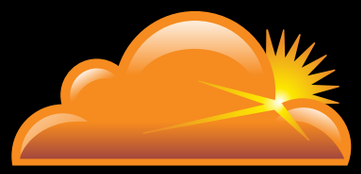


CLOUDFLARE®

Lessons from defending the indefensible

Marek Majkowski

marek@cloudflare.com @majek04

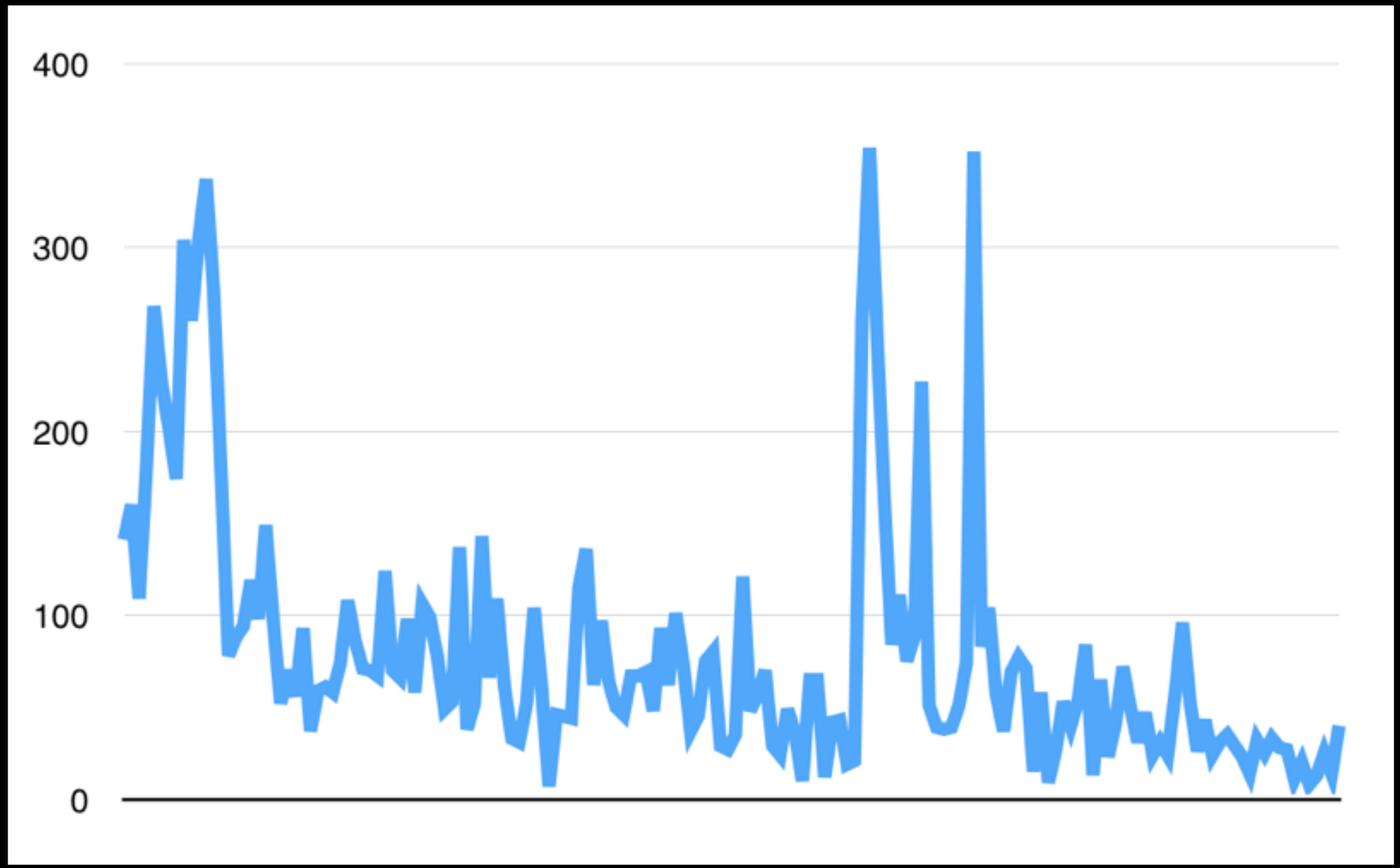
Denial of service (DoS)

(source: the internet)

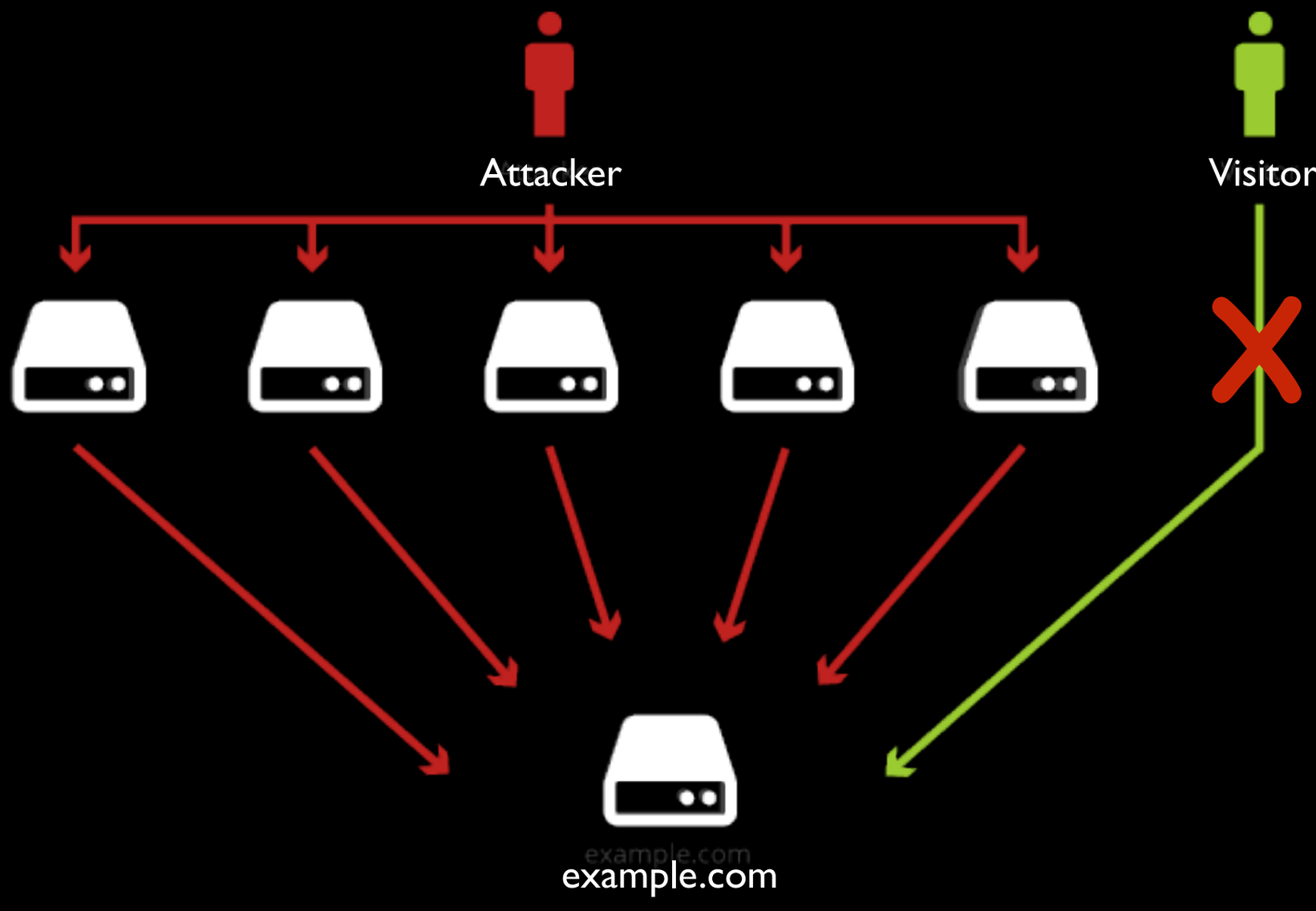
Unique view


Seattle
San Jose
Los Angeles
Chicago
Dallas
Toronto
Newark
Ashburn
Atlanta
Miami
Medellín
Lima
São Paulo
Valparaíso
Buenos Aires
Düsseldorf
Amsterdam
Manchester
Dublin
London
Paris
Madrid
Frankfurt
Stockholm
Warsaw
Prague
Vienna
Bucharest
Berlin
Milan
Marseille
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

DoS attempts daily

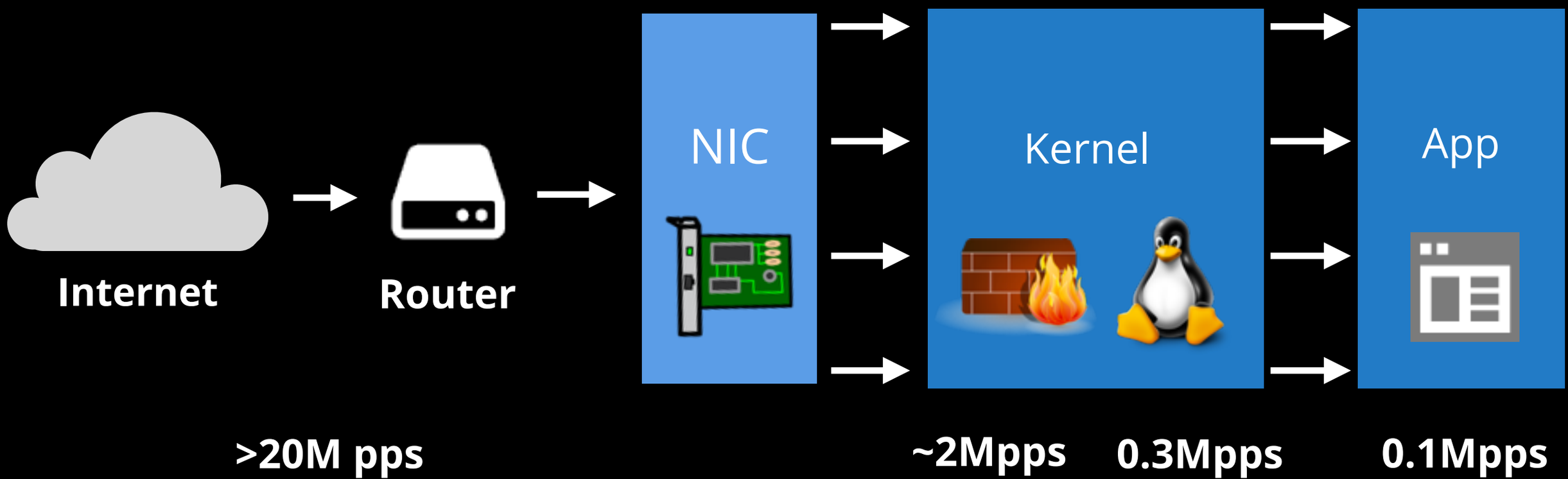


Defending from DoS is hard

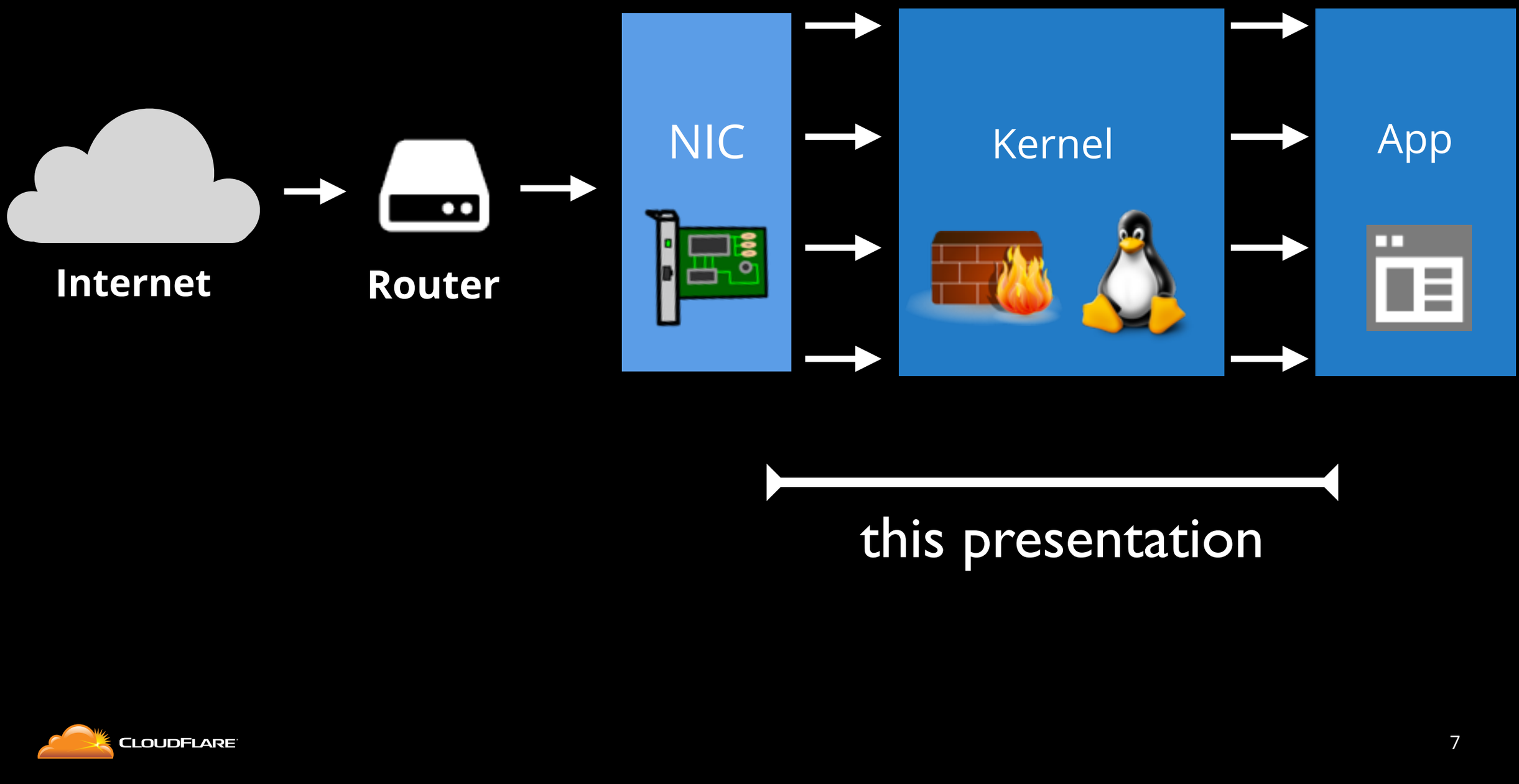


example.com

Attack surface



Attack surface



Agenda

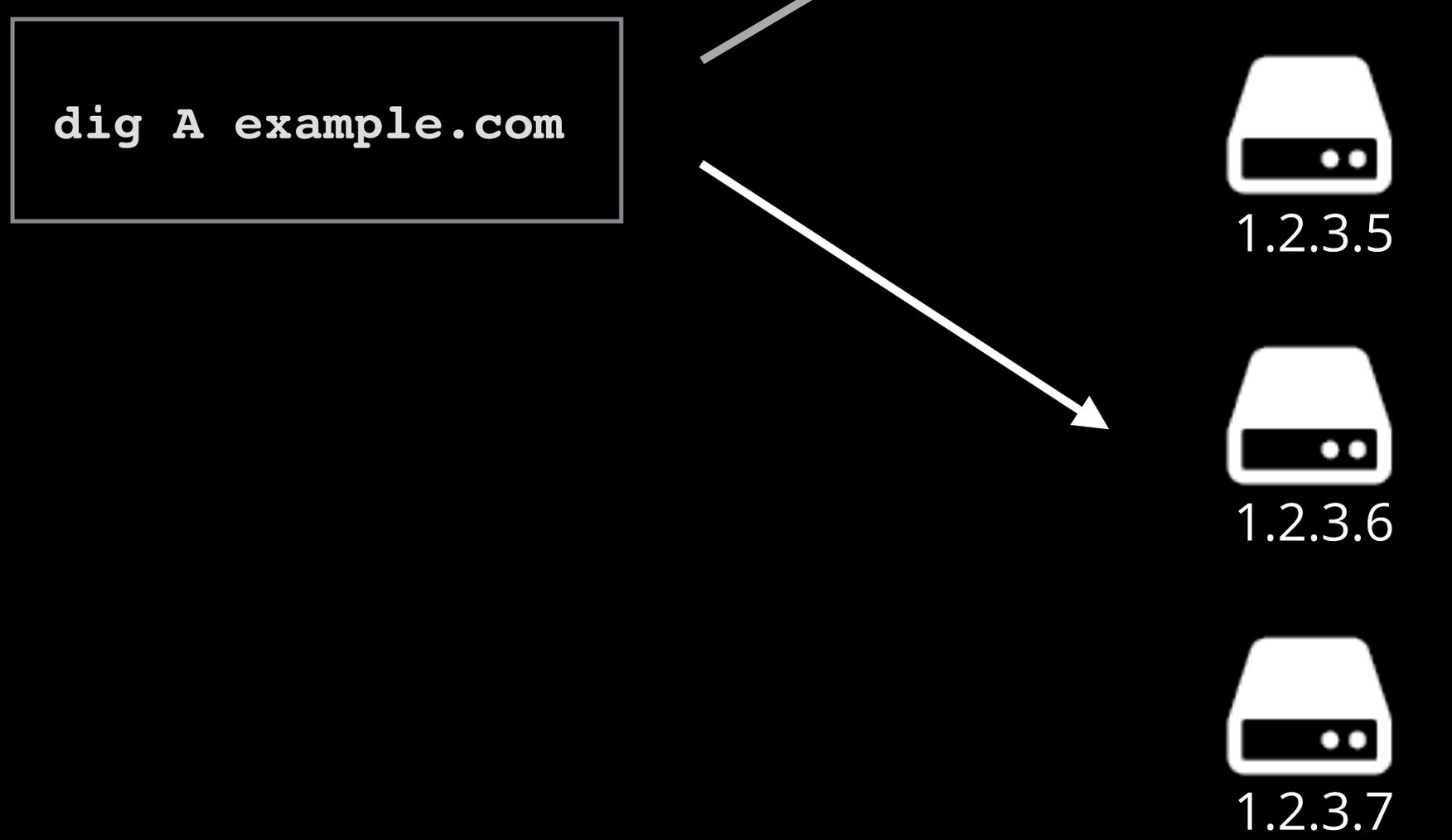
1. Network congestion
2. L3 - High volume packet floods
3. L4 - Packet floods against TCP stack
4. L7 - Botnets
5. L7+ - Very large botnets

Network congestion

BGP null routing

```
route 1.2.3.4/32 {
    discard;
    community [ 13335:666 13335:668 13335:36006 ];
}
```

Application integration

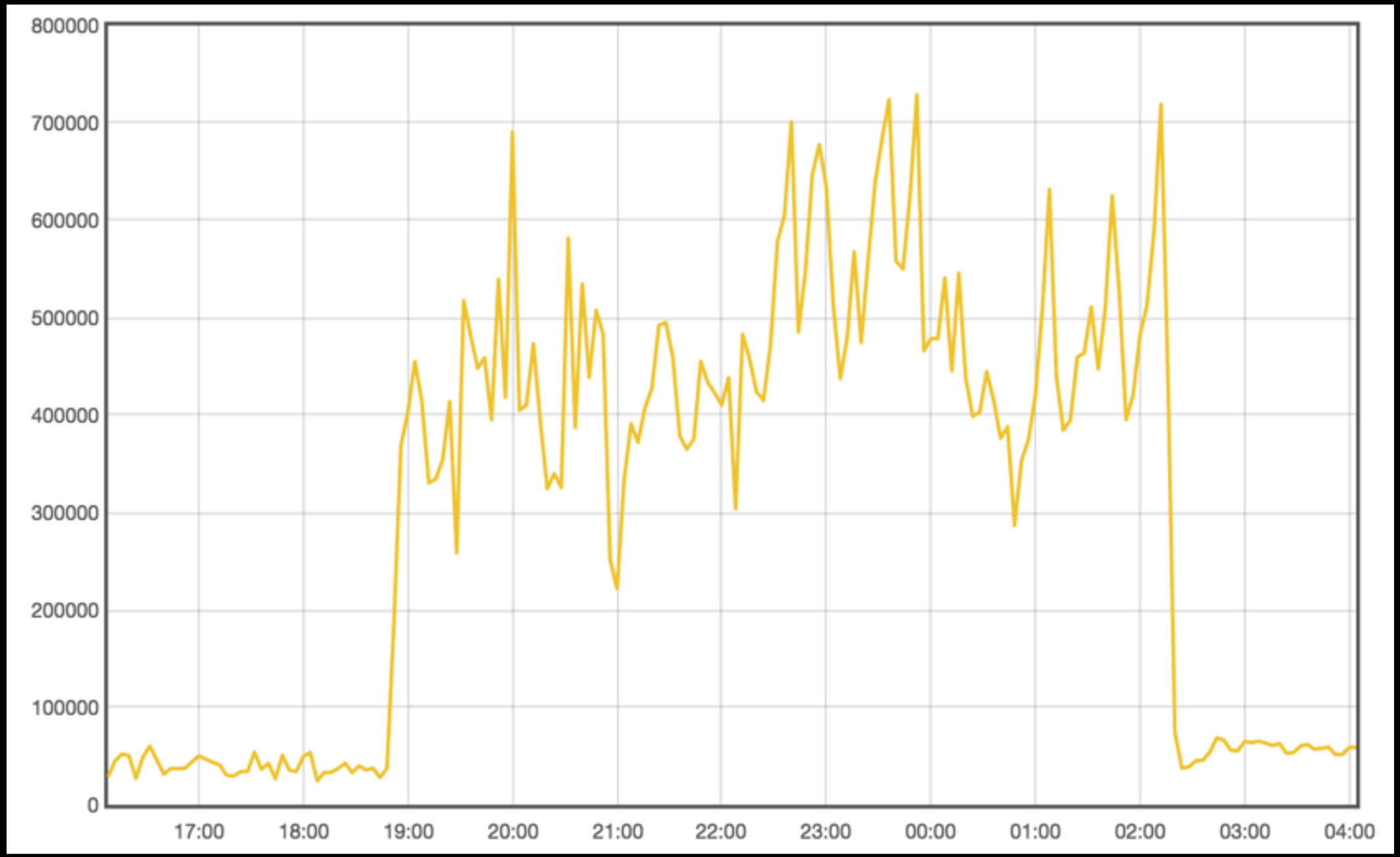


High volume packet floods (L3)

Let it flow

(source: Yogendra Joshi)

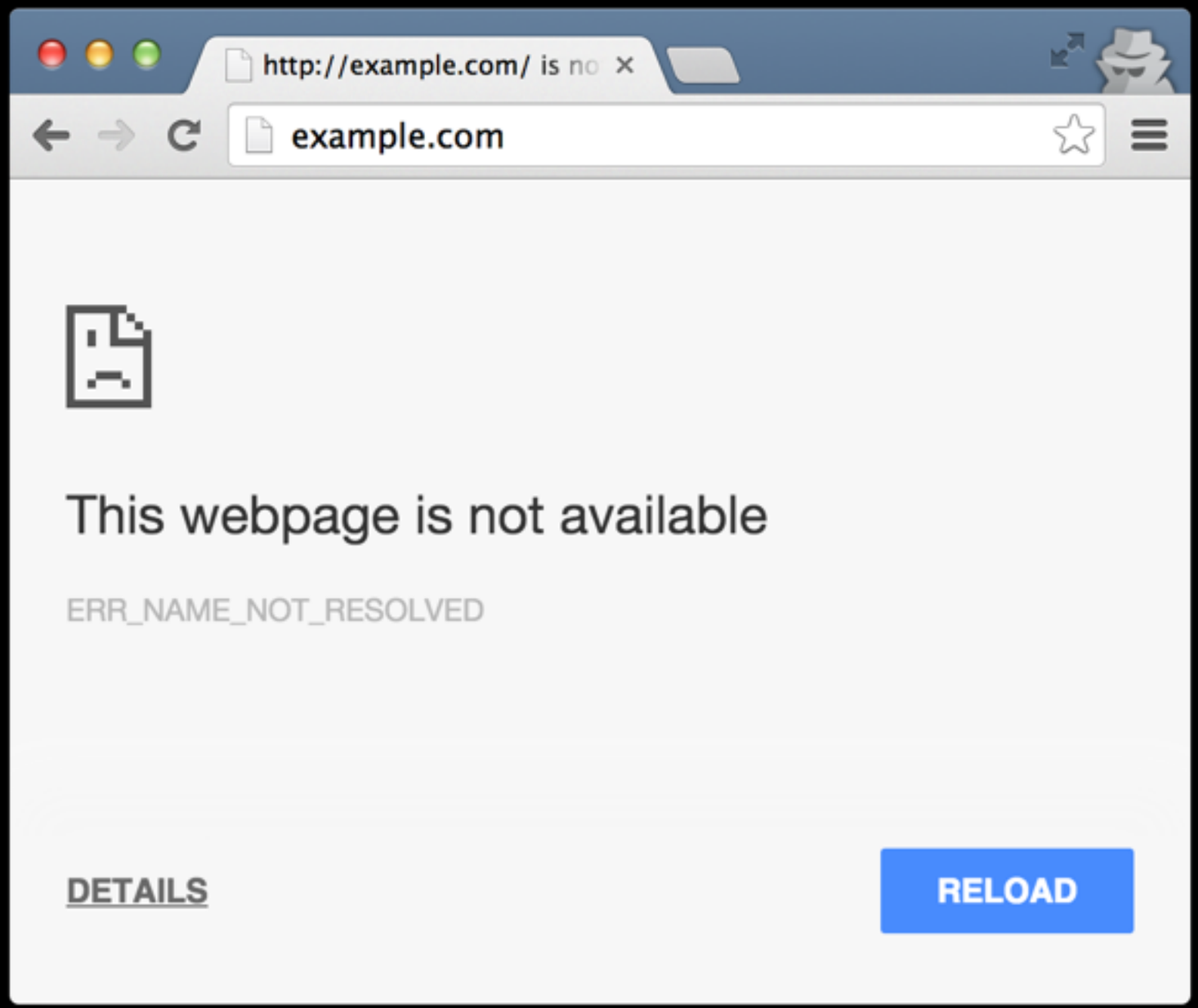
High volume packet flood



UDP DNS flood

```
IP 202.194.181.95.15443 > 1.2.3.4:53: 63476% [1au] A? example.com. (50)
IP 221.12.236.115.6570 > 1.2.3.4:53: 11406% [1au] A? example.com. (50)
IP 203.94.134.43.18473 > 1.2.3.4:53: 8559% [1au] A? example.com. (50)
IP 203.196.66.75.32573 > 1.2.3.4:53: 47971% [1au] A? example.com. (50)
IP 124.240.198.136.23336 > 1.2.3.4:53: 61152% [1au] A? example.com. (50)
IP 218.247.70.185.11679 > 1.2.3.4:53: 16360% [1au] A? example.com. (50)
IP 202.109.218.98.27549 > 1.2.3.4:53: 17829% [1au] A? example.com. (50)
IP 203.148.240.82.21825 > 1.2.3.4:53: 22590% [1au] A? example.com. (50)
IP 211.167.108.67.25782 > 1.2.3.4:53: 17663% [1au] A? example.com. (50)
IP 203.209.60.18.20221 > 1.2.3.4:53: 38257% [1au] A? example.com. (50)
IP 203.81.181.168.12749 > 1.2.3.4:53: 53492% [1au] A? example.com. (50)
```

Sad DNS server



Spoofed?

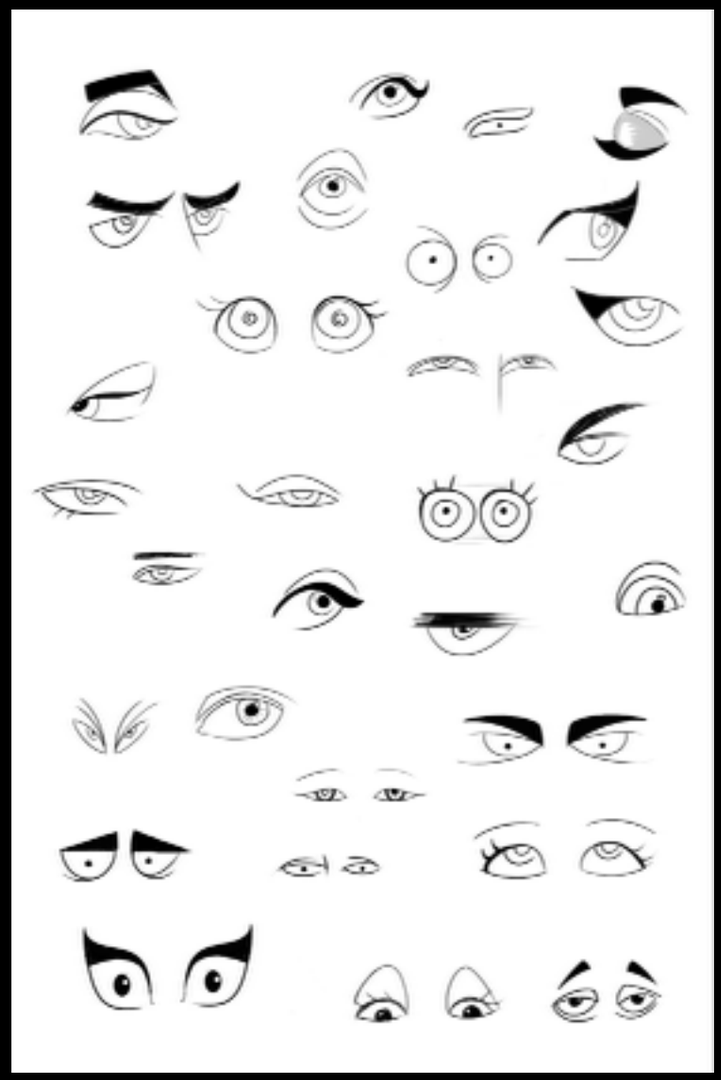
(source: DaPuglet)

Drop!

1 in 10K packets

Packet characteristics

- Packet length
- Payload
- Goal: limit false positives



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

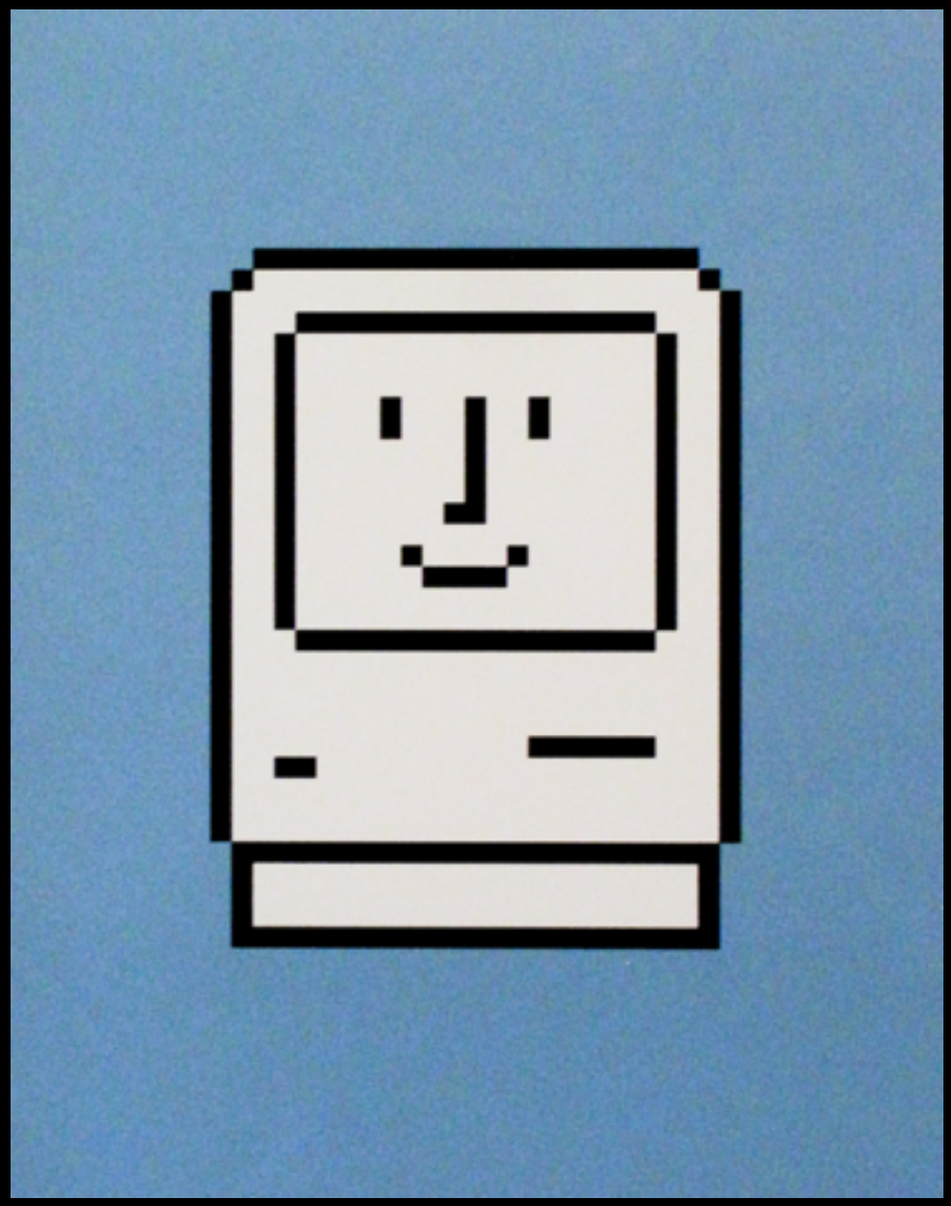
- Originally: `tcpdump -n “udp and port 53”`
- xt_bpf implemented in 2013 by Willem de Bruijn
- Tcpdump expressions are limited - no variables
- Benefits in hand-crafting BPF

BPF tools

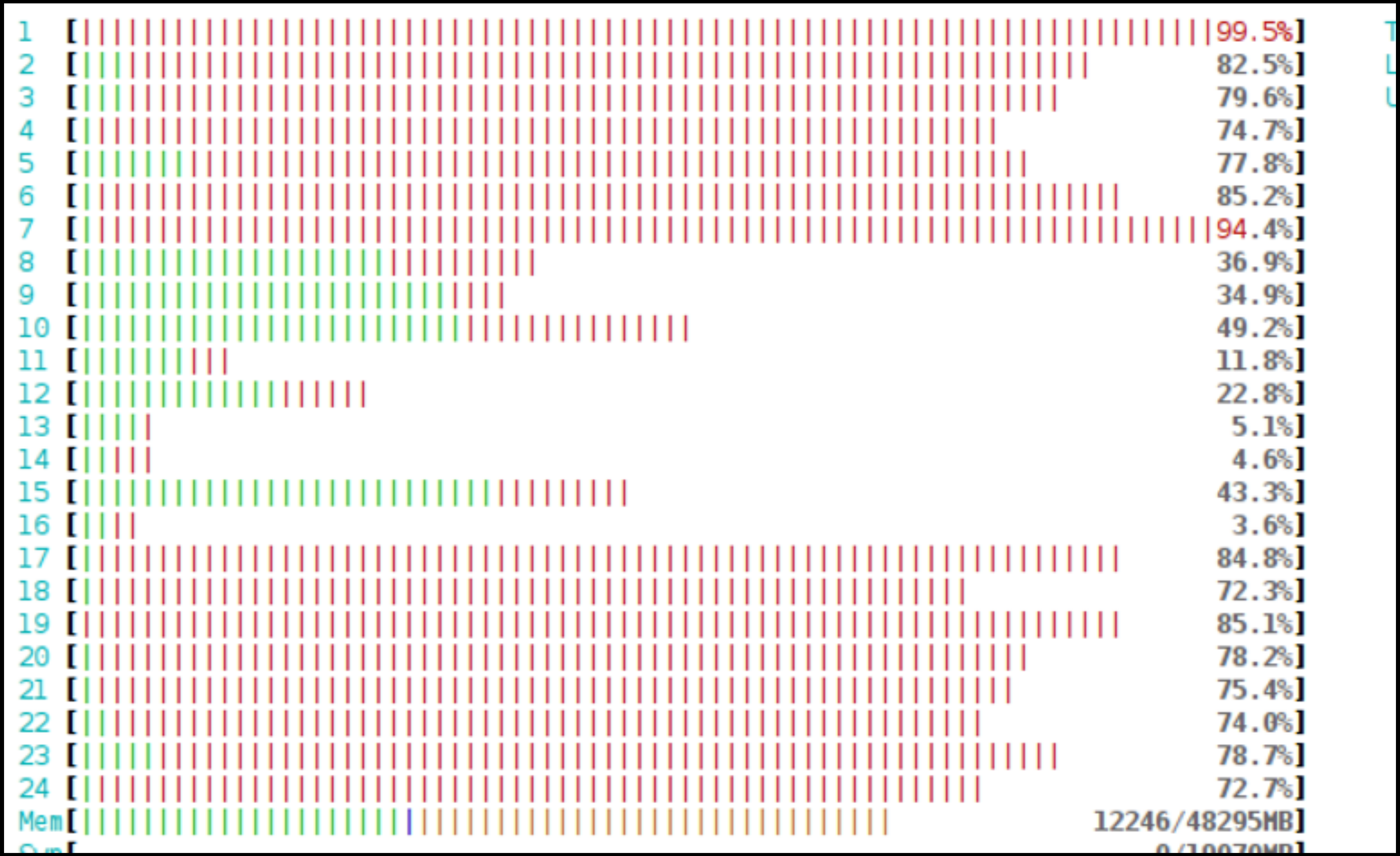
- Open source:
  - https://github.com/cloudflare/bpftools
- Can match various DNS patterns:
  - `*.example.com`
  - `--case-insensitive *.example.com`
  - `--invalid-dns`

~2M pps

Happy DNS server

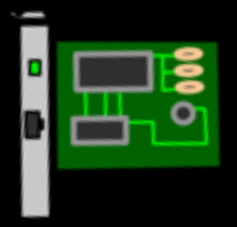


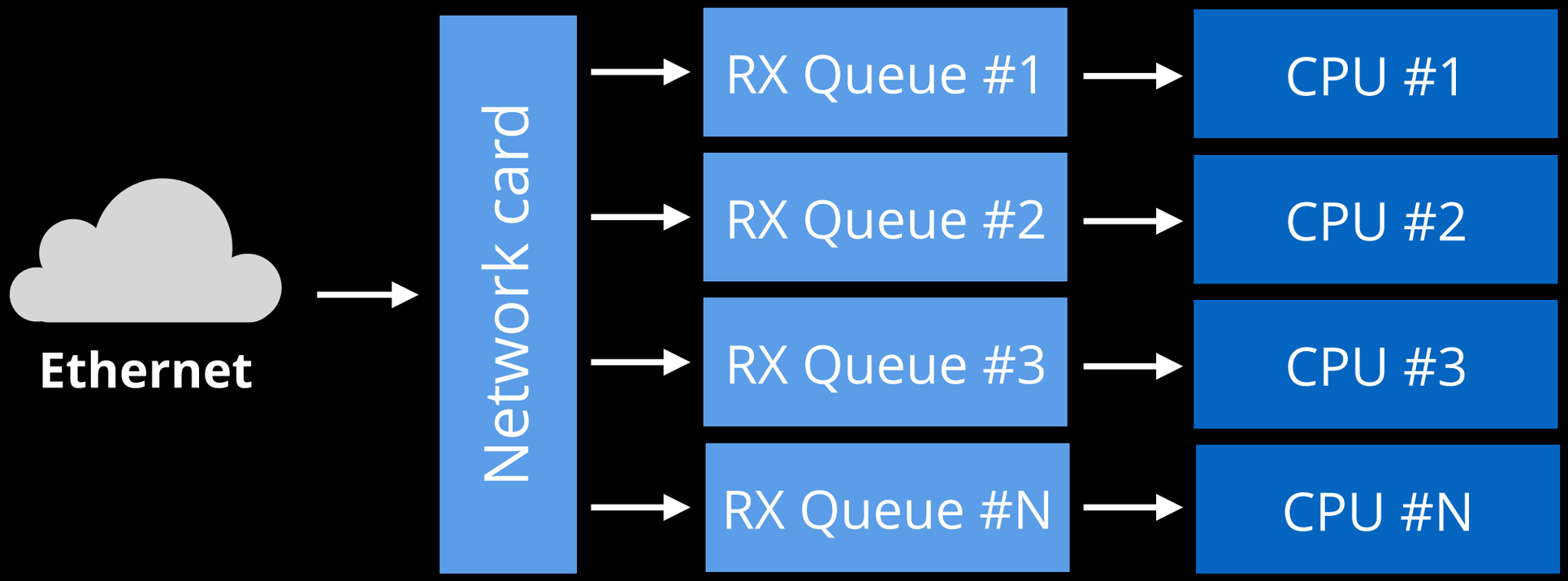
Sad OS - interrupt storms



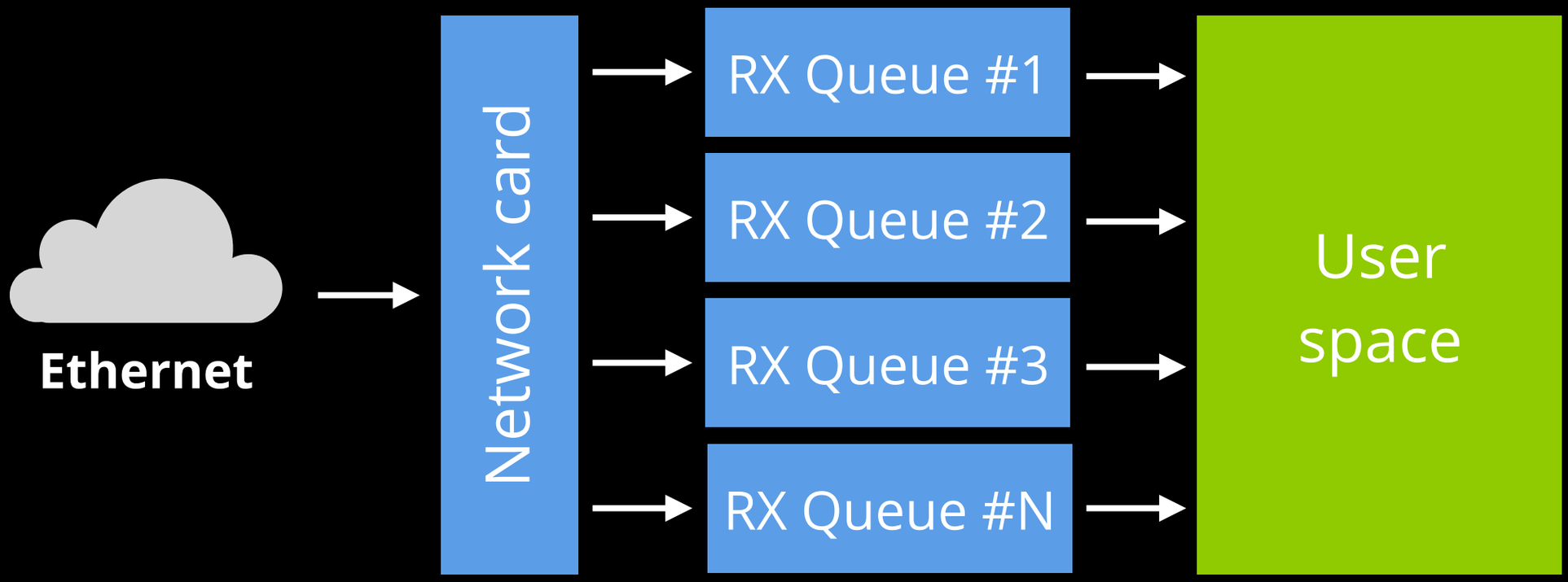
Payload matching close to NIC

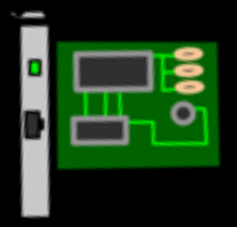
Modern NIC's




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

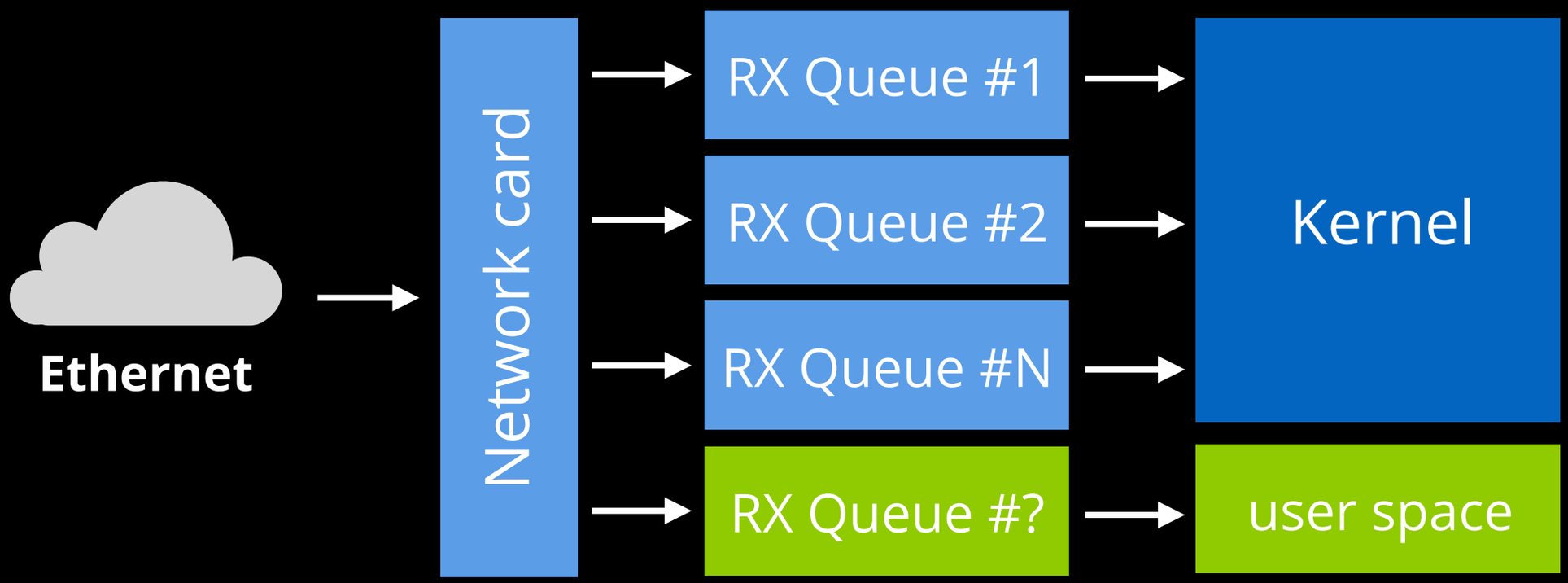
Traditional kernel bypass

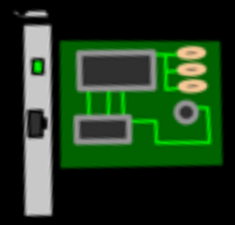




Partial kernel bypass

aka bifurcated driver

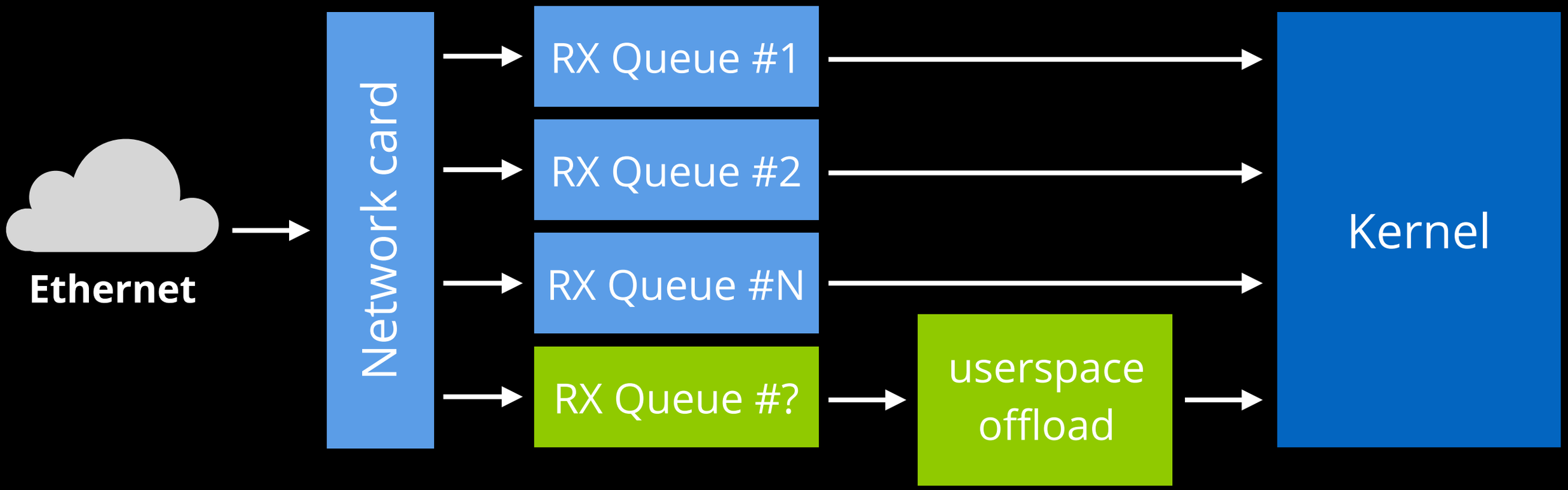




Partial kernel bypass

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

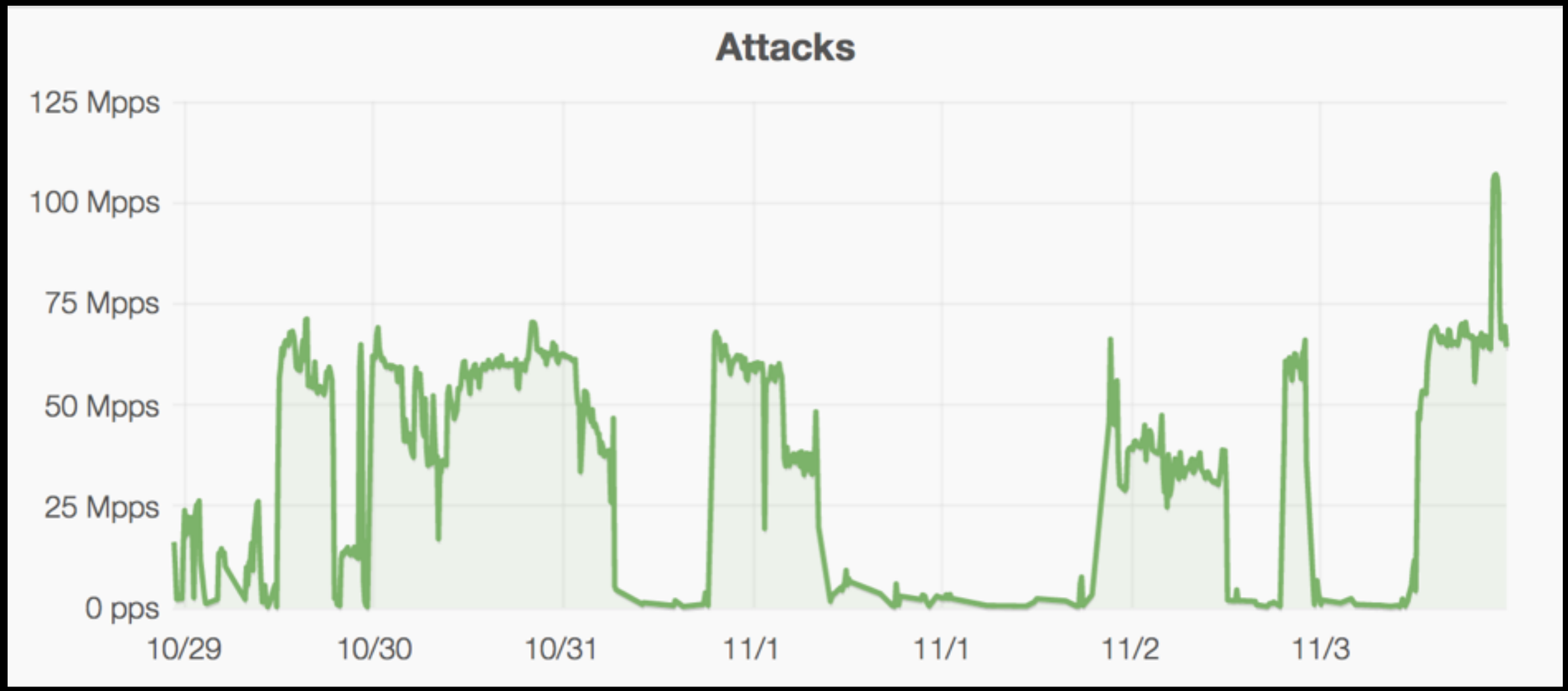
Iptables offload



It works really well

>3M pps

It works really well



No characteristics: Attacks against TCP/IP network stack (L4)

ACK floods

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

~0.3M pps

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
```

~2M pps

Effective against TCP attacks

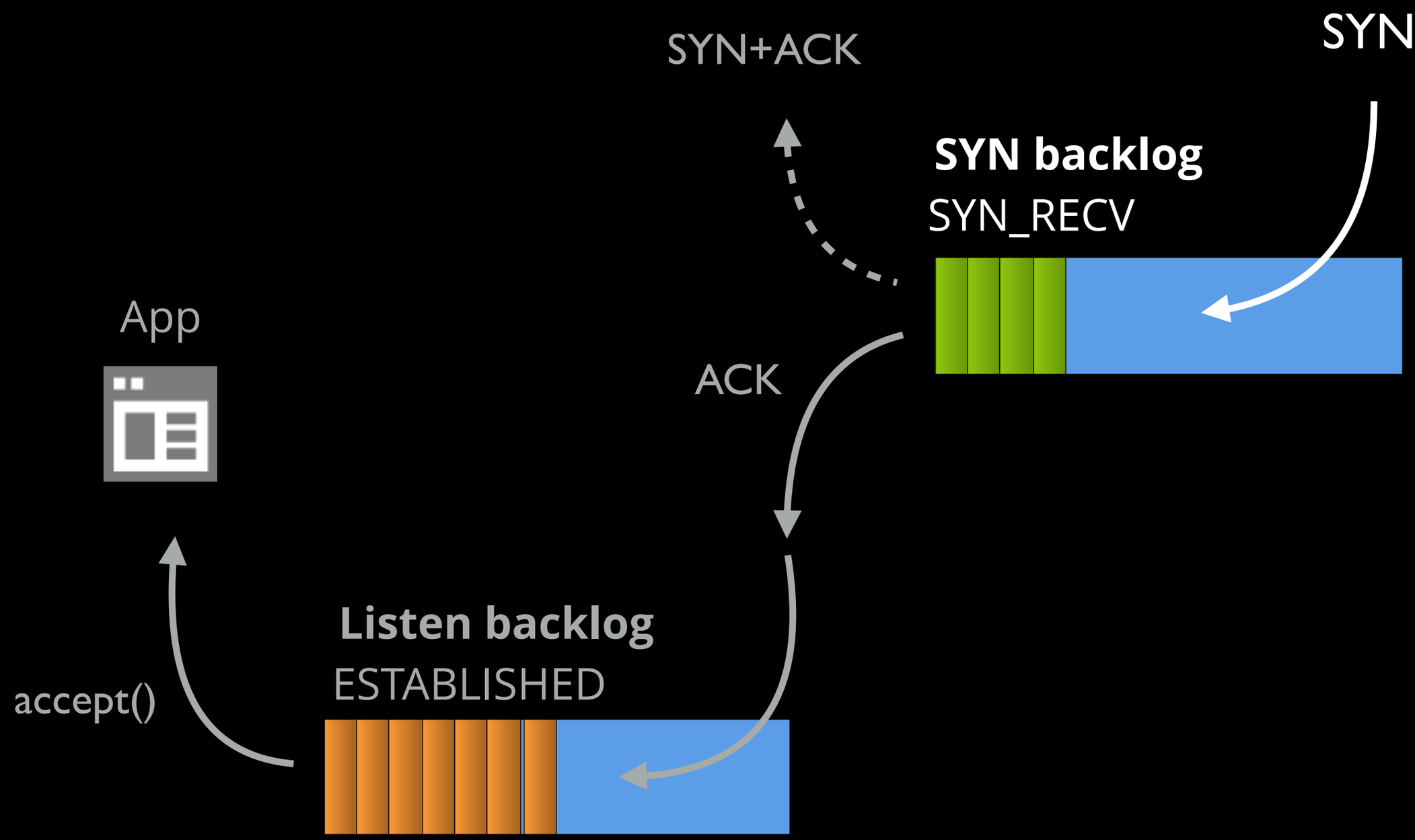
- Works well against:
  - ACK
  - FIN
  - RST
  - X-mas
- What about SYN floods?

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

SYN Cookies

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

sequence number:

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

timestamp:

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

~0.3M pps

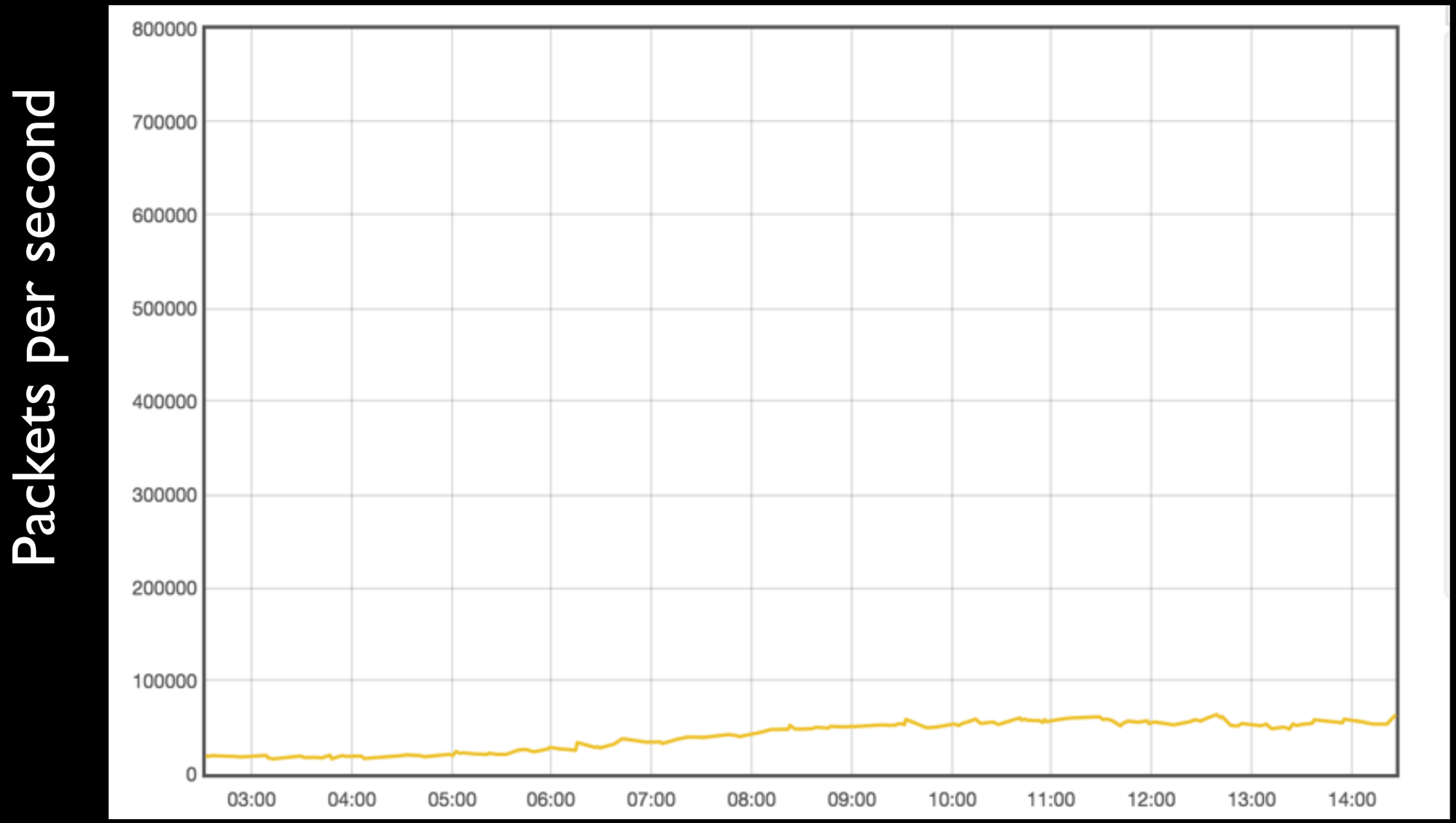
Recent changes

- 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

Connections from a botnet (L7)

Real TCP/IP connections

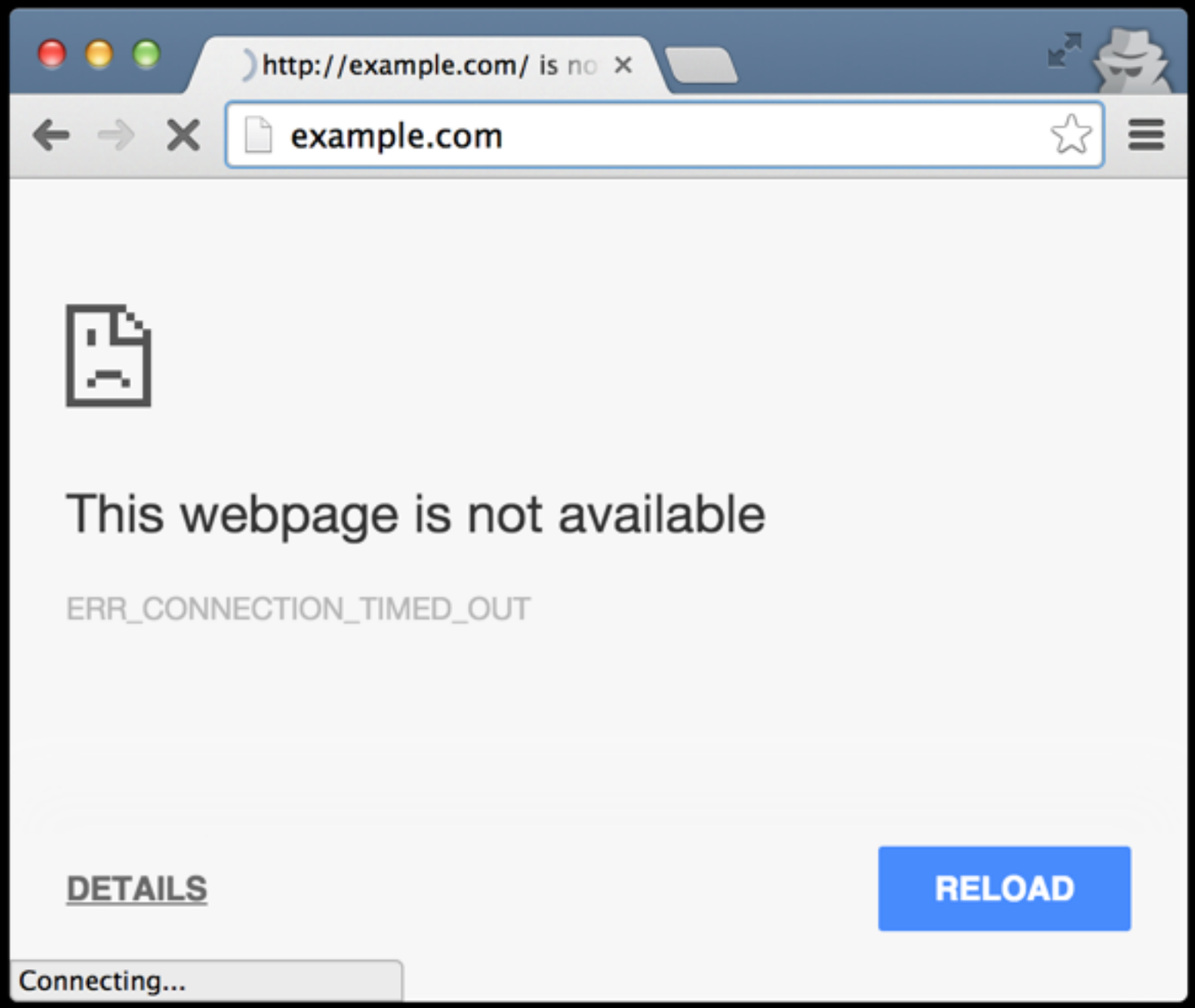
Small volume



Symptoms

- Concurrent connection count going up
- Many sockets in "orphaned" state
- "Time waits" socket state indicates churn

Sad HTTP server



IP reputation

(source: the internet)

Reputation in iptables

1. Conntrack Connlimit
2. Hashlimits
   - Rate limit SYN packets per IP
3. Ipset
   - Manual blacklisting - feed IP blacklist from HTTP server logs
   - Supports subnets, timeouts
   - Automatic blacklisting hashlimits

Make it a SYN flood

```
GET / HTTP/1.1
Host: www.example.com

GET / HTTP/1.1
Host: www.example.com

GET / HTTP/1.1
Host: www.example.com
...
```

- Disable HTTP keep-alives
  - Make it a SYN flood

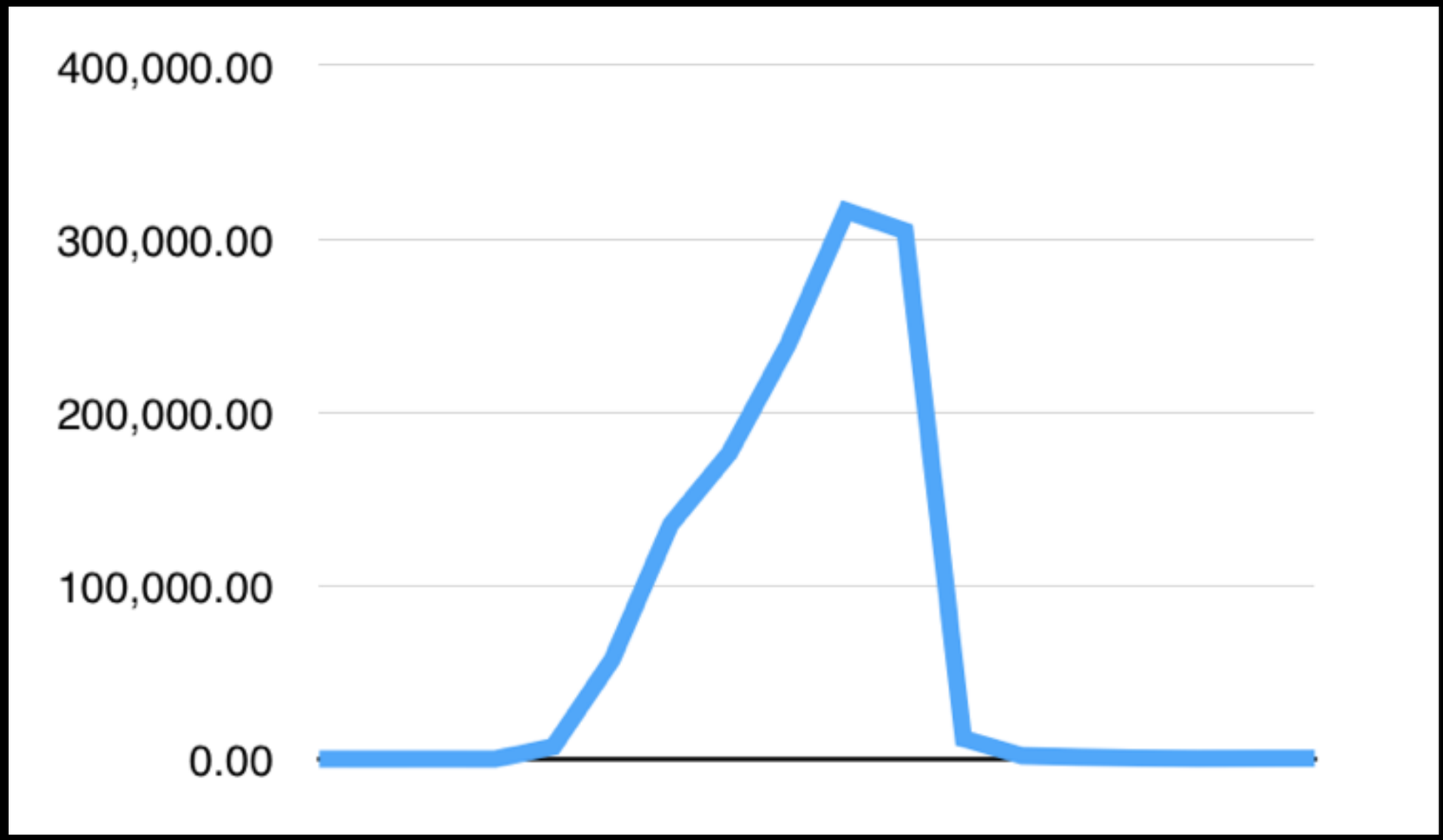
Very large botnets (L7+)

Very large botnets

- Blacklist IP's based on payload
  - "BPF" or "string" module for match + ipsets auto expiry

```
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
```

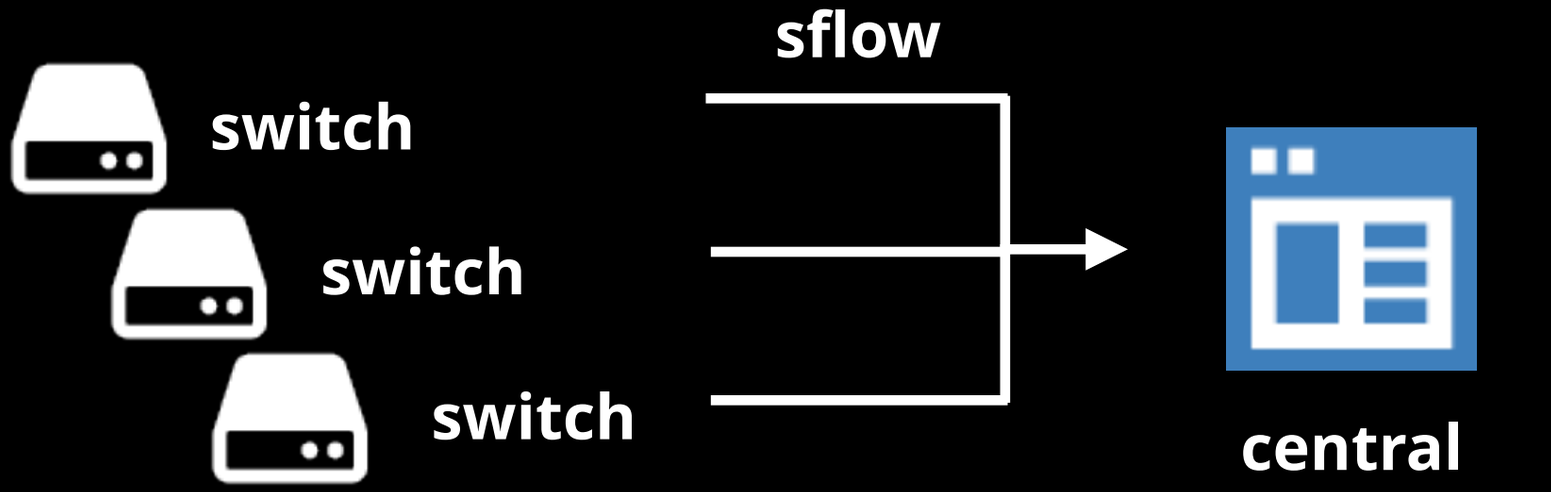
300k RPS, 650k uniques



(source: CloudFlare blog)

Detection

Sflow for real time analytics



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:
```

Host-sflowd

```
sflow {
  DNSSD = off
  collector {
    ip = 4.3.2.1
    udpport = 6343
  }
  nflogProbability = 0.00048828125
  nflogGroup = 33
  polling = 300
}
```

```
iptables -I INPUT \
  -m statistic \
  --mode random --probability 0.00048828125 \
  -j NFLOG --nflog-group 33

hsflowd -d -f hsflowd.conf -o /var/run/hsflowd.auto -
p /var/run/hsflowd.pid
```

Thanks!

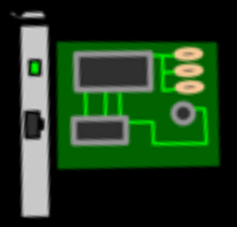
- You *WILL* BGP null-route
  - Prepare your application for that
- DROP all the packets! (only 1 in 10k could be valid!)
  - With BPF
  - Partial kernel bypass for better speed
- Iptables are powerful
  - Connlimit, hashlimits, ipsets

(please fill the attendee excitement form!)

marek@cloudflare.com @majek04

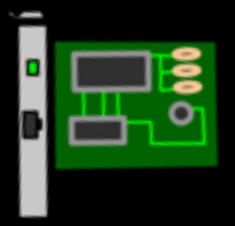
Appendix A

Exciting system tweaks



NIC: discard with flow steering

```
ethtool -N eth3 flow-type udp4 \
    dst-ip 192.168.254.30 \
    dst-port 53 action -1
```



Tip: Flow steering for priority

```
ethtool -X eth3 weight 0 1 1 1 1 1 1 1 1 1 1
ethtool -N eth3 flow-type tcp4 \
    dst-port 22 action 0
```

SYN backlog size

1. Listen backlog size

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

2. Capped by somaxconn

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

3. SYN backlog capped with

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

4. Rounded to next power of two

127 --> 128 128 -->256

SYN backlog decay

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

L7 connection count

```
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
```

Appendix B

Iptables examples

L3: u32

```
iptables -A INPUT \
 --dst 1.2.3.4 \
 -p udp -m udp --dport 53 \
 -m u32 --u32 "6&0xFF=0x6 && 4&0x1FFF=0 && 0>>22&0x3C@4=0x29"\
 -j DROP
```

L4: Conntrack

```
iptables -t raw -A PREROUTING \
   -i eth2 \
   --dst 1.2.3.4 \
   -j ACCEPT

iptables -t raw -A PREROUTING \
   -i eth2 \
   -j NOTRACK


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

Tuning conntrack

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

```
sysctl -w net.netfilter.nf_conntrack_helper=0

sysctl -w net.nf_conntrack_max=2000000
echo 2500000 > /sys/module/nf_conntrack/parameters/hashsize
```

L7: Connlimit

```
iptables -t raw -A PREROUTING \
  -i eth2 \
  --dst 1.2.3.4 \
  -j ACCEPT

iptables -A INPUT \
  --dst 1.2.3.4 \
  -p tcp -m tcp --dport 80 \
  -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN \
  -m connlimit \
  --connlimit-above 10 \
  --connlimit-mask 32 \
  --connlimit-saddr \
  -j DROP
```

L7: ipset for blacklisting

```
ipset -exist create ta_d335c5 hash:net family inet

ipset add ta_d335c5 192.168.0.0/16
ipset add ta_d335c5 10.0.0/8

iptables -A INPUT \
  -m set --match-set ta_d335c5 src \
  -j DROP
```

L7: being evil - TARPIT

```
iptables -A INPUT \
  -m set --match-set ta_d335c5 src \
  -j TARPIT
```

L7: hashlimit for rate limiting

```
iptables -A INPUT \
  --dst 1.2.3.4 -p tcp -m tcp --dport 80\
  --tcp-flags FIN,SYN,RST,PSH,ACK,URG SYN \
  -m hashlimit \
  --hashlimit-above 123/sec \
  --hashlimit-burst 5 \
  --hashlimit-mode srcip \
  --hashlimit-srcmask 24 \
  --hashlimit-name 341654b1d4af9bf \
  -j DROP
```

L7: auto-blacklisting

```
ipset -exist create blacklist hash:net timeout 60

iptables -A INPUT \
  --dst 1.2.3.4 \
  -m set --match-set blacklist src \
  -j DROP

iptables -A INPUT \
  --dst 1.2.3.4 -p tcp -m tcp --dport 80\
  --tcp-flags FIN,SYN,RST,PSH,ACK,URG SYN \
  -m hashlimit \
  --hashlimit-above 100/sec \
  --hashlimit-mode srcip \
  --hashlimit-srcmask 24 \
  --hashlimit-name hl_blacklist \
  -j SET --add-set blacklist src
```

L7+: payload in TCP - string

```
iptables -A INPUT \
 --dst 1.2.3.4 \
 -p tcp --dport 80 \
 -m string \
 --hex-string 486f73743a207777772e787878787878782e... \
--from 231 --to 300 \
-j DROP
```

L7+: payload in TCP - BPF

```
$ ./fixed_offset.py 'Host: www.xxxxxxx.com:80\r\n' 231

ip[231:4] == 0x486f7374 and ip[235:4] == 0x3a207777 and
ip[239:4] == 0x772e7878 and ip[243:4] == 0x78787878 and
ip[247:4] == 0x782e636f and ip[251:4] == 0x6d3a3830 and
ip[255:2] == 0x0d0a
```

(source: fixed_offset.py)