Load Balancer

Intro and Implementation

Xiaotian Li

Networking Terms 什么是物数网传会表应？

OSI Layer		Protocol data unit (PDU)
Host layers	7. Application	Data (HTTP, HTTPS, FTP, DNS)
	6. Presentation	
	5. Session	
	Socket (IP+port)	
	4. Transport	Segment (TCP) / Datagram (UDP)
Media layers	3. Network	Packet (IP)
	2. Data link	Frame (Mac)
	1. Physical	Bit (Electricity, Light)

Data (!"#$☺) 信件

TCP/UDP (port, 2^16-1） 邮箱<-socket

IP (192.168.1.1) 住址

Mac (f2:27:97:4a:88:2e) 身份证

Bit (01) 原子

发数据包的过程中IP和Mac谁变谁不变？

源住址（src ip）和目的住址（dest ip）不变；

发信人=>邮递员...=>收信人的身份（Mac）一直在变；

What is Load Balancer?

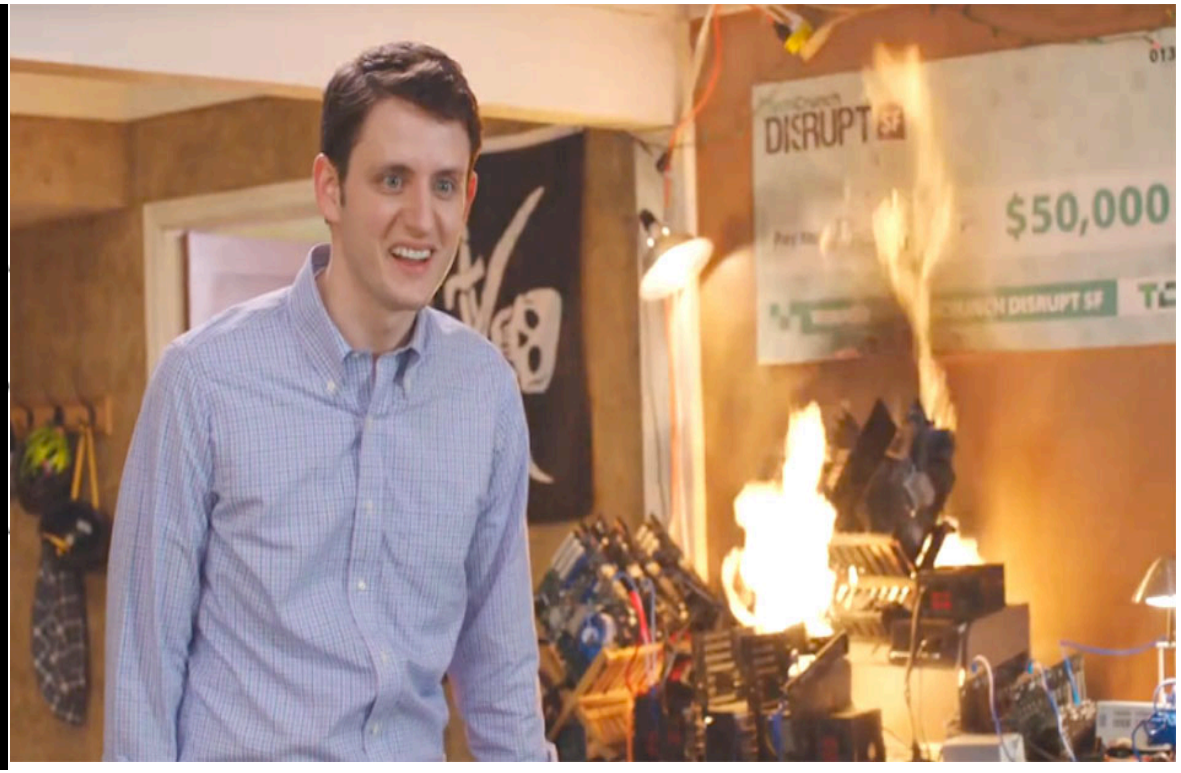
Load Balancer:

- Distributes workloads across multiple computing resources, e.g. providing a single Internet service from multiple servers.

Application Delivery Controller/Network (ADC/ADN) :

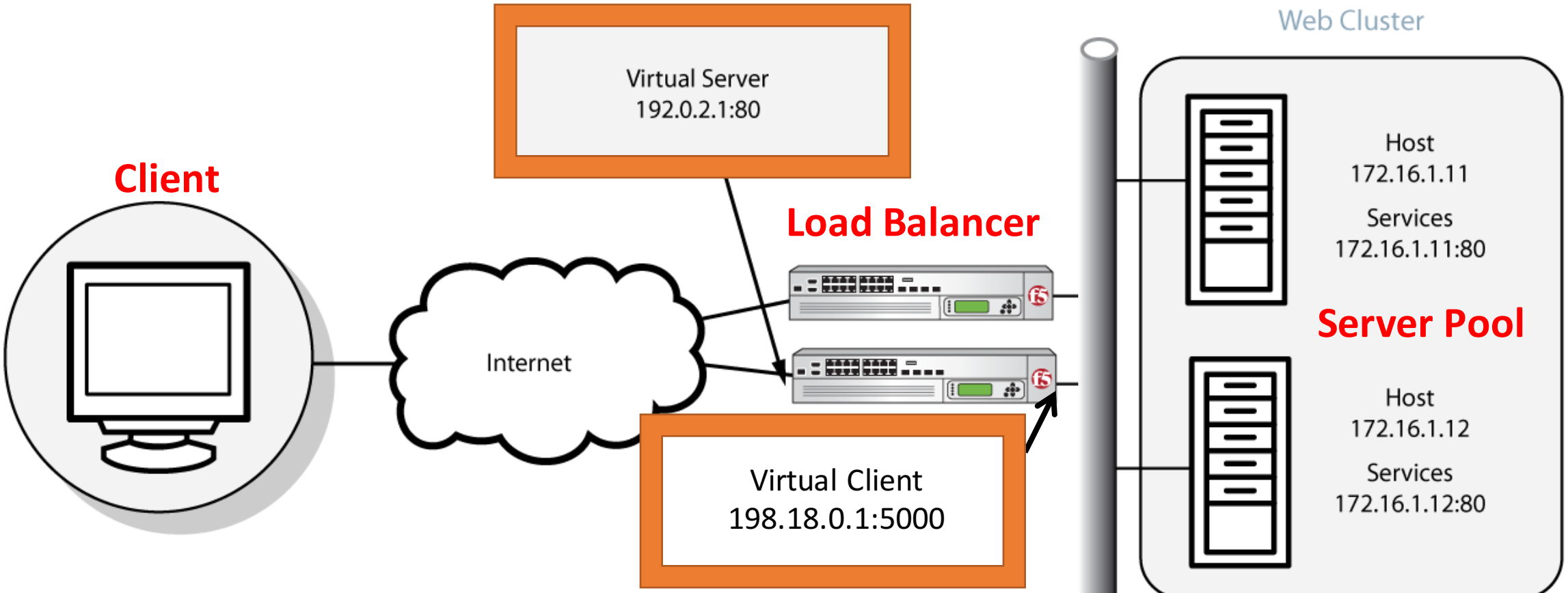
- Evolved from the first load balancers to improve **availability**, but also affect the **security** and **performance** of the application services being requested.

Client-Server(CS) model



This is why we need Load Balancer!

Client-Load Balancer-Server Model



1) Load balancer presents a **"virtual server"** address to the external **clients**
2) Load balancer presents a **"virtual client"** address to the internal **servers**
3) Packets forwarding are done by **Bi-directional Network Address Translation (BNAT=SNAT+DNAT).**
4) The load balancer can apply unique load balancing **based on the services (IP:Port)** instead of the host (IP). A server (172.16.1.10) may have more than one service available (HTTP->80, HTTPS->443, FTP->21, DNS->53, and so on)

Question 1

What happens if the selected host isn't working?

- The host doesn’t respond and eventually the connection times out and fails.
- Health monitoring
  - PING
  - “service” PING, ranging from simple TCP connections to interacting with the application via scripts

PING (ICMP echo/reply, OSI Layer3)

howtogeek@ubuntu: ~

```
howtogeek@ubuntu:~$ ping -c 4 google.com
PING google.com (173.194.33.39) 56(84) bytes of data.
64 bytes from sea09s02-in-f7.1e100.net (173.194.33.39): icmp_req=1
ttl=128 time=16.0 ms
64 bytes from sea09s02-in-f7.1e100.net (173.194.33.39): icmp_req=2
ttl=128 time=18.3 ms
64 bytes from sea09s02-in-f7.1e100.net (173.194.33.39): icmp_req=3
ttl=128 time=24.3 ms
64 bytes from sea09s02-in-f7.1e100.net (173.194.33.39): icmp_req=4
ttl=128 time=16.0 ms

--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 16.045/18.709/24.315/3.375 ms
howtogeek@ubuntu:~$
```

Question 2

How does the load balancer decide which server to send a connection request to?

- Each **virtual server** has a specific dedicated cluster of services which makes up the **all available servers**.
- Additionally, the **health monitoring** modifies that list to make a list of "**currently available servers**" that provide the indicated service.
- **Load balancing algorithms** (round robin, random, and others based on current connection count, host utilization, response time etc.)

Question 3

What about **follow-on traffic** from a known connection? To Load Balance or Not to Load Balance?

- Connection maintenance
- Persistence (aka. Server Affinity)
- All above information is stored in “Flow Table”

Connection Maintenance

Used for **long-lived** TCP connections.

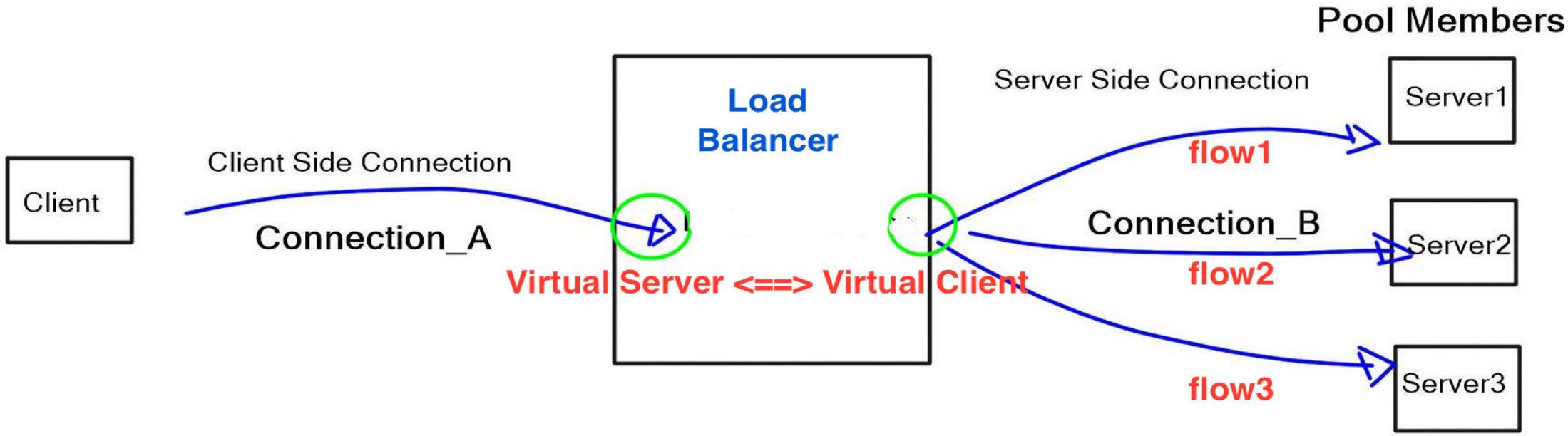
- Keep track of open connections and the server service it belongs to.
- Monitor the connection and update connection table when the connection closes.

Persistence (aka. Server affinity)

Used for multiple **short-lived** TCP connections from **the same user**.

- One of the most basic forms of persistence is **source-address affinity.** (works sometimes)
- Deep Packet Inspection (**DPI**), check unique and identifiable info, such as user id stored in cookie etc. (works always, however requires the info to be present in each request made)

Flow Table

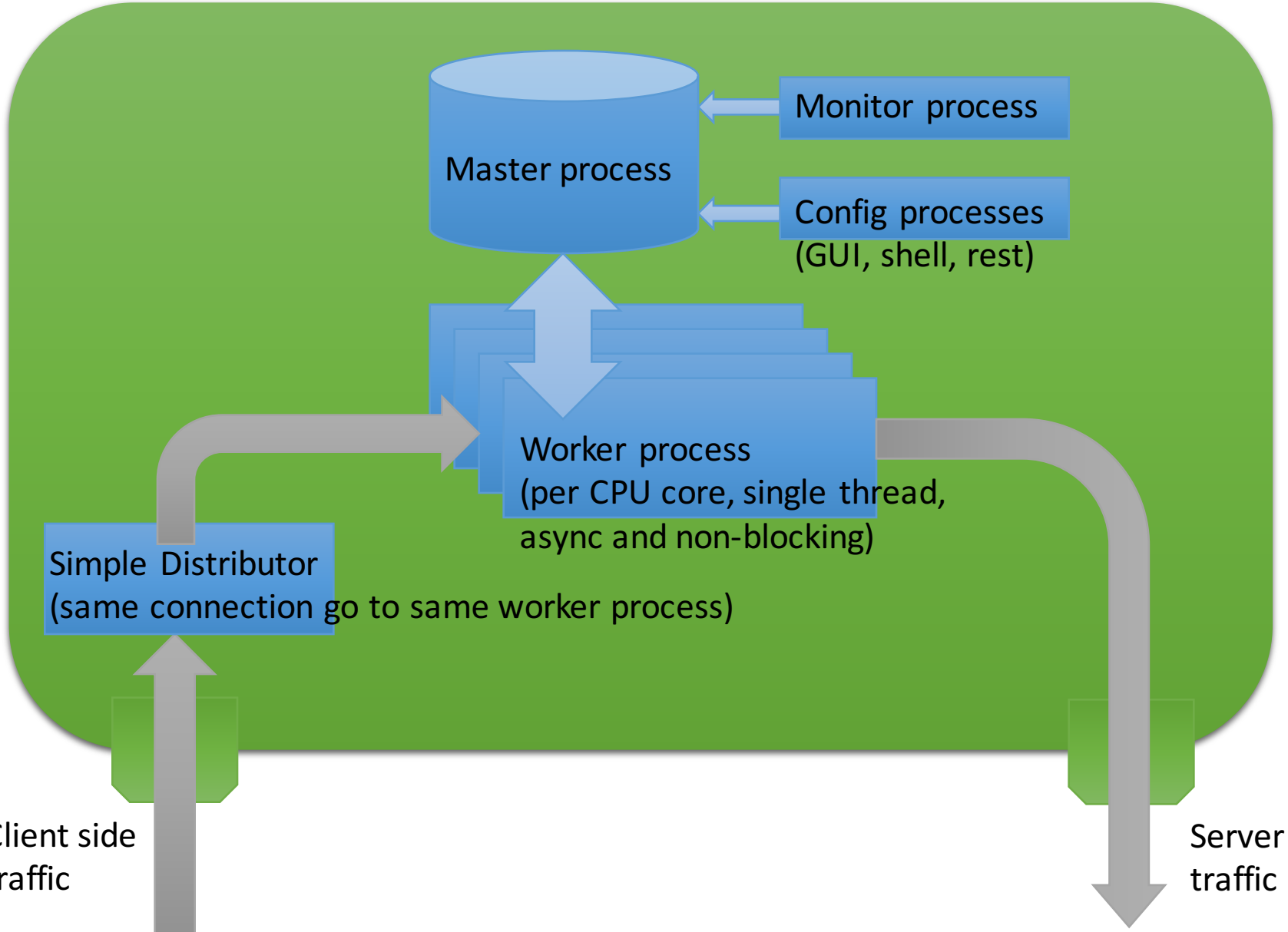


Flow id	Client	Virtual Client	Protocol	…
1	1.1.1.1:80	10.10.10.10:1111	http	…
2	2.2.2.2:443	10.10.10.10:2222	https	…
3	3.3.3.3:21	10.10.10.10:3333	ftp	…
…	…	…	…	…

F5 BIG-IP

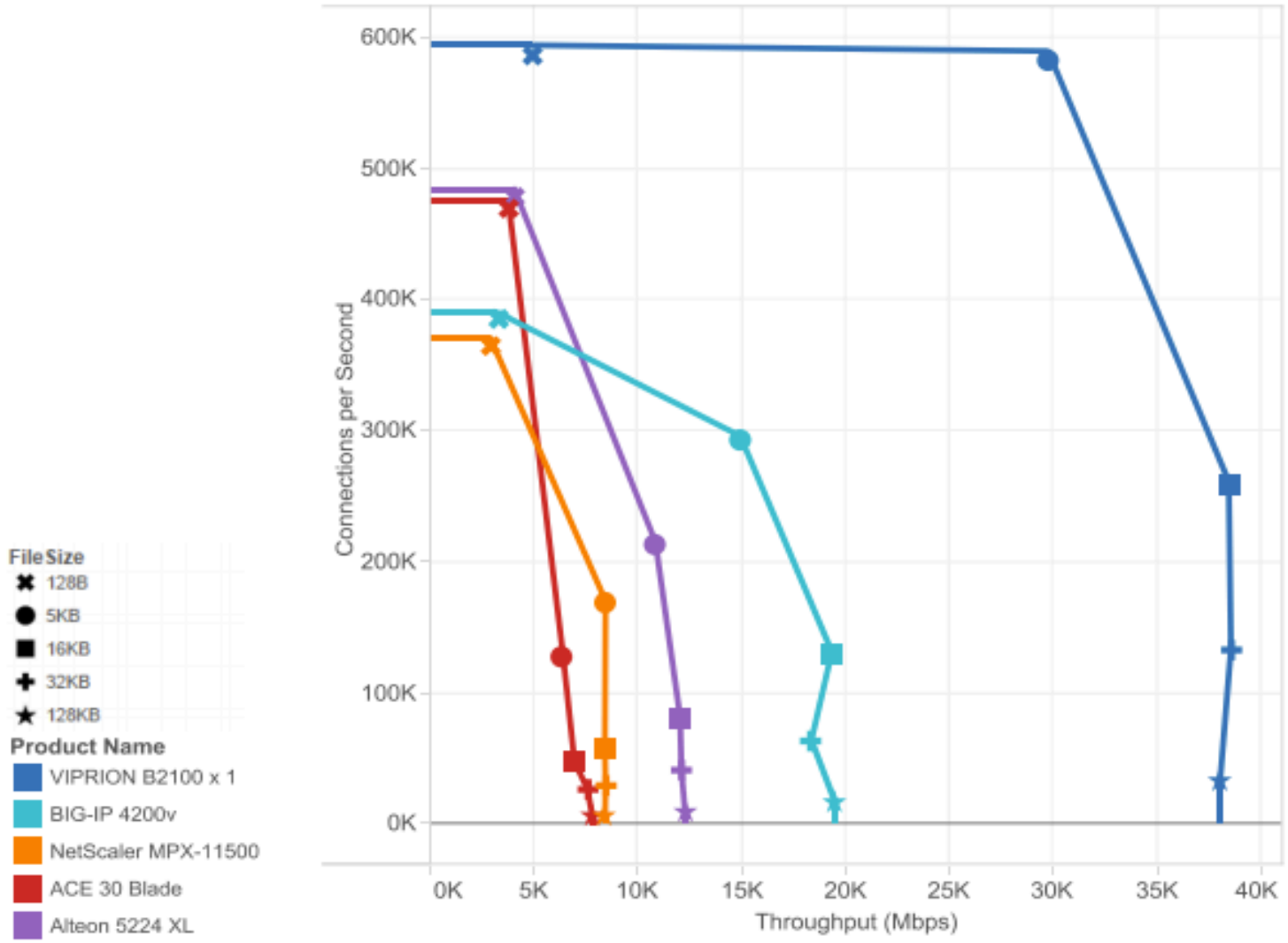
- Available on both hardware and virtual edition
- Based on CentOS, a linux distro (linux distro timeline)

Load Balancer Operating System



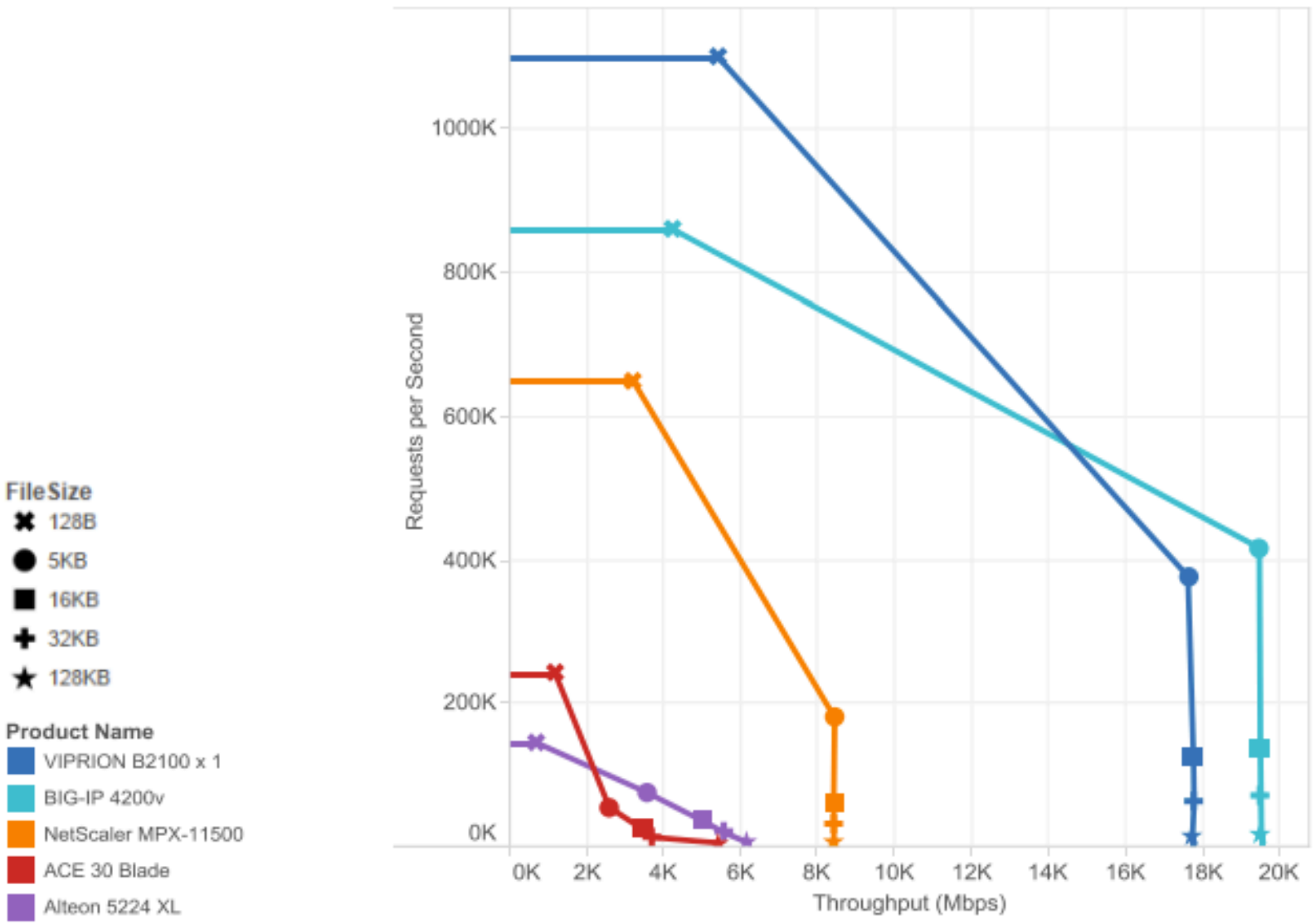
Performance

L4 Performance



Performance

L7 Performance



Project

Implement a load balancer using Python's built-in socket library.

- Server (short/long tcp connection)
  - Short: $ while true ; do nc –l ***proxy_port*** < index.html; done
  - Long: $ python socket_echo_server.py
- Client (short/long tcp connection)
  - Short: $ echo -n "GET / HTTP/1.0\r\n\r\n" | nc localhost ***proxy_port***
  - Long: $ telnet localhost ***proxy_port***
- Load Balancer
  - $ python load_balancer.py

```
→  Load Balancer  while true; do nc -l 8888 < src/server1.html; done
GET / HTTP/1.0

GET / HTTP/1.0

GET / HTTP/1.0

GET / HTTP/1.0

GET / HTTP/1.0
```

```
→  Load Balancer  while true; do nc -l 9999 < src/server2.html; done
GET / HTTP/1.0

GET / HTTP/1.0

GET / HTTP/1.0

GET / HTTP/1.0
```

```
→  Load Balancer                                                    [105/598]
→  Load Balancer  python src/demo.py
init client-side socket: ('127.0.0.1', 5555)
========================================flow start========================================
client connected: ('127.0.0.1', 50869) <==> ('127.0.0.1', 5555)
init server-side socket: ('127.0.0.1', 50870)
server connected: ('127.0.0.1', 50870) <==> ('127.0.0.1', 8888)
recving packets: ('127.0.0.1', 50869) ==> ('127.0.0.1', 5555) , data: ['GET / HTTP/1.0\r\
n\r\n']
sending packets: ('127.0.0.1', 50870) ==> ('127.0.0.1', 8888) , data: ['GET / HTTP/1.0\r\
n\r\n']
recving packets: ('127.0.0.1', 8888)  ==> ('127.0.0.1', 50870), data: ['resp from server1
\n']
sending packets: ('127.0.0.1', 5555)  ==> ('127.0.0.1', 50869), data: ['resp from server1
\n']
client ('127.0.0.1', 50869) has disconnected
========================================flow end==========================================
========================================flow start========================================
client connected: ('127.0.0.1', 50874) <==> ('127.0.0.1', 5555)
init server-side socket: ('127.0.0.1', 50875)
server connected: ('127.0.0.1', 50875) <==> ('127.0.0.1', 9999)
recving packets: ('127.0.0.1', 50874) ==> ('127.0.0.1', 5555) , data: ['GET / HTTP/1.0\r\
n\r\n']
sending packets: ('127.0.0.1', 50875) ==> ('127.0.0.1', 9999) , data: ['GET / HTTP/1.0\r\
n\r\n']
```

```
→  ~  while true; do echo -n "GET / HTTP/1.0\r\n\r\n" | nc localhost 5555; sleep 1; done
resp from server1
resp from server2
resp from server1
resp from server2
resp from server1
resp from server2
resp from server1
resp from server2
resp from server1
^C
→  ~
```

Follow our WeChat