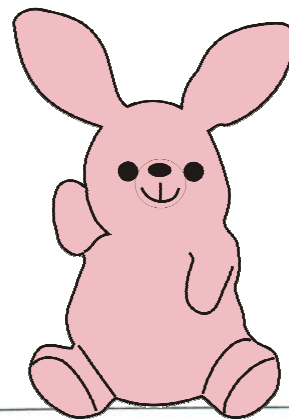




```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t8, 4
sllv $i, $v0, $t9
beqz $i, loc_2DA24
nop
sub_2DAB8
```

PortBunny

A kernel-based port-scanner



```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C
```

Invent & Verify

Copyright © 2007 Recurity Labs GmbH

A Port Scanner? *Yawn*

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal $ra, DAB8
lw $i, dword_35A6C
li $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2DA24
nop
sub $t10, $t10, 1

```

- 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

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_DAB8
addiu $a1, $v0, 1
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, 1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Why not nmap?

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2DA24
nop
sub_2DAB8

```

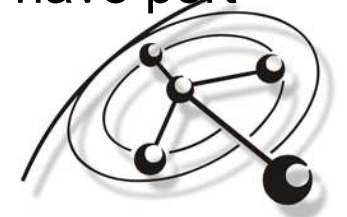
- 3 * 255 Hosts in 30 days with nmap
 - I'm actually coming of age
 - Your scanner is not 1337 if it takes 13:37 per host!
 - No, **--disable-waiting-for-things-that-dont-happen** doesn't cut it
- Professionals don't scan hosts that are ...
 - ... powered off
 - ... disassembled
 - ... currently being 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

```

move $a0, $v0
lw $a1, $v0
jal sub_2DAB8
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA24
move $v0, $0
la $1, dword_35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



And on a more abstract level...

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_35A24
null
```

- 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

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 0x30
beqz $v0, loc_35A44
move $v0, $a1
la $t1, dword_35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



Our Requirements

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2DA24
nop
sub 70000
```

- 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.

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a0, $a0, 0
beqz $v0, loc_2DA24
move $v0, $0
la $t1, 0
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t1, $t0, 2
sra $t1, $t1, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



PortBunny

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2DA24
nop
sub 2DAB8

```

- Portbunny scans faster by sending more
- Portbunny builds a bridge between **TCP congestion control** and port-scanning.
- Portbunny shows that vanilla TCP-SYN port-scans already leave you with lots of room for research.

```

move $a0, $v0
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a0, $a0, 4
beqz $v0, loc_2DA44
move $v0, $0
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($1)
sw $v0, dword_35A6C

```

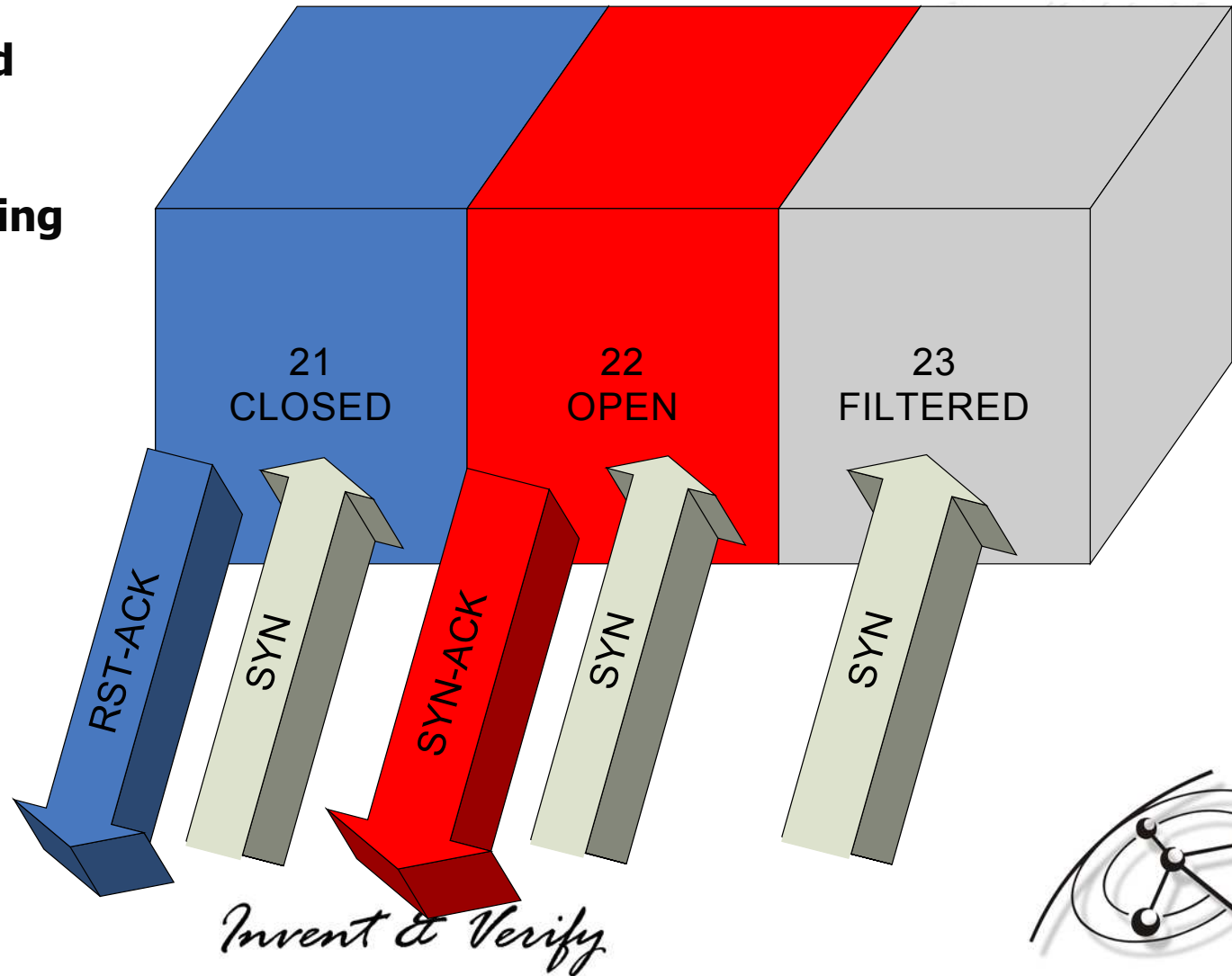
Invent & Verify



1. Port-Scanning - Basics

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal sub_35A68
sw $t1, dword_35A6C
lui $t7, 0x10
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t8, 0
```

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



(TCP-SYN or "half-open"-scanning)

```
move $a0, $t3
lw $t4, 0($a0)
jal sub_7DAD4
addu $t5, $t4, 0x10
beqz $t5, 10c_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Naive port-scanner

```

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

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sll $i, $v0, $t9
beqz $i, loc_2DA24
nop
sub_2DAB8
  
```

- 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.
- Optimal speed may change over time!

```

move $a0, dword_35A6C
lw $ra, dword_35A6C
jal sub_2DAD4
addiu $a0, $ra, 4
beqz $v0, $v0
move $v0, $0
la $i, dword_35A70
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C
  
```

Invent & Verify



Tell us to slow down, please.

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
$t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
beq $i, loc_2DA24
sub $t10, $t6, $t7

```

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

A: In general, no.

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

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```

Invent & Verify



What info do we have?

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t3, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_2DA24

```

- 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.

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

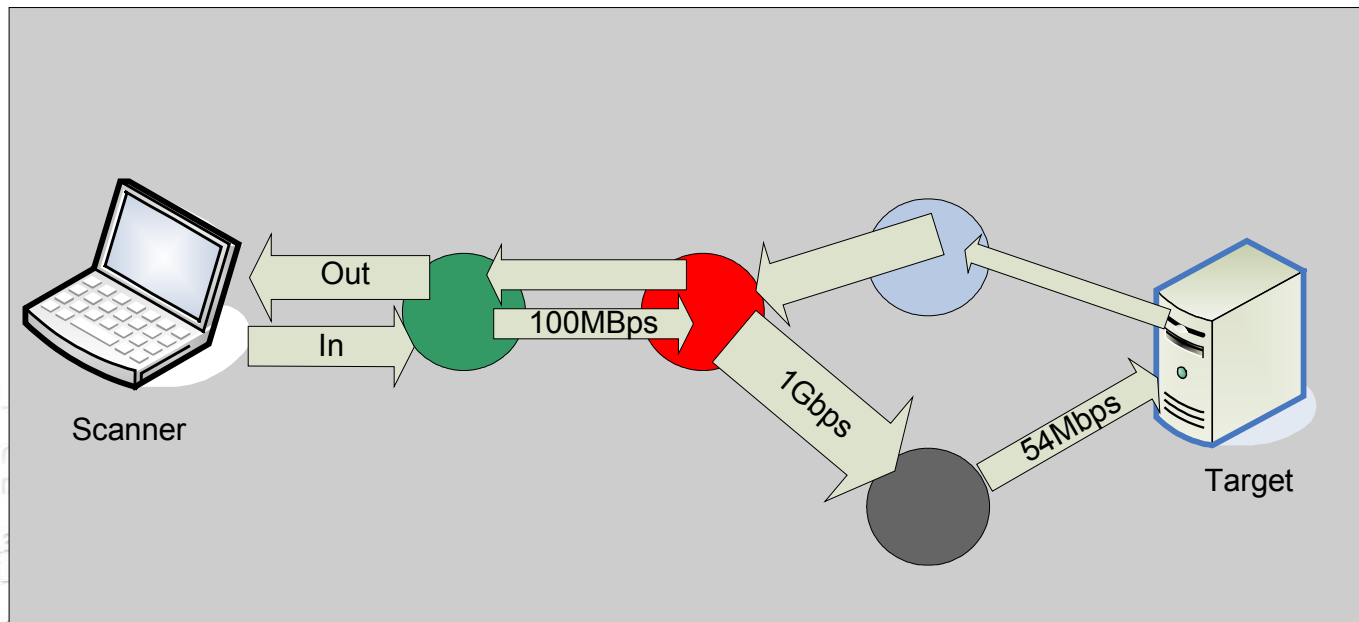
- That's all.

Invent & Verify



2. A network model

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



```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 1
beqz $v0, loc_2DAD4
move $v0, $0
la $t1, dword_35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
sub_2DAB8 $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sll $t1, $v0, $t9
beqz $t1, loc_2DAD4
nop
sub_2DAB8
```

Invent & Verify



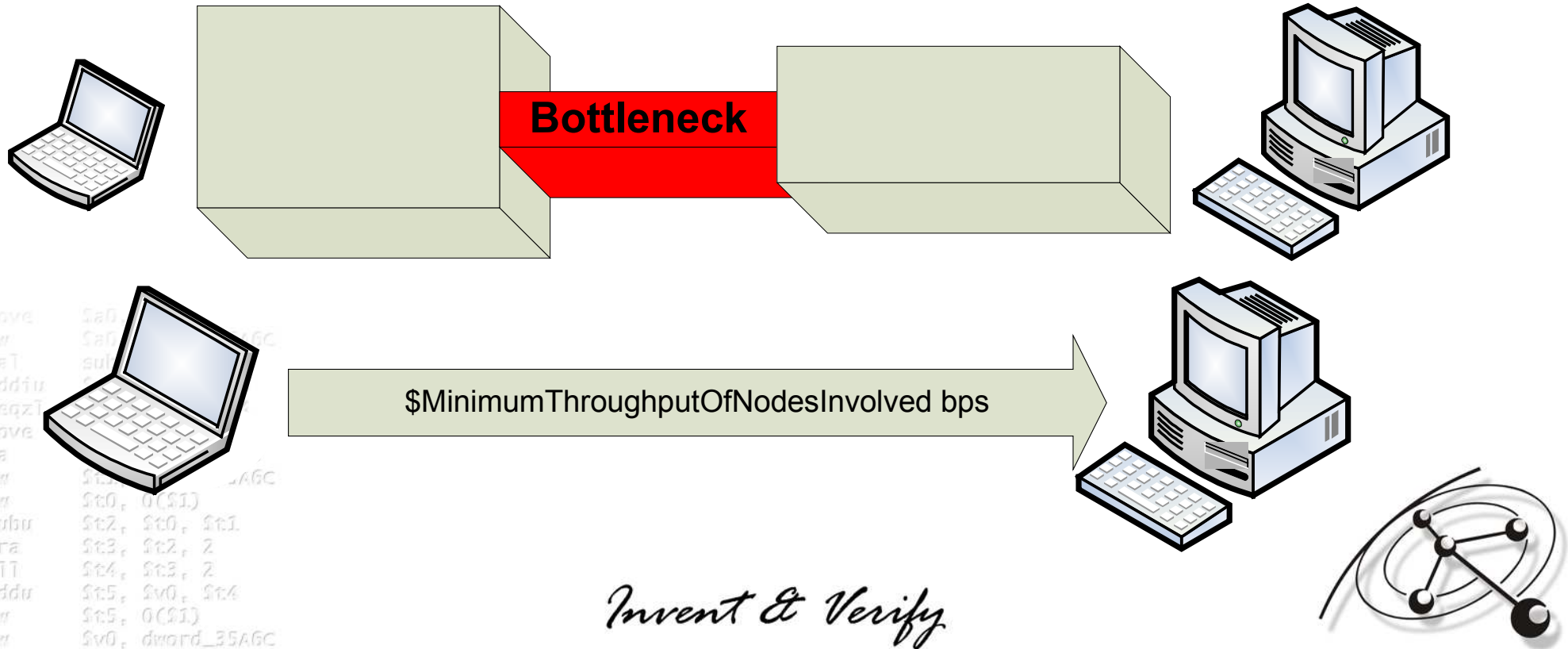
Simplification

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2DA24
nop

```

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



```

move $a0, dword_35A6C
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $t1, $a0, 1
beqz $t1, loc_2DA24
move $t2, $a0
lw $t3, dword_35A6C
lw $t4, 0($t1)
subu $t5, $t3, $t4
sra $t6, $t5, 2
sll $t7, $t6, 2
addu $t8, $t5, $v0, $t6
sw $t8, 0($t1)
sw $v0, dword_35A6C

```

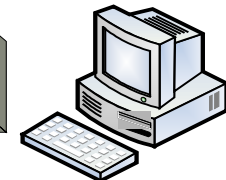
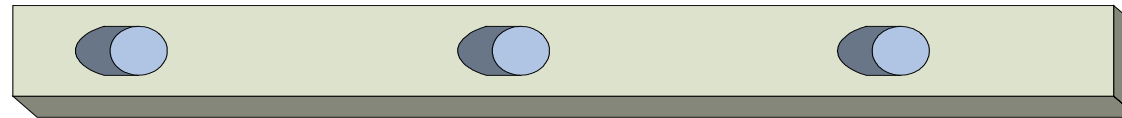
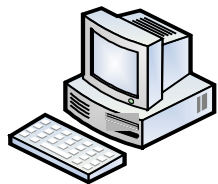
Optimal speed

```

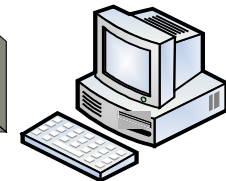
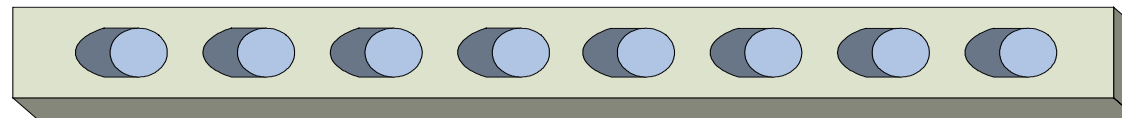
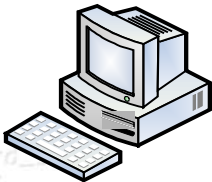
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2D624

```

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



slow

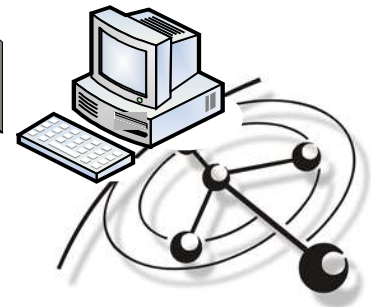
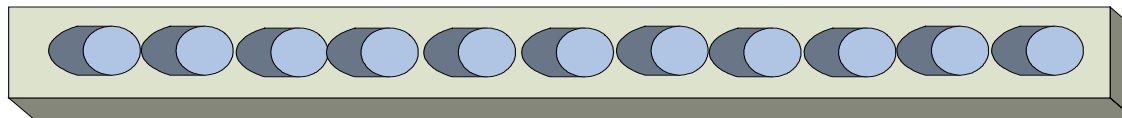
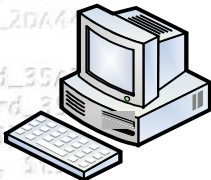


faster

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```



Optimal speed

Invent & Verify

So much for theory...

- ... but finding the optimal delay will fail in practice!

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
$1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sll $t1, $t6, $t9
bne $t1, $t6, loc_2DA24
m...
sub...

```

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

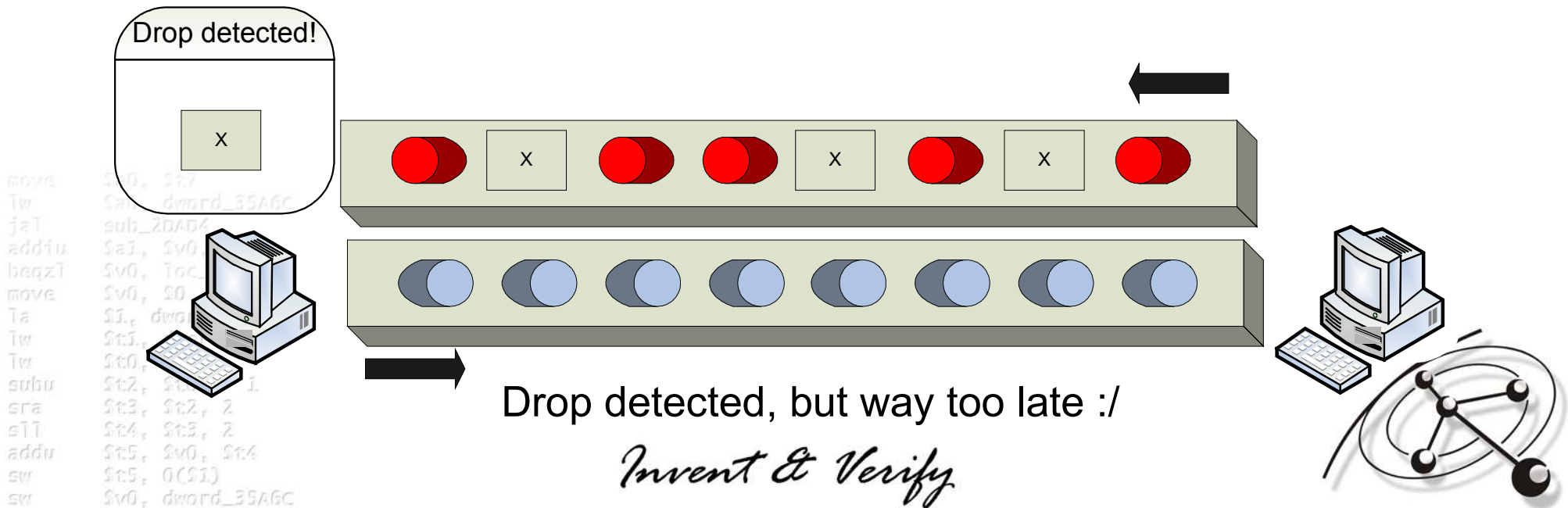
Invent & Verify



The round-trip-time problem

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
sub 2D456
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2D424
sub 2D456
```

- Dropped packets can't be detected before a complete round-trip-time has passed.
- At that time about rtt/delay other packets have already been sent to maintain the “optimal delay”.



```
move $t0, $t7
lw $a1, dword_35A6C
sub 2D4D4
addiu $a1, $v0
beqz $v0, loc_2D424
move $v0, $t0
la $i, dword_35A6C
lw $t1, $t0
lw $t0, $t1
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Queuing capacity

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAE8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
lw $t10, loc_2DA24

```

- “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.

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAE8
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```

Invent & Verify



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.

```

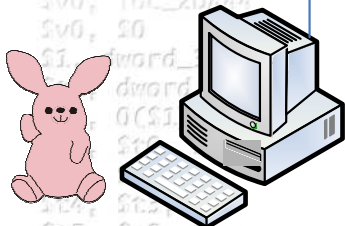
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t3, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2D624
nop
sub_2D624

```

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2D624
move $v0, $0
la $1, dword_35A6C
lw $t7, dword_35A6C
lw $t6, 0($t1)
subu $t8, $t6, $t7
sra $t4, $t5
sll $t4, $t4, $t3
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```



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?”

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
sub $t0, $t1, 2
lw $t2, dword_35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t3, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_2DA24
nop
sub $t0, $t0
```

```
move $a0, $t7
lw $a0, $t7
jal $t0, loc_2DA44
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



TCP Congestion Control

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal sub_2DAB8
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_2DA24
nop
sub 2DAB8
```

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

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



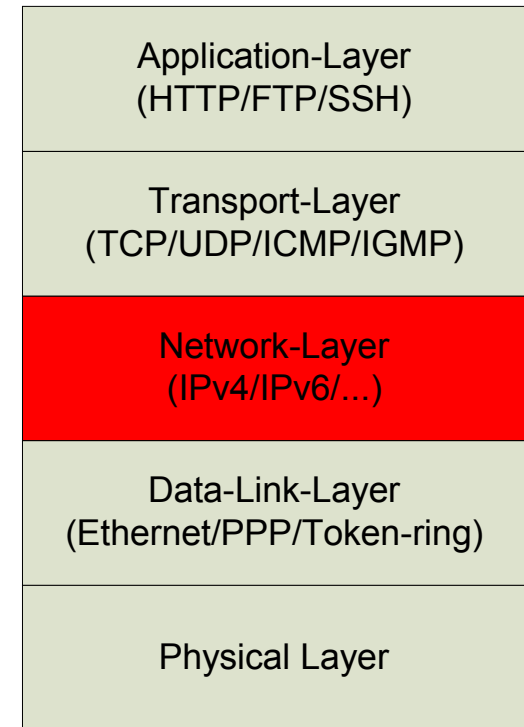
Doesn't that work automatically?

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2DA24

```

- Why do we have to implement congestion control at all?
- Doesn't TCP provide congestion control to upper layers?
- **No established TCP-connection**
- Control the emission of IP-packets which happen to be TCP-SYNs.



```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a0, 0
beqz $v0, loc_2DA44
move $v0, $0
la $t1, 0
lw $t1, 0($t1)
lw $t2, 0($t1)
subu $t3, $t2, $t1
sra $t3, $t3, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



TCP vs. Port-Scanning

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal sub_2DAB8
$a0, dword_35A6C
$t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_2DA24
nop
sub_2DAB8

```

TCP

Receiver acks packets.

Timeouts are error-conditions

Sequence-numbers are used

Port-Scanning

Packets may not produce answers.

Timeouts are not error-conditions

No sequence numbers

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 1
beqz $v0, loc_2DA24
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A70
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 8
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



... in other words:

```

addiu $sp, -0x18
sw    $ra, 0x18+var_4($sp)
sw    $a0, 0x18+arg_0($sp)
lui   $i, 3
jal   sub_2DAB8
lw    $a0, dword_35A6C
lui   $i, 3
lw    $t7, dword_35A6C
lw    $t6, dword_35A70
subu  $t8, $t6, $t7
addiu $t9, $t6, 4
sll   $i, $v0, $t9
beqz  $i, loc_2DA24
nop
    sub_2DAB8

```

- 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.

```

move   $a1, 0x7
lw     $a0, dword_35A6C
jal    sub_2DAD4
addiu  $a1, $v0, 0x10
beqz   $v0, loc_2DA44
move   $v0, 20
la     $i, dword_35A70
lw     $t1, dword_35A6C
lw     $t0, 0($i)
subu   $t2, $t0, $t1
sra    $t3, $t2, 2
sll    $t4, $t3, 2
addu   $t5, $v0, $t4
sw     $t5, 0($i)
sw     $v0, dword_35A6C

```

Invent & Verify



Triggers - forcing cooperation

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lui $t7, dword_35A6C
$t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2DA24
sub 2DA24

```

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

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA64
move $v0, 0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```

Invent & Verify



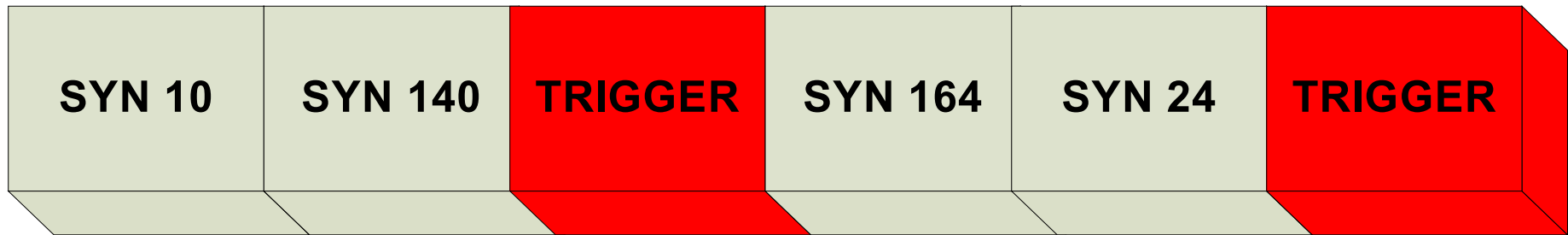
Inserting triggers into the probe-stream

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
st7, dword_35A6C
st6, dword_35A70
subu $t8, $t6, $t7
addiu $t3, $t6, 4
sllv $i, $v0, $t9
mov $i, loc_2DA24
sub $t3, $t3

```

- Insert these packets into the packet-stream and base your timing-code on the triggers



```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```

Invent & Verify



What's that good for?

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
feq $a0, dword_35A68
sw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_2DA24

```

- 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!

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DA14
addiu $a2, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, 20
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



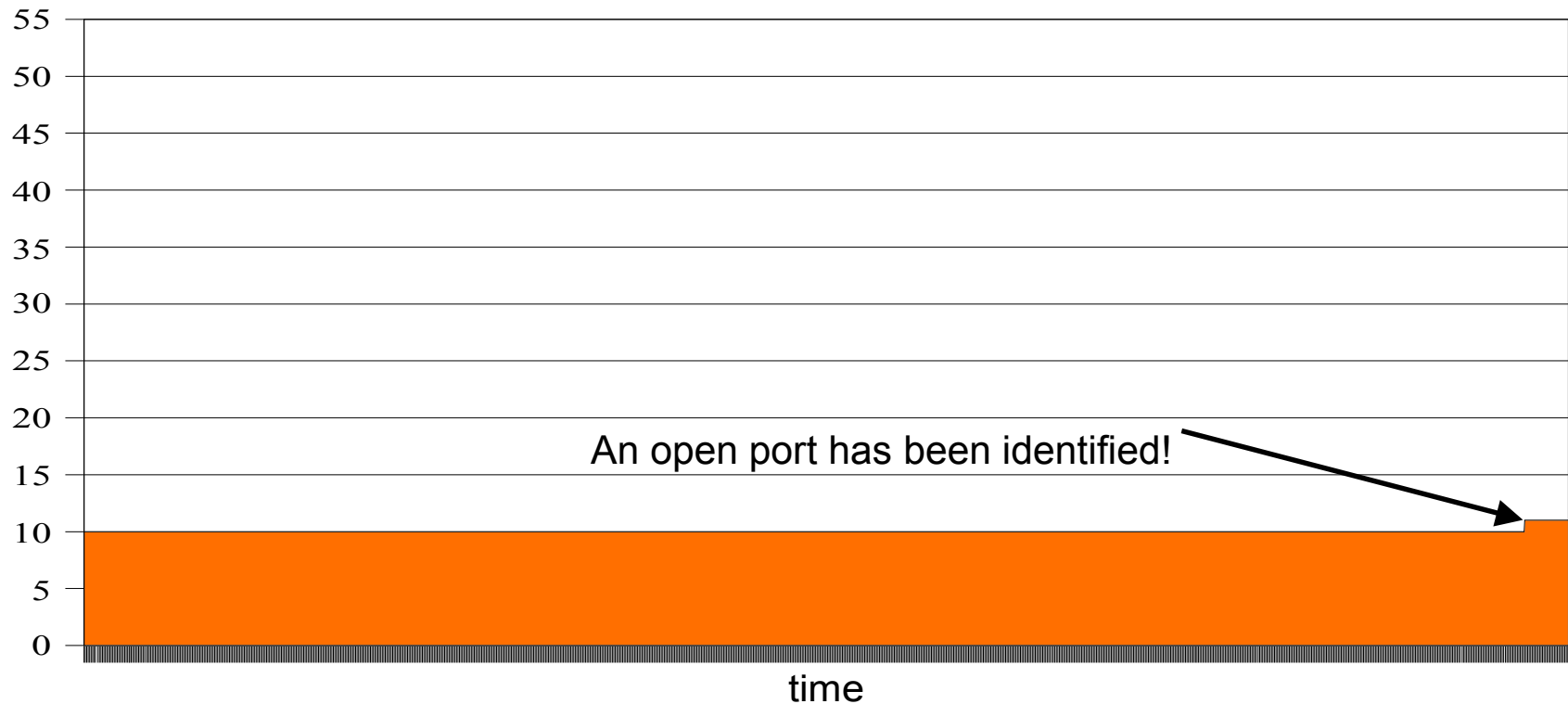
What nmap forgot.

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t3, $t6, 4
sllv $t1, $v0, $t9

```

NMAP scanning a mostly filtered host

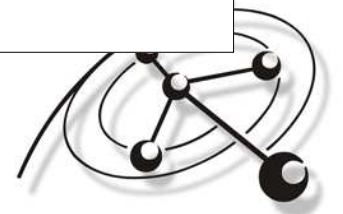


```

move $t2, $v0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



But let's be fair:

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sll $i, $v0, $t9
lw $i, loc_2DA24
sub 2DAB8

```

```

/* When a successful ping response comes back, it
counts as this many "normal" responses, because the
fact that pings are necessary 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

```

move $a0, $t0
lw $ra, dword_35A6C
jal sub_2DAD4
addiu $a0, $t0, 1
beqz $v0, $t0
move $v0, $t0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



... and then there are filtered hosts ☺

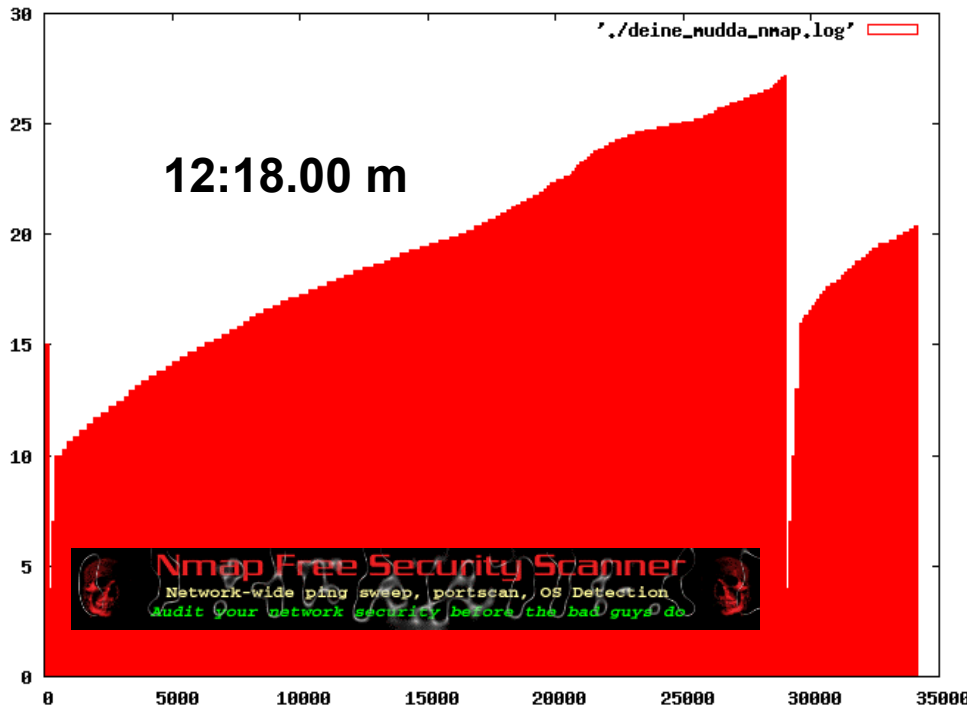
```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
$ra, 0x18+arg_0($sp)
jal $li, 3
jal sub_2DAB8
lw $ra, dword_35A6C
lui $li, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $t9, $t9, $t8
beqz $t9, loc_2DA24
nop
sub 707C0

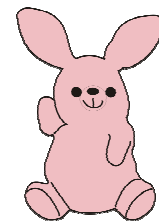
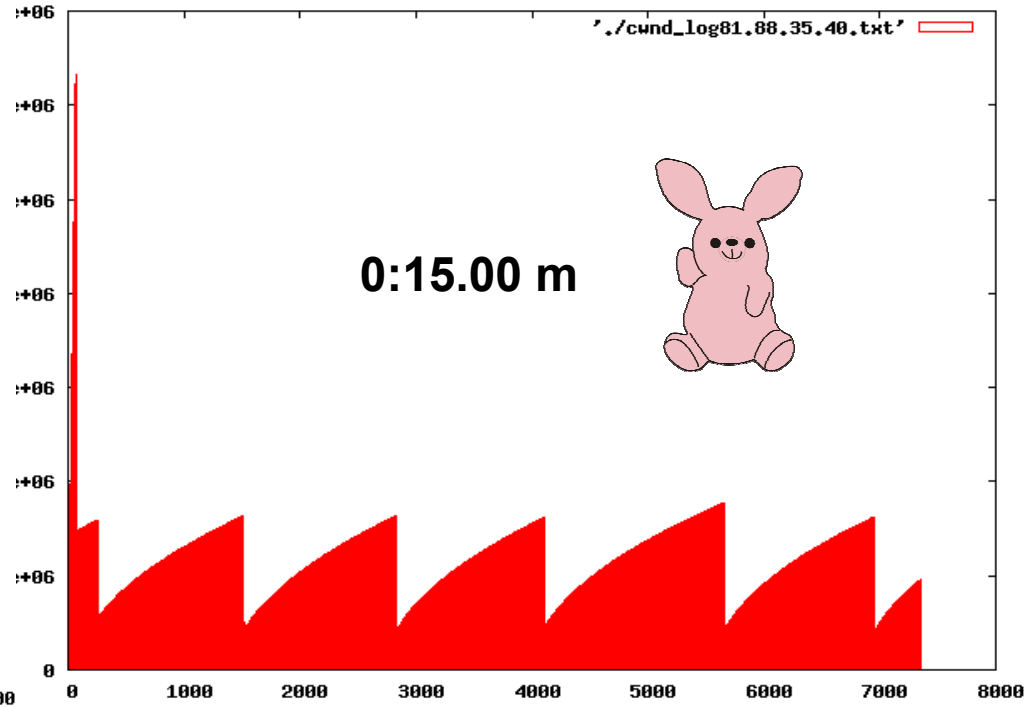
```

- 65535 ports, mostly filtered, Internet.

NMAP CMD development



PortBunny CMD development



```

mo
lw
ja
add
be
mo
la
lw
lw
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```



Invent & Verify



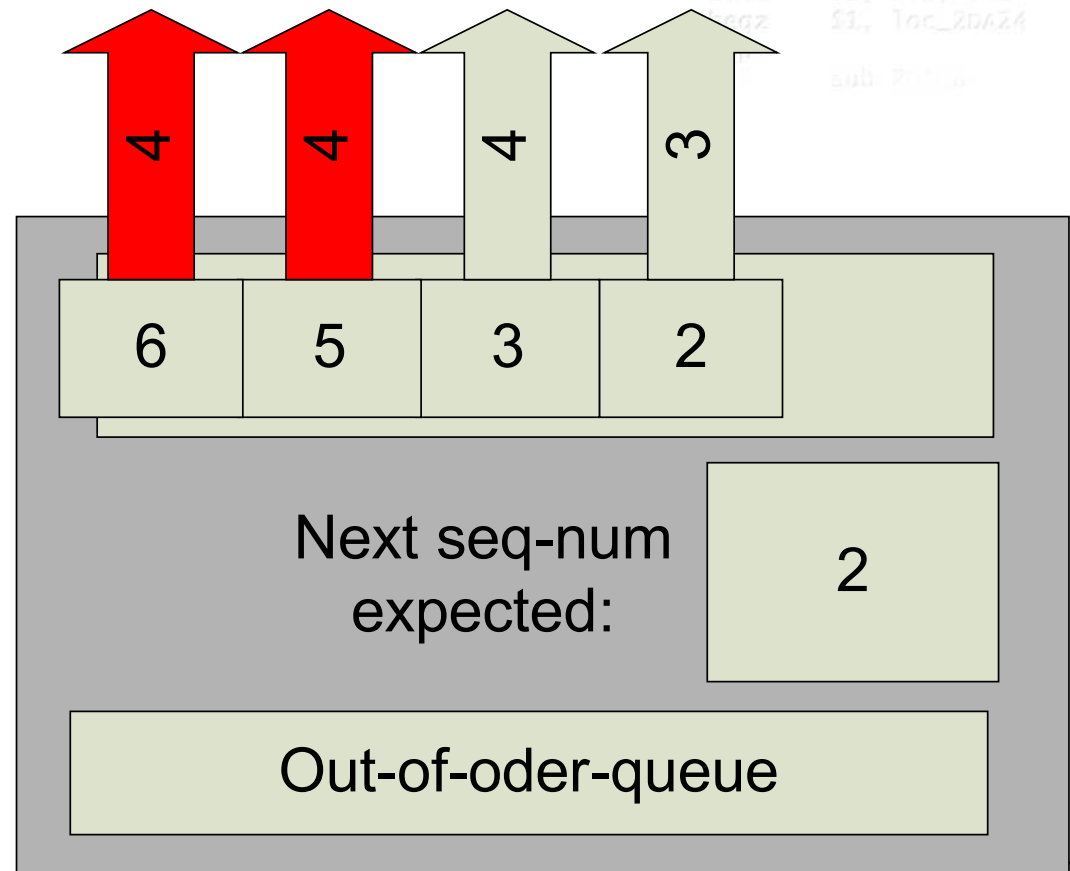
Why mention Sequence-Numbers?

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
li $i, 3
li $t7, dword_35A6C
li $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $t6, $t9
seqz $i, loc_2DA24
sub 7070

```

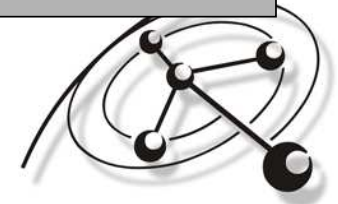
- An Ack is sent by the receiver for each packet
- Duplicate Acks indicate packet-loss!
- Fast-retransmit



```

move $a0, $t7
lw $a1, dword_35A6C
jal sub_2DA44
addiu $a2, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 31
sll $t4, $t3, 31
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```



Trigger Sequence-Numbers

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
fdiv $f12, $f12, $f12
lw $t2, 0x35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sll $t1, $t0, $t9
lwr $t1, 100_20A24
sub $t10, $t10, $t1
```

- 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

```
move $a0, $a0
lw $a0, $a0
jal sub
addiu $a1, $a1
beqz $v0, $v0
move $v0, $v0
la $t1, $t1
lw $t1, $t1
lw $t0, $t0
subu $t2, $t2
sra $t3, $t3
sll $t4, $t4
addu $t5, $t5
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

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



Timeout-detection without triggers

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sll $1, $v0, $t9
beqz $1, loc_2DA24
nop
sub_2DAB8

```

`/* 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.
- Each probe has its own timeout-clock. That doesn't scale well.

```

move $1, $v0
lw $t0, dword_35A70
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $1, $t0, $t0
move $t1, $a1
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t2, 0($t1)
subu $t3, $t2, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Consequence

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2DA24
nop
sub_2DAB8

```

- 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

```

move $a0, $v0
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 1
beqz $v0, loc_2DA44
move $v0, $0
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($1)
sw $v0, dword_35A6C

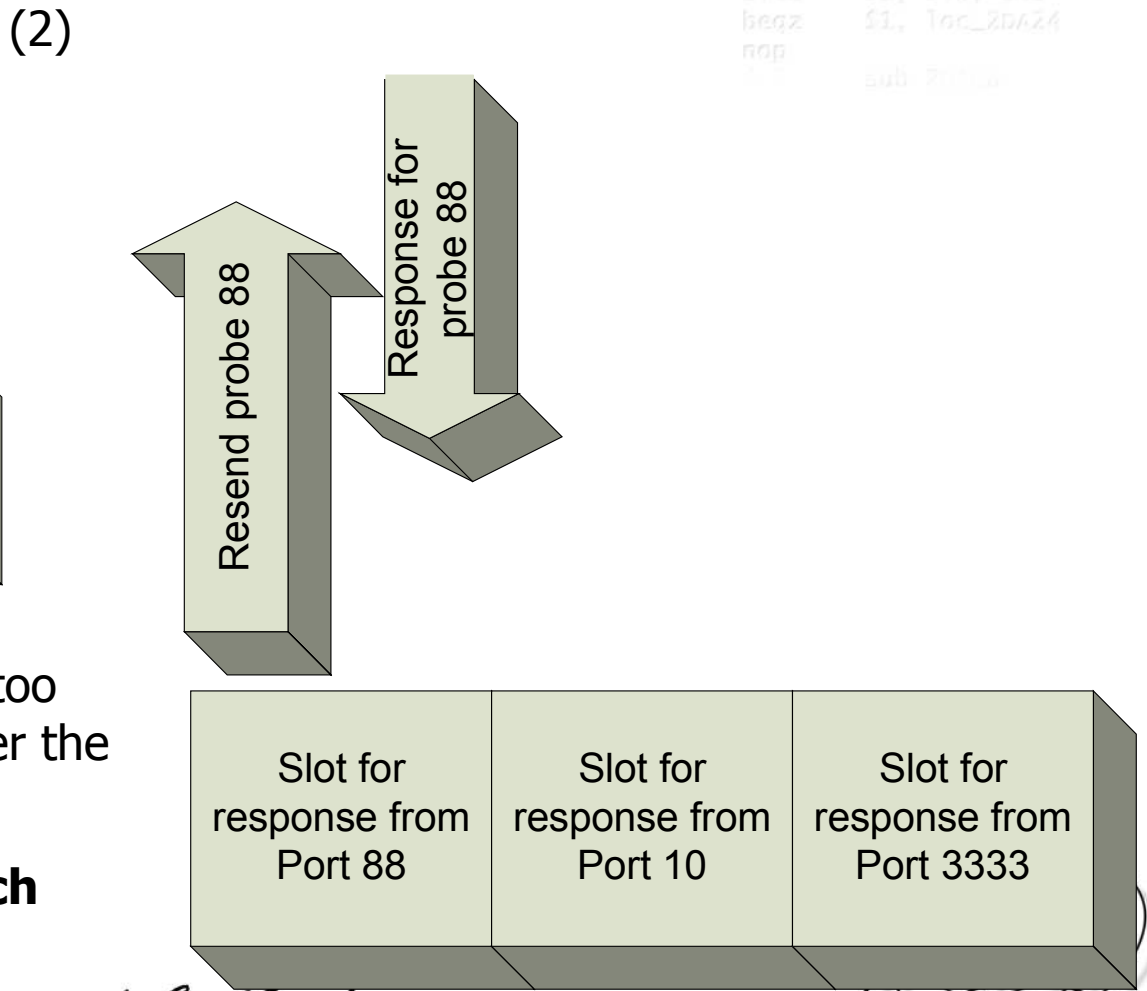
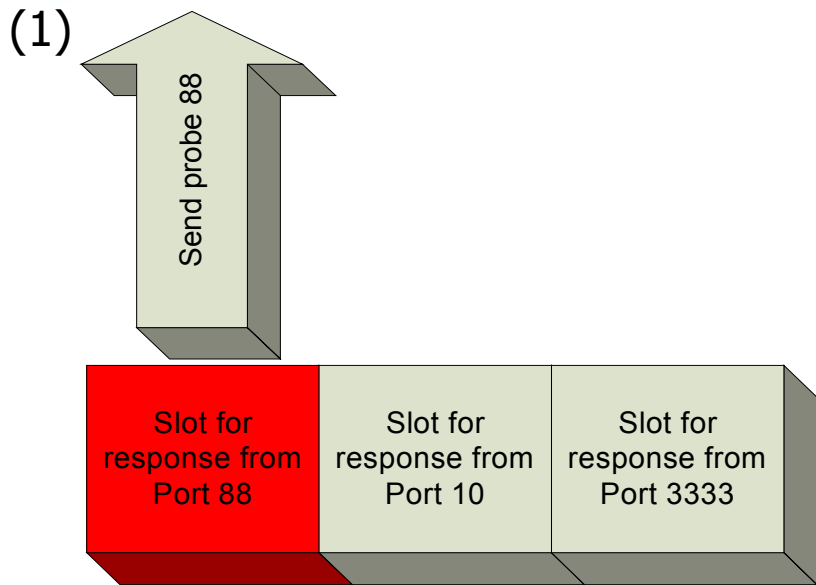
```

Invent & Verify



“Late-responses” Problem

```
addiu $sp, -0x18  
sw $ra, 0x18+var_4($sp)  
sw $a0, 0x18+arg_0($sp)  
lui $t1, 3  
fa $t1, 35A58  
lw $t7, dword_35A6C  
lw $t6, dword_35A70  
subu $t8, $t6, $t7  
addiu $t9, $t6, 4  
sllv $t1, $t0, $t9  
beqz $t1, loc_20A24  
nop  
sub 20A24
```



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.

Invent & Verify

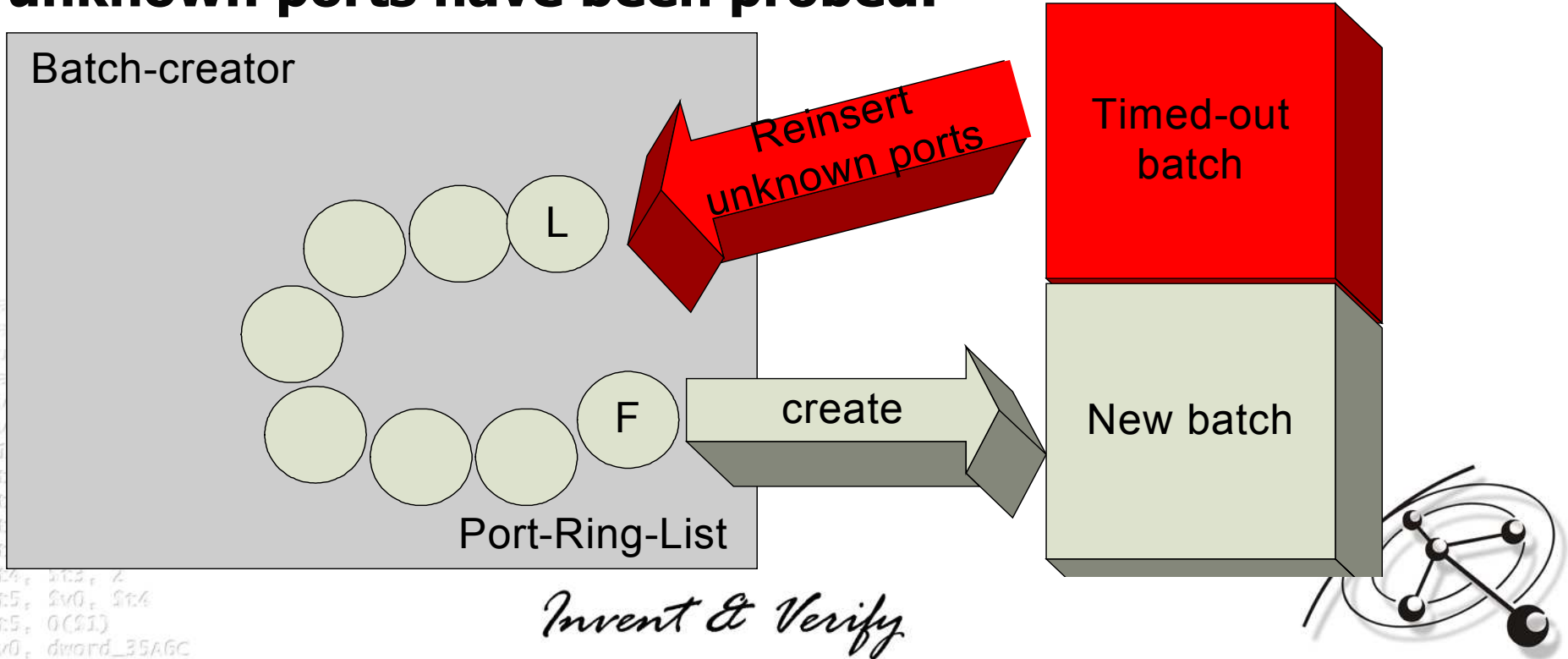
```
move $a0, $t7  
sw $a0, dword_35A6C  
subu $t1, $t0, $t7  
lw $t1, dword_35A6C  
lw $t6, dword_35A70  
subu $t8, $t6, $t7  
addiu $t5, $t0, $t4  
sw $t5, 0($t1)  
sw $t0, dword_35A6C
```

Defeating late-responses (with triggers)

```
addiu $sp, -0x18  
sw $ra, 0x18+var_4($sp)  
sw $a0, 0x18+arg_0($sp)  
lui $i, 3  
jal sub_2DAB8  
lw $a0, dword_35A6C  
lui $i, 3  
lw $a0, dword_35A6C  
lw $a0, dword_35A70  
subu $t8, $t6, $t7  
addiu $t9, $t6, 4  
sll $i, $v0, $t9  
lw $i, loc_2DA24  
sub $t8, $t8
```

PortBunny does not rely on immediate resends to detect packet-loss!

→ The probe can be resent after ALL other unknown ports have been probed!



```
move $a, $t5, 4  
lw $a, $v0, $t4  
jal addiu $a, 0($t1)  
beqz $i, $v0, $t4  
move $a, $v0, $t4  
la $t, $t  
lw $t, $t  
lw $t, $t  
subu $t, $t, $t  
sra $t, $t  
sll $t, $t, 4  
addu $t, $v0, $t  
sw $t, 0($t1)  
sw $v0, dword_35A6C
```

Invent & Verify

Triggers vs. TCP

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7

```

TCP

Receiver acks packets.

Timeouts are error-conditions

Sequence-numbers are used

Trigger-based scanning

Triggers are acknowledged.

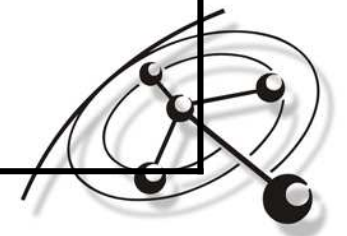
Trigger-Timeouts are error-conditions.

Sequence-numbers are used for all triggers.

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 1
beqz $v0, loc_2DAD4
move $v0, $0
la $1, dword_35A70
lw $t1, dword_35A70
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 8
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```



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!

Invent & Verify



```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal sub_2DAB8
sw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_2DA24
sub 7777
```

```
move $a0, $t7
lw $a0, dword_35A6C
sub 7777
addiu $a0, $t0
beqz $v0, loc_2DA44
move $v0, $0
la $t1, 0
lw $t1, 0
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t1, $t2, 2
sll $t4, $t5, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Problems with triggers

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal sub_2DAB8
sw $t0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_2DA24
nop
sub_2DAB8

```

- 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

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $t0
la $t1, 0x10+var_4($sp)
lw $t1, dword_35A6C
lw $t1, $t1
subu $t3, $t1, $t1
sra $t3, $t3, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

- A host may not respond to any supported trigger.

Invent & Verify



Fixes

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2DA24
nop
sub $t2, $t2, $t3

```

- Try to find TCP-SYN-triggers first and use ICMP and UDP-triggers 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.

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```

Invent & Verify



Racing on responsive hosts

```

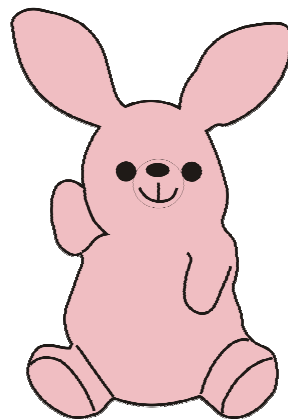
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAD4
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_35A74

```

- PortBunny sends 10% more data because of the triggers? Can it still compete with the standard tool NMAP on responsive hosts?



VS



Invent & Verify



```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```

Numbers and demonstration

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal $ra, sub_2DAB8
    sub_2DAB8:
    lw $t7, dword_35A6C
    lw $t6, dword_35A70
    subu $t8, $t6, $t7
    addiu $t9, $t6, 4
    sll $i, $v0, $t9
    and $t10, $i, loc_2DA24
    sub $t10, $t10, 1

```

- Fresh numbers will be included in the final slide-set, which you can download at <http://portbunny.recurity.com>

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```

Invent & Verify



Problems

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t3, $t6, 4
sllv $i, $v0, $t3
beqz $i, 2D674
sub
```

- The bucket-model is NOT valid for rate-limiting firewalls-configurations!
- We can implement any congestion-control-scheme designed for TCP **but we can't expect the user to know these algorithms and choose a suited one.**

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C
```

Invent & Verify



Algorithms implemented:

- Classic TCP-Reno
- TCP-Scalable
 - Slight Reno-improvement for long-distance networks
- TCP-BIC
 - for so called “long-fat pipes”
- TCP-Vegas
 - Experimental pro-active algorithm, which we want to use for WiFi.

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
sll $t1, $1, 28
lw $t2, dword_35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t3, $t6, 4
sll $t1, $t6, 2
beqz $t1, loc_2DA24
nop
sub $t1, $t1
```

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_35A70
addiu $a1, $a0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A70
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



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, not the user.

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_2DA24
sub_2DAB8
```

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



Trying to detect rate-limits

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2D624
sub_2D624

```

```

/* 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) */

```

Translates to: If packet-drops are particularly bad, break the entire timing-concept.

⇒ **The CWND will not reflect the number of probes out at once anymore!**

⇒ **The self-clocking-property is being ignored!**

```

move $a0, $t7
lw $ra, 0x18+var_4($sp)
jal sub_2D624
addiu $a1, $v0, 0x10
beqz $a1, loc_2D624
move $v0, 10
la $1, dword_35A70
lw $t0, dword_35A6C
lw $t1, dword_35A70
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($a1)
sw $v0, dword_35A6C

```

Invent & Verify



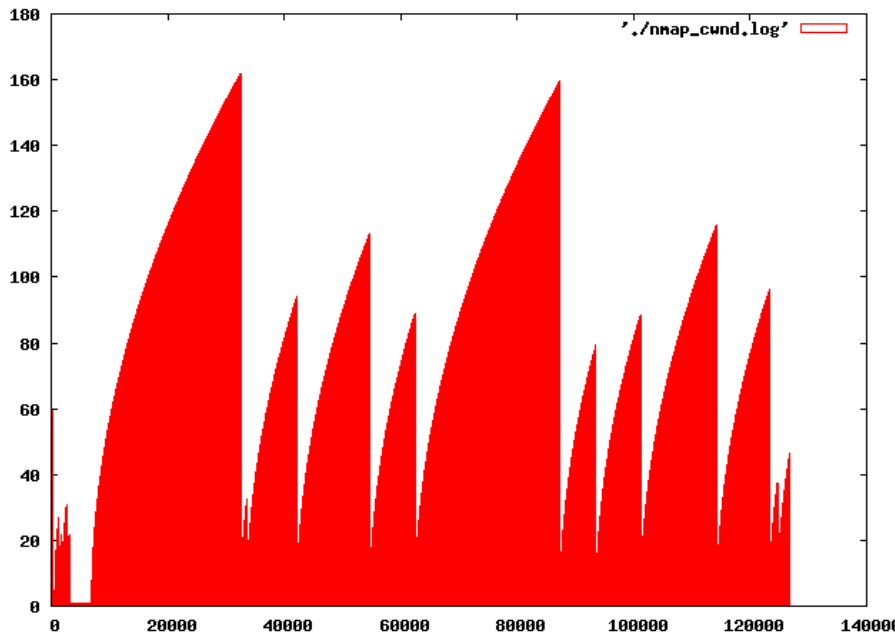
Scanning the IPHONE

```

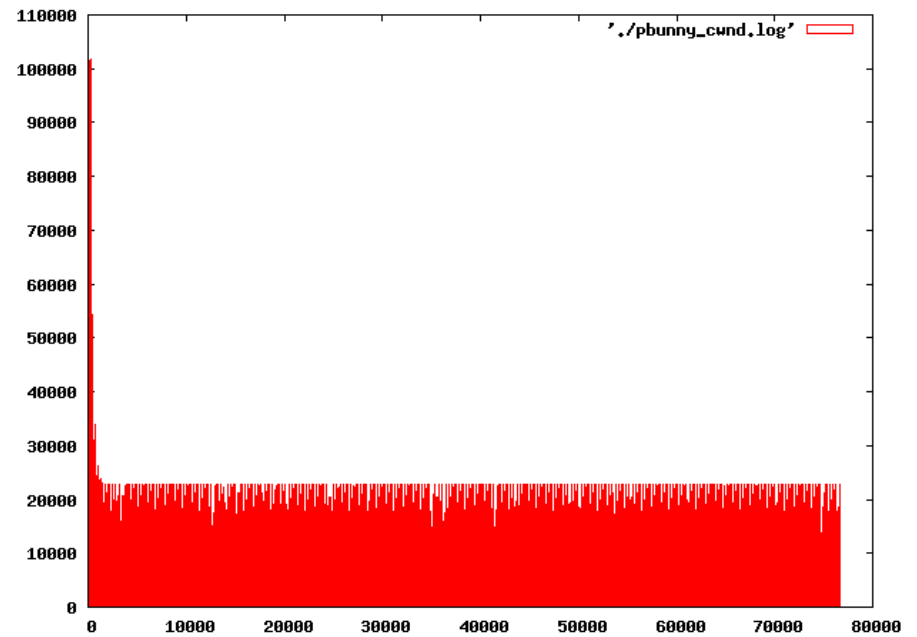
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
subu $t0, dword_35A68
$ra, dword_35A6C
$ra, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
slliu $t1, $t0, $t9
beqz $t1, loc_2DA24
nop
sub 20000

```

NMAP CHND development



NMAP CHND development



```

move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t7, dword_35A6C
subu $t8, $t6, $t7
sra $t4, $t3, 2
sll $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```



24:41.51 m

Invent & Verify

7:58.03 m

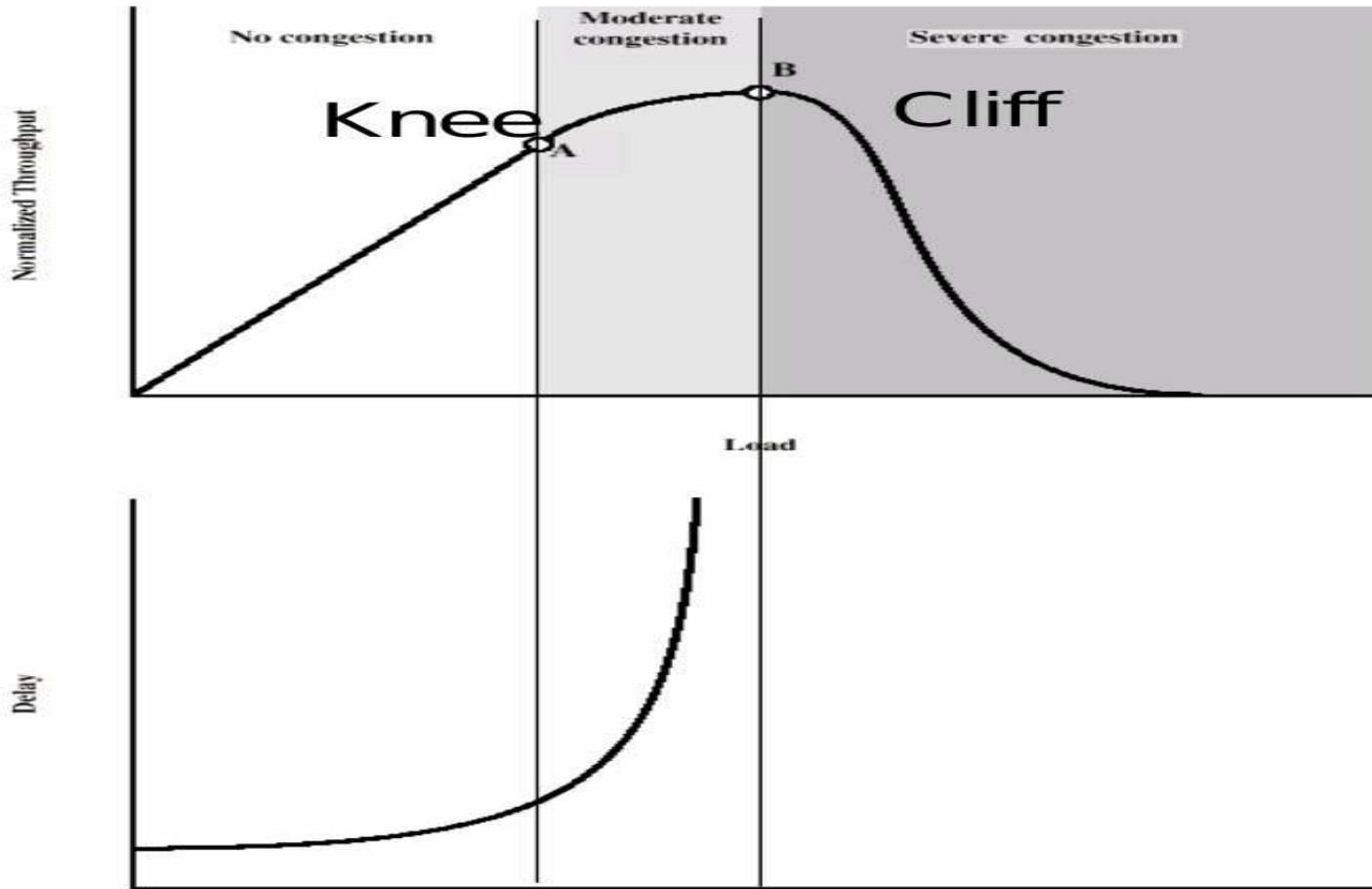


First approach: RTT-changes

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
-1... $1, $v0, $t9
$1, loc_2DA24
sub_2D7C8

```



```

move $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Exponential RTT-increase

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal sub_20658
lui $t0, dword_35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_20A24
nop
sub 20658
```

- Oh, that's easy! Just send a burst of data, which is big enough and measure RTT.
- If RTT-increase is exponential right before the drop, it's congestion, otherwise it's a rate-limiting firewall.

```
move $a0, $v0
lw $a0, dword_35A6C
jal sub_20658
addiu $a0, $a0, 1
beqz $v0, loc_20A44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify

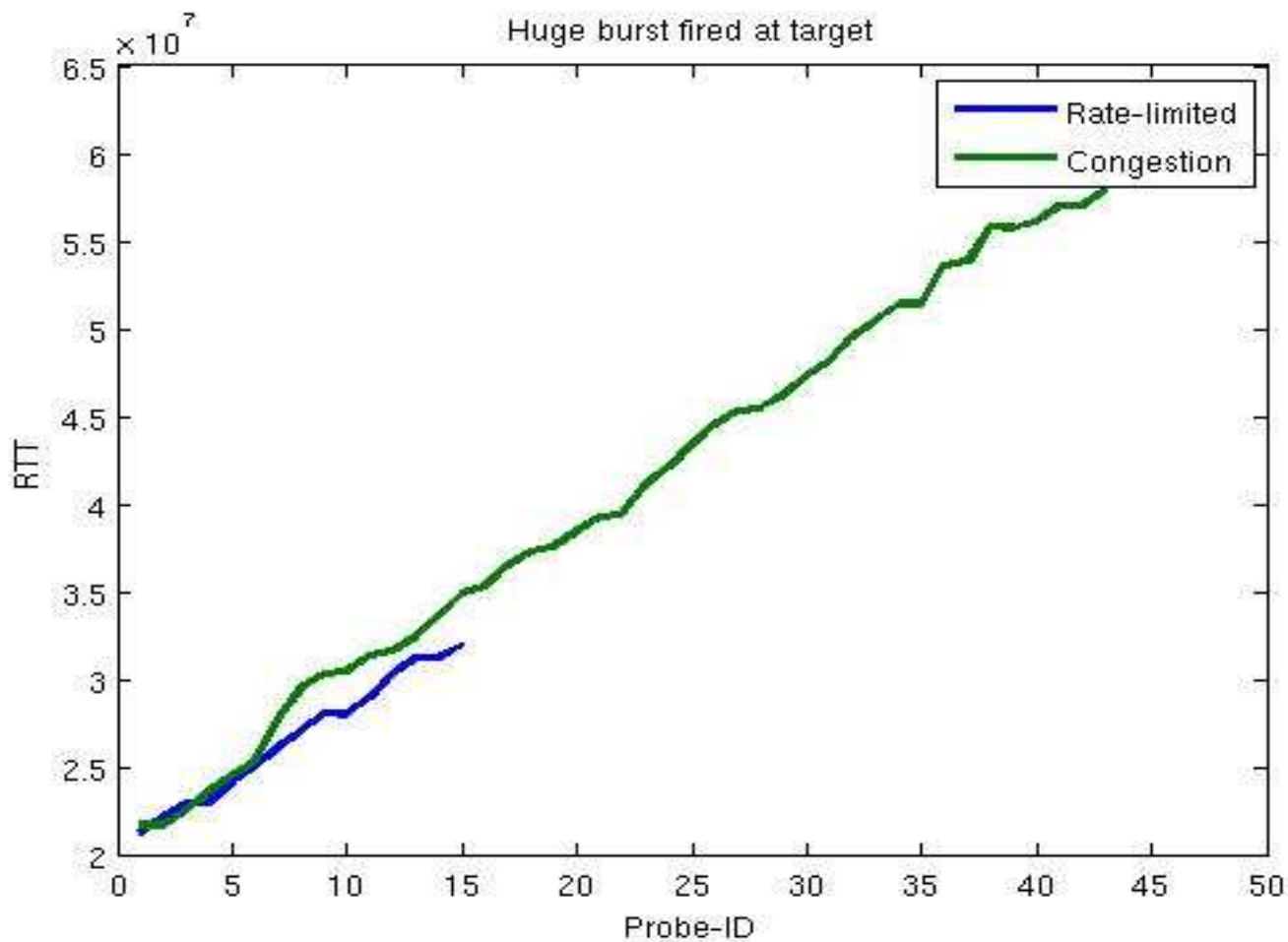


Nope, doesn't work for bursts

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
$a0, 0x18+arg_0($sp)
$1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sltu $1, $v0, $t9
l, loc_2DA24

```



```

move $a0, $
lw $a0, d
jal sub_2D
addiu $a1, $
beqz $v0, l
move $v0, $
la $1, dw
lw $t1, d
lw $t0, 0
subu $t2, $
sra $t3, $
sll $t4, $
addu $t5, $v0, $t4
sw $t5, 0($1)
sw $v0, dword_35A6C

```

Invent & Verify



What the hell?

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2DA24
sub_2DAB8
```

- Q: Where does this linear increase come from?
- A: That's a really pretty illustration of queuing.
- Experiment: Send a burst of 50 triggers to ...
 - (1) the next hop in a 100Mbit-LAN
 - (2) A host within your country over a DSL-Link.
 - (3) A host far, far away, also over a DSL-Link.
- Measure the RTT and normalize it

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 1
beqz $v0, loc_2DA24
move $v0, $0
la $t1, dword_35A70
lw $t1, $t1
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



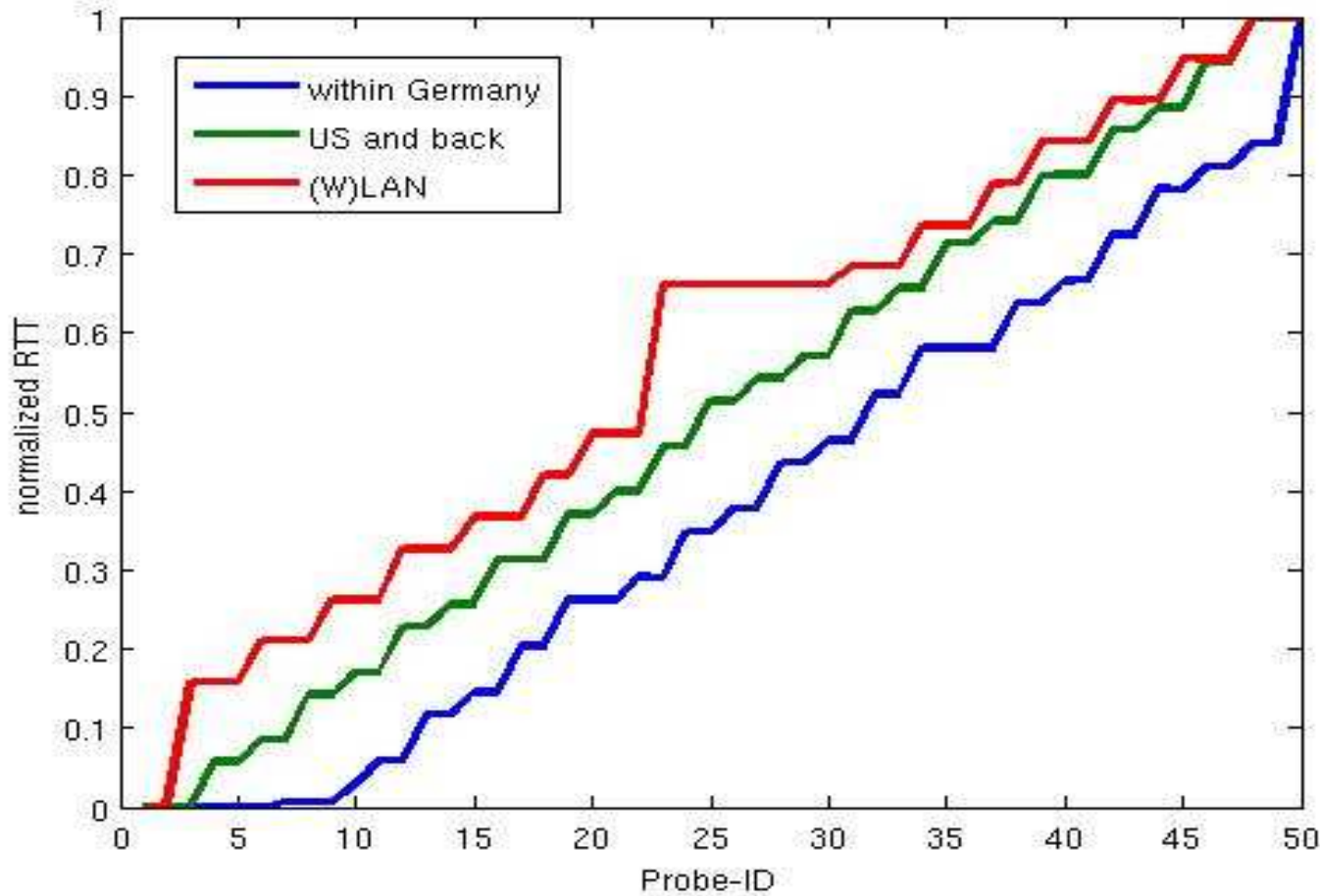
The "Burst-Response"

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
$ra, dword_35A6C
$1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
$ra, $t6, $t9
, loc_2D624

```

RTTs of bursty data



```

move $a0, $a0
lw $ra, $ra
jal sub_2DAB8
addiu $a1, $a1, 4
beqz $v0, $v0, $t4
move $v0, $v0
la $i, $i
lw $t1, $t1
lw $t0, $t0
subu $t2, $t2, $t1
sra $t3, $t3, $t2
sll $t4, $t4, $t3
addu $t5, $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



That's all very pretty, but ...

```

addiu $sp, -0x18
sw    $ra, 0x18+var_4($sp)
sw    $a0, 0x18+arg_0($sp)
lui   $t1, 3
jal   sub_2DAB8
      $a0, dword_35A6C
      $t1, 3
lw    $t7, dword_35A6C
lw    $t6, dword_35A70
subu  $t8, $t6, $t7
addiu $t2, $t6, 4
sll   $t1, $v0, $t9
beqz  $t1, loc_2DA24
nop
      sub_2D7C0

```

- ... now what do we do?
- How will we detect rate-limitation given this unfortunate result?
- Scanning does mean offering data at a certain rate over a longer time-period.
 - Detection during the scan is hard because we're constantly changing the net-load
 - A “one-” or “n-burst” solution would rock...

```

move  $a0, $t7
lw    $a0, dword_35A6C
jal   sub_2DA44
addiu $a1, $v0, 0x10
beqz  $v0, loc_2DA44
move  $v0, 0
la    $t1, dword_35A70
lw    $t1, dword_35A6C
lw    $t0, 0($t1)
subu  $t2, $t0, $t1
sra   $t3, $t2, 2
sll   $t4, $t3, 2
addu  $t5, $v0, $t4
sw    $t5, 0($t1)
sw    $v0, dword_35A6C

```

Invent & Verify

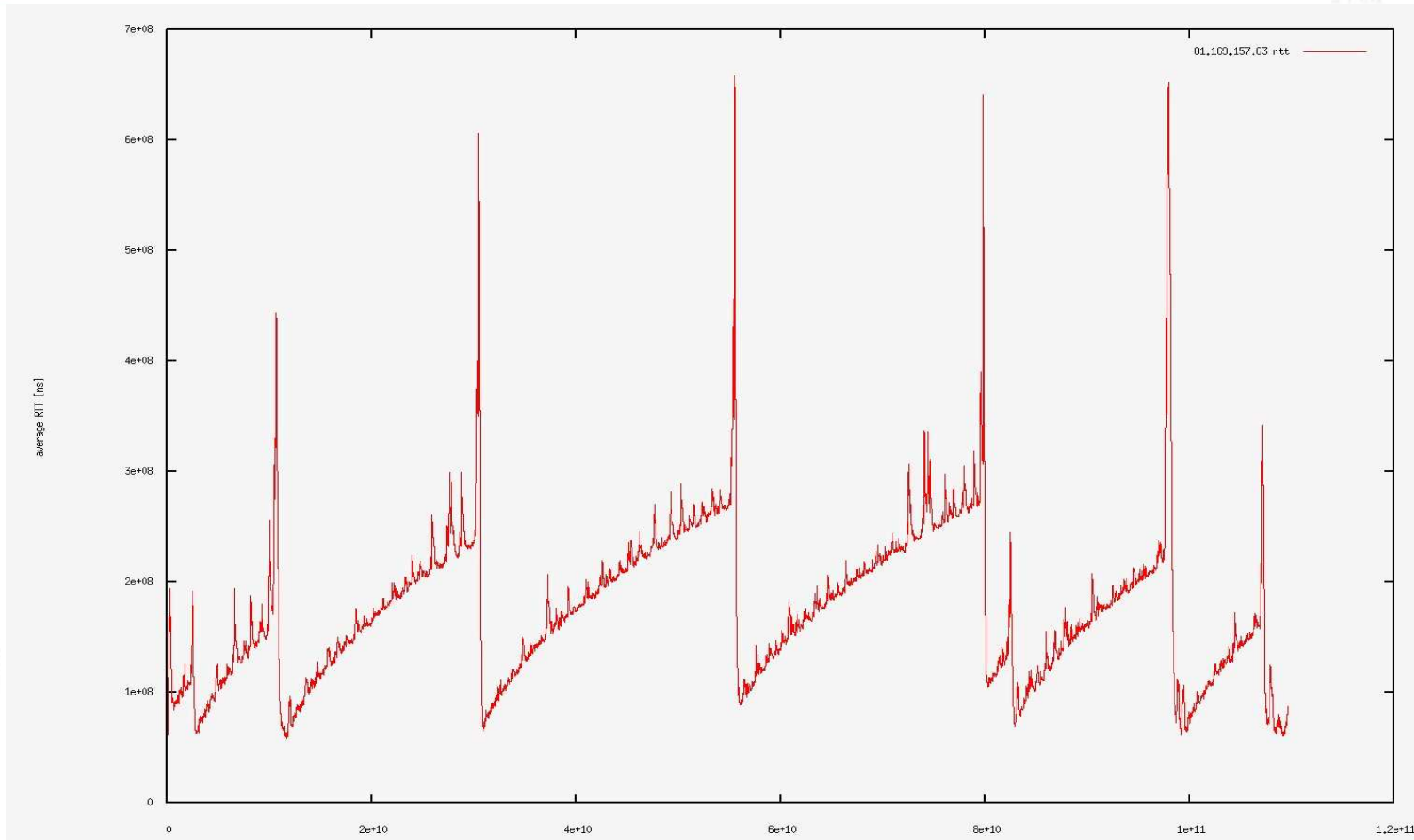


RTT-Development during scans (Reno)

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
li $i, 3
li $t7, dword_35A6C
li $t6, dword_35A70
li $t8, $t6, $t7
addiu $t9, $t6, 4
sll $i, $v0, $t9
li $i, loc_2DA24
sub $t9, $i

```



```

move $t4, $t3, 2
addiu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Decide based on number of drops?

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2DA24

```

- Of, course, we could just say "if our timing seems to somehow not work, there's a chance that there's a rate-limitation installed"
- But as we've seen with the IPHONE, that must not be the case -> device could just be a "dropper".

- Results of a false decision are disastrous

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a0, $v0, 0x18
beqz $v0, loc_2DA44
move $v0, $0
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($1)
sw $v0, dword_35A6C

```

Invent & Verify



New idea, limit is artificial!

- Rate-limitation is a totally artificial bottleneck
- It must be possible to reveal this artificial character somehow.
- And this is what we came up with ...

```
move    $a0, $t7
lw      $a0, dword_35A6C
jal     sub_2DAD4
addiu   $a1, $v0, 0x10
beqz    $v0, loc_2DA44
move    $v0, $0
la      $t1, dword_35A70
lw      $t1, dword_35A6C
lw      $t0, 0($t1)
subu    $t2, $t0, $t1
sra     $t3, $t2, 2
sll     $t4, $t3, 2
addu    $t5, $v0, $t4
sw      $t5, 0($t1)
sw      $v0, dword_35A6C
```

```
addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lw      $t1, 3
sub     $t1, $t1, $t1
lw      $t7, dword_35A6C
lw      $t6, dword_35A70
subu    $t8, $t6, $t7
addiu   $t2, $t6, 4
sll     $t1, $v0, $t9
sll     $t1, $t1, 20A24
```

Invent & Verify



Observe: This is a packet...

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal $t1, 2DAB8
l $t1, dword_35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t3, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2DA24

```

- ▷ 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



```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



... and this is, too.

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2D624

```

▸ 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)



```

move $t0, $t0
lw $t1, $t0
jal $t1
addiu $t2, $t0, 2
beqz $t2, $t0
move $t3, $t0
lw $t4, $t0
lw $t5, $t0
subu $t6, $t0, $t1
sra $t7, $t2, 2
sll $t8, $t3, 2
addu $t9, $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Now if the bucket claims...

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sleu $i, $v0, $t9
beqz $i, loc_2DA24
nop
sub 2DAB8

```

- ... that it can fit only 4 of these:



- ... or optionally 4 these:

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $i, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```

Invent & Verify

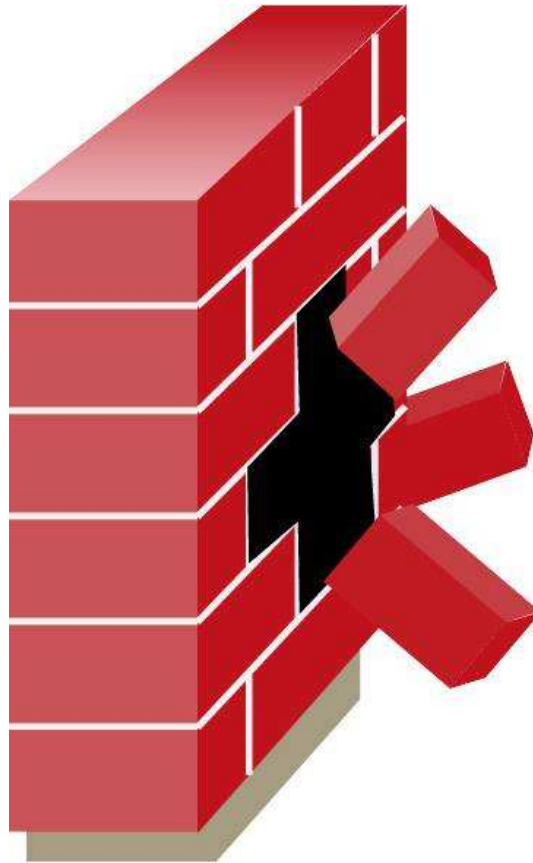


... then it's a lousy bucket.

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
fa sub $t8, $t1, 8
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $t1, $v0, $t9
beqz $t1, loc_2DA24
nop
sub 20000

```



```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Packet-size does not matter!

```

addiu $sp, -0x18
sw    $ra, 0x18+var_4($sp)
sw    $a0, 0x18+arg_0($sp)
lui   $t1, 3
jal   sub_2DAB8
lw    $a0, dword_35A6C
lui   $t1, 3
lw    $t7, dword_35A6C
lw    $t6, dword_35A70
subu  $t8, $t6, $t7
addiu $t2, $t6, 4
sll   $t1, $v0, $t9
beqz  $t1, loc_2DA24
nop
sub   sub_2DAB8

```

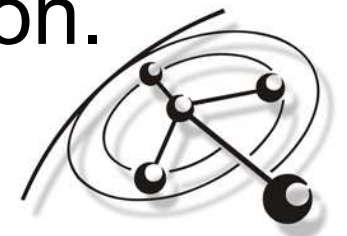
- Rate-limitation limits **number of packets**, the packet-size does not matter!
- In contrast: congestion is caused by **too much data** in the network
- **Just enlarge the packet** (Add TCP-options)
- If still the same number of packets return, we're obviously dealing with rate-limitation.

```

move  $a0, $v0
lw    $a0, dword_35A6C
jal   sub_2DAD4
addiu $a0, $v0, 0x10
beqz  $v0, loc_2DA24
move  $v0, $0
la    $t1, dword_35A70
lw    $t2, $t1
lw    $t0, 0($t1)
subu  $t2, $t0, $t1
sra   $t3, $t2, 2
sll   $t4, $t3, 2
addu  $t5, $v0, $t4
sw    $t5, 0($t1)
sw    $v0, dword_35A6C

```

Invent & Verify



Rate-limit-detector PoC

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2DA24
nop
sub_2DAB8
```

- [Demonstration]
- Packet-filter is disabled
- Two bursts of pings are sent.
- Packets of the second burst are twice as big as those of the first
- Enable packet-filter
- Send the two bursts again.

```
move $a0, $v0
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a0, $v0, 0x10
beqz $v0, loc_2DA24
move $v0, $a0
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



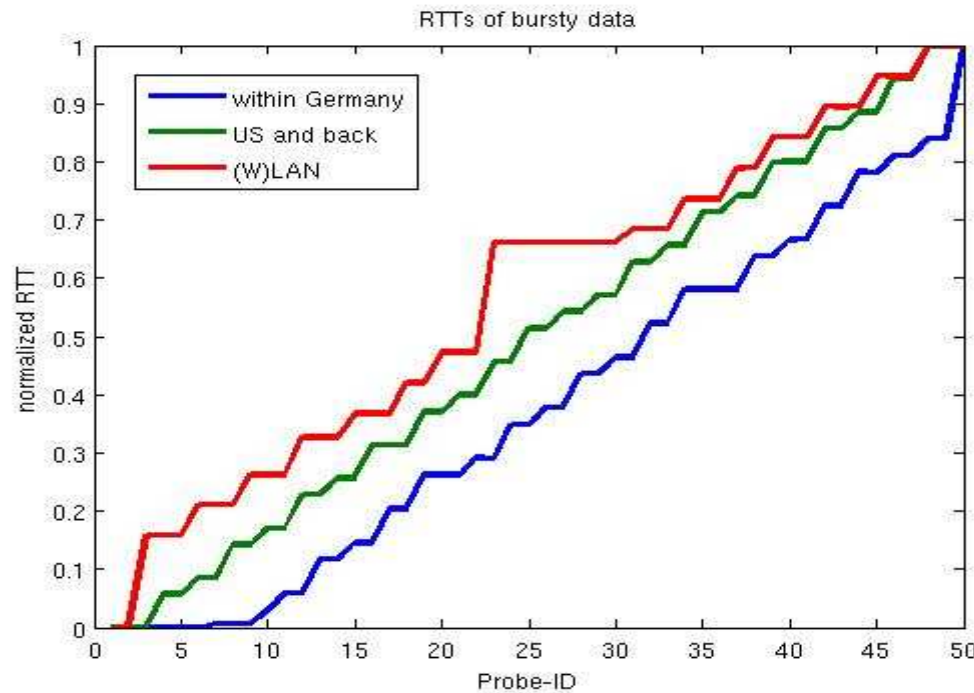
... cool, but...

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t3, $t6, 4
sleu $i, $v0, $t9
beqz $i, loc_4674

```

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



```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $i, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

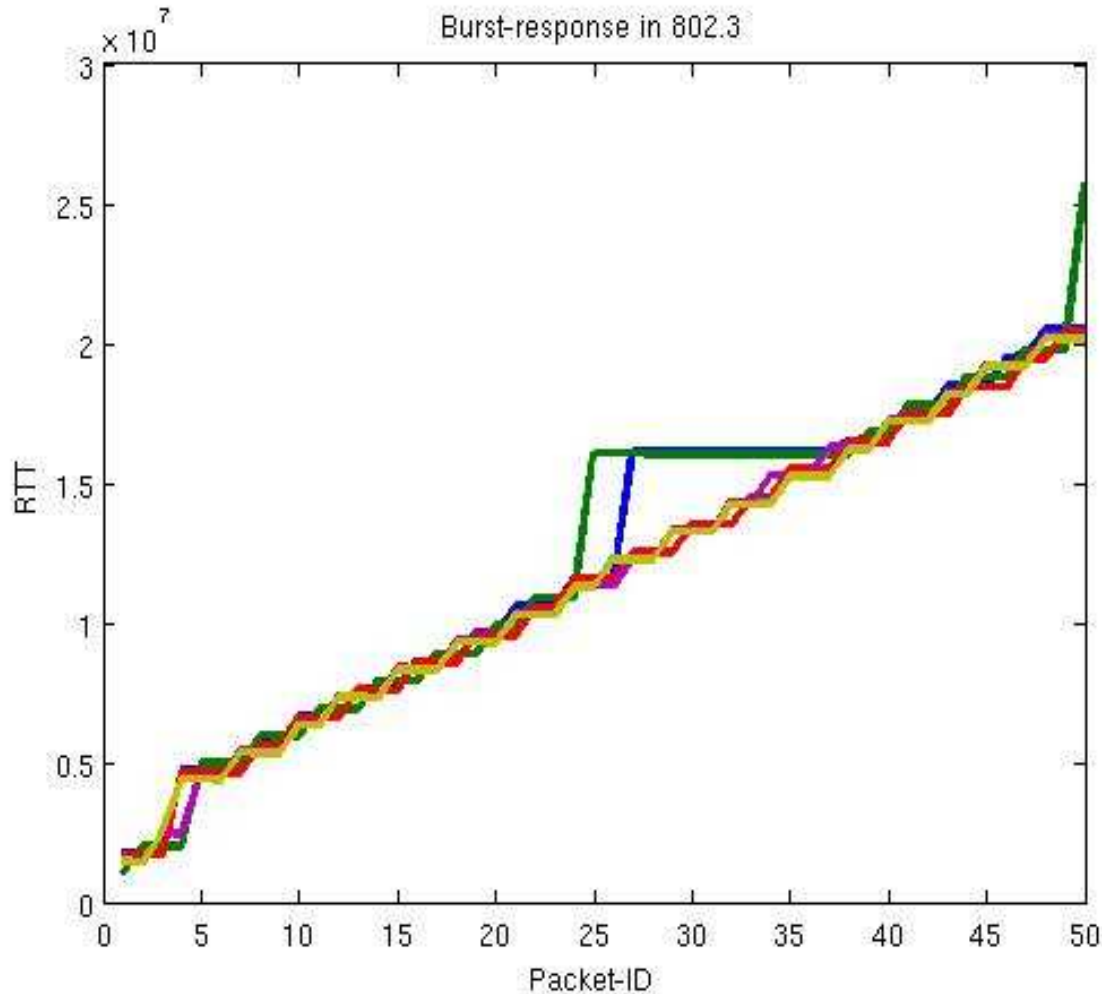
Invent & Verify



Can we detect WiFi-Links?

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2f
    dword_35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2DA24
nop
error [] = sub_2f($i)
    
```



error [] =

- 0.0589
- 0.0737
- 0.0322
- 0.0439
- 0.0421
- 0.0422

Mean =

0.0488

```

move $a0, $
lw $a0, d
jal sub_20
addiu $a1, $
beqz $v0, l
move $v0, $
la $i, dw
lw $t1, d
lw $t0, 0
subu $t2, $
sra $t3, $
sll $t4, $
addu $t5, $
sw $t5, 0($i)
sw $v0, dword_35A6C
    
```

Invent & Verify

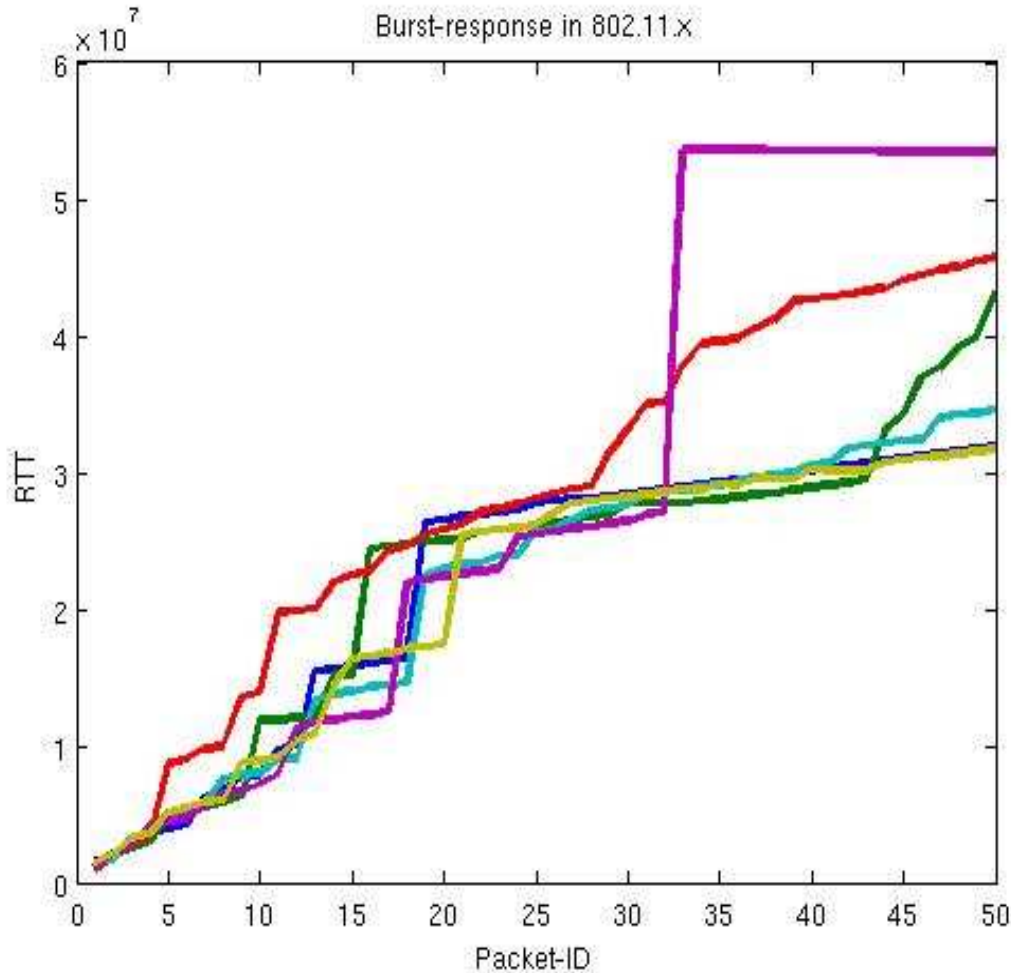


Yes, we can 😊

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7

```



error [] =

- 0.1706
- 0.0823
- 0.1154
- 0.1052
- 0.0935
- 0.1578

Mean =

0.1208

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0
beqz $v0, loc_35A70
move $v0, $0
la $i, dword_35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 31
sll $t4, $t3, 31
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Error-Calculation

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $1, $v0, $t9
beqz $1, loc_2DA24
sub $t10, $t8, $t9

```

- Given an input-vector 's' of RTTs, the error can be calculated by:
 - subtracting min(s) from each element of s
 - dividing each element of s by max(s)
 - calculating the absolute difference between the linear function (x) and s(x)
 - and summing all of these differences up.

```

move $a0, $v0
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a0, $v0, 0x18
beqz $v0, loc_2DA44
move $v0, $0
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($1)
sw $v0, dword_35A6C

```

Invent & Verify

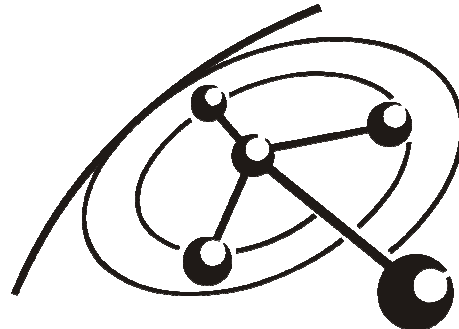


Thank you!

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $i, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $i, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 4
sllv $i, $v0, $t9
beqz $i, loc_2DA24
nop
sub 2DAB8

```



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>

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $i, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($i)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($i)
sw $v0, dword_35A6C

```

Invent & Verify

