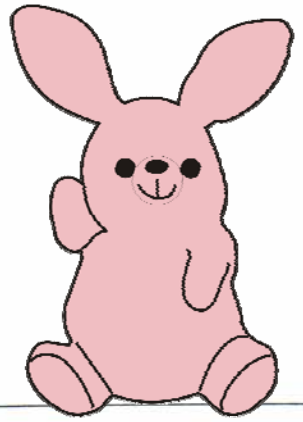
PortBunny
A kernel-based port-scanner



Invent & Verify

A Port Scanner? *Yawn*

- Hey, scanning ports is sports
  - It's not religion
  - It's competitive
- Port scanning is fun for most people
  - Needs random scanning
  - Needs 1337 output
  - Needs 23 different scanning types
- Port scanning is work for some people
  - Needs Accuracy
  - Needs Speed
    - Speed ➔ Time ➔ Money
  - Will use dedicated machines

Why not nmap?

- 3 * 255 Hosts in 30 days with nmap
  - I'm actually coming of age
  - Your scanner is not 1337 if it takes 13:37 per host!
  - No, **--disable-waiting-for-things-that-dont-happen** doesn't cut it
- Professionals don't scan hosts that are …
  … powered off
  … disassembled
  … currently being carried around in the office
- Large scale network scanning is application stocktaking, not vulnerability identification
  - Little interest in the one fully filtered host with only port 23420 open
  - Much interest in how many systems in five Class B networks have port 12345 open

And on a more abstract level...

- All discovery methods depend on a single set of information: the list of open, closed and filtered TCP ports
  - OS Fingerprinting
  - Service probing
  - Banner grabbing
- Accordingly, we need this list first, and quickly at that
- While at it, have actual algorithms

Our Requirements

- TCP SYN Scanning only, no XMAS trees
- No UDP Scanning
  - UDP scanning is a negative scan method
  - Information value of a UDP scan of a properly firewalled host with UDP services is exactly zero
- Constant access to result data
  - Offloading fingerprinting tasks right when results become available
- Design for embedded use
- Engine design with variable front ends
- Bottom line: Do just one thing, but do it right.

Invent & Verify

Hello DefCon ☺

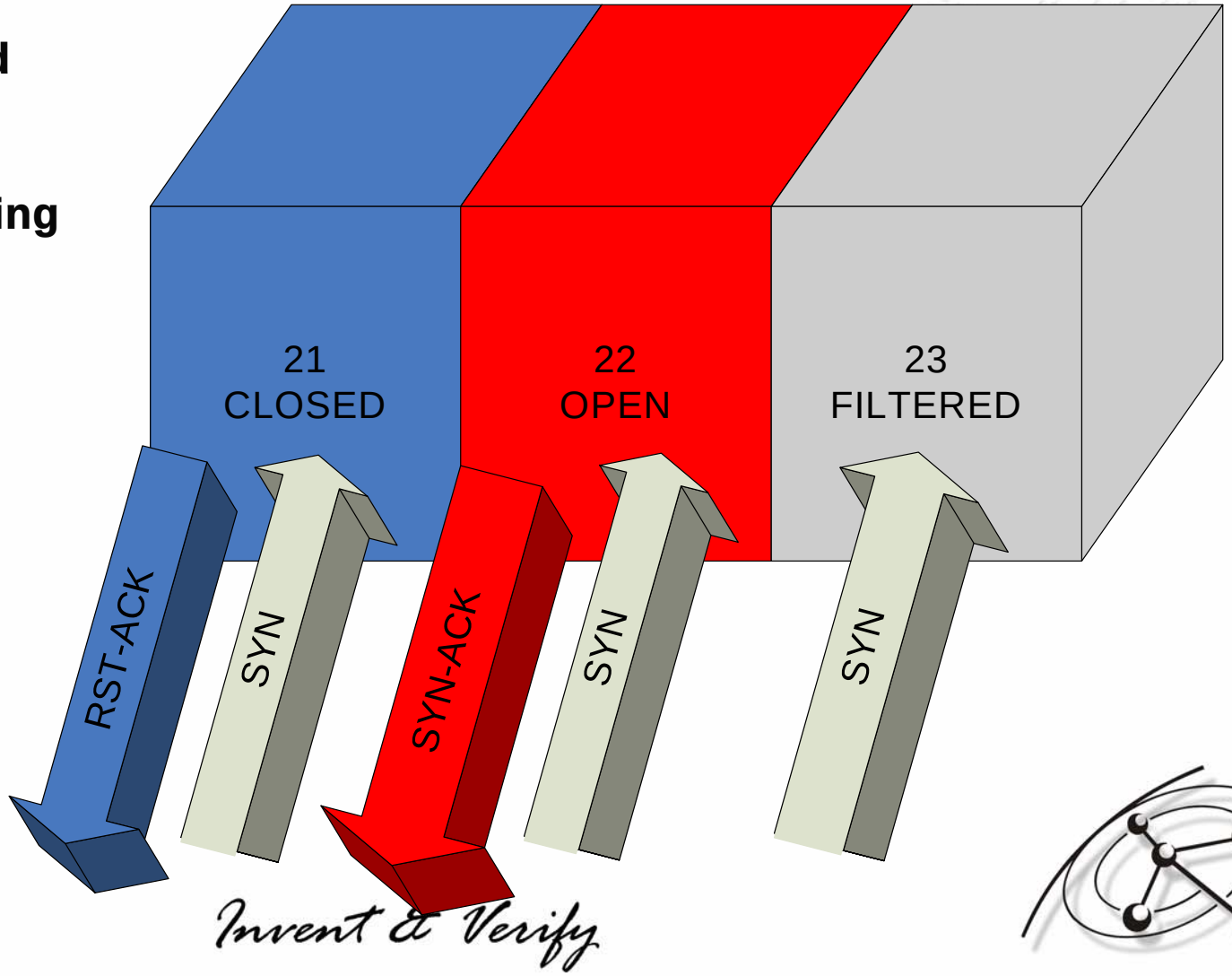
- Fabian “fabs” Yamaguchi
- 22 years old, lives in Berlin
- Works for Recurity Labs
- Studies Computer-Science/Electrical Engineering at the TUB
- Loves networking and reading/writing code

PortBunny

- Goal: Accurate results as fast as possible
- Difficulties:
  - Performance in data-networks is a complex topic
  - Large number of different setups
- Approach:
  - No “patch-work”-, “handle yet-another-condition”-scanner
  - Base algorithms on strong theoretical foundation

1. Port-Scanning - Basics

Identify open, closed and filtered ports by sending connection requests and observing responses.

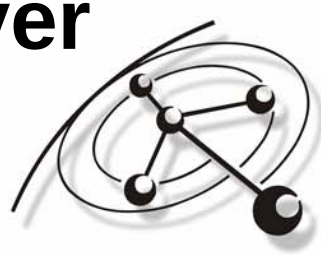


(TCP-SYN or “half-open”-scanning)

Naive port-scanner

```
foreach p in ports_to_scan:
        send_request_to(p)
        get_response()
```

- Won’t quite do it.
- Sending as fast as possible may result in dropped packets or even congestion collapse.
  - Open/Closed ports will be falsely reported as being filtered.
- **The optimal speed, which changes over time, needs to be determined.**



… wait a minute…

- … but this works just fine:

```
sock.connect(…)
While(is_data_left)
        sock.send(data_left)
sock.close()
```

- For TCP connections, you're right: Because TCP takes care of these issues for you!
- **But only for data-transfers across established TCP-connections!**

Tell us to slow down, please.

- *Q: Will the network explicitly tell us that we should slow down?*

A: In general, no.

- Exception: ICMP source-quenches,
- Exception: ECN.

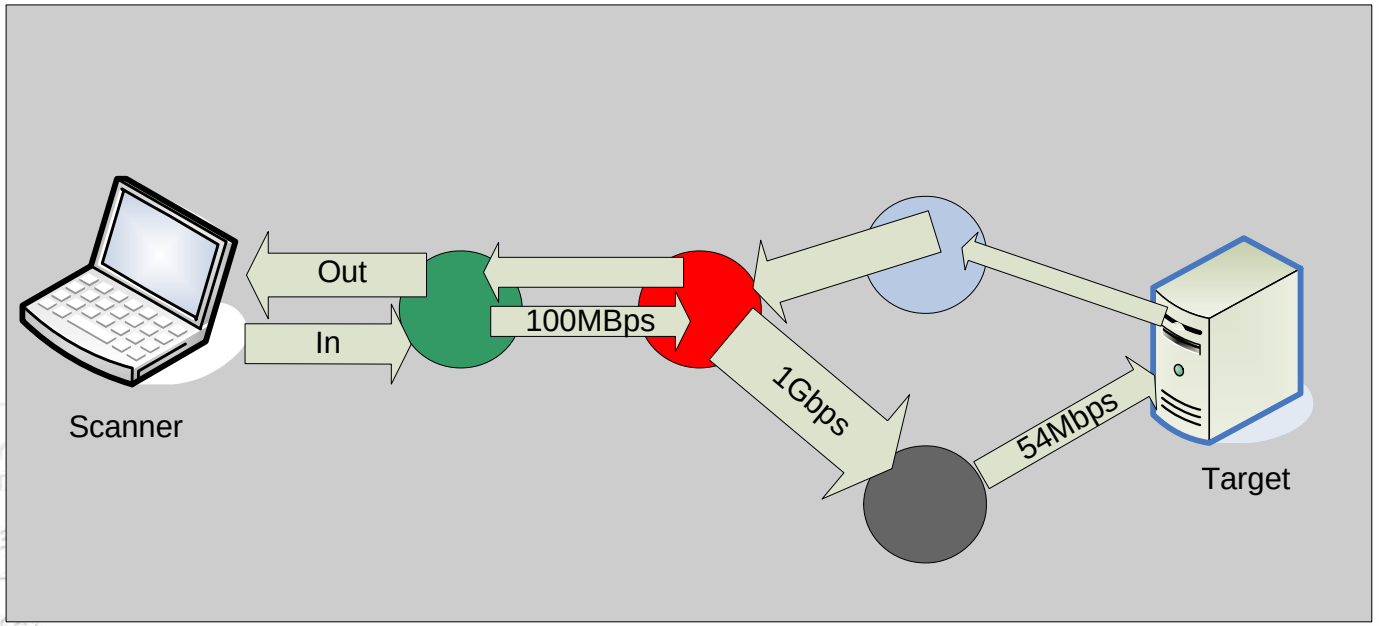
- People want forms of “explicit congestion notification” but they aren’t widely used yet.

What info do we have?

- If a response is received, we have a round-trip-time.
- Packet-drops can be detected given that we know a certain packet should have provoked an answer.
- That's all.

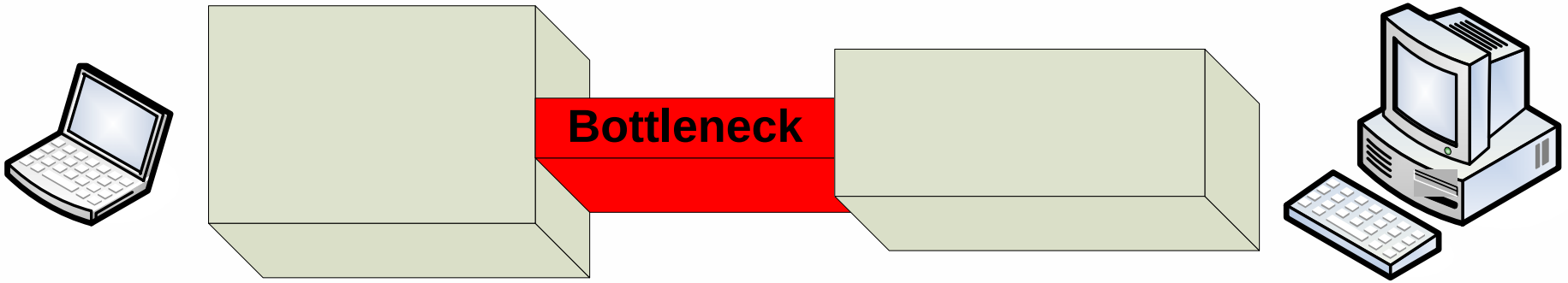
Networks – what we're dealing with

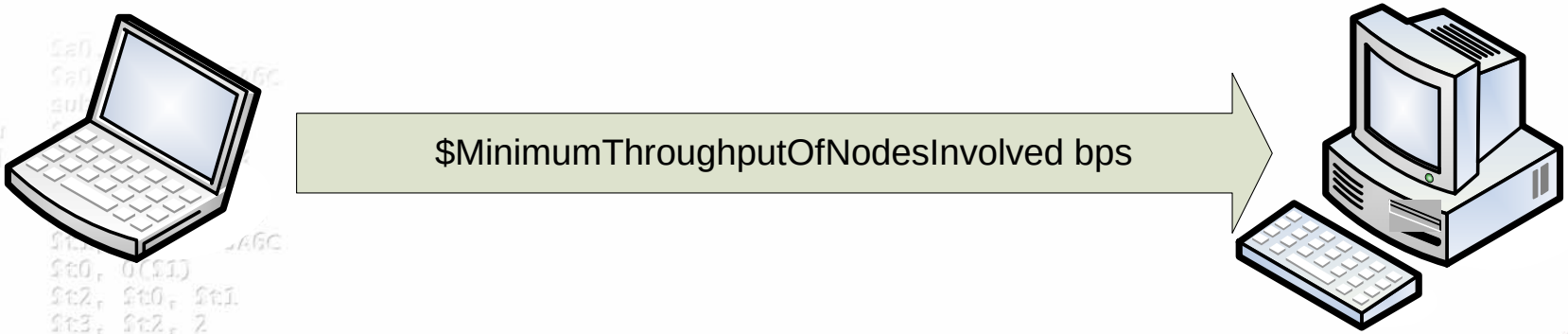
- Edges: Throughput (Delay), Reliability
- Nodes: Queuing-capacity



Simplification

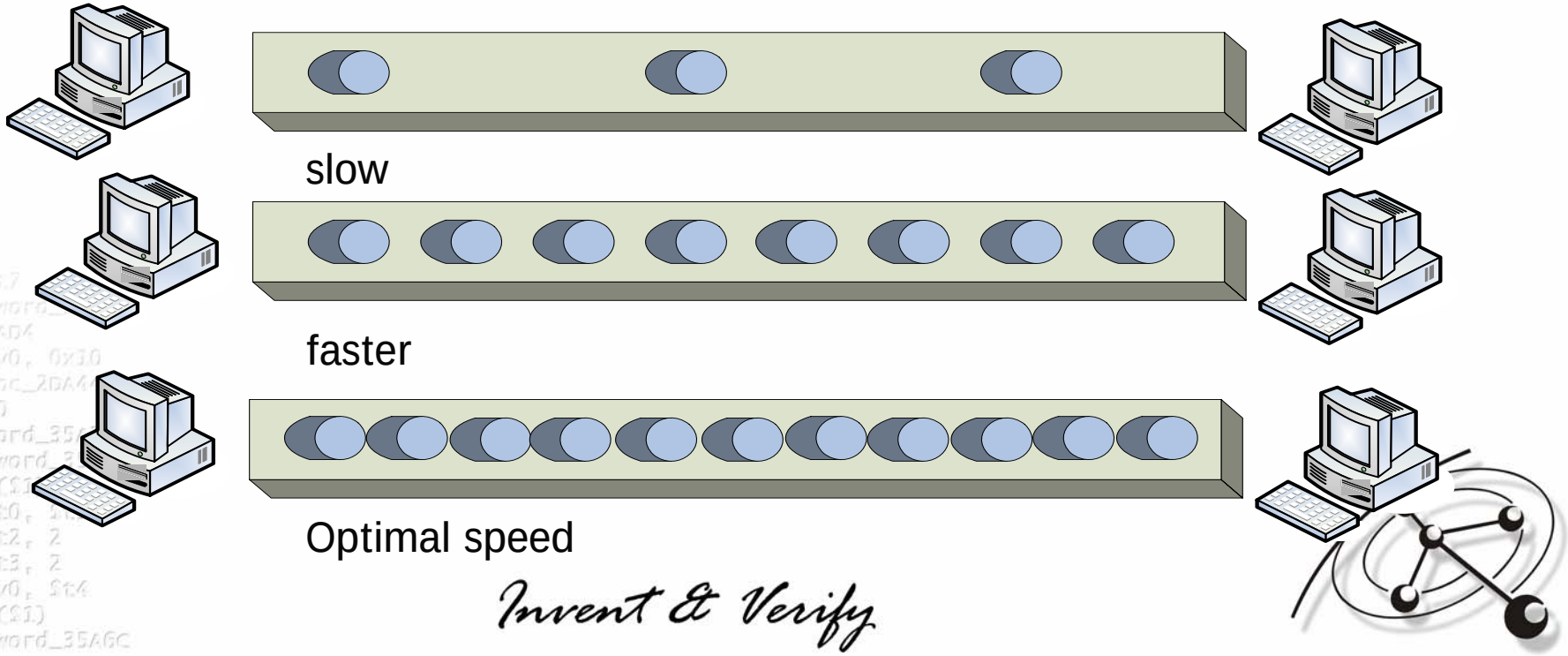
- Model implicitly suggested by the term “bottleneck” and by experience from socket-programming.





Optimal speed

- Speed is the number of packets sent per time-frame.
  Find the optimal delay.



Invent & Verify

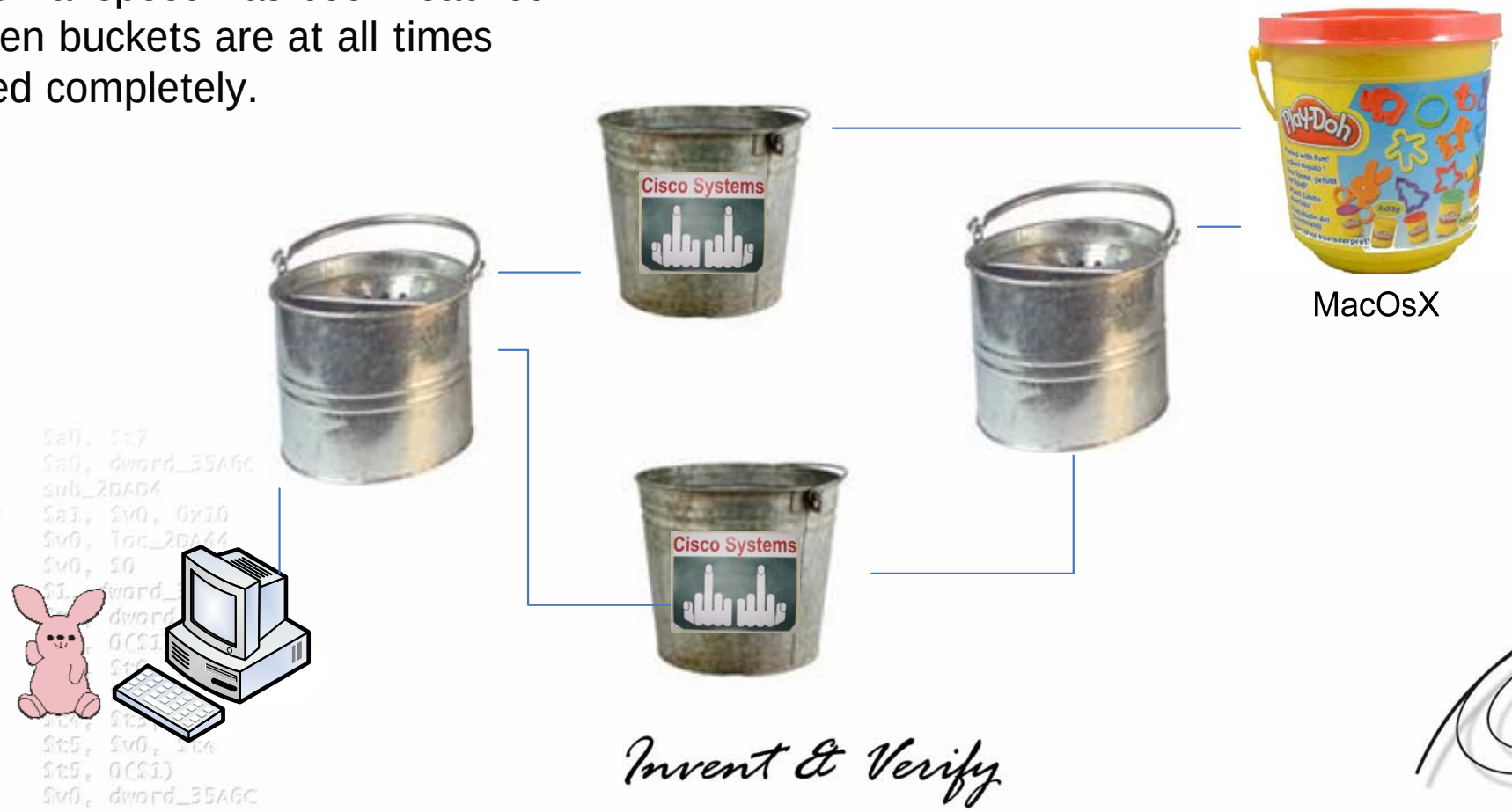
But don't forget the queuing-capacity!

- "You can fire 10 packets at a delay of 0 but that doesn't mean you can do the same with 100 packets." Why?

- The network has limited ability to queue data.
  - **This very important property of the network suggests a new model.**

The “bucket-model”

Think of each host as a bucket with a hole at the bottom. The optimal speed has been reached when buckets are at all times filled completely.



Invent & Verify

New model, new question

- Old question:
  "How long should I wait before sending the next packet"
- New question:
  "How much data can be out in the network at once?"
- **"Self-clocked"! New data is inserted when old data leaves.**

3. TCP Congestion Control

- TCP congestion control schemes **ask that exact same question!**
- Very active research-field.
- Let’s make use of those existing results!

TCP vs. Port-Scanning

TCP	**Port-Scanning**
Receiver acks packets.	Packets may not produce answers.
Timeouts are error-conditions	Timeouts are not error-conditions
Sequence-numbers are used	No sequence numbers

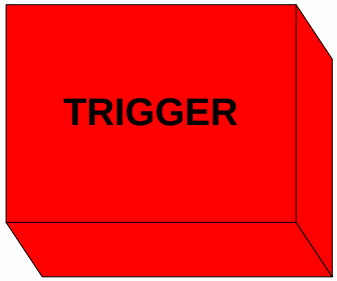
... in other words:

- The TCP-receiver is cooperative
- A port-scanned host is not cooperative.

- Of course, that doesn't mean we can't force it to be.

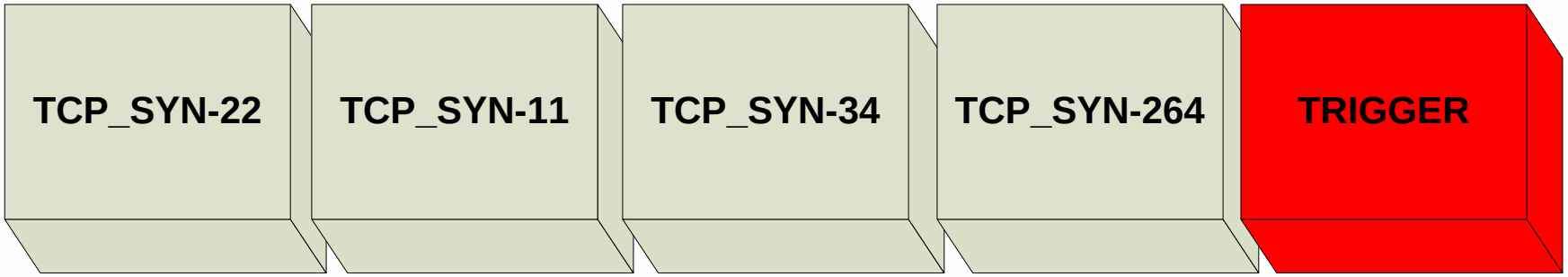
Triggers - forcing cooperation

- Before starting the scan, find one or more packets which trigger a response.
- **PortBunny tries the following**:
  - TCP-SYN Port 22/80/139/135 …
  - TCP-ACK Port …
  - ICMP-Echo Requests
  - ICMP Timestamp Requests
  - ICMP Address-Mask Requests
  - UDP Port …
  - IP-PROT Protocol …



Send probes in batches

- Terminate each batch with a trigger



- Probes are “attached” to the trigger as payload
- That way, each batch must produce an answer!

On trigger-response

- … the probability that all probes were received as well, is very high!
- Reason: Drops are caused by
  - Overflowing queues
  - Physical transmission errors (wireless)
- **In both cases, drops almost always occur in batches!**
- Exception: Random Early Drop

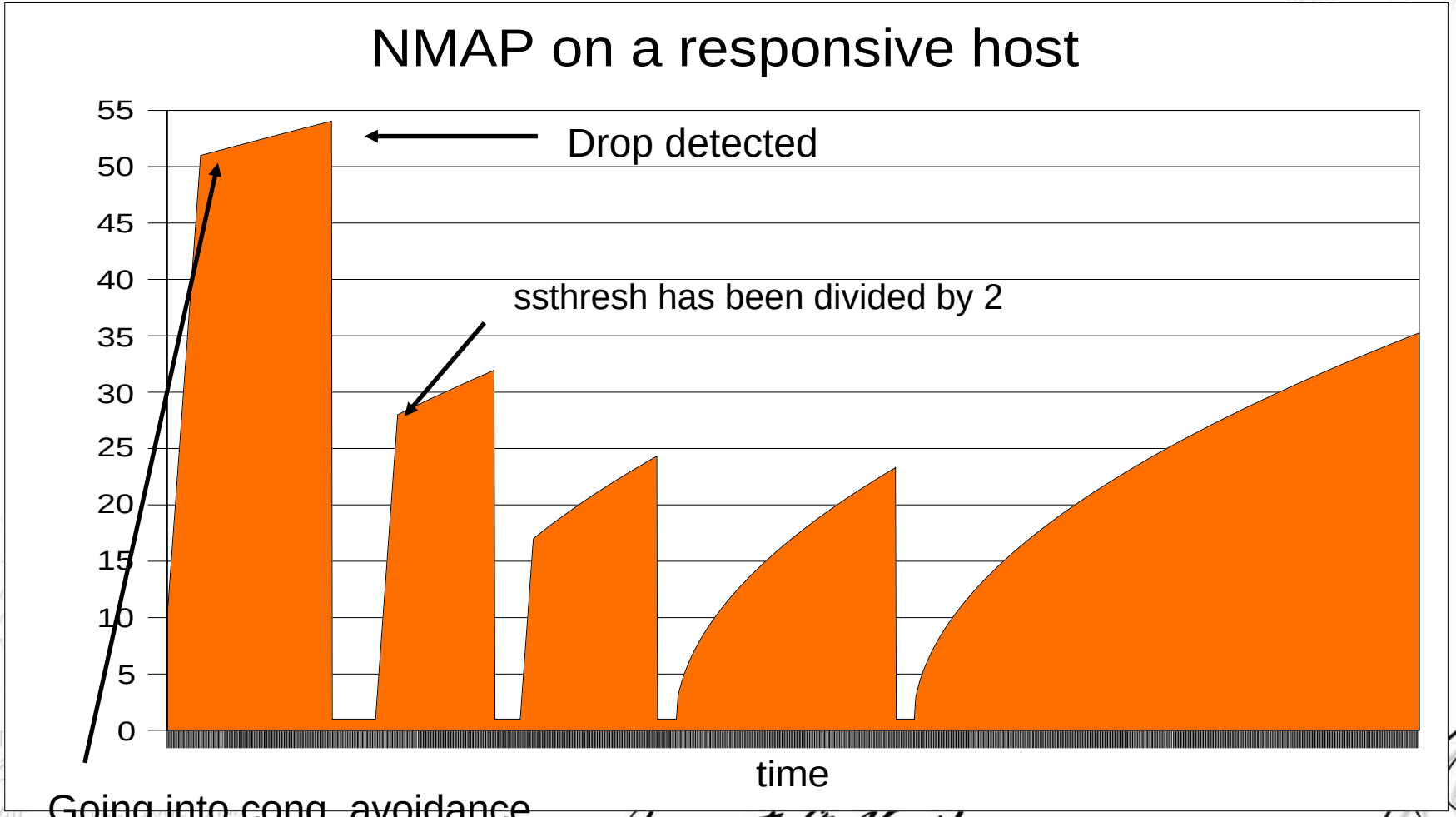
On drop

- All probes of the batch, which did not produce answers must be resent.
- To detect probes, which were dropped although their trigger was not, rescan filtered ports if their number is smaller than 30%.
- This increases accuracy.

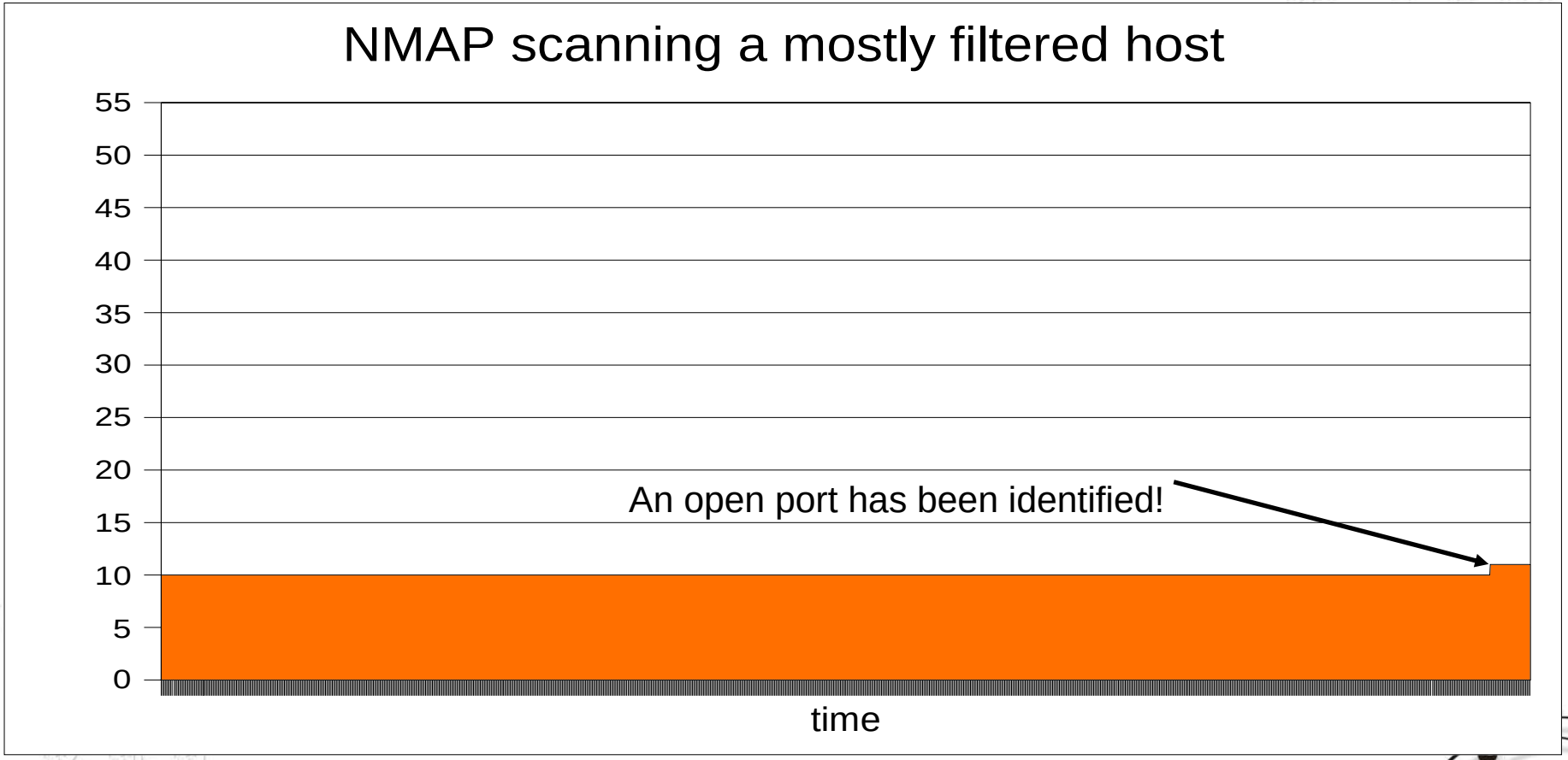
What's that good for?

- Trigger-responses now play the same role Acknowledgments play in TCP's congestion control!
- We receive constant information about the network's performance no matter if it is largely filtered or not!
- A timeout is actually a signal of error!

Probe-based congestion control (NMAP)



... and the same for a filtered host



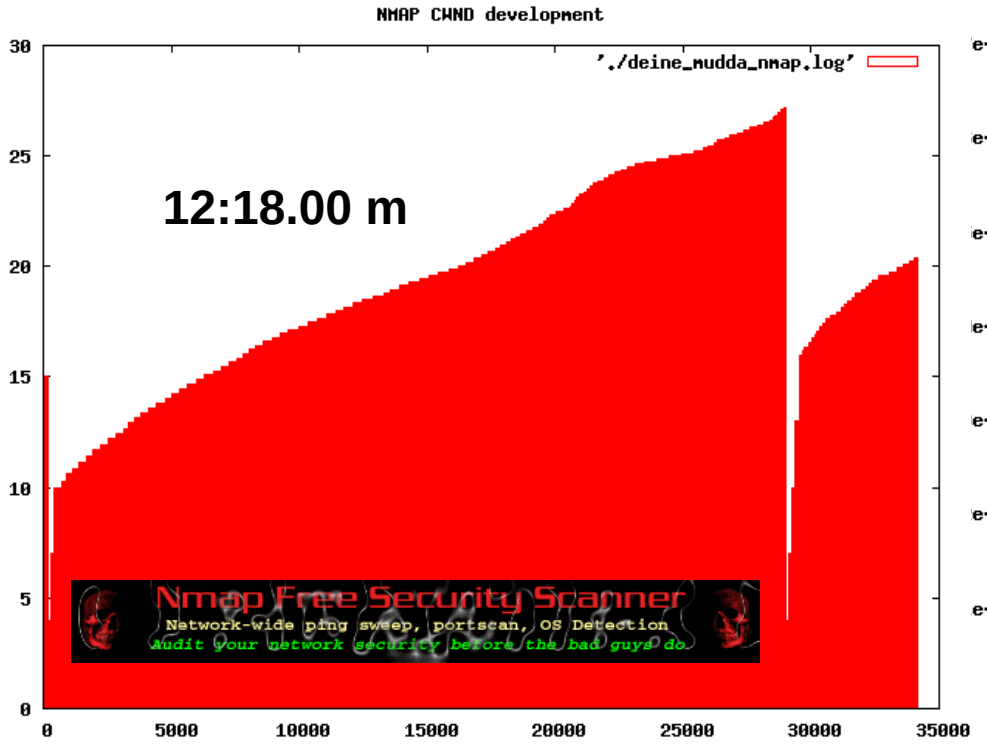
Port Scan Ping

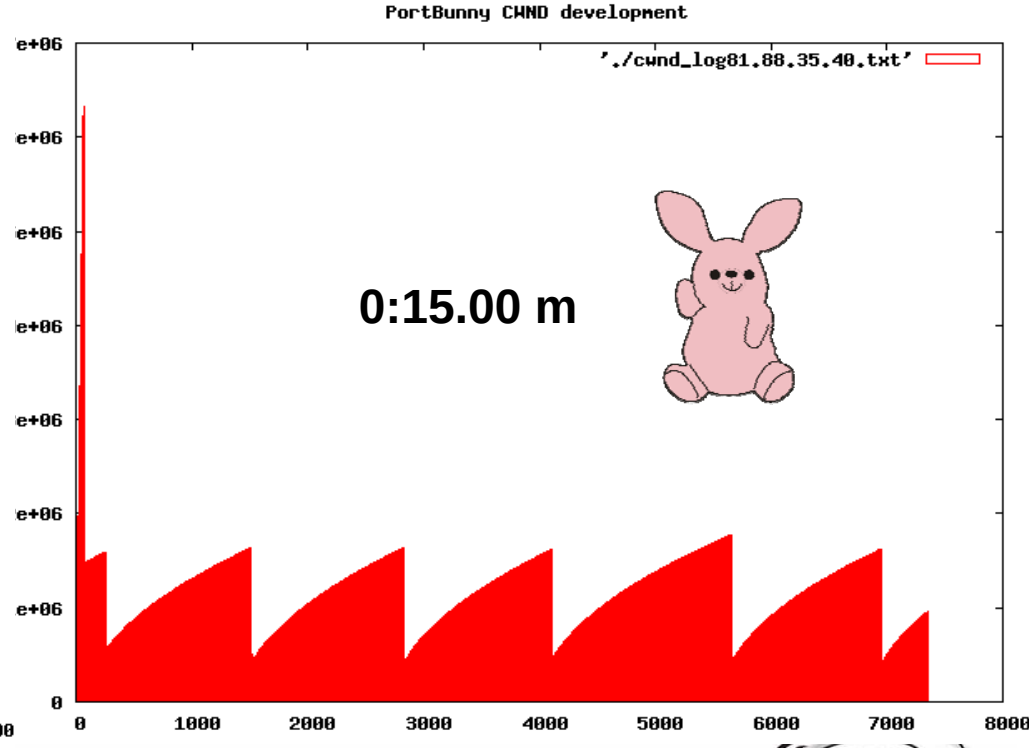
```
/* When a successful ping response comes back, it
counts as this many "normal" responses, because the
fact that pings are neccessary means we aren't
getting much input. */
```

- If a host has not responded in **5 seconds**, a ping is sent.
- A response is then counted as **3** regular responses.
- This is called the “port scan ping”-system

... and then there are filtered hosts ☺

- 65535 ports, mostly filtered, Internet.





Invent & Verify

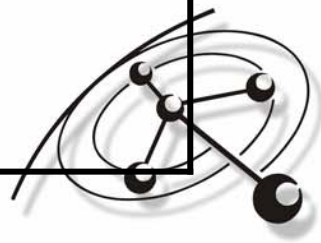
Timeout-detection - probe-based

```
/* A previous probe must have been lost ... */.
```

- Drops can only be detected after resending
- If a resent probe produces an answer, obviously, the initial probe was dropped.

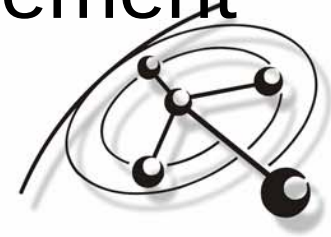
Triggers vs. TCP

TCP	**Trigger-based scanning**
Receiver acks packets.	Triggers are acknowledged.
Timeouts are error-conditions	Trigger-Timeouts are error-conditions.
Sequence-numbers are used	Sequence-numbers are used for all triggers.



Benefits of trigger-use

- Filtered hosts can be scanned properly
- Packet-drops can be detected much earlier leading to better responsiveness to drops.
- Immediate probe resends are not necessary anymore which helps reduce useless extra traffic.
- Port-Scanning has been ported to the tcp-congestion control domain! We can implement any TCP-congestion-control scheme!



Problems with triggers

- Not all triggers have the same quality:
  - ICMP-triggers and UDP-triggers could be rate-limited while probes aren't.
  - TCP-triggers are the best available triggers.
    - QoS might be a problem, some times

- A host may not respond to any supported trigger.

Fixes

- Try to find TCP-SYN-triggers first and use ICMP and UDP-triggers and others as a fallback-solution.

- If a TCP-SYN-trigger can be found at scan-time, add it to the list of triggers in use and discard fallback-triggers.

Problem solved? Not quite:

- Bucket-model is NOT valid for rate-limiting firewalls, instead, the pipe-model is valid!
  - using classical congestion-control algorithms in this case is not appropriate.

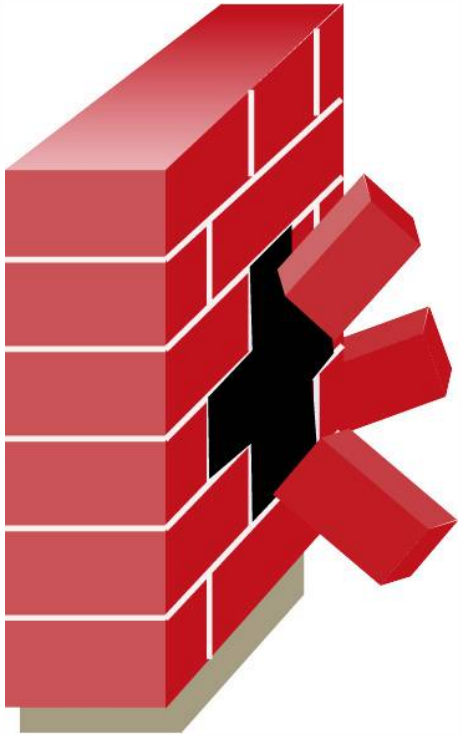
- We have implemented a number of congestion-control-schemes designed for TCP **but how will the user know which one to choose?**

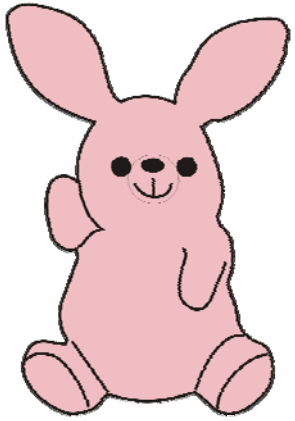
We need detection

- The scanner needs to be able to interpret network-conditions and choose a timing-algorithm, which is most suited by itself.

- The scanner is the expert on these issues because it’s communicating with the target!

Detection of rate-limiting firewalls



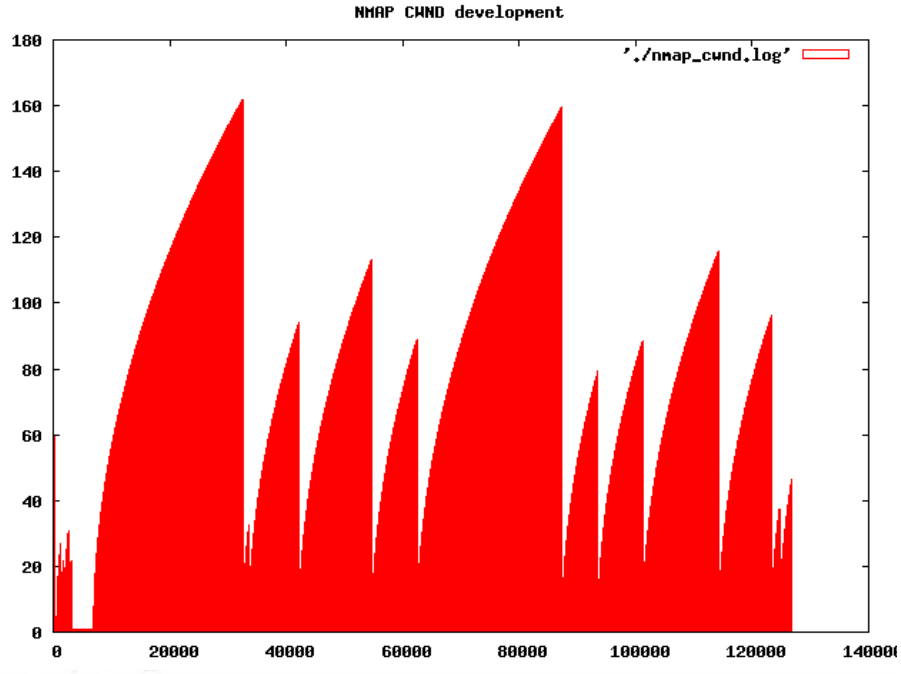


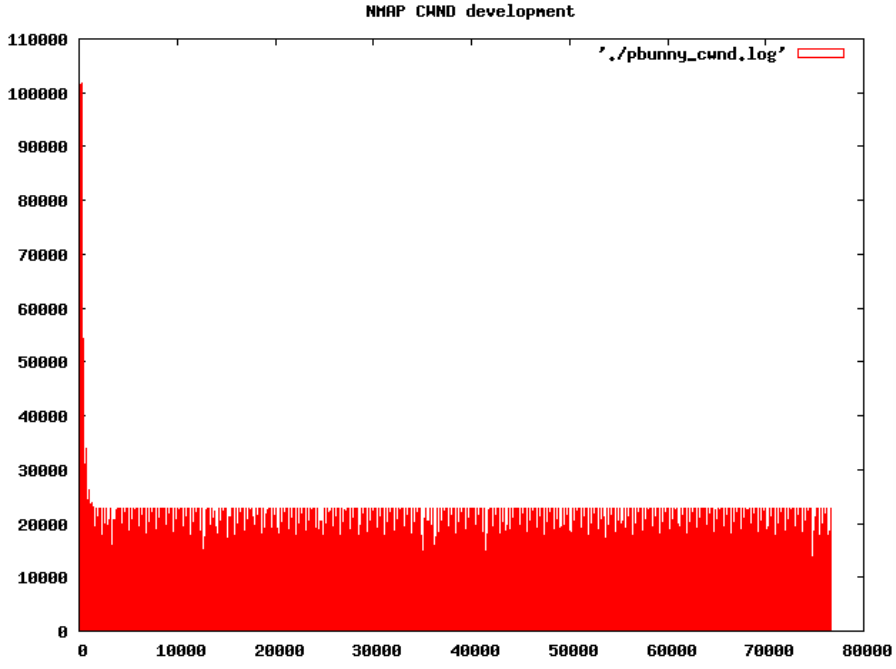
NMAP's rate-limit detection

```
/* If packet drops are particularly bad, enforce a
delay between packet sends (useful for cases such
as UDP scan where responses are frequently rate
limited by dest machines or firewalls) */
```

- "Particularly bad" is vague => False positives are common
- Artificial delays mean that the algorithm is no longer self-clocked ☹ No more theoretical foundation for the timing-algo.
- CWND does no longer reflect the number of packets out at once.

Effect of false-positive



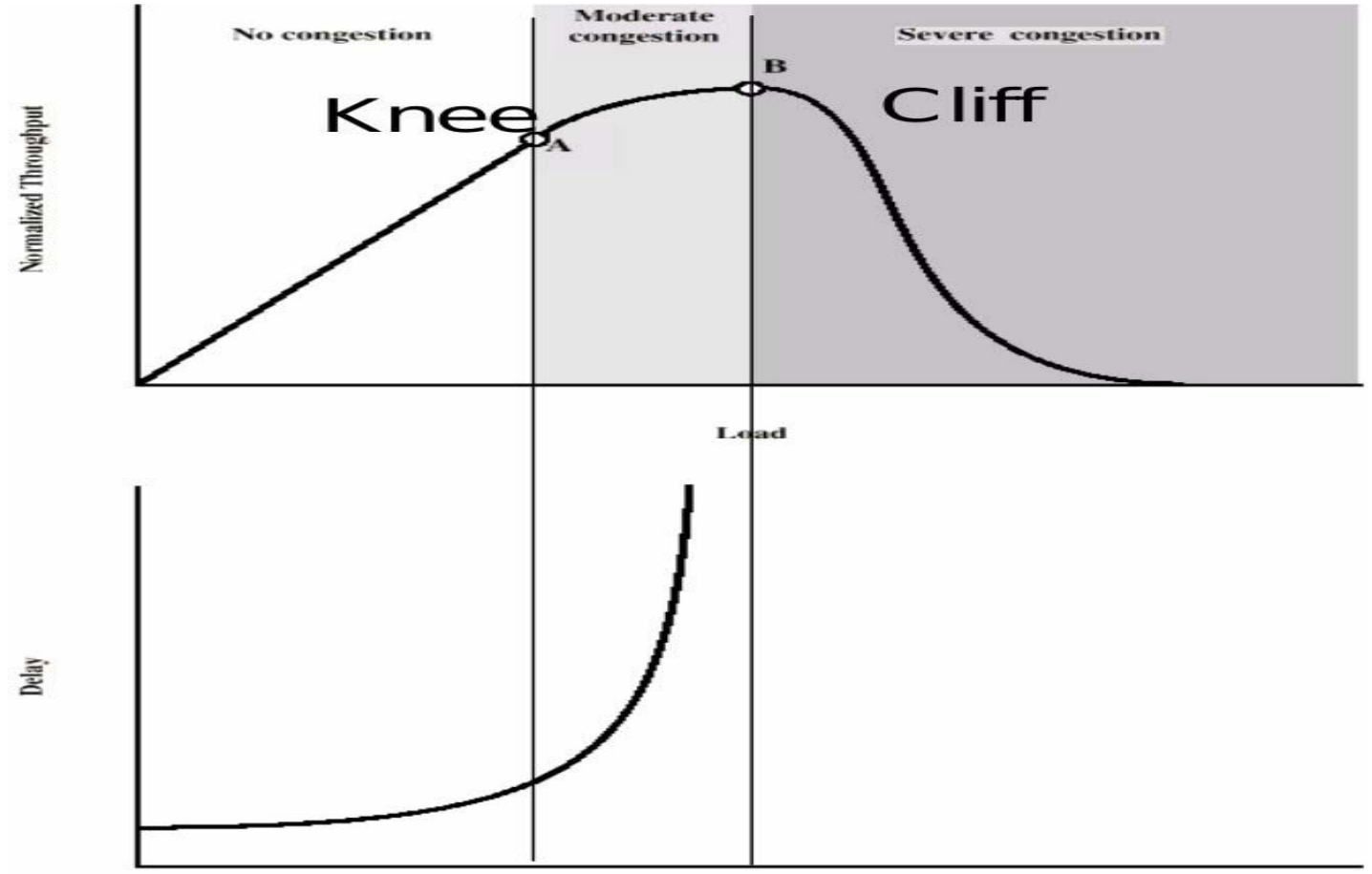


24:41.51 m

Invent & Verify

7:58.03 m

Let's take the RTT into account!

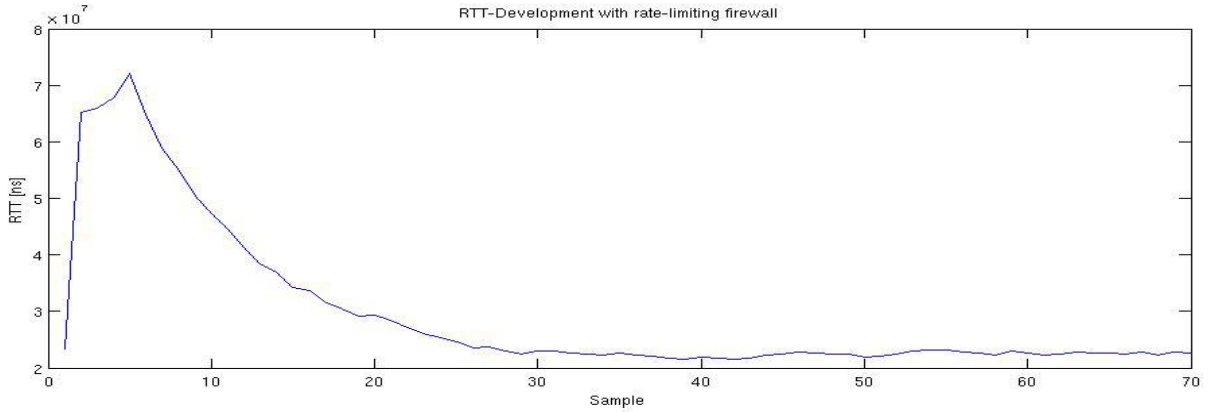


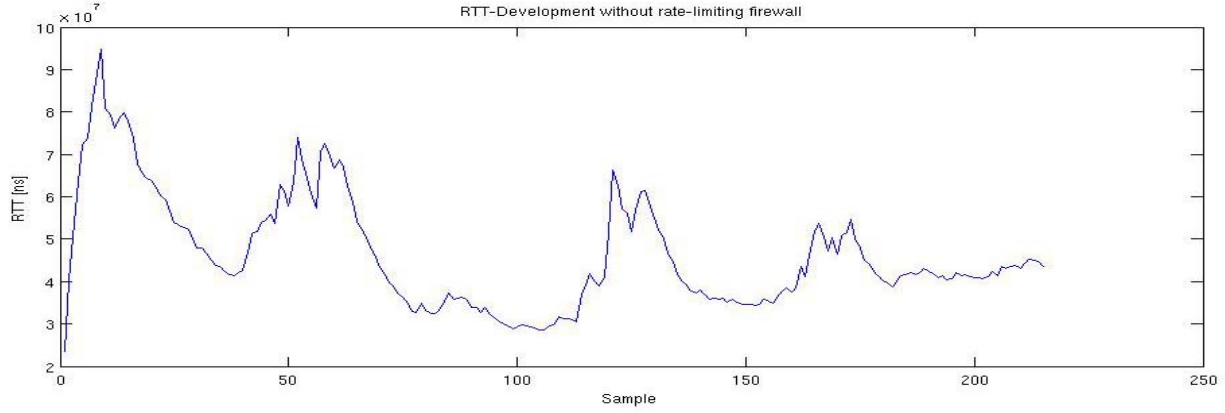
Invent & Verify

What does this look like during a scan?

- When scanning, we constantly change the network-load in reaction to network-events
- How does that effect the rtt?
- Is the data clear enough to differentiate between normal congestion and the effects of a rate-limiter?

Yes, it is ☺





Invent & Verify

Working on this approach

- If n drops occur in a row and the RTT observed is relatively constant and close to the base-RTT, then we are dealing with a rate-limiting firewall.

- This is still being tested, but another nice “two-shot” approach has been implemented.

Observe: This is a packet...

```
▷ Frame 161 (58 bytes on wire, 58 bytes captured)
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▽ Transmission Control Protocol, Src Port: 61373 (61373), Dst Port: tacacs-ds (65), Seq: 0, Len: 0
    Source port: 61373 (61373)
    Destination port: tacacs-ds (65)
    Sequence number: 0    (relative sequence number)
    Header length: 24 bytes
  ▷ Flags: 0x02 (SYN)
    Window size: 0
  ▷ Checksum: 0xaa25 [correct]
  ▽ Options: (4 bytes)
      Maximum segment size: 1460 bytes
```

Invent & Verify

... and this is, too.

```
▷ Frame 5 (74 bytes on wire, 74 bytes captured)
▷ Ethernet II, Src: Giga-Byt_bf:9a:0c (00:0f:ea:bf:9a:0c), Dst: Cisco-Li_c9:24:57 (00:14:bf:c9:24:57)
▷ Internet Protocol, Src: 192.168.1.208 (192.168.1.208), Dst: 209.85.129.99 (209.85.129.99)
▽ Transmission Control Protocol, Src Port: 41351 (41351), Dst Port: www (80), Seq: 0, Len: 0
    Source port: 41351 (41351)
    Destination port: www (80)
    Sequence number: 0    (relative sequence number)
    Header length: 40 bytes
  ▷ Flags: 0x02 (SYN)
    Window size: 5840
  ▷ Checksum: 0xe209 [correct]
  ▽ Options: (20 bytes)
      Maximum segment size: 1460 bytes
      SACK permitted
      Timestamps: TSval 177528, TSecr 0
      NOP
      Window scale: 6 (multiply by 64)
```

Invent & Verify

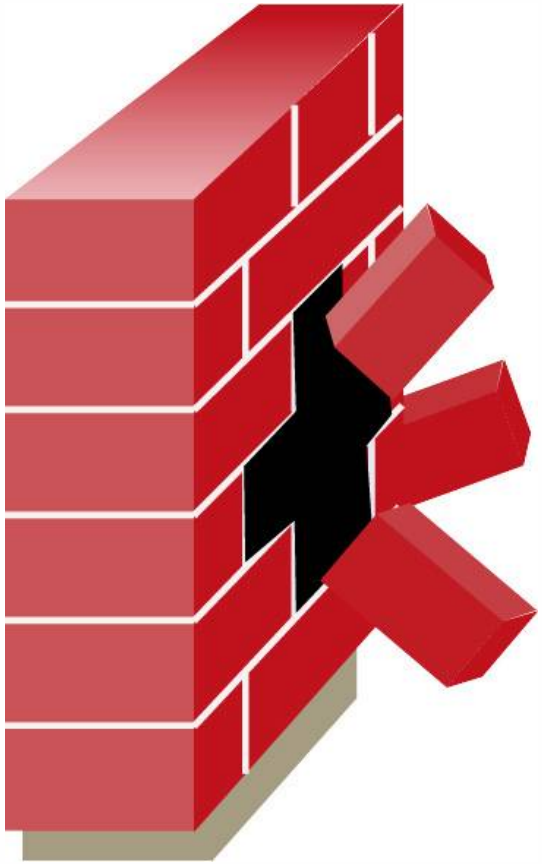
Now if the bucket claims...

- ... that 4 of these fit:

- ... but 4 of those would, too:

Invent & Verify

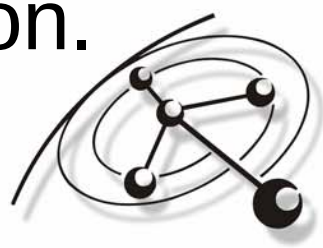
... then it's not really a bucket.



Invent & Verify

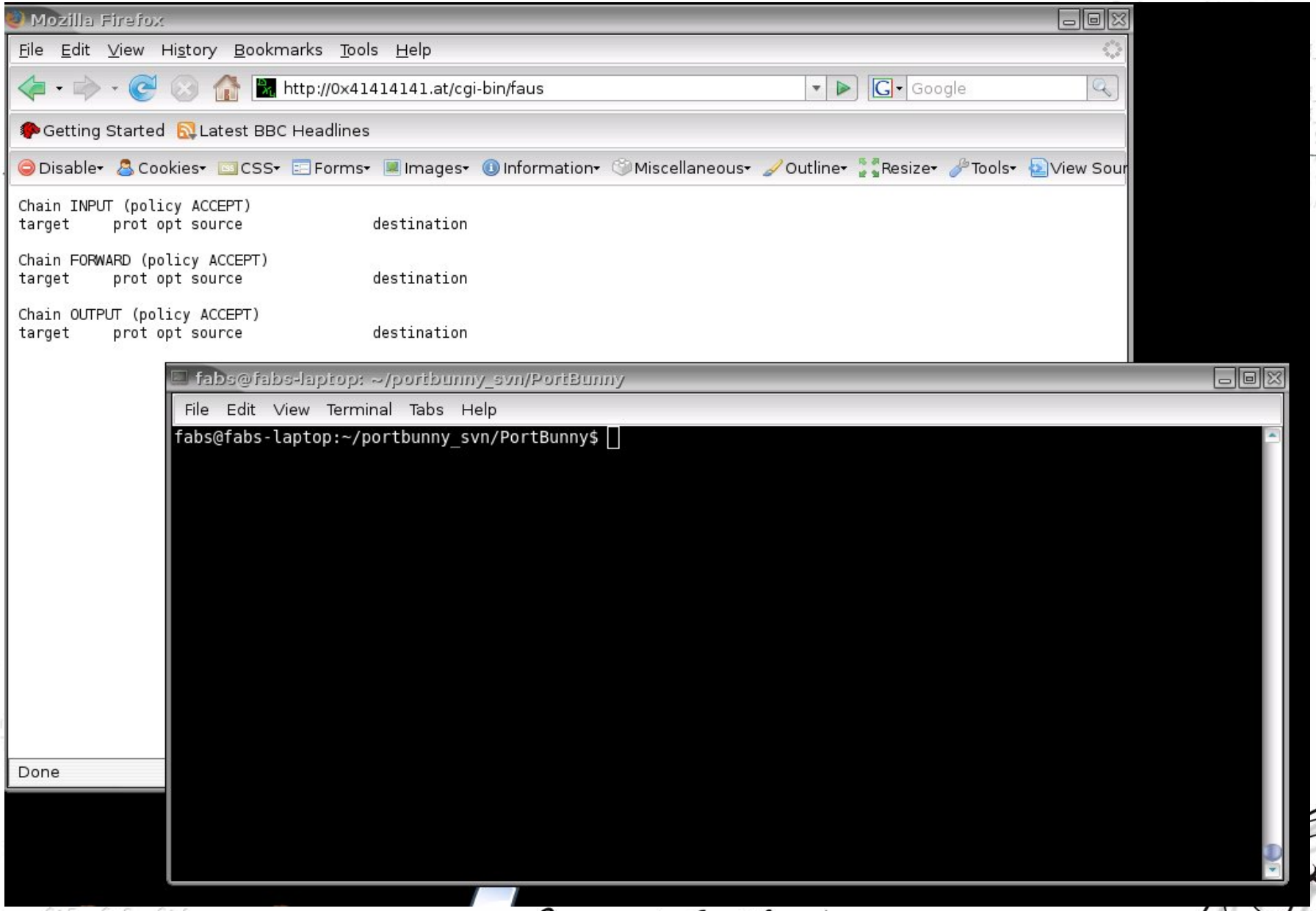
Packet-size does not matter for rate-limiters!

- Rate-limitation limits **number of packets**, the packet-size does not matter!
- Congestion is caused by **too much data** in the network
- **Just enlarge the packet** (Add TCP-options)
- If still the same number of packets return, we're obviously dealing with rate-limitation.



Or create background-traffic

- In practice, enlarging the TCP-SYN by 40 byte doesn't change much.
- In contrast, for ICMP-Echo-Request-triggers, the approach is feasible.
- For all other triggers, background-traffic can be generated instead.
- UDP-datagrams are a good option.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://0x41414141.at/cgi-bin/faus
Google
Getting Started
Latest BBC Headlines
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Sour
Chain INPUT (policy ACCEPT)
target prot opt source destination
Chain FORWARD (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
Done
fabs@fabs-laptop: ~/portbunny_svn/PortBunny
File Edit View Terminal Tabs Help
fabs@fabs-laptop:~/portbunny_svn/PortBunny$

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

```
fabs@fabs-laptop: ~/portbunny_svn/PortBunny
File  Edit  View  Terminal  Tabs  Help
fabs@fabs-laptop:~/portbunny_svn/PortBunny$ sudo ./rlimit_test.sh 193.239.165.73
Aug  8 00:07:46 fabs-laptop kernel: [ 5954.536000] nresponses_first_round: 49
Aug  8 00:07:46 fabs-laptop kernel: [ 5954.536000] nresponses_second_round: 28
```


35 0.000	10.248.240.254	193.239.165.73	TCP	61378 > 81 [SYN] Seq=35 Len=0 MSS=1460
36 0.000	10.248.240.254	193.239.165.73	TCP	61379 > 81 [SYN] Seq=36 Len=0 MSS=1460
37 0.000	10.248.240.254	193.239.165.73	TCP	61380 > 81 [SYN] Seq=37 Len=0 MSS=1460
38 0.000	10.248.240.254	193.239.165.73	TCP	61381 > 81 [SYN] Seq=38 Len=0 MSS=1460
39 0.000	10.248.240.254	193.239.165.73	TCP	61382 > 81 [SYN] Seq=39 Len=0 MSS=1460
40 0.000	10.248.240.254	193.239.165.73	TCP	61383 > 81 [SYN] Seq=40 Len=0 MSS=1460
41 0.000	10.248.240.254	193.239.165.73	TCP	61384 > 81 [SYN] Seq=41 Len=0 MSS=1460
42 0.000	10.248.240.254	193.239.165.73	TCP	61385 > 81 [SYN] Seq=42 Len=0 MSS=1460
43 0.000	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
44 0.000	10.248.240.254	193.239.165.73	TCP	61383 > 81 [SYN] Seq=40 Len=0 MSS=1460
45 0.000	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
46 0.000	10.248.240.254	193.239.165.73	TCP	61384 > 81 [SYN] Seq=41 Len=0 MSS=1460
47 0.001	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
48 0.001	10.248.240.254	193.239.165.73	TCP	61385 > 81 [SYN] Seq=42 Len=0 MSS=1460
49 0.001	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
50 0.001	10.248.240.254	193.239.165.73	TCP	61386 > 81 [SYN] Seq=43 Len=0 MSS=1460
51 0.212	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
52 0.212	10.248.240.254	193.239.165.73	TCP	61387 > 81 [SYN] Seq=44 Len=0 MSS=1460
53 0.214	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
54 0.214	10.248.240.254	193.239.165.73	TCP	61388 > 81 [SYN] Seq=45 Len=0 MSS=1460
55 0.214	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
56 0.215	10.248.240.254	193.239.165.73	TCP	61389 > 81 [SYN] Seq=46 Len=0 MSS=1460
57 0.215	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
58 0.216	10.248.240.254	193.239.165.73	TCP	61390 > 81 [SYN] Seq=47 Len=0 MSS=1460
59 0.216	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
60 0.219	10.248.240.254	193.239.165.73	TCP	61391 > 81 [SYN] Seq=48 Len=0 MSS=1460
206 2.027	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
207 2.027	10.248.240.254	193.239.165.73	TCP	61392 > 81 [SYN] Seq=49 Len=0 MSS=1460
208 2.027	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
209 2.027	10.248.240.254	193.239.165.73	TCP	61393 > 81 [SYN] Seq=50 Len=0 MSS=1460
210 2.027	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
211 2.027	10.248.240.254	193.239.165.73	TCP	61394 > 81 [SYN] Seq=51 Len=0 MSS=1460
212 2.027	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
213 2.038	10.248.240.254	193.239.165.73	TCP	61395 > 81 [SYN] Seq=52 Len=0 MSS=1460
214 2.038	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
215 2.038	10.248.240.254	193.239.165.73	TCP	61396 > 81 [SYN] Seq=53 Len=0 MSS=1460
216 2.038	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
217 2.038	10.248.240.254	193.239.165.73	TCP	61397 > 81 [SYN] Seq=54 Len=0 MSS=1460
218 2.038	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
219 2.038	10.248.240.254	193.239.165.73	TCP	61398 > 81 [SYN] Seq=55 Len=0 MSS=1460
220 2.038	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
221 2.038	10.248.240.254	193.239.165.73	TCP	61399 > 81 [SYN] Seq=56 Len=0 MSS=1460
222 2.038	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
223 2.050	10.248.240.254	193.239.165.73	TCP	61400 > 81 [SYN] Seq=57 Len=0 MSS=1460
224 2.050	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
225 2.050	10.248.240.254	193.239.165.73	TCP	61401 > 81 [SYN] Seq=58 Len=0 MSS=1460
226 2.050	10.248.240.254	193.239.165.73	UDP	Source port: 32773 Destination port: 23
227 2.050	10.248.240.254	193.239.165.73	TCP	61402 > 81 [SYN] Seq=59 Len=0 MSS=1460
228 2.213	193.239.165.73	10.248.240.254	ICMP	Destination unreachable (Port unreachable)
229 2.215	193.239.165.73	10.248.240.254	TCP	81 > 61394 [SYN, ACK] Seq=2641872645 Ack=2 Win=5840 Len=0 MS
230 2.215	10.248.240.254	193.239.165.73	TCP	61394 > 81 [RST] Seq=2 Len=0
231 2.218	193.239.165.73	10.248.240.254	TCP	81 > 61373 [SYN, ACK] Seq=2635198356 Ack=3 Win=5840 Len=0 MS

Invent & Verify

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere            tcp flags:FIN,SYN,RST,ACK/SYN limit: avg 20/sec burst 20
DROP       tcp  --  anywhere             anywhere            tcp flags:FIN,SYN,RST,ACK/SYN
ACCEPT     icmp --  anywhere             anywhere            limit: avg 20/sec burst 20
DROP       icmp --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (pol
target     prot o
```

```
fabs@fabs-laptop: ~/portbunny_svn/PortBunny
File  Edit  View  Terminal  Tabs  Help
fabs@fabs-laptop:~/portbunny_svn/PortBunny$ sudo ./rlimit_test.sh 193.239.165.73 50
Aug  8 00:07:46 fabs-laptop kernel: [ 5954.536000] nresponses_first_round: 49
Aug  8 00:07:46 fabs-laptop kernel: [ 5954.536000] nresponses_second_round: 28

fabs@fabs-laptop:~/portbunny_svn/PortBunny$ sudo ./rlimit_test.sh 193.239.165.73 50
Aug  8 00:12:01 fabs-laptop kernel: [ 6208.788000] nresponses_first_round: 20
Aug  8 00:12:01 fabs-laptop kernel: [ 6208.788000] nresponses_second_round: 20
```

Invent & Verify

Using Bunny

- Download it at http://portbunny.recurity.com

- "# portbunny $host" to scan a host
- "# portbunny $network" to scan a network
- "# portbunny $network -d" to only trigger $network for discovery-purposes
- "# portbunny $host -l" to scan and generate a detailed scan-log.

Send us your scan-log ;)

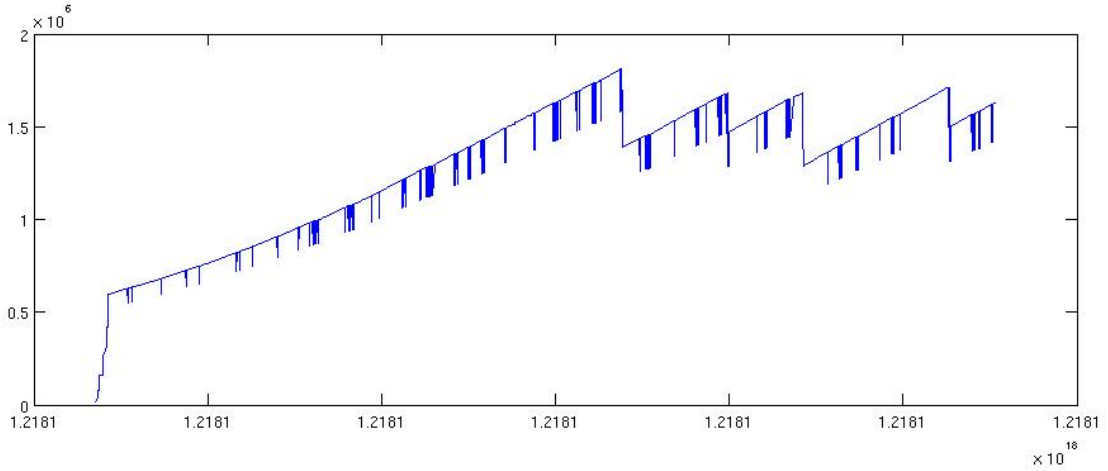
fabs@fabs-laptop: ~/portbunny_svn/PortBunny

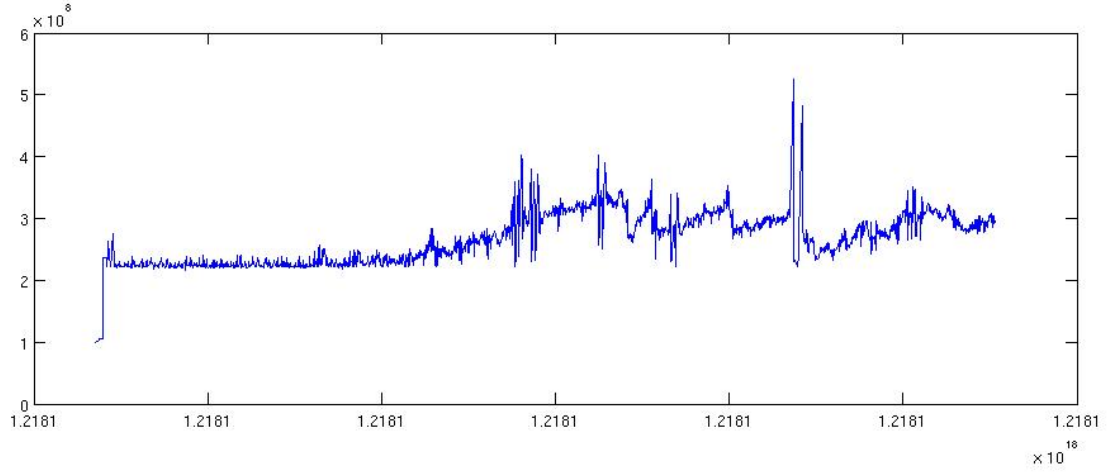
File Edit View Terminal Tabs Help

```
193.239.165.73 I TRIGGER_RECEIVED 1218146182183569854 621500 600000 229119544 1772944929 7501428 221499012 TCP_SYN 27
193.239.165.73 I TRIGGER_RECEIVED 1218146182201437459 622500 600000 227614889 1773274442 5955584 221947629 TCP_SYN 28
193.239.165.73 I TRIGGER_RECEIVED 1218146182218044636 623500 600000 226356537 1773479370 4671616 221864233 TCP_SYN 29
193.239.165.73 I TRIGGER_RECEIVED 1218146182222918647 624400 600000 227368390 1770988219 5994863 219193770 TCP_SYN 30
193.239.165.73 I TRIGGER_RECEIVED 1218146182333024739 624600 600000 226272015 1770528402 4955965 220913710 TCP_SYN 31
193.239.165.73 I TRIGGER_RECEIVED 1218146182366712574 625600 600000 225584184 1769898505 4346871 220686153 TCP_SYN 32
193.239.165.73 I TRIGGER_DROPPED 1218146182396839279 548275 310000 225584184 1769898505 4346871 220686153 TCP_SYN 33
193.239.165.73 I TRIGGER_RECEIVED 1218146182399869446 626600 600000 239435365 1783176637 16538286 234515445 TCP_SYN 33
193.239.165.73 I TRIGGER_RECEIVED 1218146182410439757 626600 600000 239697853 1787085134 16312212 226805576 TCP_SYN 34
193.239.165.73 I TRIGGER_DROPPED 1218146182429164503 549150 310000 239697853 1787085134 16312212 226805576 TCP_SYN 5
193.239.165.73 I TRIGGER_RECEIVED 1218146182435690425 627600 600000 247723356 1797843850 22992875 234144357 TCP_SYN 5
193.239.165.73 I TRIGGER_DROPPED 1218146182444468438 549150 270000 247723356 1797843850 22992875 234144357 TCP_SYN 6
193.239.165.73 I TRIGGER_DROPPED 1218146182450323566 480506 270000 247723356 1797843850 22992875 234144357 TCP_SYN 35
193.239.165.73 I TRIGGER_RECEIVED 1218146182454866860 549150 270000 255504017 1809869520 29270327 236756151 TCP_SYN 6
193.239.165.73 I TRIGGER_RECEIVED 1218146182457217782 549150 270000 257218886 1817898365 29981591 234262535 TCP_SYN 35
193.239.165.73 I TRIGGER_RECEIVED 1218146182550435359 549150 270000 258391989 1807991508 32393051 217330438 TCP_SYN 36
193.239.165.73 I TRIGGER_DROPPED 1218146182592418716 481381 270000 258391989 1807991508 32393051 217330438 TCP_SYN 37
193.239.165.73 I TRIGGER_RECEIVED 1218146182599526170 550150 270000 257828289 1814688896 30992177 232696326 TCP_SYN 37
193.239.165.73 I TRIGGER_DROPPED 1218146182622526283 481381 240000 257828289 1814688896 30992177 232696326 TCP_SYN 38
193.239.165.73 I TRIGGER_RECEIVED 1218146182628559656 550150 270000 255461108 1819471886 28027123 231619102 TCP_SYN 38
193.239.165.73 I TRIGGER_RECEIVED 1218146182641775497 550150 270000 252752260 1823292270 24840727 231254369 TCP_SYN 39
193.239.165.73 I TRIGGER_DROPPED 1218146182650270776 482256 270000 252752260 1823292270 24840727 231254369 TCP_SYN 7
193.239.165.73 I TRIGGER_RECEIVED 1218146182650571110 551150 270000 260050349 1835299621 30637897 239918884 TCP_SYN 7
193.239.165.73 I TRIGGER_DROPPED 1218146182668945861 482256 230000 260050349 1835299621 30637897 239918884 TCP_SYN 8
193.239.165.73 I TRIGGER_RECEIVED 1218146182671268915 551150 270000 266577318 1847909792 35588594 242022623 TCP_SYN 8
193.239.165.73 I TRIGGER_RECEIVED 1218146182687148095 551150 270000 270612771 1859405438 38187092 242484370 TCP_SYN 9
193.239.165.73 I TRIGGER_RECEIVED 1218146182768898268 552150 270000 273360304 1845354803 42690954 218375044 TCP_SYN 10
193.239.165.73 I TRIGGER_RECEIVED 1218146182774030847 553150 270000 268969662 1838175265 39197754 223489812 TCP_SYN 11
193.239.165.73 I TRIGGER_RECEIVED 1218146182804214127 553450 270000 275091864 1819979105 47594476 211575748 TCP_SYN 12
193.239.165.73 I TRIGGER_RECEIVED 1218146182834035258 553750 270000 277261454 1803901151 51773811 211419434 TCP_SYN 13
193.239.165.73 I TRIGGER_RECEIVED 1218146182861747133 553950 270000 269610874 1797852155 44879355 219438647 TCP_SYN 14
193.239.165.73 I TRIGGER_RECEIVED 1218146182880758664 554950 270000 273778010 1811529465 47336827 238408829 TCP_SYN 15
193.239.165.73 I TRIGGER_RECEIVED 1218146182889683840 555650 270000 267068682 1805672461 41359625 220584179 TCP_SYN 16
:
```

Invent & Verify

Contains all relevant info





Thank you!

Recurity Labs

Fabian ´fabs´ Yamaguchi
fabs@recurity-labs.com

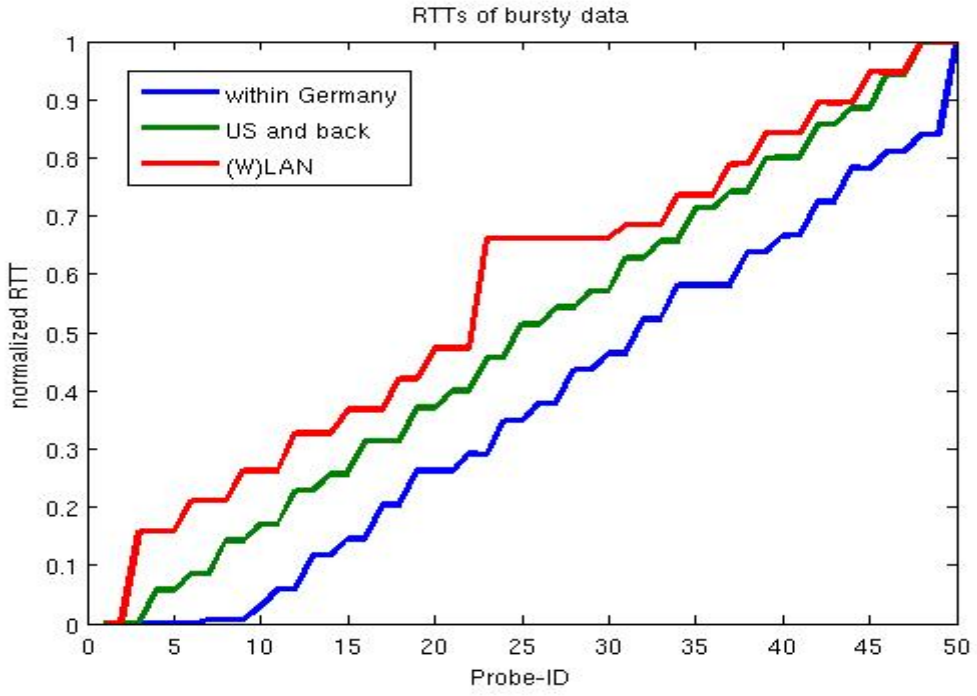
Felix ´FX´ Lindner
fx@recurity-labs.com

Recurity Labs GmbH, Berlin, Germany
http://www.recurity-labs.com

Invent & Verify

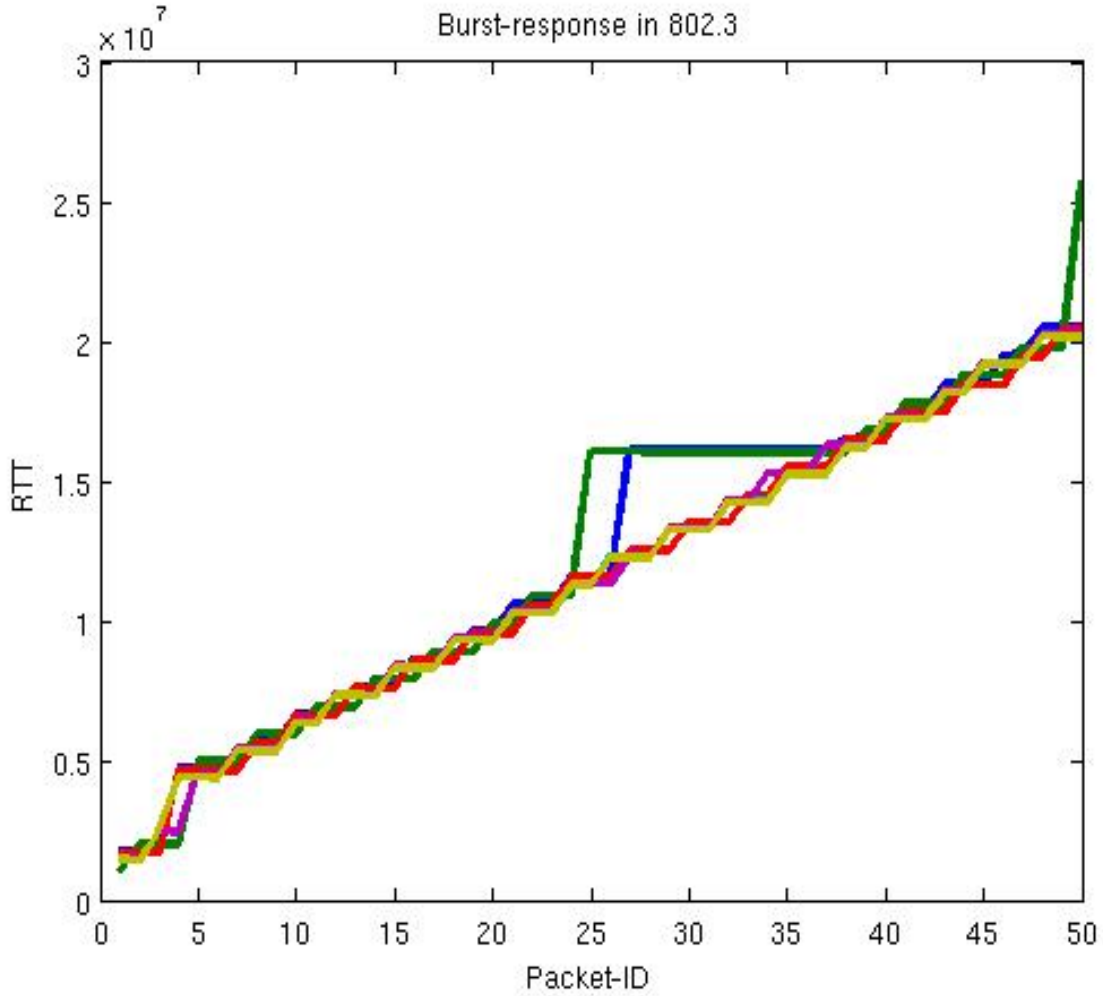
One more cool trick ;)

- … this “burst-response” just looks so pretty, is there really nothing we can do with it?



Invent & Verify

Burst-response on wired Ethernet



error [] =

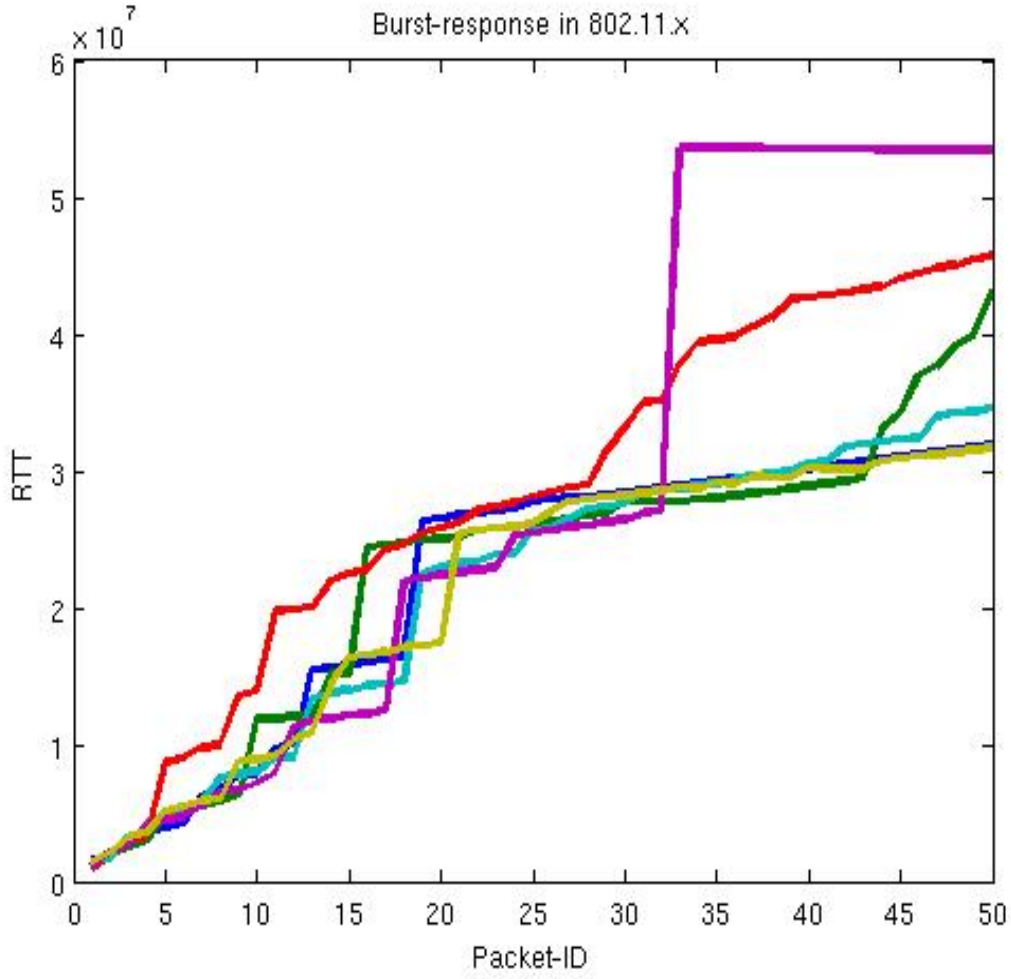
0.0589
0.0737
0.0322
0.0439
0.0421
0.0422

Mean =

0.0488

Invent & Verify

... and on wireless ☺



error [] =

0.1706
0.0823
0.1154
0.1052
0.0935
0.1578

Mean =

0.1208

'Invent & Verify

Problems with this approach

- Watch out! If the test-result is positive, we know that there is a rate-limiter. If it's negative, we don't know anything!
  - One bucket, which cannot hold the entire burst at once could be the limit.

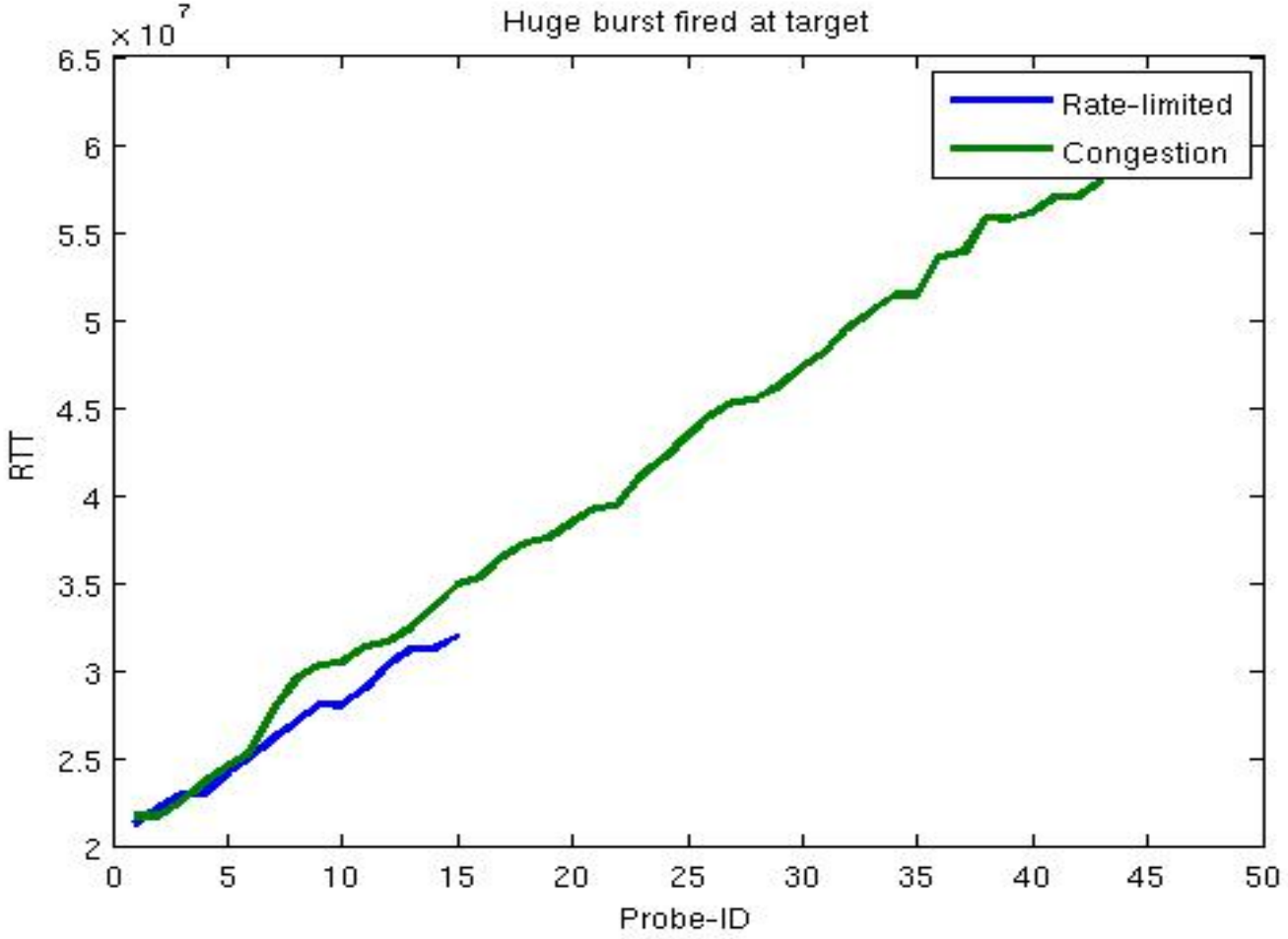
Two types of solutions

- One/Few-Shot Solutions:
  - Is it possible to detect the rate-limiter with a few shots of data?
- At-Scan-Time Solutions
  - Is it possible to detect the rate-limiter by observing the network's behavior during the scan?

Burst-response

- Can we just send a burst of triggers, which is big enough and measure the RTT to classify the bottleneck?
- RTT-increase is exponential => congestion
- Else: Firewall

Nope, doesn't work for bursts



So, were the books wrong?

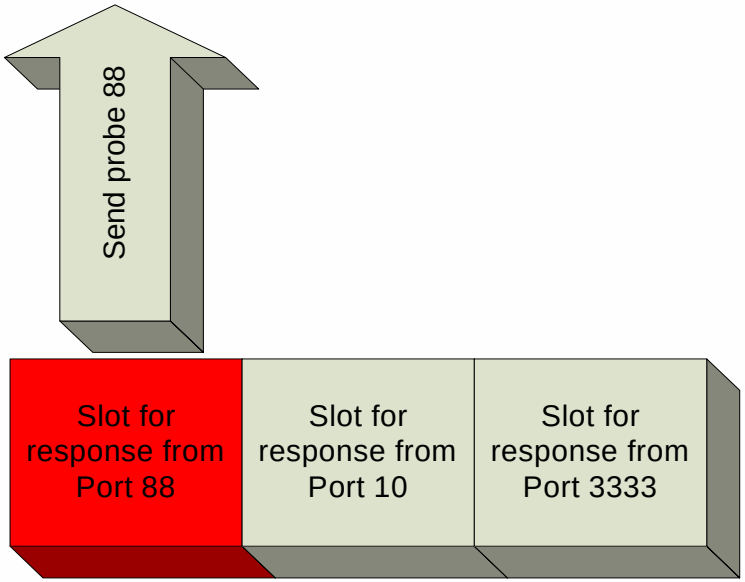
- **Result is only valid if data is offered at a constant rate over a certain time.**
- Scanning does mean offering data at a certain rate over a longer time-period.
  - Detection could be done during the scan!
  - But the rate is constantly changed by the timing-algorithm in use, will measurements be clear enough?

Consequence

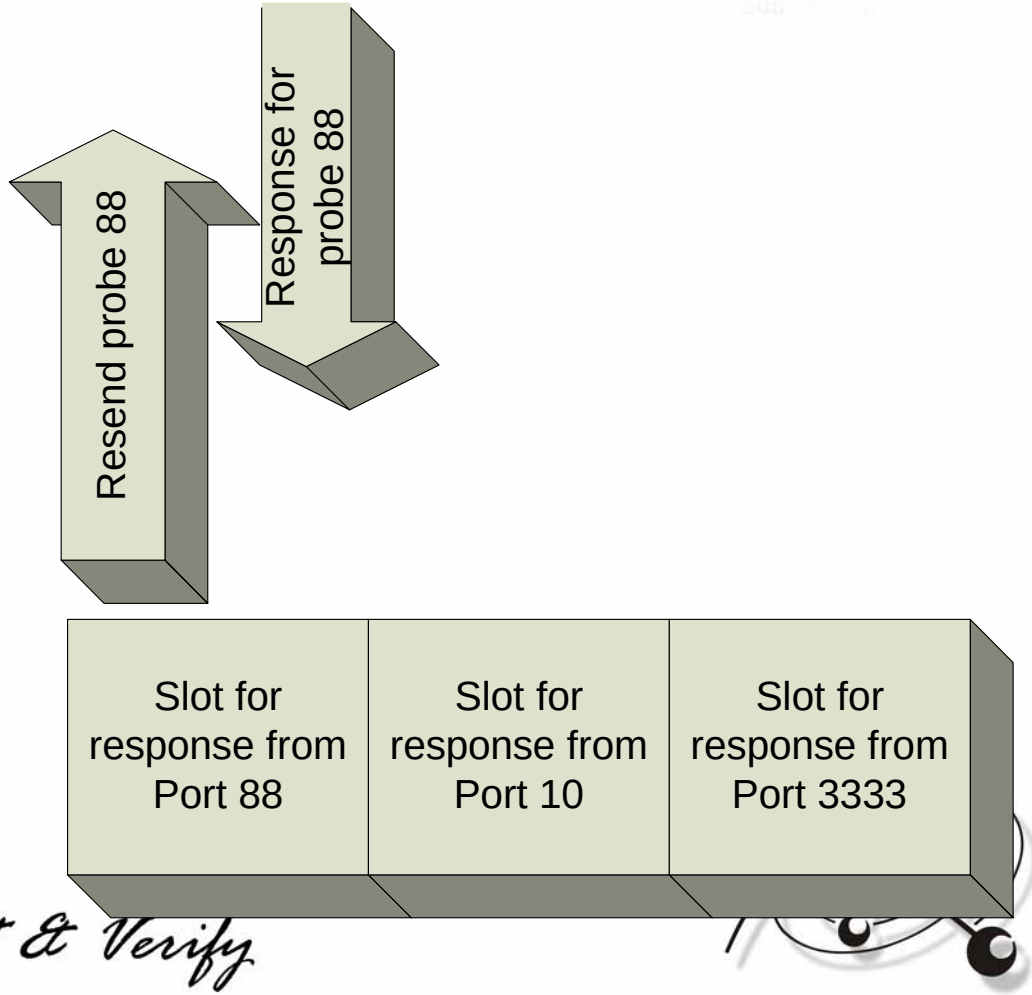
- To stay responsive to drops, probes that may have just dropped must be resent straight away!
- This makes you extremely vulnerable to the “late-responses”-problem
- (… and to “port-cloaking”, btw)

“Late-responses” Problem

(1)



(2)



If the approximation of the timeout is too optimistic, responses arrive shortly after the resend has occurred.

➔ **Lots of unnecessary traffic which reduces the scanning-speed.**

“Fast Retransmit”

- When integrating sequence-numbers into triggers, an algorithm similar to **fast-retransmit** can be implemented:

Trigger-Response 5
Trigger-Response 6 MISSING
Trigger-Response 7
Trigger-Response 8
Trigger-Response 9

Example:

- **Responses for 7, 8 and 9 have been received but there’s no response for 6.**
- **One can assume that 6 has been dropped even if its timeout-value has not been reached!**

Invent & Verify