SharkFest'23 US
San Diego, CA • June 10-15

#sf23us

Beyond Network Latency

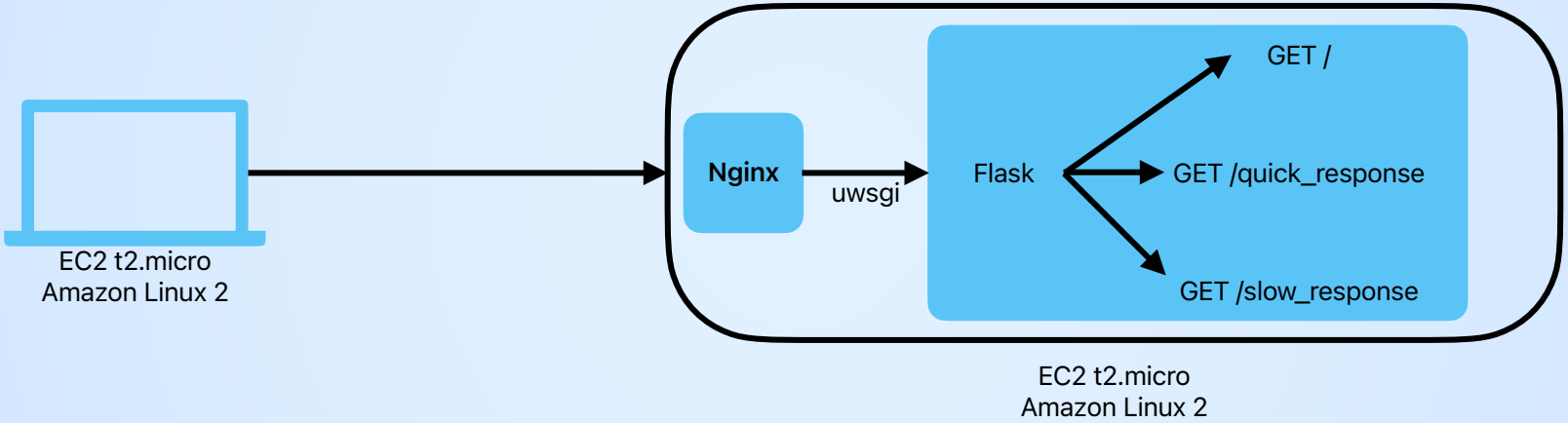
Chasing Latency Up the Stack

Josh Clark
Huntington National Bank

Who Am I?

- Distributed Performance Engineer – 2018-now
  - Performance and Latency Analysis
  - Collaborate with technical teams to identify bottlenecks in complex applications
- M.S. in Network Engineering – 2016
  - Theory of protocol design
  - Performance analysis of internet protocols

Testing Architecture - AWS



Normal internet network latency of 30ms simulated on client using tc
Python script (request_code.py) deployed on client to simulate a user
All packets captured via tcpdump on the client

Defining Terms

- We'll be using these terms throughout the talk, so let's define them
  - Network Delay – the time it takes for a packet to travel between client and server, measured as round trip time
  - Application Delay – the time it takes for an application to respond to a request. This is measured between the application process receiving a message and sending a response to the local server's network stack
  - Server Delay – the time it takes a server to transfer messages between the NIC and an application process
  - Client Wait Time – the client's version of server + application delay. For user applications, this is usually influenced by the user interacting with the application

Capture 1: Fast Response

- Normal_fast_clear.pcapng
- Normal_fast_cipher.pcapng

Interlude: THE PATTERN

Δ Conv	Source	Destination	Protocol		
0.005371000	Client	Server	TCP	1448	·····A····
0.000015000	Client	Server	TLSv1.2	789	·····AP···
0.000506000	Server	Client	TCP	0	·····A····
1.002826000	Server	Client	TCP	2896	·····AP···
0.000001000	Server	Client	TCP	2896	·····AP···

Payload = MSS, no PSH | **client REQ**
Payload < MSS, PSH | **end of REQ**
Payload = 0, ACK | **network delay**
Payload > 0 | **server + app delay**

If you only learn one thing from this talk, let it be this pattern

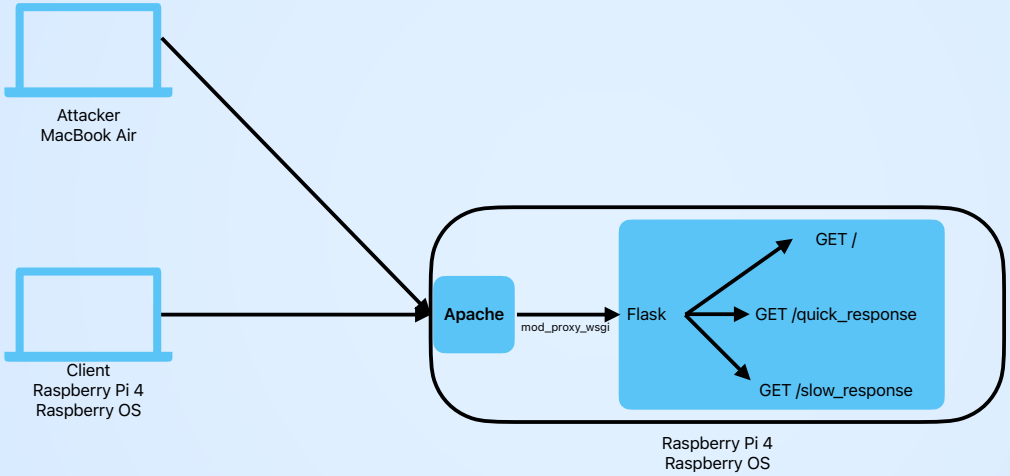
Capture 2: Network Latency

- See the following captures
  - Net_delay_clear.pcapng
  - Net_delay_cipher.pcapng

Capture 3: Application Latency

- See the following captures
  - Normal_slow_clear.pcapng
  - Normal_slow_cipher.pcapng

Testing Architecture - Raspberry Pi

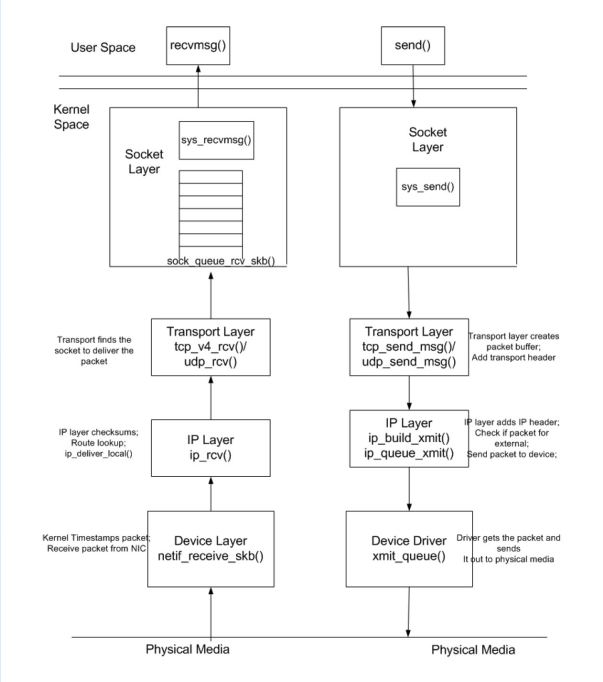


Switched to Raspberry Pis to reduce overall capacity of server
Switched to Apache as it does not handle load as well as nginx
Added my laptop to add load to the server without cluttering up client's tcpdumps

Capture 4: Raspberry Pi, GET Flood

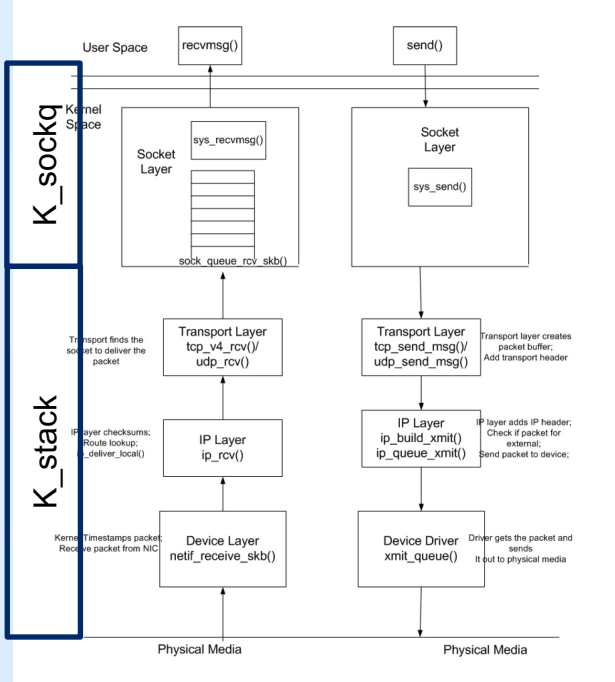
- See the following capture
  - Get_flood_fast.pcap

Why This Happened



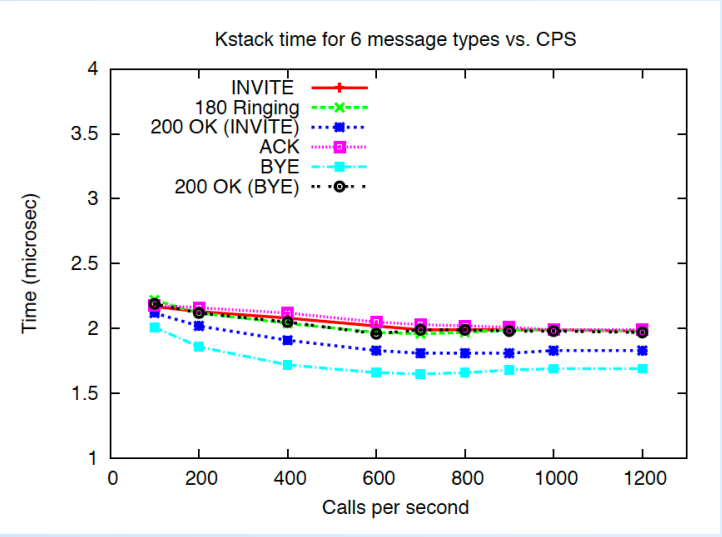
R. Krishnamurthy, "A Framework for Evaluating Server Performance: Application to SIP Proxy Servers", 2016. Available ramekris.wixsite.com/research

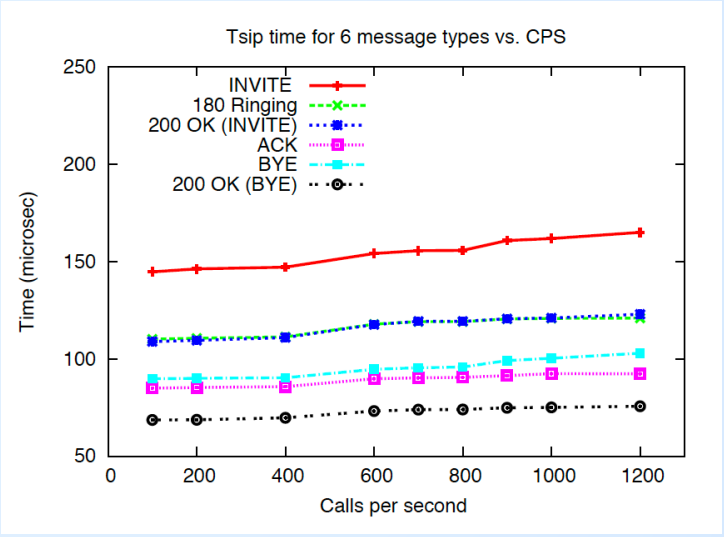
Why This Happened



R. Krishnamurthy, "A Framework for Evaluating Server Performance: Application to SIP Proxy Servers", 2016. Available ramekris.wixsite.com/research

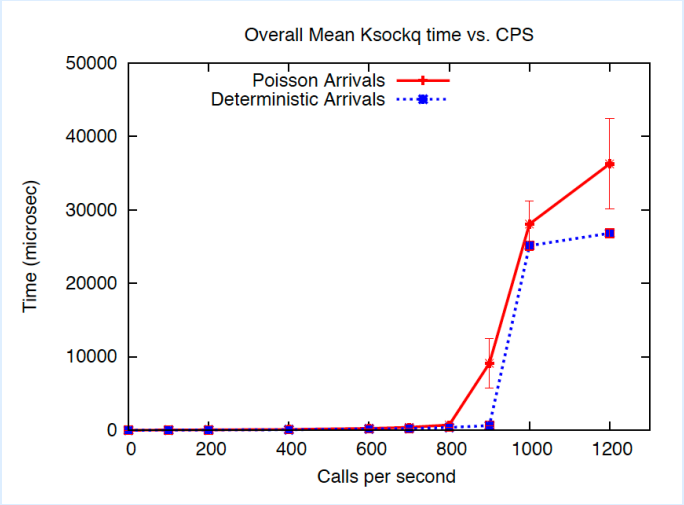
Why This Happened





R. Krishnamurthy, "A Framework for Evaluating Server Performance: Application to SIP Proxy Servers", 2016. Available ramekris.wixsite.com/research

Why This Happened



When an application is busy, it can take a long time for the kernel to get the application to pull a message from the socket buffer

R. Krishnamurthy, "A Framework for Evaluating Server Performance: Application to SIP Proxy Servers", 2016. Available ramekris.wixsite.com/research

Capture 5: Raspberry Pi, SYN Flood

- See the following capture
  - Syn_flood_fast.pcap

Why This Happened

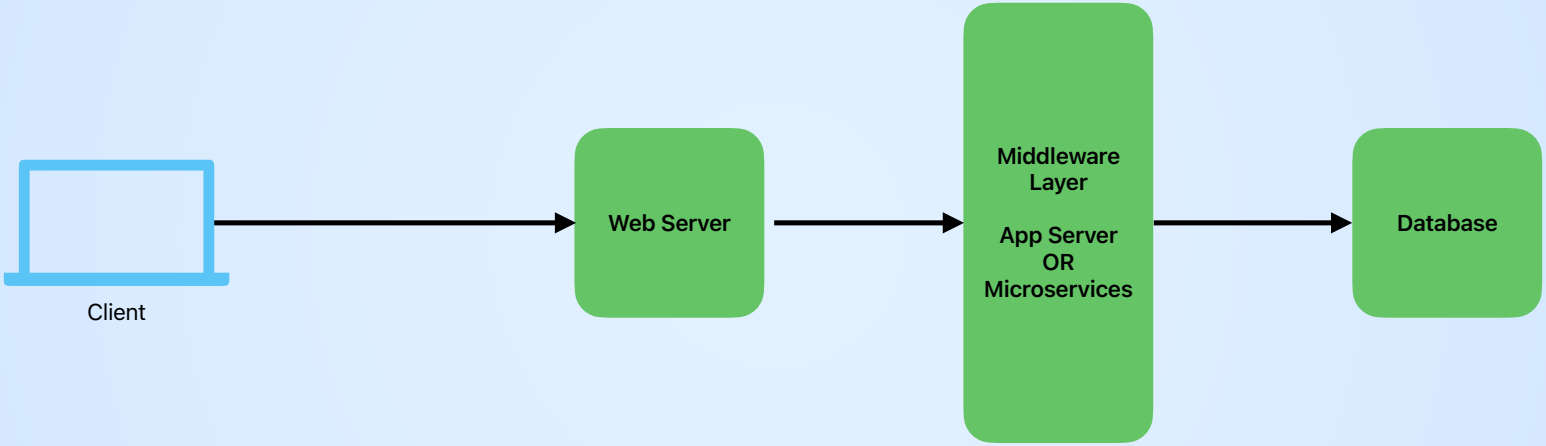
- To get a packet to the kernel, the NIC must send a soft interrupt to the CPU
- The process that handles that interrupt is ksoftirqd. Ksoftirqd calls netif_receive_skb()

Why This Happened

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
14	**root**	**20**	**0**	**0**	**0**	**0**	**R**	**90.7**	**0.0**	**0:18.56**	**ksoftirqd/0**
1074	**root**	**20**	**0**	**0**	**0**	**0**	**R**	**9.3**	**0.0**	**0:01.33**	**kworker/0:2+events**
709	root	20	0	3116	1840	1420	S	0.7	0.0	0:00.56	dhcpcd
850	root	20	0	0	0	0	I	0.3	0.0	0:00.15	kworker/2:1-events
1069	**pi**	**20**	**0**	**9860**	**3248**	**2716**	**R**	**0.3**	**0.1**	**0:05.46**	**top**
1	root	20	0	166824	10172	7436	S	0.0	0.3	0:02.86	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.03	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp

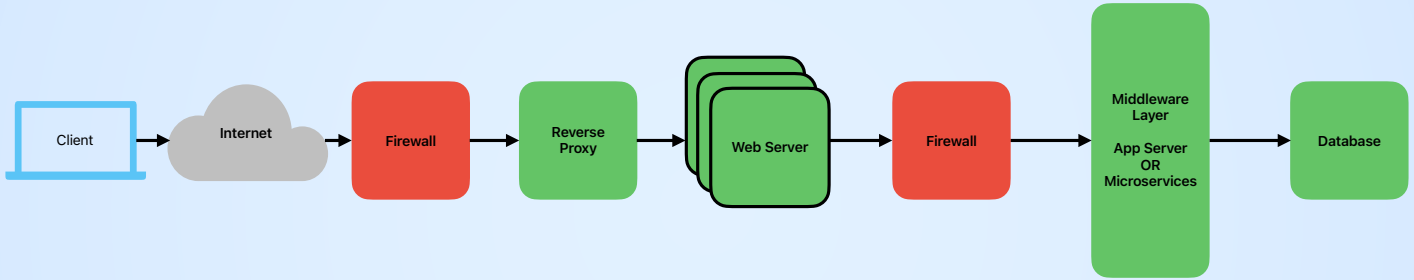
This is from the top command on the server during the SYN flood

Applications - Three Tier Application



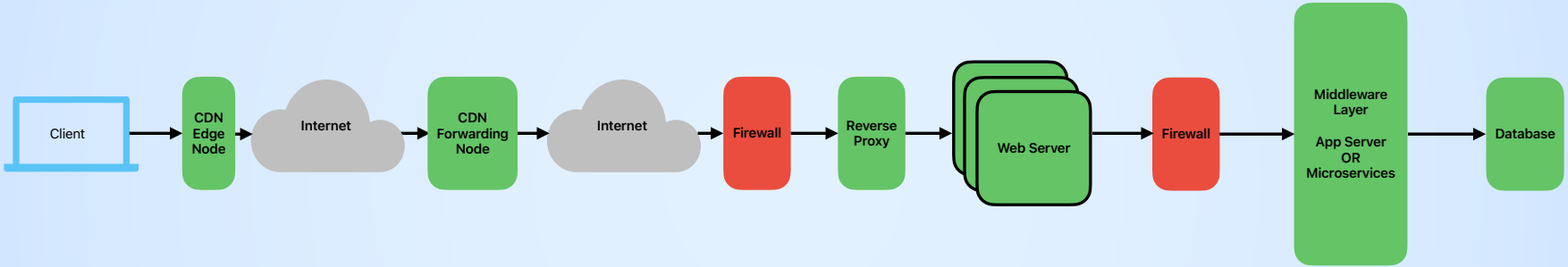
Packet captures from a given layer let us perform latency analysis on both sides of that layer.

Applications - Internet Facing



In a modern internet-facing application, any one of these devices can be causing latency, including devices deployed as appliances. Getting packet captures at multiple layers is critical to isolating a problematic layer.

Applications - CDN Fronted



With a CDN in the mix, it's difficult to determine any latency at the client. Because CDNs operate at Layer 7, we don't even get a good understanding of network latency.

Questions?