CS 3640: Introduction to Networks and Their Applications

Fall 2018, Lecture 17: The Application Layer I
(Credit: Christo Wilson @ NEU)

Instructor: Rishab Nithyanand
Teaching Assistant: Md. Kowsar Hossain
You should…

• **Be working on Assignment 3: Network layer addressing and BGP**
  • Due on Friday at 11:59 pm!

• **Get ready for the mid-term**
  • In class on Tuesday, October 23rd.

• **Know and understand:**
  • The three Internet design principles and components of the Internet.
  • Circuit- vs. packet- switched networks.
  • Components of end-to-end delay.
  • The link layer: error detection, MAC, local addressing/routing.
  • The network layer: addressing, fragmentation, IPv4 vs. IPv6, Ases, Interdomain and Intradomain routing.
  • The transport layer: core functionality, TCP vs. UDP, flow control vs. congestion control, advertised window vs. congestion window vs. effective window, dynamic congestion window size adjustment, phases of TCP congestion control, optimizations with fast retransmit and recovery.
This week in class

1. Intro to Distributed Systems
2. Network Address Translation
3. Domain Name System
The application layer

• This is where the logic for your application goes.

• Every app is different, but they all really only need to use the network through the Transport layer abstraction.

• You don’t have to write any code below this layer to make an Internet application!
  • Developers are insulated from the constant evolution of networking technology and protocols.
To understand Internet applications, you need to understand distributed systems.

“A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages.”

Essentially: Internet applications are distributed systems.

- Multiple computers connected by a network (Internet).
- Exchange messages with each other.
- To achieve a common goal (specified by the app).
The core functionality of the application layer

• **Essentially: Internet applications are distributed systems.**
  • Multiple computers connected by a network (Internet).
  • Exchange messages with each other.
  • To achieve a common goal (specified by the app).

• **Discuss: With the above view of Internet applications, what is the one core mechanism present in every Internet application?**
  • Remote Procedure Call (RPC): Invoke a method on a remote machine by sending a message to it.
  • Every Internet app is basically a collection of RPCs: Messages are received and processes are started.
Early Internet applications

- Early Internet applications were basically implementing distributed system tasks.
  - Remote procedure calls (RPC)
    - The most basic task. Send a message to invoke a function.
  - Remote access (login, telnet)
    - Send credentials to get access to resources.
  - Messaging between humans (email)
    - Send a message that gets stored remotely until the user with correct credentials can read it.
  - Bulletin boards (Usenet)
    - Post messages to a common computer so all users with correct credentials can read and respond.
Early Internet applications: Sabre

• **Sabre was the earliest airline Global Distribution System**
  • This is still the system used in airports.
Early Internet applications: Sabre

• **Before 1960: Each airline had a central office with one card per flight per day.**
  - Travel agent calls in, worker marks the sold seat on the card.

• **1960s: Removed the human from the loop.**
  - Disk with each memory location representing number of seats sold on a flight.
  - Built network connecting various agencies.
  - A terminal at each agency allowing lookup of memory locations (seat availability).
Early Internet applications: Sabre

RESERVATIONS PROCESSING SYSTEM
computers speed air travel reservations...

In addition to handling the passenger's reservation, this new IBM system also:
- Answers requests for space from other airlines
- Advises agents to rebook passengers to pick up tickets
- Maintains and processes passengers waiting lists for fully-booked flights
- Supplies fare quotations
- Supplies information on arrival and departure times
- Reminds agents to advise scheduled passengers of any flight changes
Microcomputers and Internet applications

• In the 1980s, personal computers became more popular.
  • We moved away from mainframes and terminals. Applications could expect more resources than before.

• Developed many new distributed systems:
  • Email
  • Web
  • DNS
  • …

• The scale of networks grew quickly, the Internet came to dominate.
Internet applications today

• **Growing amounts of mobile computing.**
  • End users connect via a variety of networks and devices.
  • More challenging to build systems: Need to care about energy, interoperation, etc.

• **Cloud computing is increasingly the norm.**
  • Computation and connectivity is purchased.
  • All data is stored and served from the cloud.

• **Applications have evolved in complexity and requirements.**
  • Mainframes & terminals → personal computers → mobile devices & the cloud.
Example Internet application: Domain Name System

- **DNS is a distributed database: Maps domain names to IP addresses and vice-versa.**
  - No single system has the entire mapping.

- **DNS has a hierarchical structure.**
  - Servers divide up the administrative tasks among themselves.
  - Clients can more efficiently resolve names.

- **Uses a simple client/server architecture. More in the next lecture.**
Example Internet application: The Web

• The Web is another widely popular distributed system.

• Two types of entities:
  • Web browsers: Clients that render web pages.
  • Web servers: Machines that send data to clients.

• All communication occurs over the Hyper Text Transfer Protocol (HTTP). More about HTTP next week.
Example Internet application: BitTorrent

• **A popular distributed system for content distribution.**

• **Only one type of entity:**
  • A BitTorrent client

• **All clients collaboratively download data using a custom protocol over HTTP. Robust to disconnections and failure. More next week.**
Example Internet application: The Stock Market

- Large distributed system with many different players and unaligned economic interests.

- Centralized database system with many concurrent transactions.

- All transactions have to get processed in order.

- Transmission delays are a huge concern.
  - Hedge funds will often buy rack space closer to the exchange datacenter.
  - This helps shave of millisecond differences in delay and improve profit margins.
Challenges with building Internet applications: Incomplete knowledge

• **No host has global knowledge.**

• **Hosts need to use the network to exchange state information.**
  • But the network capacity is limited.

• **Information can be incorrect or out of date.**
  • New information can take time to propagate.
  • New changes can happen in the mean time.

• **Key issue: How can you detect and address inconsistencies?**
Challenges with building Internet applications: Imperfect time

• **Time cannot be measured perfectly.**
  - Hosts have different clocks and skew.
  - The network may delay synchronization messages.

• **How do you determine the order of messages and events?**
  - In a game, which player shot first?
  - In Sabre, who bought the last seat on the plane?

• **We need a more nuanced abstraction to represent time.**
Challenges with building Internet applications: Handling failures

• “A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”— Leslie Lamport.

• Failure is the common case.
  • As systems get more complex, failures are more likely.
  • Systems have to be built with fault/failure tolerance in mind.

• Example: In a file sharing system, what happens if a node fails?
Challenges with building Internet applications: Scalability

• The size of a system will grow over time. How does your system handle future users, hosts, networks, etc?

• For example: In a multiplayer game, each user needs to send a location to all other users.
  • $O(n^2)$ message complexity. Will quickly overwhelm the network when $n > 100$.
  • Option: Reduce the frequency of updates.
  • Option: Choose which nodes actually need to send/receive updates.
Challenges with building Internet applications: Concurrency

• To work at Internet scale, a distributed system must try to leverage concurrency – i.e., enable multiple simultaneous operations on the system state.

• For example: Web servers are often clustered and replicated.
  • How do you ensure that each object is in a consistent state?
    • Example: Web server for making stock market trades.
Challenges with building Internet applications: Security

• **Distributed systems often have many different entities.**
  • They may not trust each other (example: stock market)
  • They may not be under centralized control (example: Web)
  • There are many economic incentives for abuse by each of these entities.

• **Systems often will need to provide:**
  • Confidentiality: Only intended parties can read
  • Integrity: No messages have been modified
  • Availability: The system cannot be brought offline.
Distributed system design decisions: System architectures

• **Discuss:** What system architectures did we see in our application examples (Web, BitTorrent, etc)? What types of entities and entity interactions did we see?

• **There are two primary architectures:** Client-server and peer-to-peer.
  • **Client-server:** The system is divided into clients (limited in power and scope) and servers (more powerful and more visibility than clients).
    • Clients send requests to servers.
  • **Peer-to-peer:** All hosts are “equal” or hosts act as both clients and servers.
    • Requests are sent to other hosts.
    • More complicated to design, but higher availability and resilience to failure.
Distributed system design decisions: Messaging interfaces

• **Discuss:** What should a programmer do after a message is sent?

• **Messaging is fundamentally asynchronous.** Client asks the network to deliver a message and waits for the response.

• **Programmers see two messaging interfaces:**
  
  • **Synchronous:** A thread is “blocked” until a response comes back (question, answer, question, answer, …). This is much more intuitive and easy to debug.
  
  • **Asynchronous:** After a message is sent, control is immediately returned to the thread. Responses may come in later. Much higher performance, but harder to implement and debug.
Distributed system design decisions: Transport protocol

• At a minimum, system designers have two choices for transport
  • UDP
    • Good: low overhead (no retries or order preservation), fast (no congestion control)
    • Bad: no reliability, may increase network congestion
  • TCP:
    • Good: highly reliable, fair usage of bandwidth
    • Bad: high overhead (handshake), slow (slow start, ACK clocking, retransmissions)

• However, you can always roll your own protocol on top of UDP
  • Microtransport Protocol (uTP) – used by BitTorrent
  • QUIC – invented by Google, used in Chrome to speed up HTTP
Distributed system design decisions: Serialization and data format

• All hosts must be able to exchange data, thus choosing data formats is crucial
  • On the Web – form encoded, URL encoded, XML, JSON, …
  • In “hard” systems – MPI, Protocol Buffers, Thrift

• Discuss: What should you consider while contemplating data formats for your distributed system?
  • Openness: is the format human readable or binary? Proprietary?
  • Efficiency: text is bloated compared to binary
  • Versioning: can you upgrade your protocol to v2 without breaking v1 clients?
  • Language support: do your formats and types work across languages?
Distributed system design decisions: Naming

• **Need to be able to refer to hosts/processes**

• **Naming decisions should reflect system organization**
  
  • E.g., with different entities, hierarchal system may be appropriate (entities name their own hosts)

• **Discuss: What should you think about while naming hosts in a distributed system?**
  
  • Mobility: hosts may change locations
  • Authenticity: how do hosts prove who they are?
  • Scalability: how many hosts can a naming system support?
  • Convergence: how quickly do new names propagate?
This week in class

1. Intro to Distributed Systems
2. Network Address Translation
3. Domain Name System
The IPv4 address shortage

- **Problem: consumer ISPs typically only give one IP address per-household**
  - Additional IPs cost extra
  - More IPs may not be available

- **Today’s households have more networked devices than ever**
  - Laptops and desktops
  - TV, Blu-ray players, game consoles
  - Tablets, smartphones, eReaders

- **Discuss: How to get all these devices online? How should we give them IP addresses?**
Working around the IPv4 address shortage

• Idea: create a range of private IPs that are separate from the rest of the network
  • Use the private IPs for internal routing
  • Use a special router to bridge the LAN and the WAN

• Properties of private IPs
  • Not globally unique
  • Usually taken from non-routable IP ranges

• Typical private IP ranges
  • 10.0.0.0 – 10.255.255.255
  • 172.16.0.0 – 172.31.255.255
  • 192.168.0.0 – 192.168.255.255
Working around the IPv4 address shortage
Network Address Translation (NAT)

- **NAT allows hosts on a private network to communicate with the Internet**

- **Special router at the boundary of a private network**
  - Replaces internal IPs with external IP by modifying packet headers
    - This is “Network Address Translation”
  - May also replace TCP/UDP port numbers

- **Maintains a table of active flows**
  - Outgoing packets initialize a table entry
  - Incoming packets are rewritten based on the table
Basic NAT operation

<table>
<thead>
<tr>
<th>Private Network</th>
<th>Internet</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Private Address</th>
<th>Public Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.0.1:2345</td>
<td>74.125.228.67:80</td>
</tr>
</tbody>
</table>

192.168.0.1  66.31.210.69  74.125.228.67

Source: 74.125.228.67:80 Dest: 192.168.0.1:2345
Discuss: Advantages of NAT

• **Allow multiple hosts to share a single public IP**

• **Allow migration between ISPs**
  - Even if the public IP address changes, you don’t need to reconfigure the machines on the LAN

• **Load balancing**
  - Forward traffic from a single public IP to multiple private hosts
Discuss: Problems with NAT

• **Performance/scalability issues**
  • Per flow state!
  • Modifying IP and Port numbers means NAT must recompute IP and TCP checksums

• **Breaks the layered network abstraction**

• **Breaks end-to-end Internet connectivity**
  • 192.168.*.* addresses are private
  • Cannot be routed to on the Internet
  • Problem is worse when both hosts are behind NATs

• **What about IPs embedded in data payloads?**
Enabling outside connectivity through NATs

• **Discuss: How do 2 hosts behind a NAT communicate?**
Enabling outside connectivity through NATs

- **The TURN (Traversal Using Relays around NAT) protocol**