ADDRESSING, TCP, AND RELIABLE PACKET DELIVERY

George Porter
Jan 11, 2019

ATTRIBUTION

• These slides are released under an Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0) Creative Commons license
• These slides incorporate material from:
  • Alex C. Snoeren, UC San Diego
  • Michael Freedman and Kyle Jamieson, Princeton University
  • Internet Society
ANNOUNCEMENTS
Peterson & Davie 1.3 (network architectures), 1.4, 1.5, 2.5, 3.2
Class Q&A forum (via Google) now available (link off course web page)
Practice problem set (ungraded) going out Monday 1/14
Project 1 going out Monday 1/14

Outline

1. Addressing (con’t)
2. TCP and sockets
3. Reliable transmission
4. Sockets API
CIDR

- Classless Inter-Domain Routing (1993)
- Networks described by variable-length prefix and length
- Allows arbitrary allocation between network and host address

<table>
<thead>
<tr>
<th>Network</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>Mask=# significant bits representing prefix</td>
</tr>
</tbody>
</table>

- e.g. 10.95.1.2 contained within 10.0.0.0/8:
  - 10.0.0.0 is network and remainder (95.1.2) is host

- Pro: Finer grained allocation; aggregation
- Con: More expensive lookup: longest prefix match

ADDRESS AGGREGATION EXAMPLE

Advertise 212.56.132.0/22

Customer A
212.56.132.0/24

Customer B
212.56.133.0/24

Customer C
212.56.134.0/24

Customer D
212.56.135.0/24

Common Prefix: 11010100.00111000.10000100.00000000

^^^^^ 22 bits in common  ^^^^^
FORWARDING TABLE EXAMPLE (R2)

• Packet to 10.1.1.6
• Matches 10.1.0.0/23

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>loopback</td>
</tr>
<tr>
<td>Default or 0/0</td>
<td>10.1.16.1</td>
</tr>
<tr>
<td>10.1.8.0/24</td>
<td>interface1</td>
</tr>
<tr>
<td>10.1.2.0/23</td>
<td>interface2</td>
</tr>
<tr>
<td>10.1.0.0/23</td>
<td>10.1.2.2</td>
</tr>
<tr>
<td>10.1.16.0/24</td>
<td>interface3</td>
</tr>
</tbody>
</table>

FORWARDING TABLE EXAMPLE 2

• Packet to 10.1.1.6
• Matches 10.1.1.4/30
  • Longest prefix match

Routing table at R1

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>loopback</td>
</tr>
<tr>
<td>Default or 0/0</td>
<td>10.1.2.1</td>
</tr>
<tr>
<td>10.1.0.0/24</td>
<td>interface1</td>
</tr>
<tr>
<td>10.1.1.0/24</td>
<td>interface2</td>
</tr>
<tr>
<td>10.1.2.0/23</td>
<td>interface3</td>
</tr>
<tr>
<td>10.1.1.4/30</td>
<td>10.1.1.101</td>
</tr>
</tbody>
</table>
### FORWARDING TABLE EXAMPLE 3

- Packet to 10.1.1.6
- Direct route
  - Longest prefix match

#### Routing table at H1

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>loopback</td>
</tr>
<tr>
<td>Default or 0/0</td>
<td>10.1.1.1</td>
</tr>
<tr>
<td>10.1.1.0/24</td>
<td>interface1</td>
</tr>
<tr>
<td>10.1.1.4/30</td>
<td>interface2</td>
</tr>
</tbody>
</table>

### ASSIGNING ADDRESSES VIA DHCP

- DHCP server is responsible for providing configuration information to hosts
- There is at least one DHCP server for an administrative domain
- DHCP server maintains a pool of available addresses
**DHCP IN ACTION**

- Newly booted or attached host sends DHCPDISCOVER message to a special IP address (255.255.255.255)
- DHCP relay agent unicasts the message to DHCP server and waits for the response

![DHCP Flow Diagram](image)

**DNS HOSTNAME VERSUS IP ADDRESS**

- **DNS host name** (e.g. www.cs.ucsd.edu)
  - Mnemonic name appreciated by humans
  - Variable length, full alphabet of characters
  - Provides little (if any) information about location
- **IP address** (e.g. 128.112.136.35)
  - Numerical address appreciated by routers
  - Fixed length, decimal number
  - Hierarchical address space, related to host location
MAPPING NAMES TO ADDRESSES

GETADDRINFO(3)  Linux Programmer's Manual  GETADDRINFO(3)

NAME
getaddrinfo, freeaddrinfo, gai_strerror — network address and service
translation

SYNOPSIS
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *node, const char *service,
               const struct addrinfo *hints,
               struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

const char *gai_strerror(int errcode);

LINKED LIST OF ‘ADDRINFO’ STRUCTS

struct addrinfo {
    int      ai_flags;
    int      ai_family;
    int      ai_socktype;
    int      ai_protocol;
    socklen_t ai_addrlen;
    struct sockaddr *ai_addr;
    char     *ai_canonname;
    struct addrinfo *ai_next;
};

• We will cover the implementation of DNS during week 3
Outline

1. Packet switching
2. Addressing
3. Performance
4. TCP and sockets
5. Reliable transmission

PERFORMANCE METRICS

- Bandwidth: number of bits transmitted per unit of time
- Latency = Propagation + Transmit + Queue
  - Propagation = Distance/SpeedOfLight(*)
  - Transmit = 1 bit/Bandwidth
- Overhead
  - # secs for CPU to put message on wire
- Error rate
  - Probability P that message will not arrive intact

* In that particular medium
BANDWIDTH VS. LATENCY

1 Byte Object

<table>
<thead>
<tr>
<th></th>
<th>Latency: 1 ms</th>
<th>Latency: 100 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth: 1 Mbps</td>
<td>1,008 μs</td>
<td>100,008 μs</td>
</tr>
<tr>
<td>Bandwidth: 100 Mbps</td>
<td>1,000 μs</td>
<td>100,000 μs</td>
</tr>
</tbody>
</table>

10 MB Object

<table>
<thead>
<tr>
<th></th>
<th>Latency: 1 ms</th>
<th>Latency: 100 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth: 1 Mbps</td>
<td>80.001 s</td>
<td>80.1 s</td>
</tr>
<tr>
<td>Bandwidth: 100 Mbps</td>
<td>.801 s</td>
<td>.9 s</td>
</tr>
</tbody>
</table>

TERMINOLOGY STYLE

- Mega versus Mega, Kilo versus Kilo
- Computer architecture: Mega $\rightarrow 2^{20}$, Kilo $\rightarrow 2^{10}$
- Computer networks: Mega $\rightarrow 10^6$, Kilo $\rightarrow 10^3$
- Mbps versus MBps
  - Networks: typically megabits per second
  - Architecture: typically megabytes per second
- Bandwidth versus throughput
  - Bandwidth: available over link
  - Throughput: available to application
1. Addressing (con’t)

2. TCP and sockets

3. Reliable transmission

4. Sockets API

**LAYERING: A MODULAR APPROACH TO COMM**

- Sub-divide responsibilities
  - Each layer relies on services from layer below
  - Each layer exports services to layer above

- Interface between layers defines interaction
  - Hides implementation details (encapsulation)
  - Layers can change without disturbing other layers (modularity)

- Interface among peers in a layer is a protocol
  - If peers speak same protocol, they can interoperate
LAYERED ARCHITECTURES

Request/Response
downcall

Layer N

Layer N-1

Layer 2

Layer 1

One-way call

Layer N

Layer N-1

Layer N-2

Layer N-3

Layer N

Layer N-1

Layer N-2

Handle

Upcall

SERVICE AND PROTOCOL INTERFACES

Party A

Layer N

Layer N-1

Protocol

Interface

Service

Party B

Layer N

Layer N-1
WHAT ARE PROTOCOLS?

- Explicit and implicit conventions for how to communicate
- Not for what is communicated

WHERE DO PROTOCOLS COME FROM?

- Standards bodies
  - IETF: Internet Engineering Task Force
  - ISO: International Standards Organization
- Community efforts
  - “Request for comments”
- Corporations/industry
  - RealAudio™, Call of Duty multiplayer, Skype
HOW ARE PROTOCOLS SPECIFIED?

Prose/BNF

Three rules show a field metasyntax, without regard for the particular type or internal syntax. Their purpose is to permit detection of fields; also, they prevent to higher-level parsers an image of each field as fitting on one line.

```
field  =  field-name  "."  [  field-body  ]
field-name  =  1*any OCTET, excluding (\0x00, \0x0A), and ";"
field-body  =  \0x01 UDP-er field-body
field-body-contexts  =< the ASCII characters making up the field-body, as defined in the following sections, and consisting of combinations of one, quoted-string, and special tokens, or also consisting of text>
```

State transition diagrams

Message Sequence Diagram

Packet formats

ROLE OF LAYERING IN PROTOCOLS

- Each layer offers useful semantics to layer above
  - IP gets packets to a destination host/server on the Internet (but is unreliable)
  - TCP uses IP to offer reliable, in-order bytestream abstraction
  - TCP useful for file transfer, as well as HTTP/web
INTERNET PROTOCOL SUITE

The Hourglass Model

“Thin Waist”

TRANSMISSION CONTROL PROTOCOL (TCP)

- Remember 1 GB ~ 715,000 packets?
  - Don’t want to keep track of each packet, whether it got there, did it get lost? Did some get reordered??
- TCP offers infinite bytestream abstraction
  - If you put $N$ bytes into TCP connection as sender, those $N$ bytes will arrive to the destination in order, without loss, and without corruption
    - Compelling abstraction for higher-level applications such as web, video, gaming, ...
ENCAPSULATION VIA HEADERS

- Typical web packet:
  - Ethernet Hdr
  - IP Hdr
  - TCP Hdr
  - HTTP Hdr
  - Payload (Web object)

- Notice that layers add overhead
  - Space (headers), effective bandwidth
  - Time (processing headers), latency
Simulated idea: ARQ

- Receiver sends acknowledgments (ACKs)
- Sender “times out” and retransmits if it doesn’t receive them
- Basic approach is generically referred to as Automatic Repeat Request (ARQ)
NOT SO FAST!

- Loss can occur on ACK channel as well
- Sender cannot distinguish data loss from ACK loss
- Sender will retransmit the data frame
- ACK loss—or early timeout—results in duplication
- The receiver thinks the retransmission is new data

SEQUENCE NUMBERS

- Sequence numbers solve this problem
- Receiver can simply ignore duplicate data
- But must still send an ACK!
- Simplest ARQ: Stop-and-wait
- Only one outstanding frame at a time
STOP AND WAIT PERFORMANCE

- Lousy performance if xmit 1 pkt << prop. delay
  - How bad?

- Want to utilize all available bandwidth
  - Need to keep more data “in flight”
  - How much? Called the bandwidth-delay product

PIPELINED TRANSMISSION

- Keep multiple packets “in flight”
  - Allows sender to make efficient use of the link
  - Sequence numbers ensure receiver can distinguish frames
  - Sender buffers outstanding un-acked packets
  - Receiver ACKs the highest consecutive frame received
    - ACKs are cumulative (covers current frame and all previous)
SLIDING WINDOW: SENDER

- Window bounds outstanding unACKed data
  - Implies need for buffering at sender
- "Last" ACK applies to in-order data
- What to do on a timeout?
  - Go-Back-N: resend all unacknowledged data on timeout
  - Selective Repeat: timer per packet, resend as needed

SLIDING WINDOW: RECEIVER

- Receiver buffers too:
  - data may arrive out-of-order
  - or faster than can be consumed
    - Flow control: tell sender how much buffer left at receiver
- Receiver ACK choices:
  - Cumulative, Selective (exempt missing frames), Negative
DECIDING WHEN TO RETRANSMIT

• How do you know when a packet has been lost?
  • Ultimately sender uses timers to decide when to retransmit

• But how long should the timer be?
  • Too long: inefficient (large delays, poor use of bandwidth)
  • Too short: may retransmit unnecessarily (causing extra traffic)

• Right timer is based on the round-trip time (RTT)
  • Which can vary greatly

NEXT WEEK
C/C++ UNIX “sockets” API