A Variety of Ways to Capture and Analyze Packets:
A Network Engineer's Perspective

Timothy Chung
June 15, 2010

SHARKFEST **'10**
Stanford University
June 14-17, 2010

Agenda

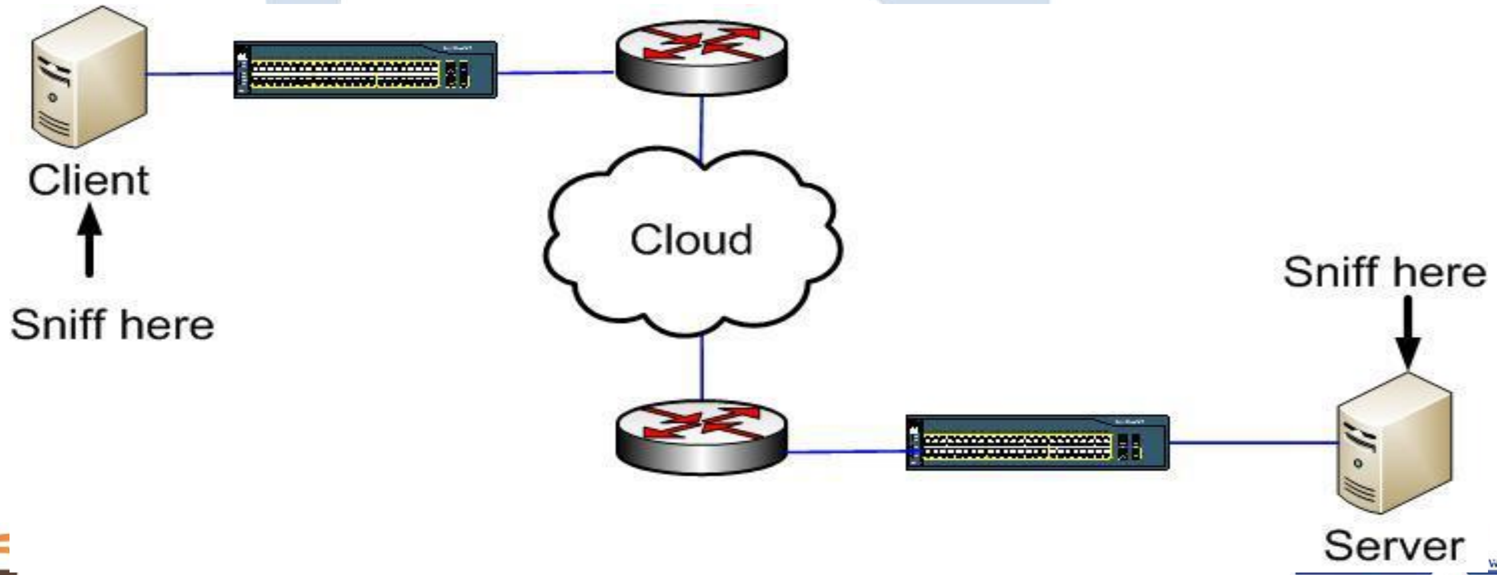
- Why Packet Capture
- Types of port mirroring
- Case Study: Browser hangs
- TAPs, SPAN, RSPAN, ERSPAN,
- ERSPAN Sample Config
- Mini Protocol Analyzer
- Case Study: Voip Phone
- Capture Exception Traffic to CPU
- Case Study: High CPU
- VACL Granularity, VACL Redirect
- Case Study: Network Congested

Why Packet Capture

- Validate proper protocol behavior
- Troubleshoot performance related issues
- Validate QoS ToS markings
- Troubleshoot “complex” network problems
- Identify anomalous traffic flows
- The smoking gun/definitive proof

Sniff Directly on hosts

- Get sniff of both client and server ends, if possible
  - E.g., Run wireshark/tshark on client and server
- Pro: Extremely convenient
- Con: Inband sniffing "may" exacerbate issue, not truly passive; Can't install wireshark on IP Phone end nodes



Case Study: Frozen Browser

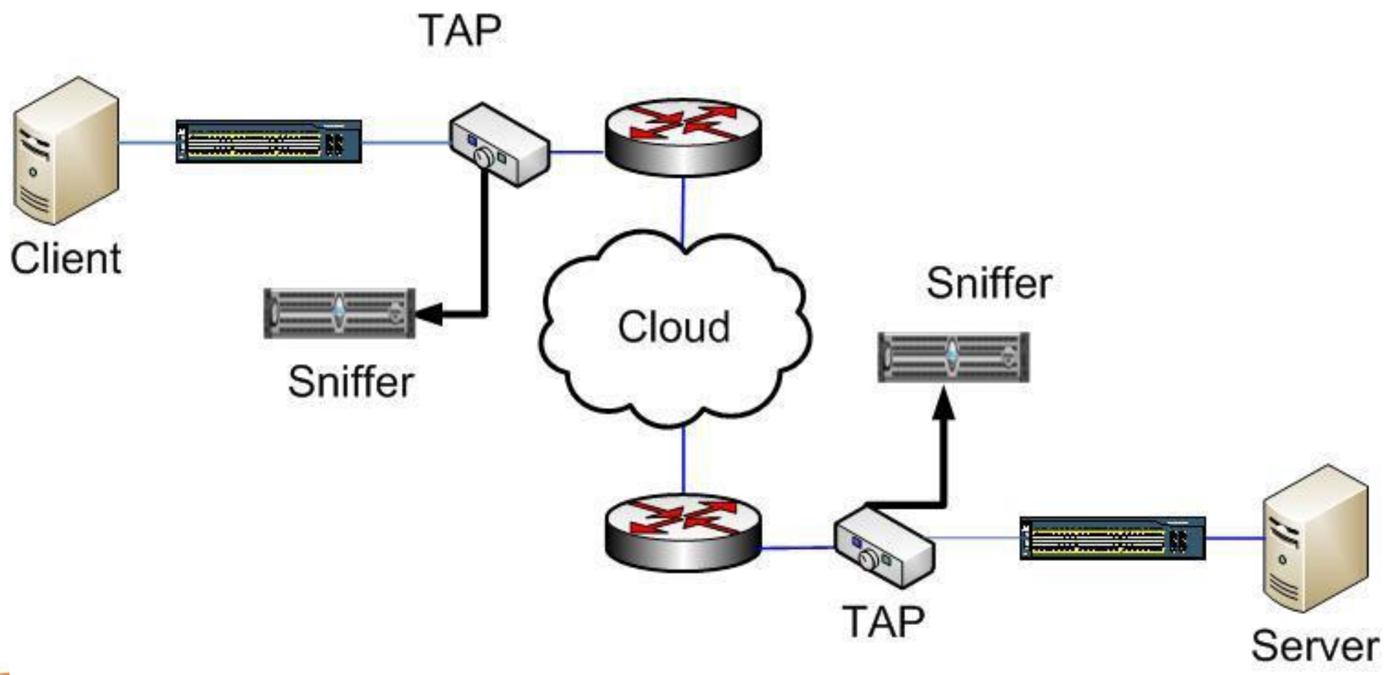
- Some users complain that their browsers are frozen intermittently on one particular website only
- The same website is fine if used from DSL line
- Pcap captured at client OS

Traceroute

```
root@nsx:/# tcptraceroute -n www.example.com
    traceroute to www.example.com (5.1.1.222), 30 hops max, 40 byte
    packets
     1  192.168.1.253  0.297 ms  0.334 ms  0.408 ms
     2  192.168.2.253  0.400 ms  0.484 ms  0.529 ms
     3  1.1.1.1  0.349 ms  0.435 ms  0.438 ms
     4  1.1.2.253  0.344 ms  0.402 ms  0.396 ms
     5  1.1.3.254 0.433 ms  0.499 ms  0.624 ms
     6  2.1.1.60  1.624 ms  2.092 ms  2.030 ms
     7  2.1.2.230  39.054 ms  38.999 ms  38.912 ms
     8  2.1.3.170  31.191 ms  2.618 ms  3.009 ms
     9  2.1.4.210  28.441 ms  28.842 ms  28.346 ms
    10  5.5.5.67  54.165 ms  53.608 ms  54.176 ms
    11  5.5.4.26  53.782 ms  53.390 ms  53.328 ms
    12  5.1.1.222  53.015 ms  52.941 ms  53.618 ms
```

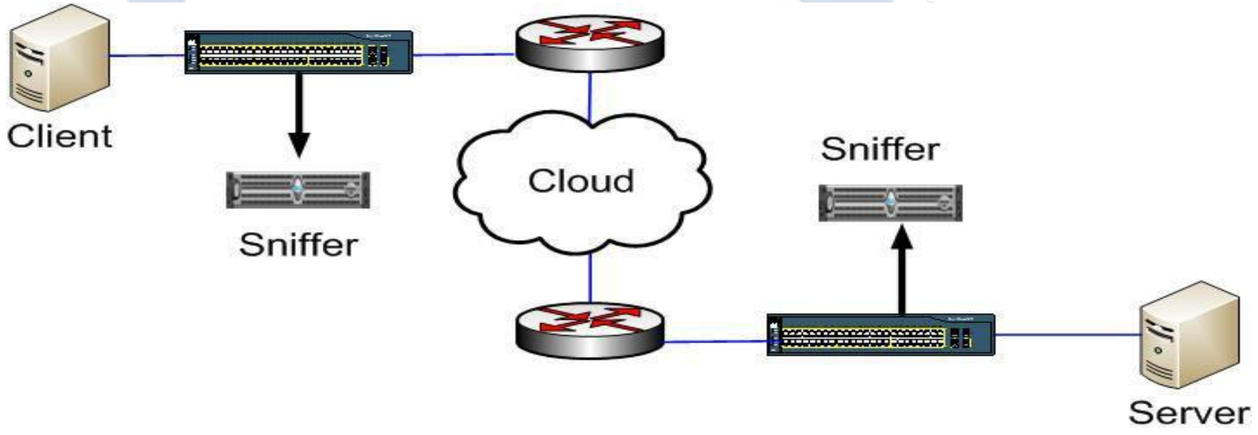
TAPs

- Deploy inline TAPs to capture data flow
- Pro: Truly passive, no frame drops by TAP
- Con: Another device to deploy, placement of TAP critical, cost



Port Mirroring

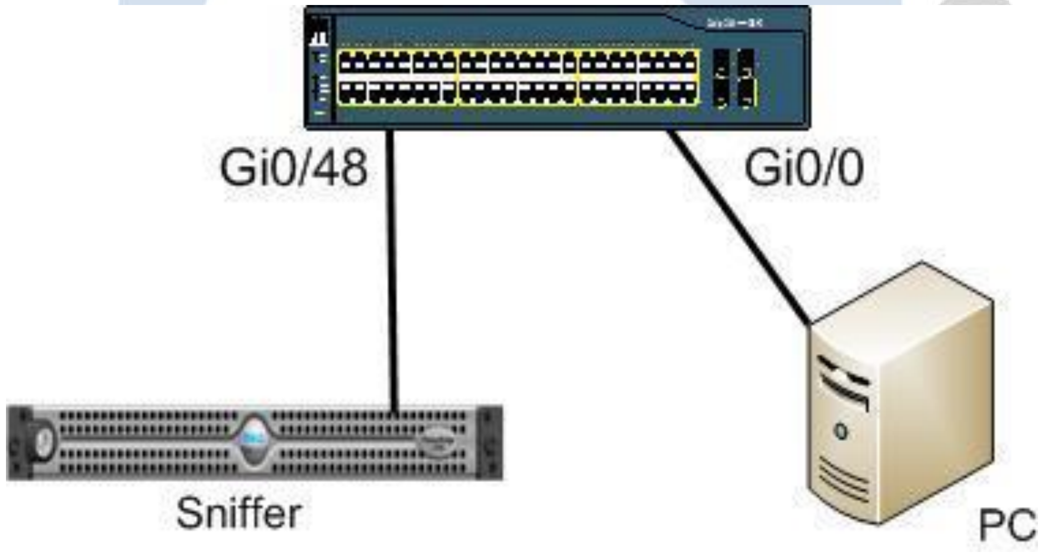
- Port mirror traffic on the switches to dedicated sniffer
- Pro: Convenient, port mirroring/SPAN feature included in most modern switches
- Con: Typically, 2 Span session limitation, dropped frames during oversubscription, Added latency of 20 MicroSeconds, Not truly passive technology, Multiple sniffers required



Local Port Mirror (SPAN)

- Mirrors traffic from interface to another

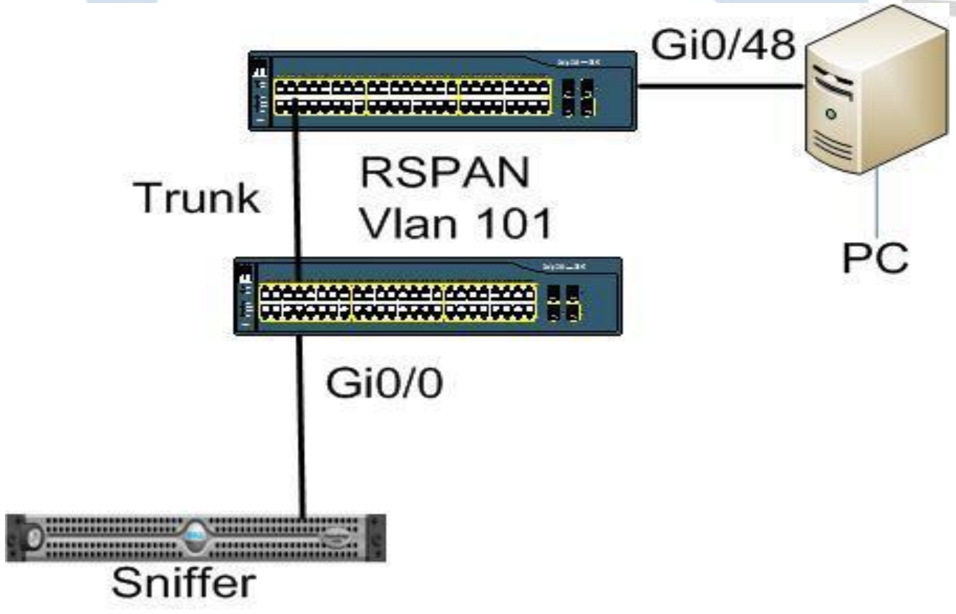
```
Switch(config)#monitor session 1 source interface Gi0/0
Switch(config)# monitor session 1 destination interface Gi0/48
```



Remote SPAN (RSPAN)

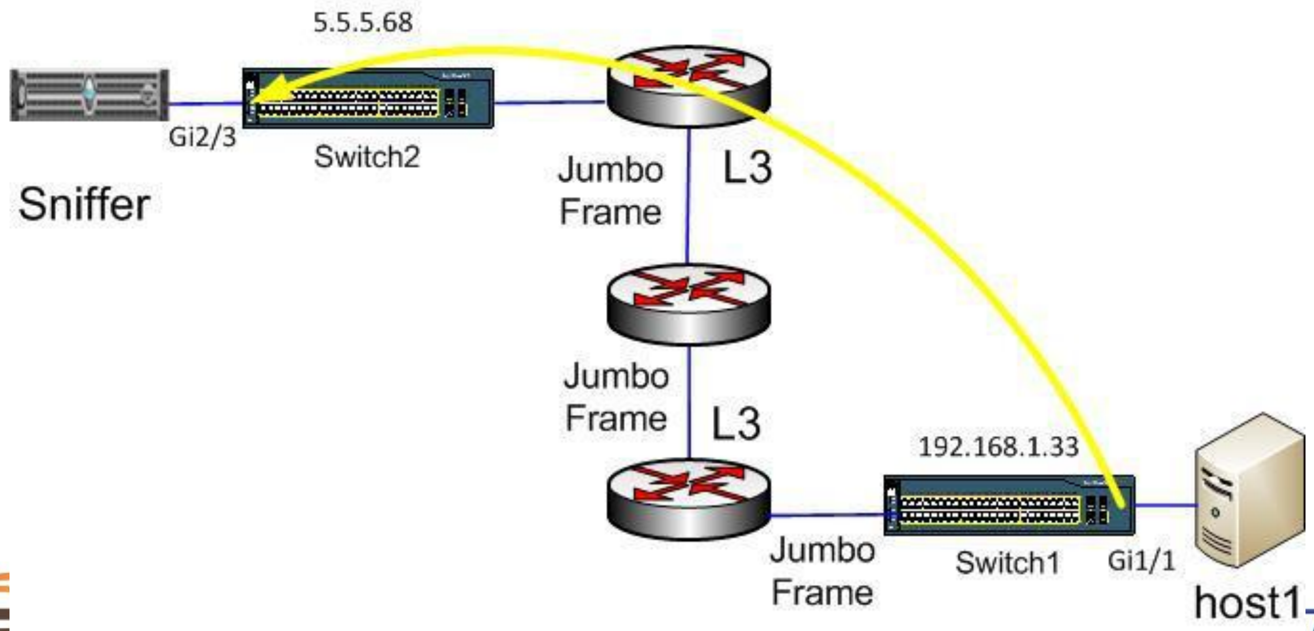
- Mirrors traffic from one interface on switch1 to a special L2 RSPAN VLAN across a trunk to switch2, which mirrors the traffic from the RSPAN VLAN to local interface

```
Switch1(config)#monitor session 1 source interface Gi0/48
Switch1(config)# monitor session 1 destination remote vlan 101
Switch2(config)#monitor session 2 source remote vlan 101
Switch2(config)# monitor session 2 destination interface Gi0/0
```



Encapsulated SPAN

- Mirrors traffic from one interface on switch1 into an IP GRE tunnel across arbitrary number of Layer 3 hops to destination switch, which decapsulates and mirrors traffic to its local interface.



ERSPAN

- Encapsulate entire Ethernet Frame in GRE
- Adds 50 Byte header
- DF bit is set to prevent fragmentation
- GRE Header protocol type of 0x88BE
- PFC3 and above supports ERSPAN (sup720, sup32)
- Cisco ASR supports ERSPAN as well
- ERSPAN ID uniquely identifies source sessions
- Full 1500 Byte packets cause performance Issue unless you use jumbo frames for interswitch links!

Sample ERSPAN Run

Source of ERSPAN:

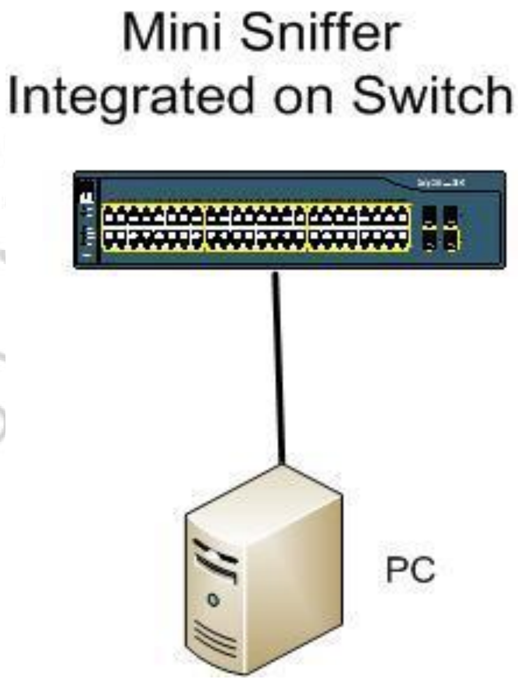
```
Switch1#sh run | b monitor
monitor session 3 type erspan-source
 source interface Gi1/1  <====host1 is connected here
 destination
  erspan-id 3
  ip address 5.5.5.68 <===IP of switch2 used for erspan
  origin ip address 192.168.1.33 <===IP of int on switch1
```

Destination of ERSPAN:

```
Switch2#sh run | b monitor
!
monitor session 1 type erspan-destination
 destination interface Gi2/3  <====sniffer attached here
 source
  erspan-id 3
  ip address 5.5.5.68
```

Mini Protocol Analyzer (Catalyst 6500)

- Captures traffic on an access port on the local switch and stores the captured packets in a local memory buffer for local or remote analysis
- Cat6500 SXH Release or later



Sample Capture Session

```
switch1(config)#monitor session 3 type capture
switch1(config-mon-capture)#buffer-size 65535
switch1(config-mon-capture)#source interface gi4/15 both

switch1#sh monitor capture
Capture instance [1] :
======================
Capture Session ID : 3
Session status     : up
rate-limit value   : 10000
redirect index     : 0x809
buffer-size        : 2097152
capture state      : OFF
capture mode       : Linear
capture length     : 68
```

Export Capture

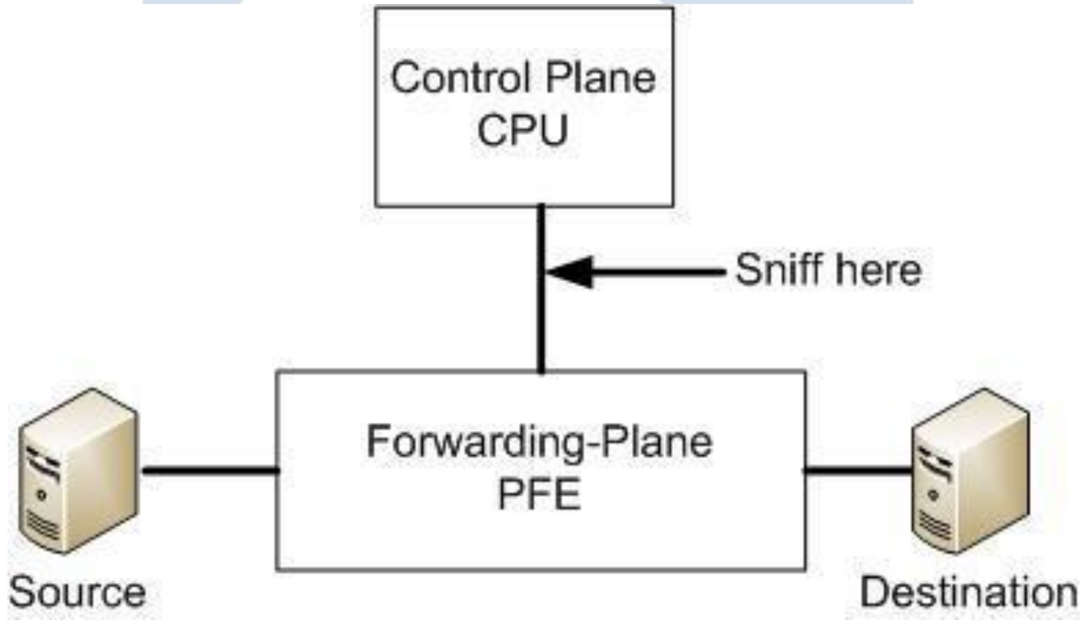
```
switch1#monitor capture length 1500 start
switch1#monitor capture stop

switch1#monitor capture export buffer disk0:cap1.pcap
Copying capture buffer of session [3] to location disk0:cap1.pcap
switch1#copy disk0:cap1.pcap scp:
```

Case Study: IP Phone on Infinite Reboot

- To support PC plugged behind IP Phone, there needs to be 2 distinct vlans – Voice for phone and Data for PC
- How does an IP phone know how to get itself into the voice vlan but place PC into Data?
- Cannot sniff directly on the IP Phone
- Use ERSPAN or Mini Protocol Analyzer

Sniff Traffic Punted to CPU

- Capture Traffic punted to CPU/Routing Engine
- Capture software processed/exception packets



Example Config

- Cisco Catalyst 6500

```
Switch1(config)#monitor session 2 type local
Switch1 (config-mon-local)#source cpu rp tx
Switch1(config-mon-local)#destination interface gi4/15
Switch1 (config-mon-local)#no shut
```

- Juniper

```
tim@R1> monitor traffic interface xe-0/0/0 no-resolve size 1500 write-file a.pcap
Address resolution is OFF.
Listening on xe-0/0/0, capture size 1500 bytes
^C
33 packets received by filter
0 packets dropped by kernel
```

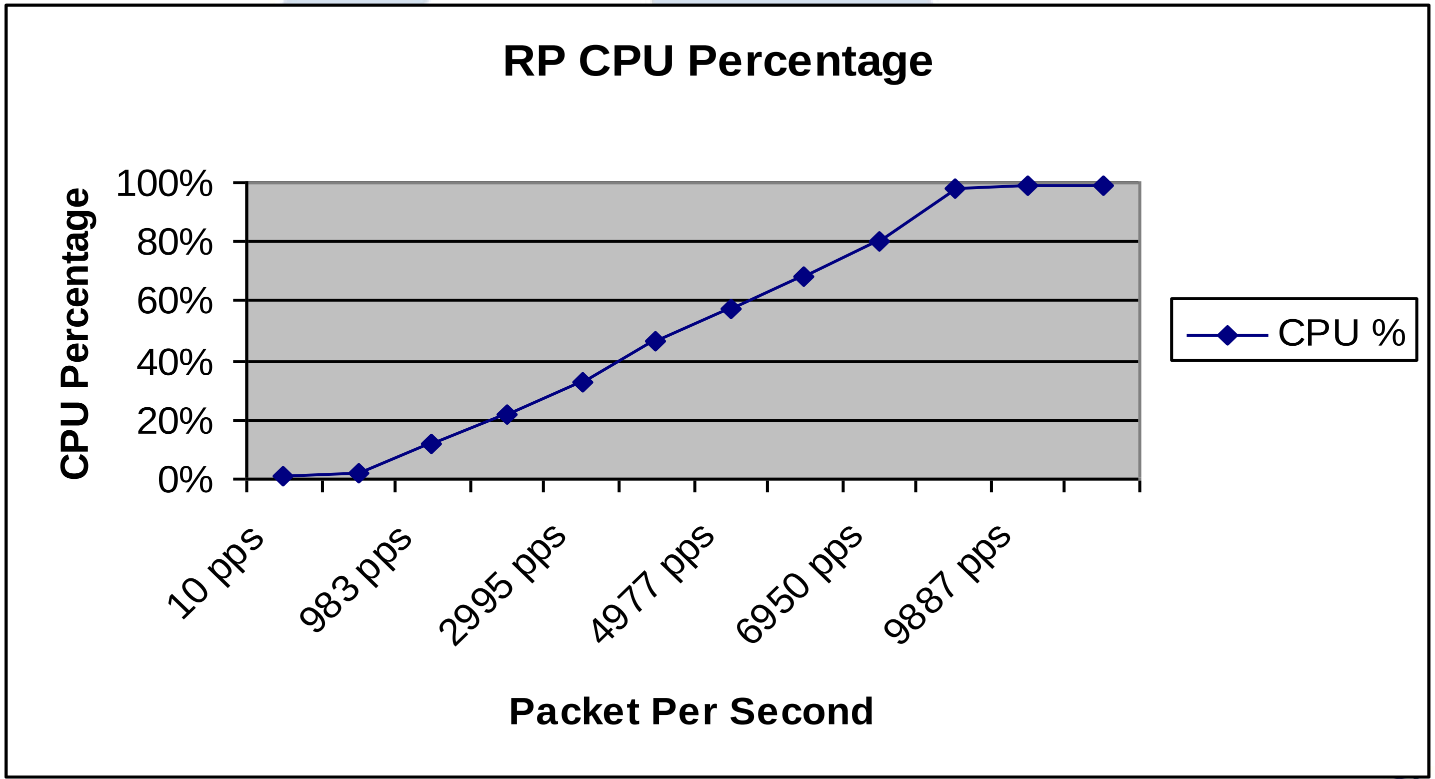
Case Study: High CPU

```
Switch1#sh proc cpu | e 0.00
CPU utilization for five seconds: 96%/96%; one minute: 39%; five minutes: 17%
 PID Runtime(ms)   Invoked      uSecs   5Sec   1Min   5Min TTY Process
  65       25768     26618        968  3.67%  0.96%  0.36%   1 SSH Process
 262       62028    300722        206 59.43% 14.07%  4.97%   0 IP Input
 449        6164     47191        130  0.23%  0.07%  0.06%   0 Port manager per
```

- Help! How do I identify what is causing this high CPU?
- Use in-band sniffing!

TTL Expiry on Catalyst 6500

- Traceroute and mtr are legitimate use of TTL expiry
- If transit packet has TTL=1, then punt to CPU for processing



Capture Granularity

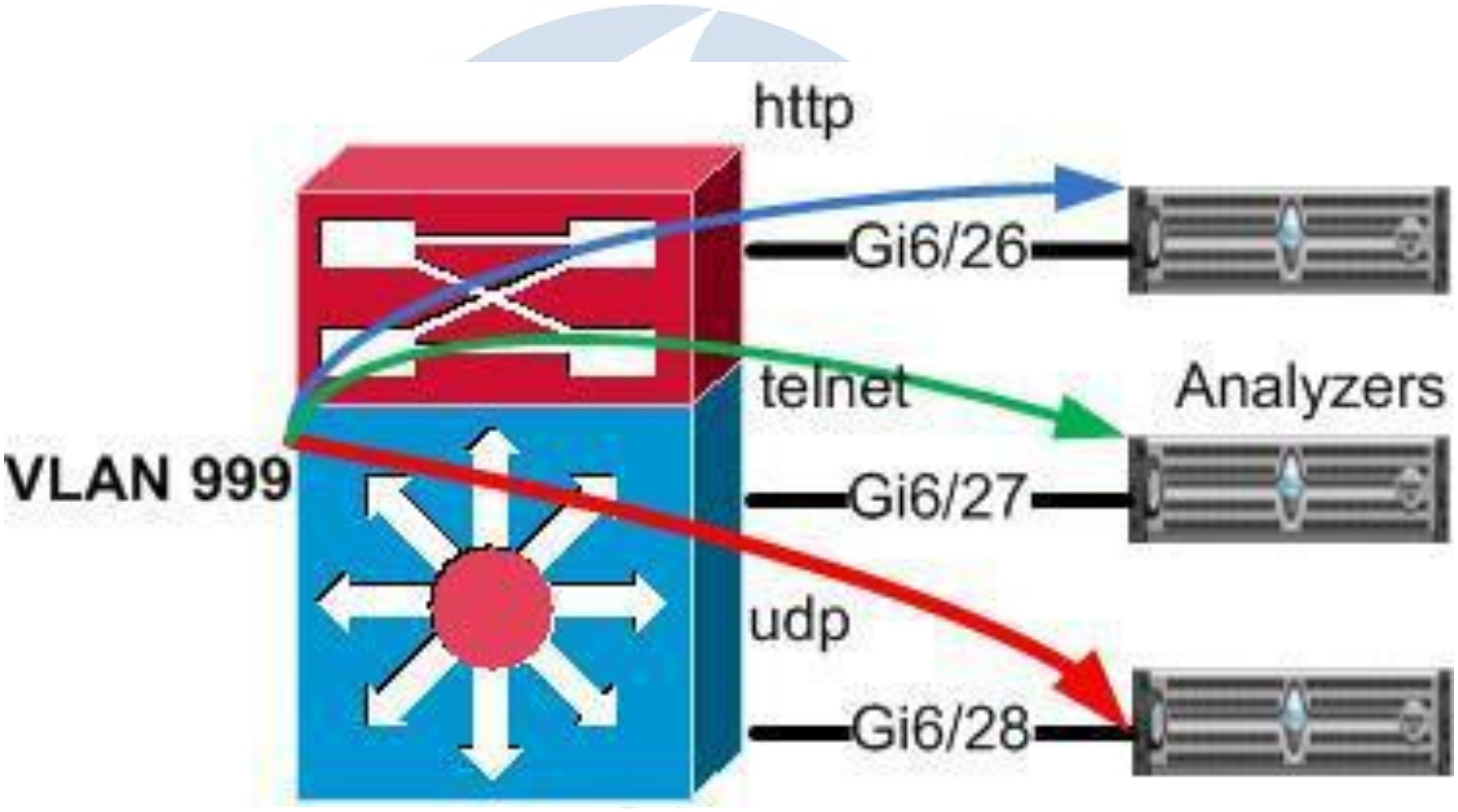
- You can use VACL to target specific protocols
- Supported on Catalyst 6500

```
vlan access-map AMAP 10
  match ip address http_acl
  action forward capture
 vlan access-map AMAP 20
  match ip address telnet_acl
  action forward capture
 !
 vlan filter AMAP vlan-list 999
 !
```

Capture Granularity

```
ip access-list extended http_acl
 permit tcp any any eq www
 permit tcp any eq www any
!
ip access-list extended telnet_acl
 permit tcp any any eq telnet
!
interface GigabitEthernet6/37
  switchport capture
  switchport capture allowed vlan 999
```

Sent to Multiple Analyzers



VACL Redirect to Multiple Interfaces

- You can split traffic and redirect to different interfaces

```
!  vlan access-map SPLIT-to-3Dest 10
      match ip address http_acl
      action redirect GigabitEthernet6/26
   vlan access-map SPLIT-to-3Dest 20
      match ip address telnet_acl
      action redirect GigabitEthernet6/27
   vlan access-map SPLIT-to-3Dest 30
      match ip address udp_acl
      action redirect GigabitEthernet6/28
    !
    vlan filter SPLIT-to-3Dest vlan-list 999
```

VACL Redirect to Multiple Interfaces

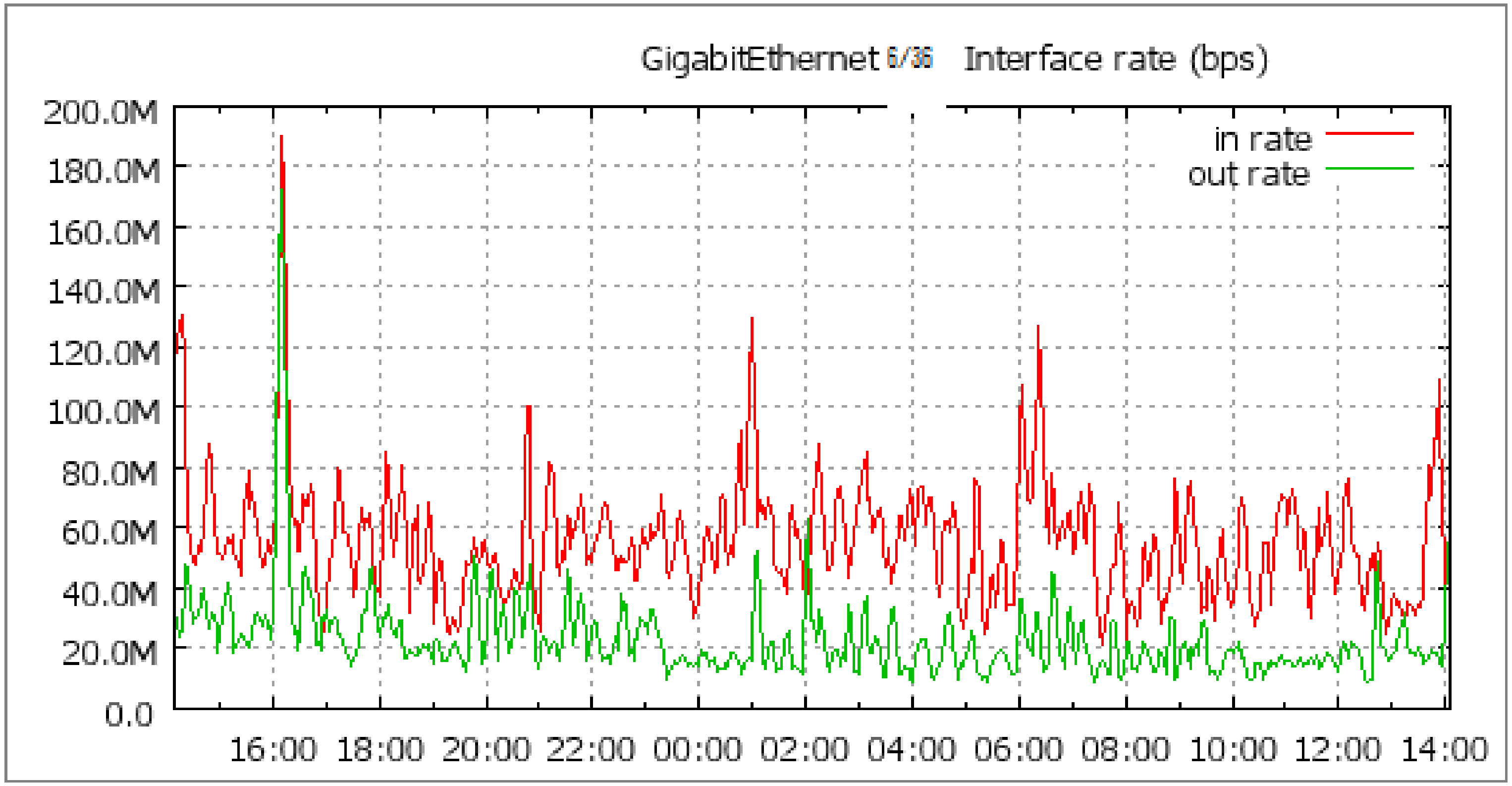
```
ip access-list extended telnet_acl
  permit tcp any any eq telnet
 ip access-list extended udp_acl
  permit udp any any
 ip access-list extended http_acl
  permit tcp any any eq www
  permit tcp any eq www any
```

Case Study: Network Congested

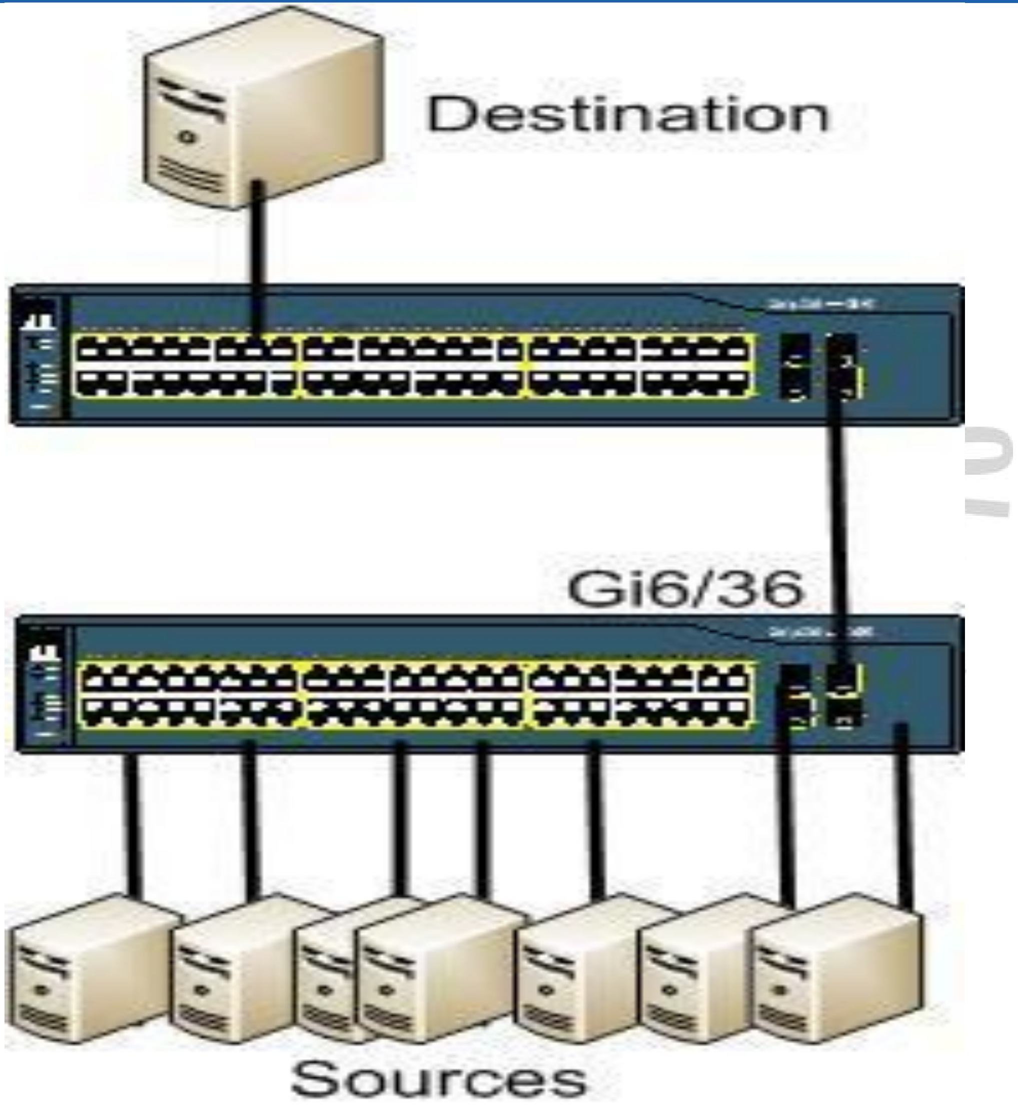
- Developer files a ticket with network team claiming severe packet drops for his application
- Network team says everything appears to be ok

```
Racksw1#sh int gi6/36 | i rate
   Queueing strategy: fifo
   30 second input rate 21578001 bits/sec, 2001 packets/sec
   30 second output rate 11578000 bits/sec, 1411 packets/sec
Racksw1#sh int gi6/36 | i drops
   Input queue: 0/2000/0/0 (size/max/drops/flushes); Total
  output drops: 3558
```

Utilization Graph



Topology



Proof is in Wireshark

```
tim@nsx:~/$ capinfos burst.pcap
File name: burst.pcap
File type: Wireshark/tcpdump/... - libpcap
File encapsulation: Ethernet
Number of packets: 4360
File size: 2284664 bytes
Data size: 2214880 bytes
Capture duration: 0.018531 seconds
Start time: Sun Jun  6 13:51:28 2010
End time: Sun Jun  6 13:51:28 2010
Data rate: 119522419.34 bytes/s
Data rate: 956179354.75 bits/s
Average packet size: 508.00 bytes
```

Questions?