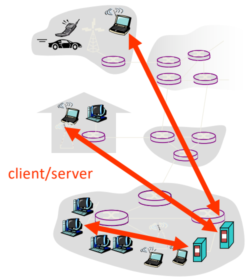
Topic 6 – Applications

- Overview
- Infrastructure Services (DNS)
- Traditional Applications (web)
- Multimedia Applications (SIP)
- P2P Networks

1

1

Client-server paradigm



server:
- always-on host
- permanent IP address
- server farms for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

2

2

Relationship Between Names&Addresses

- Addresses can change underneath
  - Move www.bbc.co.uk to 212.58.246.92
  - Humans/Apps should be unaffected
- Name could map to multiple IP addresses
  - www.bbc.co.uk to multiple replicas of the Web site
  - Enables
    - Load-balancing
    - Reducing latency by picking nearby servers
- Multiple names for the same address
  - E.g., aliases like www.bbc.co.uk and bbc.co.uk
  - Mnemonic stable name, and dynamic canonical name
    - Canonical name = actual name of host

10

10

Mapping from Names to Addresses

- Originally: per-host file /etc/hosts
  - SRI (Menlo Park) kept master copy
  - Downloaded regularly
  - Flat namespace
- Single server not resilient, doesn't scale
  - Adopted a distributed hierarchical system
- Two intertwined hierarchies:
  - Infrastructure: hierarchy of DNS servers
  - Naming structure: www.bbc.co.uk

11

11

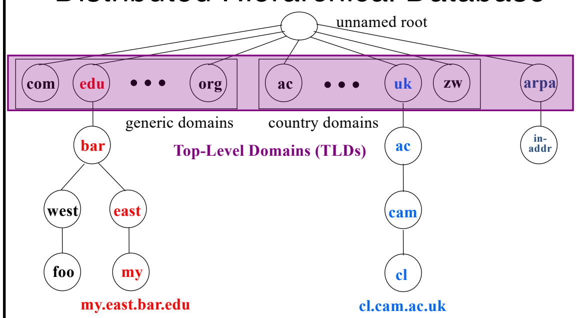
Domain Name System (DNS)

- Top of hierarchy: Root
  - Location hardwired into other servers
- Next Level: Top-level domain (TLD) servers
  - .com, .edu, etc.
  - .uk, .au, .to, etc.
  - Managed professionally
- Bottom Level: Authoritative DNS servers
  - Actually do the mapping
  - Can be maintained locally or by a service provider

12

12

Distributed Hierarchical Database

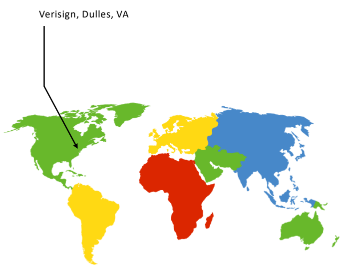


13

13

DNS Root

- Located in Virginia, USA
- How do we make the root scale?

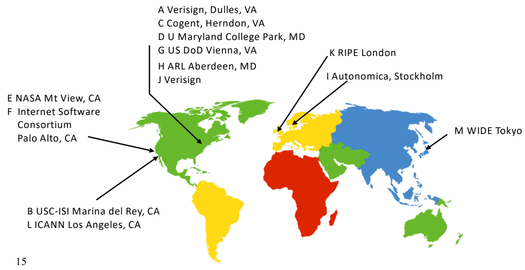


14

14

DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
  - Labeled A through M
- Does this scale?

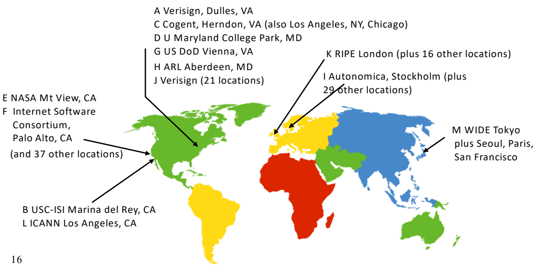


15

15

DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
  - Labeled A through M
- Replication via any-casting (localized routing for addresses)



16

16

Using DNS

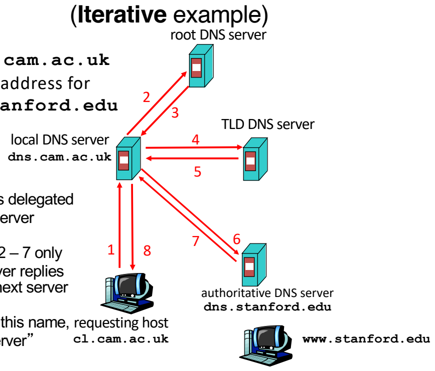
- Two components
  - Local DNS servers
  - Resolver software on hosts
- Local DNS server ("default name server")
  - Usually near the endhosts that use it
  - Local hosts configured with local server (e.g., /etc/resolv.conf) or learn server via DHCP
- Client application
  - Extract server name (e.g., from the URL)
  - Do gethostbyname() to trigger resolver code

17

17

How Does Resolution Happen?
(**Iterative** example)

Host at **cl.cam.ac.uk** wants IP address for **www.stanford.edu**



iterated query:

- Host enquiry is delegated to local DNS server
- Consider transactions 2 – 7 only
- contacted server replies with name of next server to contact
- "I don't know this name, but ask this server"

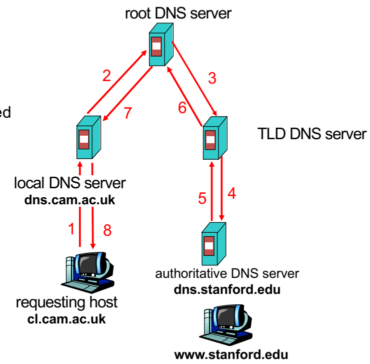
18

18

DNS name resolution **recursive** example

recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

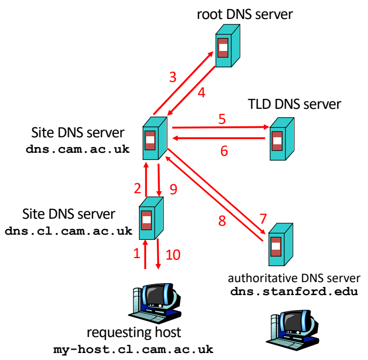


19

19

Recursive and Iterative Queries - **Hybrid** case

- Recursive query
  - Ask server to get answer for you
  - E.g., requests 1,2 and responses 9,10
- Iterative query
  - Ask server who to ask next
  - E.g., all other request-response pairs



20

20

DNS Caching

- Performing all these queries takes time
  - And all this before actual communication takes place
  - E.g., 1-second latency before starting Web download
- Caching can greatly reduce overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., www.bbc.co.uk) visited often
  - Local DNS server often has the information cached
- How DNS caching works
  - DNS servers cache responses to queries
  - Responses include a "time to live" (TTL) field
  - Server deletes cached entry after TTL expires

21

21

Negative Caching

- Remember things that don't work
  - Misspellings like *bbcc.co.uk* and *www.bbc.com.uk*
  - These can take a long time to fail the first time
  - Good to remember that they don't work
  - … so the failure takes less time the next time around
- But: negative caching is optional
  - And not widely implemented

22

22

Reliability

- DNS servers are replicated (primary/secondary)
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- Usually, UDP used for queries
  - Need reliability: must implement this on top of UDP
  - Spec supports TCP too, but not always implemented
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don't care which server responds

23

23

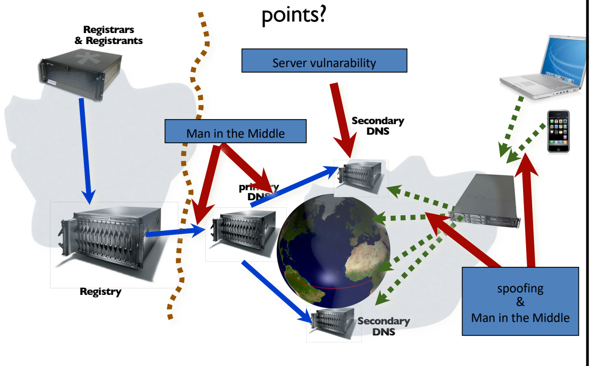
DNS and Security

- No way to verify answers
  - Opens up DNS to many potential attacks
  - DNSSEC fixes this
- Most obvious vulnerability: recursive resolution
  - Using recursive resolution, host must trust DNS server
  - When at Starbucks, server is under their control
  - And can return whatever values it wants
- More subtle attack: Cache poisoning
  - Those "additional" records can be anything!

24

24

Data flow through the DNS
Where are the vulnerable points?



25

DNSSEC protects all these end-to-end

- provides message authentication and integrity verification through cryptographic signatures
  - You know who provided the signature
  - No modifications between signing and validation
- It does **not** provide authorization
- It does **not** provide confidentiality
- It does **not** provide protection against DDOS

26

DNSSEC in practice

- Scaling the key signing and key distribution

Solution: Using the DNS to Distribute Keys

- Distributing keys through DNS hierarchy:
  - Use one trusted key to establish authenticity of other keys
  - Building chains of trust from the root down
  - Parents need to sign the keys of their children
- Only the root key needed in ideal world
  - Parents always delegate security to child

27

27

Why is the web so successful?

- What do the web, youtube, facebook, twitter, instagram, ….. have in common?
  - The ability to self-publish
- Self-publishing that is easy, independent, *free*
- No interest in collaborative and idealistic endeavor
  - People aren't looking for Nirvana (or even Xanadu)
  - People also aren't looking for technical perfection
- Want to make their mark, and find something neat
  - Two sides of the same coin, creates synergy
  - "Performance" more important than dialogue….

28

28

Web Components

- Infrastructure:
  - Clients
  - Servers
  - Proxies
- Content:
  - Individual objects (files, etc.)
  - Web sites (coherent collection of objects)
- Implementation
  - HTML: formatting content
  - URL: naming content
  - HTTP: protocol for exchanging content

    Any content not just HTML!

29

29

HTML: HyperText Markup Language

- A *Web page* has:
  - Base HTML file
  - Referenced objects (*e.g.*, images)
- HTML has several functions:
  - Format text
  - Reference images
  - Embed *hyperlinks* (HREF)

30

30

URL Syntax

protocol://hostname[:port]/directorypath/resource

protocol	http, ftp, https, smtp, rtsp, *etc.*
hostname	DNS name, IP address
port	Defaults to protocol's standard port *e.g.* http: 80 https: 443
directory path	Hierarchical, reflecting file system
resource	Identifies the desired resource Can also extend to program executions: `http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b`

31

31

HyperText Transfer Protocol (HTTP)

- Request-response protocol
- Reliance on a global namespace
- Resource *metadata*
- *Stateless*
- ASCII format (ok this changed….)

$ telnet www.cl.cam.ac.uk 80
GET /win HTTP/1.0
<blank line, i.e., CRLF>

32

32

Steps in HTTP Request

- HTTP Client initiates TCP connection to server
  - SYN
  - SYNACK
  - ACK
- Client sends HTTP request to server
  - Can be piggybacked on TCP's ACK
- HTTP Server responds to request
- Client receives the request, terminates connection
- TCP connection termination exchange

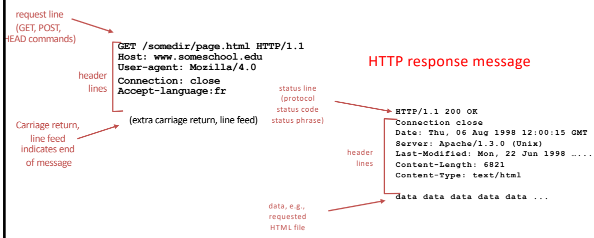
How many RTTs for a single request?

33

33

Client-Server Communication

- two types of HTTP messages: *request, response*
- HTTP request message: (GET POST HEAD ….)



34

34

Different Forms of Server Response

- Return a file
  - URL matches a file (*e.g.,* `/www/index.html`)
  - Server returns file as the response
  - Server generates appropriate response header
- Generate response dynamically
  - URL triggers a program on the server
  - Server runs program and sends output to client
- Return meta-data with no body

35

35

HTTP Resource Meta-Data

- Meta-data
  - Info *about* a resource, stored as a separate entity
- Examples:
  - Size of resource, last modification time, type of content
- Usage example: Conditional GET Request
  - Client requests object "`If-modified-since`"
  - If unchanged, "`HTTP/1.1 304 Not Modified`"
  - No body in the server's response, only a header

36

36

HTTP is *Stateless*

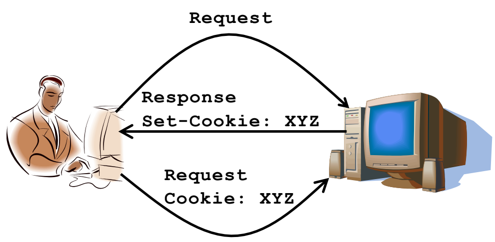
- Each request-response treated independently
  - Servers *not* required to retain state
- **Good**: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn't matter
- **Bad**: Some applications need persistent state
  - Need to uniquely identify user or store temporary info
  - *e.g.,* Shopping cart, user profiles, usage tracking, …

37

37

State in a Stateless Protocol:
Cookies

- *Client-side* state maintenance
  - Client stores small(?) state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication



38

38

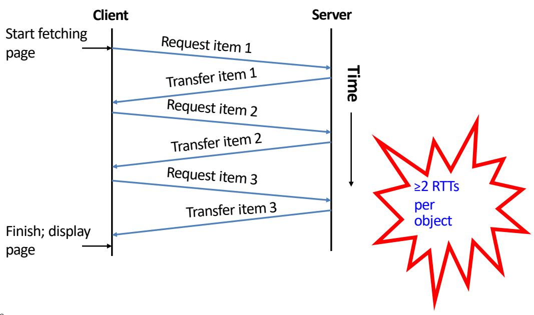
HTTP Performance

- Most Web pages have multiple objects
  - *e.g.,* HTML file and a bunch of embedded images
- How do you retrieve those objects (naively)?
  - *One item at a time*
- Put stuff in the optimal place?
  - *Where is that precisely?*
    - ***Enter the Web cache and the CDN***

39

39

Fetch HTTP Items: Stop & Wait



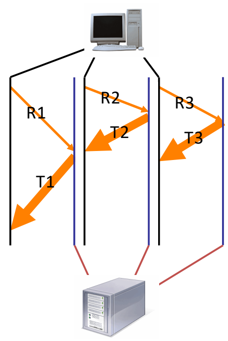
40

40

Improving HTTP Performance:
Concurrent Requests & Responses

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses

- Client = ☺
- Server = ☺
- Network = ☹ Why?

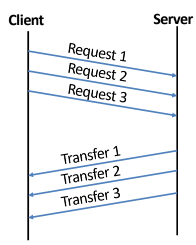


41

41

Improving HTTP Performance:
Pipelined Requests & Responses

- *Batch* requests and responses
  - Reduce connection overhead
  - Multiple requests sent in a single batch
  - Maintains order of responses
  - Item 1 always arrives before item 2
- How is this different from concurrent requests/responses?
  - Single TCP connection



42

42

Improving HTTP Performance:
Persistent Connections

- Enables multiple transfers per connection
  - Maintain TCP connection across multiple requests
  - Including transfers subsequent to current page
  - Client or server can tear down connection
- Performance advantages:
  - Avoid overhead of connection set-up and tear-down
  - Allow TCP to learn more accurate RTT estimate
  - Allow TCP congestion window to increase
  - i.e., leverage previously discovered bandwidth
- Default in HTTP/1.1

43

43

HTTP *evolution*

- 1.0 – one object per TCP: simple but slow
- Parallel connections - multiple TCP, one object each: wastes b/w, may be svr limited, out of order
- 1.1 pipelining – aggregate retrieval time: ordered, multiple objects sharing single TCP
- 1.1 persistent – aggregate TCP overhead: lower overhead in time, increase overhead at ends (e.g., when should/do you close the connection?)

44

44

Scorecard: Getting n Small Objects

Time dominated by latency

- One-at-a-time: ~2n RTT
- Persistent: ~ (n+1)RTT
- M concurrent: ~2[n/m] RTT
- Pipelined: ~2 RTT
- Pipelined/Persistent: ~2 RTT first time, RTT later

45

45

Scorecard: Getting n Large Objects

Time dominated by bandwidth

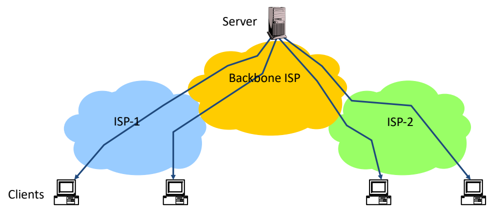
- One-at-a-time: ~ nF/B
- M concurrent: ~ [n/m] F/B
  - assuming shared with large population of users
- Pipelined and/or persistent: ~ nF/B
  - The only thing that helps is getting more bandwidth..

46

46

Improving HTTP Performance:
Caching

- Many clients transfer the same information
  - Generates redundant server and network load
  - Clients experience unnecessary latency



47

47

Improving HTTP Performance:
Caching: How

- Modifier to GET requests:
  - `If-modified-since` – returns "not modified" if resource not modified since specified time
- Response header:
  - `Expires` – how long it's safe to cache the resource
  - `No-cache` – ignore all caches; always get resource directly from server

48

48

Improving HTTP Performance:
Caching: Why

- Motive for placing content closer to client:
  - User gets better response time
  - Content providers get happier users
    - Time is money, really!
  - Network gets reduced load
- Why does caching work?
  - Exploits *locality of reference*
- How well does caching work?
  - Very well, up to a limit
  - Large overlap in content
  - But many unique requests

49

49

Improving HTTP Performance:
Caching on the Client

Example: Conditional GET Request

- Return resource only if it has changed at the server
  - Save server resources!

Request from client to server:

```
GET /~awm22/win HTTP/1.1
Host: www.cl.cam.ac.uk
User-Agent: Mozilla/4.03
If-Modified-Since: Sun, 27 Aug 2006 22:25:50 GMT
```

- How?
  - Client specifies “if-modified-since” time in request
  - Server compares this against “last modified” time of desired resource
  - Server returns “304 Not Modified” if resource has not changed
  - …. or a “200 OK” with the latest version otherwise

50

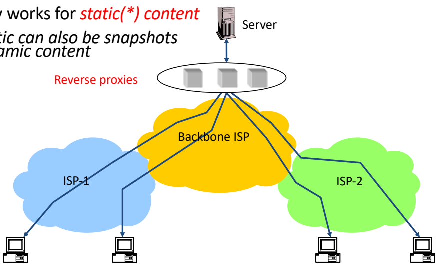
50

Improving HTTP Performance:
Caching with Reverse Proxies

Cache documents close to **server**
→ decrease server load

- Typically done by content providers
- Only works for *static(*) content*

(*) static can also be snapshots of dynamic content



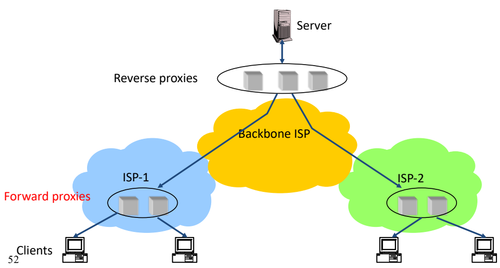
51

51

Improving HTTP Performance:
Caching with Forward Proxies

Cache documents close to **clients**
→ reduce network traffic and decrease latency

- Typically done by ISPs or corporate LANs



52

52

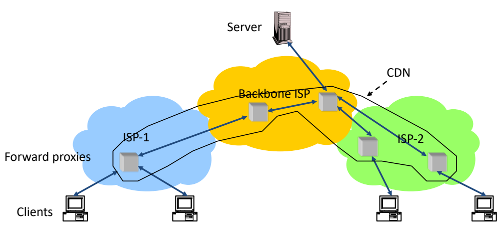
Improving HTTP Performance:
Caching w/ Content Distribution Networks

- Integrate forward and reverse caching functionality
  - One overlay network (usually) administered by one entity
  - *e.g.,* Akamai
- Provide document caching
  - **Pull:** Direct result of clients’ requests
  - **Push:** Expectation of high access rate
- Also do some processing
  - Handle *dynamic* web pages
  - *Transcoding*
  - *Maybe do some security function – watermark IP*

53

53

Improving HTTP Performance:
Caching with CDNs (cont.)



54

54

Improving HTTP Performance:
CDN Example – Akamai

- Akamai creates new domain names for each client content provider.
  - e.g., *a128.g.akamai.net*
- The CDN’s DNS servers are authoritative for the new domains
- The client content provider modifies its content so that embedded URLs reference the new domains.
  - “Akamaize” content
  - e.g.: *http://www.bbc.co.uk/popular-image.jpg* becomes *http://a128.g.akamai.net/popular-image.jpg*
- *Requests now sent to CDN’s infrastructure…*

55

55

Hosting: Multiple Sites Per Machine

- Multiple Web sites on a single machine
  - Hosting company runs the Web server on behalf of multiple sites (*e.g.*, www.foo.com and www.bar.com)
- Problem: `GET /index.html`
  - `www.foo.com/index.html` or `www.bar.com/index.html`?
- Solutions:
  - Multiple server processes on the same machine
    - Have a separate IP address (or port) for each server
  - Include site name in HTTP request
    - Single Web server process with a single IP address
    - Client includes "Host" header (*e.g.*, `Host: www.foo.com`)
    - *Required header* with HTTP/1.1

56

56

Hosting: Multiple Machines Per Site

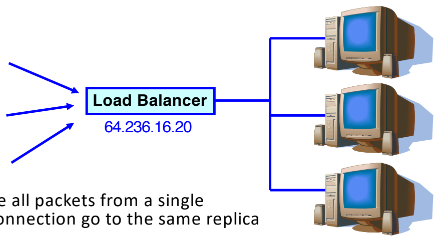
- Replicate popular Web site across many machines
  - Helps to handle the load
  - Places content closer to clients
- Helps when content isn't cacheable
- Problem: Want to direct client to particular replica
  - Balance load across server replicas
  - Pair clients with nearby servers

57

57

Multi-Hosting at Single Location

- Single IP address, multiple machines
  - Run multiple machines behind a single IP address



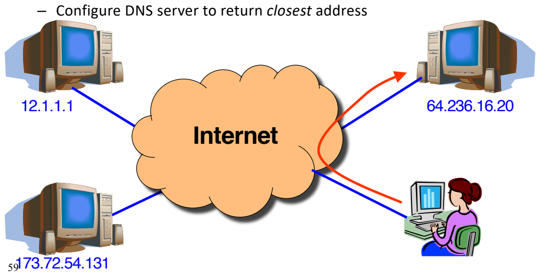
  - Ensure all packets from a single TCP connection go to the same replica

58

58

Multi-Hosting at Several Locations

- Multiple addresses, multiple machines
  - Same name but different addresses for all of the replicas
  - Configure DNS server to return *closest* address



59

59

CDN examples round-up

- CDN using DNS
  DNS has information on loading/distribution/location
- CDN using anycast
  same address from DNS name but local routes
- CDN based on rewriting HTML URLs
  (akami example just covered – akami uses DNS too)

60

60

After HTTP/1.1

SPDY (speedy) and its moral successor HTTP/2

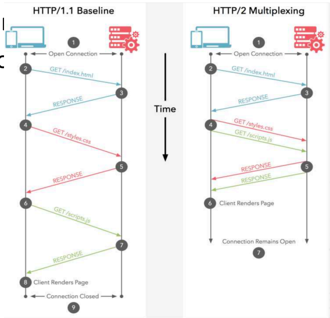
- Binary protocol
  - More efficient to parse
  - More compact on the wire
  - Much less error prone as compared
  - to textual protocols

61

61

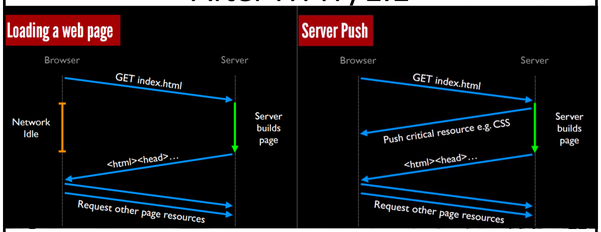
After HTTP/1.1

SPDY (speedy) a
- Binary protoco
- Multiplexing
  - Interleaved



62

After HTTP/1.1



- Server Push
  - Proactively push stuff to client that it will need

63

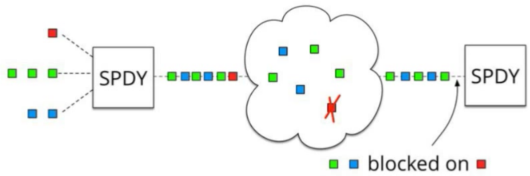
After HTTP/1.1

SPDY (speedy) and its moral successor HTTP/2
- Binary protocol
- Multiplexing
- Priority control over Frames
- Header Compression
- Server Push

64

SPDY

- SPDY + HTTP/2: One single TCP connection instead of multiple
- Downside: Head of line blocking
- In TCP, packets need to be processed in



65

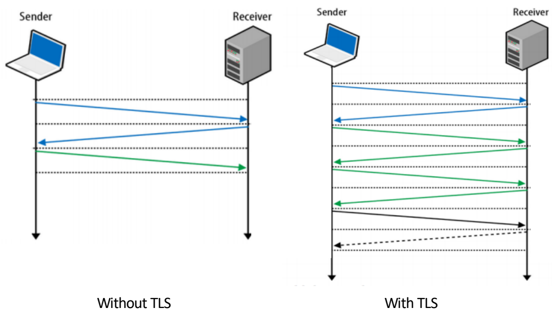
Add QUIC and stir…
Quick UDP Internet Connections

Objective: Combine speed of UDP protocol with TCP's reliability
- Very hard to make changes to TCP
- *Faster to implement new protocol on top of UDP*
- Roll out features in TCP if they prove theory

QUIC:
- Reliable transport over UDP (seriously)
- Uses FEC
- Default crypto
- Restartable connections

66

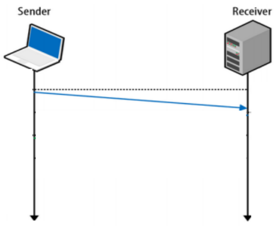
3-Way Handshake



67

UDP

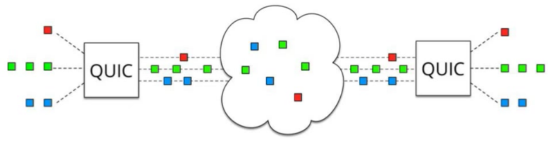
- Fire and forget
  - Less time spent to validate packets
  - Downside - no reliability, has to be built on top of UDP



68

QUIC

- UDP does NOT depend on order of arriving packets
- Lost packets will only impact an individual resource, e.g., CSS or JS file.
- QUIC is combining best parts of HTTP/2 over UDP:
  - Multiplexing on top of non-blocking transport protocol



69

QUIC – more than just UDP

- QUIC outshines TCP under poor network conditions, shaving a full second off the Google Search page load time for the slowest 1% of connections.

- These benefits are even more apparent for video services like YouTube. Users report 30% fewer rebuffers when watching videos over QUIC.

70

70

SIP – Session Initiation Protocol

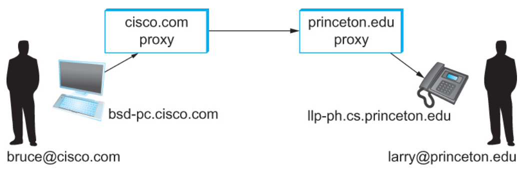
Session?

Anyone smell an OSI / ISO standards document burning?

71

71

SIP - VoIP



Establishing communication through SIP proxies.

72

72

SIP?

- SIP – bringing the fun/complexity of telephony to the Internet
  - User location
  - User availability
  - User capabilities
  - Session setup
  - Session management
    - (e.g. "call forwarding")

73

73

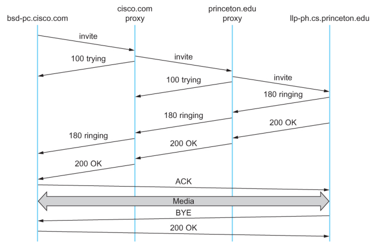
H.323 – ITU

- Why have one standard when there are at least two….
- The full H.323 is hundreds of pages
  - The protocol is known for its complexity – an ITU hallmark
- SIP is not much better
  - IETF grew up and became the ITU….

74

74

Multimedia Applications

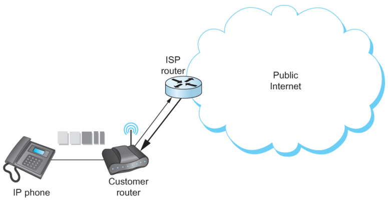


Message flow for a basic SIP session

75

75

The (still?) missing piece:
Resource Allocation for Multimedia Applications



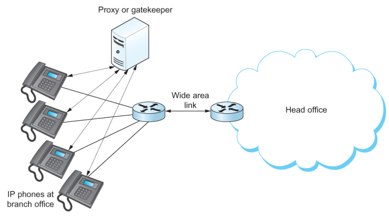
I can 'differentiate' VoIP from data but…
I can only control data going into the Internet

76

76

Multimedia Applications

- Resource Allocation for Multimedia Applications



Admission control using session control protocol.

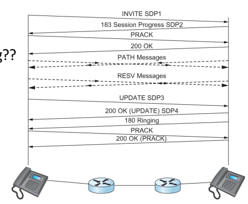
77

77

Resource Allocation for Multimedia Applications

Coming soon… ~~1995~~ ~~2000~~ ~~2010~~ who are we kidding??

Co-ordination of SIP signaling and resource reservation.



So where does it happen?
Inside single institutions or *domains of control…..*
(Universities, Hospitals, big corp…)

What about my aDSL/CABLE/etc it combines voice and data?
Phone company **controls** the multiplexing on the line
and throughout their own network too……

78

78

P2P – efficient network use that annoys the ISP

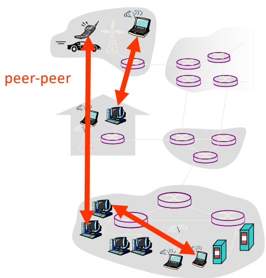
79

79

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

- Three topics:
  - File distribution
  - Searching for information
  - Case Study: Skype

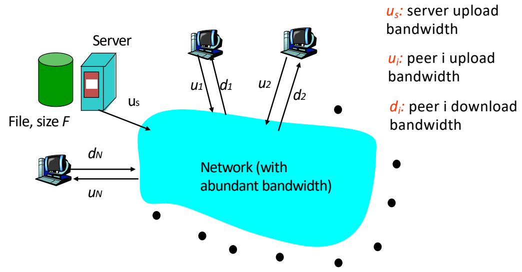


80

80

File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to *N* *peers*?

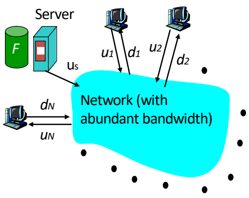


81

81

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download



Time to distribute *F* to *N* clients using client/server approach $= d_{cs} = \max_i \{ NF/u_s, F/\min(d_i) \}$

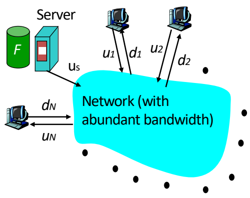
increases linearly in N (for large N)

82

82

File distribution time: P2P

- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \sum u_i$



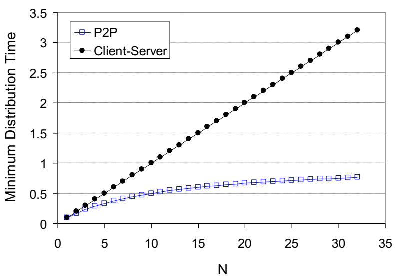
$d_{P2P} = \max_i \{ F/u_s, F/\min(d_i), NF/(u_s + \sum u_i) \}$

83

83

Server-client vs. P2P: example

Client upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



84

84

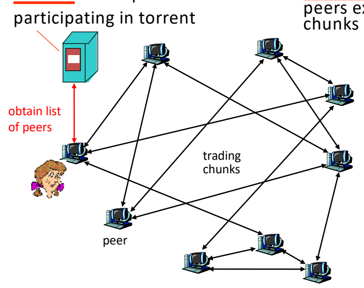
File distribution: BitTorrent*

*rather old BitTorrent

- P2P file distribution

tracker: tracks peers participating in torrent

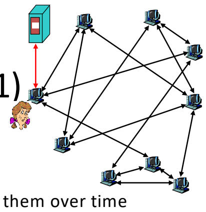
torrent: group of peers exchanging chunks of a file



85

85

BitTorrent (1)



- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

86

86

BitTorrent (2)

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

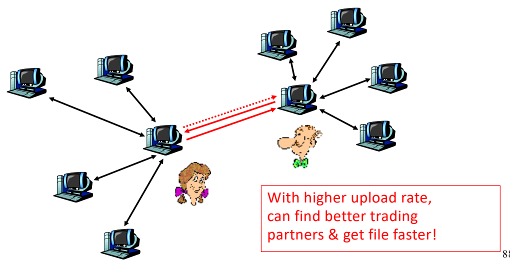
- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

87

87

BitTorrent: Tit-for-tat

(1) Alice "optimistically unchokes" Bob
(2) Alice becomes one of Bob's top-four providers; Bob reciprocates
(3) Bob becomes one of Alice's top-four providers



88

88

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

89

89

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

90

90

DHT Identifiers

- Assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
  - eg, key = h("Game of Thrones season 29")
  - This is why they call it a distributed "hash" table

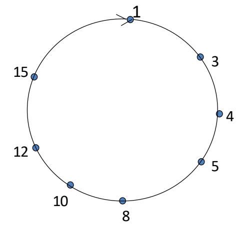
91

How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention in lecture: closest is the immediate successor of the key.
- Ex: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

92

Circular DHT (1)

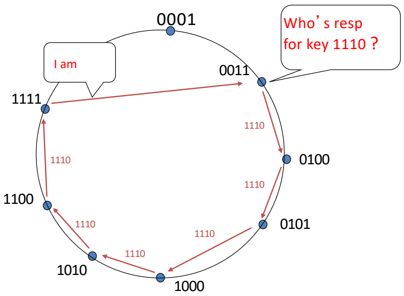


- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network"

93

Circle DHT (2)

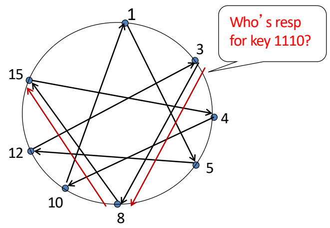
O(N) messages on avg to resolve query, when there are N peers



Define closest as closest successor

94

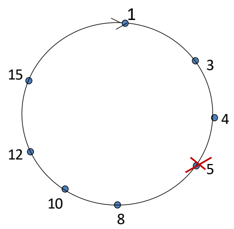
Circular DHT with Shortcuts



- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Reduced from 6 to 2 messages.
- Possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

95

Peer Churn



- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- What if peer 13 wants to join?

96

Hybrid of client-server and P2P

Skype
- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

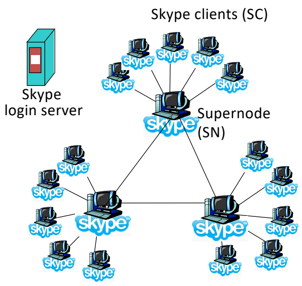
Instant messaging
- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

97

97

P2P Case study: Skype (pre-Microsoft)

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes
- Index maps usernames to IP addresses; distributed over SNs

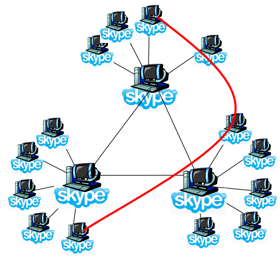


98

98

Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's supernodes, a relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



99

99

Summary.

- Apps need protocols too
- We covered examples from
  - Traditional Applications (web)
  - Scaling and Speeding the web (CDN/Cache tricks)
- Infrastructure Services (DNS)
  - Cache and Hierarchy
- Multimedia Applications (SIP)
  - Extremely hard to do better than worst-effort
- P2P Network examples

100

100