Transport Protocols & DNS

EE 122: Intro to Communication Networks
Fall 2006 (MW 4-5:30 in Donner 155)

Vern Paxson
TAs: Dilip Antony Joseph and Sukun Kim

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, and colleagues at Princeton and UC Berkeley

Announcements

• We’re soliciting feedback
  – What’s not working? What’s working well?

• Send via email …

• … or if you want to be anonymous, put a note in my box in Soda, 3rd floor (near dept. admins)
Goals for Today’s Lecture, Part 1

• Principles underlying transport-layer services
  – (De)multiplexing via port numbers
  – Reliable delivery
  – Performance issues:
    • Stop-and-Wait vs. Sliding Window
    • Flow control

• Service models of Internet transport protocols
  – User Datagram Protocol (UDP)
  – Transmission Control Protocol (TCP)

• Not a goal for today: details of TCP’s operation

Goals of Today’s Lecture, Part 2

• Concepts & principles underlying Domain Name System (DNS)
  – Indirection: names in place of addresses
  – Hierarchy: in names, addresses, and servers
  – Caching: of mappings from names to/from addresses

• Inner workings of DNS
  – DNS resolvers and servers
  – Iterative and recursive queries
  – TTL-based caching
  – Use of the `dig` utility
Role of Transport Layer

- **Application layer**
  - Communication for specific applications
  - E.g., HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), Network News Transfer Protocol (NNTP)

- **Transport layer**
  - Communication between *processes* (e.g., socket)
  - Relies on network layer; serves the application layer
  - E.g., TCP and UDP

- **Network layer**
  - Logical communication between nodes
  - Hides details of the link technology
  - E.g., IP

Transport Protocols

- Provide *logical communication* between application processes running on different hosts
- Run on end hosts
  - Sender: breaks application messages into *segments*, and passes to network layer
  - Receiver: reassembles segments into messages, passes to application layer
- Multiple transport protocol available to applications
  - Internet: TCP and UDP (mainly)
Internet Transport Protocols

- Datagram messaging service (UDP)
  - No-frills extension of “best-effort” IP

- Reliable, in-order delivery (TCP)
  - Connection set-up
  - Discarding of corrupted packets
  - Retransmission of lost packets
  - Flow control
  - Congestion control

- Services not available
  - Delay guarantees
  - Bandwidth guarantees
  - Sessions that survive change-of-IP-address

Multiplexing and Demultiplexing

- Host receives IP datagrams
  - Each datagram has source and destination IP address,
  - Each datagram carries one transport-layer segment
  - Each segment has source and destination port number

- Host uses IP addresses and port numbers to direct the segment to appropriate socket

TCP/UDP segment format

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>other header fields</td>
<td></td>
</tr>
<tr>
<td>application data (message)</td>
<td></td>
</tr>
</tbody>
</table>

32 bits
Unreliable Message Delivery Service

• Lightweight communication between processes
  – Avoid overhead and delays of ordered, reliable delivery
  – Send messages to and receive them from a socket

• User Datagram Protocol (UDP; RFC 768 - 1980!)
  – IP plus port numbers to support (de)multiplexing
  – Optional error checking on the packet contents
    • (checksum field = 0 means “don’t verify checksum”)

Why Would Anyone Use UDP?

• Finer control over what data is sent and when
  – As soon as an application process writes into the socket
  – … UDP will package the data and send the packet

• No delay for connection establishment
  – UDP just blasts away without any formal preliminaries
  – … which avoids introducing any unnecessary delays

• No connection state
  – No allocation of buffers, sequence #s, timers …
  – … making it easier to handle many active clients at once

• Small packet header overhead
  – UDP header is only 8 bytes
Popular Applications That Use UDP

- Multimedia **streaming**
  - Retransmitting lost/corrupted packets often pointless - by the time the packet is retransmitted, it’s too late
  - E.g., telephone calls, video conferencing, gaming

- Simple query protocols like Domain Name System
  - Connection establishment overhead would double cost
  - Easier to have application retransmit if needed

Transmission Control Protocol (TCP)

- Connection oriented
  - Explicit set-up and tear-down of TCP session

- Stream-of-bytes service
  - Sends and receives a stream of bytes, not messages

- Congestion control
  - Dynamic adaptation to network path’s capacity

- Reliable, in-order delivery
  - TCP tries **very** hard to ensure byte stream (eventually) arrives intact
    - In the presence of corruption and loss

- Flow control
  - Ensure that sender doesn’t overwhelm receiver
Reliable Delivery

• How do we design for reliable delivery?
  – One possible model: how does it work talking on your cell phone?

• Positive acknowledgment (“Ack”)
  – Explicit confirmation by receiver
  – TCP acknowledgments are cumulative (“I’ve received everything up through sequence #N”)
    • With an option for acknowledging individual segments (“SACK”)

• Negative acknowledgment (“Nack”)
  – “I’m missing the following: …”
  – How might the receiver tell something’s missing? Can they always do this?
  – (Only used by TCP in implicit fashion - “fast retransmit”)

Reliable Delivery, con’t

• Timeout
  – If haven’t heard anything from receiver, send again
  – Problem: for how long do you wait?
    • TCP uses function of estimated RTT
  – Problem: what if no Ack for retransmission?
    • TCP (and other schemes) employs exponential backoff
    • Double timer up to maximum - tapers off load during congestion

• A very different approach to reliability: send redundant data
  – Cell phone analogy: “Meet me at 3PM - repeat 3PM”
  – Forward error correction
  – Recovers from lost data nearly immediately!
  – But: only can cope with a limited degree of loss
  – And: adds load to the network
TCP Support for Reliable Delivery

- Sequence numbers
  - Used to detect missing data
  - ... and for putting the data back in order

- Checksum
  - Used to detect corrupted data at the receiver
  - ...leading the receiver to drop the packet
  - No error signal sent - recovery via normal retransmission

- Retransmission
  - Sender retransmits lost or corrupted data
  - Timeout based on estimates of round-trip time (RTT)
  - Fast retransmit algorithm for rapid retransmission

Efficient Transport Reliability
Automatic Repeat reQuest (ARQ)

• Automatic Repeat Request
  – Receiver sends acknowledgment (ACK) when it receives packet
  – Sender waits for ACK and times out if does not arrive within some time period

• Simplest ARQ protocol
  – Stop and Wait
  – Send a packet, stop and wait until ACK arrives

How Fast Can Stop-and-Wait Go?

• Suppose we’re sending from UCB to New York:
  – Bandwidth = 1 Mbps (megabits/sec)
  – RTT = 100 msec
  – Maximum Transmission Unit (MTU) = 1500 B = 12,000 b
  – No other load on the path and no packet loss

• What (approximately) is the fastest we can transmit using Stop-and-Wait?

• How about if Bandwidth = 1 Gbps?
Allowing Multiple Packets in Flight

• “In Flight” = “Unacknowledged”

• Sender-side issue: how many packets (bytes)?

• Receiver-side issue: how much buffer for data that’s “above a sequence hole”?  
  – I.e., data that can’t be delivered since previous data is missing
  – Assumes service model is in-order delivery (like TCP)

Sliding Window

• Allow a larger amount of data “in flight”  
  – Allow sender to get ahead of the receiver  
  – … though not too far ahead
**Sliding Window, con’t**

- Both sender & receiver maintain a **window** that governs amount of data in flight (sender) or not-yet-delivered (receiver)

- **Left edge** of window:
  - Sender: beginning of **unacknowledged** data
  - Receiver: beginning of **undelivered** data

- **Right edge** of window: left edge + size
  - Sender: size = maximum amount of data in flight
    - Determines **rate**
    - Sender must have at least this much buffer (maybe more)
  - Receiver: size = maximum amount of undelivered data
    - Receiver has this much buffer

**Sliding Window, con’t**

- Sender: window **advances** when new data ack’d
- Receiver: window advances as receiving process **consumes** data

- What happens if Send Window Size (SWS) exceeds Receive Window Size (RWS)?
- Receiver **advertises** RWS to the sender
  - Sender agrees not to exceed this amount
  - Given this, when can SWS ≠ RWS?

![Window Size Diagram](image)
Performance with Sliding Window

• Given previous UCB ↔ New York 1 Mbps path
  + Sender window = 100 Kb = 12.5 KB

• How fast can we transmit?

• What about with 12.5 KB window & 1 Gbps path?

• Window required to fully utilize path:
  • Bandwidth-delay product (or “delay-bandwidth product”)
  • 1 Gbps * 100 msec = 100 Mb = 12.5 MB
  • Can picture as path’s storage capacity
  • Note: large window = many packets in flight

5 Minute Break

Questions Before We Proceed?
Domain Name System (DNS)

Host Names vs. IP addresses

• Host names
  – Mnemonic name appreciated by humans
  – Variable length, full alphabet of characters
  – Provide little (if any) information about location
  – Examples: www.cnn.com and bbc.co.uk

• IP addresses
  – Numerical address appreciated by routers
  – Fixed length, binary number
  – Hierarchical, related to host location
  – Examples: 64.236.16.20 and 212.58.228.155
Separating Naming and Addressing

• Names are easier to remember
  – www.cnn.com vs. 64.236.16.20

• Addresses can change underneath
  – Move www.cnn.com to 64.236.16.20
  – E.g., renumbering when changing providers

• Name could map to multiple IP addresses
  – www.cnn.com to multiple (8) replicas of the Web site

• Map to different addresses* in different places
  – Address of a nearby copy of the Web site
  – E.g., to reduce latency, or return different content

• Multiple names for the same address
  – E.g., aliases like www.cnn.com and cnn.com

Scalable (Name ↔ Address) Mappings

• Originally: per-host file
  – Flat namespace
  – /etc/hosts (what is this on your computer today?)
  – SRI kept master copy
  – Downloaded regularly

• Single server doesn’t scale
  – Traffic implosion (lookups & updates)
  – Single point of failure
  – Amazing politics

Need a distributed, hierarchical collection of servers
Domain Name System (DNS)

- Properties of DNS
  - Hierarchical name space divided into zones
  - Zones distributed over collection of DNS servers

- Hierarchy of DNS servers
  - Root (hardwired into other servers)
  - Top-level domain (TLD) servers
  - Authoritative DNS servers

- Performing the translations
  - Local DNS servers
  - Resolver software

Distributed Hierarchical Database

```
com  edu  • • •  org  ac  • • •  uk  zw  arpa

bar
  west  east

foo  my

my.east.bar.edu

Top-Level Domains (TLDs)

ac  cam  usr

usr.cam.ac.uk

12.34.56.0/24
```
DNS Root

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

DNS Root Servers

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

A Verisign, Dulles, VA
B USC-ISI Marina del Rey, CA
C Cogent, Herndon, VA
D U Maryland College Park, MD
E NASA Mt View, CA
F Internet Software Consortium, Palo Alto, CA
G US DoD Vienna, VA
H ARL Aberdeen, MD
I Autonomica, Stockholm
J Verisign
K RIPE London
L ICANN Los Angeles, CA
M WIDE Tokyo
DNS Root Servers

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

TLD and Authoritative DNS Servers

- Top-level domain (TLD) servers
  - Generic domains (e.g., com, org, edu)
  - Country domains (e.g., uk, fr, cn, jp)
  - Special domains (e.g., arpa)
  - Typically managed professionally
    • Network Solutions maintains servers for "com"
    • Educause maintains servers for "edu"
- Authoritative DNS servers
  - Provide public records for hosts at an organization
    • Private records may differ, though not part of original design’s intent
  - For the organization’s servers (e.g., Web and mail)
  - Can be maintained locally or by a service provider
Using DNS

- 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

- Server application
  - Extract client IP address from socket
  - Optional `gethostbyaddr()` to translate into name

Example

Host at `cis.poly.edu` wants IP address for `gaia.cs.umass.edu`

1. Request from `cis.poly.edu`
2. Forwarded to local DNS server `dns.poly.edu`
3. Forwarded to root DNS server
4. Forwarded to TLD DNS server `com`
5. Forwarded to authoritative DNS server `dns.cs.umass.edu`
6. Response from authoritative DNS server
7. Forwarded back to local DNS server `dns.poly.edu`
8. Response to `cis.poly.edu`
Recursive vs. Iterative Queries

- **Recursive** query
  - Ask server to get answer for you
  - E.g., request 1 and response 8

- **Iterative** query
  - Ask server who to ask next
  - E.g., all other request-response pairs

Reverse Mapping (Address → Host)

- How do we go the other direction, from an IP address to the corresponding hostname?
- Addresses already have natural “quad” hierarchy:
  - 12.34.56.78

- But: quad notation has most-sig. hierarchy element on left, while www.cnn.com has it on the right
- Idea: reverse the quads = 78.56.34.12 …
  - … and look that up in the DNS

- Under what TLD?
  - Convention: in-addrarpa
  - So lookup is for 78.56.34.12.in-addr.arpa
Distributed Hierarchical Database

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.cnn.com) 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
Negative Caching

• Remember things that don’t work
  – Misspellings like www.cnn.comm and www.cnnn.com
  – 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

DNS Resource Records

**DNS:** distributed DB storing resource records (RR)

**RR format:** (name, value, type, ttl)

• Type=A
  – *name* is hostname
  – *value* is IP address

• Type=NS
  – *name* is domain (e.g. foo.com)
  – *value* is hostname of authoritative name server for this domain

• Type=PTR
  – *name* is reversed IP quads
    • E.g. 78.56.34.12.in-addr.arpa
  – *value* is corresponding hostname

• Type=CNAME
  – *name* is alias name for some “canonical” name
    E.g., www.cs.mit.edu is really eecsweb.mit.edu
  – *value* is canonical name

• Type=MX
  – *value* is name of mailserver associated with *name*
  – Also includes a weight/preference
DNS Protocol

**DNS protocol:** *query* and *reply* messages, both with *same message format*

Message header
- **Identification:** 16 bit # for query, reply to query uses same #
- **Flags:**
  - Query or reply
  - Recursion desired
  - Recursion available
  - Reply is authoritative

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>identification</td>
<td>flags</td>
</tr>
<tr>
<td>number of questions</td>
<td>number of answer RRs</td>
</tr>
<tr>
<td>number of authority RRs</td>
<td>number of additional RRs</td>
</tr>
<tr>
<td>questions (variable number of questions)</td>
<td></td>
</tr>
<tr>
<td>answers (variable number of resource records)</td>
<td></td>
</tr>
<tr>
<td>authority (variable number of resource records)</td>
<td></td>
</tr>
<tr>
<td>additional information (variable number of resource records)</td>
<td></td>
</tr>
</tbody>
</table>

Interactive DNS lookups using *dig*

- **dig** program on Unix
  - Allows querying of DNS system
  - Dumps each field in DNS responses
  - By default, executes recursive queries
    - Disable via `+norecurse` so that operates one step at a time
unix> dig +norecurse @a.root-servers.net www.cnn.com

;; DiG 9.2.2 <<>> +norecurse @a.root-servers.net www.cnn.com
;; global options:  printcmd
got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21041
;; flags: qr QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 14

;; QUESTION SECTION:
www.cnn.com. IN A

;; AUTHORITY SECTION:
com. 172800 IN NS A.GTLD-SERVERS.NET.
com. 172800 IN NS A.GTLD-SERVERS.NET.
com. 172800 IN NS G.GTLD-SERVERS.NET.
com. 172800 IN NS C.GTLD-SERVERS.NET.
com. 172800 IN NS I.GTLD-SERVERS.NET.
com. 172800 IN NS B.GTLD-SERVERS.NET.
com. 172800 IN NS D.GTLD-SERVERS.NET.
com. 172800 IN NS L.GTLD-SERVERS.NET.
com. 172800 IN NS F.GTLD-SERVERS.NET.
com. 172800 IN NS J.GTLD-SERVERS.NET.
com. 172800 IN NS K.GTLD-SERVERS.NET.
com. 172800 IN NS E.GTLD-SERVERS.NET.

;; ADDITIONAL SECTION:
A.GTLD-SERVERS.NET.  172800 IN A  192.5.6.30
A.GTLD-SERVERS.NET.  172800 IN A  192.42.93.30
G.GTLD-SERVERS.NET.  172800 IN A  192.54.112.30
C.GTLD-SERVERS.NET.  172800 IN A  192.26.92.30
I.GTLD-SERVERS.NET.  172800 IN A  192.43.172.30
B.GTLD-SERVERS.NET.  172800 IN A  192.33.14.30
L.GTLD-SERVERS.NET.  172800 IN A  192.31.80.30
F.GTLD-SERVERS.NET.  172800 IN A  192.35.51.30
J.GTLD-SERVERS.NET.  172800 IN A  192.48.79.30
K.GTLD-SERVERS.NET.  172800 IN A  192.52.178.30
E.GTLD-SERVERS.NET.  172800 IN A  192.12.94.30

;; Query time: 117 msec
;; SERVER: 198.41.0.4#53(a.root-servers.net)
;; WHEN: Mon Sep 25 11:13:15 2006
;; MSG SIZE  rcvd: 501
dig +norecurse @g.gtld-servers.net www.cnn.com

; <<>> DiG 9.2.4 <<>> +norecurse @g.gtld-servers.net www.cnn.com
; global options: printcmd
; Got answer:
; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 74
; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 4

; QUESTION SECTION:

; AUTHORITY SECTION:
cnn.com.
  172800 IN NS twdns-01.ns.aol.com.
cnn.com.
  172800 IN NS twdns-02.ns.aol.com.
cnn.com.
  172800 IN NS twdns-03.ns.aol.com.
cnn.com.
  172800 IN NS twdns-04.ns.aol.com.

; ADDITIONAL SECTION:
twdns-01.ns.aol.com.
  172800 IN A 149.174.213.151
twdns-02.ns.aol.com.
  172800 IN A 152.163.239.216
twdns-03.ns.aol.com.
  172800 IN A 207.200.73.85
twdns-04.ns.aol.com.
  172800 IN A 64.12.147.120

dig +norecurse @twdns-01.ns.aol.com www.cnn.com

... 

; QUESTION SECTION:

; ANSWER SECTION:
  300 IN CNAME cnn.com.
cnn.com.
  300 IN A 64.236.24.12
cnn.com.
  300 IN A 64.236.24.20
cnn.com.
  300 IN A 64.236.24.28
cnn.com.
  300 IN A 64.236.29.120
cnn.com.
  300 IN A 64.236.16.20
cnn.com.
  300 IN A 64.236.16.52
cnn.com.
  300 IN A 64.236.16.84
cnn.com.
  300 IN A 64.236.16.116

; AUTHORITY SECTION:
cnn.com.
  600 IN NS twdns-02.ns.aol.com.
cnn.com.
  600 IN NS twdns-03.ns.aol.com.
cnn.com.
  600 IN NS twdns-04.ns.aol.com.
cnn.com.
  600 IN NS twdns-01.ns.aol.com.
Reliability

• DNS servers are replicated
  – 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

Inserting Resource Records into DNS

• Example: just created startup “FooBar”

• Register foobar.com at Network Solutions (say)
  – Provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  – Registrar inserts two RR pairs into the com TLD server:
    • (foobar.com, dns1.foobar.com, NS)
    • (dns1.foobar.com, 212.212.212.1, A)

• Put in authoritative server dns1.foobar.com
  – Type A record for www.foobar.com
  – Type MX record for foobar.com
Summary

- Transport protocols
  - Multiplexing and demultiplexing via port numbers
  - UDP gives simple datagram service
  - TCP gives reliable byte-stream service
  - Reliability immediately raises performance issues
    - Stop-and-Wait vs. Sliding Window

- Domain Name System (DNS)
  - Distributed, hierarchical database
  - Distributed collection of servers
  - Caching to improve performance
  - Examine using \texttt{dig} utility

Next Week

- Security analysis of DNS
- Application protocols:
  - File transfer: FTP
  - Email: SMTP
- Reading:
  - Sections 9.2.1 and skim RFC 959
- Project #1 due this Thursday by 11PM PDT
- Reminder: feedback solicited