What's the Internet: "nuts and bolts" view

- PC
- server
- wireless laptop
- cellular handheld
- access points
- wired links
- router

- millions of connected computing devices: hosts = end systems
- running network apps
- communication links: fiber, copper, radio, satellite
- transmission rate = bandwidth
- routers: forward packets (chunks of data)

What's the Internet: "nuts and bolts" view

- protocols control sending, receiving of msgs
  - e.g., TCP, IP, HTTP, Skype, Ethernet
- Internet: "network of networks"
  - loosely hierarchical
  - public Internet versus private intranet
- Internet standards
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force

What's the Internet: a service view

- communication infrastructure enables distributed applications:
  - Web, VoIP, email, games, e-commerce, file sharing
- communication services provided to apps:
  - reliable data delivery from source to destination
  - "best effort" (unreliable) data delivery

How do loss and delay occur?

packets queue in router buffers
- packet arrival rate to link exceeds output link capacity
- packets queue, wait for turn

Delay in packet-switched networks

1. Nodal processing:
   - check bit errors
   - determine output link
2. Queueing
   - time waiting at output link for transmission
3. Transmission delay:
   - link bandwidth (bps)
   - bit-packet length (bits)
   - time to send bits into link = L/R
4. Propagation delay:
   - d = length of physical link
   - c = propagation speed in medium (~2x10^8 m/sec)
   - propagation delay = d/c

Why layering?

Dealing with complex systems:
- explicit structure allows identification, relationship of complex system's pieces
- layered reference model for discussion
- modularization eases maintenance, updating of system
- change of implementation of layer's service transparent to rest of system
- e.g., change in gate procedure doesn't affect rest of system
- layering considered harmful?
**Internet protocol stack**

- **application**: supporting network applications
  - FTP, SMTP, STTP
- **transport**: host-host data transfer
  - TCP, UDP
- **network**: routing of datagrams from source to destination
  - IP, routing protocols
- **link**: data transfer between neighboring network elements
  - PPP, Ethernet
- **physical**: bits "on the wire"

**Creating a network app**

write programs that
- run on (different) end systems
- communicate over network
  - e.g., web server software communicates with browser software
- little software written for devices in network core
  - network core devices do not run user applications
  - applications on end systems allows for rapid app development, propagation

**Processes communicating**

- **Process**: program running within a host.
  - within same host, two processes communicate using **inter-process communication** (defined by OS).
  - processes in different hosts communicate by exchanging **messages**

**Addressing processes**

- **to receive messages, process must have** **identifier**
  - host device has unique 32-bit IP address
- **Q**: does IP address of host on which process runs suffice for identifying the process?
  - **A**: No, many processes can be running on same host
  - **identifier** includes both IP address and port numbers associated with process on host.
  - Example port numbers:
    - HTTP server: 80
    - Mail server: 25
  - to send HTTP message to goa.cs.umass.edu web server:
    - IP address: 128.119.245.12
    - Port number: 80
  - more shortly...

**Sockets**

- process sends/receives messages to/from its **socket**
  - socket analogous to door
    - sending process shoves message out door
    - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process
- **API**: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)
App-layer protocol defines

- Types of messages exchanged:
  - e.g., request, response
- Message syntax:
  - what fields in messages & how fields are delineated
- Message semantics:
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:
- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:
- e.g., Skype

What transport service does an app need?

**Data loss**
- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

**Bandwidth**
- some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- other apps ("elastic apps") make use of whatever bandwidth they get

Transport service requirements of common apps

<table>
<thead>
<tr>
<th>Application</th>
<th>Data loss</th>
<th>Bandwidth</th>
<th>Time Sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>file transfer</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>e-mail</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>Web documents</td>
<td>loss-tolerant</td>
<td>audio: 5kpps-1Mpps, video: 10kpps-5Mpps</td>
<td>yes, 100's m/sec</td>
</tr>
<tr>
<td>real-time audio/video</td>
<td>loss-tolerant</td>
<td>audio: 5kpps-1Mpps, video: 10kpps-5Mpps</td>
<td>yes, 100's m/sec</td>
</tr>
<tr>
<td>stored audio/video</td>
<td>loss-tolerant</td>
<td>same as above</td>
<td>yes, few sec</td>
</tr>
<tr>
<td>interactive games</td>
<td>loss-tolerant</td>
<td>few kpps up</td>
<td>yes, 100's m/sec</td>
</tr>
<tr>
<td>instant messaging</td>
<td>no loss</td>
<td>elastic</td>
<td>yes and no</td>
</tr>
</tbody>
</table>

Internet transport protocols services

TCP service:
- connection-oriented: setup required between client and server processes
- reliable transport: between sending and receiving process
- flow control: sender won’t overwhelm receiver
- congestion control: throttle sender when network overloaded
- does not provide: timing, minimum bandwidth guarantees

UDP service:
- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee
- Why bother? Why is there a UDP?

Internet apps: application, transport protocols

<table>
<thead>
<tr>
<th>Application</th>
<th>Application layer protocol</th>
<th>Underlying transport protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-mail</td>
<td>SMTP [RFC 2821]</td>
<td>TCP</td>
</tr>
<tr>
<td>remote terminal access</td>
<td>Telnet [RFC 854]</td>
<td>TCP</td>
</tr>
<tr>
<td>Web</td>
<td>HTTP [RFC 2616]</td>
<td>TCP</td>
</tr>
<tr>
<td>file transfer</td>
<td>FTP [RFC 959]</td>
<td>TCP</td>
</tr>
<tr>
<td>streaming multimedia</td>
<td>proprietary</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>proprietary</td>
<td>typically UDP</td>
</tr>
</tbody>
</table>

Transport services and protocols

- provide logical communication between app processes running on different hosts
- transport protocols run in end systems
- send side: breaks app messages into segments, passes to network layer
- rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to app
- Internet: TCP and UDP
**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

**TCP: Overview** [RFCs: 793, 1122, 1323, 2018, 2581]

- **point-to-point:**
  - one sender, one receiver
- **reliable, in-order byte stream:**
  - no "message boundaries"
- **pipelined:**
  - TCP congestion and flow control set window size
- **send & receive buffers**

**TCP segment structure**

- **source port #**
- **dest port #**
- **sequence number**
- **acknowledgement number**
- **data**
- **options**

**TCP: retransmission scenarios**

**TCP retransmission scenarios (more)**
Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs:
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

TCP Flow Control

- receive side of TCP connection has a receive buffer:
- flow control: receiver won’t overflow receiver’s buffer by transmitting too much, too fast
- speed-matching service: matching the send rate to the receiving app’s drain rate

TCP congestion control: additive increase, multiplicative decrease

- Approach: increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - additive increase: increase CongWin by 1 MSS every RTT until loss detected
  - multiplicative decrease: cut CongWin in half after loss

TCP Congestion Control: details

- sender limits transmission: LastByteSent - LastByteAcked <= CongWin
- Roughly, rate = CongWin / RTT Bytes/sec
- CongWin is dynamic, function of perceived network congestion
- How does sender perceive congestion?
  - loss event = timeout or 3 duplicate acks
  - TCP sender reduces rate (CongWin) after loss event
  - three mechanisms:
    - AIMD
    - slow start
    - conservative after timeout events

TCP Slow Start

- When connection begins, CongWin = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate

TCP Slow Start (more)

- When connection begins, increase rate exponentially until first loss event:
  - double CongWin every RTT
  - done by incrementing CongWin for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast
Refinement

Q: When should the exponential increase switch to linear?
A: When CongWin gets to 1/2 of its value before timeout.

Implementation:
- Variable Threshold
- At loss event, Threshold is set to 1/2 of CongWin just before loss event

Refinement: inferring loss

- After 3 dup ACKs:
  - CongWin is cut in half
  - window then grows linearly
- But after timeout event:
  - CongWin instead set to 1 MSS,
  - window then grows exponentially
  - to a Threshold, then grows linearly

Network layer

- transport segment from sending to receiving host
- on sending side:
  - encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in every host, router
- router examines header fields in all IP datagrams passing through it

Two Key Network-Layer Functions

- forwarding: move packets from router's input to appropriate router output
- routing: determine route taken by packets from source to dest.
  - routing algorithms

Datagram networks

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of "connection"
- packets forwarded using destination host address
  - packets between some source/dest pair may take different paths

Forwarding table

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010000 00000000</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00010111 00010111</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011000 00000000</td>
<td>2</td>
</tr>
<tr>
<td>11001000 00010111 00011001 00000000</td>
<td>3</td>
</tr>
<tr>
<td>otherwise</td>
<td></td>
</tr>
</tbody>
</table>
**Longest prefix matching**

<table>
<thead>
<tr>
<th>Prefix Match</th>
<th>Link Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010110</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011000</td>
<td>2</td>
</tr>
<tr>
<td>11001000 00010111 00011</td>
<td>3</td>
</tr>
</tbody>
</table>

Examples

DA: 11001000 00010111 00011000 Which interface?

DA: 11001000 00010111 00011 Which interface?

**The Internet Network layer**

Host, router network layer functions:

- **Transport layer**: TCP, UDP
- **Routing protocols**: RIP, OSPF, BGP
- **Forwarding table**
- **IP protocol**: addressing conventions, datagram format, packet handling conventions
- **ICMP protocol**: error reporting, router "signaling"

**IP Addressing: introduction**

- **IP address**: 32-bit identifier for host, router interface
- **Interface**: connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one interface
  - IP addresses associated with each interface

**Routing Algorithm classification**

- **Global or decentralized information?**
  - **Global**: all routers have complete topology, link cost info
  - "link state" algorithms
  - **Decentralized**: router knows physically-connected neighbors, link costs to neighbors
  - Iterative process of computation, exchange of info with neighbors
  - "distance vector" algorithms

- **Static or dynamic?**
  - **Static**: routes change slowly over time
  - **Dynamic**: routes change more quickly
    - periodic update
    - in response to link cost changes
**Hierarchical Routing**
- aggregate routers into regions, "autonomous systems" (AS)
- routers in same AS run same routing protocol
  - "intra-AS" routing protocol
  - routers in different AS can run different intra-AS routing protocol

**Intra-AS Routing**
- also known as Interior Gateway Protocols (IGP)
- most common Intra-AS routing protocols:
  - RIP: Routing Information Protocol
  - OSPF: Open Shortest Path First
  - IGRP: Interior Gateway Routing Protocol (Cisco proprietary)

**Internet inter-AS routing: BGP**
- BGP (Border Gateway Protocol): the de facto standard
- BGP provides each AS a means to:
  1. Obtain subnet reachability information from neighboring ASs.
  2. Propagate reachability information to all AS-internal routers.
  3. Determine "good" routes to subnets based on reachability information and policy.
- allows subnet to advertise its existence to rest of Internet: "I am here"

**Link Layer: Introduction**
- Some terminology:
  - hosts and routers are nodes
  - communication channels that connect adjacent nodes along communication path are links
    - wired links
    - wireless links
    - LANs
  - layer-2 packet is a frame, encapsulates datagram
- data-link layer has responsibility of transferring datagram from one node to adjacent node over a link

**Link Layer Services**
- framing, link access:
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, dest
    - different from IP address!
- reliable delivery between adjacent nodes
  - we learned how to do this already (chapter 3)
  - seldom used on low bit-error link (fiber, some twisted pair)
  - wireless links: high error rates
    - Q: why both link-level and end-end reliability?

**Link Layer Services (more)**
- flow control:
  - pacing between adjacent sending and receiving nodes
- error detection:
  - errors caused by signal attenuation, noise
  - receiver detects presence of errors
    - signals sender for retransmission or drops frame
- error correction:
  - receiver identifies and corrects bit error(s) without resorting to retransmission
- half-duplex and full-duplex
  - with half duplex, nodes at both ends of link can transmit, but not at same time
**Where is the link layer implemented?**

- in each and every host
- link layer implemented in "adaptor" (aka network interface card (NIC))
  - Ethernet card, PCMCIA card, 802.11 card
  - implements link, physical layer
- attaches into host’s system busses
- combination of hardware, software, firmware

---

**Adaptors Communicating**

- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side:
  - looks for errors, rdt, flow control, etc.
  - extracts datagram, passes to upper layer at receiving side

---

**Multiple Access Links and Protocols**

Two types of "links":

- point-to-point
  - PPP for dial-up access
  - point-to-point link between Ethernet switch and host
- broadcast (shared wire or medium)
  - old-fashioned Ethernet
  - upstream HFC
  - 802.11 wireless LAN

---

**Multiple Access protocols**

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
  - collision if node receives two or more signals at the same time
  - multiple access protocol
- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
  - no out-of-band channel for coordination

---

**CSMA (Carrier Sense Multiple Access)**

- **CSMA**: listen before transmit:
  - If channel sensed idle: transmit entire frame
  - If channel sensed busy, defer transmission
- human analogy: don't interrupt others!

---

**CSMA/CD (Collision Detection)**

- **CSMA/CD**: carrier sensing, deferral as in CSMA
  - collisions detected within short time
  - colliding transmissions aborted, reducing channel wastage
- collision detection:
  - easy in wired LANs: measure signal strengths, compare transmitted, received signals
  - difficult in wireless LANs: received signal strength overwhelmed by local transmission strength
- human analogy: the polite conversationalist
**CSMA/CD collision detection**

- Time
- A, B, C, D
- Collision detection
- Space

**LAN Addresses and ARP**

Each adapter on LAN has unique LAN address

- LAN (wired or wireless)
- 71-65-F7-28-08-53
- 58-23-D7-FA-20-80
- DC-C4-11-6F-E3-98

**ARP: Address Resolution Protocol**

- Question: how to determine MAC address of B knowing B's IP address?
- ARP table: IP/MAC address mappings for some LAN nodes
- ARP protocol: Same LAN (network)

**Addressing: routing to another LAN**

- Addressing: routing to another LAN
- walkthrough: send datagram from A to B via R
- assume A knows B’s IP address
- two ARP tables in router R, one for each IP network (LAN)
**Ethernet**

“dominant” wired LAN technology:
- cheap $20 for NIC
- first widely used LAN technology
- simpler, cheaper than token LANs and ATM
- kept up with speed race: 10 Mbps - 10 Gbps

---

**Star topology**

- bus topology popular through mid 90s
  - all nodes in same collision domain (can collide with each other)
- today: star topology prevails
  - active switch in center
  - each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)

---

**Ethernet CSMA/CD algorithm**

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel idle, starts frame transmission
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame!
4. If NIC detects another transmission while transmitting, aborts and sends jam signal
5. After aborting, NIC enters exponential backoff: after mth collision, NIC chooses K at random from \(0, 1, 2, \ldots, 2^k - 1\). NIC waits \(K \cdot 512\) bit times, returns to Step 2

---

**Hubs**

- physical-layer ("dumb") repeaters:
  - bits coming in one link go out all other links at same rate
  - all nodes connected to hub can collide with one another
  - no frame buffering
  - no CSMA/CD at hub: host NICs detect collisions

---

**Switch**

- link-layer device: smarter than hubs, take active role
  - store, forward Ethernet frames
  - examine incoming frame’s MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- transparent
  - hosts are unaware of presence of switches
- plug-and-play, self-learning
  - switches do not need to be configured