Hyperconverged Infrastructure

- Internet as a global system
- Seamless integration of compute, network and storage
- Performance vs. Layering
- New technologies
- New Applications
Subjects To Be Covered

- Software Defined Network
- Software Defined Storage
- Solid State Drives
- Non-Volatile Memory
- Virtual Machine + Docker Container
- Data Deduplication
- Key-Value Store
A Global System: Future Internet

• Data can be stored and accessed from any where on the earth (as long as they are parts of Internet)
• Internet consists of compute, storage and networking components
• Services are offered via Internet (where is end-to-end?)
• A new thinking and new design of Internet are required
• Most of Internet components become white-boxes
Review of Old Internet Architecture

- Internet in a Nutshell:
  - Internet service model
  - Fundamental issues in network design

- Basic Internet Architecture
  - “Hour-glass” architecture
  - IP datagram formats; UDP/TCP segment formats
  - IP addressing and routing protocols

- Internet Philosophy (and Design Principles)
  - “end-to-end” argument
What is a Network/Internet?

Compare Internet with Postal Service and Telephone System

- Various Key Pieces and Their Functions
- Services Provided
- How the pieces work together to provide services
Service Perspective

Basic Services Provided

- **Postal**: deliver mail/package from people to people
  - First class, express mail, bulk rate, certified, registered, ...
- **Telephone**: connect people for talking
  - You may get a busy dial tone
  - Once connected, consistently good quality, unless using cell phones
- **Internet**: transfer information between people/machines
  - Reliable connection-oriented or unreliably connectionless services!
  - You never get a busy dial tone, but things can be very slow!
  - You can’t ask for express delivery (not at the moment at least!)
IP Service Model

- **Packet-switching data network**
  - shared infrastructure, *statistical multiplexing*!
  - each packet carries source and destination
  - “logical” network of networks, “overlaid” on top of various “physical networks, running TCP/IP protocol suite

- **Best-effort delivery** (unreliable service)
  - connectionless (“packet” or datagram-based)
  - packets may be lost, duplicated, delivered out of order
  - packets can be delayed for a long time
  - …..

- **Global reachability**
  - global addressing (public IPv4 and IPv6 addresses)
    - but firewalls, NATs, …
  - BGP network reachability announcement (next class!)
Fundamental Issues in Networking

• Naming/Addressing
  - How to find name/address of the party (or parties) you would like to communicate with
  - Address: byte-string that identifies a node
  - Types of addresses
    • Unicast: node-specific
    • Broadcast: all nodes in the network
    • Multicast: some subset of nodes in the network

• Routing/Forwarding: process of determining how to send packets towards the destination based on its address
  - Finding out neighbors, building routing tables
Fundamental Problems in Networking ...

What can go wrong?

- Bit-level errors: due to electrical interferences
- Packet-level errors: packet loss due to buffer overflow/congestion
- Out of order delivery: packets may take different paths
- Link/node failures: cable is cut or system crash
- Human configuration/operational errors
- Malicious attacks!
Internet Architecture

• packet-switched datagram network
• IP is the glue (network layer overlay)
• IP hourglass architecture
  - all hosts and routers run IP
• stateless architecture
  - no per flow state inside network
Internet Protocol “Zoo”
The Internet Network layer

Transport layer: TCP, UDP

- Routing protocols
  - path selection
  - RIP, OSPF, BGP

- IP protocol
  - addressing conventions
  - packet handling conventions

- ICMP protocol
  - error reporting
  - router “signaling”

Data Link layer (Ethernet, WiFi, PPP, …)

- Physical Layer (SONET, …)
## IP Datagram Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP protocol version</td>
<td>Number of IP version supported by protocol.</td>
</tr>
<tr>
<td>header length</td>
<td>Length (bytes) of header portion.</td>
</tr>
<tr>
<td>“type” of data</td>
<td>Type of data - e.g., TCP, UDP, ICMP, etc.</td>
</tr>
<tr>
<td>max number remaining</td>
<td>Maximum number of hops remaining.</td>
</tr>
<tr>
<td>remaining hops</td>
<td>(Decrement at each router)</td>
</tr>
<tr>
<td>upper layer protocol</td>
<td>Protocol used to deliver payload.</td>
</tr>
<tr>
<td>32 bit source IP address</td>
<td>Source IP address of the sender.</td>
</tr>
<tr>
<td>32 bit destination IP address</td>
<td>Destination IP address of the receiver.</td>
</tr>
<tr>
<td>Options (if any)</td>
<td>Options - e.g., timestamp, record route taken, specify list of routers to visit.</td>
</tr>
<tr>
<td>data</td>
<td>Data - (variable length, typically a TCP or UDP segment)</td>
</tr>
<tr>
<td>Internet checksum</td>
<td>Checksum for integrity of datagram.</td>
</tr>
<tr>
<td>time to live</td>
<td>Time the packet can live (seconds).</td>
</tr>
<tr>
<td>upper layer</td>
<td>Protocol used to deliver payload.</td>
</tr>
<tr>
<td>fragment</td>
<td>For fragmentation/reassembly.</td>
</tr>
<tr>
<td>offset</td>
<td>Fragment offset.</td>
</tr>
<tr>
<td>length</td>
<td>Total datagram length (bytes).</td>
</tr>
<tr>
<td>header length</td>
<td>Length (bytes) of header portion.</td>
</tr>
<tr>
<td>“type” of data</td>
<td>Type of data - e.g., TCP, UDP, ICMP, etc.</td>
</tr>
<tr>
<td>max number remaining</td>
<td>Maximum number of hops remaining.</td>
</tr>
<tr>
<td>remaining hops</td>
<td>(Decrement at each router)</td>
</tr>
<tr>
<td>upper layer protocol</td>
<td>Protocol used to deliver payload.</td>
</tr>
<tr>
<td>32 bit source IP address</td>
<td>Source IP address of the sender.</td>
</tr>
<tr>
<td>32 bit destination IP address</td>
<td>Destination IP address of the receiver.</td>
</tr>
<tr>
<td>Options (if any)</td>
<td>Options - e.g., timestamp, record route taken, specify list of routers to visit.</td>
</tr>
<tr>
<td>data</td>
<td>Data - (variable length, typically a TCP or UDP segment)</td>
</tr>
<tr>
<td>Internet checksum</td>
<td>Checksum for integrity of datagram.</td>
</tr>
<tr>
<td>time to live</td>
<td>Time the packet can live (seconds).</td>
</tr>
<tr>
<td>upper layer</td>
<td>Protocol used to deliver payload.</td>
</tr>
<tr>
<td>fragment</td>
<td>For fragmentation/reassembly.</td>
</tr>
<tr>
<td>offset</td>
<td>Fragment offset.</td>
</tr>
</tbody>
</table>
IP Addresses & Datagram Forwarding

• IPv4 Address
  - 32 bits
  - two-parts: network prefix and host parts
  - E.g., 128.101.33.101
    network prefix: 128.101.0.0/16

• Forwarding and IP address
  - forwarding based on network prefix
    • Delivers packet to the appropriate network
    • Once on destination network, direct delivery using host id

• IP destination-based next-hop forwarding paradigm
  - Each host/router has IP forwarding table
    • Entries like <network prefix, next-hop, output interface>
Datagram Networks: the Internet model

• routers: no state about end-to-end connections
  - no network-level concept of “connection”
• packets forwarded using destination host address
  - packets between same source-dest pair may take different paths, when intermediate routes change!
Routing in the Internet

- The Global Internet consists of Autonomous Systems (AS) interconnected with each other:
  - Stub AS: small corporation: one connection to other AS’s
  - Multihomed AS: large corporation (no transit): multiple connections to other AS’s
  - Transit AS: provider, hooking many AS’s together

- Two-level routing:
  - Intra-AS: administrator responsible for choice of routing algorithm within network
  - Inter-AS: unique standard for inter-AS routing: BGP
Internet: “networks of networks”!
Intra-AS border (exterior gateway) routers

Inter-AS interior (gateway) routers
Intra-AS vs. Inter-AS Routing

Inter-AS routing between A and B

Intra-AS routing within AS A

Intra-AS routing within AS B
Inter-AS Routing in the Internet: BGP

Figure 4.5.2-new2: BGP use for inter-domain routing
Internet Transport Protocols

TCP service:
- connection-oriented: setup required between client, server
- reliable transport between sender and receiver
- flow control: sender won’t overwhelm receiver
- congestion control: throttle sender when network overloaded

Both provide logical communication between app processes running on different hosts!

UDP service:
- unreliable data transfer between sender and receiver
- does not provide: connection setup, reliability, flow control, congestion control
Multiplexing/Demultiplexing

**Demultiplexing at rcv host:**
delivering received segments to correct application process

**Multiplexing at send host:**
gathering data from multiple app processes, enveloping data with header (later used for demultiplexing)

- = API (“socket”)
- = process

---

<table>
<thead>
<tr>
<th></th>
<th>application</th>
<th>transport</th>
<th>network</th>
<th>link</th>
<th>physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>host 1</td>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>application</th>
<th>transport</th>
<th>network</th>
<th>link</th>
<th>physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>host 2</td>
<td>P1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>application</th>
<th>transport</th>
<th>network</th>
<th>link</th>
<th>physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>host 3</td>
<td>P4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
UDP: User Datagram Protocol [RFC 768]

- “no frills,” “bare bones” Internet transport protocol
- “best effort” service, UDP segments may be:
  - lost
  - delivered out of order to app
- connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?
- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired
UDP (cont’d)

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive

- other UDP uses
  - DNS
  - SNMP

- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!

UDP segment format:
- source port #
- dest port #
- length
- checksum

Length, in bytes of UDP segment, including header

Application data (message)

UDP segment format
TCP: Overview

• **point-to-point:**
  - one sender, one receiver

• **reliable, in-order byte steam:**
  - no “message boundaries”

• **pipelined:**
  - TCP congestion and flow control
  - set window size

• **send & receive buffers**

• **full duplex data:**
  - bi-directional data flow in same connection
  - MSS: maximum segment size

• **connection-oriented:**
  - handshaking (exchange of control msgs) init’s sender, receiver state before data exchange

• **flow controlled:**
  - sender will not overwhelm receiver

![Diagram of TCP send and receive buffers](image)
TCP Segment Structure

<table>
<thead>
<tr>
<th>Source port #</th>
<th>Dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgement number</td>
<td></td>
</tr>
<tr>
<td>Head Length</td>
<td>Not Used</td>
</tr>
<tr>
<td>Checksum</td>
<td>Pointer to urgent data</td>
</tr>
<tr>
<td>Options (variable length)</td>
<td></td>
</tr>
<tr>
<td>Application data (variable length)</td>
<td></td>
</tr>
</tbody>
</table>

**URG**: urgent data (generally not used)

**ACK**: ACK # valid

**PSH**: push data now (generally not used)

**RST, SYN, FIN**: connection establishment (setup, teardown commands)

Internet checksum (as in UDP)

Counting by bytes of data (not segments!)

# bytes receiver willing to accept
Domain Name System (DNS)

- Properties of DNS
  - Hierarchical name space divided into zones
  - Translation of names to/from IP addresses
  - Distributed over a collection of DNS servers

- Client application
  - Extract server name (e.g., from the URL)
  - Invoke system call to trigger DNS resolver code
  - E.g., `gethostbyname()` on “www.foo.com”

- Server application
  - Extract client IP address from socket
  - Optionally invoke system call to translate into name
  - E.g., `gethostbyaddr()` on “12.34.158.5”
Domain Name System

generic domains

com  edu  •••  org

country domains

ac  •••  uk  zw

uname root

12.34.56.0/24

afer.cs.umn.edu

usr.cam.ac.uk
Caching based on a time-to-live (TTL) assigned by the DNS server responsible for the host name to reduce latency in DNS translation.
Application-Layer Protocols

- **Messages exchanged between applications**
  - Syntax and semantics of the messages between hosts
  - Tailored to the specific application (e.g., Web, e-mail)
  - Messages transferred over transport connection (e.g., TCP)

- **Popular application-layer protocols**
  - Telnet, FTP, SMTP, NNTP, HTTP, ...

---

Client

GET /index.html HTTP/1.1

Server

HTTP/1.1 200 OK
Example: Many Steps in Web Download

Sources of variability of delay

- Browser cache hit/miss, need for cache revalidation
- DNS cache hit/miss, multiple DNS servers, errors
- Packet loss, high RTT, server accept queue
- RTT, busy server, CPU overhead (e.g., CGI script)
- Response size, receive buffer size, congestion
- ... downloading embedded image(s) on the page
This course focuses on the routers...
Happy Routers Make Happy Packets

• Routers forward packets
  - Forward incoming packet to outgoing link
  - Store packets in queues
  - Drop packets when necessary

• Routers compute paths
  - Routers run routing protocols
  - Routers compute forwarding tables

• A famous quotation from RFC 791
  - “A name indicates what we seek. An address indicates where it is. A route indicates how we get there.”
  -- Jon Postel
Internet Philosophy and Design Principles

Architecture: the big picture

Goals:
- identify, study principles that can guide network architecture
- “bigger” issues than specific protocols or implementation tricks
- synthesis: the really big picture
Key questions

• How to decompose the complex system functionality into protocol layers?
• Which functions placed *where* in network, at which layers?
• Can a function be placed at multiple levels?

Answer these questions in context of Internet, telephone network
Common View of the Telco Network

brick (dumb)  lock (you can’t get in)

brain (smart)
Common View of the IP Network
Readings: Saltzer84

• End-to-end argument
  - Better to implement functions close to application
  - ... except when performance requires otherwise

• Why?
  - ...

• What should be the “end” for network “functionalities”, e.g., routing?
  - Router?
  - End host?
  - Enterprise edge?
  - Autonomous System?
Internet End-to-End Argument

According to [Saltzer84]:

• “...functions placed at the lower levels may be redundant or of little value when compared to the cost of providing them at the lower level...”

• “...sometimes an incomplete version of the function provided by the communication system (lower levels) may be useful as a performance enhancement...”

• This leads to a philosophy diametrically opposite to the telephone world of dumb end-systems (the telephone) and intelligent networks.
Example: Reliable File Transfer

- Solution 1: make each step reliable, and then concatenate them
- Solution 2: each step unreliable: end-to-end check and retry
Discussion

- Solution 1 not good enough!
  - what happens if the sender or/and receiver misbehave?
- so receiver has to do check anyway!
- Thus, full functionality can be entirely implemented at application layer; no need for reliability from lower layers
Discussion

**Q:** Is there any reason to implement reliability at lower layers?

**A:** Yes, but only to improve performance

- Example:
  - assume high error rate in network
  - reliable communication service at data link layer might help (why)?
  - fast detection /recovery of errors
E2E Argument: Interpretations

• One interpretation:
  - A function can only be completely and correctly implemented with the knowledge and help of the applications standing at the communication endpoints

• Another: (more precise...)
  - a system (or subsystem level) should consider only functions that can be completely and correctly implemented within it.

• Alternative interpretation: (also correct ...)
  - Think twice before implementing a functionality that you believe that is useful to an application at a lower layer
  - If the application can implement a functionality correctly, implement it a lower layer only as a performance enhancement
Internet & End-to-End Argument

- network layer provides one simple service: best effort datagram (packet) delivery
- transport layer at network edge (TCP) provides end-end error control  
  - performance enhancement used by many applications  
    (which could provide their own error control)
- all other functionalities ...
  - all application layer functionalities
  - network services: DNS  
    implemented at application level
Internet & End-to-End Argument

Discussion: congestion control, “error” control, flow control: why at transport, rather than link or application layers?

• Claim: common functions should migrate down the stack
  - Everyone shares same implementation: no need to redo it (reduces bugs, less work, etc…)
  - Knowing everyone is doing the same thing, can help

• congestion control too important to leave up to application/user: true but hard to police
  - TCP is “outside” the network; compliance is “optional”
  - We do this for fairness (but realize that people could cheat)

• Why error control, flow control in TCP, not (just) in app
Trade-offs

- application has more information about the data and semantics of required service (e.g., can check only at the end of each data unit)
- lower layer has more information about constraints in data transmission (e.g., packet size, error rate)
- Note: these trade-offs are a direct result of layering!
End-to-End Argument: Critical Issues

• end-to-end principle emphasizes:
  - *function placement*
  - *correctness, completeness*
  - *overall system costs*

• Philosophy: if application can do it, don’t do it at a lower layer -- application best knows what it needs
  - add functionality in lower layers iff (1) used by and improves performances of many applications, (2) does not hurt other applications

• allows *cost-performance* tradeoff
End-to-End Argument: Discussion

• end-end argument emphasizes correctness & completeness, but not
  - complexity: is complexity at edges result in a “simpler” architecture?
  - evolvability, ease of introduction of new functionality: ability to evolve because easier/cheaper to add new edge applications than change routers?
  - technology penetration: simple network layer makes it “easier” for IP to spread everywhere
Summary: End-to-End Arguments

• If the application can do it, don’t do it at a lower layer -- anyway the application knows the best what it needs
  - add functionality in lower layers iff it is (1) used and improves performances of a large number of applications, and (2) does not hurt other applications

• Success story: Internet
  - But ...
Next Week

• Read the required readings:
  - Internet design philosophy: Clark88,
    • also [Clark:Tussle] and [CerfKahn] if you have time
  - Cisco BGP Tutorial and [Huston99]
  - no need to submit reviews, but use your brain!

• Questions for you to think about:
  - What are the “architectural” advantages of Internet, and also its limitations?
  - If you can redesign it, how would you do it?