Network Applications: Load Balancing among Homogeneous Servers; Application Overlays (P2P)

Qiao Xiang

https://qiaoxiang.me/courses/cnns-xmuf21/index.shtml

11/4/2021

This deck of slides are heavily based on CPSC 433/533 at Yale University, by courtesy of Dr. Y. Richard Yang.

Outline

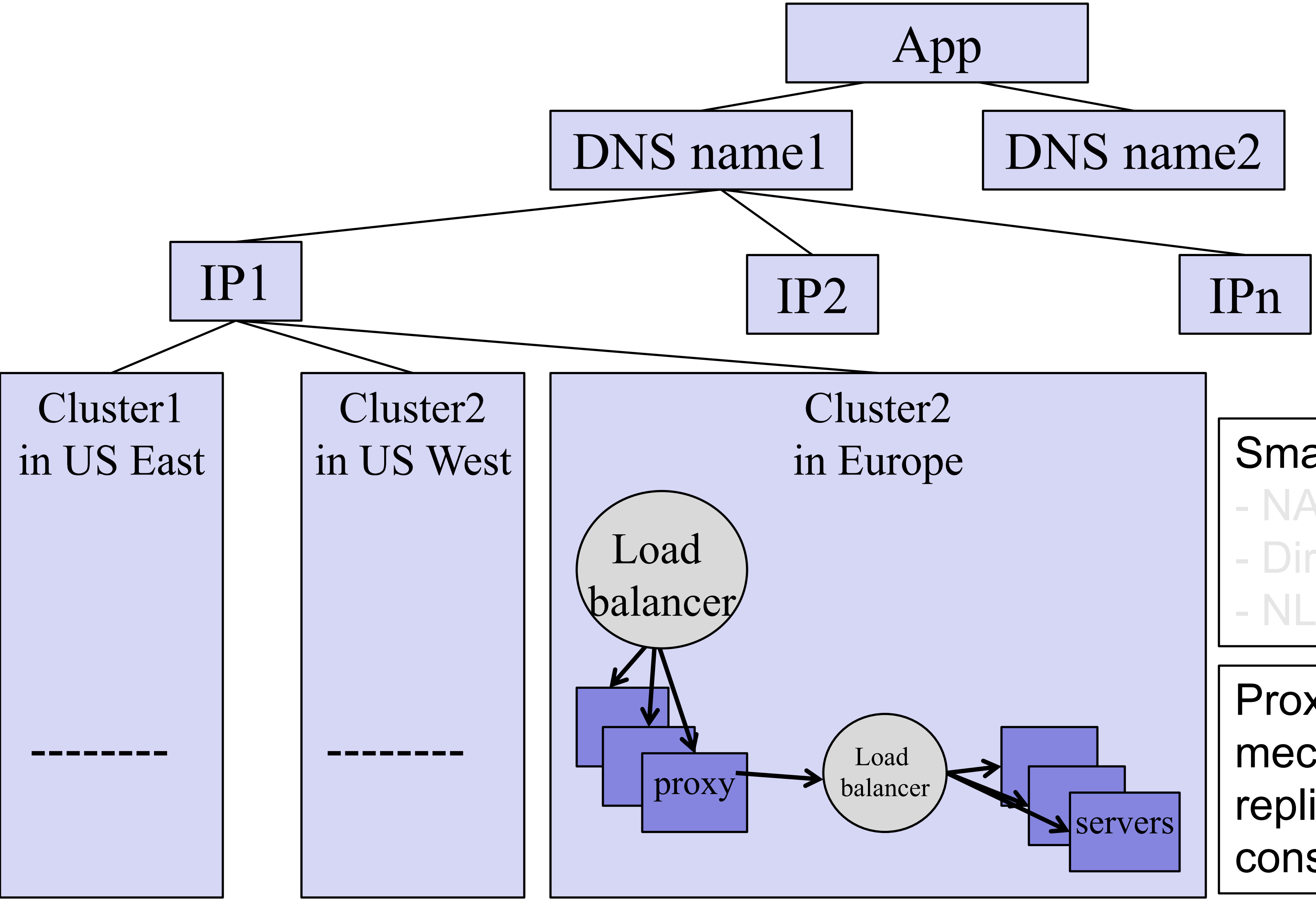
- Admin and recap
- Multi-servers
  - Basic issues
  - Load direction
    - DNS (IP level)
    - Load balancer/smart switch (sub-IP level)
- Application overlays (distributed network applications) to
  - scale bandwidth/resource (BitTorrent)

Admin

- Assignment three status and questions?

- Midterm exam 1: Thursday next week (i.e., Nov. 11)
  - cover from lecture 1 to lecture 16 (the one on Nov. 9)

Recap: Direction Mechanisms



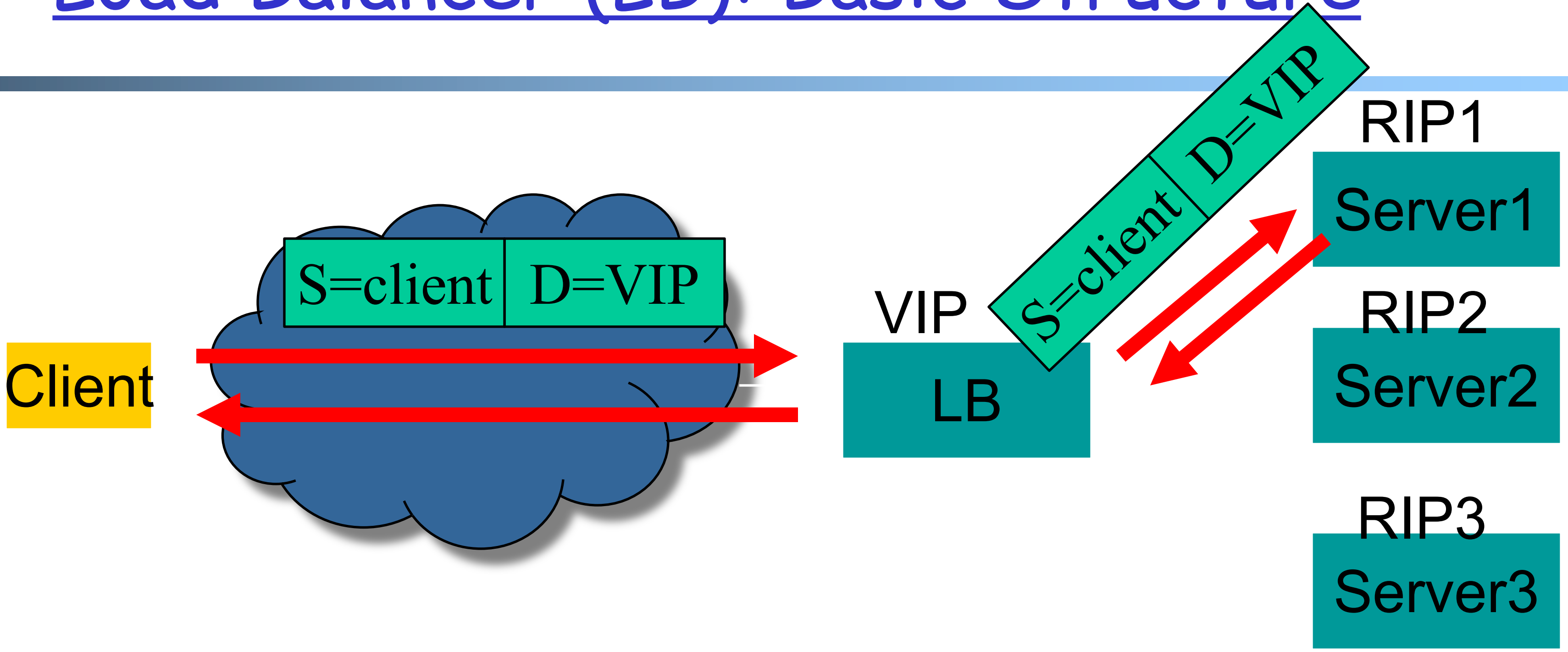
- cname
- hierarchy

Smart switch/LB

- NAT rewrite
- Direct reply
- NLB

Proxy as a mechanism for replica content consistency

Load Balancer (LB): Basic Structure

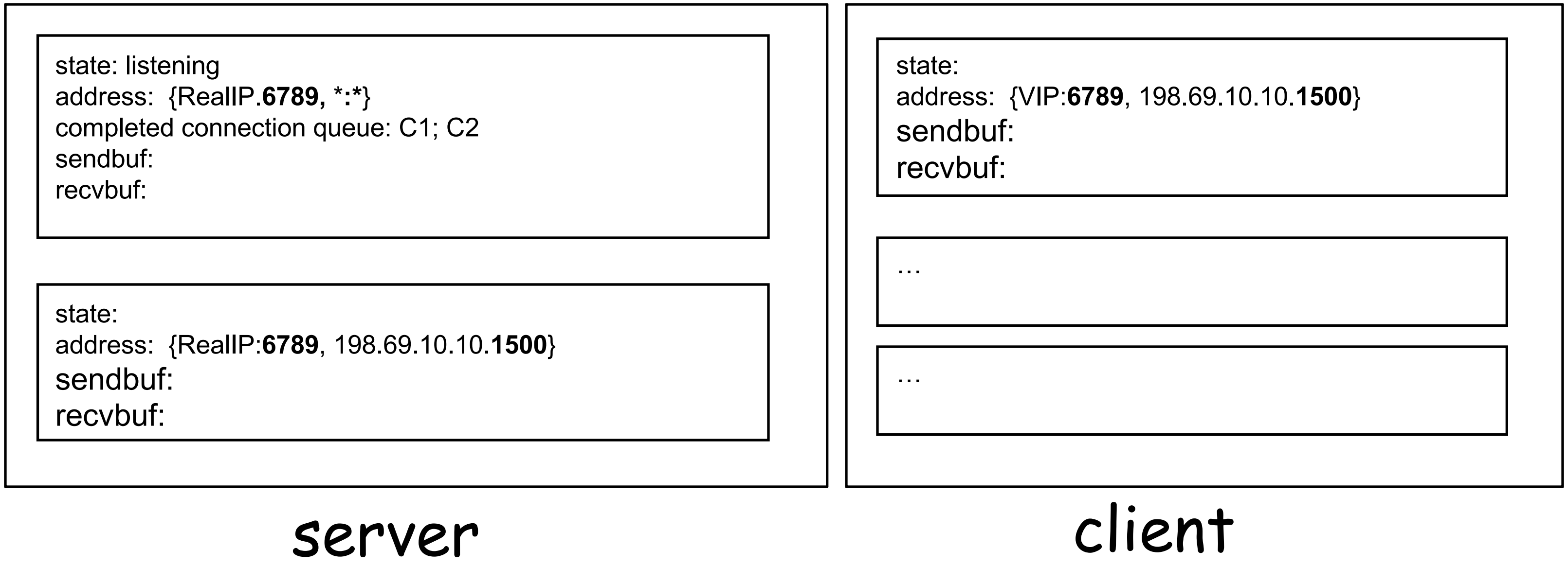


Problem of the basic structure?

Problem

S=client	D=VIP

- Client to server packet has VIP as destination address, but real servers use RIPs
  - if LB just forwards the packet from client to a real server, the real server drops the packet
  - reply from real server to client has real server IP as source -> client will drop the packet



state: listening
address: {RealIP.6789, *:*}
completed connection queue: C1; C2
sendbuf:
recvbuf:

state:
address: {RealIP:**6789**, 198.69.10.10.**1500**}
sendbuf:
recvbuf:

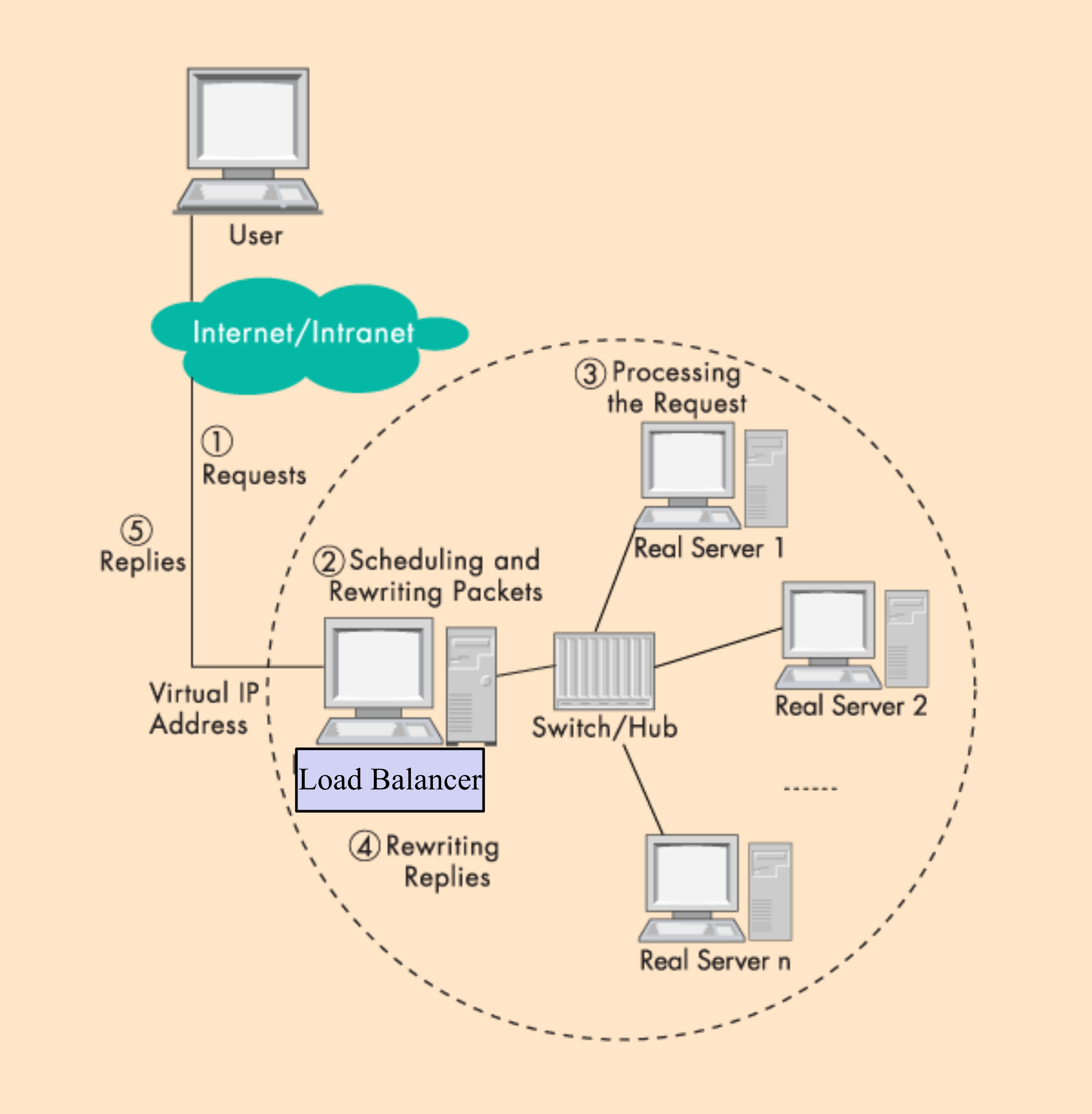
state:
address: {VIP:**6789**, 198.69.10.10.**1500**}
sendbuf:
recvbuf:

…

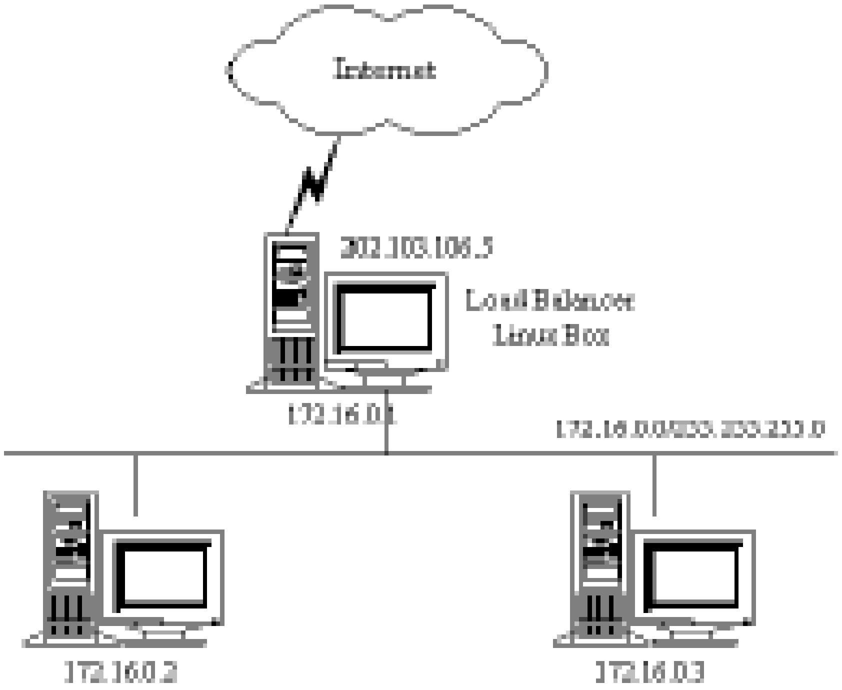
…

Solution 1: Network Address Translation (NAT)

- LB does rewriting/ translation
- Thus, the LB is similar to a typical NAT gateway with an additional scheduling function



Example Virtual Server via NAT



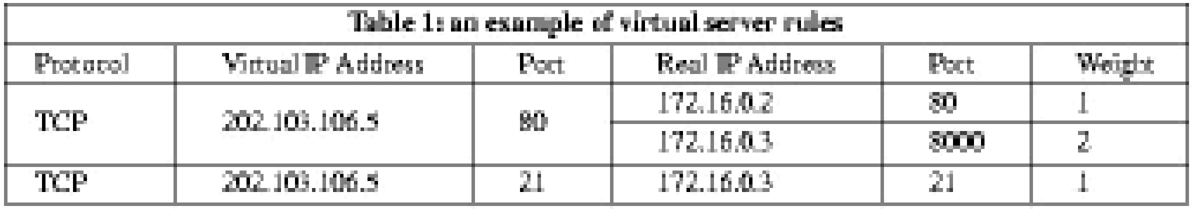
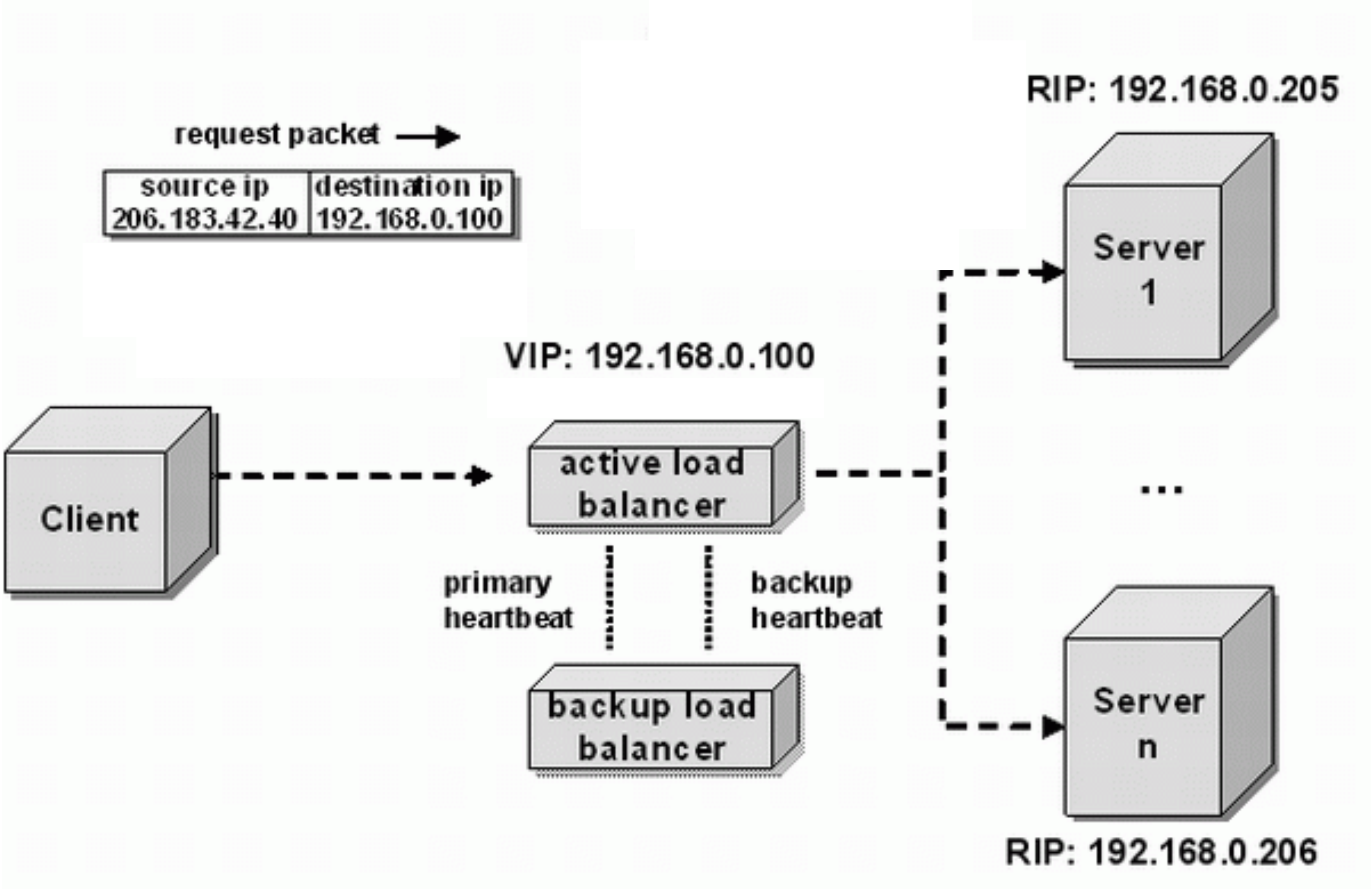


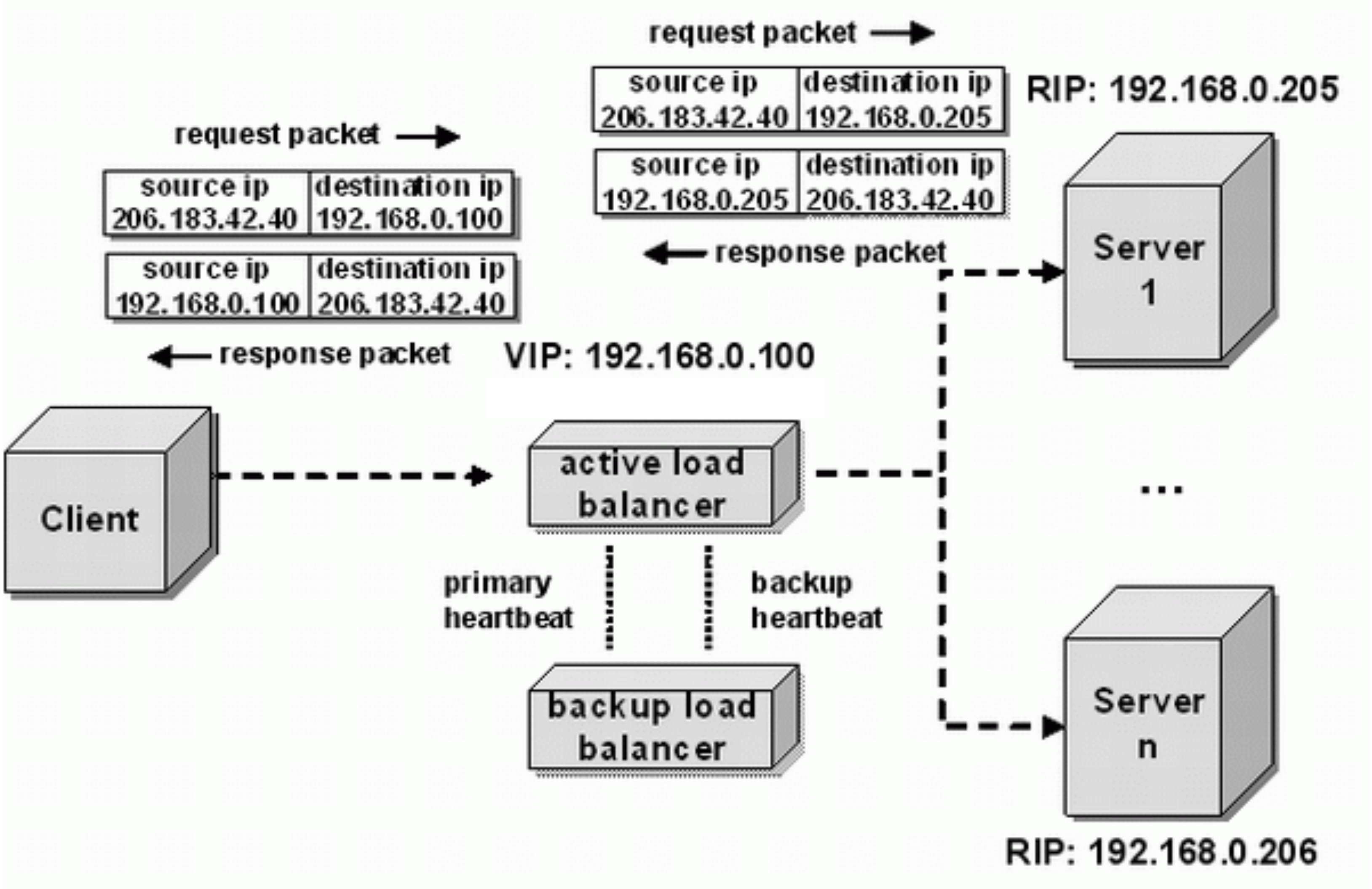
Table 1: an example of virtual server rules

Protocol	Virtual IP Address	Port	Real IP Address	Port	Weight
TCP	202.103.106.5	80	172.16.0.2	80	1
			172.16.0.3	8000	2
TCP	202.103.106.5	21	172.16.0.3	21	1

LB/NAT Flow


request packet
source ip
206.183.42.40
destination ip
192.168.0.100
RIP: 192.168.0.205
Server 1
VIP: 192.168.0.100
Client
active load balancer
primary heartbeat
backup heartbeat
backup load balancer
...
Server n
RIP: 192.168.0.206

LB/NAT Flow



source ip	destination ip
206.183.42.40	192.168.0.100

source ip	destination ip
192.168.0.100	206.183.42.40

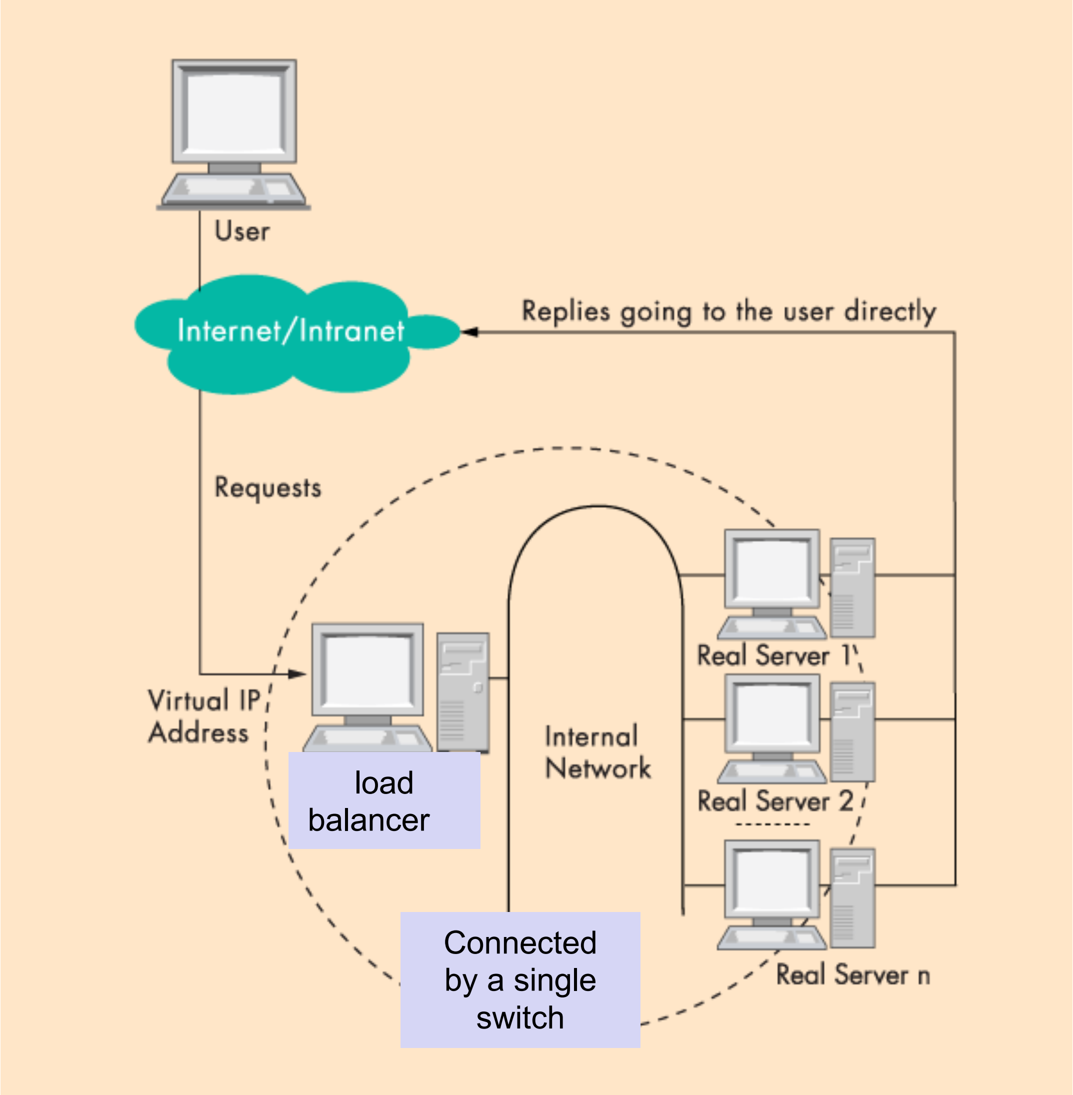
source ip	destination ip
206.183.42.40	192.168.0.205

source ip	destination ip
192.168.0.205	206.183.42.40

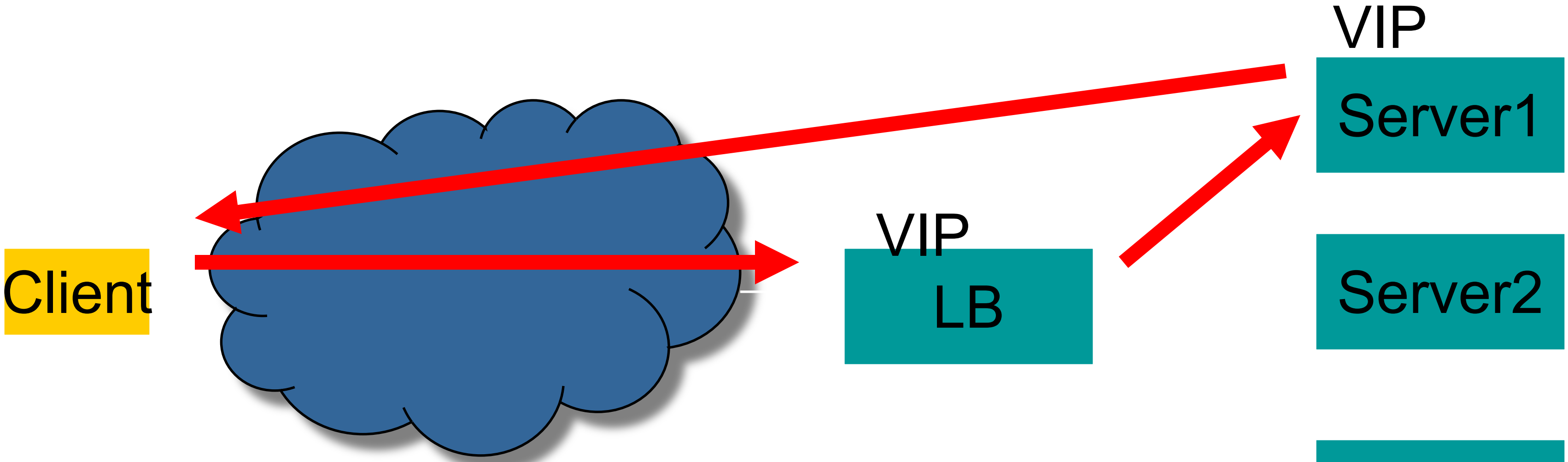
LB/NAT Advantages and Disadvantages

- Advantages:
  - Only one public IP address is needed for the load balancer; real servers can use private IP addresses
  - Real servers need no change and are not aware of load balancing
- Problem
  - The load balancer must be on the critical path and hence may become the bottleneck due to load to rewrite request and response packets
    - Typically, rewriting responses has more load because there are more response packets

Goal: LB w/ Direct Reply



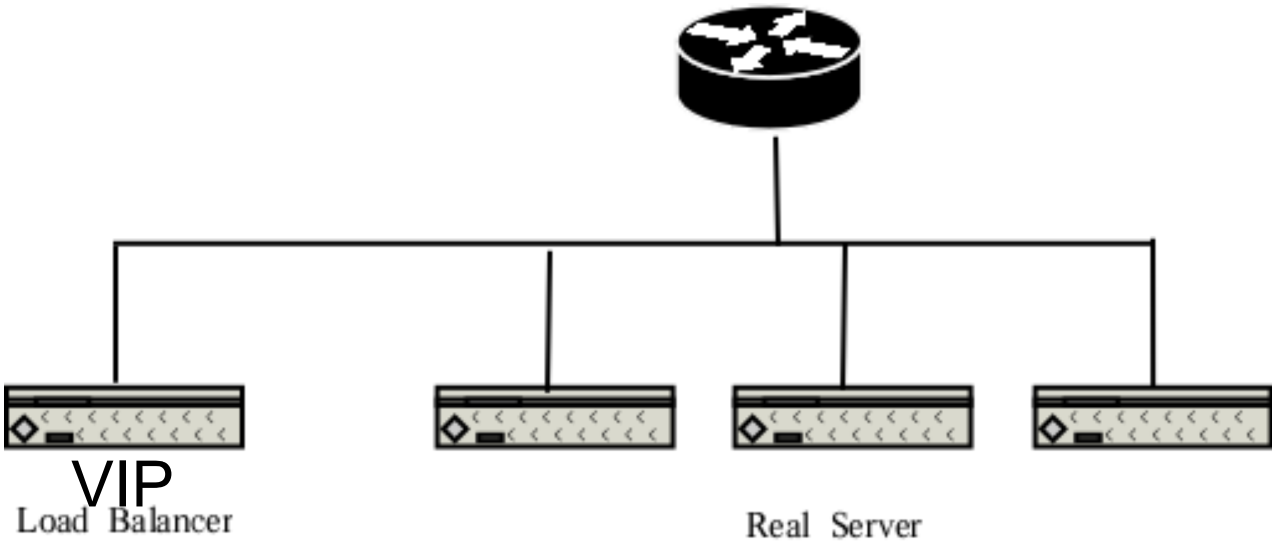
LB with Direct Reply: Implication



Direct reply → Each real server uses VIP as its IP address

→ Address conflict: multiple devices w/ the same IP addr

Why IP Address Matters?



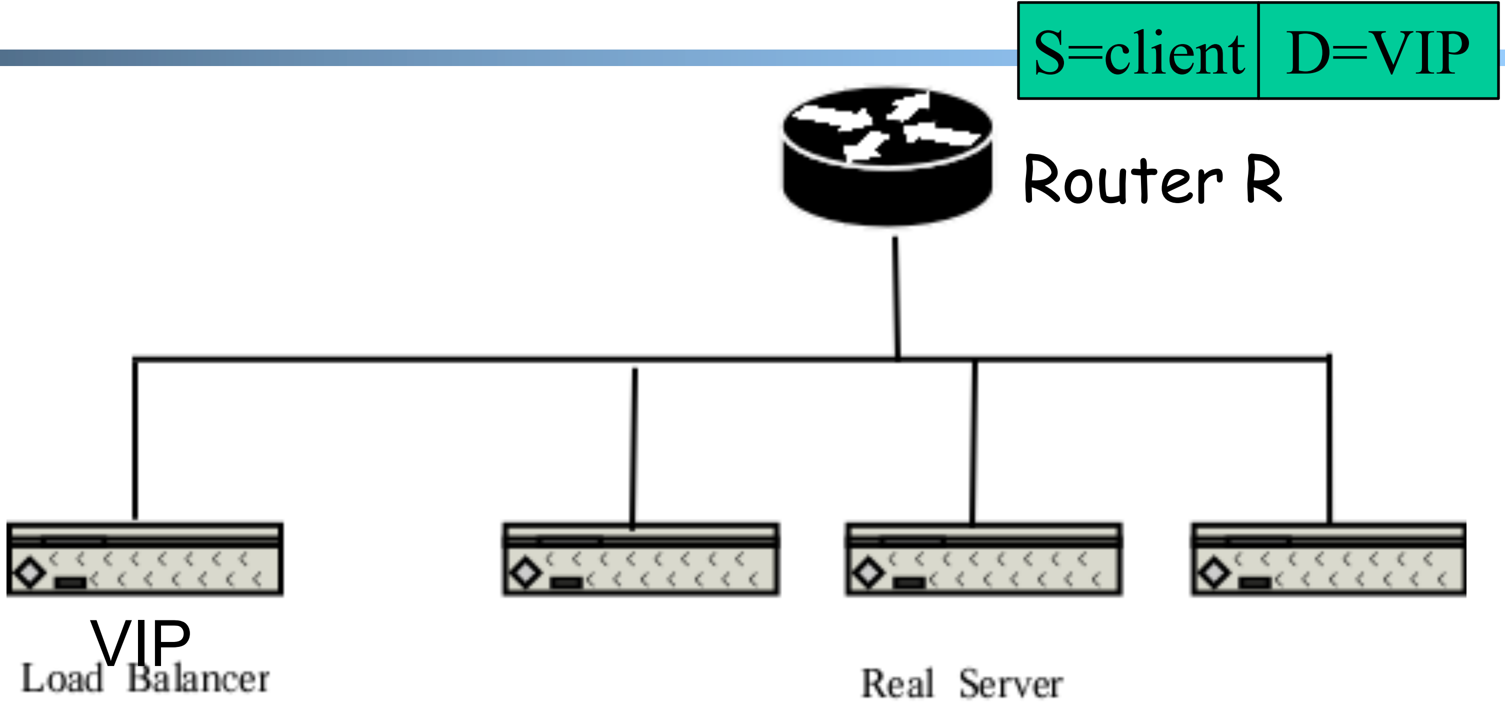
- Each network interface card listens to an assigned MAC address
- A router is configured with the range of IP addresses connected to each interface (NIC)
- To send to a device with a given IP, the router needs to translate IP to MAC (device) address
- The translation is done by the Address Resolution Protocol (ARP)

ARP Protocol

- ❑ ARP is “plug-and-play”:
  - nodes create their ARP tables without intervention from net administrator

- ❑ A broadcast protocol:
  - Router broadcasts query frame, containing queried IP address
    - all machines on LAN receive ARP query
  - Node with queried IP receives ARP frame, replies its MAC address

ARP in Action

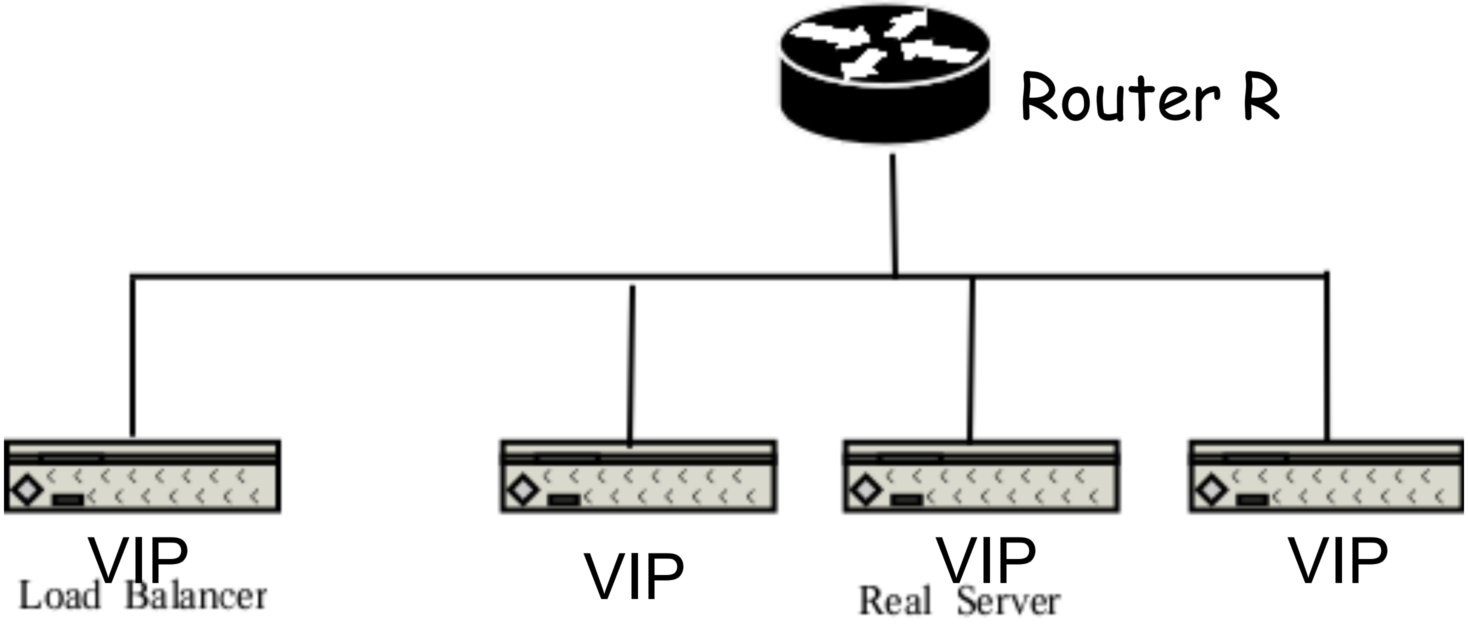


- Router broadcasts ARP broadcast query: who has VIP?

- ARP reply from LB: I have VIP; my MAC is MAC_{LB}

- Data packet from R to LB: destination MAC = MAC_{LB}

LB/DR Problem



ARP and race condition:

- When router R gets a packet with dest. address VIP, it broadcasts an Address Resolution Protocol (ARP) request: who has VIP?
- One of the real servers may reply before load balancer

Solution: configure real servers to not respond to ARP request

LB via Direct Routing

- The virtual IP address is shared by real servers and the load balancer.
- Each real server has a non-ARPing, loopback alias interface configured with the virtual IP address, and the load balancer has an interface configured with the virtual IP address to accept incoming packets.
- The workflow of LB/DR is similar to that of LB/NAT:
  - the load balancer directly routes a packet to the selected server
    - the load balancer simply changes the MAC address of the data frame to that of the server and retransmits it on the LAN (how to know the real server's MAC?)
  - When the server receives the forwarded packet, the server determines that the packet is for the address on its loopback alias interface, processes the request, and finally returns the result directly to the user

LB/DR Advantages and Disadvantages

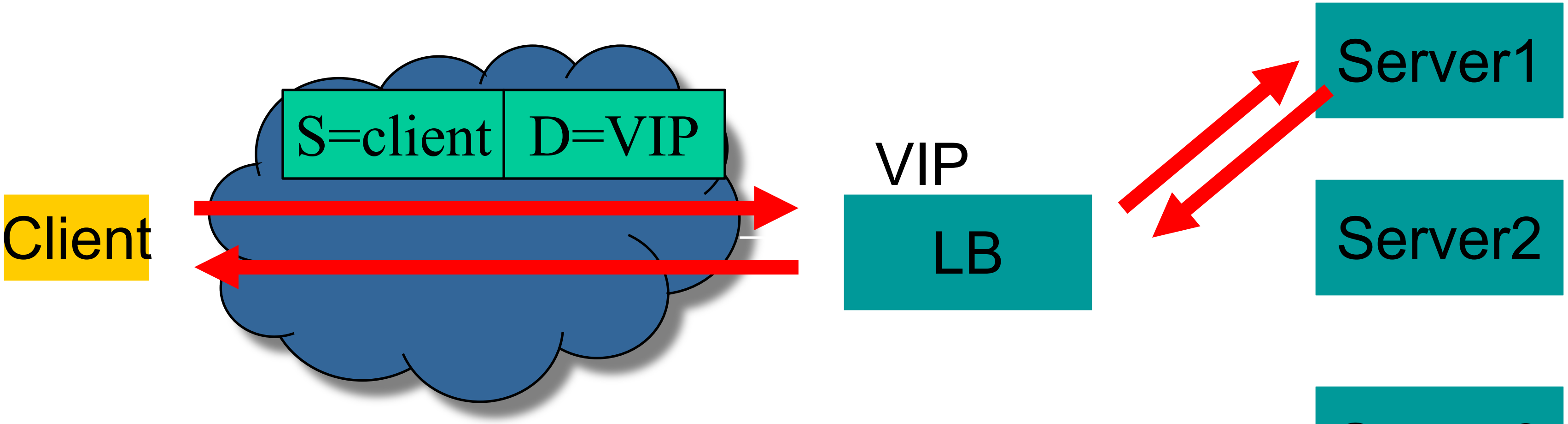
- Advantages:
  - Real servers send response packets to clients directly, avoiding LB as bottleneck

- Disadvantages:
  - Servers must have non-arp alias interface
  - The load balancer and server must have one of their interfaces in the same LAN segment
  - Considered by some as a hack, not a clean architecture

Example Implementation of LB

- An example open source implementation is Linux virtual server (linux-vs.org)
  - Used by
    - www.linux.com
    - sourceforge.net
    - wikipedia.org
  - More details on ARP problem: http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.arp_problem.html
  - Many commercial LB servers from F5, Cisco, …
- More details please read chapter 2 of Load Balancing Servers, Firewalls, and Caches

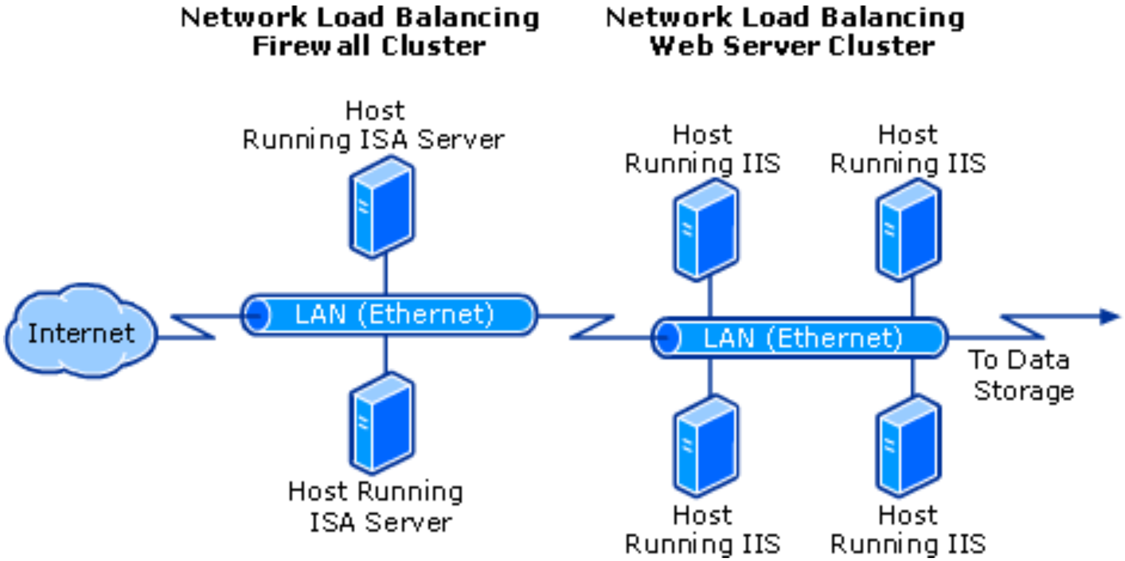
Problem of the Load Balancer Architecture



One major problem is that the LB becomes a single point of failure (SPOF).

Solutions

- Redundant load balancers
  - E.g., two load balancers (a good question to think offline)
- Fully distributed load balancing
  - e.g., Microsoft Network Load Balancing (NLB)



Microsoft NLB

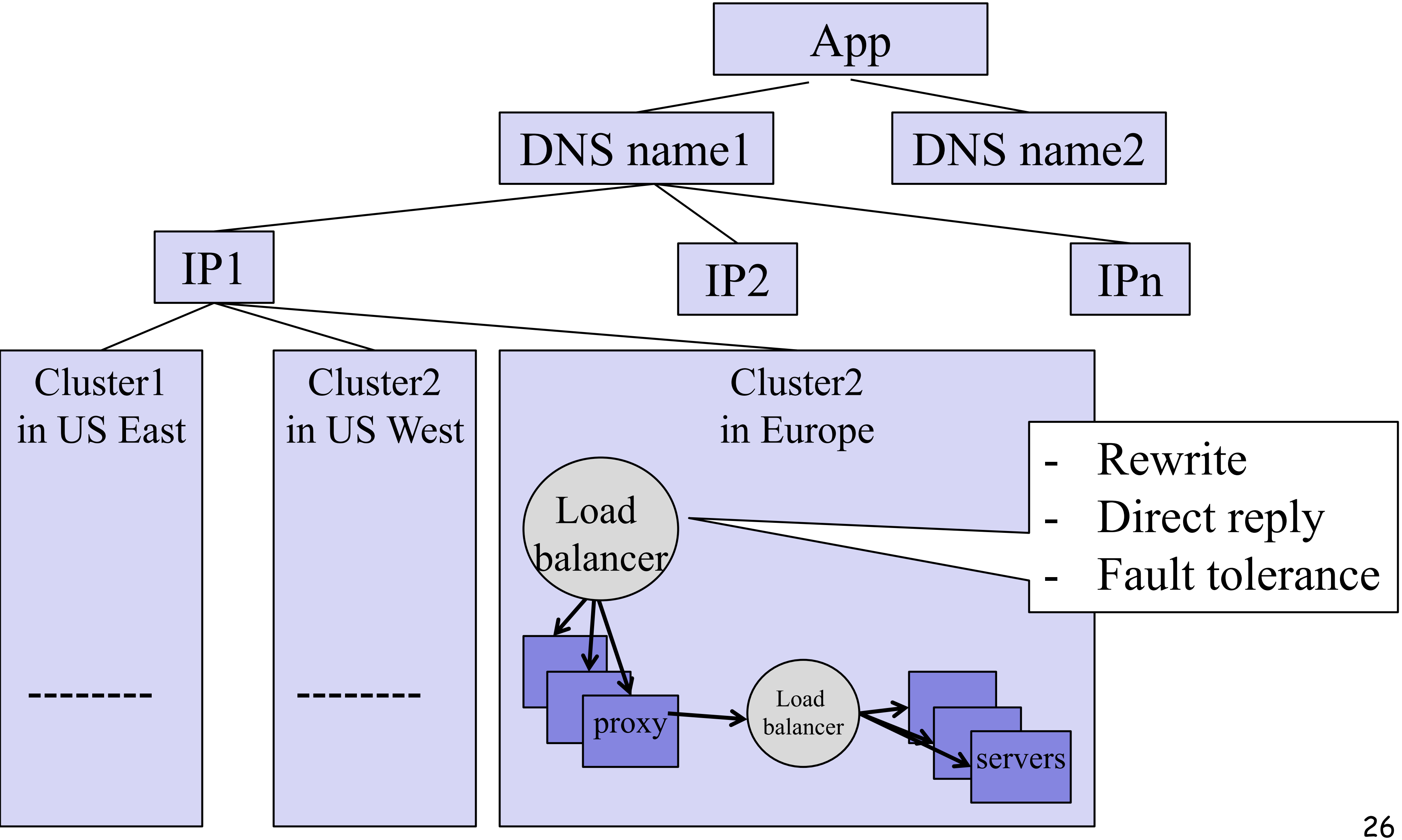
- No dedicated load balancer
- All servers in the cluster receive all packets
- Key issue: one and only one server processes each packet
  - All servers within the cluster simultaneously run a mapping algorithm to determine which server should handle the packet. Those servers not required to service the packet simply discard it.
    - Mapping (ranking) algorithm: computing the "winning" server according to host priorities, multicast or unicast mode, port rules, affinity, load percentage distribution, client IP address, client port number, other internal load information

http://technet.microsoft.com/en-us/library/cc739506%28WS.10%29.aspx

Discussion

- Compare the design of using Load Balancer vs Microsoft NLB

Recap: Direction Mechanisms



- Rewrite
- Direct reply
- Fault tolerance

Outline

- Admin and recap
- Single, high-performance network server
- Multiple servers
  - Overview
  - Basic mechanisms
  - Example: YouTube (offline read)

http://video.google.com/videoplay?docid=-6304964351441328559#

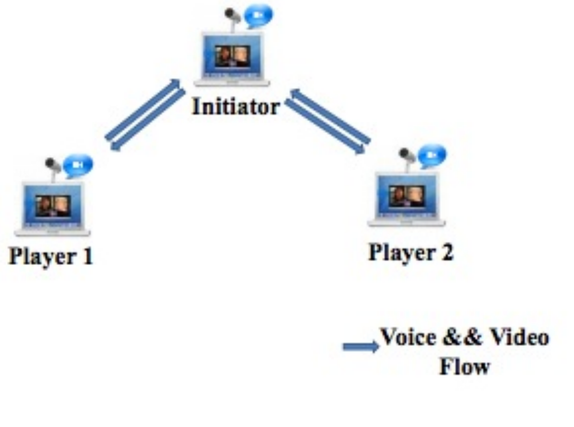
You Tube

- 02/2005: Founded by Chad Hurley, Steve Chen and Jawed Karim, who were all early employees of PayPal.
- 10/2005: First round of funding ($11.5 M)
- 03/2006: 30 M video views/day
- 07/2006: 100 M video views/day
- 11/2006: acquired by Google
- 10/2009: Chad Hurley announced in a blog that YouTube serving well over 1 B video views/day (avg = 11,574 video views /sec)

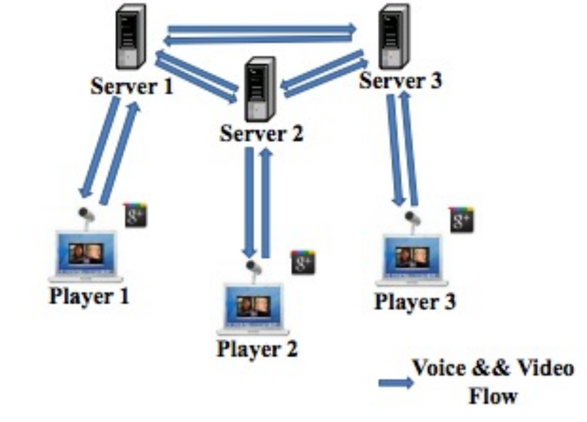
Pre-Google Team Size

- 2 Sysadmins
- 2 Scalability software architects
- 2 feature developers
- 2 network engineers
- 1 DBA
- 0 chefs

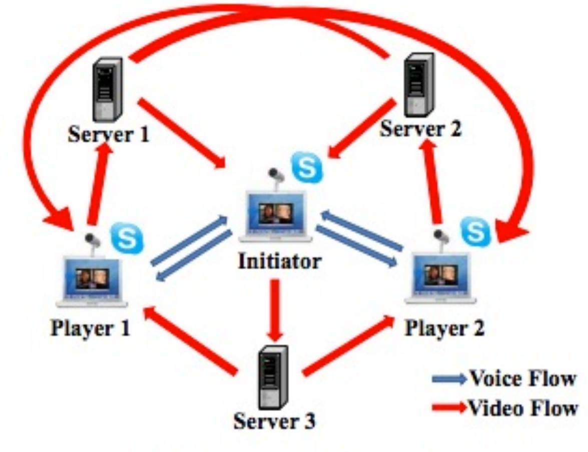
Example Server Systems



(a) iChat



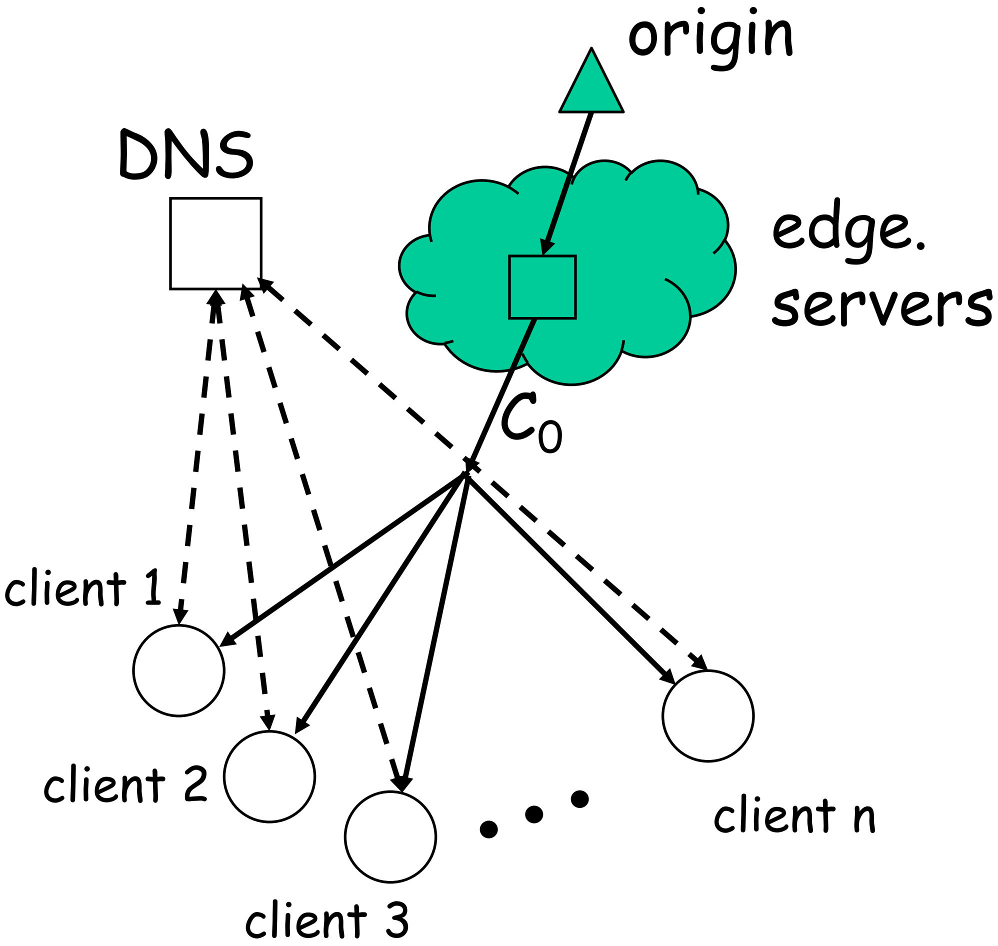
(b) Google Plus Hangout



(c) Skype multi-party

http://eeweb.poly.edu/faculty/yongliu/docs/imc12.pdf

Scalability of Server-Only Approaches

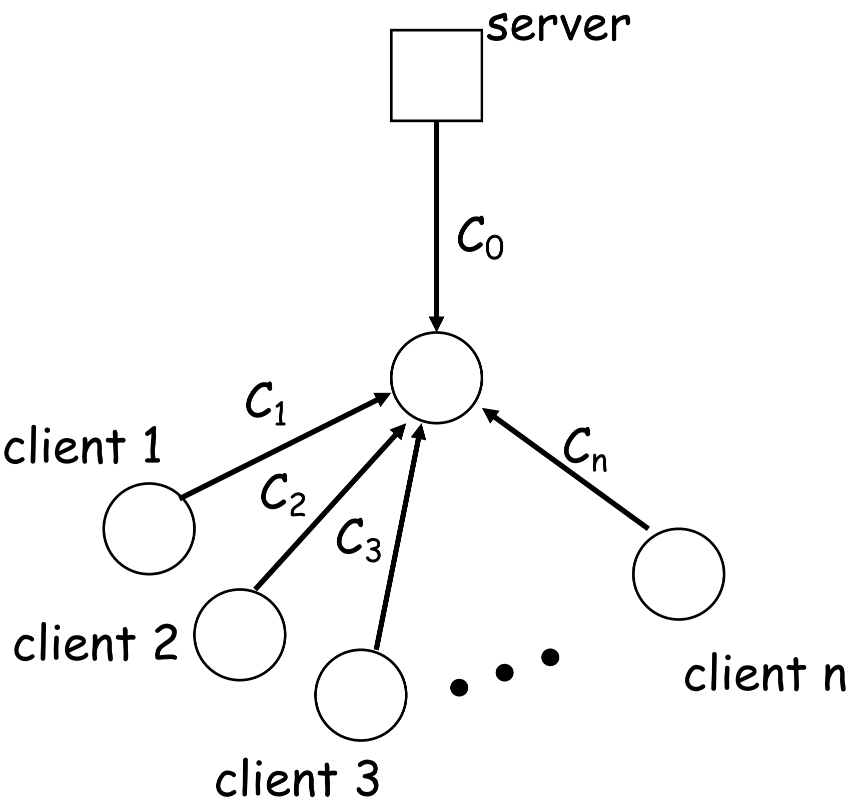


Outline

- Admin and recap
- Multiple servers
- Application overlays
  - potential

An Upper Bound on Scalability

- Idea: use resources from both clients and the server
- Assume
  - need to achieve same rate to all clients
  - only uplinks can be bottlenecks
- What is an upper bound on scalability?

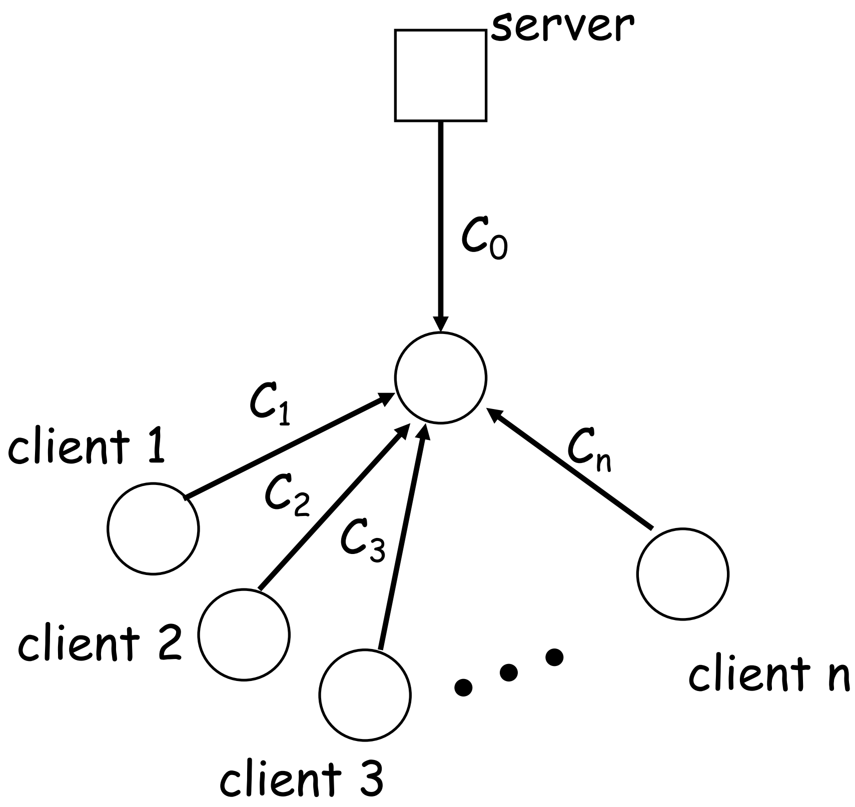


The Scalability Problem

- Maximum throughput

  $R = \min\{C_0, (C_0+\Sigma C_i)/n\}$

- The bound is theoretically approachable

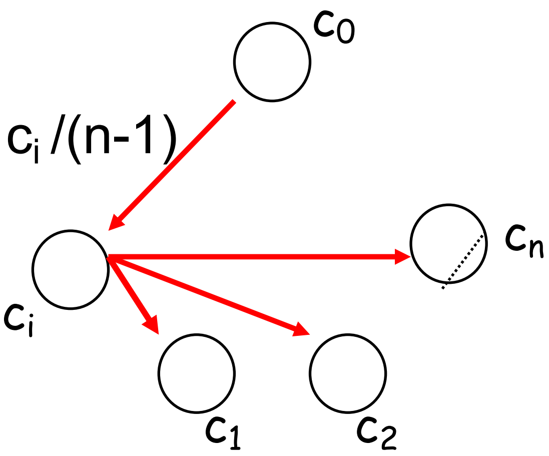


Theoretical Capacity: upload is bottleneck

$R = \min\{C_0, (C_0+\Sigma C_i)/n\}$

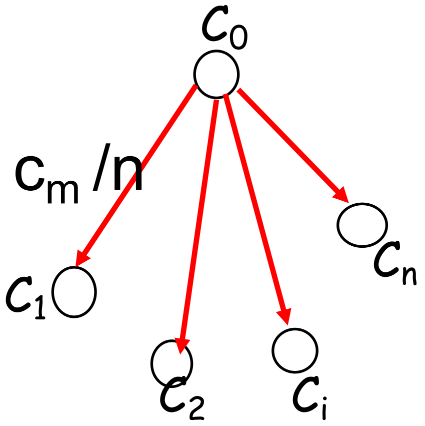
- Assume $c_0 > (C_0+\Sigma C_i)/n$
- Tree i:

  server → client i: $c_i/(n-1)$
  client i → other n-1 clients

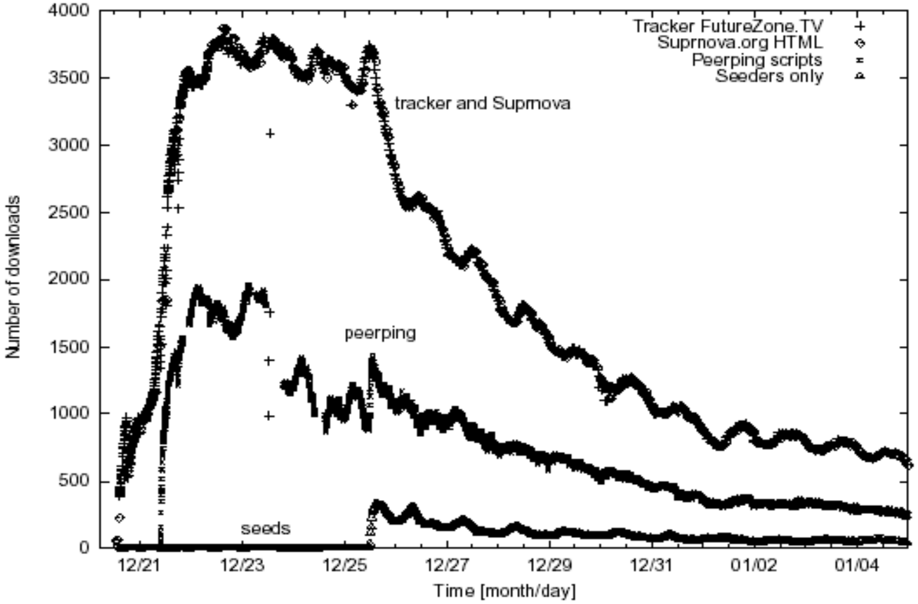


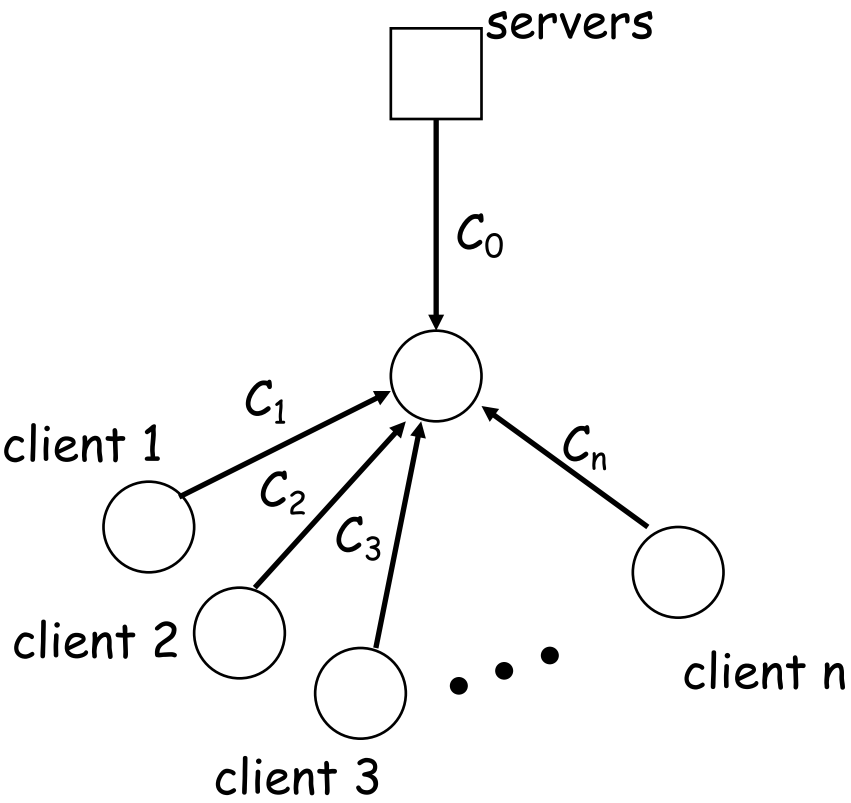
- Tree 0:

  server has remaining
  $c_m = c0 - (c1 + c2 + \ldots cn)/(n-1)$
  send to client i: c_m/n



Why not Building the Trees?

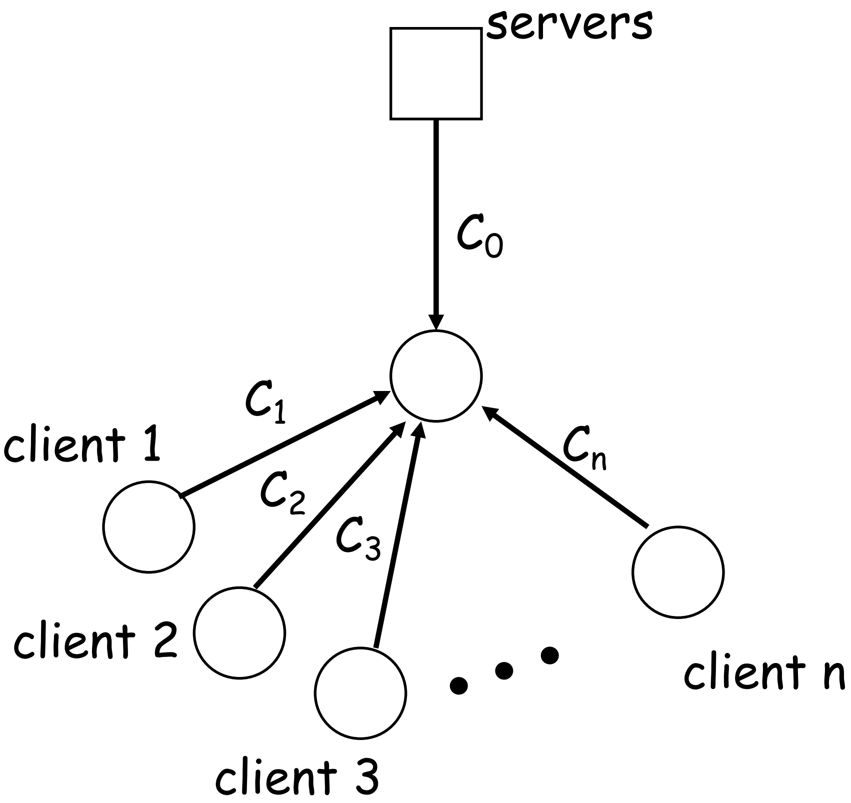




- ❑ Clients come and go (churns): maintaining the trees is too expensive
- ❑ Each client needs N connections

Server+Host (P2P) Content Distribution: Key Design Issues

- Robustness
  - Resistant to churns and failures
- Efficiency
  - A client has content that others need; otherwise, its upload capacity may not be utilized
- Incentive: clients are willing to upload
  - Some real systems nearly 50% of all responses are returned by the top 1% of sharing hosts

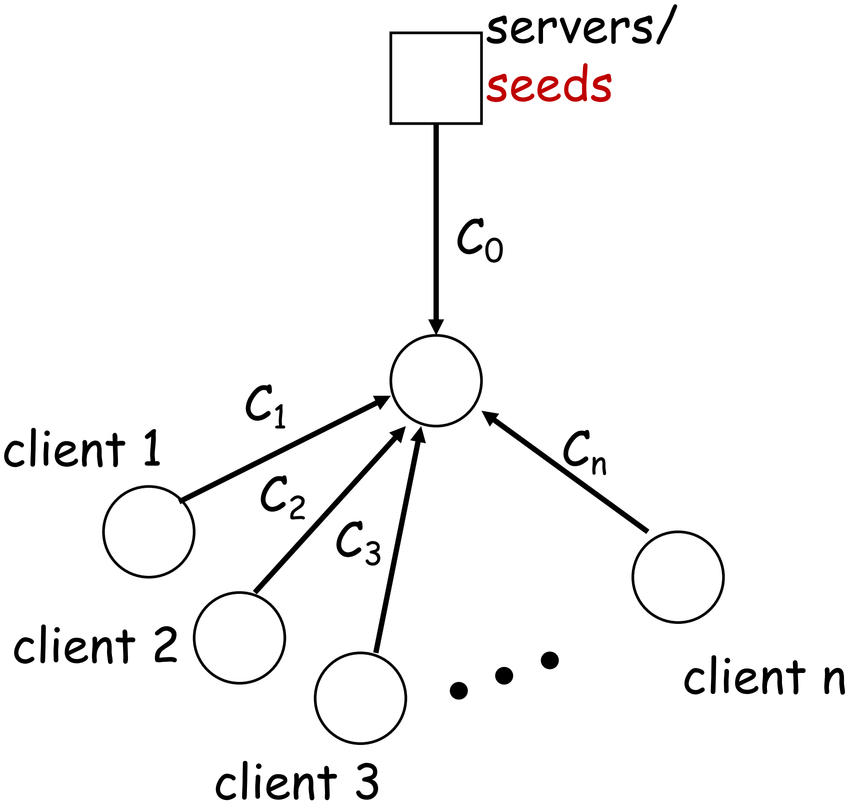


Discussion: How to handle the issues?

- Robustness

- Efficiency

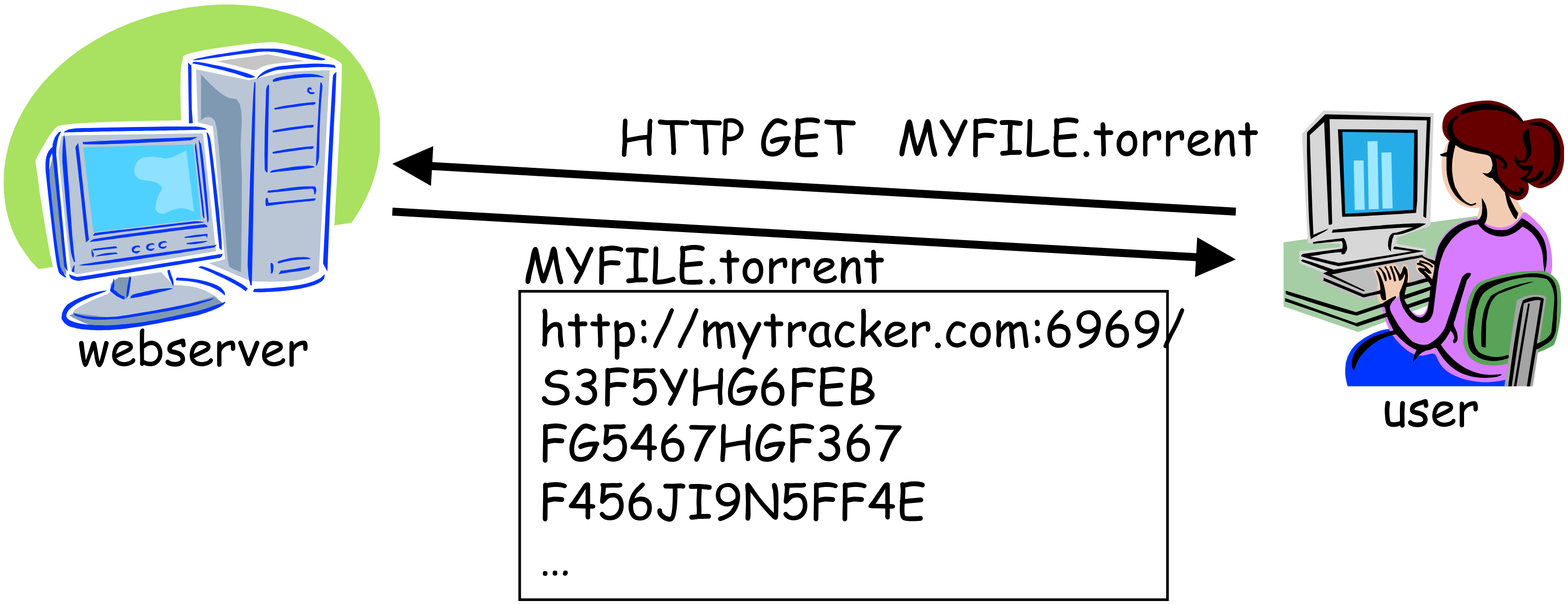
- Incentive



Example: BitTorrent

- A P2P file sharing protocol
- Created by Bram Cohen in 2004
  - Spec at bep_0003: http://www.bittorrent.org/beps/bep_0003.html

BitTorrent: Lookup



HTTP GET MYFILE.torrent

MYFILE.torrent

http://mytracker.com:6969/
S3F5YHG6FEB
FG5467HGF367
F456JI9N5FF4E
…

webserver

user

Metadata (.torrent) File Structure

- Meta info contains information necessary to contact the tracker and describes the files in the torrent
  - URL of tracker
  - file name
  - file length
  - piece length (typically 256KB)
  - SHA-1 hashes of pieces for verification
  - also creation date, comment, creator, …

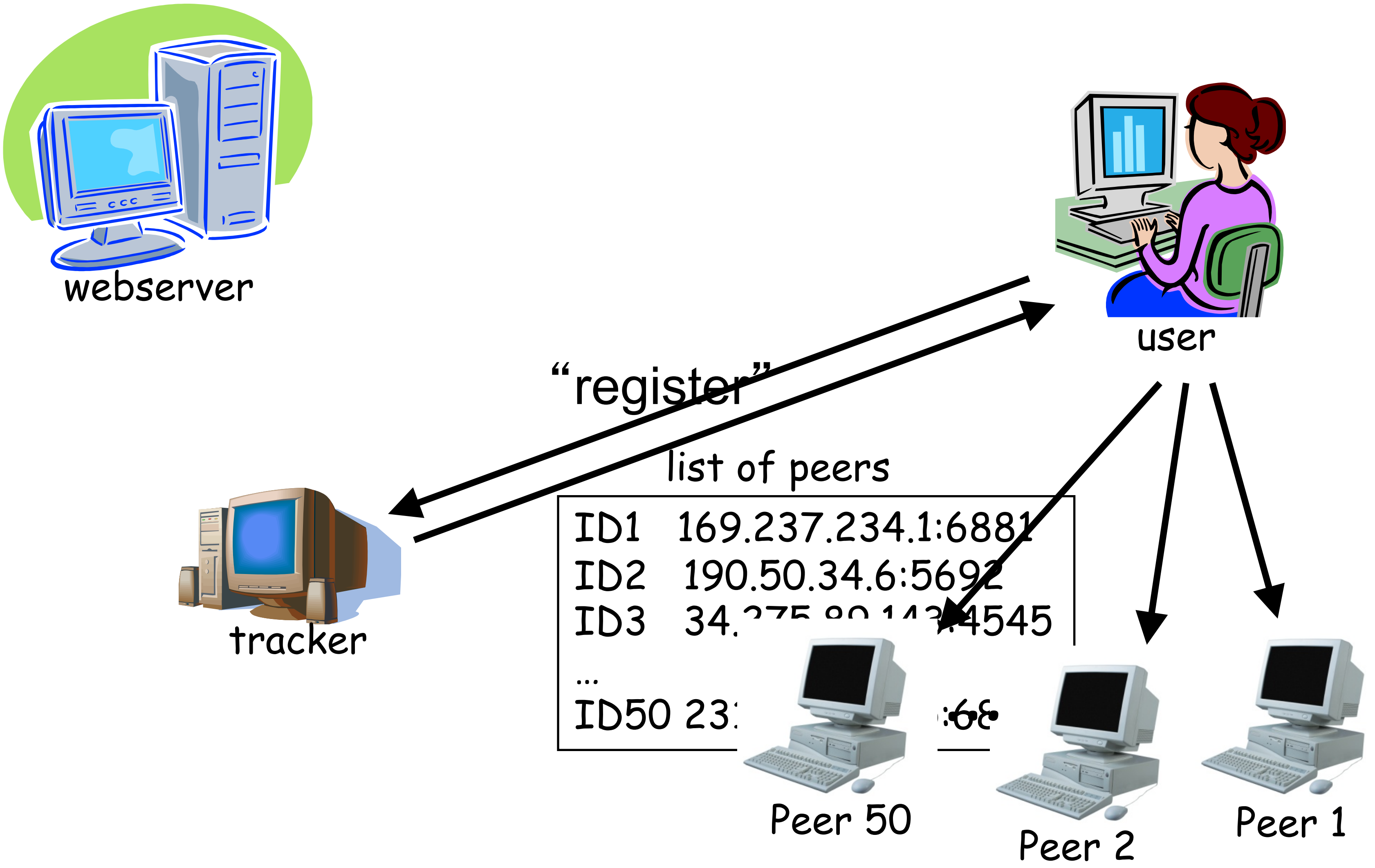
Tracker Protocol

- Communicates with clients via HTTP/HTTPS

- Client GET request
  - info_hash: uniquely identifies the file
  - peer_id: chosen by and uniquely identifies the client
  - client IP and port
  - numwant: how many peers to return (defaults to 50)
  - stats: e.g., bytes uploaded, downloaded

- Tracker GET response
  - interval: how often to contact the tracker
  - list of peers, containing peer id, IP and port
  - stats

Tracker Protocol



webserver

user

"register"

tracker

list of peers

ID1 169.237.234.1:6881
ID2 190.50.34.6:5692
ID3 34.275.89.143:4545
…
ID50 23[illegible]:68

Peer 50

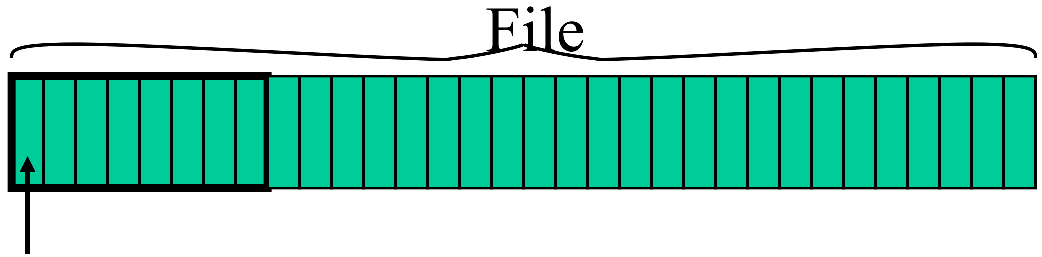
Peer 2

Peer 1

Robustness and efficiency: Piece-based Swarming

- Divide a large file into small blocks and request block-size content from different peers (why?)

Block: unit of download

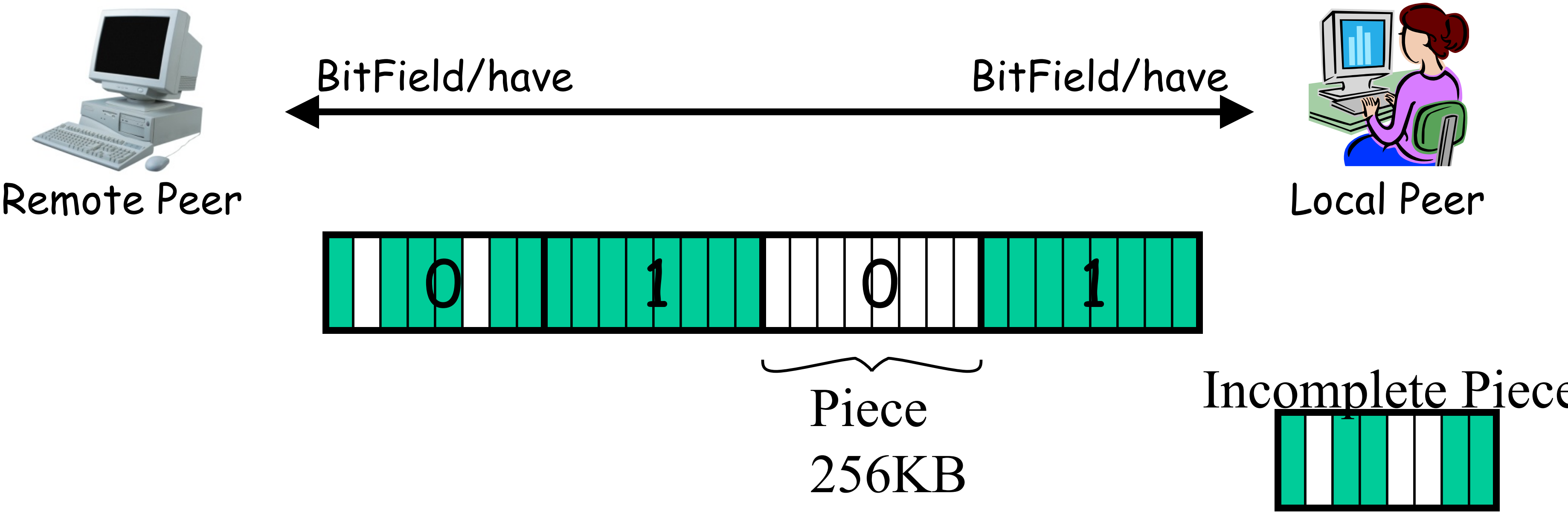


Block: 16KB

- If do not finish downloading a block from one peer within timeout (say due to churns), switch to requesting the block from another peer

Detail: Peer Protocol

(Over TCP)

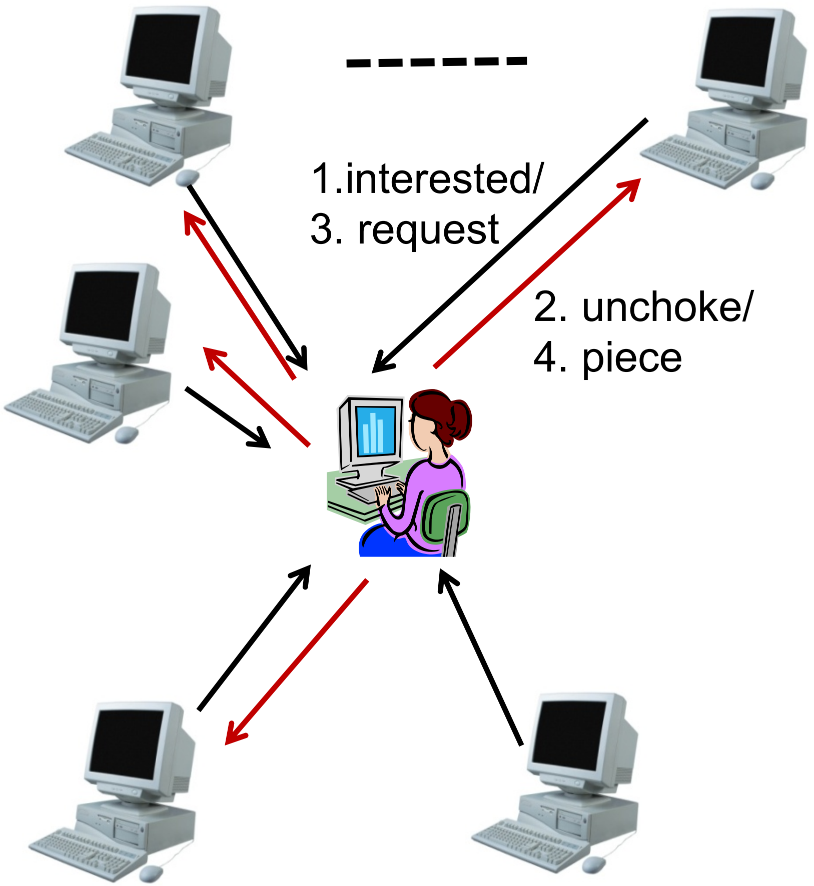


- Peers exchange bitmap representing content availability
  - `bitfield` msg during initial connection
  - `have` msg to notify updates to bitmap
  - to reduce bitmap size, aggregate multiple blocks as a piece

Peer Request

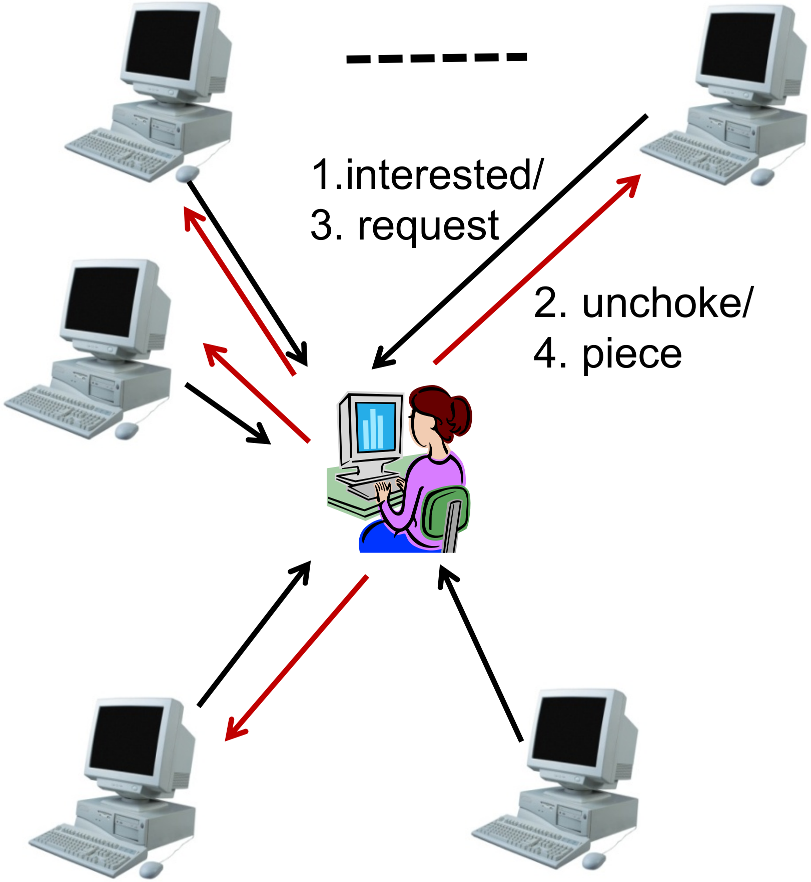
http://www.bittorrent.org/beps/bep_0003.html

- If peer A has a piece that peer B needs, peer B sends `interested` to A
- `unchoke`: indicate that A allows B to request
- `request`: B requests a specific block from A
- `piece`: specific data



Key Design Points

- request:
  - which data blocks to request?
- unchoke:
  - which peers to serve?



Request: Block Availability

- Request (local) rarest first
  - achieves the fastest replication of rare pieces
  - obtain something of value

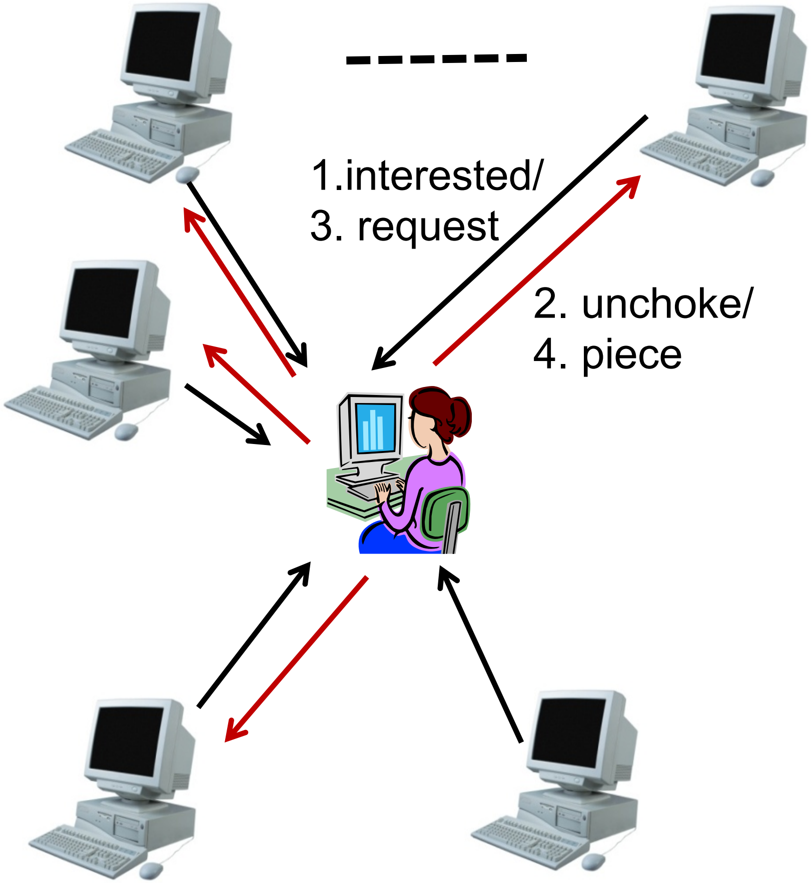
Block Availability: Revisions

- When downloading starts (first 4 pieces): choose at random and request them from the peers
  - get pieces as quickly as possible
  - obtain something to offer to others

- Endgame mode
  - defense against the “last-block problem”: cannot finish because missing a few last pieces
  - send requests for missing pieces to all peers in our peer list
  - send `cancel` messages upon receipt of a piece

BitTorrent: Unchoke

- Periodically (typically every 10 seconds) calculate data-receiving rates from all peers
- Upload to (*unchoke*) the fastest
  - constant number (4) of unchoking slots
  - partition upload bw equally among unchoked



commonly referred to as "tit-for-tat" strategy

Optimistic Unchoking

- Periodically select a peer at random and upload to it
  - typically every 3 unchoking rounds (30 seconds)

- Multi-purpose mechanism
  - allow bootstrapping of new clients
  - continuously look for the fastest peers (exploitation vs exploration)

BitTorrent Fluid Analysis

- Normalize file size to 1
- x(t): number of downloaders (also known as leechers) who do not have all pieces at time t.
- y(t): number of seeds in the system at time t.
- λ: the arrival rate of new requests.
- μ: the uploading bandwidth of a given peer.
- c: the downloading bandwidth of a given peer, assume $c \geq \mu$.
- θ: the rate at which downloaders abort download.
- γ: the rate at which seeds leave the system.
- η: indicates the effectiveness of downloader sharing, η takes values in [0, 1].

System Evolution

$$\begin{aligned}
\frac{\mathrm{d}x}{\mathrm{d}t} &= \lambda - \theta x(t) - \min\{cx(t), \mu(\eta x(t) + y(t))\}, \\
\frac{\mathrm{d}y}{\mathrm{d}t} &= \min\{cx(t), \mu(\eta x(t) + y(t))\} - \gamma y(t),
\end{aligned}$$

Solving steady state: $\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{\mathrm{d}y(t)}{\mathrm{d}t} = 0$

Define $\frac{1}{\beta} = \max\{\frac{1}{c}, \frac{1}{\eta}(\frac{1}{\mu} - \frac{1}{\gamma})\}$

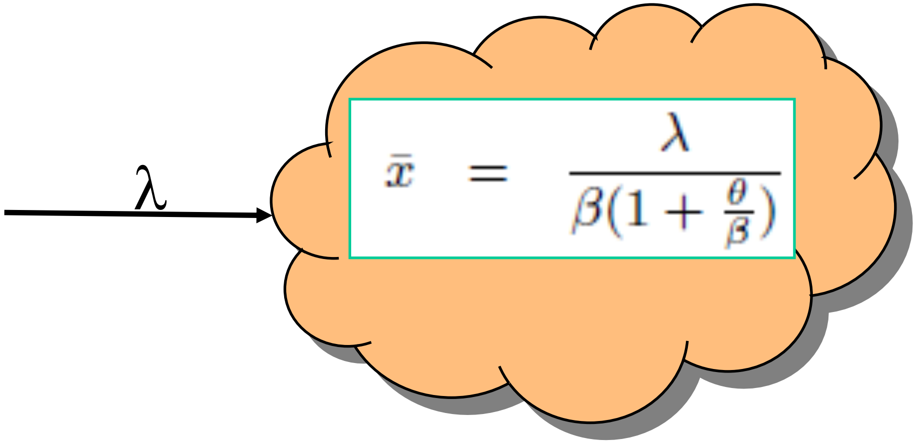
$$\boxed{\begin{aligned}
\bar{x} &= \frac{\lambda}{\beta(1 + \frac{\theta}{\beta})} \\
\bar{y} &= \frac{\lambda}{\gamma(1 + \frac{\theta}{\beta})}.
\end{aligned}}$$

"Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks", SIGCOMM'04
https://conferences.sigcomm.org/sigcomm/2004/papers/p444-qiu1.pdf

System State

$$
\begin{aligned}
\bar{x} &= \frac{\lambda}{\beta(1+\frac{\theta}{\beta})} \\
\bar{y} &= \frac{\lambda}{\gamma(1+\frac{\theta}{\beta})}.
\end{aligned}
$$

Q: How long does each downloader stay as a downloader?



$$\bar{x} = \frac{\lambda}{\beta(1+\frac{\theta}{\beta})}$$

$$T = \frac{1}{\theta + \beta}$$

$$\frac{1}{\beta} = \max\{\frac{1}{c}, \frac{1}{\eta}(\frac{1}{\mu} - \frac{1}{\gamma})\}$$

Key take-away: not scaling inverse with system size (x)

- New requests comes, new bandwidth also comes