16: Application, Transport, Network and Link Layers

Last Modified:
10/30/2002 9:15:15 PM

Roadmap

- Application Layer (User level)
- Transport Layer (OS)
- Network Layer (OS)
- Link Layer (Device Driver, Adapter Card)

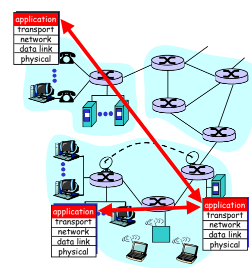
Application Layer

- Network Applications Drive Network Design
- Important to remember that network applications are the reason we care about building a network infrastructure
- Applications range from text based command line ones popular in the 1980s (like telnet, ftp, news, chat, etc) to multimedia applications (Web browsers, audio and video streaming, realtime videoconferencing, etc.)

Applications and application-layer protocols

Application: communicating, distributed processes
- running in network hosts in "user space"
- exchange messages to implement app
- e.g., email, file transfer, the Web

Application-layer protocols
- one "piece" of an app
- define messages exchanged by apps and actions taken
- user services provided by lower layer protocols

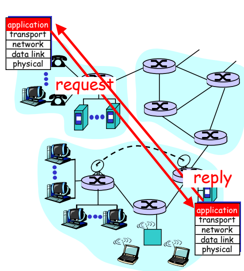


Client-server paradigm

Typical network app has two pieces: *client* and *server*

Client:
- initiates contact with server ("speaks first")
- typically requests service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader

Server:
- Running first (always?)
- provides requested service to client e.g., Web server sends requested Web page, mail server delivers e-mail



How do clients and servers communicate?

API: application programming interface
- defines interface between application and transport layer
- socket: Internet API
  - two processes communicate by sending data into socket, reading data out of socket

Q: how does a process "identify" the other process with which it wants to communicate?
- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered

… more on this later.

Socket programming

Goal: learn how to build client/server application that communicate using sockets

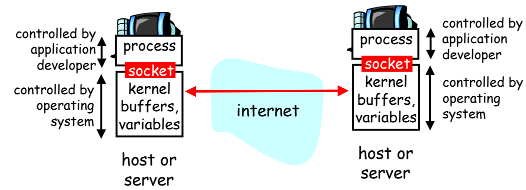
Socket API
- introduced in BSD4.1 UNIX, 1981
- Sockets are explicitly created, used, released by applications
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

socket

a *host-local*, *application-created/owned*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another (remote or local) application process

Sockets

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

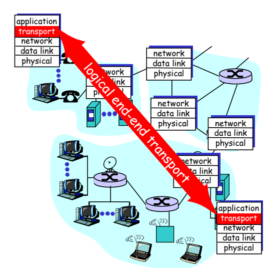


Languages and Platforms

- Socket API is available for many languages on many platforms:
  - C, Java, Perl, Python,...
  - *nix, Windows,...
- Socket Programs written in any language and running on any platform can communicate with each other!
- Client and server must agree on the type of socket, the server port number and the protocol

Transport services and protocols

- provide *logical communication* between app' processes running on different hosts
- transport protocols run in end systems
- transport vs network layer services:
- *network layer:* data transfer between **end systems**
- *transport layer:* data transfer between **processes**
  - relies on, enhances, network layer services



Services provided by Internet transport protocols

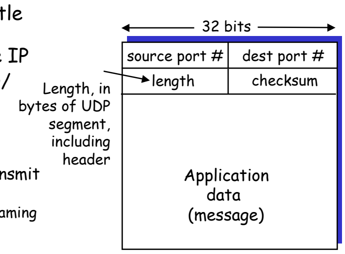
TCP service:
- *connection-oriented:* setup required between client, server
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not providing:* timing, minimum bandwidth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

Q: why bother? Why is there a UDP?

UDP

- UDP adds very little functionality (or overhead) to bare IP
- Adds multiplexing/demultiplexing
- other UDP uses (why?):
  - DNS: small, retransmit if necessary
  - often used for streaming multimedia apps
    - Loss tolerant
    - rate sensitive

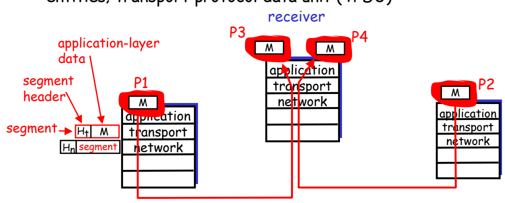


UDP segment format

Process-to-Process Message Delivery

Goal : Deliver application data to correct process (and more particularly to the right socket)

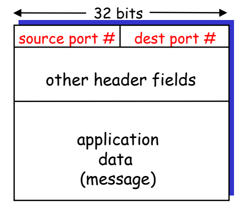
Segment - unit of data exchanged between transport layer entities; transport protocol data unit (TPDU)



Multiplexing/demultiplexing

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

Demultiplexing:
Stream of incoming data into one machine separated into smaller streams destined for individual processes



TCP/UDP segment format

Demultiplexing based on IP addresses of sender and and port numbers of both sender and receiver
- Can distinguish traffic coming to same port but part of separate conversations (like multiple client connections to a web server)

TCP adds functionality

- TCP adds lots of functionality over bare IP and over UDP
  - Still has multiplexing/demultiplexing
  - Adds reliable, in-order delivery
  - Adds flow control and congestion control
- How can you guarantee that other side gets "A B C D E" when network could:
  - Lose data "A B D E"
  - Duplicate data "A B C C D E"
  - Corrupt data "A B X D E"
  - Reorder data "A C D E B"
  - Or all of the above!

Common Sense

- Consider faxing a document with flaky machine
  - Can't talk to person on the other side any other way
- What would you do to make sure they got the transmission?
  - Number the pages - so receiver can put them in order/detect duplicates/detect losses
  - Need feedback from the receiver!!!
  - Resend data that is missing or if don't hear from receiver
- Put some info on cover sheet that lets person verify fax info (summarize info like checksum)
- What if it is a really big document? Receiver might like to be able to tell you send first 10 pages then 10 more...

TCP Connection Management

Recall: TCP sender, receiver establish "connection" before exchanging data segments
- initialize TCP variables:
  - seq. #s
  - buffers, flow control info (e.g. `RcvWindow`)
- *client:* connection initiator

```
Socket clientSocket = new
Socket("hostname","port
number");
```

- *server:* contacted by client

```
Socket connectionSocket =
welcomeSocket.accept();
```

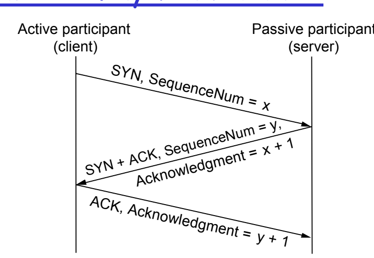
Three way handshake:

Step 1: client end system sends TCP SYN control segment to server
- specifies initial seq #

Step 2: server end system receives SYN, replies with SYNACK control segment
- ACKs received SYN
- allocates buffers
- specifies server-> receiver initial seq. #

Step 3: client acknowledges servers initial seq. #

Three-Way Handshake

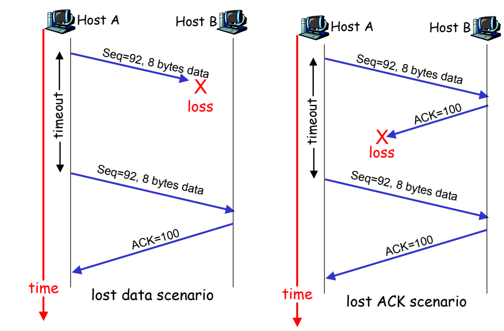


Note: SYNs take up a sequence number even though no data bytes

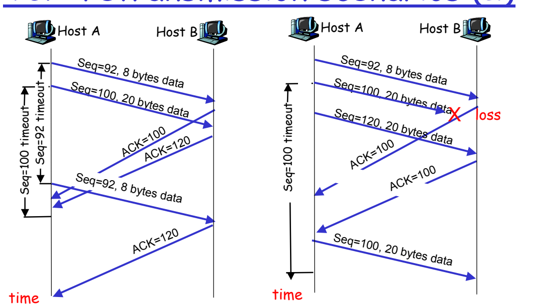
Timeout and Retransmission

- Receiver must acknowledge receipt of all packets
- Sender sets a timer if acknowledgement has not arrived before timer expires then sender will retransmit packet
- Adaptive retransmission: timer value computed as a function of average round trip times and variance

TCP: retransmission scenarios (1)



TCP: retransmission scenarios (2)



premature timeout, cumulative ACKs

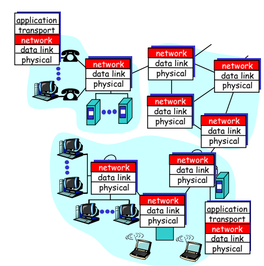
Duplicate ACK, fast retransmit (really need 3 dup acks before fast retransmit)

Network layer functions

- transport packet from sending to receiving hosts
- network layer protocols in *every* host, router (Recall transport layer is end-to-end)

three important functions:

- *path determination:* route taken by packets from source to dest. *Routing algorithms*
- *switching:* move packets from router's input to appropriate router output
- *call setup:* some network architectures (e.g. telephone, ATM) require router call setup along path before data flow

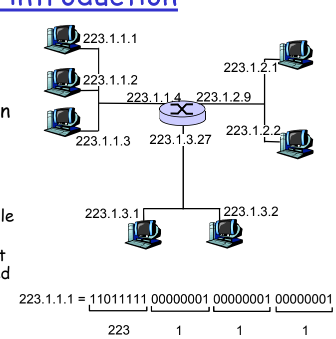


Internet Protocol

- The Internet is a network of **heterogeneous** networks:
  - using different technologies (ex. different maximum packet sizes)
  - belonging to different administrative authorities (ex. Willing to accept packets from different addresses)
- Goal of IP: interconnect all these networks so can send end to end without any knowledge of the intermediate networks
- Routers, switches, bridges: machines to forward packets between heterogeneous networks

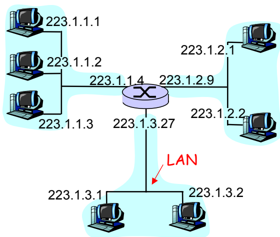
IP Addressing: introduction

- IP address: 32-bit identifier for host, router *interface*
- *interface:* connection between host and physical link
  - router's must have multiple interfaces
  - host may have multiple interfaces
  - IP addresses (unicast addresses) associated with interface, not host, router



IP Addressing

- IP address:
  - 32 bits
  - network part (high order bits)
  - host part (low order bits)
  - Defined by class of IP address?
  - Defined by subnet mask
- *What's a network ?* (from IP address perspective)
  - device interfaces with same network part of IP address
  - can physically reach each other without intervening router

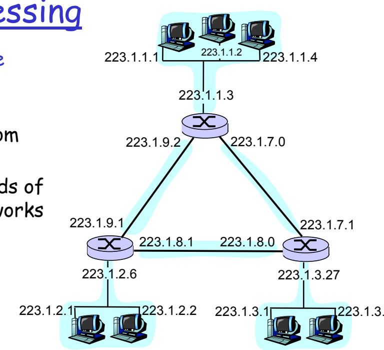


network consisting of 3 IP networks (223.1.1, 223.1.2, 223.1.3)

IP Addressing

How to find the networks?

- Detach each interface from router, host
- create "islands of isolated networks



Interconnected system consisting of six networks

IP Addresses (Classes)

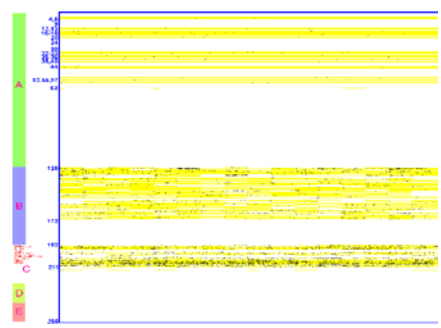
given notion of "network", let's re-examine IP addresses:

"class-full" addressing

	class	format	range
	A	0 network / host	1.0.0.0 to 127.255.255.255
Unicast	B	10 network / host	128.0.0.0 to 191.255.255.255
	C	110 network / host	192.0.0.0 to 223.255.255.255
Multicast	D	1110 multicast address	224.0.0.0 to 239.255.255.255
Reserved	E	1111 reserved	240.0.0.0 to 255.255.255.255

← 32 bits →

IP Address Space Allocation



CAIDA 1998

IP addressing: CIDR

- classful addressing:
  - inefficient use of address space, address space exhaustion
  - e.g., class B net allocated enough addresses for 65K hosts, even if only 2K hosts in that network
- CIDR: Classless InterDomain Routing
  - network portion of address of arbitrary length
  - address format: a.b.c.d/x, where x is # bits in network portion of address

← network part → ← host part →

11001000 00010111 00010000 00000000

200.23.16.0/23

Recall: How to get an IP Address?

- Answer 1: Normally, answer is get an IP address from your upstream provider
  - This is essential to maintain efficient routing!
- Answer 2: If you need lots of IP addresses then you can acquire your own block of them.
  - IP address space is a scarce resource - must prove you have fully utilized a small block before can ask for a larger one and pay $$ (Jan 2002 - $2250/year for /20 and $18000/year for a /14)

How to get lots of IP Addresses? Internet Registries

RIPE NCC (Riseaux IP Europiens Network Coordination Centre) for Europe, Middle-East, Africa

APNIC (Asia Pacific Network Information Centre) for Asia and Pacific

ARIN (American Registry for Internet Numbers) for the Americas, the Caribbean, sub-saharan Africa

Note: Once again regional distribution is important for efficient routing!

Can also get Autonomous System Numbers (ASNs) from these registries

Classful vs Classless

- Class A = /8
- Class B = /16
- Class C = /24

IP addresses: how to get one? revisted

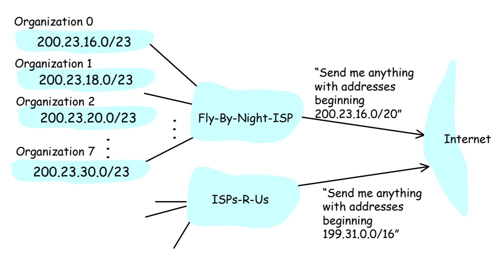
Network (network portion):

- get allocated portion of ISP's address space:

ISP's block	11001000 00010111 00010000 00000000	200.23.16.0/20
Organization 0	11001000 00010111 00010000 00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010 00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100 00000000	200.23.20.0/23
...	....	....
Organization 7	11001000 00010111 00011110 00000000	200.23.30.0/23

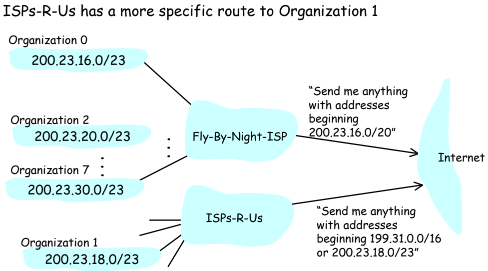
Hierarchical addressing: route aggregation

Hierarchical addressing allows efficient advertisement of routing information:



Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1



IP Address Allocation

- CIDR is great but must work around existing allocations of IP address space
  - Company 1 has a /20 allocation and has given out sub portions of it to other companies
  - University has a full class B address
  - Company 2 has a /23 allocation from some other class B
  - ALL use the same upstream ISP – that ISP must advertise routes to all these blocks that cannot be described with a simple CIDR network ID and mask!
- Estimated reduction in routing table size with CIDR
  - If IP addresses reallocated, CIDR applied to all, IP addresses reallocated based on geographic and service provider divisions that current routing tables with 10000+ entries could be reduced to 200 entries [Ford, Rekhter and Brown 1993]
  - How stable would that be though? Leases for all?

Current Allocation

- Interesting to exam current IP address space allocation (who has class A's ? Etc)
  - Who has A's?
  - Computer companies around during initial allocation (IBM, Apple)
  - Universities (Stanford, MIT)
  - CAIDA has info on complete allocation

Routing

- IP Routing – each router is supposed to send each IP datagram one step closer to its destination
- How do they do that?
  - Hierarchical Routing – in ideal world would that be enough?   Well its not an ideal world
  - Other choices
    - Static Routing
    - Dynamic Routing
      - Before we cover specific routing protocols we will cover principles of dynamic routing protocols

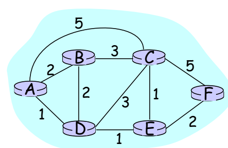
Routing

Routing protocol

Goal: determine "good" path (sequence of routers) thru network from source to dest.

Graph abstraction for routing algorithms:

- graph nodes are routers
- graph edges are physical links
  - link cost: delay, $ cost, or congestion level



- "good" path:
  - typically means minimum cost path
  - other definitions possible

Routing Algorithm classification: Static or Dynamic?

Choice 1: Static or dynamic?

Static:

- routes change slowly over time
- Configured by system administrator
- Appropriate in some circumstances, but obvious drawbacks (routes added/removed? sharing load?)
- Not much more to say?

Dynamic:

- routes change more quickly
  - periodic update
  - in response to link cost changes

Routing Algorithm classification: Global or decentralized?

Choice 2, if dynamic: global or decentralized information?

Global:

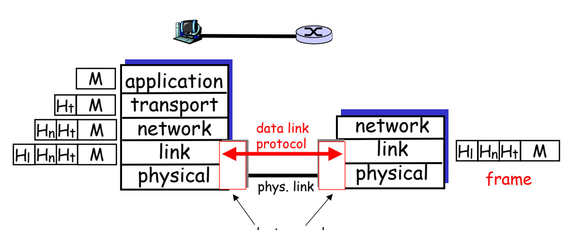
- all routers have complete topology, link cost info
- "link state" algorithms

Decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors (gossip)
- "distance vector" algorithms

Link Layer: setting the context

- two *physically connected* devices:
  - host-router, router-router, host-host
- unit of data: *frame*



Link Layer Services

- Framing, link access:
  - encapsulate datagram into frame, adding header, trailer
  - implement channel access if shared medium,
  - 'physical addresses' used in frame headers to identify source, dest
    - different from IP address!
- Reliable delivery between two physically connected devices:
  - Reliable delivery over an unreliable link (like TCP but done at link layer)
  - 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?

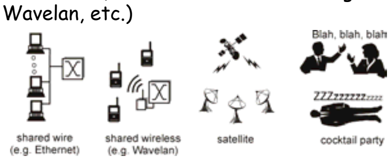
Link Layer Services (more)

- Flow Control:
  - pacing between sender and receivers
- *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

Multiple Access Links and Protocols

Three types of "links":

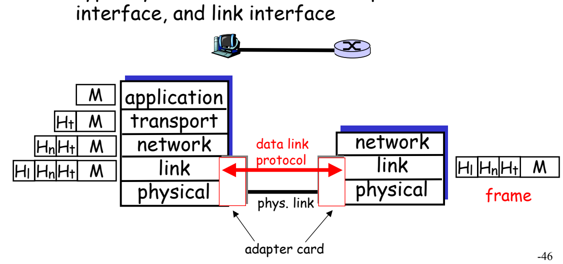
- broadcast (shared wire or medium; e.g, Ethernet, Wavelan, etc.)



- point-to-point (single wire, e.g. PPP, SLIP)
- switched (e.g., switched Ethernet, ATM etc)

Link Layer: Implementation

- implemented in "adapter"
  - e.g., PCMCIA card, Ethernet card
  - typically includes: RAM, DSP chips, host bus interface, and link interface



Multiple Access protocols

- single shared communication channel
- two or more simultaneous transmissions by nodes: interference
  - only one node can send successfully at a time
- *multiple access protocol:*
  - distributed algorithm that determines how stations share channel, i.e., determine when station can transmit
- claim: humans use multiple access protocols all the time

CSMA: Carrier Sense Multiple Access

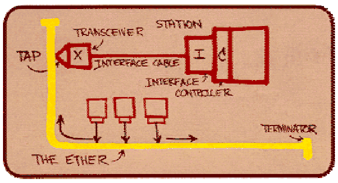
CSMA: listen before transmit:

- If channel sensed idle: transmit entire pkt
- If channel sensed busy, defer transmission
  - Persistent CSMA: retry immediately with probability p when channel becomes idle (may cause instability)
  - Non-persistent CSMA: retry after random interval
- human analogy: don't interrupt others!

Ethernet

"dominant" LAN technology:

- cheap $20 for 100Mbs!
- first widely used LAN technology
- Simpler, cheaper than token LANs and ATM
- Kept up with speed race: 10, 100, 1000 Mbps
- Uses CSMA with collision detection



Metcalfe's Ethernet sketch