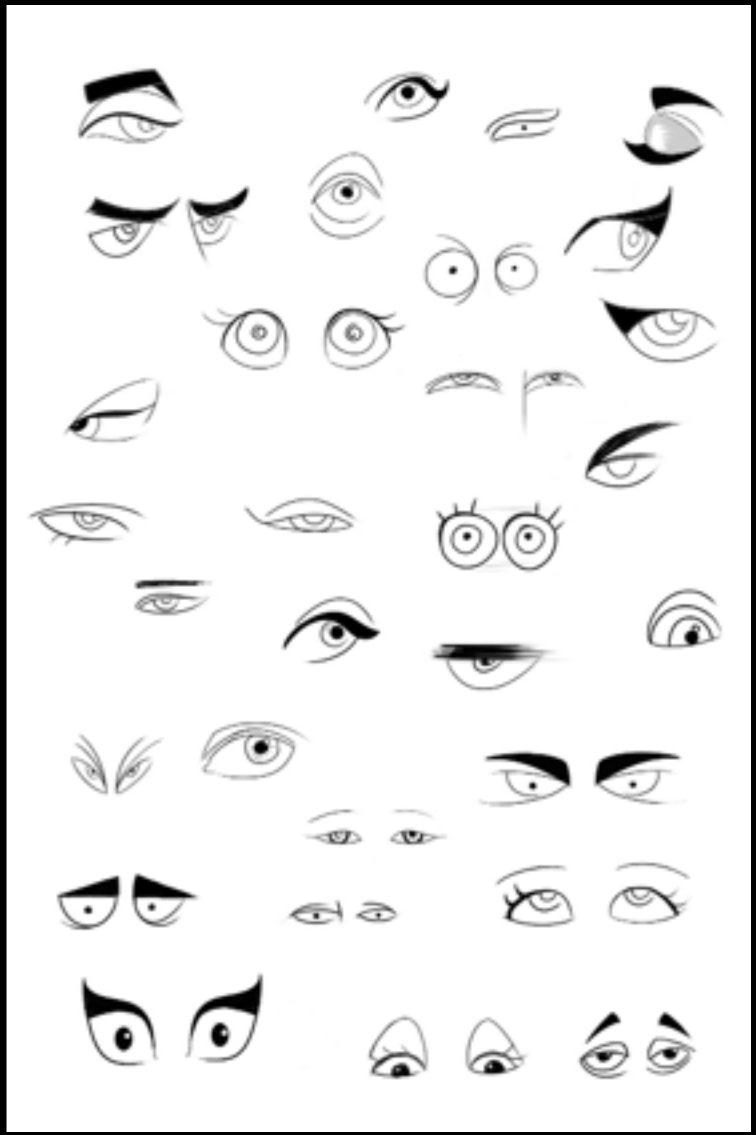
Drop!

UDP flood

```
IP 23.243.202.207.1076 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
IP 98.151.75.108.32856 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
IP 23.118.154.219.33894 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
IP 23.242.35.159.1036 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
IP 23.240.170.79.33842 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
IP 23.241.212.223.2052 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
IP 187.204.126.111.59011 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
IP 24.24.164.88.2759 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
IP 23.242.122.1.32778 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
IP 23.240.26.33.1043 > 1.2.3.4:53: 18516 updateD% [b2&3=0x5450] [11825a]
```

Packet characteristics

- packet length
- source IP's
- source port
- IPID field
- payload



~1.2M pps

Matching on payload in iptables

Payload matching with BPF

```
iptables -A INPUT \
    --dst 1.2.3.4 \
    -p udp --dport 53 \
      -m bpf --bytecode "14,0 0 0 20,177 0 0 0,12 0 0
 0,7 0 0 0,64 0 0 0,21 0 7 124090465,64 0 0 4,21 0 5
 1836084325,64 0 0 8,21 0 3 56848237,80 0 0 12,21 0 1
 0,6 0 0 1,6 0 0 0" \
    -j DROP
```

BPF bytecode

```
    ldx 4*([14]&0xf)
    ld #34
    add x
    tax
lb_0:
    ldb [x + 0]
    add x
    add #1
    tax
    ld [x + 0]
    jneq #0x07657861, lb_1
    ld [x + 4]
    jneq #0x6d706c65, lb_1
    ld [x + 8]
    jneq #0x03636f6d, lb_1
    ldb [x + 12]
    jneq #0x00, lb_1
    ret #1
lb_1:
    ret #0
```

Tcpdump expressions

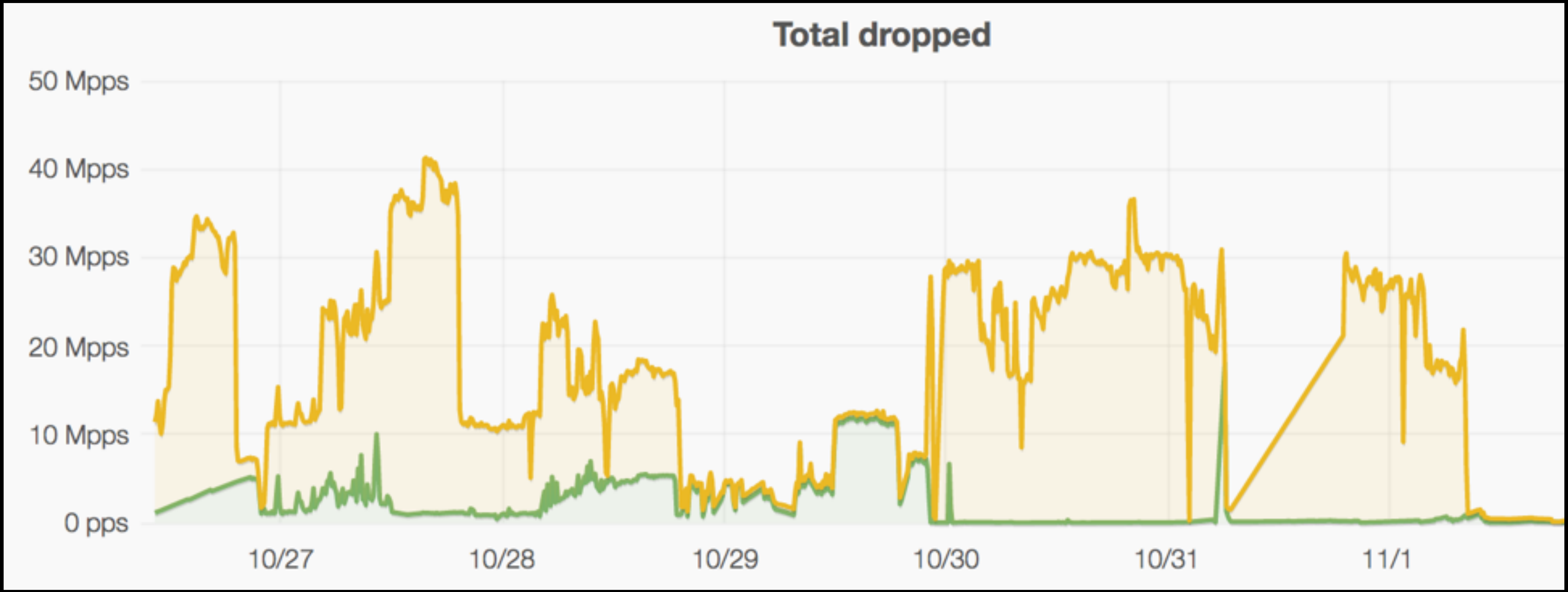
- Originally:

```
tcpdump -n “udp and port 53”
```

- xt_bpf implemented in 2013 by Willem de Bruijn
- Need to deal with BPF byte code
- Tools around it are scarce (tcpdump expressions)
- Tcpdump expressions are limited - for example matching valid DNS packets case insensitive
- Need to hand-craft BPF

BPF tools

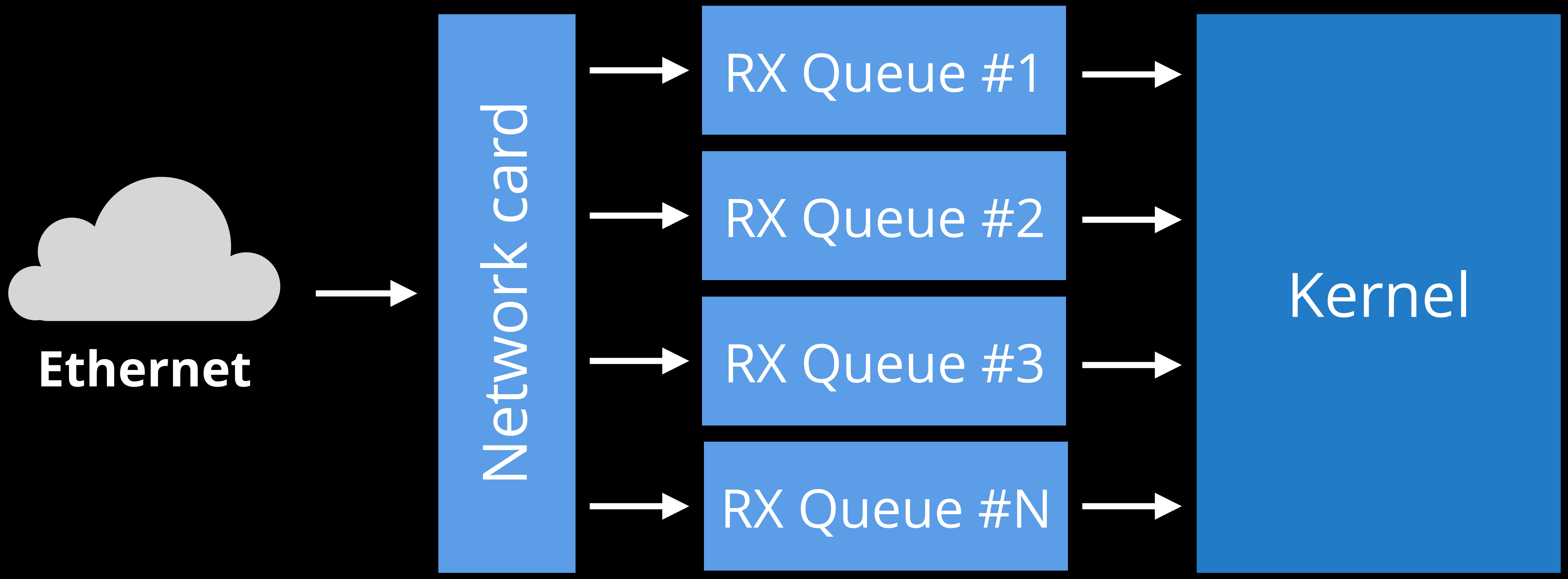
- Open source:
  - https://github.com/cloudflare/bpftools
- Can match various DNS patterns:
  - `*.example.com`
  - `??.example.com`
  - `*{1-4}.example.com`
  - `--case-insensitive *.example.com`
  - `--invalid-dns`


Total dropped
50 Mpps
40 Mpps
30 Mpps
20 Mpps
10 Mpps
0 pps
10/27
10/28
10/29
10/30
10/31
11/1

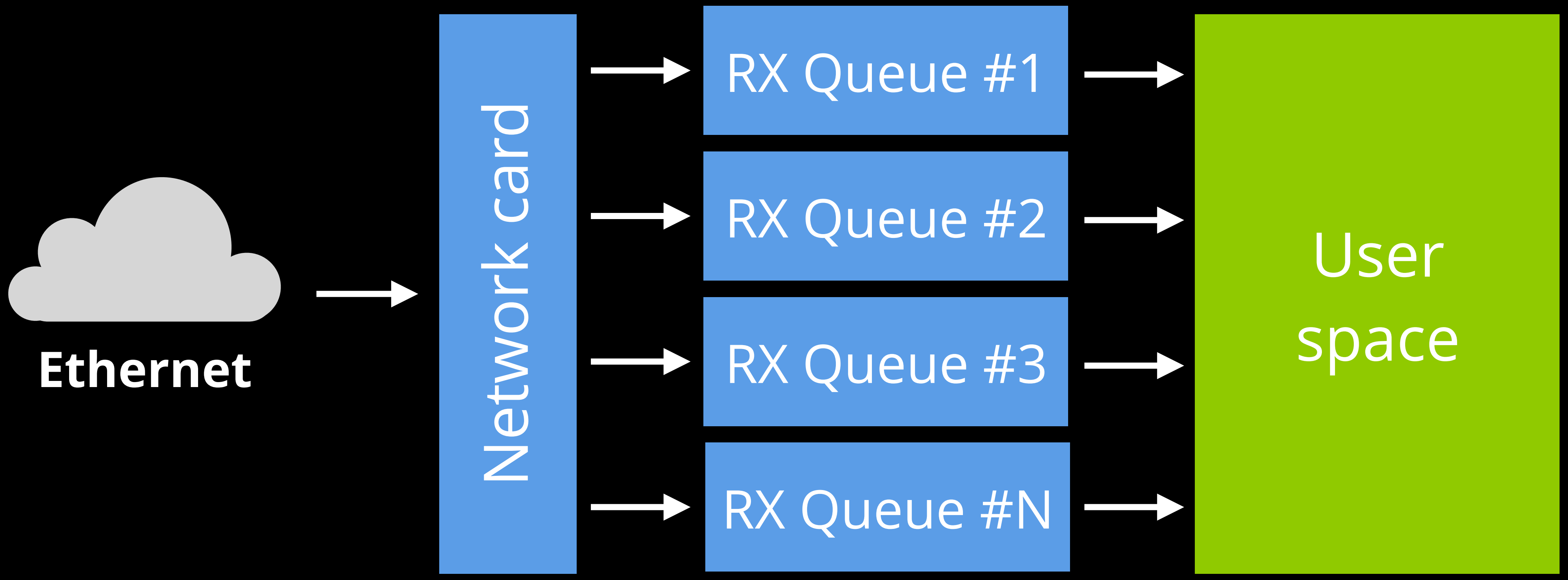
~4M pps

Payload matching close to NIC

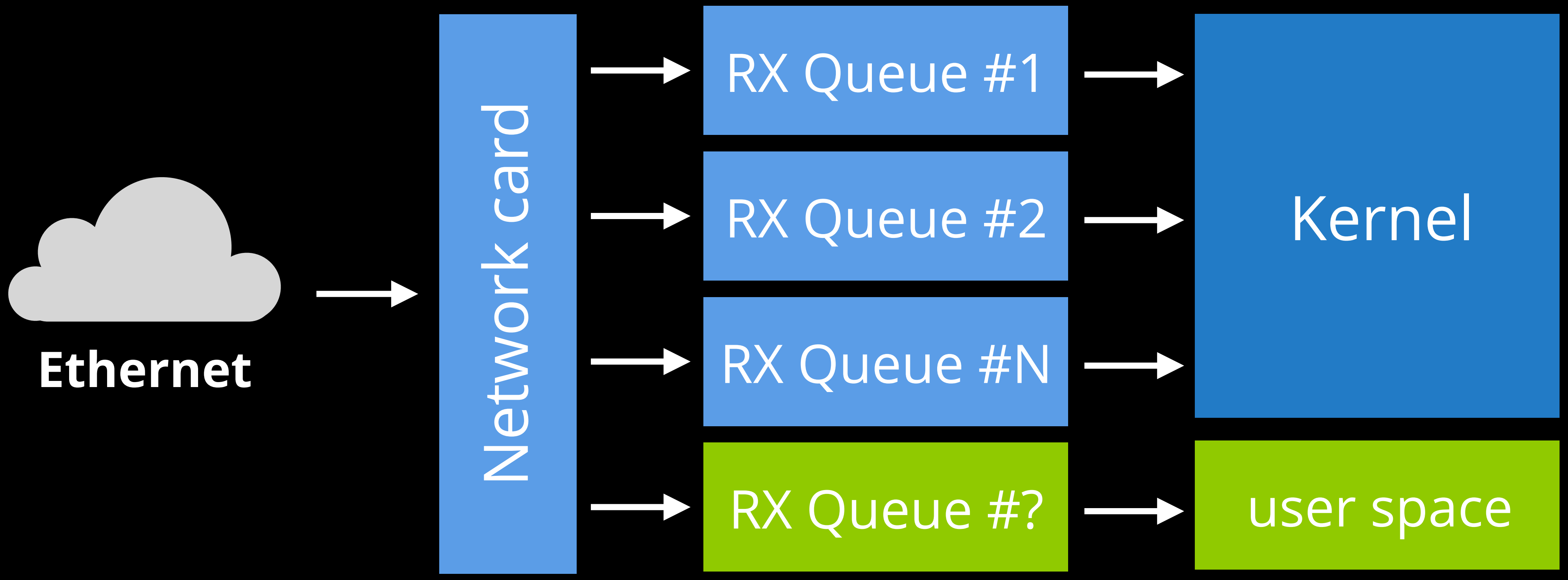
Modern NIC's

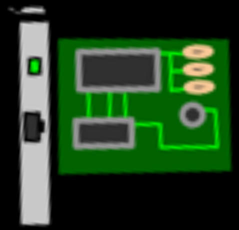

Ethernet
Network card
RX Queue #1
RX Queue #2
RX Queue #3
RX Queue #N
Kernel

Traditional kernel bypass



Partial kernel bypas

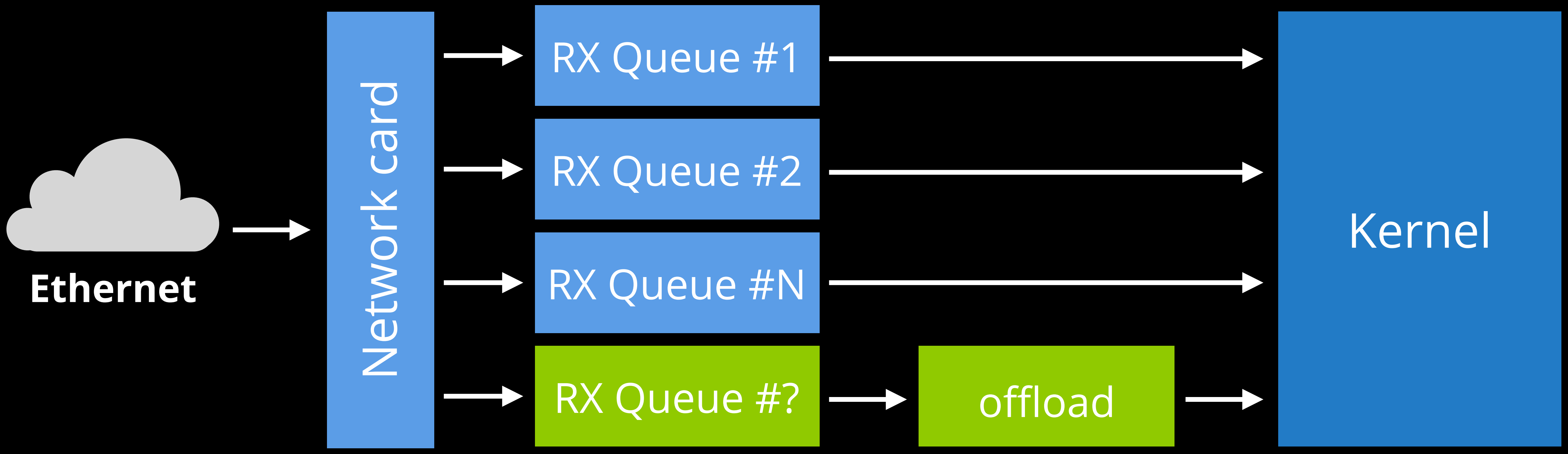




Partial kernel bypass

- Or EFVI for SolarFlares:
  - http://www.openonload.org/
- Open sourced netmap patch, tested on Intel:
  - https://github.com/luigirizzo/netmap/pull/87

Iptables offload



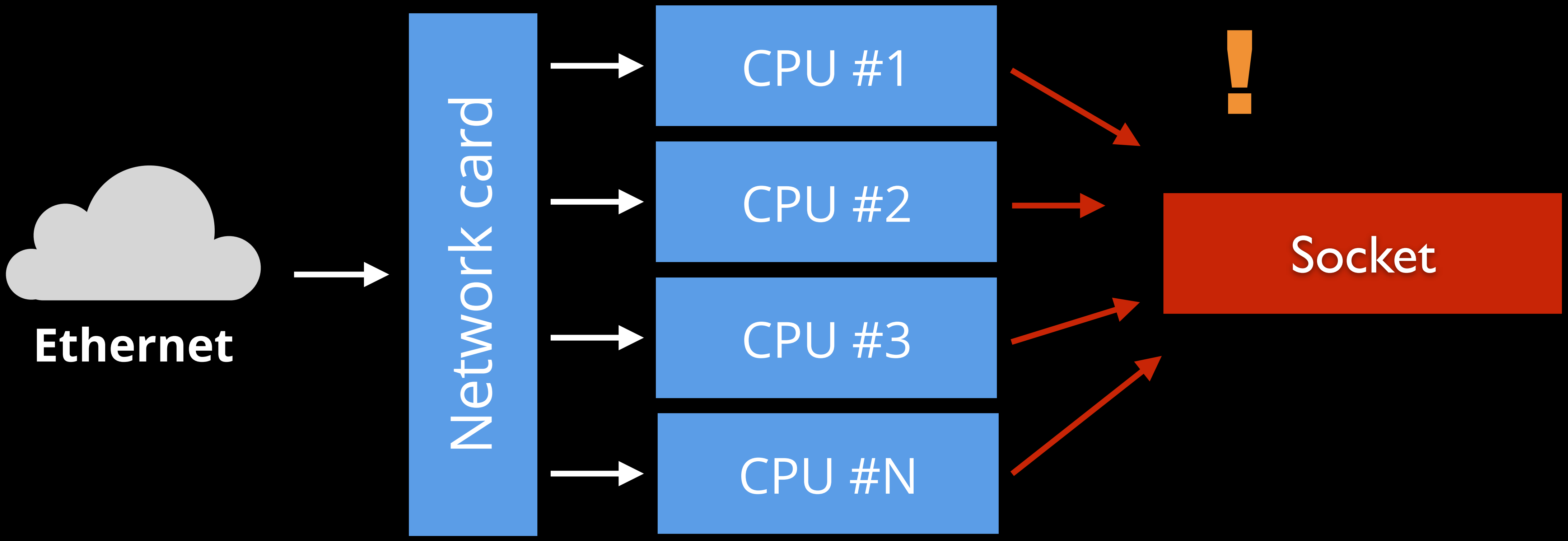
No characteristics: Attacks against TCP/IP network stack

ACK floods

```
IP 48.60.32.50.15244 > 1.2.3.4.80: Flags [P.], ack 1754729313, win 16153
IP 31.102.214.103.13396 > 1.2.3.4.80: Flags [P.], ack 1569851274, win 15707
IP 112.36.216.55.56515 > 1.2.3.4.80: Flags [P.], ack 2051477187, win 16102
IP 65.130.63.30.10341 > 1.2.3.4.80: Flags [P.], ack 2108282782, win 16112
IP 16.18.205.115.15962 > 1.2.3.4.80: Flags [P.], ack 1359019408, win 16119
IP 128.177.247.54.13752 > 1.2.3.4.80: Flags [P.], ack 1416531343, win 16102
IP 204.59.118.78.61528 > 1.2.3.4.80: Flags [P.], ack 348671255, win 16101
IP 119.195.142.20.3344 > 1.2.3.4.80: Flags [P.], ack 1917538144, win 16161
IP 70.197.6.24.39340 > 1.2.3.4.80: Flags [P.], ack 1920842431, win 16124
```

~0.3M pps

Fight for the lock



Statefull firewall - conntrack

```
iptables -A INPUT \
   --dst 1.2.3.4 \
   -m conntrack --ctstate INVALID \
   -j DROP
```

```
sysctl -w net/netfilter/nf_conntrack_tcp_loose=0
```

~1.2M pps

Effective

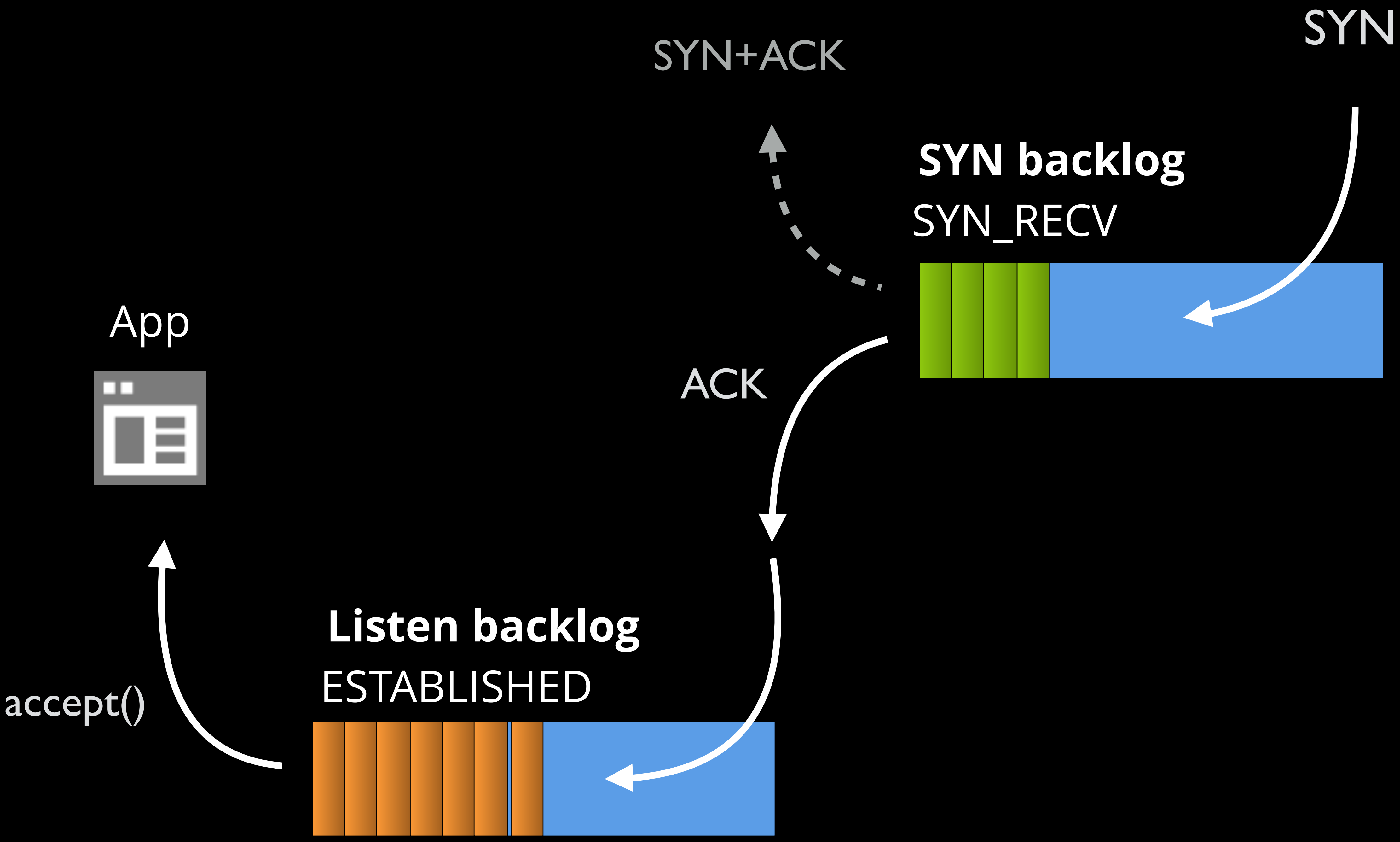
- Works well against:
  - ACK
  - FIN
  - RST
  - X-mas
- What about SYN floods?

SYN floods

```
IP 94.242.250.109.47330 > 1.2.3.4:80: Flags [S], seq 1444613291, win 63243
IP 188.138.1.240.61454 > 1.2.3.4:80: Flags [S], seq 1995637287, win 60551
IP 207.244.90.205.17572 > 1.2.3.4:80: Flags [S], seq 1523683071, win 61607
IP 94.242.250.224.65127 > 1.2.3.4:80: Flags [S], seq 928944042, win 61778
IP 207.244.90.205.43074 > 1.2.3.4:80: Flags [S], seq 137074667, win 63891
IP 64.22.81.44.23865 > 1.2.3.4:80: Flags [S], seq 838596928, win 63808,
IP 188.138.1.137.23373 > 1.2.3.4:80: Flags [S], seq 593106072, win 60272
IP 207.244.90.205.39653 > 1.2.3.4:80: Flags [S], seq 47289666, win 63210
IP 208.66.78.204.64197 > 1.2.3.4:80: Flags [S], seq 1850809890, win 62714
IP 207.244.90.205.33108 > 1.2.3.4:80: Flags [S], seq 319707959, win 63351
IP 207.244.90.205.6937 > 1.2.3.4:80: Flags [S], seq 1591500126, win 63902
IP 213.152.180.151.60560 > 1.2.3.4:80: Flags [S], seq 1902119375, win 62511
IP 64.22.79.127.11061 > 1.2.3.4:80: Flags [S], seq 1456438676, win 62148
```

0M pps

SYN in Linux


SYN
SYN+ACK
SYN backlog
SYN_RECV
App
ACK
Listen backlog
ESTABLISHED
accept()

SYN backlog size

1. Listen backlog size

```
listen(int sockfd, int backlog)
```

2. Listen backlog size capped by

```
sysctl -w net.core.somaxconn = 65535
```

3. SYN backlog capped with

```
sysctl -w net.ipv4.tcp_max_syn_backlog = 65535
```

4. Rounded to the *NEXT* power of two

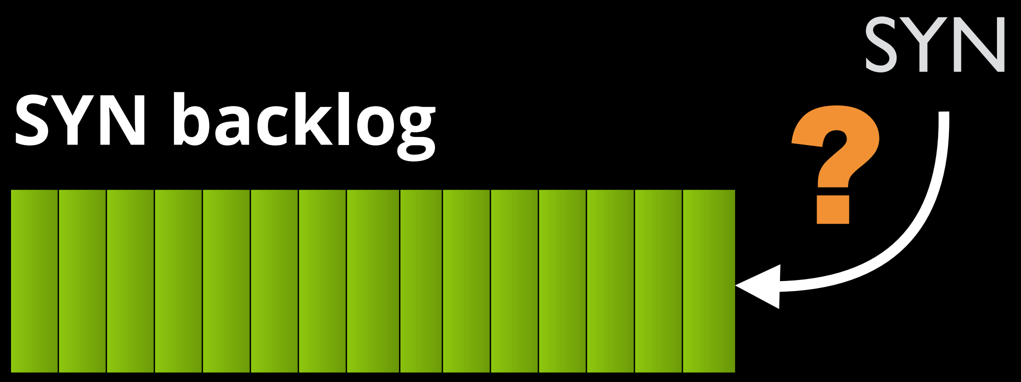
127 --> 128 128 --> 256

CLOUDFLARE

SYN backlog churn

```
sysctl -w net.ipv4.tcp_synack_retries=1
```

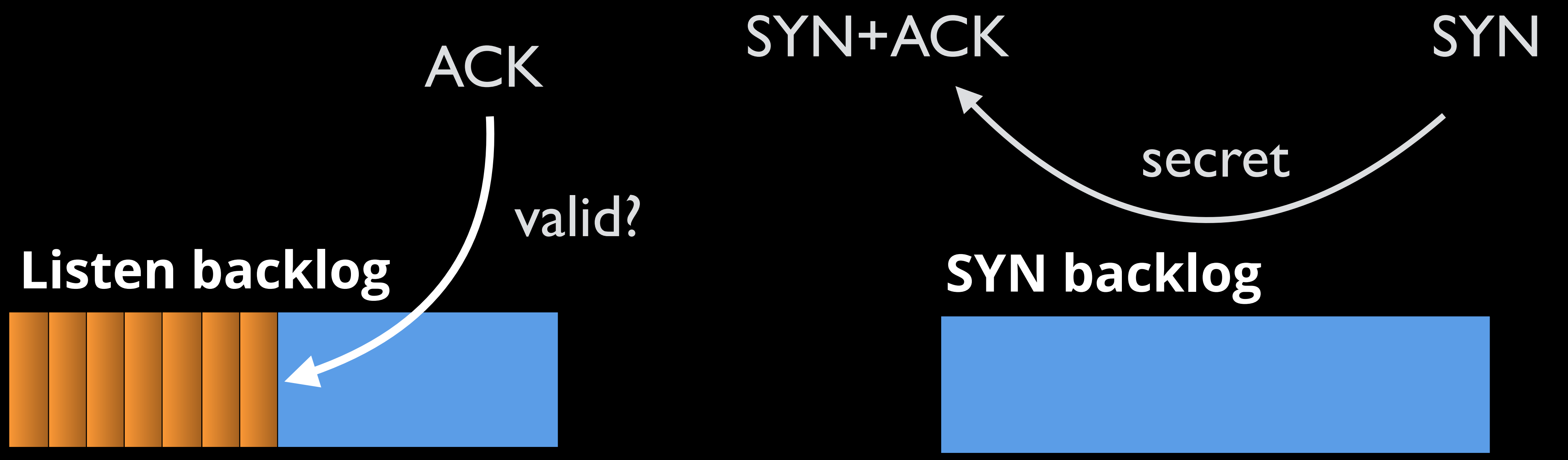
SYN backlog overflow



- Normal case - DROP
- Fixed with SYN cookies

```
TCP: Possible SYN flooding on port 80
```

SYN cookies



sequence number:

5 bits	3 bits	24 bits
t mod 32	MSS	hash(ip, port, t)

Tip: TCP timestamps

```
sysctl -w net.ipv4.tcp_timestamps=1
```

timestamp:

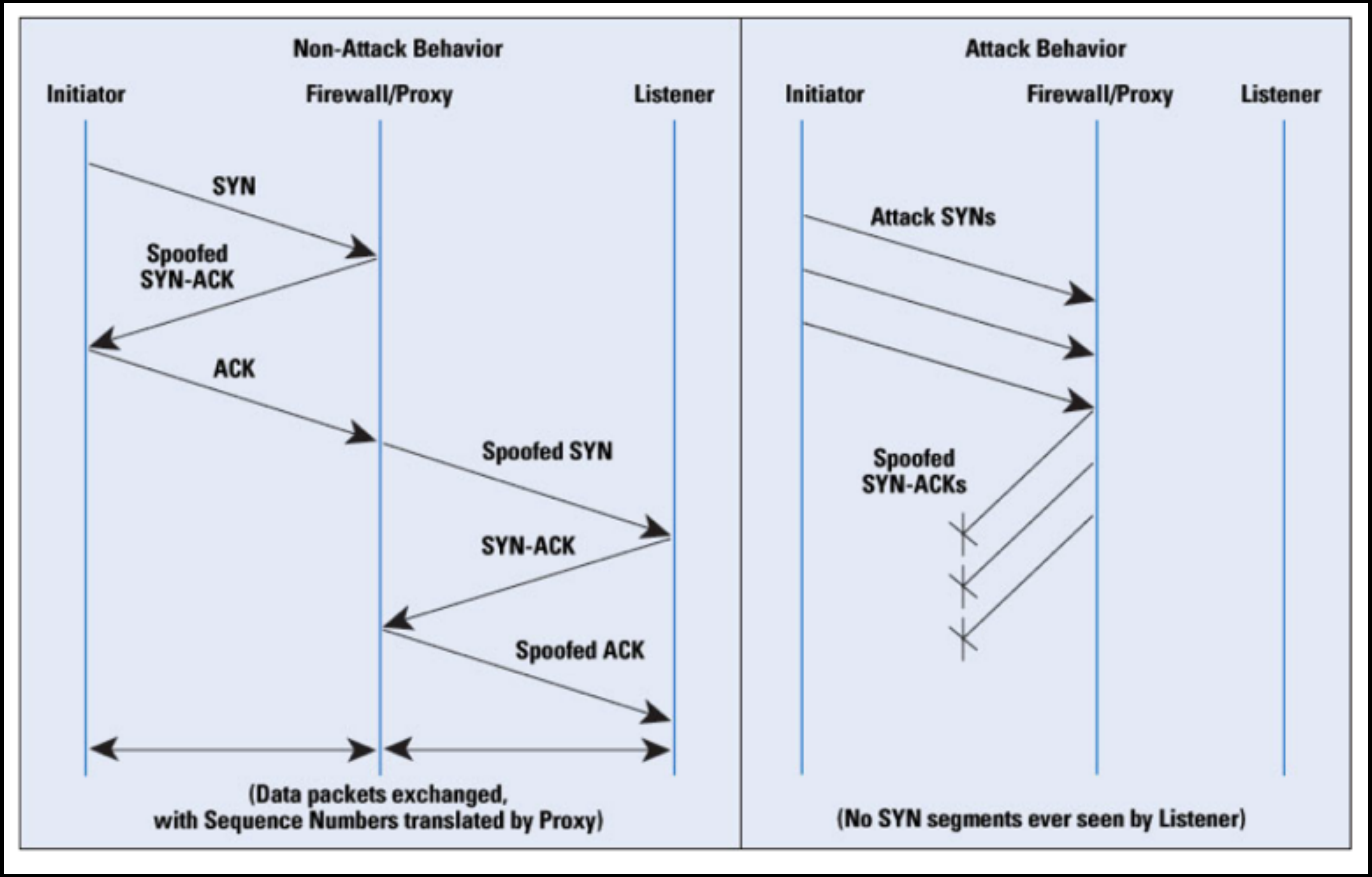
26 bits timestamp	1 bit ECN	1 bit SACK	4 bits wscale

sequence number:

5 bits t mod 32	3 bits MSS	24 bits hash(ip, port, t)

0.3M pps

Fight for the lock



(source: Jesper Brouer presentation)

Recent changes

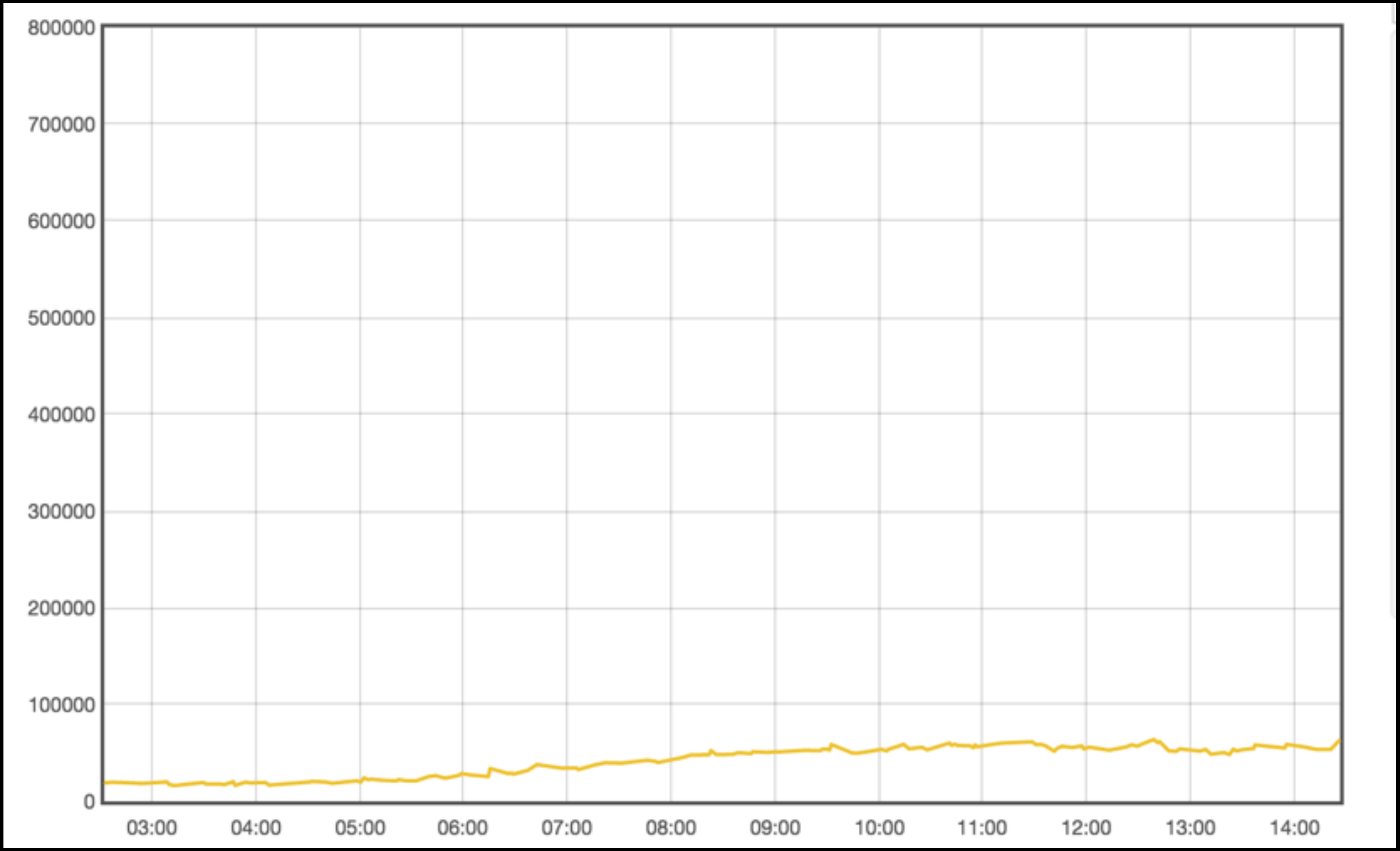
- The idea is to remove the LISTEN lock
  - Heavy refactoring of the SYN queue
- Submitted by Eric Dumazet in early October 2015
- Merged to net-next, will land in 4.4

Real connections from a botnet

Real TCP/IP connections

Looks like this

Packets per second



Symptoms

- Connection count grows
- "Orphaned" sockets count grow
- "Time waits" growing

```
sysctl -w net.ipv4.tcp_max_orphans=262144
sysctl -w net.ipv4.tcp_orphan_retries=1

sysctl -w net.ipv4.tcp_max_tw_buckets=360000
sysctl -w net.ipv4.tcp_tw_reuse=1
sysctl -w net.ipv4.tcp_fin_timeout=5
```

IP reputation

Iptables to the rescue

- Ipset
  - Supports subnets
  - Manual blacklisting
- Hashlimits
  - Rate limit packets per subnet
  - Automatic blacklisting with timeout
- Connlimit
- TARPIT

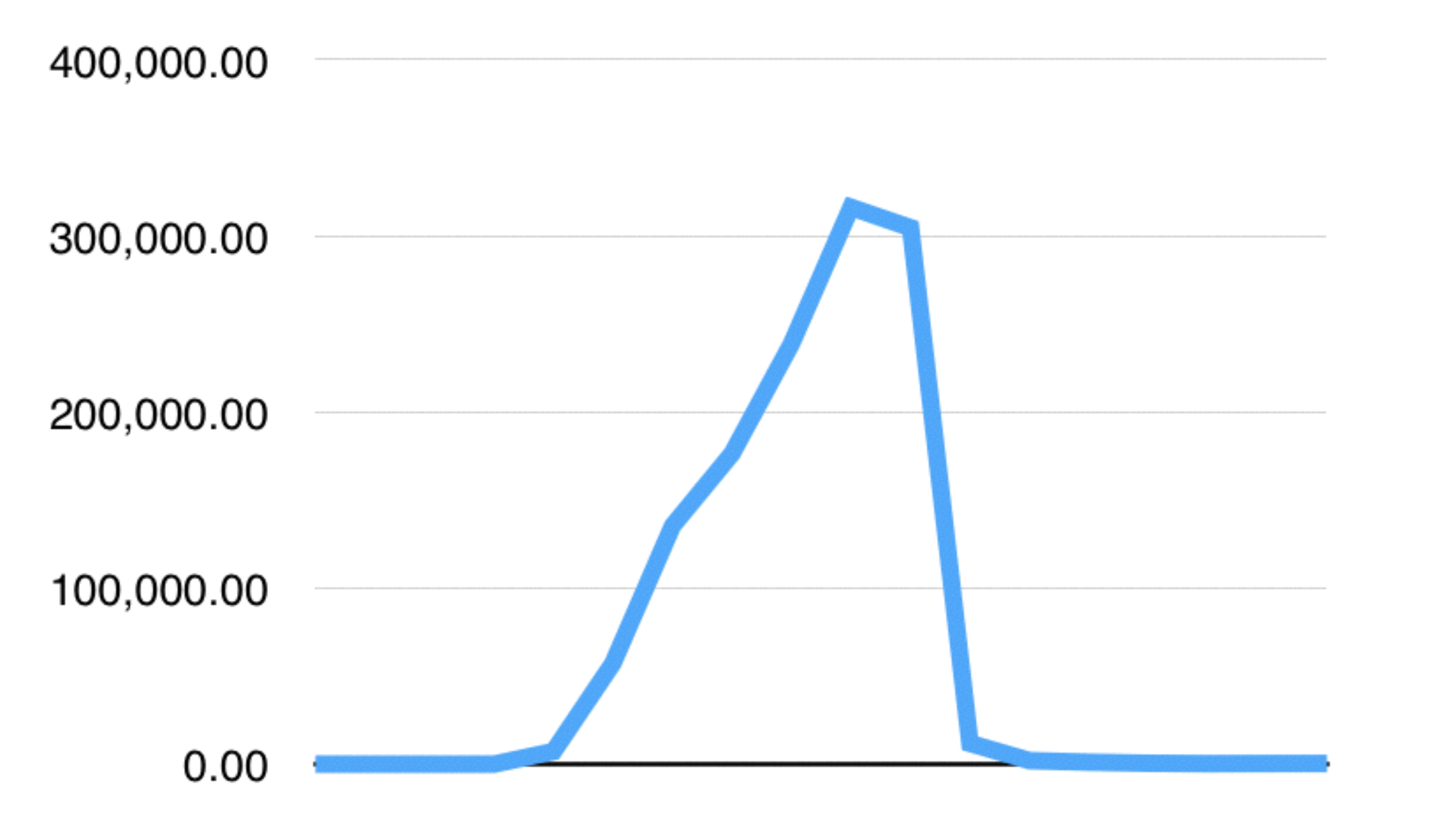
Note on large botnets

- Make it a SYN flood
  - Disable HTTP keep-alives
- Blacklist IP's based on payload
  - Typo in request
  - BPF or string module for match + ipsets auto expiry

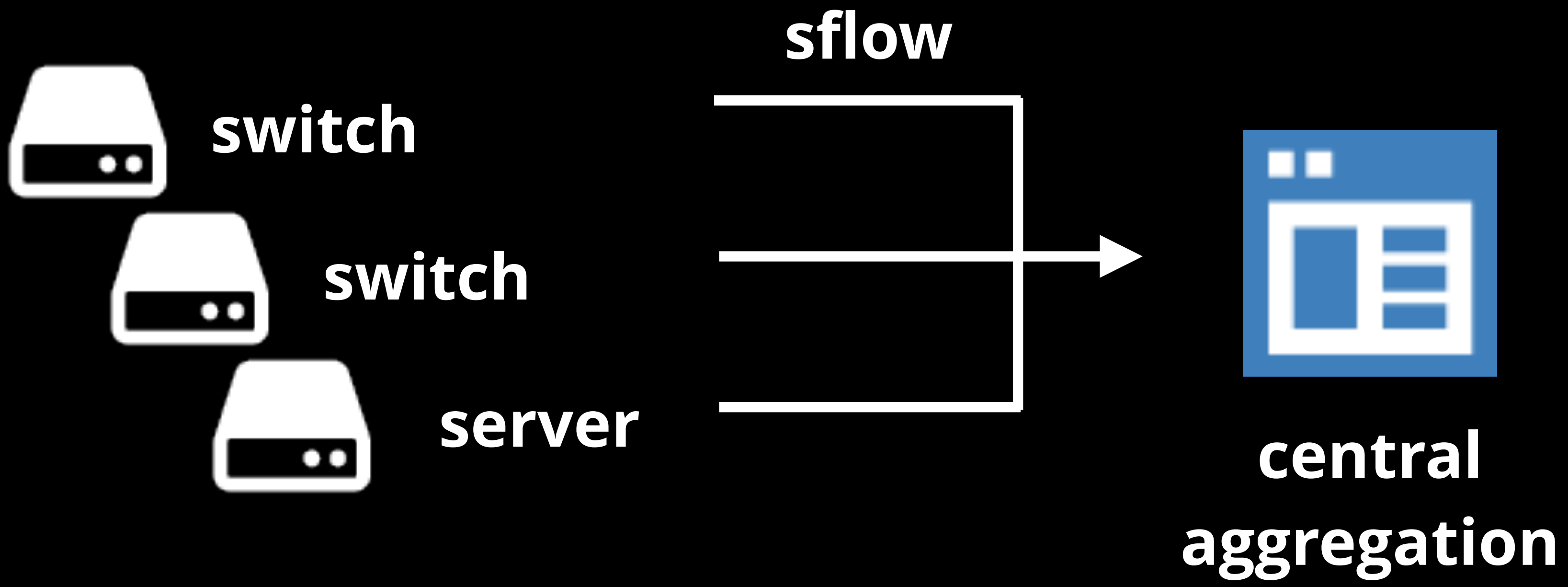
Note on large botnets

```
GET /forum.php HTTP/1.1
Accept: */*
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; Baiduspider/2.0;...
Host: www.example.com:80
Connection: Keep-Alive
```

310k RPS, 650k uniques



Tip: sflow



Centralized Sflow

```
$ tailsflow -i sflow | tcpdump -n -r - -c 10 'vlan and ip'
reading from file -, link-type EN10MB (Ethernet)
IP 10.11.8.17.8070 > 10.11.8.82.24982:
IP 10.16.8.95.8070 > 10.16.10.139.33176: 18:55:22.345369
IP 70.215.131.237.3232 > 104.16.19.35.80: 18:55:22.345371
IP 162.222.178.71.35563 > 173.245.58.146.53:
IP 199.71.213.20.40150 > 173.245.58.146.53: 18:55:22.345430
IP 195.175.255.138.62803 > 173.245.58.221.53:
IP 220.213.193.137.52163 > 104.31.188.8.80:
IP 10.40.8.97.8070 > 10.40.8.59.46943:
IP 115.231.91.118.35120 > 173.245.58.146.53:
IP 10.12.11.5.8070 > 10.12.8.106.24514:
```

Takeaways

- You *WILL* null-route
- Linux firewall is awesome
- Sflow for detection

Thanks

and good luck!

(please fill the speaker excitement form!)

marek@cloudflare.com @majek04

CLOUDFLARE