Networking Basics

Tyler Bletsch
Duke University

Slides are derived from work by Andrew Hilton (Duke)
Networking

• How do computers communicate?
  • Two computers connected by a direct wire?
    • Relatively straightforward: move bits across wire
  • Internet?
    • Many computers
    • All around the world
    • With other communications going on...
    • And unreliable links
    • And tons of different systems, media, protocols...
• Pretty complicated, so how could we possibly manage it?
  **Abstraction**
  (oh right, the answer to like...everything)
7-layer OSI model

- 7 layer networking stack
- (Theoretically) can change out any layer at a time
Layer 1: Physical Layer

• Defines physical how physical media work
  • Pin layout
  • Voltages
  • Timing Requirements

• Examples:
  • Cat 5 Cable ("Ethernet Cable")
  • Wireless radio signal specifications

• Not much interesting to say here
Layer 2: Data-link Layer

- How to move bits across the wires in a meaningful way
  - Communication between two computers on same physical network
  - May include some error checking

- Example: Ethernet
  - Data transmitted in frames
  - Frame has:
    - **Pre-amble**: used to detect collisions
    - **Header**: source and destination MAC address
    - **Payload**: actual data
    - **CRC check**: detect corrupt data
  - Carrier Sense, Multiple Access, Collision Detect (**CSMACD**)
    - Carrier Sense: listen for if anyone else transmitting
    - Multiple Access: can wire up many computers to it
    - Collision Detect: two transmissions at once? Detect and retry
CSMACD

- Ethernet uses CSMACD for multiple systems on a network
  - Other options, but we won’t go into them
  - Detection of collisions?
    - Pre-amble is fixed pattern
    - Network card senses medium while transmitting
    - Mismatch with expected? Collision
  - Collision happens?
    - Exponential backoff
    - Pick random number of time units
    - Retry
    - Fail again? Pick random number from 2x as big a range
- Analogy: crowded dinner party
  - Try to talk. Someone else talking? Wait. Try again. Fail again? Wait longer
Abstraction: Joys and Limitations

• Joys of abstraction:
  • Can build an Ethernet card without any info about higher layers
  • Will work with all of them

• Limitations:
  • 7-layer model’s abstraction not perfect
  • Ethernet protocol imposes max limit on cable length
    • E.g., layer 2 constrains layer 1
    • This arises from the need to detect collisions before finishing sending
• Reminder where we are so far
Our messages so far

| Preamble | Header | Payload | CRC |

- Header says what network layer protocol the payload is
Layer 3: The Network Layer

• Layer 2 lets computers on same network talk
• Layer 3 lets computers talk across networks
  • Addressing
    • How do we specify what computer to talk to?
  • Routing
    • How do we get from here to there?

• Example: IP protocol
  • IPv4 and IPv6: pretty similar in most core regards
  • Best effort delivery
  • Addressing (IP addresses)
  • Routing
  • Analogy: Mailing a letter
**IP addresses**

- **IPv4 addresses: 32-bit numbers**
  - Could write as decimal or hex...but that’s not how it’s done
  - Instead write as four separate bytes (each expressed in decimal), separated by dots: **dotted decimal notation**
    - IPv4 address: 152.3.34.5
    - As four hex bytes: 98, 03, 22, 05
    - As a 32-bit hex value: 98032205
    - 32-bit binary value: 1001100000000011001000100000101

- **IPv6: 128-bit addresses**
  - More bits needed to address more separate things on the internet
  - Been around for a long time, seeing very slow adoption
  - Addresses represented as eight two-byte groups in hex separated by colons, for example 3001:0db3:0000:0032:0000:3a2e:0330:7384
  - We’ll use IPv4 to explain for now
• Computer 1 wants to send data to Computer 2
  • For now, assume it knows IP address (we’ll see DNS later)
  • Has direct connection to its ISP... but then what?
    • The internet is a big place after all..
Let's zoom in on ISP 1

- ISP has connections to a handful of other places
  - Generally very high bandwidth connections
  - Will send your data (packet) to one of these, but which one?
IP Routing

- IP addresses are hierarchical
  - May not know how to find 74.125.130.105...
  - But know which way to go to get to 74._______
  - Move one step closer
  - Within 74 network, know how to find 74.125
  - Then 74.125.130
  - Then find 74.125.130.105

- Analogy:
  - How do I get to 2200 Mission College Blvd, Santa Clara, CA?
IP Routing

- Analogy:
  - How do I get to 2200 Mission College Blvd, Santa Clara, CA?
  
  - I have no idea, but I can get you to I-40 West
    - Then you can ask someone else when you get to CA

- Once in CA, you ask someone else:
  - “I only know its north of here, so take the 5 North and ask someone else”

- Etc..

- This works because our physical addresses are hierarchical: Country, State, City, Street, Number
Routing Basics

- Routing is done with tables
  - CIDR notation: 40.1.0.0/16
    - Match first 16 bits of 40.1.0.0, ignore remaining 16 bits
  - Find match, entry tells what link to send out on
  - Example
    - 40.0.0.0/8 => Link 0
    - 50.1.0.0/16 => Link 1
    - 50.3.27.0/24 => Link 3
    - 50.2.0.0/16 => Link 2
    - 50.3.42.0/24 => Link 1
Routing: More Complex

• Approach one: Static Routing
  • Enter all routes
  • Let system run
  • Hope nothing goes down
  • Works fine for small networks

• Reality:
  • Network links/systems go down
  • Often multiple paths to same place
    • Changing traffic patterns = changing fastest route
Distance Vector Protocols

- Routers
  - Know distances to immediate neighbors
  - Compute distance vector
    - How far to any destination from all known info
  - Transmit distance vectors to neighbors
    - Discover better (shorter) route? Update table
  - Now know more info, so repeat process
Distance Vector Routing

**Routing Table:**

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
<td>—</td>
</tr>
</tbody>
</table>

**Network Diagram:**

- A connected to C and B.
- B connected to D.
- C connected to A and D.
- D connected to B and A.

**Routing Updates:**

1. **A:**
   - Routing Table:
     - A: ∞ —
     - B: ∞ —
     - C: ∞ —
     - D: ∞ —

2. **C:**
   - Routing Table:
     - A: 2 A
     - B: ∞ —
     - C: 2 C
     - D: ∞ —

3. **D:**
   - Routing Table:
     - A: ∞ —
     - B: ∞ —
     - C: ∞ —
     - D: 3 D

4. **B:**
   - Routing Table:
     - A: ∞ —
     - B: 1 B
     - C: ∞ —
     - D: 3 D
Distance Vector Routing

Dest | Cost | Rt
--- | --- | ---
A | ∞ | —
B | ∞ | —
C | ∞ | —
D | ∞ | —

Dest | Cost | Rt
--- | --- | ---
A | ∞ | —
B | 2 | B
C | ∞ | —
D | ∞ | —

Dest | Cost | Rt
--- | --- | ---
A | 2 | A
B | ∞ | —
C | 2 | C
D | ∞ | —

Dest | Cost | Rt
--- | --- | ---
A | ∞ | —
B | ∞ | —
C | ∞ | —
D | 3 | D

2+1 via v
2+3 via x
2+1 via v
Distance Vector Routing

### Table 1

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
<td>—</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
<td>—</td>
</tr>
</tbody>
</table>

### Table 3

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>D</td>
</tr>
</tbody>
</table>

### Table 4

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>D</td>
</tr>
</tbody>
</table>

**no improvements**
Distance Vector Routing

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>v</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>x</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>v</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>t</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>v</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>t</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>v</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>t</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>—</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>D</td>
</tr>
</tbody>
</table>
Distance Vector Routing

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>v</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>x</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>v</td>
</tr>
<tr>
<td>D</td>
<td>9</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
<td>w</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>w</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>t</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
<td>z</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>z</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>D</td>
</tr>
</tbody>
</table>
Distance Vector Routing

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>x</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>v</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>v</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>V</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>w</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
<td>w</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>w</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>t</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>v</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>x</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>v</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>v</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>z</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest</th>
<th>Cost</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6</td>
<td>z</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>z</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>D</td>
</tr>
</tbody>
</table>
Link State Protocols

• Another option: link state protocols
  • Send info about direct connections to all routers
  • All routers build global pictures of network
  • Run graph algorithms to find shortest paths
    • E.g., Dijkstra’s shortest path algorithm

• Global information is nice, but...
  • Complex for very large systems
  • How many routers on the internet?
  • Do they all exchange all their info and run Dijkstra’s?
    • Of course not..
    • So... what do we do?

Use Abstraction… (and hierarchy)
- Divide internet up into Autonomous Systems (ASes)
  - Each AS can advertise routes to other ASes
  - Routing internal to AS is hidden from outside world
    - Can be Link State, Distance Vector, other...
  - We won’t go into too many details
7-layer OSI model

- **IP**: hierarchical addresses + best effort delivery
Our messages so far

- IP Header has
  - Src IP
  - Dest IP
  - Payload type (what protocol)
  - Other info

- Side note: IP does fragmentation to fit within frame size
  - We aren’t covering that
Layer 4: Transport Layer

- Reliability (if used)
  - Acknowledgements of data receipt
  - Retries of failed data

- Flow Control
  - Restrict rate of data sending

- Multiplexing/De-multiplexing data
  - E.g., Ports: identify which program some data is for
  - Keep data streams separate
Layer 5: Session Layer

• Concept of “a connection”
  • Establish/terminate
  • (OSI includes a variety of obscure features not often used)

• TCP: combines these two layers together
  • Sets up/terminates sessions
  • Has sequence numbering for packets
  • Acknowledges (ACKs) packets that are received
  • Establishes flow control (responds to congestion by throttling sending)
• We’ll draw diagram with computers on each side
• Time goes down
• Three messages above (TCP’s “3 way handshake”)
• To open a new connection:
  • 1 computer sends SYN (“Hi, lets talk”)
     • All messages have sequence numbers including SYN
     • First sequence number of a new connection is random
     • TCP sequence numbers by byte
TCP

- Other computer
  - ACKs (Acknowledges) the message (says what sequence # it ACKs)
  - Also sends SYN “Hey sure, lets talk”
TCP

- First computer then ACKs this SYN
  - And probably sends data along with the ACK
- TCP control info (SYN, ACK, FIN): bits in TCP header
  - Packets can have multiple control bits on + carry data
TCP: Normal operation

- Data going right in blue
- ACKS coming left in green
  - note: ACK #ed by expected next data
- Sliding window (flow control)
  - Limit amount of un-ACKed data at a time
TCP: Re-ordered Data

- Data may get re-ordered in network
  - One packet takes one route, another takes another
- TCP: no problem
  - Sender re-orders data properly
  - Sends ACK for as much data as it has
TCP: Lost data

- Data may also get lost in the network
  - E.g., router is backlogged, can’t handle it has to drop from queue
- TCP will re-send un-ACKed data after a timeout
TCP: Duplicate Data

- Receiver may get duplicate data
  - 7000-7499 gets lost
  - But 7500-7599 arrives
  - Then sender re-sends both: no ACK for either (why not?)
  - No problem: receiver drops duplicate (can tell: sequence #s)
TCP: Closing connection

- Connection closed with FIN message
  - Receiver ACKs
- Other side may close (with FIN) [typical]
  - Or remain open: can still send data
  - Side that closed cannot send, but should receive/ACK
  - FIN/ACK may be one message
TCP: Closing connection

- What if ACK for FIN gets lost?
  - FIN gets retried... but other side expects connection is closed?
- TCP has a state to handle this
  - Connection expected to be closed, but resources/state still held
  - Times out if no activity (assumes ACK got through if no retry)
Flow control: Sliding Window

• Problem:
  • Congestion -> dropped packets
  • Dropped packets -> Retries
  • Retries = duplicates of data -> More congestion

  • Vicious cycle...

• TCP implements **flow control** with a **sliding window**
  • Limitation of amount of un-ACKed data out at a time
  • Retry required? Shrink window
    • Assumes congestion, tries to avoid it
  • No retries in a while? Grow window back
    • Maybe it cleared up?
7-layer OSI model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Physical Layer</td>
<td>Physical transmission</td>
</tr>
<tr>
<td>2 Data-link Layer</td>
<td>Data-link transmission</td>
</tr>
<tr>
<td>3 Network Layer</td>
<td>Network address and control</td>
</tr>
<tr>
<td>4 Transport Layer</td>
<td>Transport service</td>
</tr>
<tr>
<td>5 Session Layer</td>
<td>Session management</td>
</tr>
<tr>
<td>6 Presentation Layer</td>
<td>Presentation service</td>
</tr>
<tr>
<td>7 Application Layer</td>
<td>Application-specific service</td>
</tr>
</tbody>
</table>

- TCP: It’s the coolest thing since memory got sliced into pages!
Our messages so far

<table>
<thead>
<tr>
<th>Preamble</th>
<th>Header</th>
<th>Header</th>
<th>Header</th>
<th>Payload</th>
<th>CRC</th>
</tr>
</thead>
</table>

- **TCP header has**
  - Source/Dest port
  - Sequence numbers
  - Control bits (SYN/ACK/FIN)
  - Check sum over data
  - Other stuff
Layer 6: Presentation Layer

• Responsible for data formats
  • Examples
    • Character encoding schemes
    • Serialization of objects

• We’re not really going to talk about it much
Layer 7: Application Layer

- Protocol specific to how applications want to communicate
  - Examples:
    - HTTP(S)
    - (S)FTP
    - SSH
    - SMTP
    - IMAP
    - ...
- Again, not going into this much...
• Flexibility Example: Wired vs Wireless
  • Change out two layers, rest stay the same
• Flexibility Example: A different application on top of both
How to find out IP addresses?

- I don’t know about you, but whenever I want to visit Google, I just type “172.217.2.206” into my browser
  - Not really
- People don’t memorize IP addresses – need convenient way to translate human-readable names to IP addresses
- This is the **domain name system (DNS)**
- Hierarchical servers with hierarchical database:

  ![DNS Diagram]

Figure from [DNS – Domain Name System, CS234, UC Irvine](https://example.com).
DNS name resolution example

Recursive query:
- Resolves any query by consulting servers that are authoritative for the query
- To do so, servers may recursively query other servers higher up in the hierarchy

Adapted from DNS – Domain Name System, CS234, UC Irvine.
Additional DNS facts

- DNS contains multiple kinds of records:
  - Name -> IP translations ("A" records)
  - IP -> Name translations ("PTR" records)
  - Name -> Name translations ("CNAME" records)
  - more

- DNS clients and servers can cache results
  - The local Duke DNS server *definitely* has "google.com" cached

- Second-level domains ("duke.edu", "google.com", etc.) are registered, usually by paying a fee, with a registrar (e.g. "easydns.com") who interacts with the top-level domain authority (Verisign for .com, Educause for .edu, etc.)

- Sub-domains can be created administratively by the owner of a domain (e.g. "whatever.duke.edu" can be made by the "duke.edu" admin).
Wireshark demo
Network programming

• Coding networking code in...
  • Java: Look in java.net, start with Socket
  • C:
    • socket()
    • connect()
    • accept()
    • bind()
    • listen()
Example C sockets client
Does a simple HTTP request of google.com

// adapted from simple-client.c by Sean Walton and
// Macmillan Publishers
// adapted by Tyler Bletsch for Duke University

// This program will request http://google.com/ and
// print the first 1024 bytes of the response
// Note: for simplicity, the google IP address is hard
coded

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>     // needed for close
#include <string.h>     // needed for bzero
#include <sys/socket.h> // needed for socket calls
#include <resolv.h>     // needed for socket type
#include <arpa/inet.h>  // needed for inet_aton

#define PORT 80
#define SERVER_ADDR "172.217.1.14"
// ^ this is google.com
#define MAXBUF 1024

char request[] = "GET / HTTP/1.0\r\n\r\n";
int request_len = sizeof(request)-1;
#define die(s) { perror(s); exit(1); }

int main() {
  int sockfd;
  struct sockaddr_in dest;
  char buffer[MAXBUF];

  /*---Open socket for streaming---*/
  if ( ( sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 )
    die("socket");

  /*---Initialize server address/port struct---*/
  bzero(&dest, sizeof(dest));
  dest.sin_family = AF_INET;
  dest.sin_port = htons(PORT);
  if ( inet_aton(SERVER_ADDR, &dest.sin_addr) == 0 )
    die(SERVER_ADDR);

  /*---Connect to server---*/
  if ( connect(sockfd, (struct sockaddr*)&dest, sizeof(dest)) != 0 )
    die("connect");

  /*---Send request---*/
  send(sockfd, request, request_len, 0);

  /*---Get and print response---*/
  bzero(buffer, MAXBUF);
  recv(sockfd, buffer, sizeof(buffer), 0);
  printf("%s", buffer);

  /*---Clean up---*/
  close(sockfd);
  return 0;
}
Test run
Summary

• Networking Overview
  • 7-layer model
  • Emphasis on IP (Layer 3) and TCP (Layers 4 and 5)

• Not comprehensive, but...
• You are now at least conversant enough to discuss the OSI stack at parties