Network Applications: Multi-Server Request Routing; Distributed Content Distribution

Y. Richard Yang

http://zoo.cs.yale.edu/classes/cs433/

10/23/2018

Outline

- Admin and recap
- Multiple servers

Admin

- Assignment Three office hours this week
  - Wednesday: 1:30-2:30pm
  - Instructor out of town Thursday and Friday: office hours Saturday/Sunday, Defer to Sunday 5 pm?
- Exam 1
  - Examples linked on the Schedule page

Recap: Designing Load-Balancing Multiple Servers

- Requirements/goals
  - naming abstraction, server load balancing, failure detection, access control filtering, priorities/QoS, request locality, transparent caching
- Components
  - Service/resource discovery (static, zookeeper, etcs, consul)
  - Health/state monitoring of servers/connecting networks
  - Load balancing mechanisms/algorithm (also called request routing)
    - DNS request routing (indirection, hierarchy)
    - Network request routing
      - NAT
      - Direct Server Return (DSR)
    - Request routing director reliability
      - Fully distributed servers
      - VRRP (active/passive, priority selection multiple routers)

Outline

- Admin and recap
- Request routing to multiple servers
  - overview
  - DNS request routing
  - Network request routing
    - Overview of structure and issue
    - Routing direction
      - NAT
      - Direct reply (also called Direct Server Return, DSR)
    - Director reliability
      - Fully distributed
      - Active/passive
      - Failure handling

Basic Setting: Normal

hash(p) = 7

servers[hash(p) % 6]

Basic Setting: Server Failure

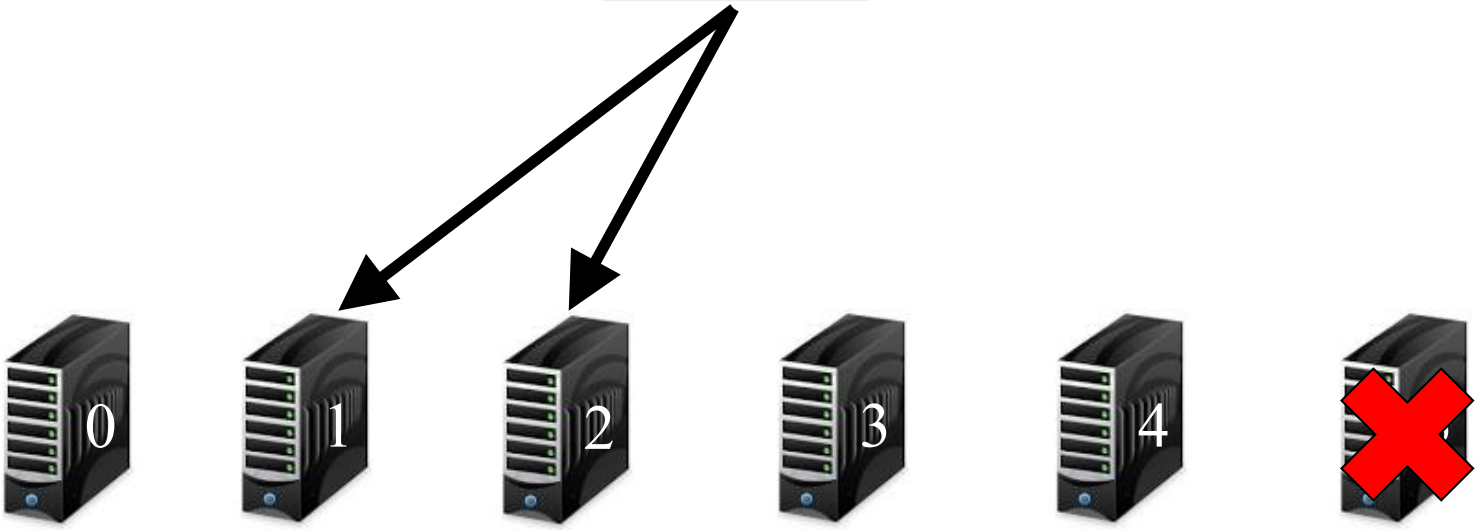
hash(p) = 7

servers[hash(p) % 6]

Basic Setting: Server Failure Fix

hash(p) = 7

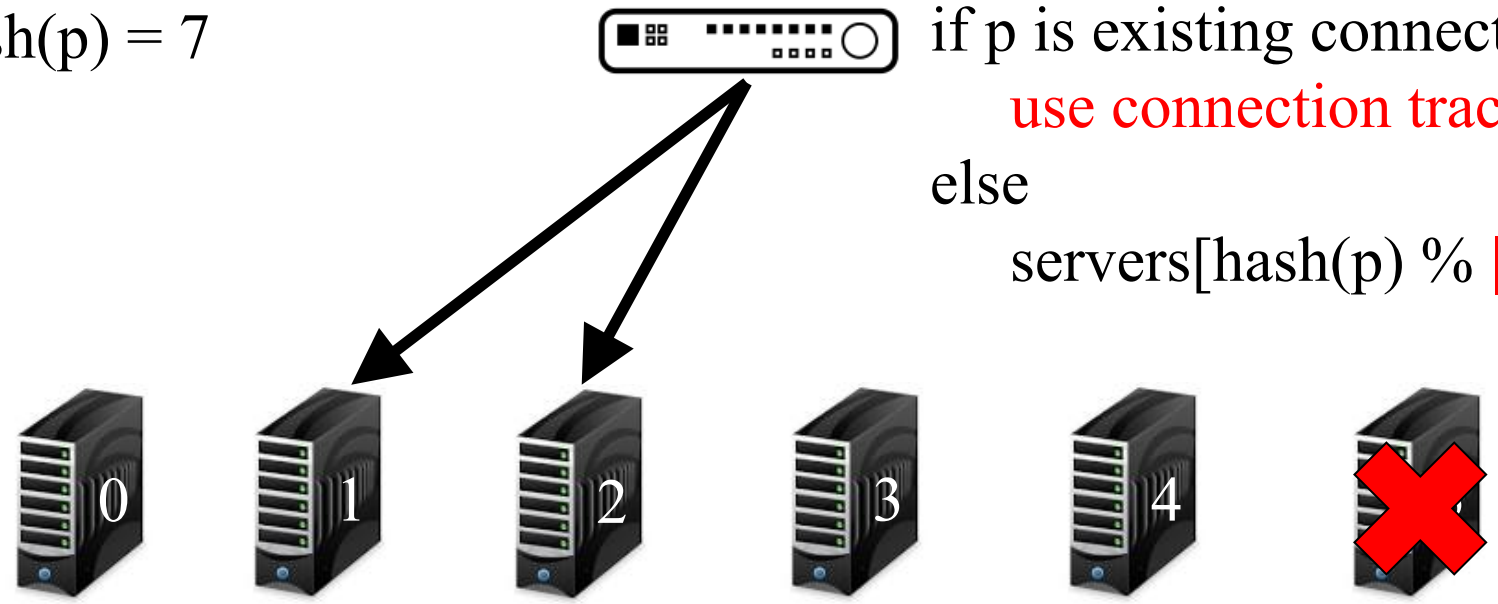
servers[hash(p) % |S|]



Problem of fix: Existing connection using S1 is now directed to S2.

Basic Setting: Server Failure Fix Fix

hash(p) = 7



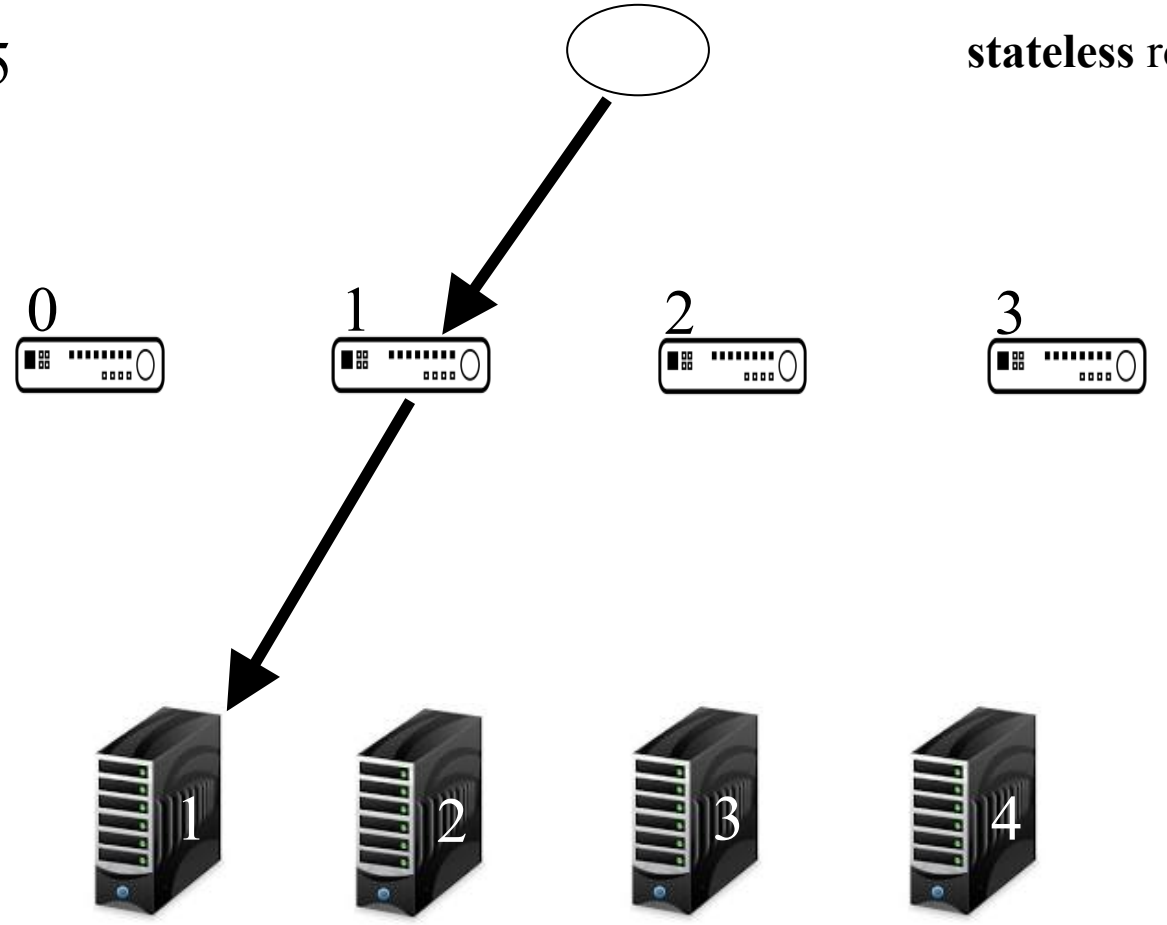
if p is existing connection
    use connection tracking
else
    servers[hash(p) % |S|]

Multiple Request Routers (Large Internet Content Providers)

hash0(p) = 5

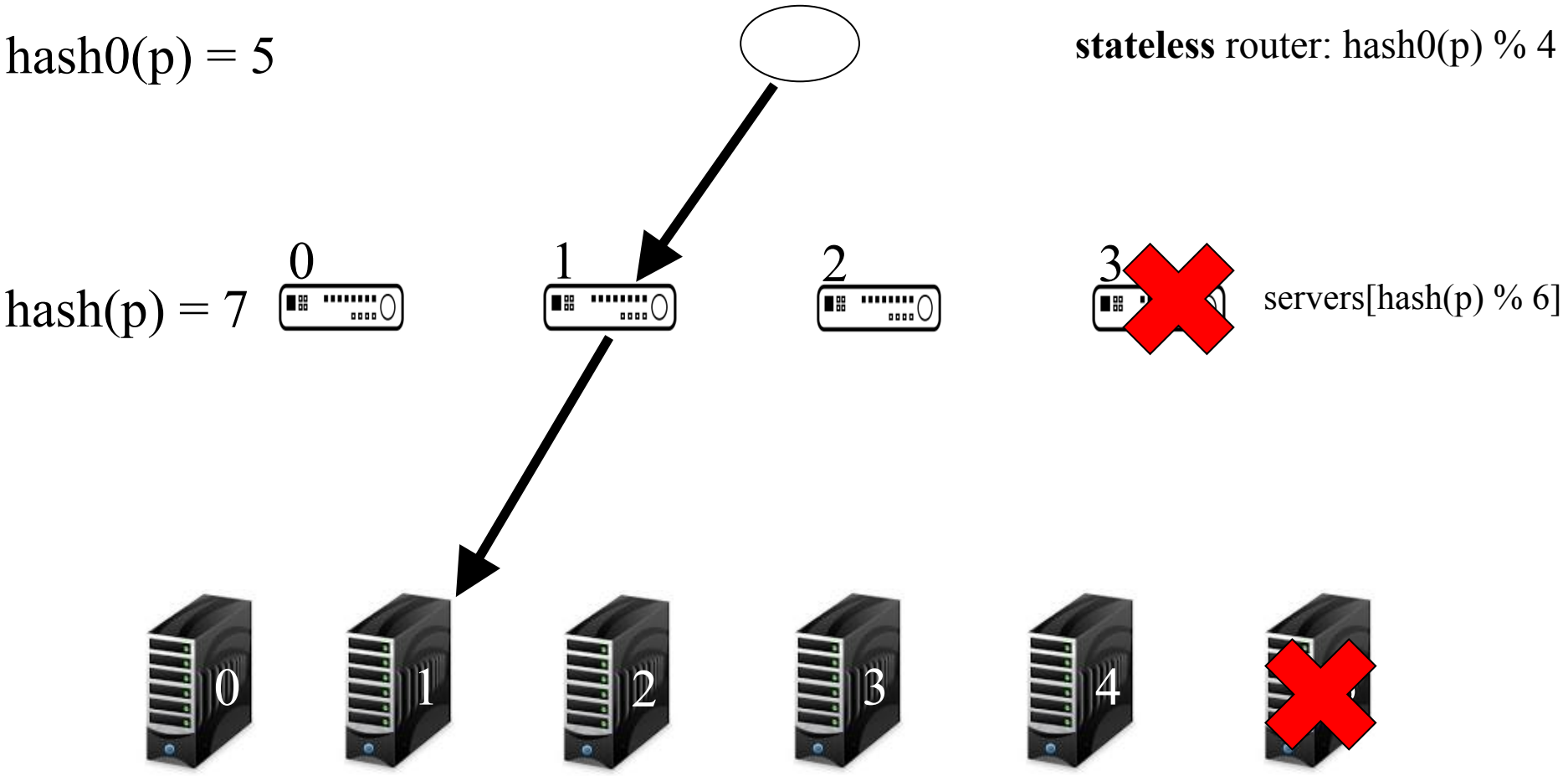
stateless router: hash0(p) % 4

hash(p) = 7



servers[hash(p) % 6]

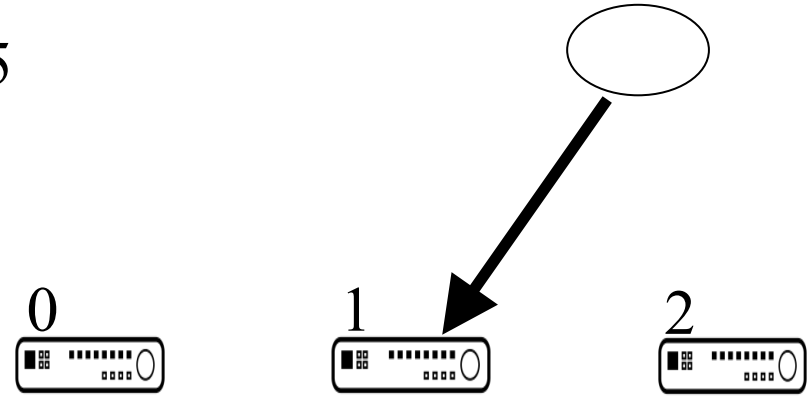
Multiple Request Routers: Failures



Multiple Request Routers: Fix Server Failure

hash0(p) = 5

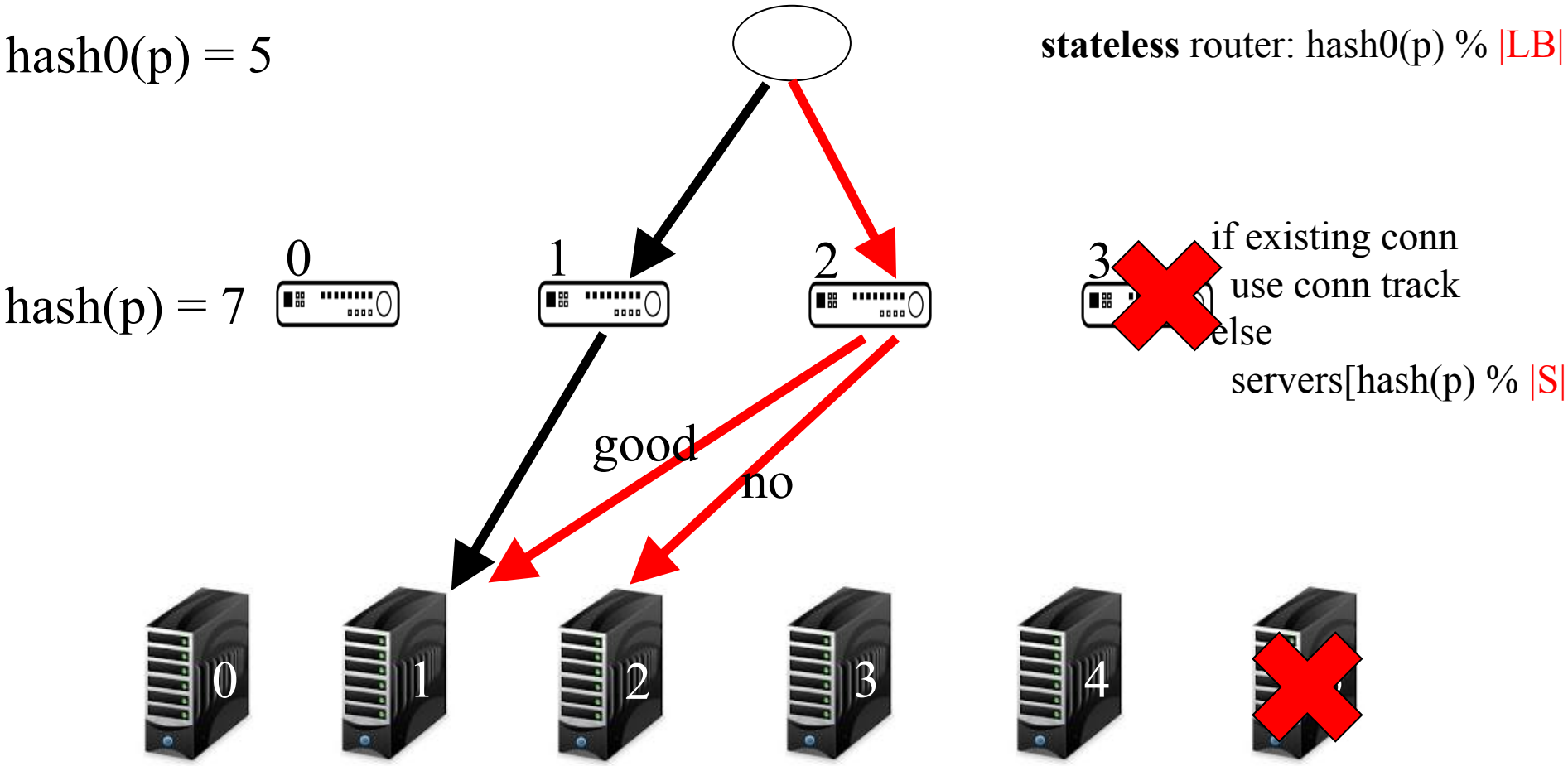
hash(p) = 7



stateless router: hash0(p) % |LB|

if existing conn
  use conn track
else
  servers[hash(p) % |S|]

Multiple Request Routers: Problem

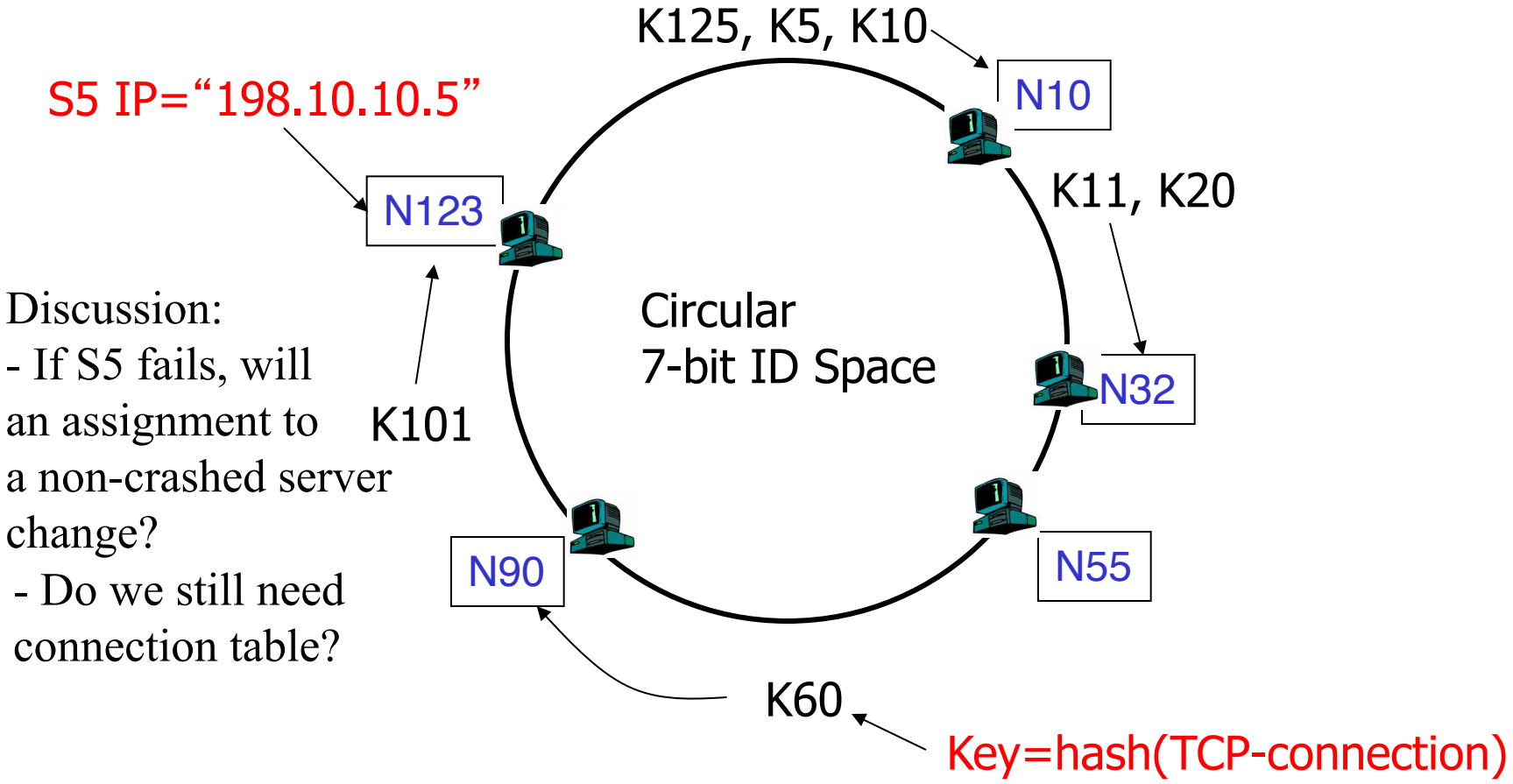


Cause of problem: In simple hashing, failure of one server (S5) causes the connection assigned to another still-up server (S1) changed.

Ring Consistent Hashing

- Space is a ring
- Consistent hashing: m bit identifier space for both keys and nodes (servers)
  - key identifier = SHA-1(key), where SHA-1() is a popular hash function, e.g.,
    - Key=<TCP connection tuples> → ID=60
  - node identifier = SHA-1(IP address)
    - IP="198.10.10.5" → ID=123

Request Routing using Ring



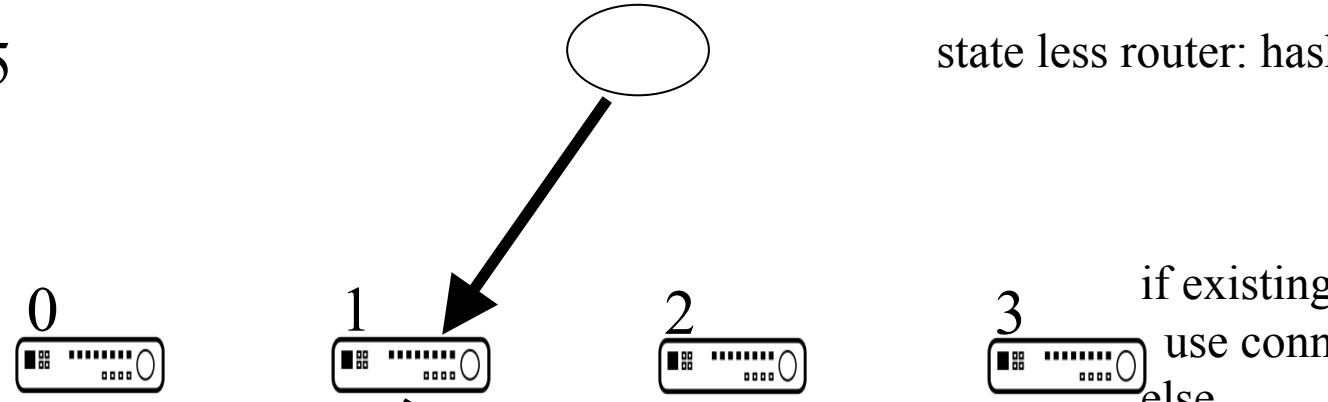
- A key is assigned at its successor: node with next higher or equal ID

Multiple Request Routers: Consistent Hash

hash0(p) = 5

state less router: hash0(p) % |LB|

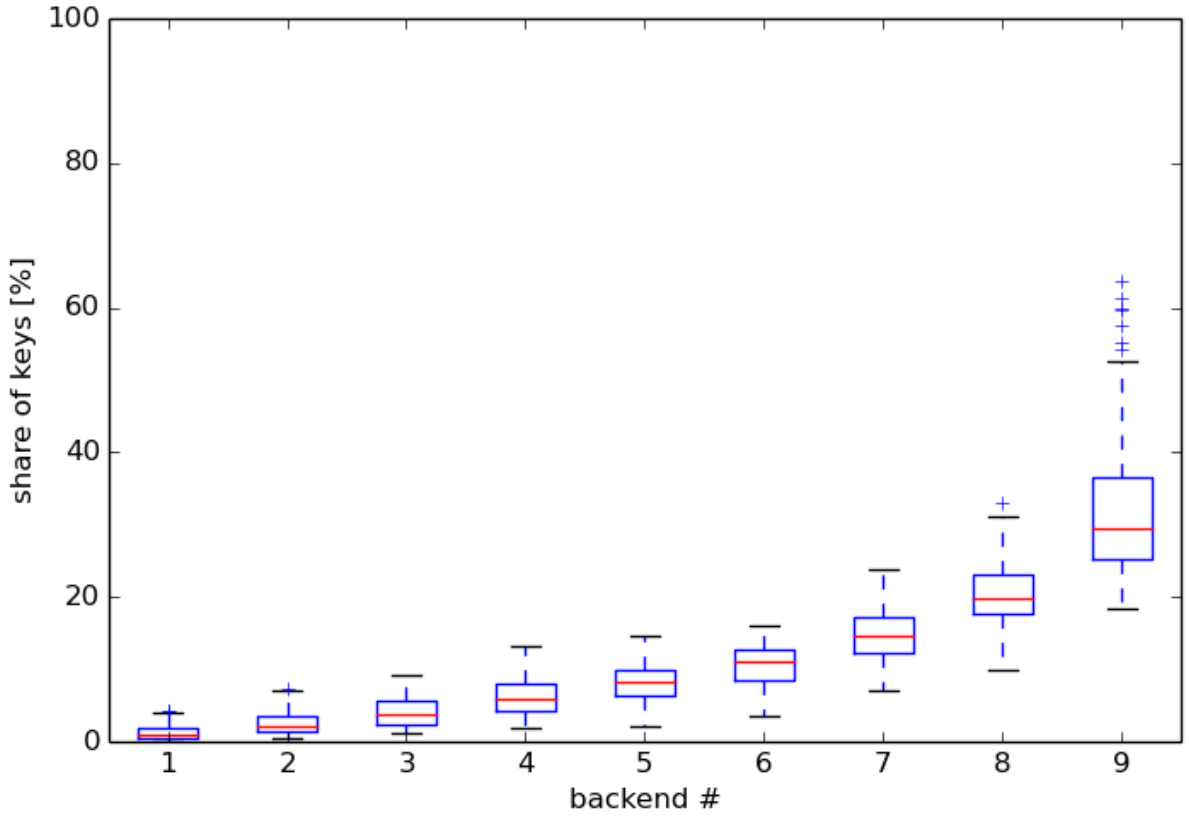
hash(p) = 7



if existing conn
use conn track
else
~~servers[hash(p) % |LB|]~~
consistent_hash(p)

Offline exercise: Failure scenarios of Ring Hash Request Routing.

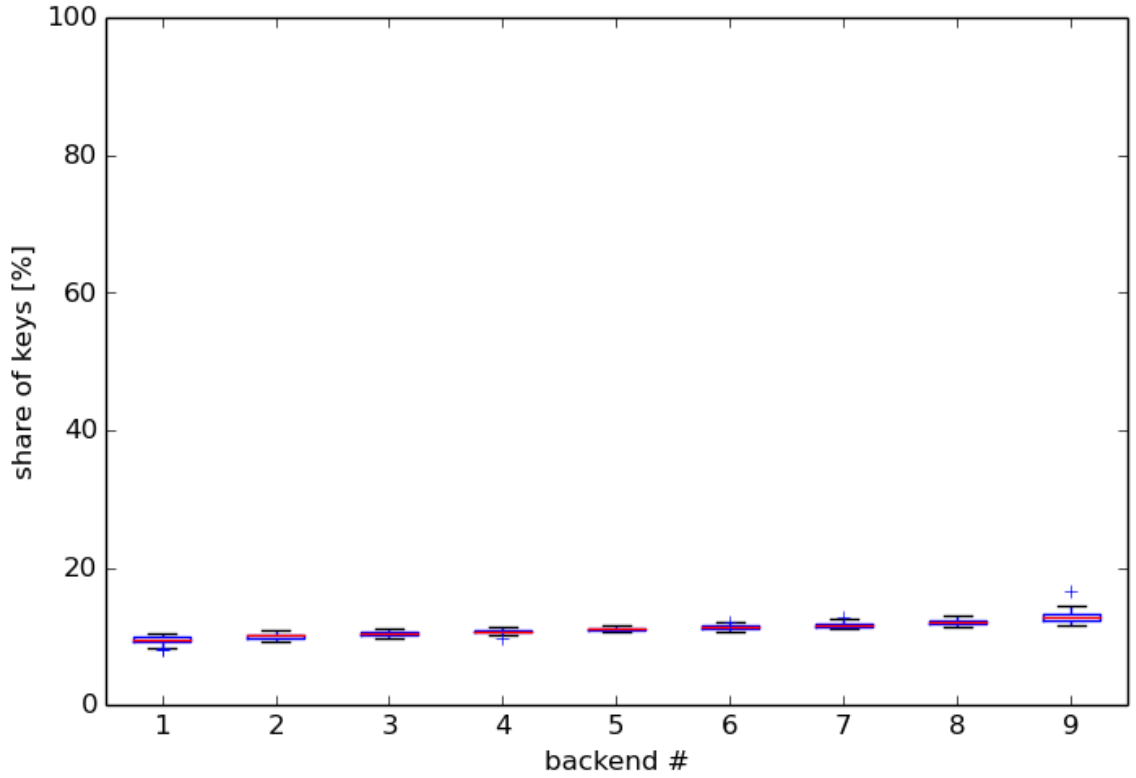
Load Balance of Ring Hash



1 point per server on the ring

https://blog.memcachier.com/2017/09/01/maglev-our-new-consistent-hashing-scheme/

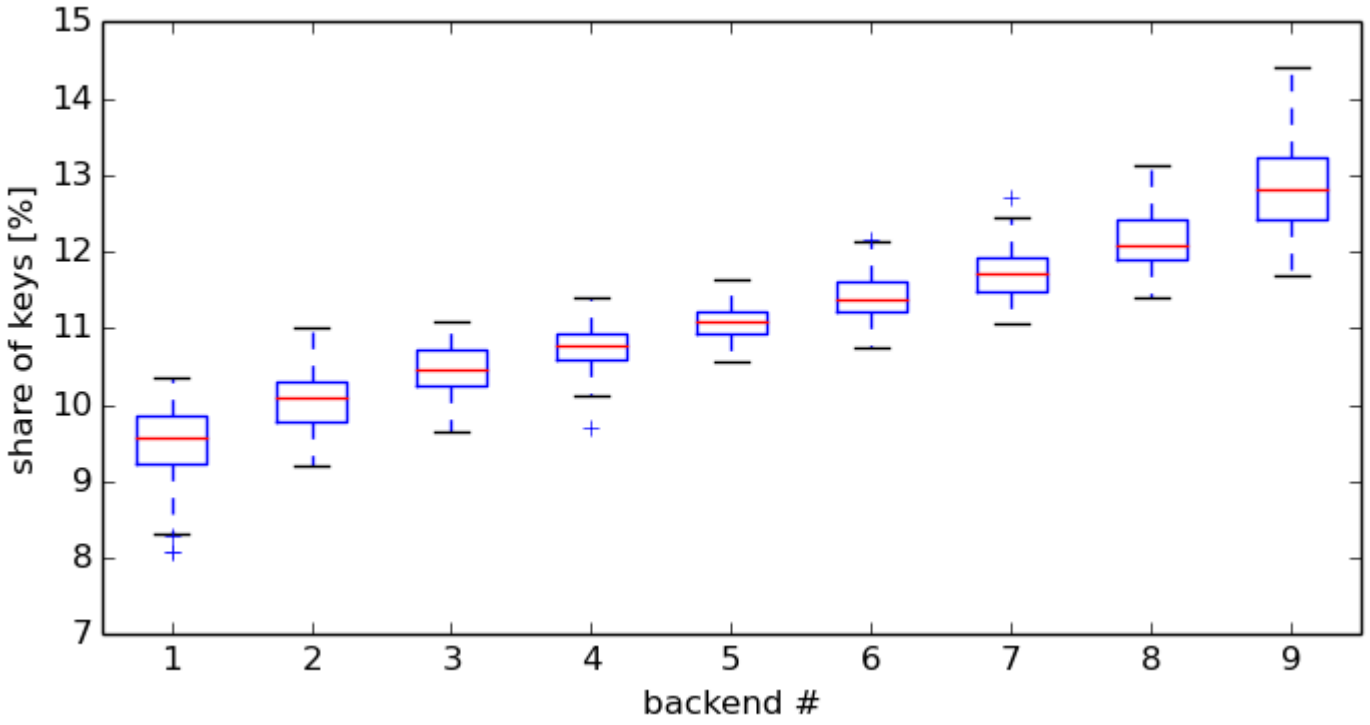
Load Balance of Ring Hash



100 points per server on the ring

https://blog.memcachier.com/2017/09/01/maglev-our-new-consistent-hashing-scheme/

Load Balance of Ring Hash



100 points per server on the ring

Goal: Strict load balance among servers, still consistent hash.

https://blog.memcachier.com/2017/09/01/maglev-our-new-consistent-hashing-scheme/

Google Maglev Consistent Hashing

- Hash every backend server to preference list of connection table positions
- Prime table size P for easy computation
- Hash every backend server to (offset, skip) $\in$ [0, P-1] $\times$ [1, P-1]
- Each backend server i'th preference is (offset + i $\times$ skip) mod P
- Backend servers take turns claiming most-preferred empty bucket

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/44824.pdf

Maglev Lookup Alg

Pseudocode 1 Populate Maglev hashing lookup table.

```
1:  function POPULATE
2:      for each i < N do next[i] ← 0 end for
3:      for each j < M do entry[j] ← −1 end for
4:      n ← 0
5:      while true do
6:          for each i < N do
7:              c ← permutation[i][next[i]]
8:              while entry[c] ≥ 0 do
9:                  next[i] ← next[i] + 1
10:                 c ← permutation[i][next[i]]
11:             end while
12:             entry[c] ← i
13:             next[i] ← next[i] + 1
14:             n ← n + 1
15:             if n = M then return end if
16:         end for
17:     end while
18: end function
```

Consistent Hashing Example

	S0	S1	S2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

Permutation Table

	S0	S1	S2
0			
1			
2			
3			
4			
5			
6			

Consistent Hashing Example

	S0	S1	S2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

Permutation Table

	S0	S1	S2
0	3		
1			
2			
3			
4			
5			
6			

Consistent Hashing Example

	S0	S1	S2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

Permutation Table

	S0	S1	S2
0	3		
1	0		
2			
3			
4			
5			
6			

Consistent Hashing Example

	S0	S1	S2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

Permutation Table

	S0	S1	S2
0	3		
1	0		
2	4		
3			
4			
5			
6			

Consistent Hashing Example

	S0	S1	S2
Offset	3	0	3
Skip	4	2	1

permutation[i] = (offset + i * skip) % 7

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	
1	
2	
3	
4	
5	
6	

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	
1	
2	
3	S0
4	
5	
6	

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	S1
1	
2	
3	S0
4	
5	
6	

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	S1
1	
2	
3	S0
4	
5	
6	

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	S1
1	
2	
3	S0
4	S2
5	
6	

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	S1
1	
2	
3	S0
4	S2
5	
6	

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	S1
1	
2	
3	S0
4	S2
5	
6	

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	S1
1	S0
2	
3	S0
4	S2
5	
6	

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	S1
1	S0
2	S1
3	S0
4	S2
5	S2
6	S0

Consistent Hashing Example

Permutation Table

	S0	S1	S2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

	Before	After
0	S1	S0
1	S0	S0
2	S1	S0
3	S0	S0
4	S2	S2
5	S2	S2
6	S0	S2

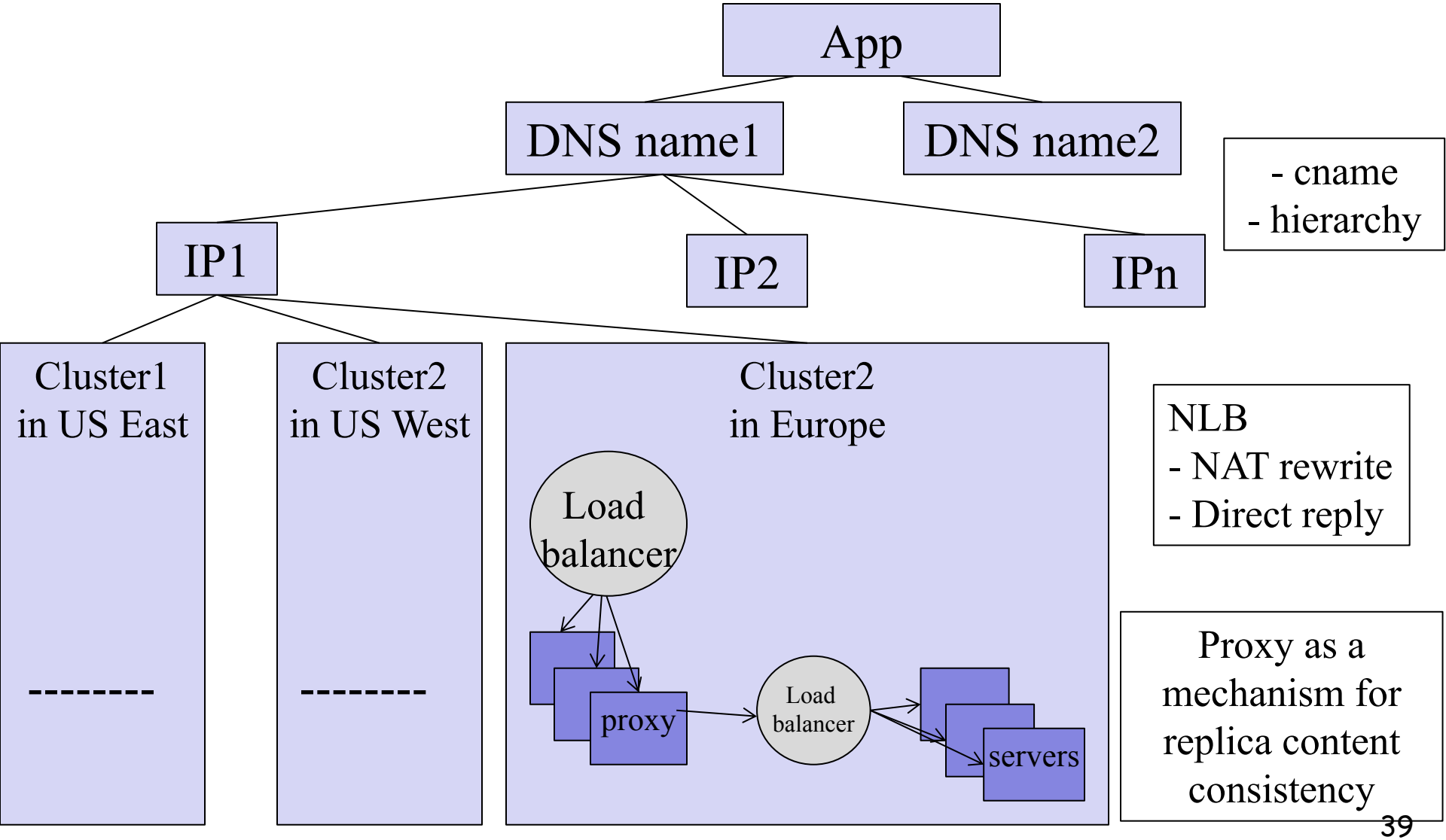
Maglev Analysis

- Load balancing of Maglev is relatively clear
- Offline exercise: Conduct analysis on Maglev hash (used by both Google and Facebook load balancers) on disruption tolerant performance

Recap: The Request Routing Journey

- Requirements/goals
  - naming abstraction, server load balancing, failure detection, access control filtering, priorities/QoS, request locality, transparent caching
- Components
  - Service/resource discovery (static, zookeeper, etcs, consul)
  - Health/state monitoring of servers/connecting networks
  - Load balancing mechanisms/algorithm (also called request routing)
    - Request routing mechanisms (DNS, network, hybrid)
    - DNS request routing (indirection, hierarchy)
    - Network request routing
      - ARP (one level of indirection to decouple L2 IP and L3 MAC)
      - NAT (to address socket multiplexing issue)
      - Direct Server Return (DSR)
    - Request routing director reliability
      - Fully distributed
      - VRRP (active/passive, broadcast leader election based on priority)
      - Connection tracking to handle server churn
      - Consistent hashing to reduce disruption (Ring Hash)
      - Consistent hashing for load balancing (Megalev)

Recap: Typical Request Routing Direction Mechanisms

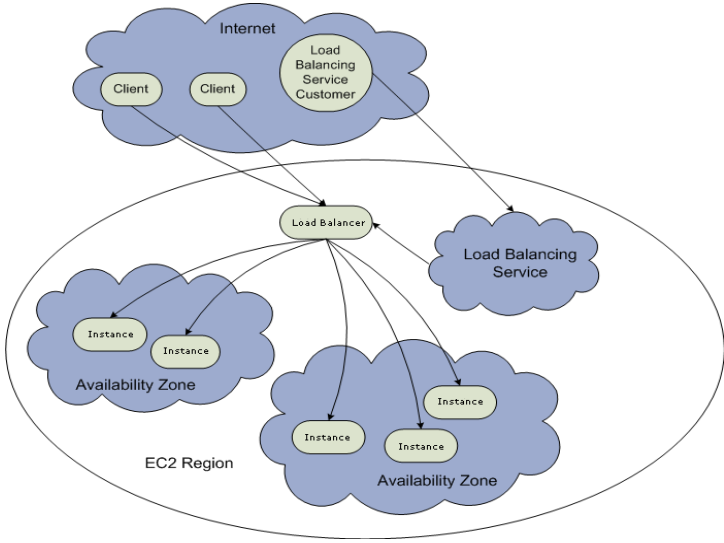


Offline Read: Amazon Load Balance

- https://aws.amazon.com/elasticloadbalancing/features/#Details_for_Elastic_Load_Balancing_Products
- https://aws.amazon.com/elasticloadbalancing/getting-started/?nc=sn&loc=4
  - Application LB
    - https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html
  - Network LB
    - https://docs.aws.amazon.com/elasticloadbalancing/latest/network/introduction.html

Offline Read: Amazon Elastic Cloud (EC2) Elastic Load Balancing: Network Load Balance

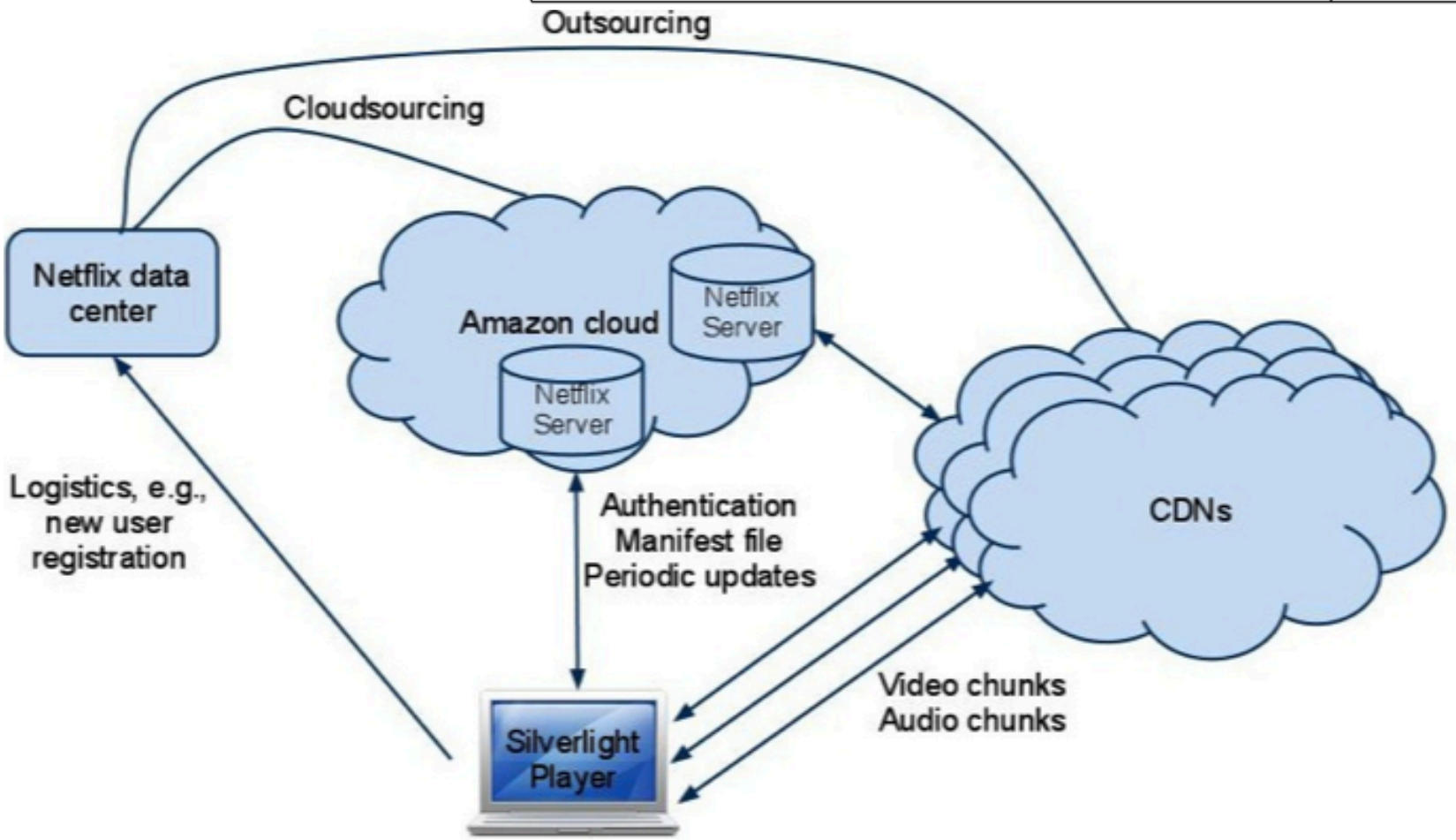
- Use the *create-load-balancer* command to create an Elastic Load Balancer.
- Use the *create-target-group* command to create a target group.
- Use the *register-targets* command to register the Amazon EC2 instances with the target group.
- Use the *create-listener* command to create a listener to forward requests to target group



https://docs.aws.amazon.com/elasticloadbalancing/latest/network/network-load-balancer-cli.html

Offline Example: Netflix

Hostname	Organization
www.netflix.com	Netflix
signup.netflix.com	Amazon
movies.netflix.com	Amazon
agmoviecontrol.netflix.com	Amazon
nflx.i.87f50a04.x.lcdn.nflximg.com	Level 3
netflix-753.vo.llnwd.net	Limelight
netflix753.as.nflximg.com.edgesuite.net	Akamai



Offline Example: Netflix Manifest File

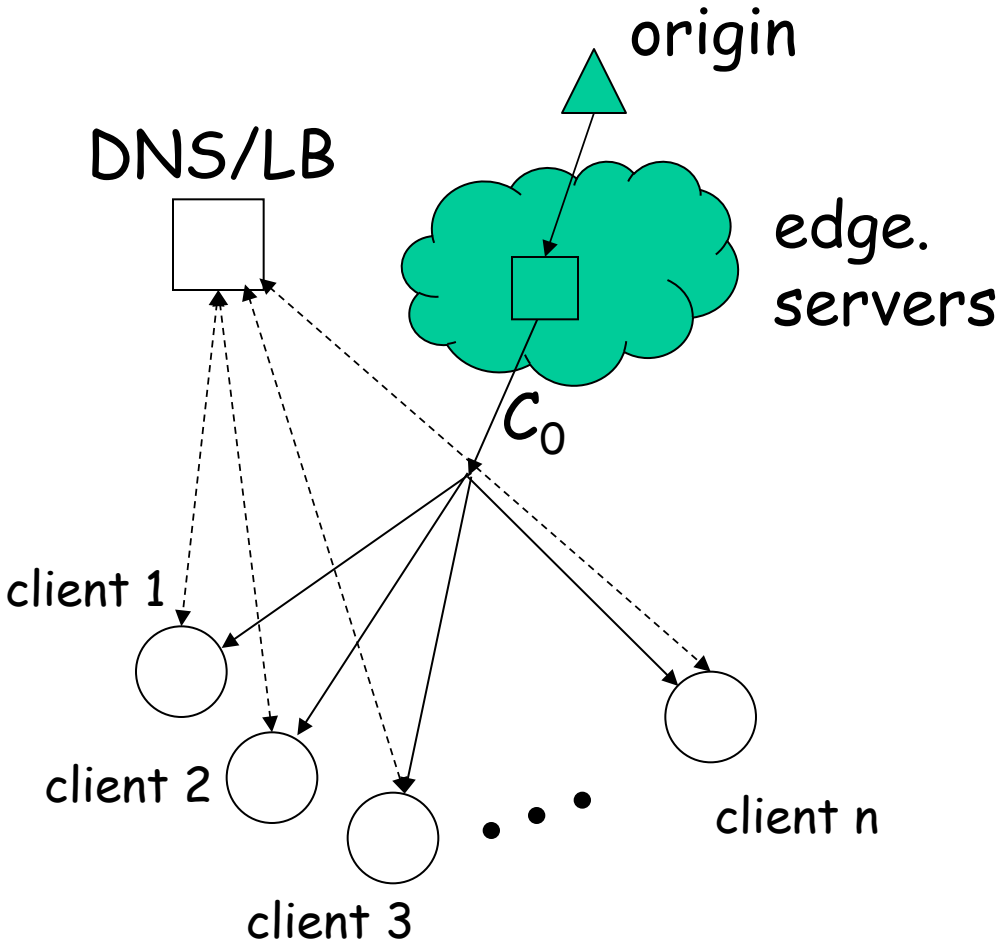
- Client player authenticates and then downloads manifest file from servers at Amazon Cloud

```
<nccp:cdns>
    <nccp:cdn>
        <nccp:name>level3</nccp:name>
        <nccp:cdnid>6</nccp:cdnid>
        <nccp:rank>1</nccp:rank>
        <nccp:weight>140</nccp:weight>
    </nccp:cdn>
    <nccp:cdn>
        <nccp:name>limelight</nccp:name>
        <nccp:cdnid>4</nccp:cdnid>
        <nccp:rank>2</nccp:rank>
        <nccp:weight>120</nccp:weight>
    </nccp:cdn>
    <nccp:cdn>
        <nccp:name>akamai</nccp:name>
        <nccp:cdnid>9</nccp:cdnid>
        <nccp:rank>3</nccp:rank>
        <nccp:weight>100</nccp:weight>
    </nccp:cdn>
</nccp:cdns>
```

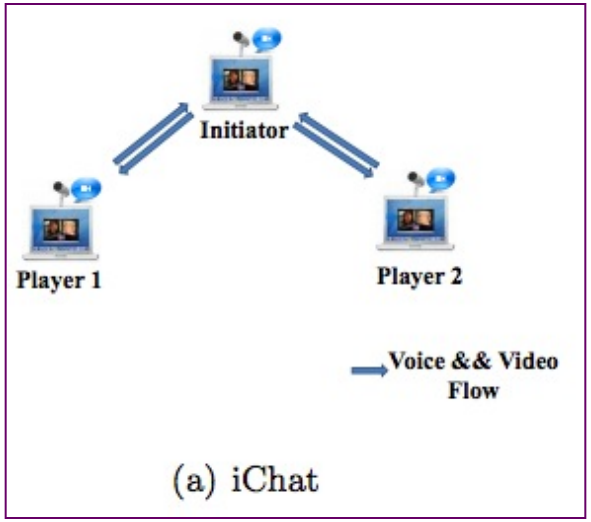
Offline Example: Netflix Manifest File

```
<nccp:bitrate>560</nccp:bitrate>
<nccp:videoprofile>
   playready-h264mpl30-dash
</nccp:videoprofile>
<nccp:resolution>
   <nccp:width>512</nccp:width>
   <nccp:height>384</nccp:height>
</nccp:resolution>
<nccp:pixelaspect>
   <nccp:width>4</nccp:width>
   <nccp:height>3</nccp:height>
</nccp:pixelaspect>v
<nccp:downloadurls>
   <nccp:downloadurl>
      <nccp:expiration>131xxx</nccp:expiration>
      <nccp:cdnid>6</nccp:cdnid>
      <nccp:url>http://nflx.i.../...</nccp:url>
   </nccp:downloadurl>
   <nccp:downloadurl>
      <nccp:expiration>131xxx</nccp:expiration>
      <nccp:cdnid>4</nccp:cdnid>
      <nccp:url>http://netflix.../...</nccp:url>
   </nccp:downloadurl>
   <nccp:downloadurl>
      <nccp:expiration>131xxx</nccp:expiration>
      <nccp:cdnid>9</nccp:cdnid>
      <nccp:url>http://netflix.../...</nccp:url>
   </nccp:downloadurl>
</nccp:downloadurls>
```

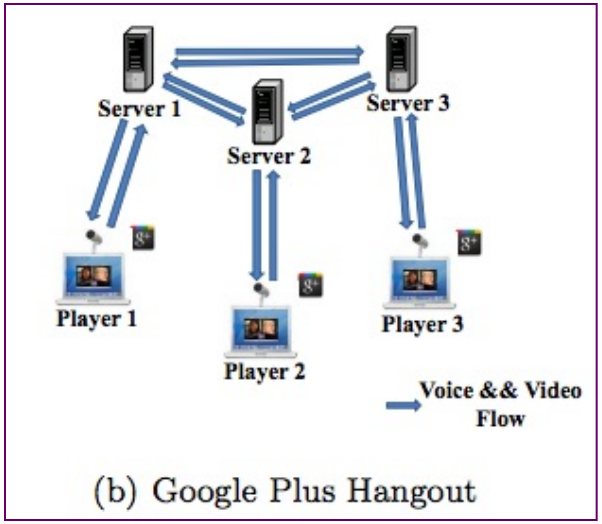
Scalability of Server-Only Approaches



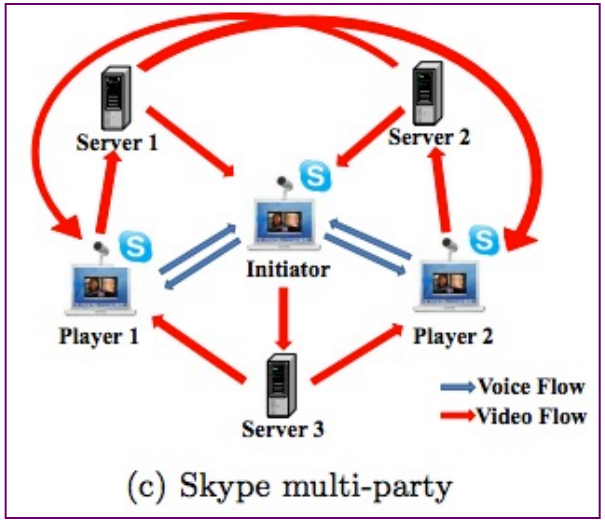
Server + Host Systems



(a) iChat



(b) Google Plus Hangout

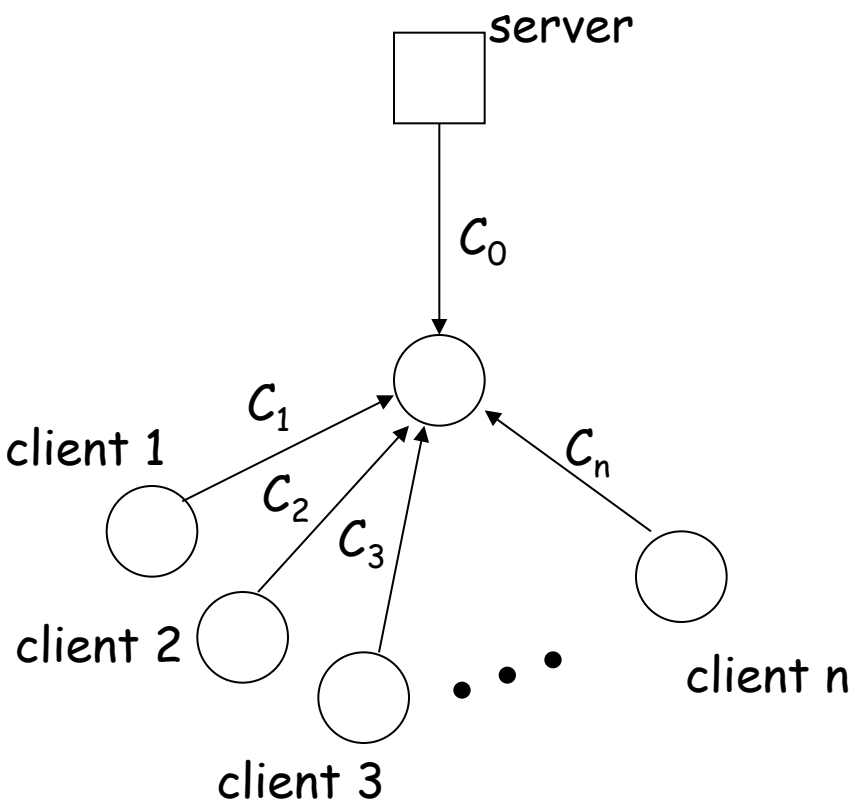


(c) Skype multi-party

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

An Upper Bound on Server+Host Scalability

- Assume
  - need to achieve same rate to all clients
  - only uplinks can be bottlenecks
- What is an upper bound on scalability?

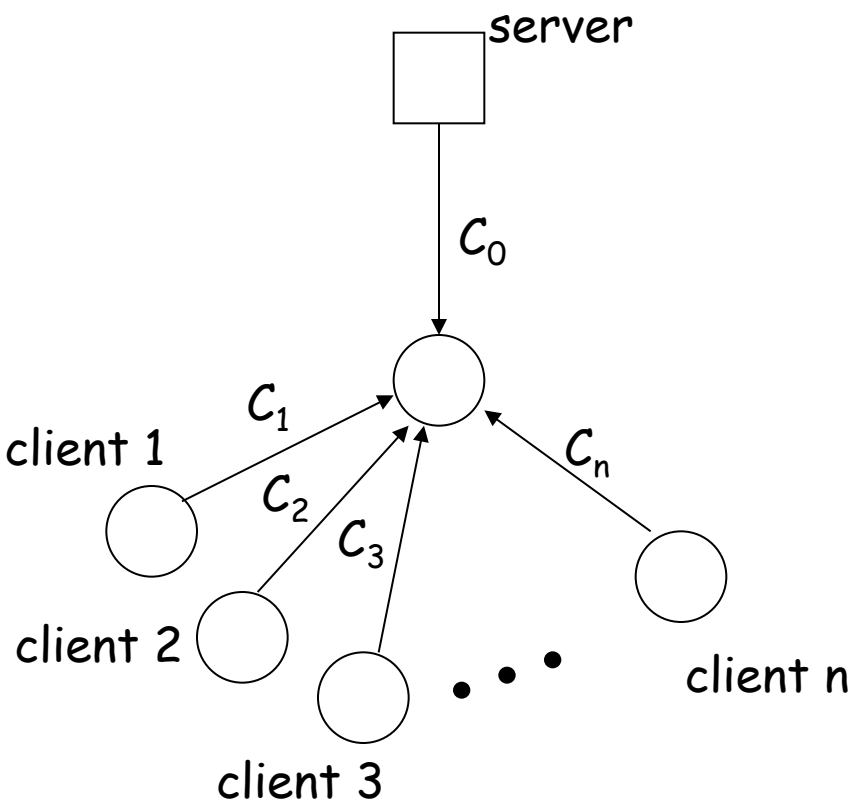


The Scalability Problem

- Maximum throughput

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

- The bound is theoretically approachable



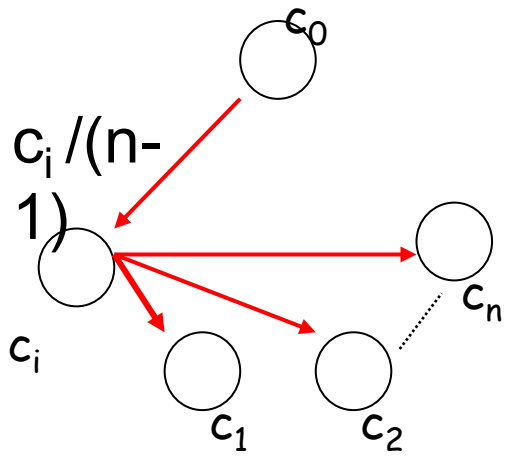
Theoretical Capacity: upload is bottleneck

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

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

  server → client i: $c_i/(n-1)$

  client i → other n-1 clients

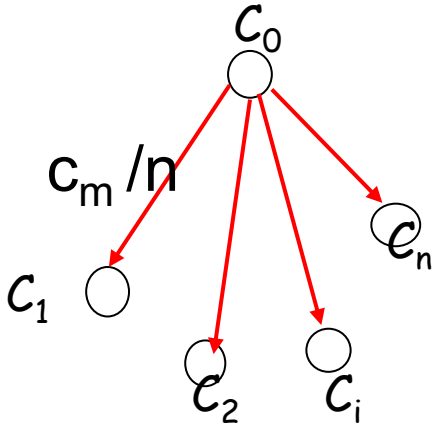


- Tree 0:

  server has remaining

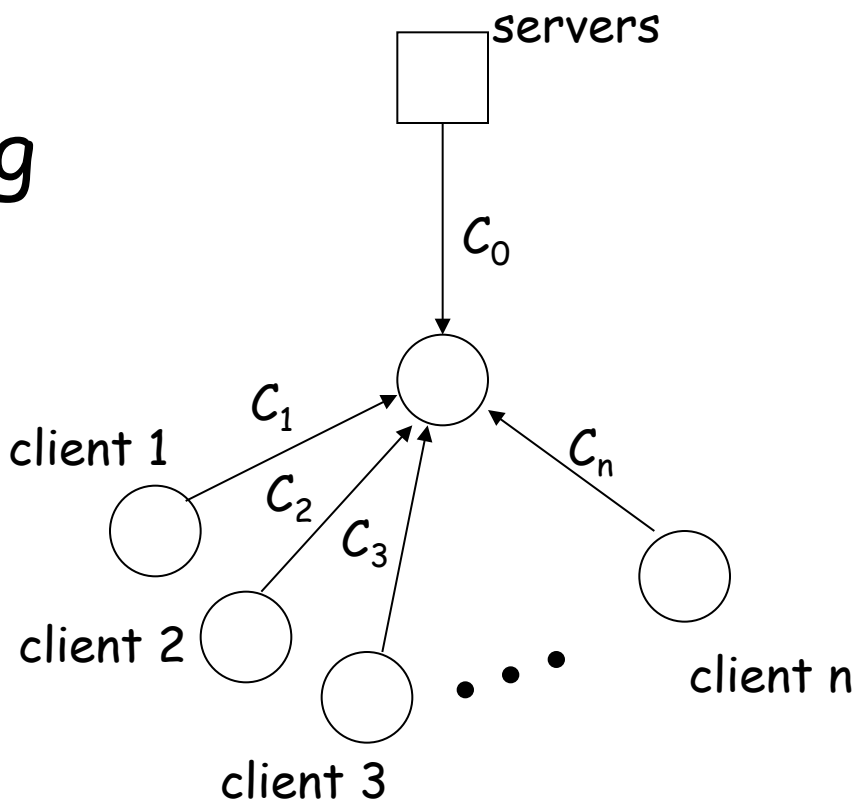
  $c_m = c0 - (c1 + c2 + \ldots cn)/(n-1)$

  send to client i: cm/n



Why not Building the Trees?

- Clients come and go (churns): maintaining the trees is too expensive
- Each client needs N connections (not feasible for large systems)

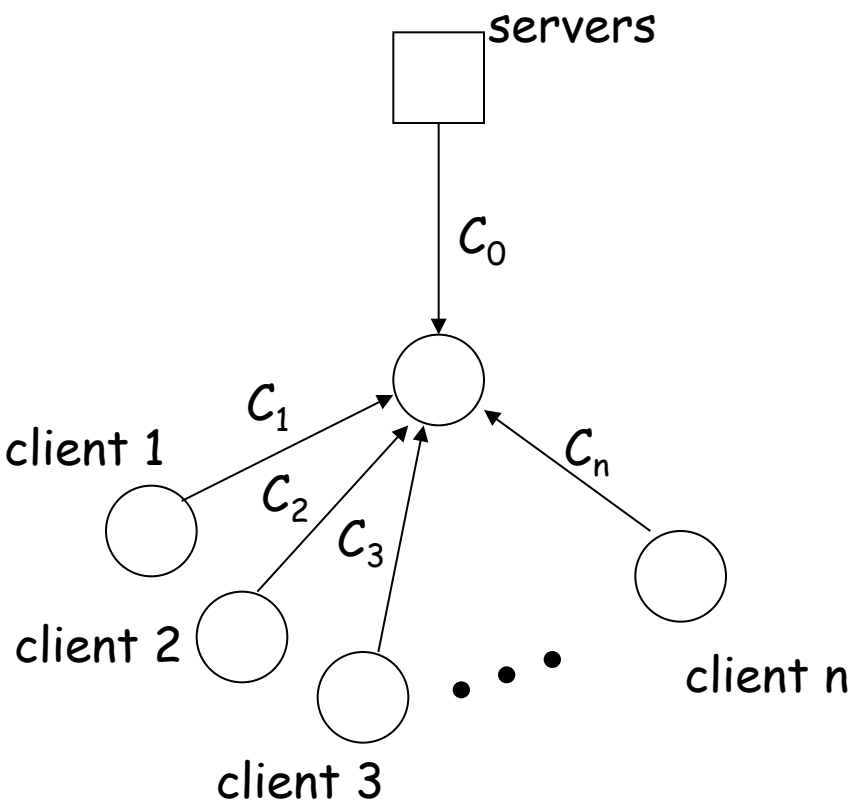


Outline

- Admin and recap
- Multi servers systems
- Distributed servers
  - distributed content distribution
    - upper bound analysis
    - BitTorrent design
    - distributed content distribution using Freenet
    - distributed content distribution anonymity (Tor)
    - distributed content verification (Block chain)

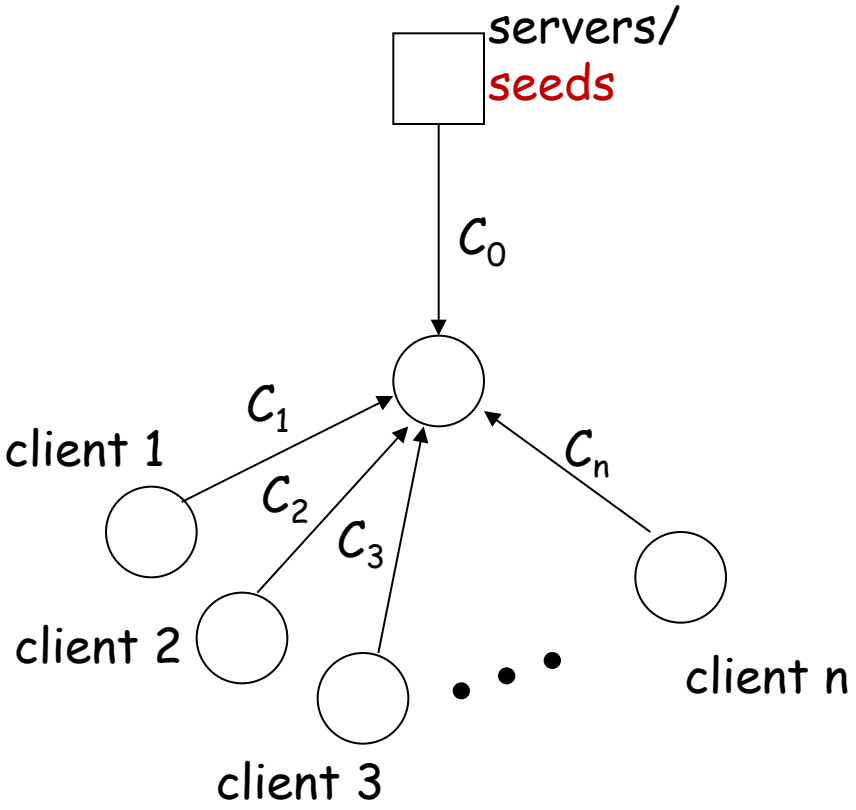
Server+Host Content Distribution: Key Design Issues

- 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?

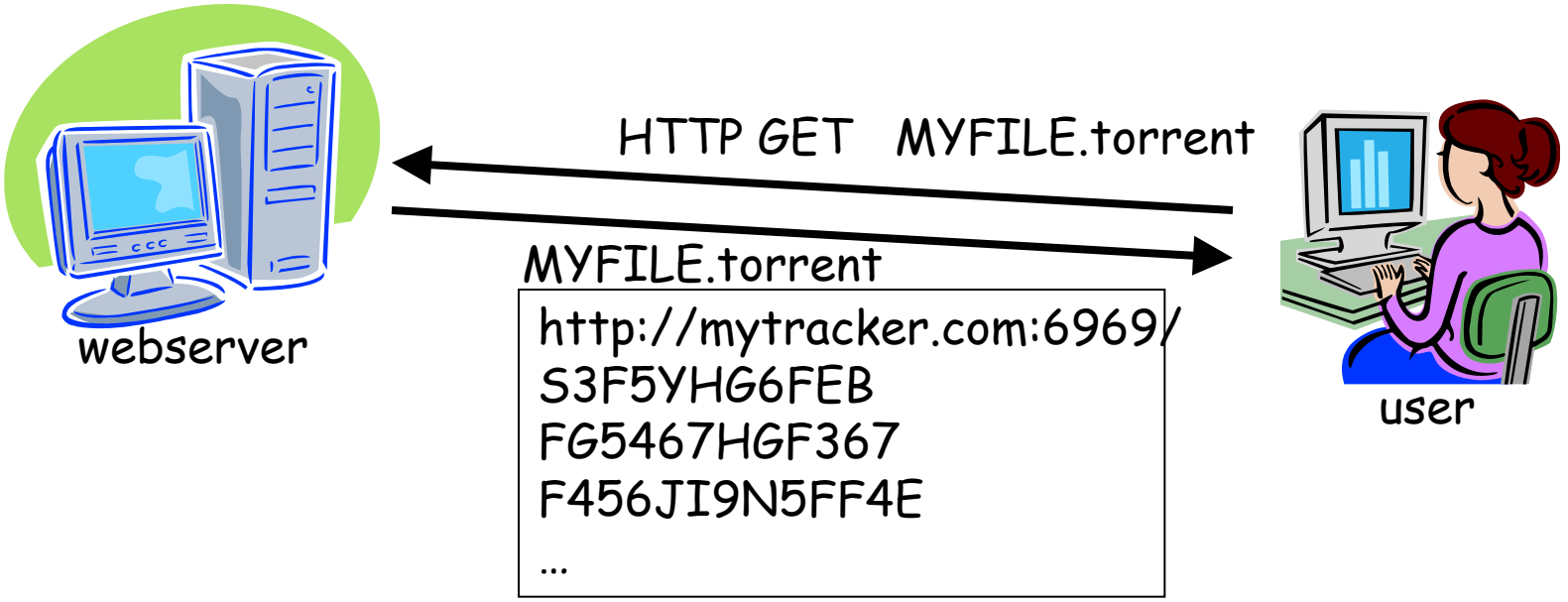
- Robustness
- Efficiency
- Incentive



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: Content Lookup



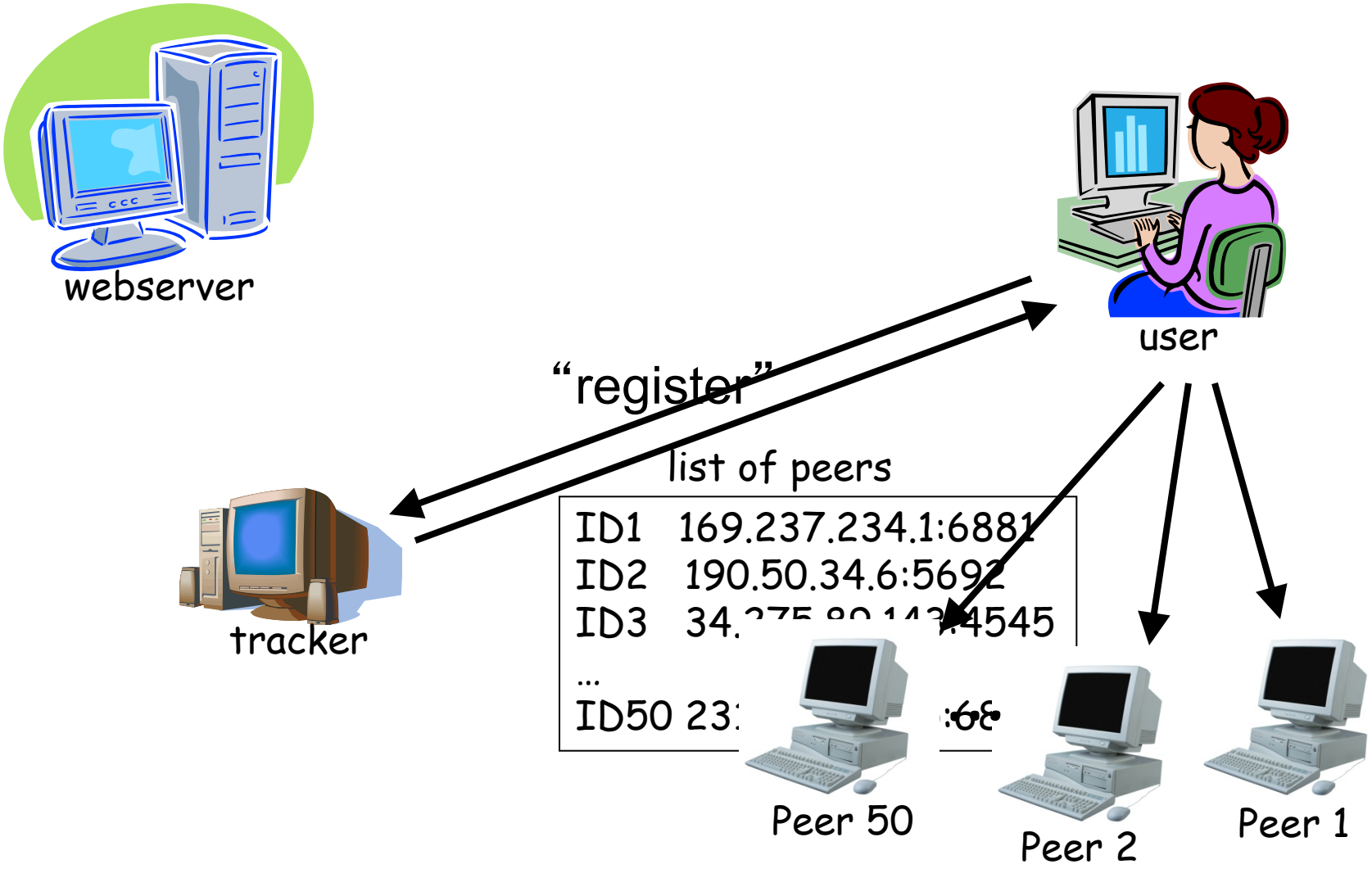
Metadata (.torrent) File Structure

- 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

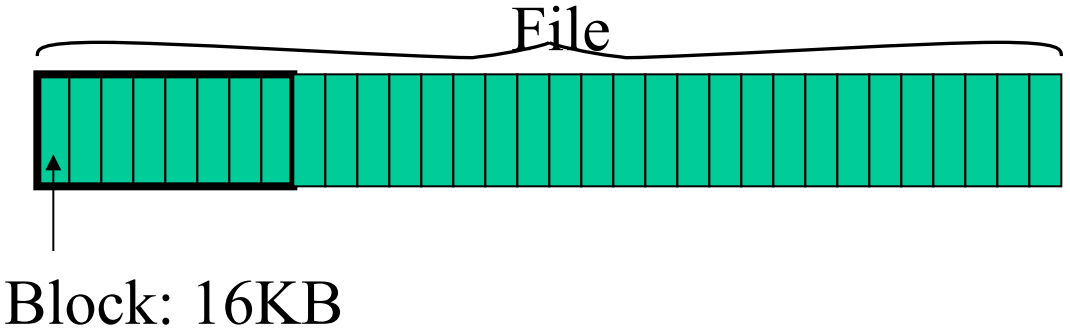
Tracker Protocol



Peer Protocol: Piece-based Swarming

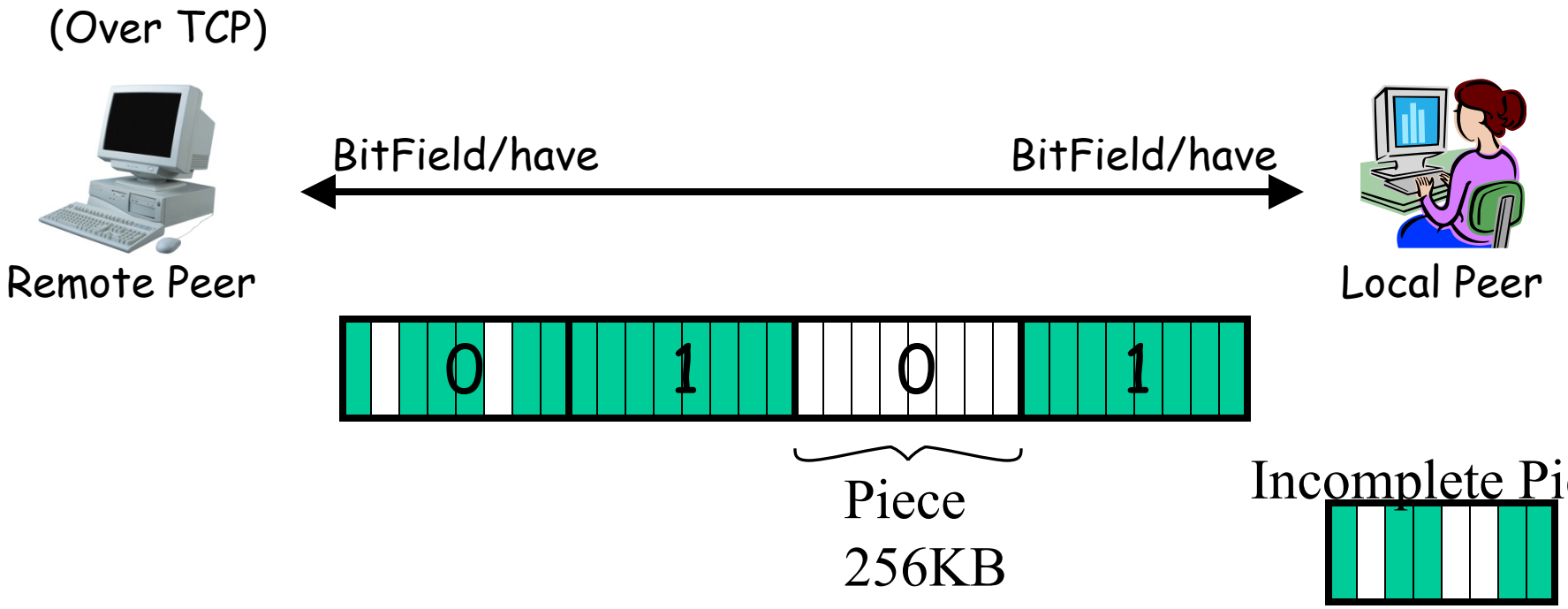
- Divide a large file into small blocks and request block-size content from different peers (why?)

  Block: unit of download



- 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

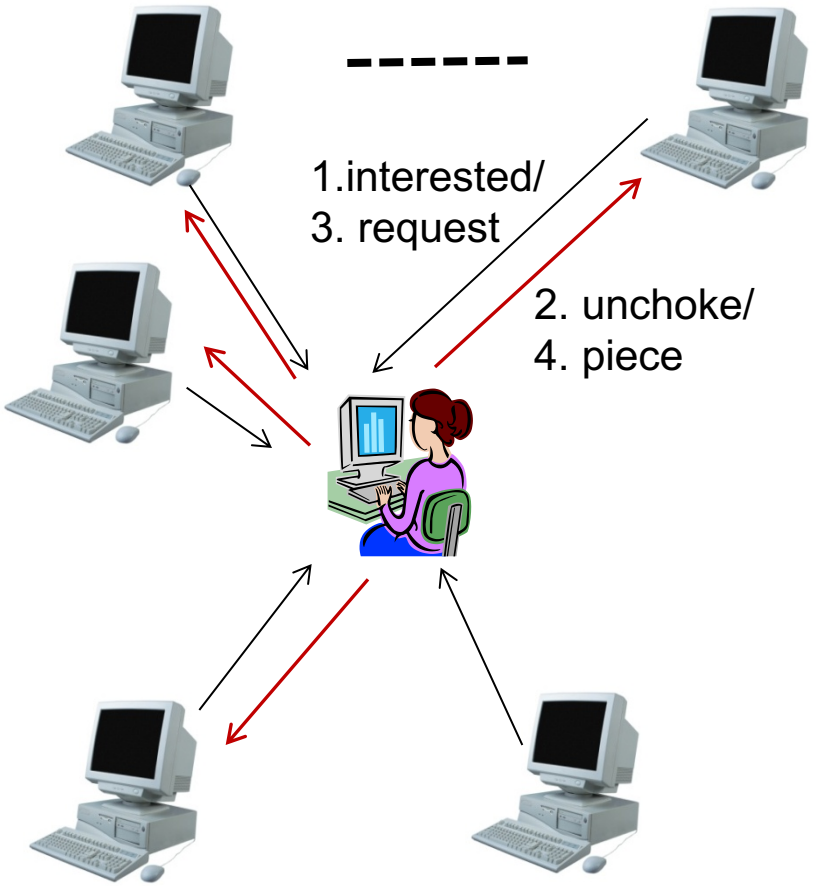


- Peers exchange bitmap representing content availability
  - `bitfield` msg during initial connection
  - `have` msg to notify updates to bitmap
  - to reduce bitmap size, aggregate multiple blocks as a piece

Peer Request

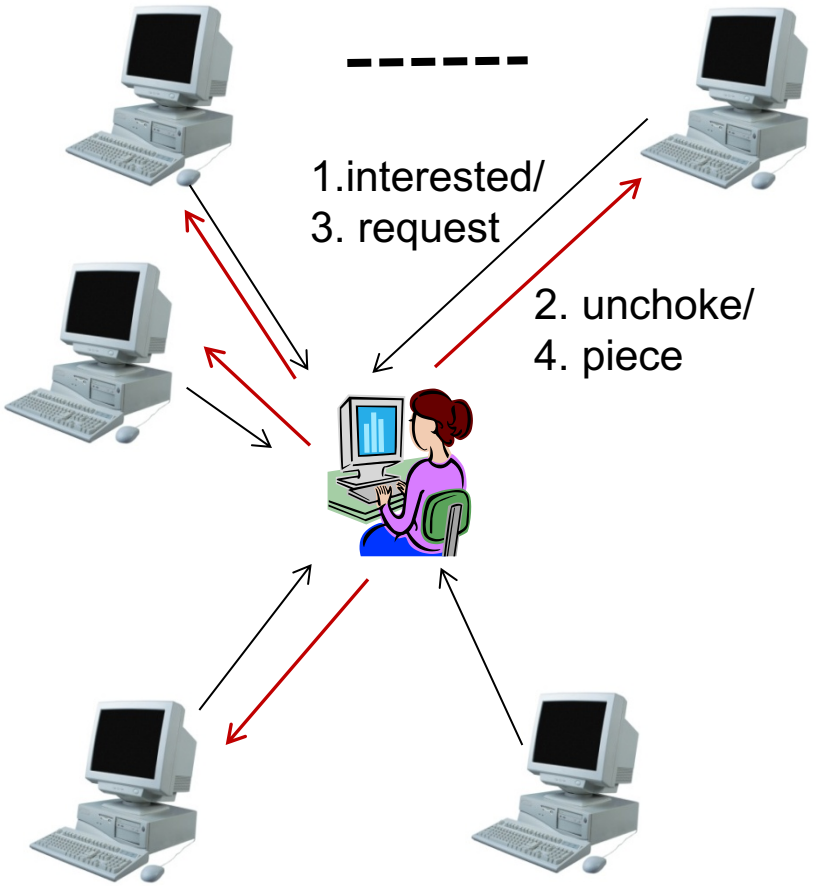
http://www.bittorrent.org/beps/bep_0003.html

- 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

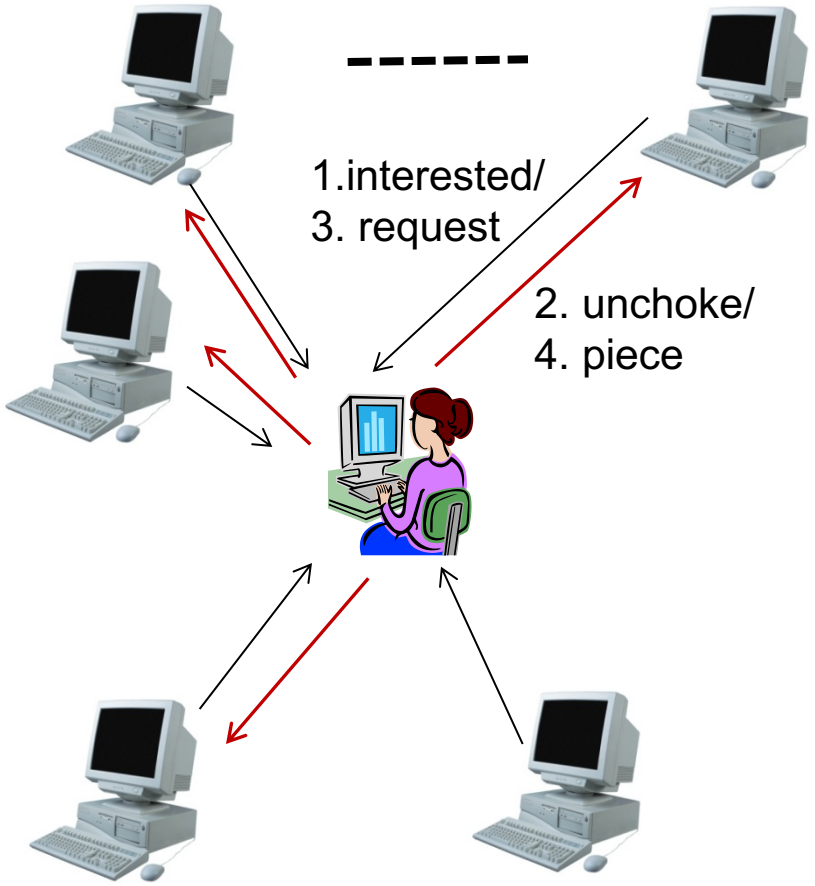
- Request (local) rarest first
  - achieves the fastest replication of rare pieces
  - obtain something of value

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 (why?)

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 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

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

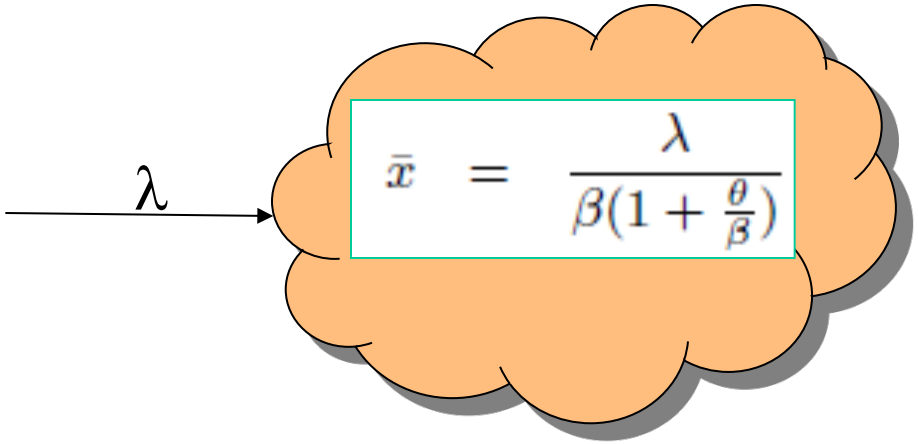
Solving steady state: $\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{\mathrm{d}y(t)}{\mathrm{d}t} = 0$

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

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

System State

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



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

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

Discussion

- Which protocol features of BitTorrent do you like?
- What are some remaining issues that the BitTorrent system does not address?