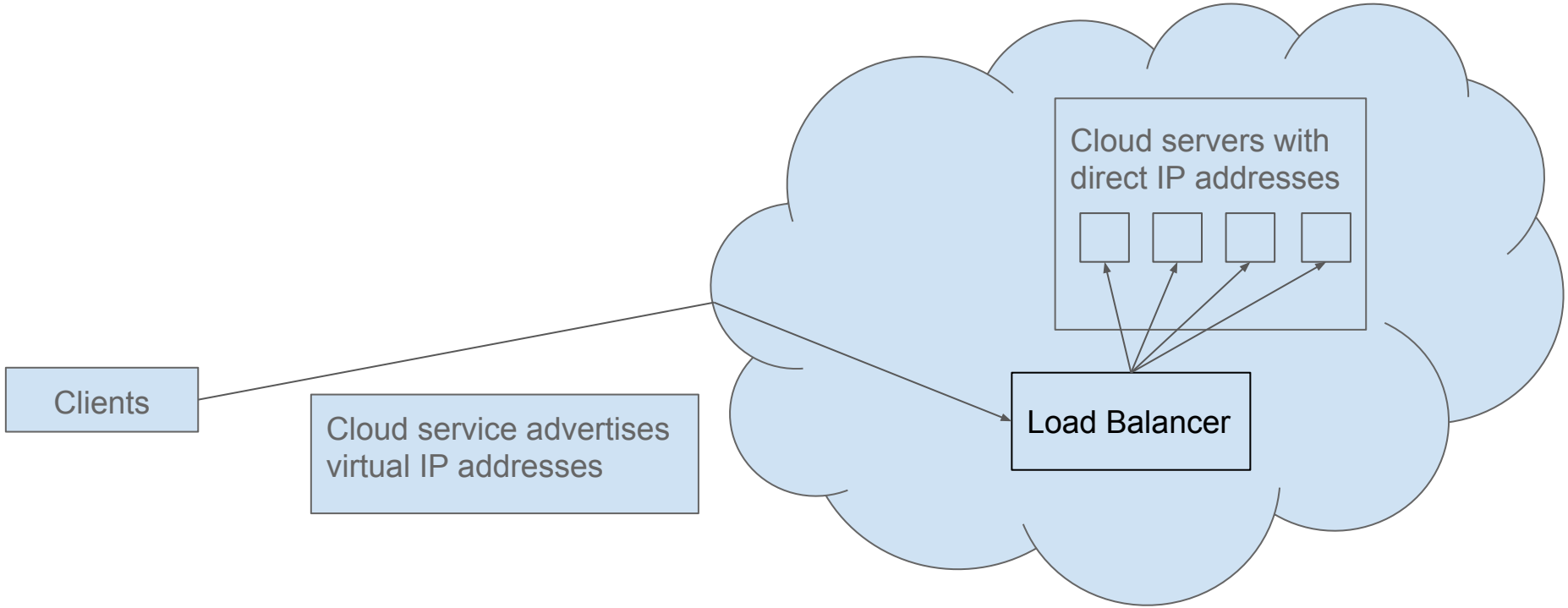
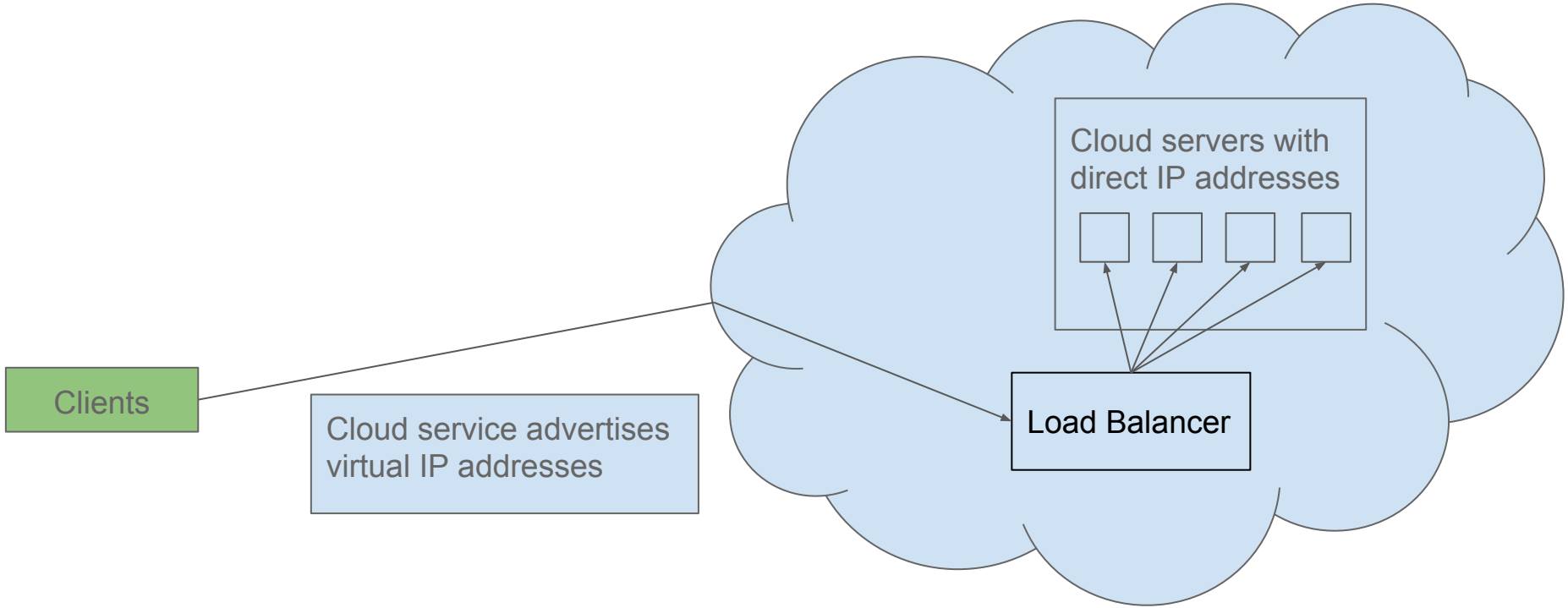
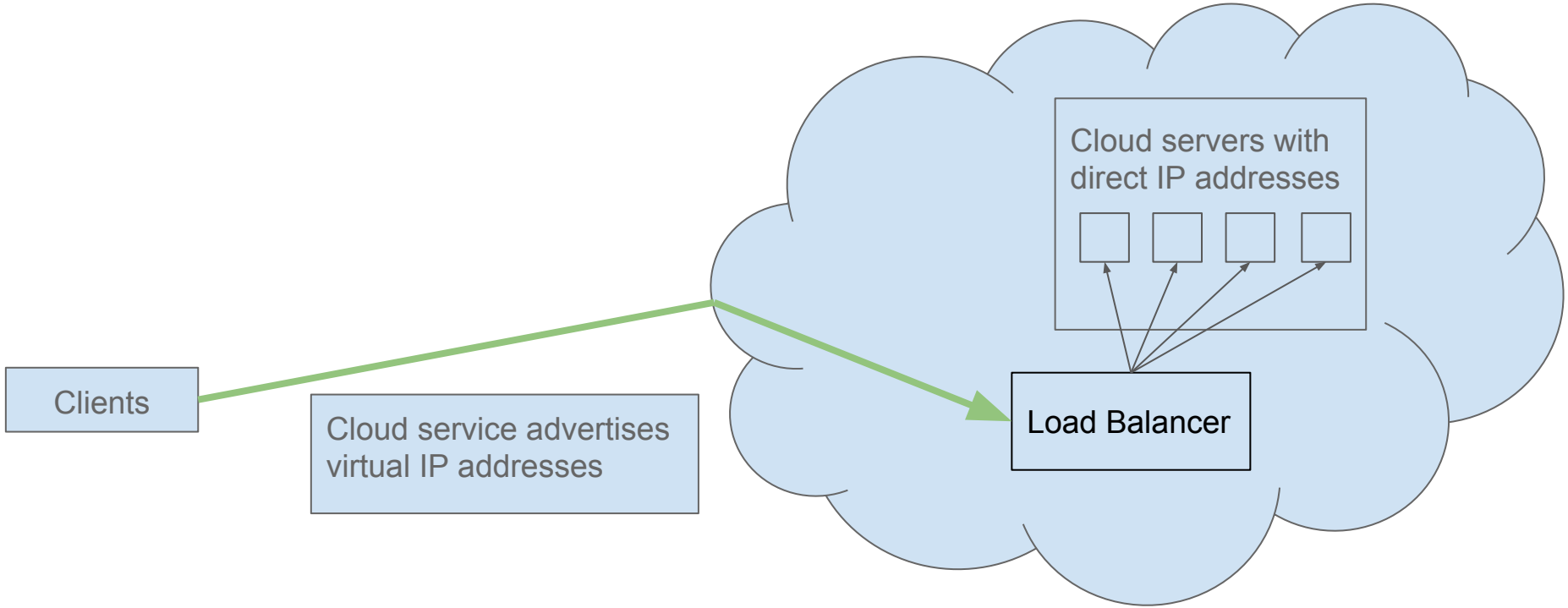
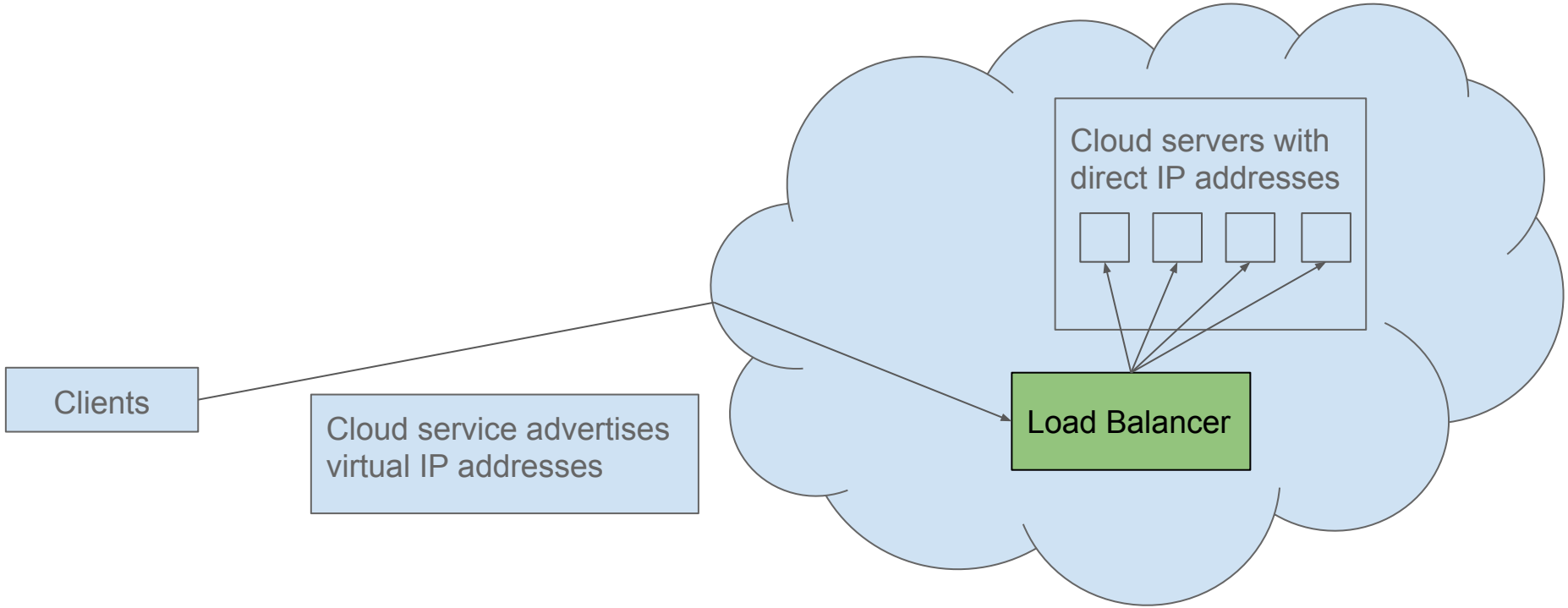
Load Balancing TCP Traffic in Cloud Systems

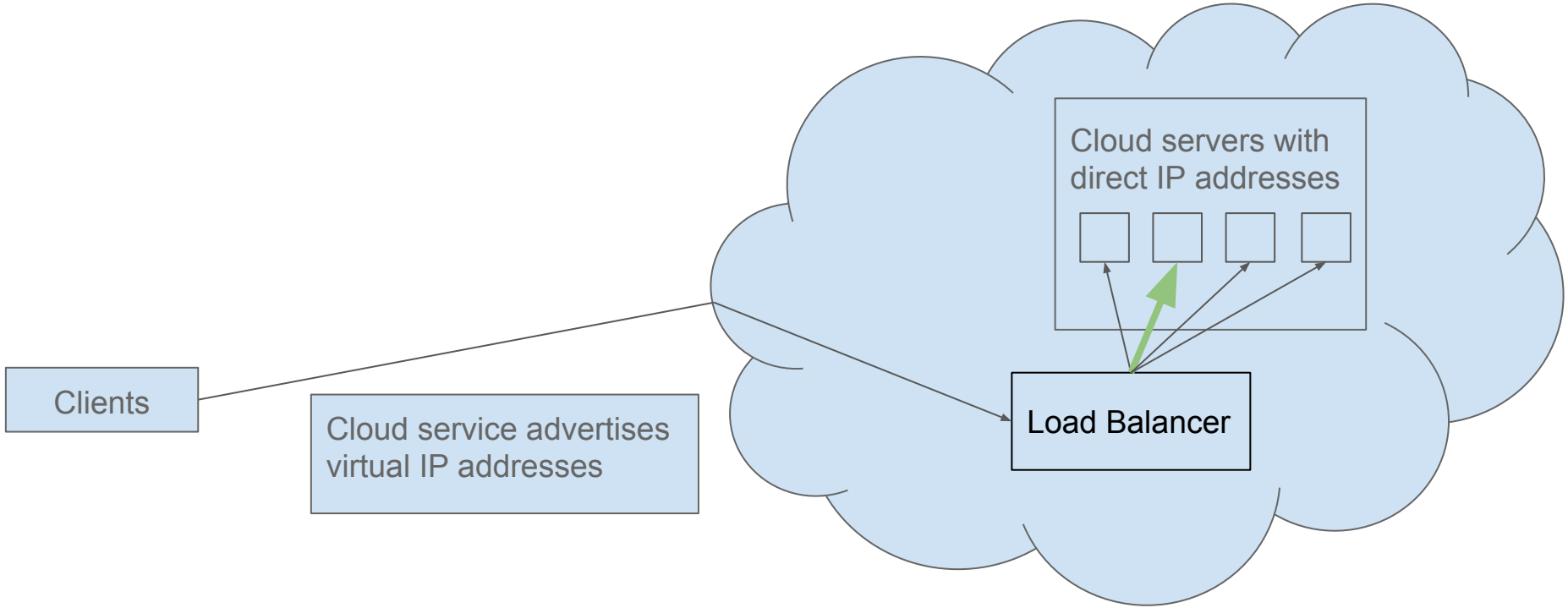
Michael Kirsche
Johns Hopkins University
Advanced Computer Networks
December 6, 2018

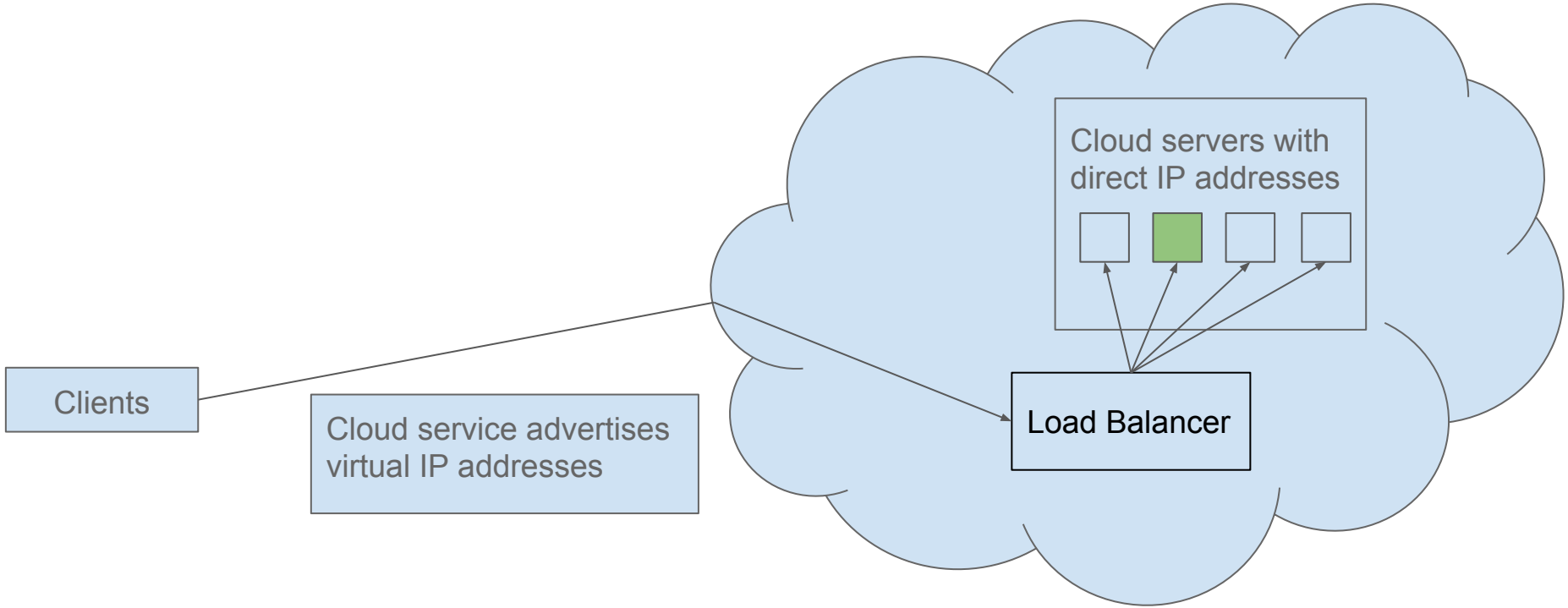

Cloud servers with
direct IP addresses
Clients
Cloud service advertises
virtual IP addresses
Load Balancer


Clients
Cloud service advertises
virtual IP addresses
Cloud servers with
direct IP addresses
Load Balancer

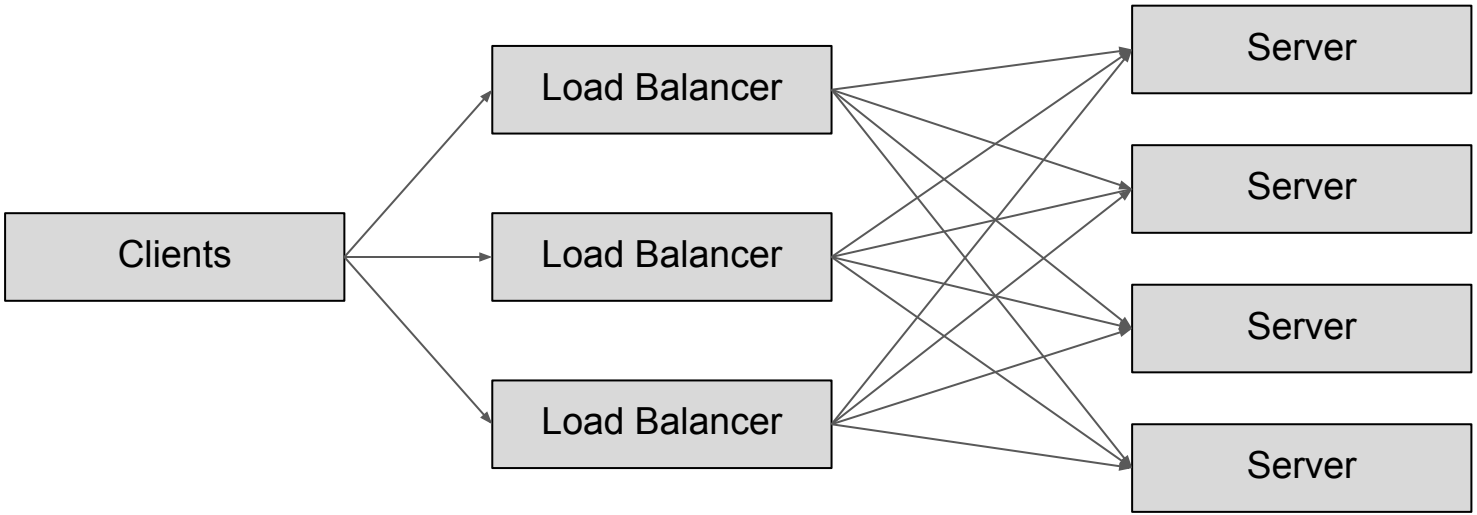

Clients
Cloud service advertises
virtual IP addresses
Cloud servers with
direct IP addresses
Load Balancer


Cloud servers with
direct IP addresses
Clients
Cloud service advertises
virtual IP addresses
Load Balancer


Cloud servers with
direct IP addresses
Clients
Cloud service advertises
virtual IP addresses
Load Balancer

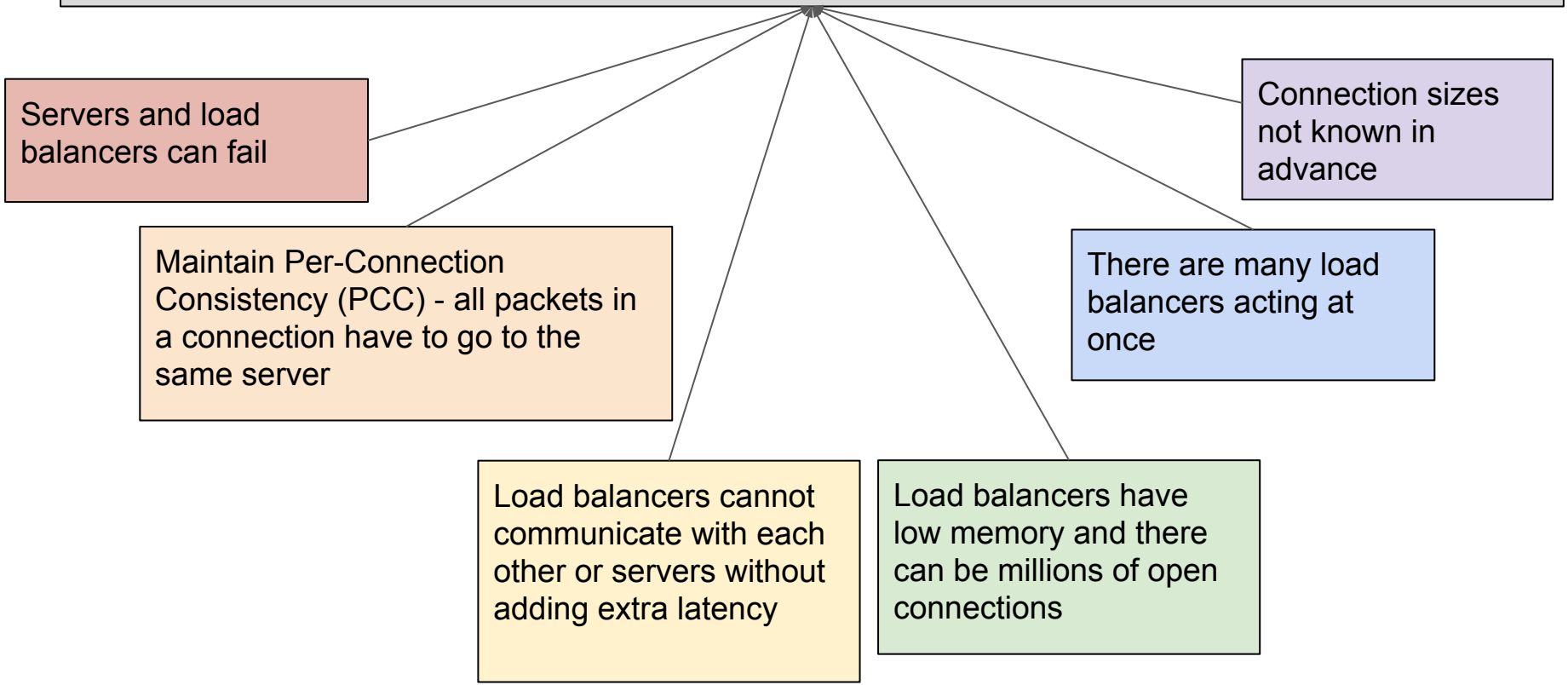

Cloud servers with
direct IP addresses
Clients
Cloud service advertises
virtual IP addresses
Load Balancer

Lots of servers, lots of load balancers, lots of ongoing connections...



Goal: Balance load among servers as evenly as possible

Goal: Balance load among servers as evenly as possible



Existing Approaches

- Maglev (Google) - each load balancer has the same hash function mapping packet headers to servers
- Ananta (Microsoft) - each load balancer has the same hash function mapping packet headers to servers
- Silkroad - load balancers on switches but still same hashing approach
- Duet - load balancers on switches but still same hashing approach
- GoneWild - hash function but handle servers with different desired load by mapping differently sized portions of hash space to servers

The State of the Art

How to solve the load balancing problem?

“Pick a random server!”
-Ananta

“Pick a random server!”
-Maglev

“Pick a random server!”
-GoneWild

“Pick a random server!”
-Duet

“Pick a random server!”
-Silkroad

Can we do better than randomly choosing a server for each connection to be sent to?

What information would help the load balancer make a better decision about which server to send a packet to?

What information would help the load balancer make a better decision about which server to send a packet to?

- Some measure of server load
- But even more importantly…

What information would help the load balancer make a better decision about which server to send a packet to?

- Some measure of server load
- But even more importantly…
  - Where earlier packets in that connection went!!!

How to give that information to the load balancer?

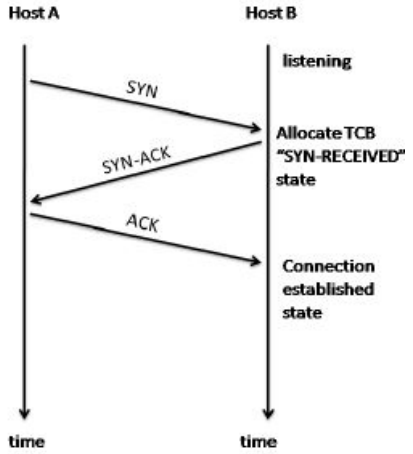
- Not enough memory to store per-connection information on a load balancer
- Instead, encode it in each packet before it reaches the load balancer
- But then the client needs load balancing information - how?

TCP Traffic

- In most cloud services, vast majority of traffic is TCP
  - Table below shows 2013 traffic distribution in Amazon EC2 and Windows Azure
- Every TCP connection starts with a 3-way handshake
  - Client sends SYN packet to server
  - Server sends SYN-ACK packet to client
  - Client sends ACK packet to server
  - All of these packets go through the load balancer

	EC2		Azure		Overall	
Protocol	Bytes	Flows	Bytes	Flows	Bytes	Flows
ICMP	0.01	0.03	0.01	0.18	0.01	0.06
HTTP (TCP)	16.26	70.45	59.97	65.41	24.24	69.48
HTTPS (TCP)	80.90	6.52	37.20	6.92	72.94	6.60
DNS (UDP)	0.11	10.33	0.10	11.59	0.11	10.58
Other (TCP)	2.40	0.40	2.41	1.10	2.40	0.60
Other (UDP)	0.28	0.19	0.31	14.77	0.28	3.00
Total	100	100	100	100	100	100

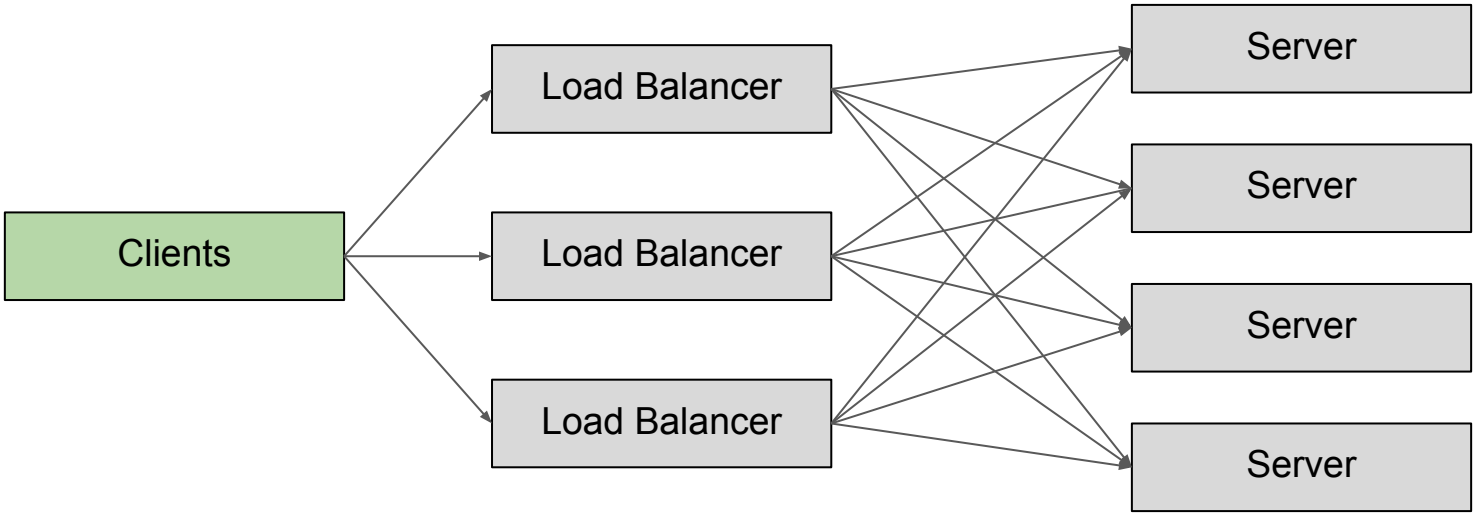
Table 2: Percent of traffic volume and percent of flows associated with each protocol in the packet capture.



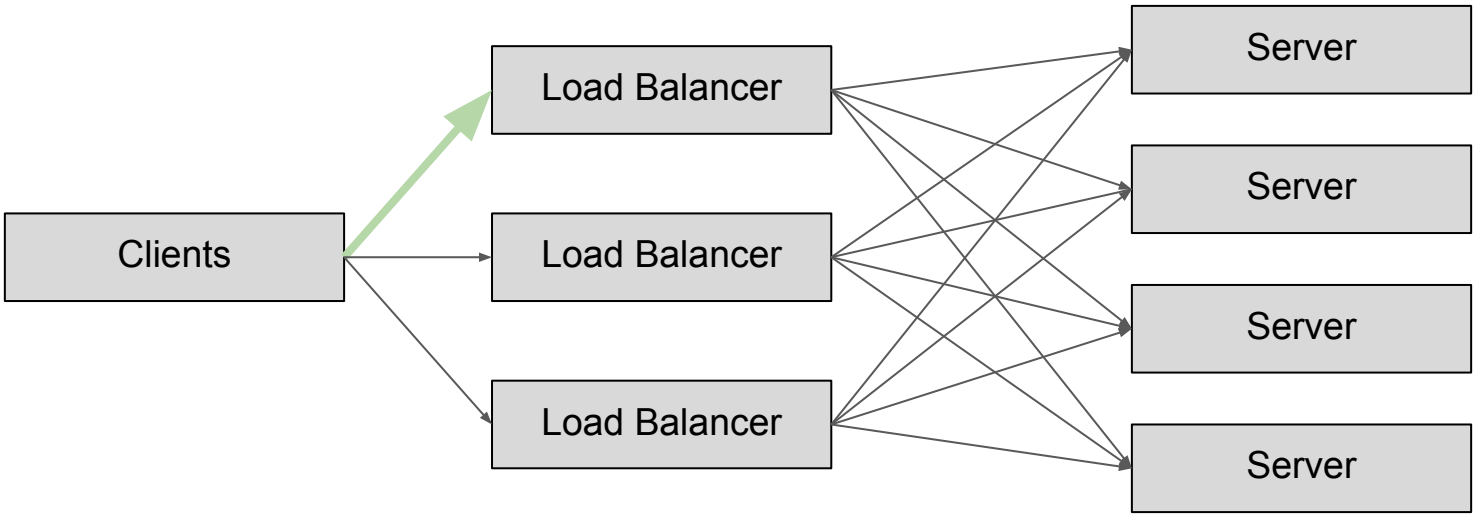
Using TCP Handshake

- Load balancing decisions for the entire connection made when SYN packet received
- Decision sent back to client in SYN-ACK packet
- This decision can then be added to all future packets in the connection by the client

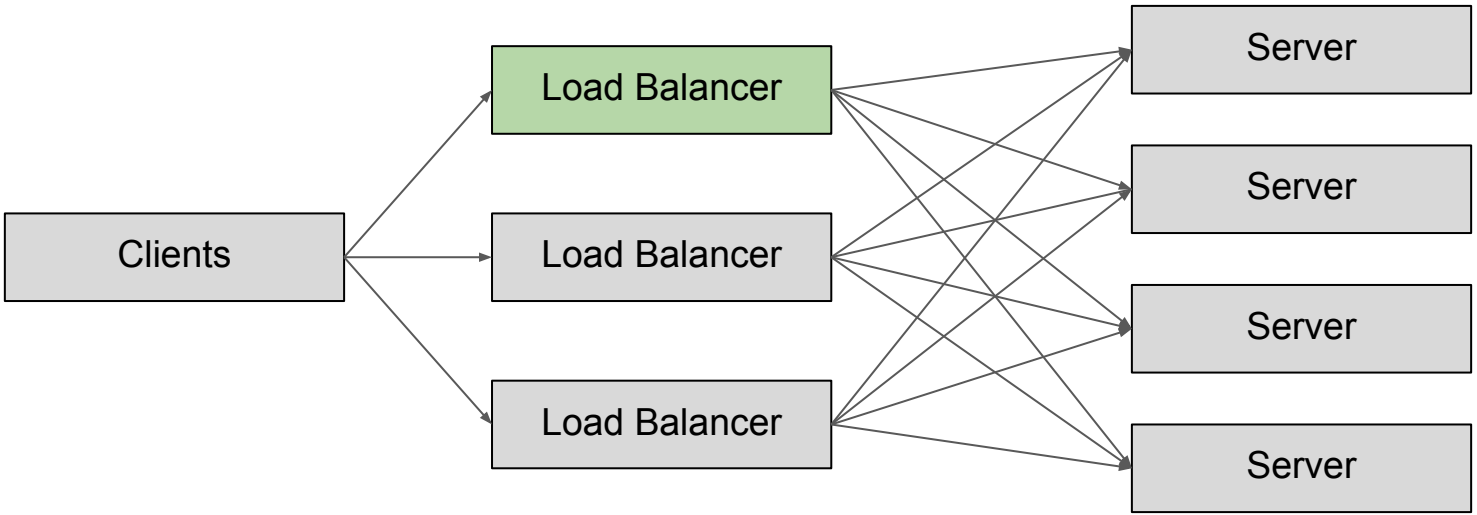
Client sends SYN packet to load balancer



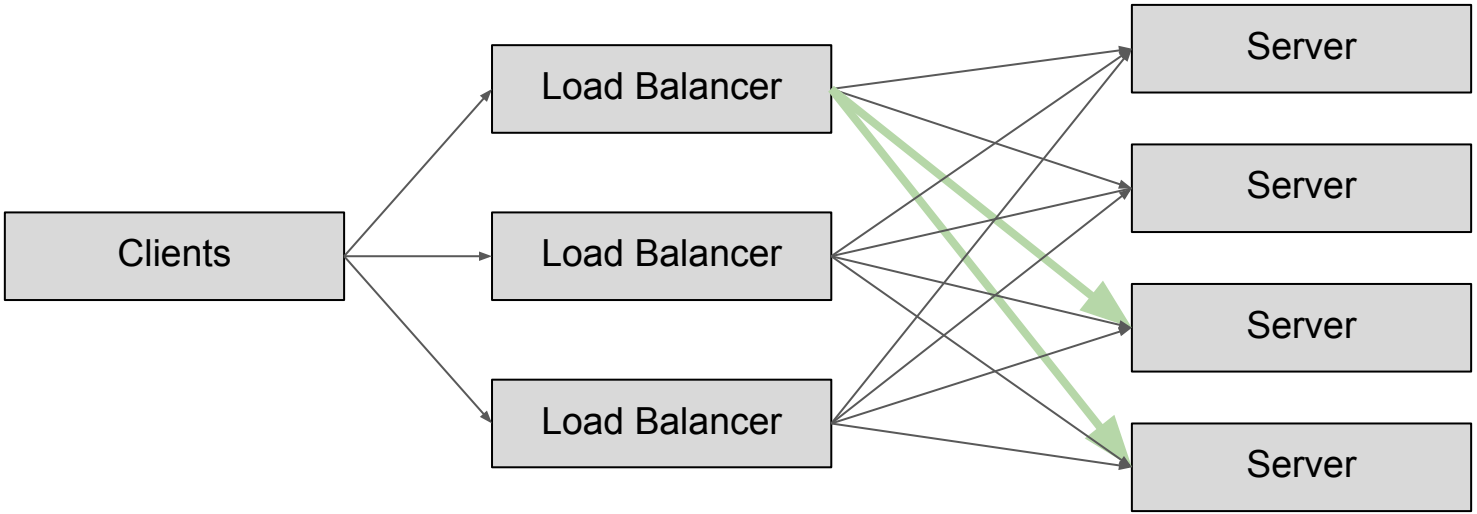
Client sends SYN packet to load balancer



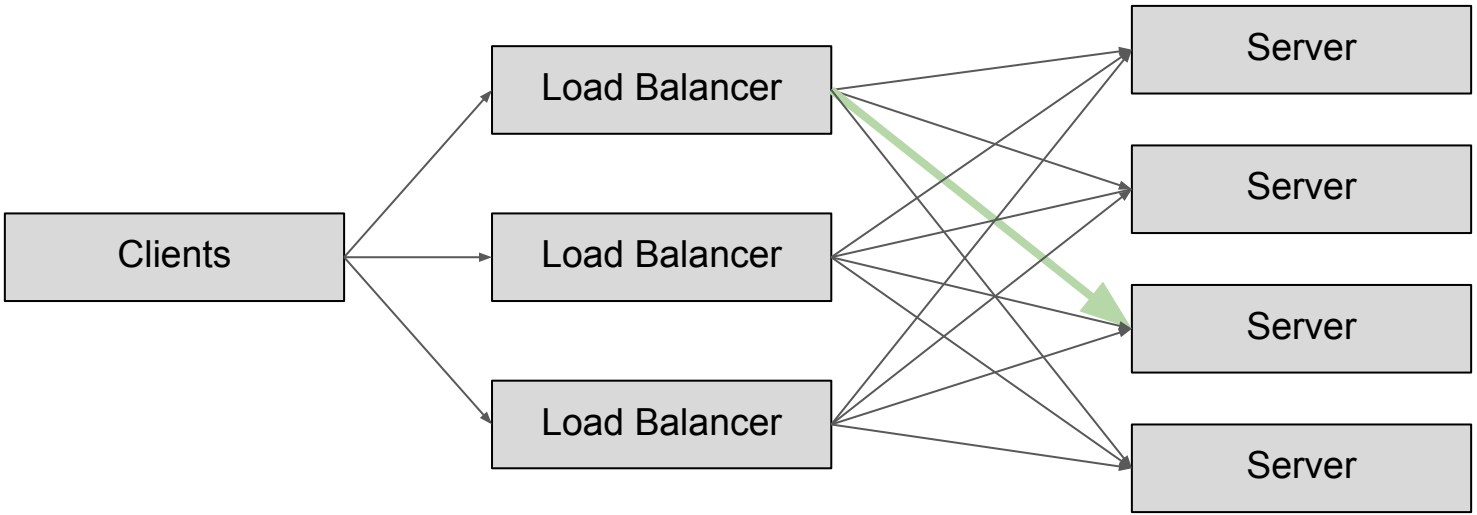
Client sends SYN packet to load balancer



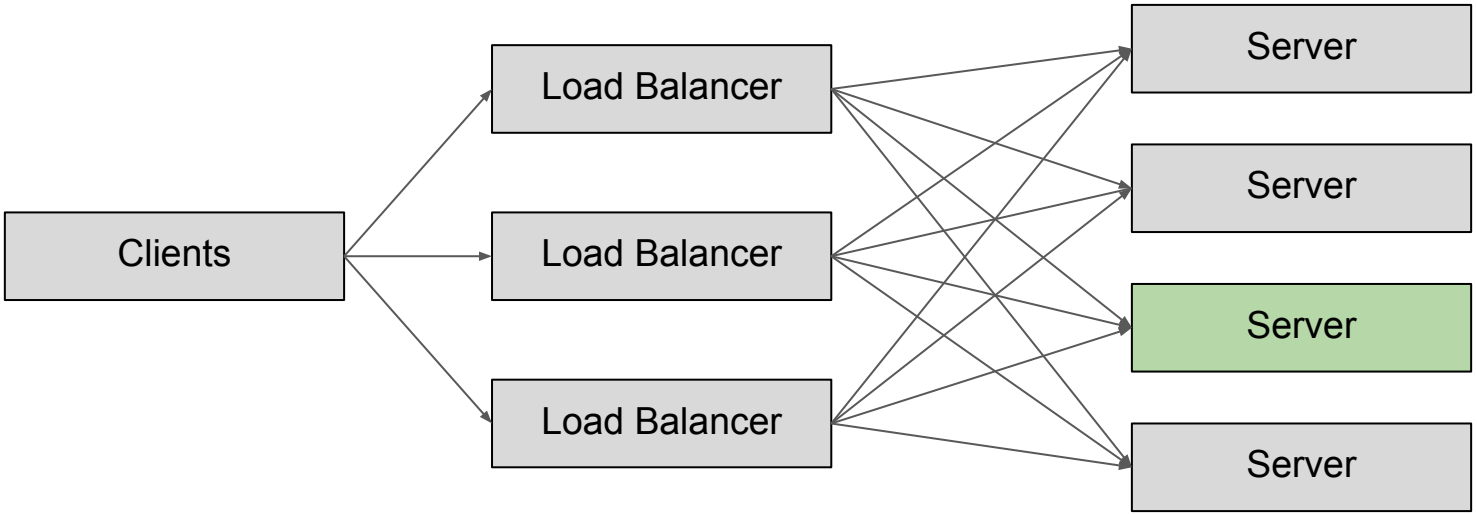
Client queries multiple servers
for their status



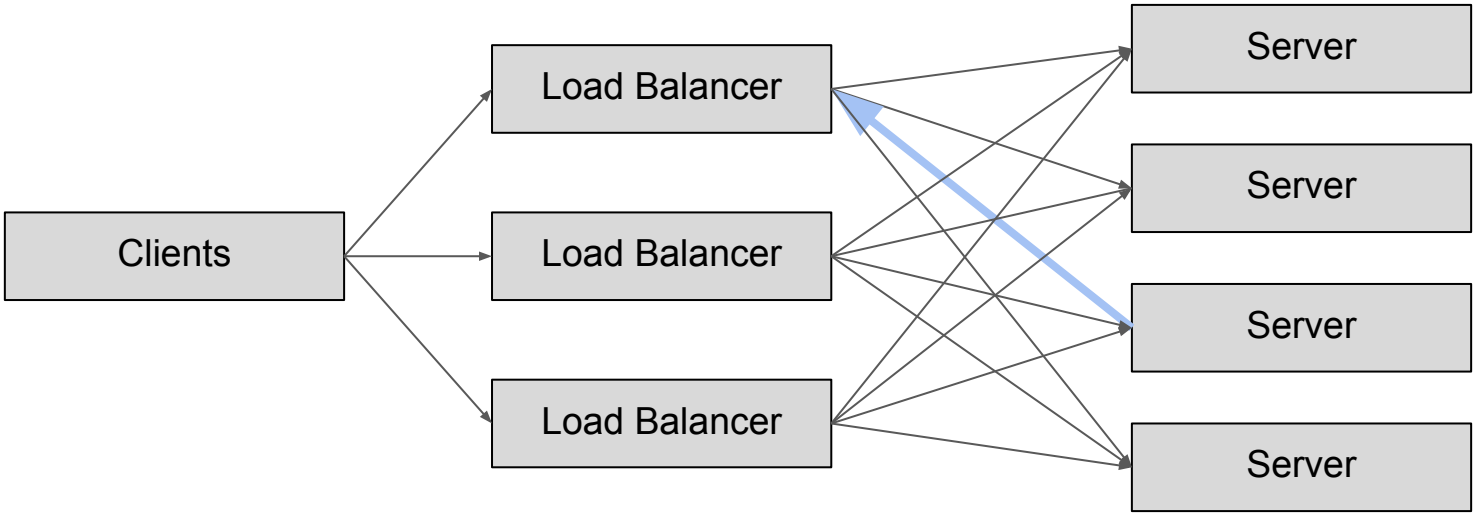
Client chooses a server to route this connection to



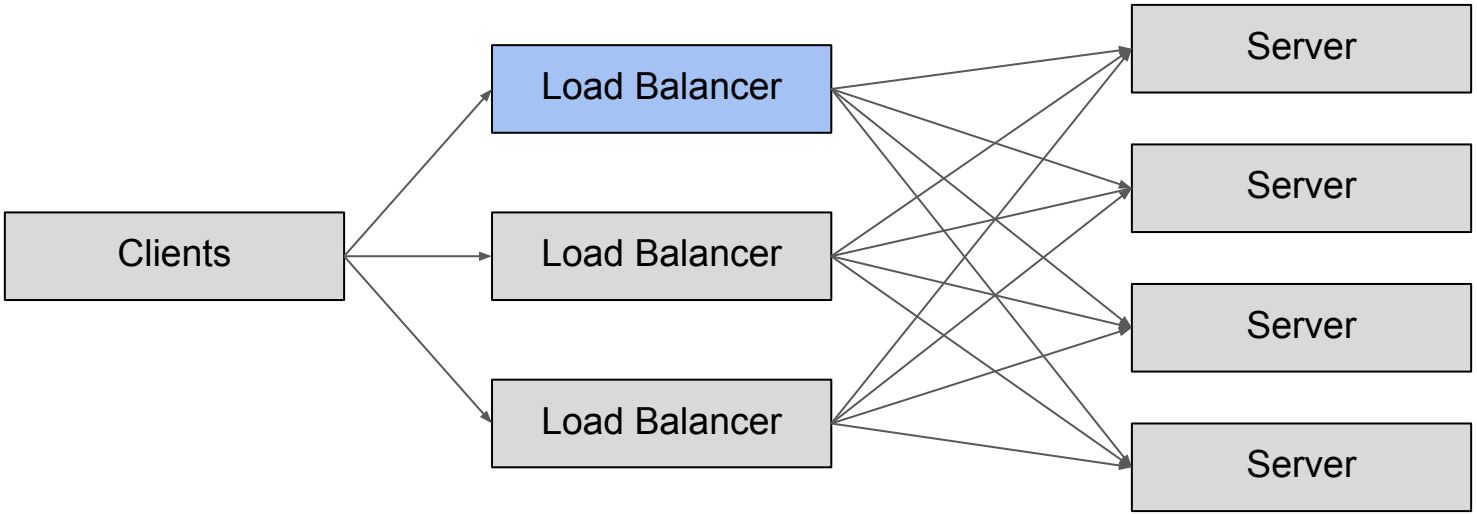
Client sends SYN packet to chosen server



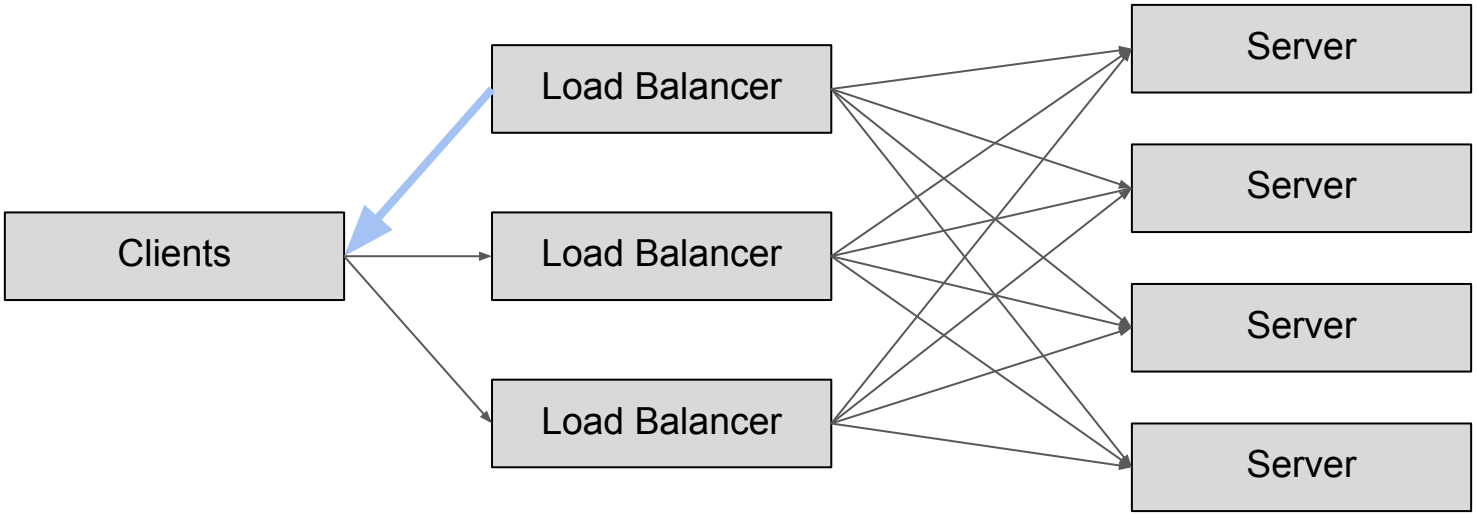
Server sends SYN-ACK packet to load balancer



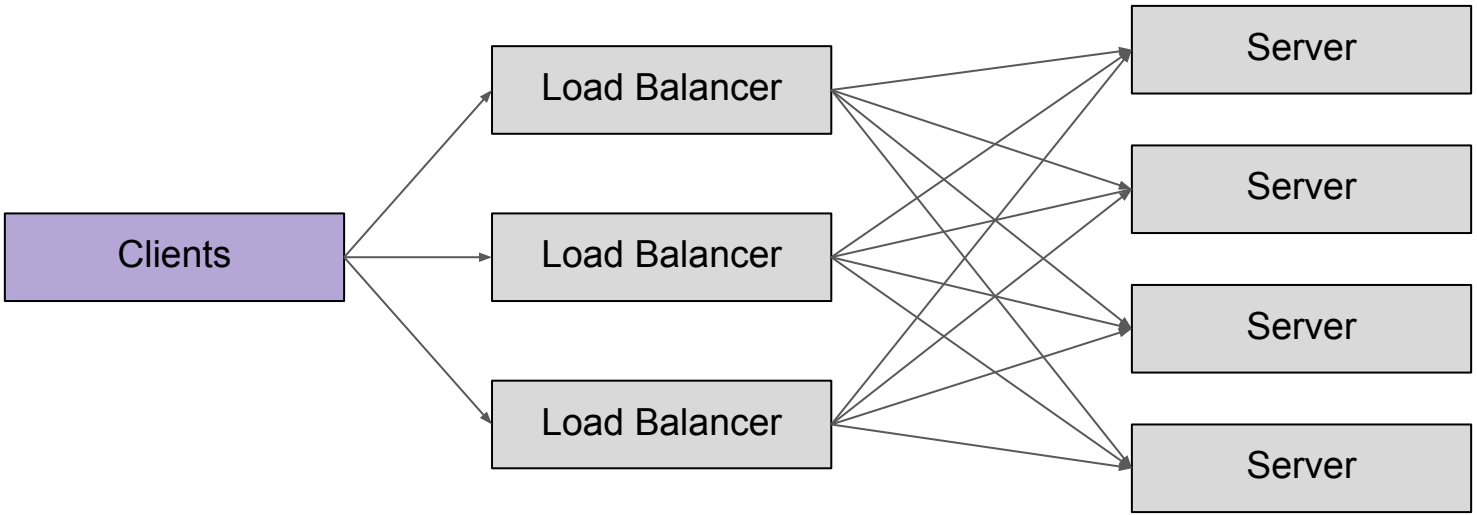
Load balancer encodes choice of server in SYN-ACK packet



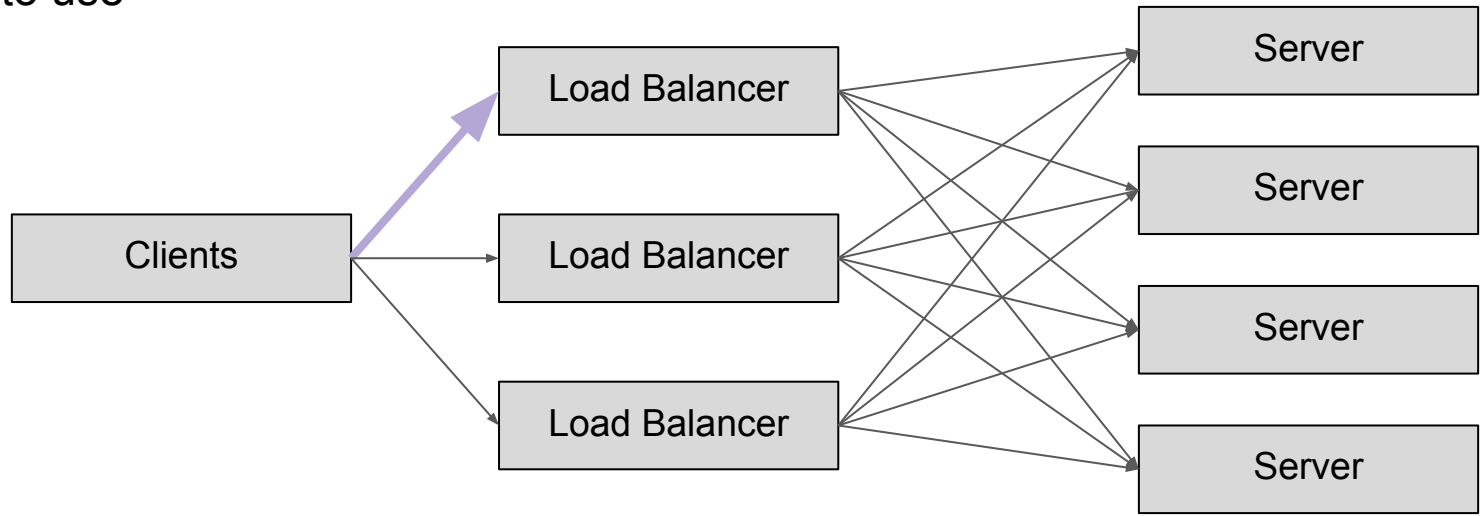
Load balancer sends
SYN-ACK packet to client



Client encodes server choice into ACK packet as well



Client sends remaining packets with info about which server to use



Potential Issues

- Choice of server can take a prohibitive amount of space in packet headers
- Algorithm is blocking - load balancer has to wait for multiple servers to respond before sending SYN packet

Potential Issues

- Choice of server can take a prohibitive amount of space in packet headers
- Algorithm is blocking - load balancer has to wait for multiple servers to respond before sending SYN packet

Best of 2 algorithm:

- For each connection, hash packet header with two different hash functions to choose two servers
- Query load of both servers and send packet to server with less load
- Encode choice with a single bit (0 for first hash function, 1 for second hash function)

Method - Best of 2 Algorithm

3 cases:

1. If the traffic is not TCP, use hash function to randomly choose server to send connection to
2. If the traffic is SYN packet P initiating TCP connection, use two hash functions $h_1(P)$and $h_2(P)$ to select two servers. If load($h_1(P)$) < load($h_2(P)$), then send packet to $h_1(P)$, increase load($h_1(P)$), and append 0-bit to ACK packet when sending it back to client
3. If the traffic is other TCP packet P, client appends the bit that was received in this connection's SYN packet to this packet, and load balancer sends packet to $h_1(P)$ and increments load($h_1(P)$) if this bit is 0 and $h_2(P)$ otherwise.

Potential Issues

- Choice of server can take a prohibitive amount of space in packet headers
- Algorithm is blocking - load balancer has to wait for multiple servers to respond before sending SYN packet

Sampling:

- In each load balancer, keep hash table storing load for each server
- Keep track of when each server's load was last updated
- Only query actual server if load information is outdated enough

Evaluation - Network Traffic Model

Cloud service is parameterized by the tuple (N, M, P_{TCP}, T_{DELAY}, T_{LENGTH})

- N = number of servers
- M = number of connections which pass through the network
- P_{TCP} = proportion of connections which are TCP connections
- T_{DELAY} = distribution for time between the starts of consecutive connections
- T_{LENGTH} = distribution for the number of packets in each connection

Not included in the model

- Number of load balancers - all load balancers are combined to create load balancing pool - assume no knowledge of any prior connections or packets since they could have gone through different load balancers
- Number of clients - assume that each connection could come from a client with no knowledge of earlier connections

Load Balancing Algorithms as Functions

- Each connection has a unique ID which consists of information from the packet headers
  - Same for all packets in the connection
  - Different for packets of different connections
  - For example, source/destination IP and port plus IP protocol (Ananta)
  - Denote this information for a given packet P
- Each server in network has a unique integer ID in range [0, N)
- Standard load balancing algorithm: Function mapping P to [0, N)
  - Guarantees per-connection consistency
  - P randomly generated for each connection
  - We assume that function has equal probability of mapping to any server

Enhanced Load Balancing Function

Three parts

1. Function for mapping non-TCP packet headers to servers
2. Function for SYN packets which maps packet header plus server load information to server plus additional information X
3. Function for other TCP packets which maps packet header plus additional information X to servers

Best of Two Algorithm

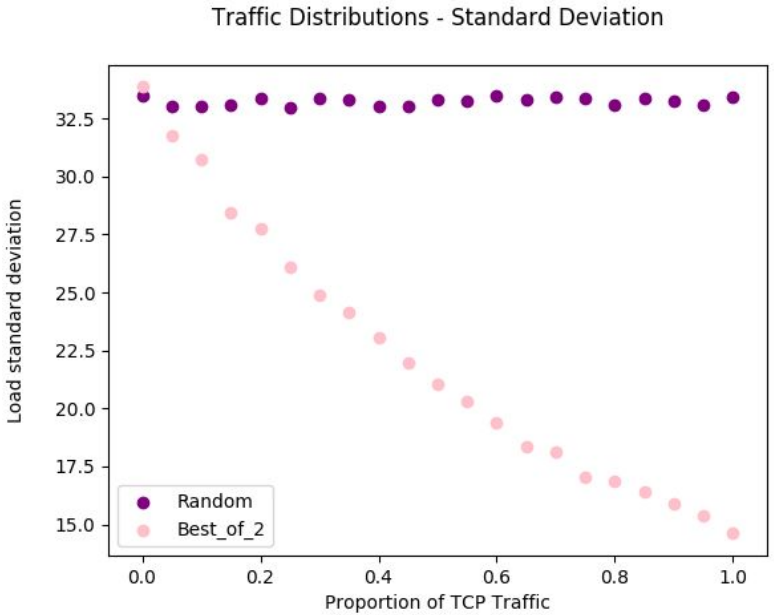
Three parts

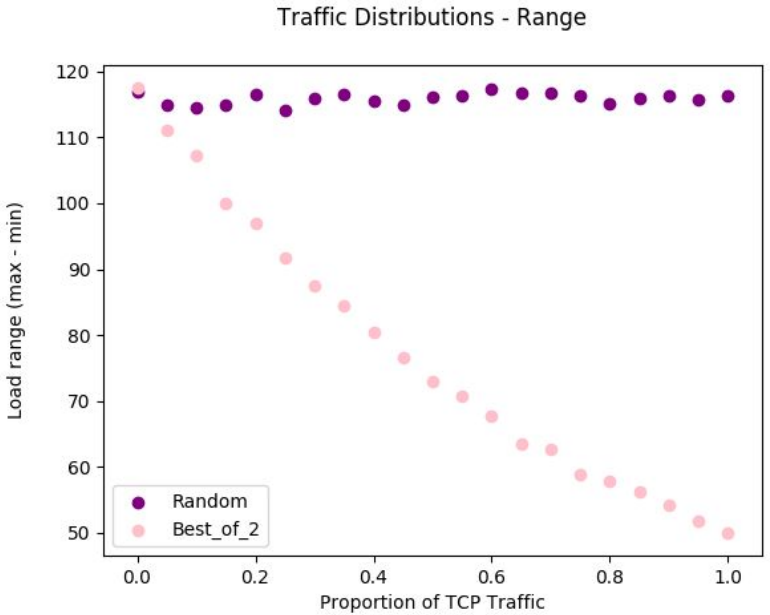
1. Function for mapping non-TCP packet headers to servers - **Hash function**
2. Function for SYN packets which maps packet header plus server load information to server plus additional information X - **Two hash functions, returning server with smaller load, and single bit depending on which hash function was used**
3. Function for other TCP packets which maps packet header plus additional information X to servers - **One of two hash functions depending on if X is 0 or 1**

Evaluation

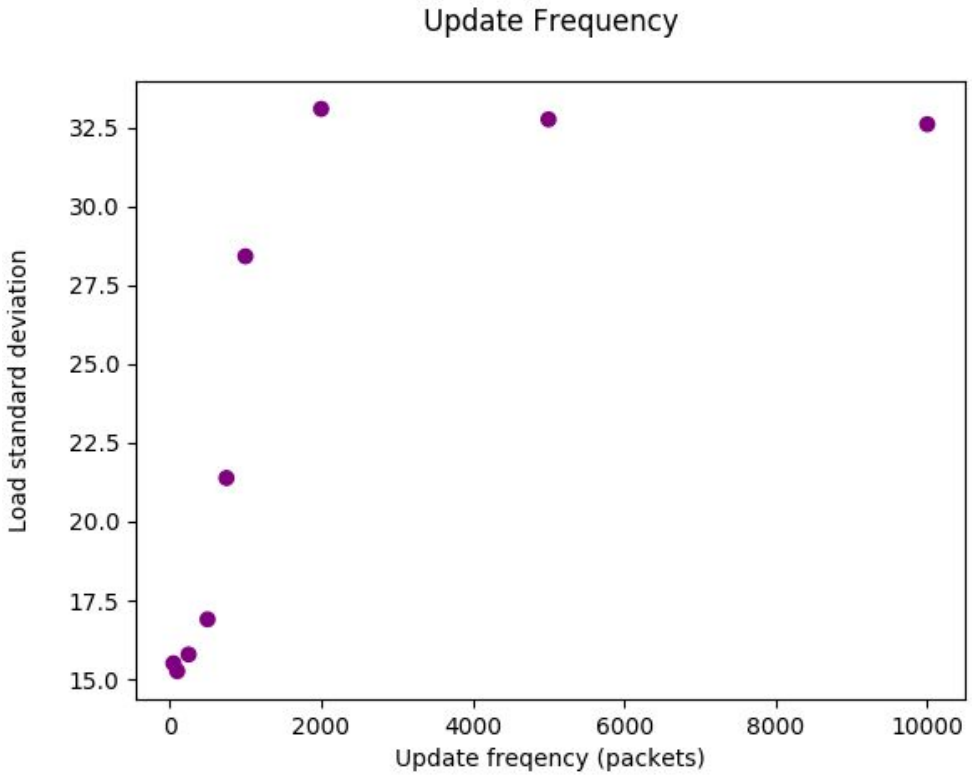
- Default Parameters:
  - 10 Servers
  - 95% TCP traffic
  - 100 requests (repeated 1000 times)
  - Request size uniformly distributed in range [1, 20]
  - Time between requests log-normal distributed with mean of 4 units and standard deviation of 2 units
  - 1 packet sent per time unit
- Measures of load balancing quality:
  - Range of count of packets sent to each server
  - Standard deviation of packets sent to each server

Varying proportion of TCP traffic





Non-blocking algorithm



Used stale load information to reduce number of connections which were blocked by waiting for querying of server load - tradeoff of frequency of updating information with quality of load balancing

Generalizing Best of 2 Algorithm

Algorithm	Standard Deviation	Range
Baseline	118.711	594.300
Best-of-2	22.915	105.060
Best-of-4	7.061	38.080
Best-of-8	5.736	29.360

Future Work

- Find ways to improve load balancing for other types of traffic
- Incorporate into real load balancers
- Explore other ways of using a few bits to encode load balancing information

Conclusion

- Getting information to help load balancing to the load balancers is difficult
- TCP traffic not only common but also provides great mechanism for passing such information with little additional overhead
- Even one bit of information can make a big difference
- Questions?