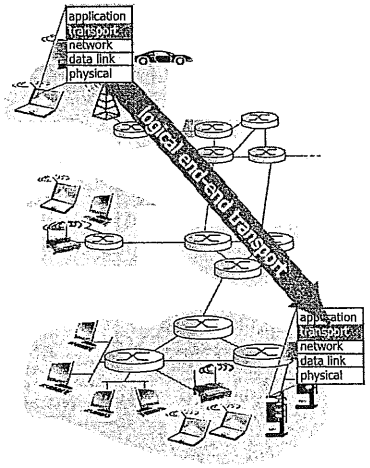
Transport Layer

- Transport Layer Services
- Multiplexing and Demultiplexing
- Connectionless Transport: UDP
- Connection-Oriented Transport: TCP
- TCP Congestion Control

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 apps

– Internet : TCP and UDP

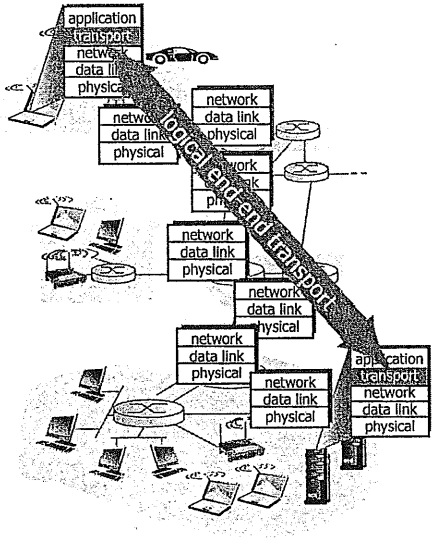


Internet Transport-Layer Protocols

- TCP - Reliable, in-order delivery
  - Congestion control
  - Flow control

– Connection setup

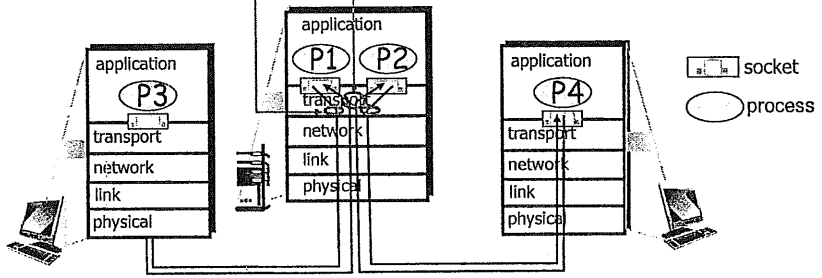
- UDP - Unreliable, unordered delivery
  - No-frills extension of "best-effort" IP
- Services not available
  - Delay guarantees



Multiplexing/Demultiplexing

multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:
use header info to deliver received segments to correct socket

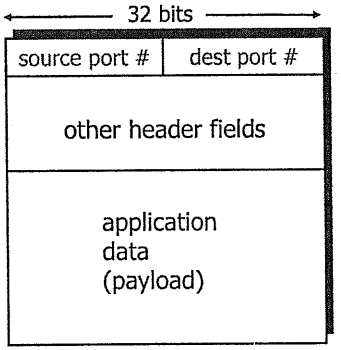


you need this data to find the right socket in a machine ← 4 unique identifiers:
- source IP address
- dest IP address
- source port #
- dest port #

How Demultiplexing Works

5

- Host receives IP datagrams
  - Each datagram has source IP address, destination IP address
  - Each datagram carries one transport-layer segment
  - Each segment has source, destination port number



TCP/UDP segment format

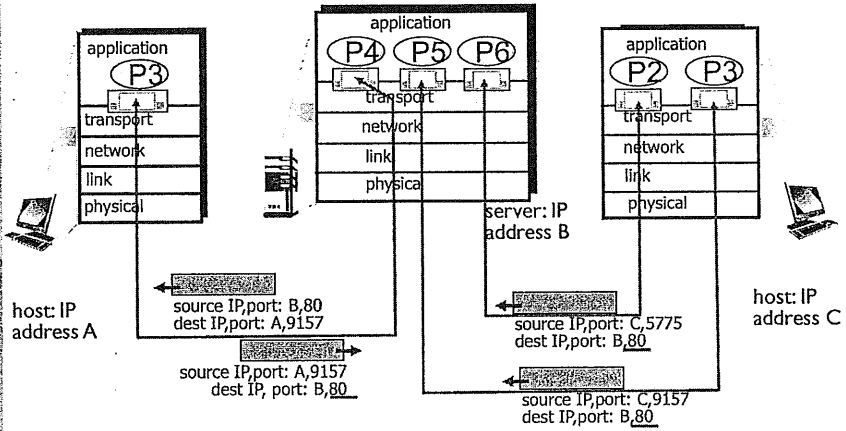
Connection-Oriented Demux

6

- TCP socket identified by 4-tuple
  - Source IP address
  - Source port number
  - Dest IP address
  - Dest port number
- Receiver uses all four values to direct segment to appropriate socket
  - Demultiplexing
- Server host may support many simultaneous TCP sockets
  - Each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - Non-persistent HTTP will have a different socket for each request

Connection-Oriented Demux Example

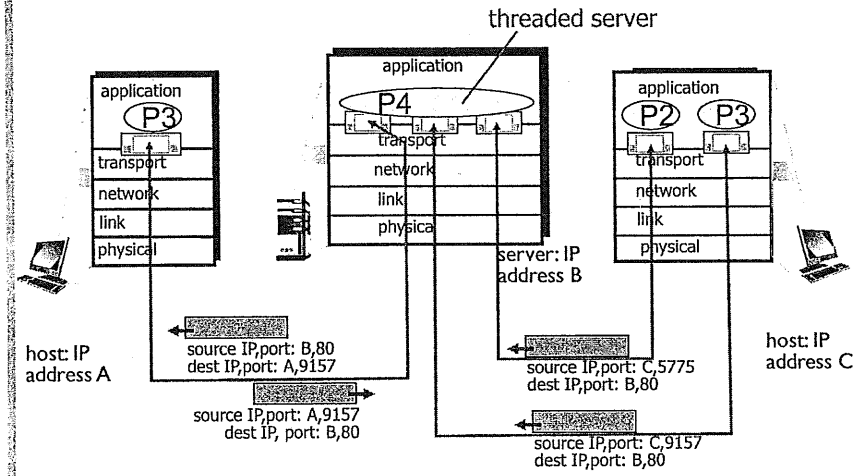
7



three segments, all destined to IP address: B, dest port: 80 are demultiplexed to *different* sockets

Connection-Oriented Demux Example II

8

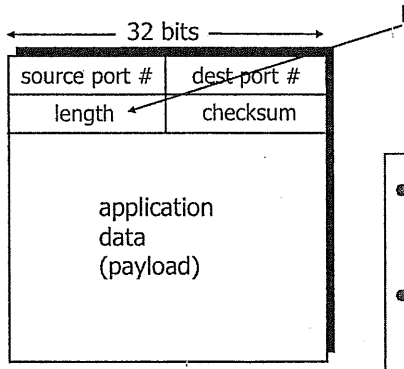


Q: Why use threads?
A: A *thread* is a lightweight sub-process

User Datagram Protocol (UDP)

- "No frills," "bare bones" Internet transport protocol
- "Best effort" service, UDP segments may be
  - Lost
  - Delivered out-of-order to app
- Connectionless
  - Each UDP segment handled independently of others
- Uses
  - Streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP

UDP Header



UDP segment format

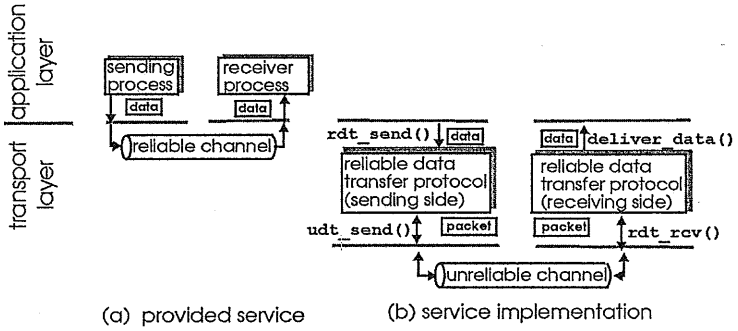
Why is there a UDP?

- No connection establishment
  - Which can add delay
- Simple - No connection state at sender, receiver
- Small header size
- No congestion control
  - UDP can blast away as fast as desired

(can pump as much data into network as possible)

Principles of Reliable Data Transfer

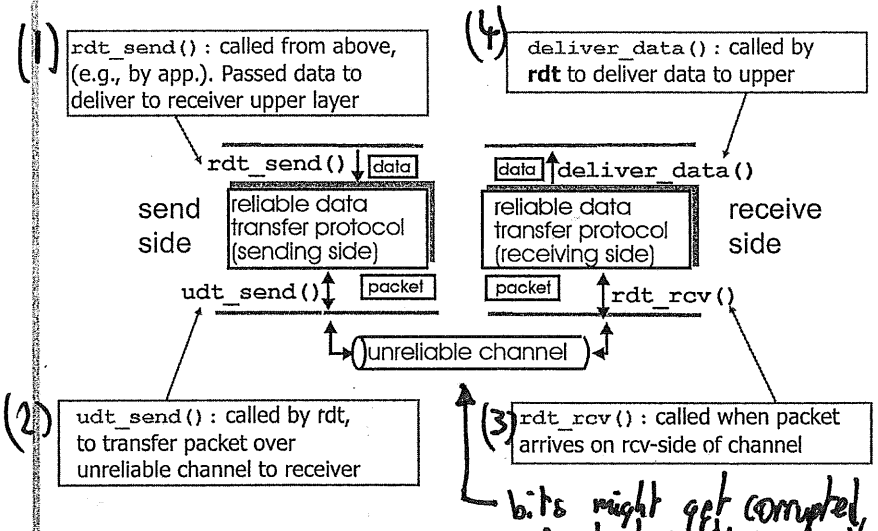
- Important in application, transport, link layers



(a) provided service (b) service implementation

- Characteristics of unreliable channel will determine complexity of *reliable data transfer* protocol (rdt)

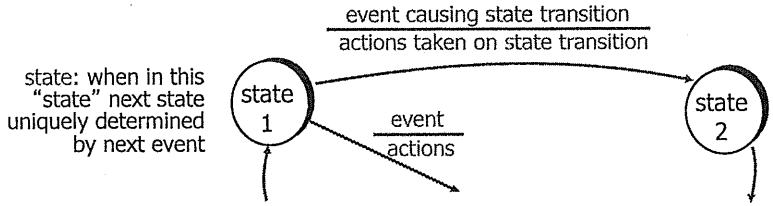
Reliable Data Transfer (RTD)



bits might get corrupted, packets might get lost etc.
∴ unreliable

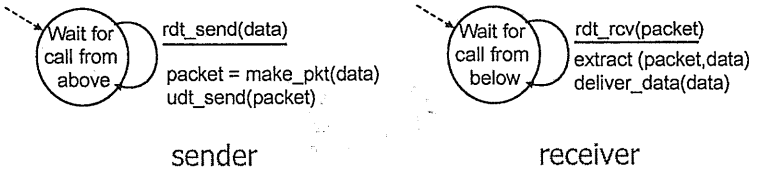
RDT

- We will incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- Consider only unidirectional data transfer
  - But control info will flow on both directions!
- Use finite state machines (FSM) to specify sender, receiver



rdt1.0: Reliable Transfer Over a Reliable Channel

- Underlying channel perfectly reliable
  - No bit errors
  - No loss of packets
- Separate FSMs for sender and receiver
  - Sender sends data into underlying channel
  - Receiver reads data from underlying channel

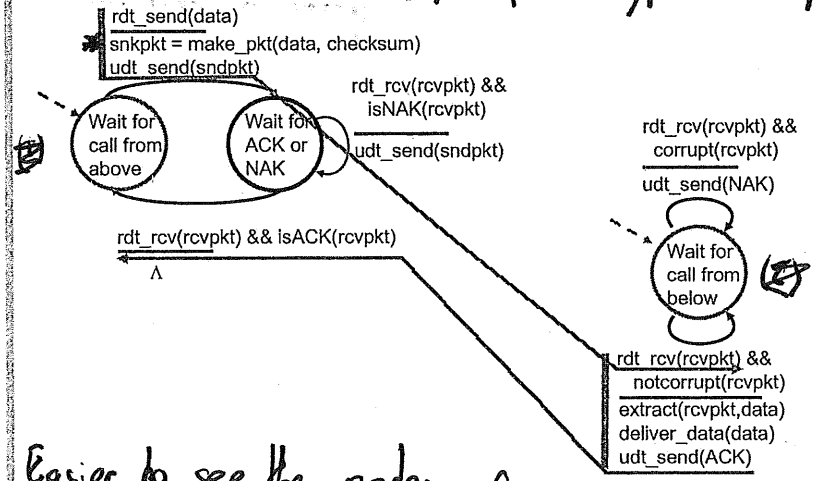


rdt2.0: Channel with Bit Errors

- Underlying channel may flip bits in packet
  - Checksum to detect bit errors
- Q: How to recover from errors?
  - Acknowledgements (ACKs)
    - Receiver explicitly tells sender that pkt received OK
- Negative Acknowledgements (NAKs)
  - Receiver explicitly tells sender that pkt had errors
    - Sender retransmits pkt on receipt of NAK
- New mechanisms in rdt2.0 (beyond rdt1.0)
  - Error Detection
  - Feedback
    - Control msgs (ACK, NAK) from receiver to sender

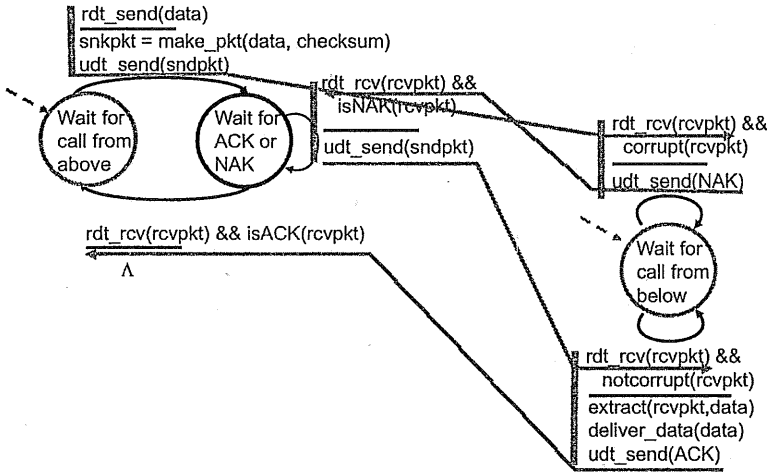
rdt2.0: Operation with No Errors

*sndpkt (:rs a typo, not snkpkt)



Easier to see the order of how this all works from the PowerPoint presentation

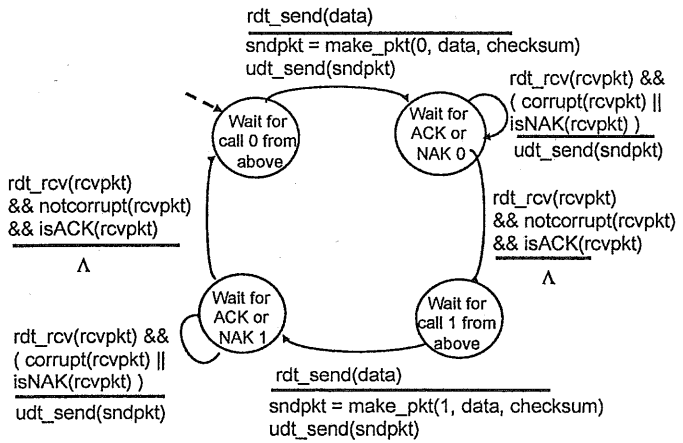
rdt2.0: Error Scenario



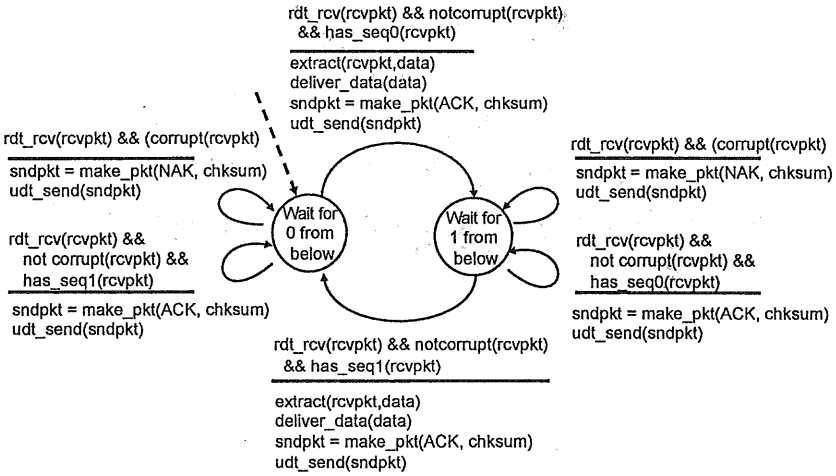
rdt2.0: Fatal Flaw

- What happens if ACK/NAK corrupted?
  - Sender does not know what happened at receiver!
  - Cannot just retransmit
    - possible duplicate
- Handling Duplicates
  - Sender retransmits current pkt if ACK/NAK corrupted
  - Sender adds sequence number to each pkt
  - Receiver discards (does not deliver up) duplicate pkt
- Stop and Wait
  - Sender sends one packet, then waits for receiver response

rdt2.1: Sender Handles Garbled ACK/NAKs



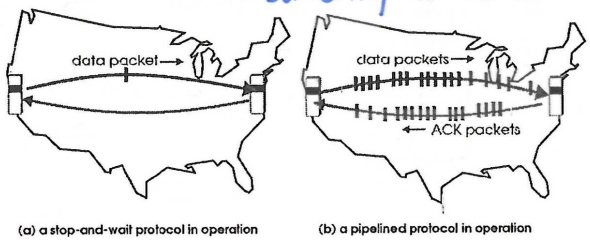
rdt2.1: Receiver Handles Garbled ACK/NAKs



Pipelined Protocols

- Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

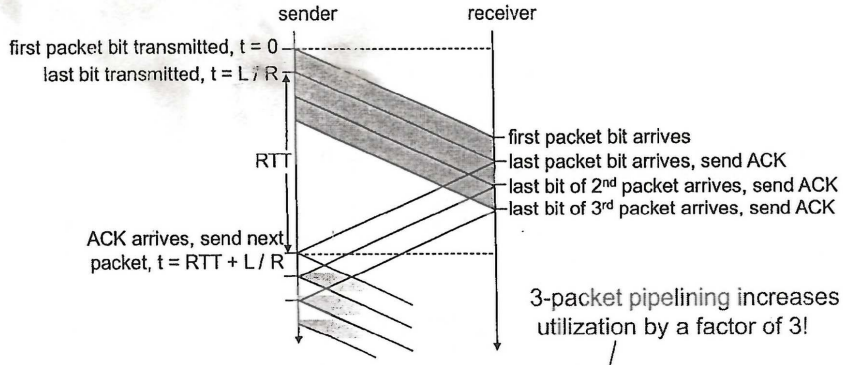
- Range of sequence numbers must be increased
- Buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

- Two generic forms of pipelined protocols
  - Go-Back-N and Selective Repeat

Pipelining: Increased Utilization



3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

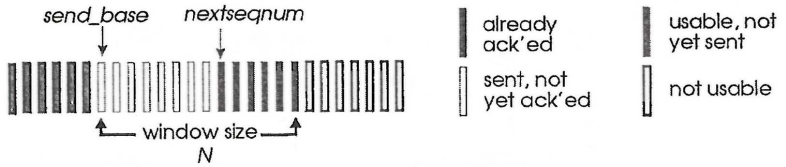
Pipelined Protocols: Overview

- Go-Back-N (GBN)
  - Sender can have up to *N* unacked packets in pipeline
  - Receiver can send cumulative acks
  - Sender has timer for oldest unacked packet
    - When timer expires, retransmit all unacked packets
- Selective Repeat (SR)
  - Sender can have up to *N* unacked packets in pipeline
    - Recv sends individual ack for each packet
  - Sender maintains timer for each unacked packet
    - When timer expires, retransmit only that unacked packet

(less bandwidth used in SR, may be preferable over GBN)

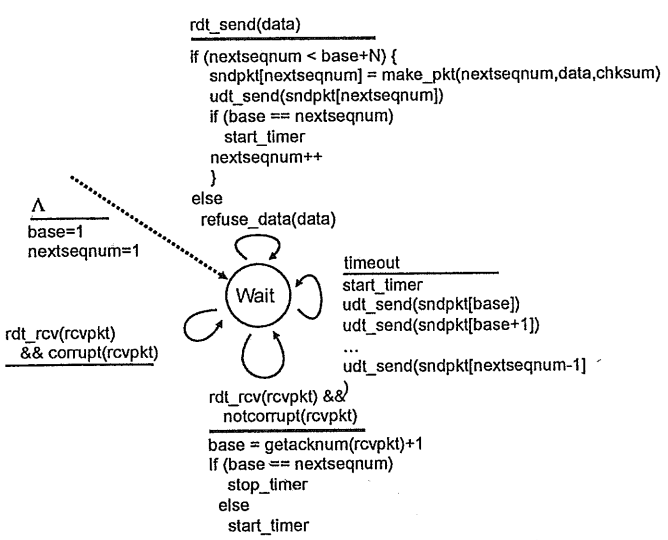
Go-Back-N: Sender

- *k*-bit seq # in pkt header
  - "Window" of up to *N*, consecutive unacked pkts allowed

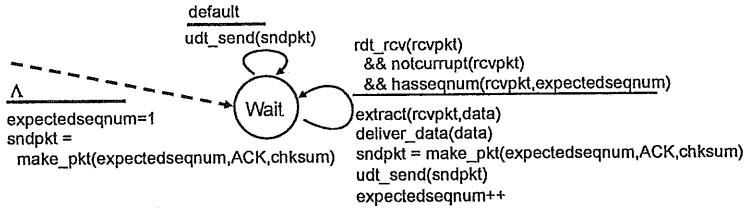


- Timer for oldest in-flight pkt
  - Timeout (n): Retransmit packet n and all higher seq # pkts in window

GBN: Sender Extended FSM

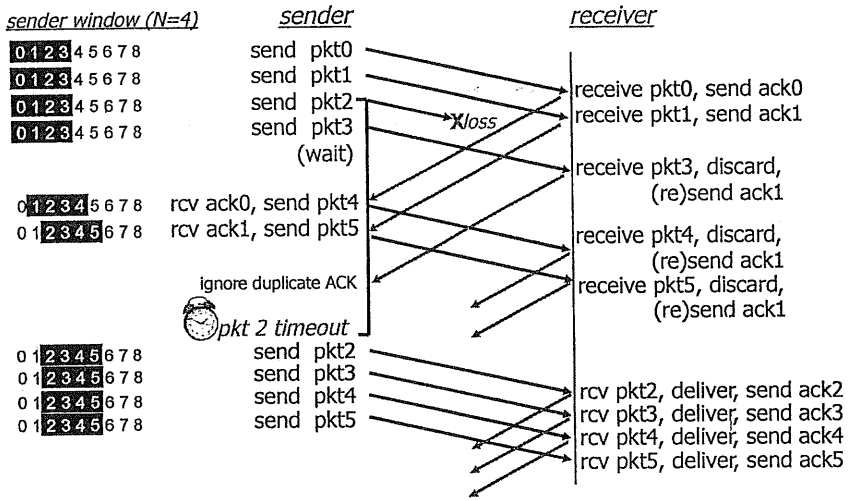


GBN: Receiver Extended FSM



- ACK-only: always send ACK for correctly-received pkt with highest in-order seq #
  - Need only remember **expectedseqnum**
- Out-of-order pkt
  - Discard: no receiver buffering!

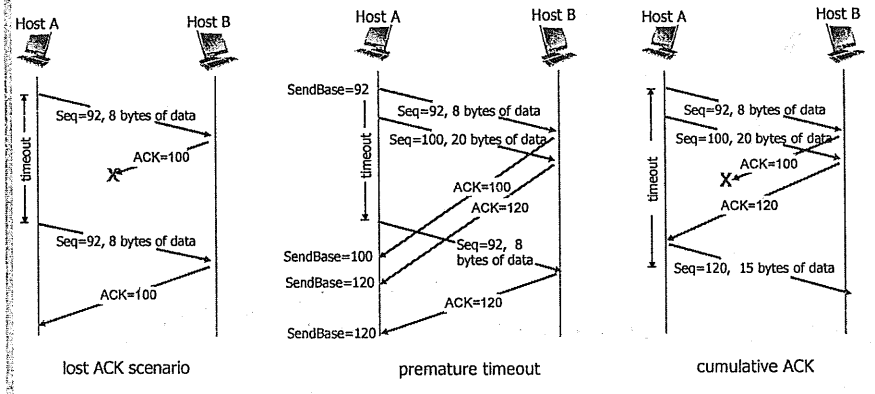
GBN in Action



Selective Repeat

- Receiver individually acknowledges all correctly received pkts
  - Buffers pkts, as needed, for eventual in-order delivery to upper layer
- Sender only resends pkts for which ACK not received
- Sender window
  - *N* consecutive seq #'s
  - Limits seq #*s* of sent, unACKed pkts

TCP: Retransmission Scenarios

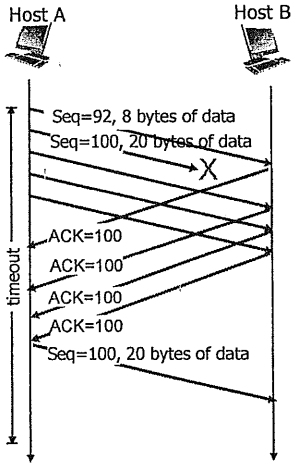


TCP ACK Generation

Event at receiver	*TCP receiver action*
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send *duplicate ACK*, indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

TCP 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
- TCP Fast Retransmit
  - If sender receives 3 ACKs for same data - "Triple Duplicate ACKs"
    - Resend unacked segment with smallest sequence number
  - Likely that unacked segment lost, so do not wait for timeout

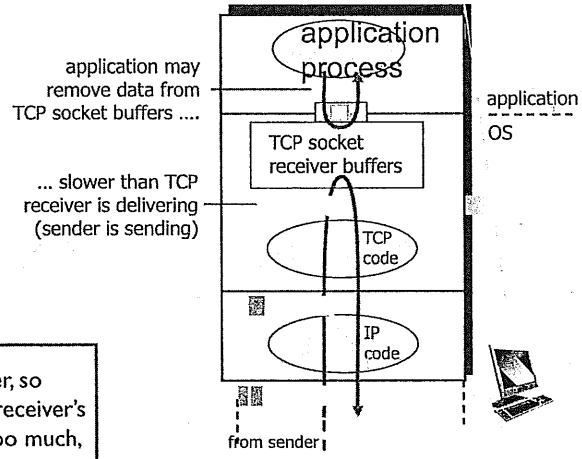


fast retransmit after sender receipt of triple duplicate ACK

TCP Flow Control

flow control

Receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast



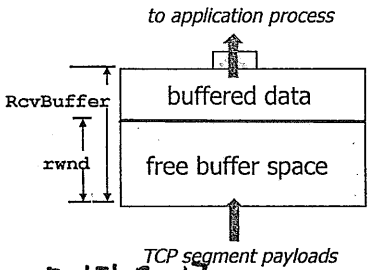
receiver protocol stack

TCP Flow Control II

- Receiver "advertises" free buffer space by including *rwnd* (receive window) value in TCP header of receiver-to-sender segments
  - RcvBuffer size set via socket options
    - Typical default is 4096 bytes
  - Many operating systems auto-adjust RcvBuffer
- Sender limits amount of unacked ("in-flight") data to receiver's *rwnd* value

rwnd = RcvBuffer - [LastByteRcvd - LastByteRead]

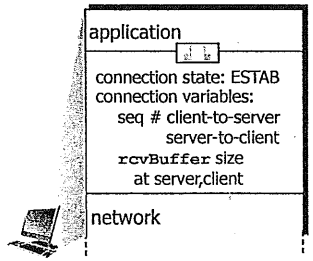
- Guarantees receivebuffer will not overflow



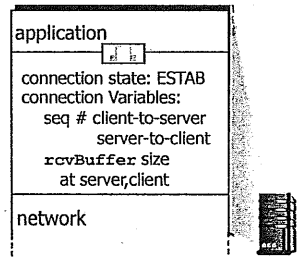
receiver-side buffering

Connection Management

- Before exchanging data, sender/receiver "handshake"
  - Agree to establish connection (each knowing the other willing to establish connection)
  - Agree on connection parameters



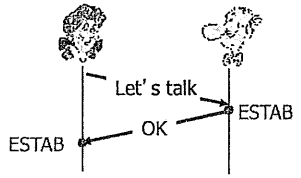

```
Socket clientSocket =
  newSocket("hostname","port
  number");
```

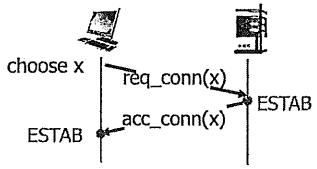



```
Socket connectionSocket =
  welcomeSocket.accept();
```


Agreeing to Establish a Connection

2-way handshake:

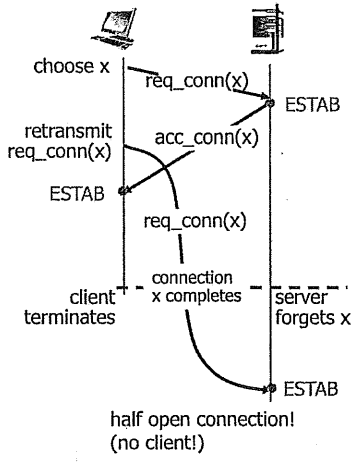


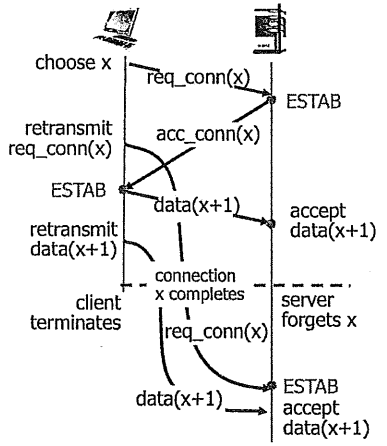


Q: Will 2-way handshake always work in network?

- Variable delays
- Message reordering
- Can't "see" other side

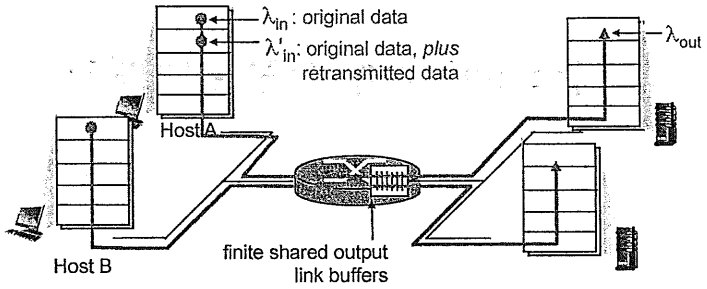
2-way Handshake Failure Scenarios





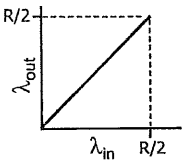
Causes/Costs of Congestion: Scenario 2

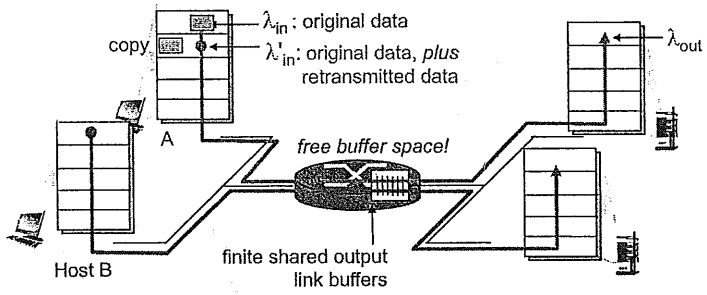
- One router, *finite* buffers
  - Pkts will be dropped
- Sender retransmission of timed-out packet
  - Application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - Transport-layer input includes retransmissions : $\lambda'_{in} \geq \lambda_{in}$



Causes/Costs of Congestion: Scenario 2

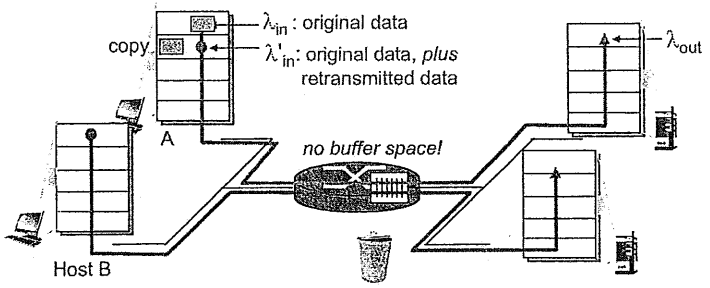
- Idealization: *perfect knowledge*
  - Sender sends only when router buffers available





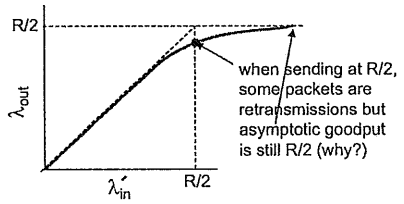
Causes/Costs of Congestion: Scenario 2

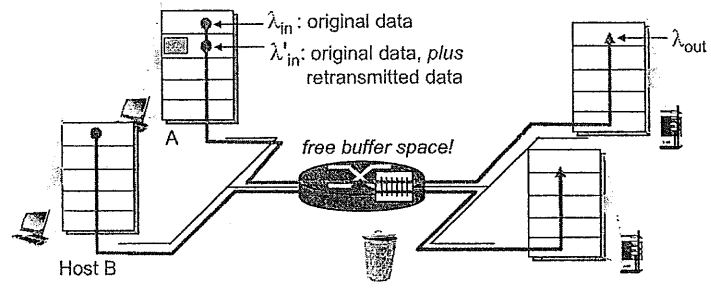
- Idealization: *known loss*
  - Packets can be lost, dropped at router due to full buffers



Causes/Costs of Congestion: Scenario 2

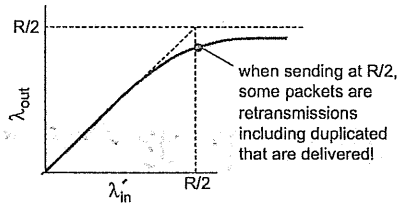
- Idealization: *known loss*
  - Packets can be lost, dropped at router due to full buffers
  - Sender only resends if packet known to be lost

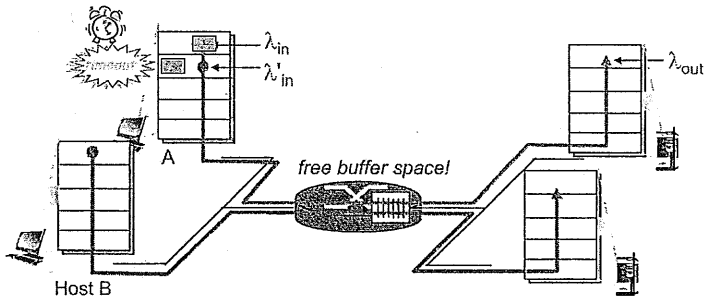




Causes/Costs of Congestion: Scenario 2

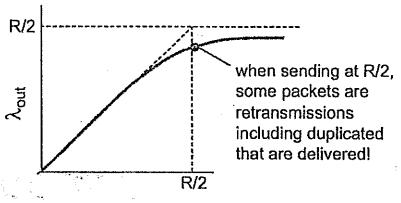
- Realistic: *duplicates*
  - Packets can be lost, dropped at router due to full buffers





Causes/Costs of Congestion: Scenario 2

- Realistic: *duplicates*
  - Packets can be lost, dropped at router due to full buffers
  - Sender times out prematurely, sending *two* copies, both of which are delivered



- "Costs of Congestion"
  - More work (retrans) for given "goodput"
  - Unneeded retransmissions

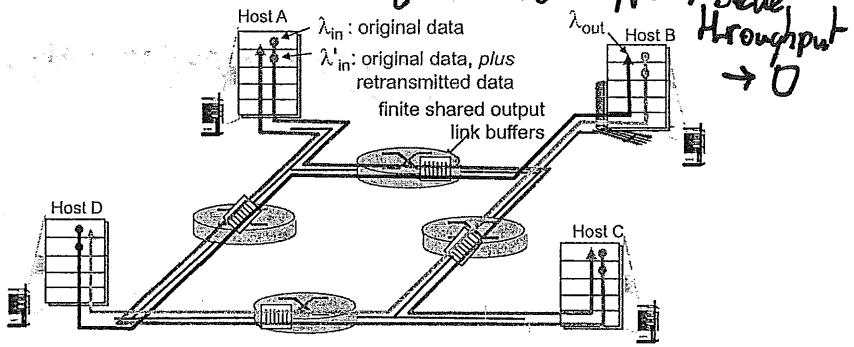
• Link carries multiple copies of pkt
• Decreasing goodput

Causes/Costs of Congestion: Scenario 3

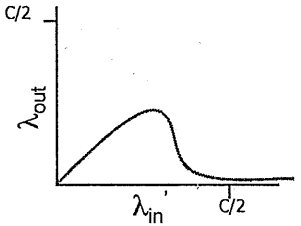
- Four Senders
- Multihop Paths
- Timeout/Retransmit

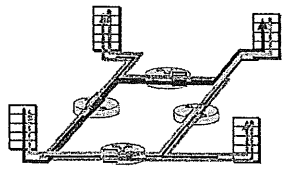
Q: What happens as λ_{in} and λ_{in}' increase?

A: As red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput → 0



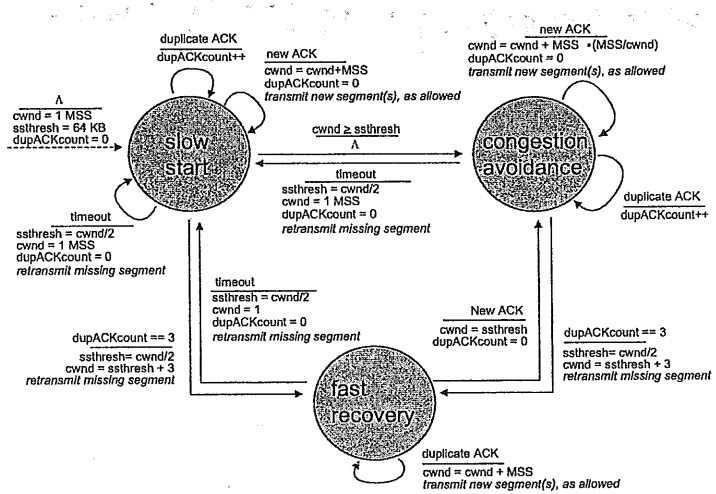
Causes/Costs of Congestion: Scenario 3





- Another "cost" of congestion
  - When packet dropped, any upstream transmission capacity used for that packet is wasted!

Summary: TCP Congestion Control



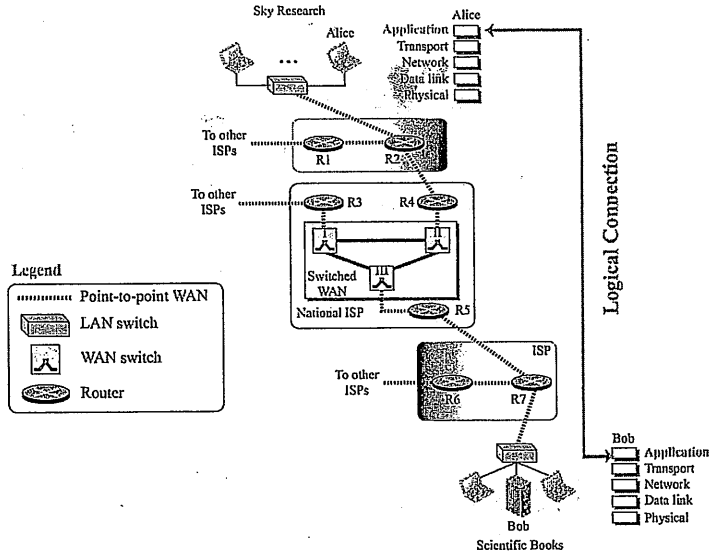
Securing TCP

- TCP & UDP
  - No encryption
  - Cleartext passwds sent into socket traverse Internet in cleartext
- SSL
  - Provides encrypted TCP connection
  - Data integrity
- SSL is at app layer
  - Apps use SSL libraries, which "talk" to TCP
- SSL socket API
  - Cleartext passwds sent into socket traverse Internet encrypted

Application Layer Protocols

- Network Application Architectures
- WWW & HTTP
- Simple Mail Transfer System (SMTP)
- Domain Name System (DNS)
- P2P Applications
- Web Applications

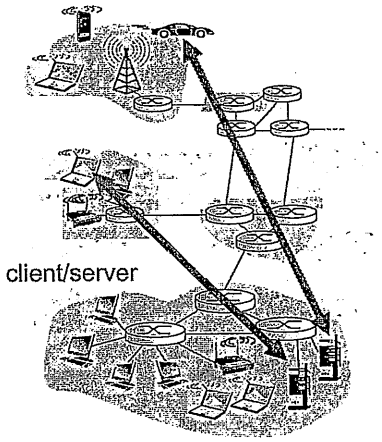
Connecting a Network Application



Connecting a Network App

- Write programs that
  - Run on (different) *end systems*
  - Communicate over network
  - e.g., web server software communicates with browser software
- No need to write software for network-core devices
  - Network-core devices do not run user applications
  - Applications on end systems allows for rapid app development, propagation
- Possible structure of applications
  - Client-Server
  - Peer-to-Peer (P2P)

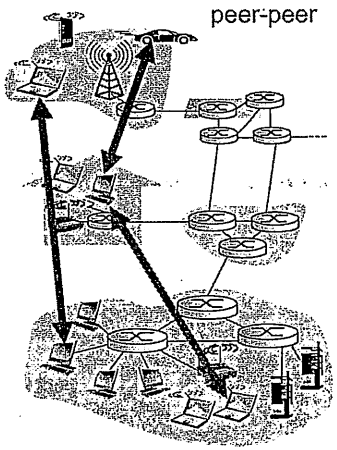
Client-Server Architecture



- Server
  - Always-on host
  - Permanent IP address
  - Data centers for scaling
- Clients
  - Communicate with server
  - May be intermittently connected
  - May have dynamic IP addresses
  - Do not communicate directly with each other

P2P Architecture e.g. Bittorrent

- *No* always-on server
- Arbitrary end systems directly communicate
- Peers request service from other peers, provide service in return to other peers
  - *Self scalability* – new peers bring new service capacity, as well as new service demands
- Peers are intermittently connected and change IP addresses
  - Complex management



Processes

- 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*

clients, servers

client process: process that initiates communication

server process: process that waits to be contacted

- Applications with P2P architectures have client processes & server processes

What Transport Services Does and App Need?

Data Loss

- Some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- Other apps (e.g. audio) can tolerate some loss

Throughput

- Some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- Other apps ("elastic apps") make use of whatever throughput they get

Timing

- Some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Security

- Encryption, authentication, data integrity, …

Transport Service Requirements: Common Apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100's msec yes and no

WWW and HTTP

- A web page consists of objects
  - An object can be HTML file, JPEG image, Java applet, audio file, ...
- A web page consists of base HTML file which includes several referenced objects

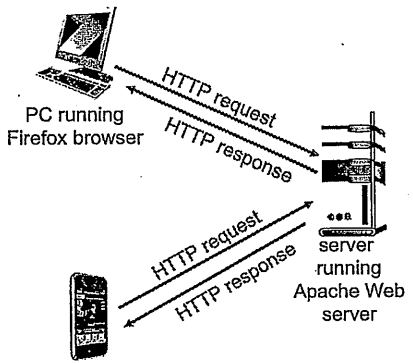
-Each object is addressable by a Uniform Resource Locator (URL)

`www.someschool.edu/someDept/pic.gif`

host name | path name

Hyper Text Transfer Protocol (HTTP)

- HTTP uses TCP
  - Client initiates TCP connection (creates socket) to server, port 80
  - Server accepts TCP connection from client
  - HTTP msgs exchanged between Web browser (HTTP client) and Web server (HTTP server)
  - TCP connection closed



- HTTP is "stateless"

-Server maintains no information about past client requests - why?

HTTP Connections

- Non-persistent HTTP
  - At most one object sent over TCP connection
  - Connection then closed
  - Downloading multiple objects requires multiple connections
- Persistent HTTP
  - Multiple objects can be sent over single TCP connection between client and server
    - What is the advantage?
- Default mode of HTTP

-Persistent connections with pipelining

Non-persistent HTTP

suppose user enters URL:
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Non-persistent HTTP II

4. HTTP server closes TCP connection

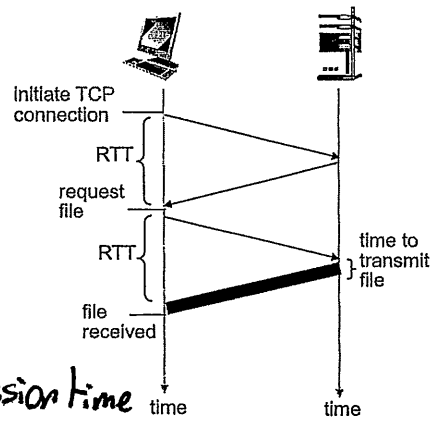
5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

time

6. Steps 1-5 repeated for each of 10 jpeg objects

Non-persistent HTTP: Response Time

- One RTT to initiate TCP connection
- One RTT for HTTP request, and first few bytes of HTTP response to return
- File transmission time
- Non-persistent HTTP response time =



- 2RTT + File transmission time

Exercise: What is the typical RTT for persistent HTTP for a file with ten objects?

Persistent HTTP

Non-persistent HTTP issues

- Requires 2 RTTs per object
- OS overhead for each TCP connection
- Browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- Server leaves connection open after sending response
- Subsequent HTTP messages between same client/server sent over open connection
- Client sends requests as soon as it encounters a referenced object
- As little as one RTT for all the requested objects

HTTP Request Message

- Two types of HTTP messages
  - *request, response*
- HTTP request message
  - ASCII (human-readable format)

carriage-return
line-feed character

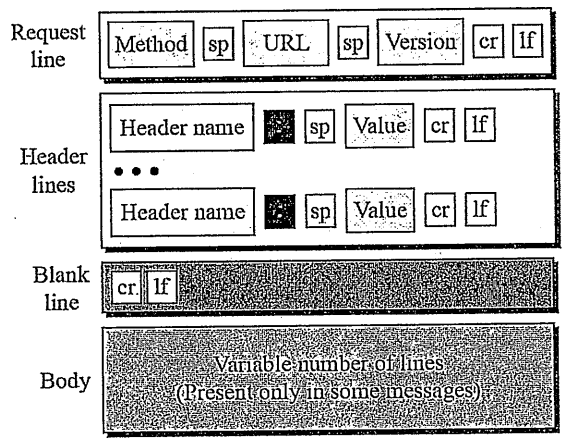
request line (GET, POST, HEAD commands)

header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP Request Msg Format



Request message

HTTP/1.1 Method Types

- GET, POST, HEAD
- PUT
  - Uploads file in entity body to path specified in URL field
    - Used in conjunction with Web publishing tools
- DELETE
  - Deletes file specified in the URL field
- Uploading Form Input
  - POST method
    - Web page often includes form input
    - Input is uploaded to server in entity body
  - URL method
    - Uses GET method
    - Input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

Exercise: What are the advantages and disadvantages of the GET & POST methods?

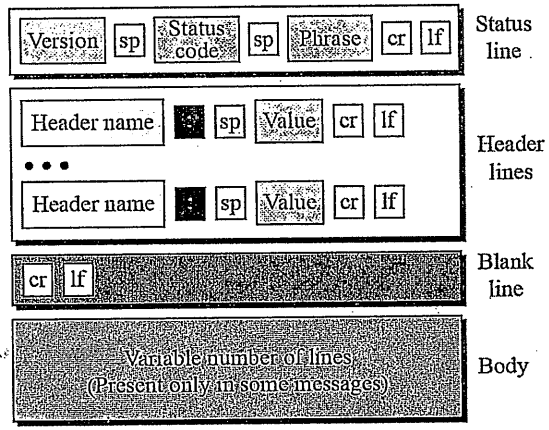
HTTP Response Message

header lines

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
  GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
  1\r\n
\r\n
data data data data data ...
```

data, e.g., requested HTML file

HTTP Response Msg Format



Response message

HTTP Response Codes

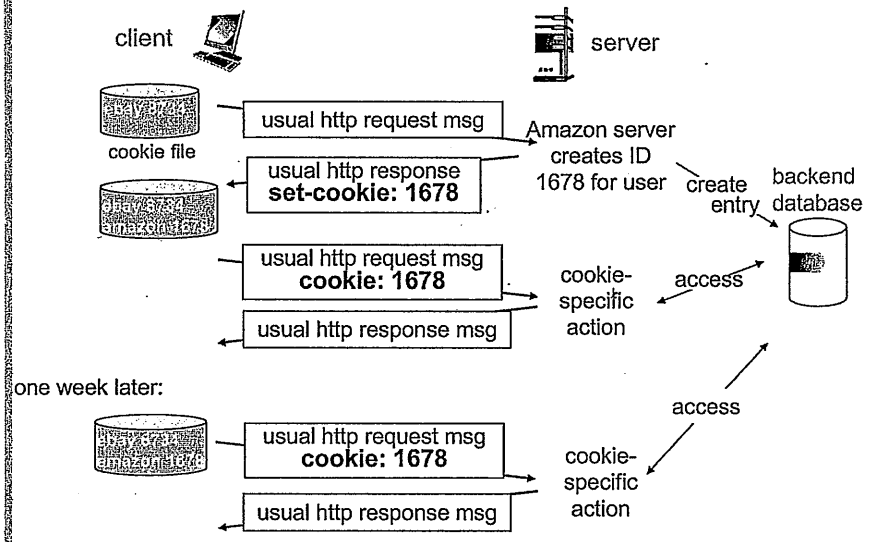
- Status code appears in 1st line in server-to-client response msg
- Some sample codes
  - 200 OK
    - Request succeeded, requested object later in this msg
  - 301 Moved Permanently
    - Requested object moved, new location specified later in this msg (Location)
  - 400 Bad Request
    - Request msg not understood by server
  - 404 Not Found
    - Requested document not found on this server
  - 505 HTTP Version Not Supported

Cookies

- What can cookies be used for?
  - Authorisation
  - Shopping carts
  - Recommendations
  - User session state (Web Mail)
- How to keep "state"
  - HTTP has been designed as a stateless protocol
  - Maintain state at sender/receiver over multiple transactions
    - Cookies: HTTP messages carry state

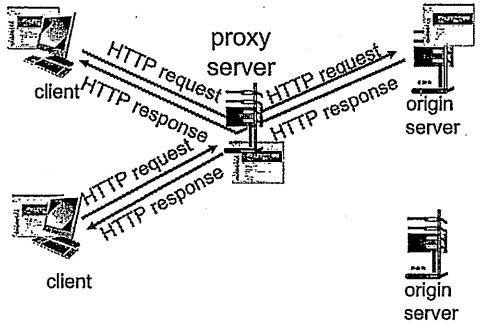
cookies and privacy:
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

Cookies: Keeping State



Web Caches (Proxy Server)

- Goal: satisfy client request without involving origin server
- User sets browser to access the web via proxy
- Browser sends all HTTP requests to proxy server
  - Object in cache
    - Proxy returns object
  - Else proxy requests object from origin server
    - Returns object to client



More About Web Caching

- Cache acts as both client and server
  - Server for original requesting client
  - Client to origin server
- Typically cache is installed by ISP
  - University, company, residential ISP

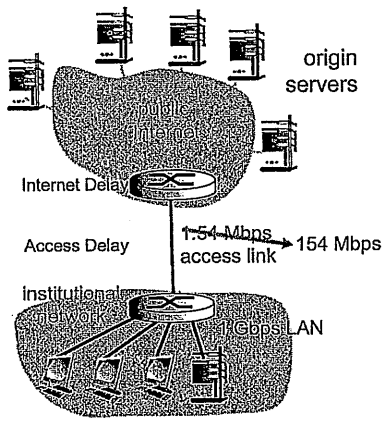
Q: Why Web caching?

- Reduce response time for client request

– Reduce traffic on an institutions access link

Caching Example: Fatter Access Link

- Assumptions
  - Avg object size: 100K bits
  - Avg request rate from browsers to origin servers:15 requests/sec
  - Avg data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec (Internet Delay)
  - Access link rate: ~~1.54 Mbps~~ → 154 Mbps
- Consequences
  - LAN utilization: $15 * 10^5/10^9 = 0.15\%$
  - Access link utilization = ~~97%~~ → 0.97%
    - $15 * 10^5/1.54 * 10^6$
  - Total Delay = Internet Delay + Access Delay + LAN Delay
  - = 2 sec + ~~minutes + usecs~~ msecs

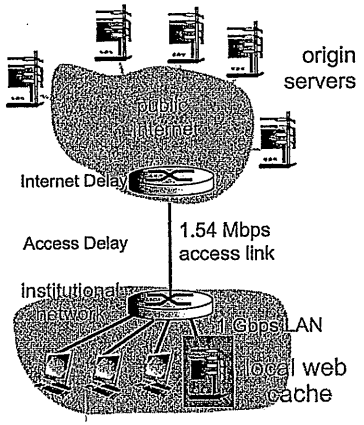


Caching Example: Install Local Cache

- Assumptions
  - Avg object size: 100K bits
  - Avg request rate from browsers to origin servers:15 requests/sec
  - Avg data rate to browsers: 1.50 Mbps
  - RTT from institutional router to any origin server: 2 sec
  - Access link rate: 1.54 Mbps
- Consequences
  - LAN utilization: 0.15%
  - Access link utilization = ?
  - Total delay = ?

How to compute link utilization, delay?

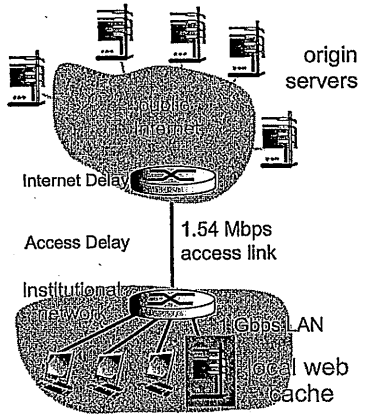
Cost: web cache (cheap!)



Caching Example: Install Local Cache II

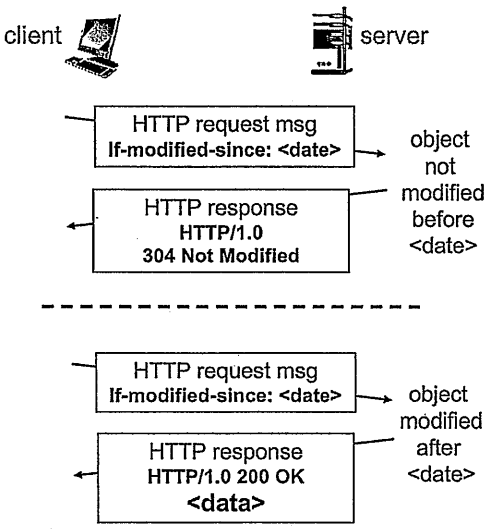
Calculating access link utilization, delay with cache

- Suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
- Access link utilization
  - 60% of requests use access link
- Data rate to browsers over access link = 0.6*1.50 Mbps = 0.9 Mbps
  - Utilization = 0.9/1.54 = 0.58
- Total Delay
  - = 0.6 * (delay from origin servers) + 0.4 * (delay when satisfied at cache)
  - = ~ 1.2 secs
  - Less than with 154 Mbps link (and cheaper too!)



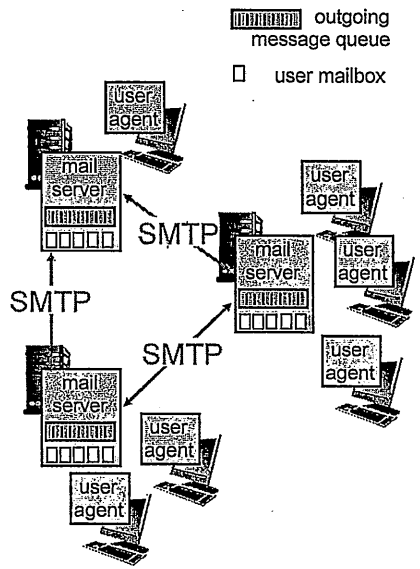
Conditional GET

- Goal: Do not send object if cache has up-to-date cached version
  - No object transmission delay
- Cache: specify date of cached copy in HTTP request
  If-modified-since: <date>
- Server: response contains no object if cached copy is up-to-date:
  HTTP/1.0 304 Not Modified



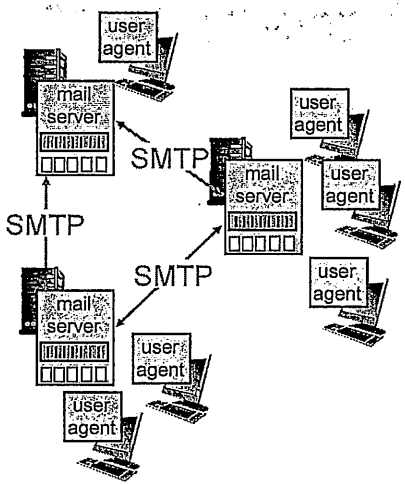
Electronic Mail

- Three major components
  - User Agents
  - Mail Servers
  - Simple Mail Transfer Protocol (SMTP)
- User Agent
  - a.k.a. "mail reader"
  - Composing, editing, reading mail messages
  - e.g., Outlook, Thunderbird, iPhone mail client
  - Outgoing, incoming messages stored on server



Mail Servers

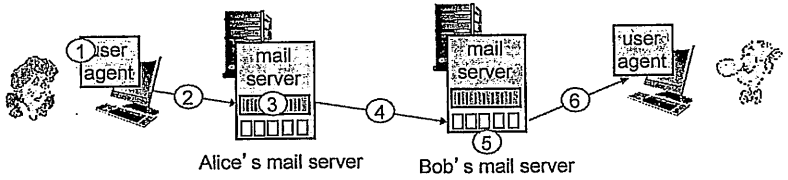
- Mailbox contains incoming messages for user
- Message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
  - Client: sending mail server
  - Server: receiving mail server



SMTP

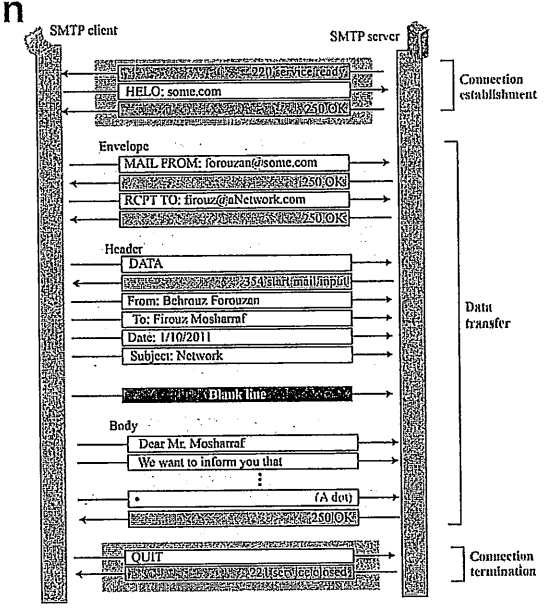
- Uses TCP to reliably transfer email message from client to server on port 25
- Direct transfer: sending server to receiving server
- Three phases of transfer
  - Handshaking (greeting)
  - Transfer of messages
  - Closure
- Command/response interaction (like HTTP, FTP)
  - Commands: ASCII text
  - Response: status code and phrase

Alice Sends Msg to Bob



- 1) Alice uses UA to compose message "to" bob@someschool.edu
- 2) Alice's UA sends message to her mail server
  - Msg placed in queue
- 3) Client side of SMTP opens TCP connection with Bob's mail server
- 4) SMTP client sends Alice's message over the TCP connection
- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his user agent to read message

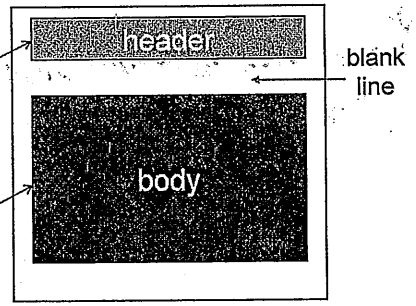
SMTP Interaction



Exercise: Try SMTP interaction for yourself
– telnet *servername* 25

Mail Message Format

- RFC 822: standard for text message format
  - header lines, e.g.,
  - To:
  - From:
  - Subject:
- Body: the "message"
  - ASCII characters only



SMTP Final Words

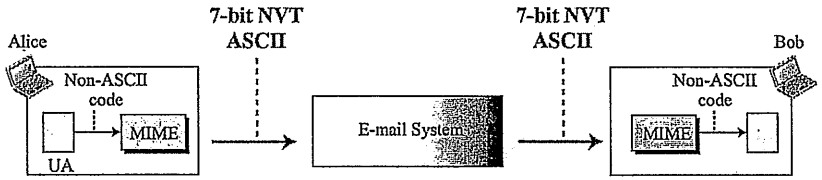
- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF to determine end of message

Comparison with HTTP

- HTTP: pull
- SMTP: push
- Both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg

- SMTP: multiple objects in a single message

Multipurpose Internet Mail Extension (MIME)



- Can only send messages in 7-bit ASCII format
  - Cannot be used for other languages e.g. French, Chinese etc
- MIME is a supplementary protocol that allows non-ASCII data to be sent via SMTP
  - Converts non-ASCII to ASCII and vice versa

MIME Header

MIME headers

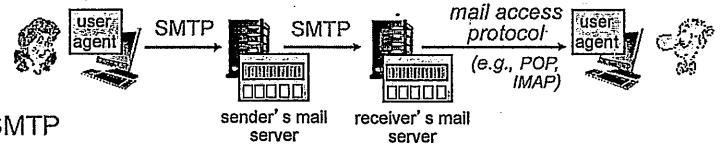
Don't need to understand a huge amount about this

E-mail header
MIME-Version: 1.1
Content-Type: type/subtype
Content-Transfer-Encoding: encoding type
Content-ID: message ID
Content-Description: textual explanation of nontextual contents
E-mail body

Type	*Subtype*	*Description*
Text	Plain	Unformatted
	HTML	HTML format (see Appendix C)
Multipart	Mixed	Body contains ordered parts of different data types
	Parallel	Same as above, but no order
	Digest	Similar to Mixed, but the default is message/RFC822
	Alternative	Parts are different versions of the same message
Message	RFC822	Body is an encapsulated message
	Partial	Body is a fragment of a bigger message
	External-Body	Body is a reference to another message
Image	JPEG	Image is in JPEG format
	GIF	Image is in GIF format
Video	MPEG	Video is in MPEG format
Audio	Basic	Single channel encoding of voice at 8 KHz
Application	PostScript	Adobe PostScript
	Octet-stream	General binary data (eight-bit bytes)

Type	*Description*
7-bit	NVT ASCII characters with each line less than 1000 characters
8-bit	Non-ASCII characters with each line less than 1000 characters
Binary	Non-ASCII characters with unlimited-length lines
Base64	6-bit blocks of data encoded into 8-bit ASCII characters
Quoted-printable	Non-ASCII characters encoded as an equal sign plus an ASCII code

Mail Access Protocols



- SMTP
  - Delivery/storage to receiver's server
- Mail Access Protocol
  - Retrieval from server
- POP: Post Office Protocol [RFC 1939]
  - Authorization, Download
- IMAP: Internet Mail Access Protocol [RFC 1730]
  - More features, including manipulation of stored msgs on server
- HTTP
  - Gmail, Hotmail, Yahoo! mail, etc.

POP3 Protocol

Authorization Phase
- Client commands
  - user: declare username
  - pass: password
- Server responses
  - +OK
  - -ERR

Transaction phase, client:
- list: list message numbers
- retr: retrieve message by number
- dele: delete
- quit

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 and IMAP

POP3

- Previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

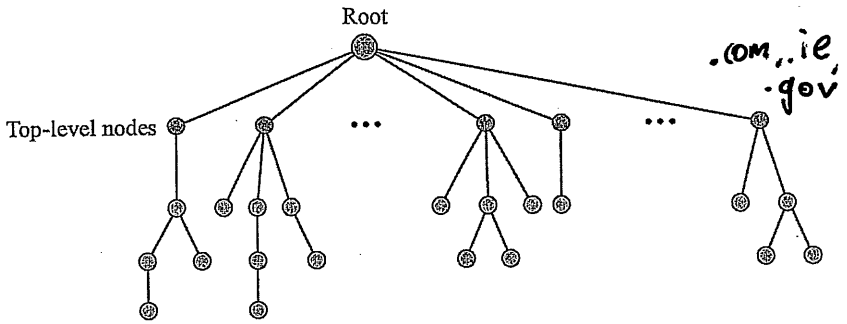
IMAP

- Keeps all messages in one place: at server
- Allows user to organize messages in folders
- Keeps user states across sessions
  - Names of folders and mappings between message IDs and folder name

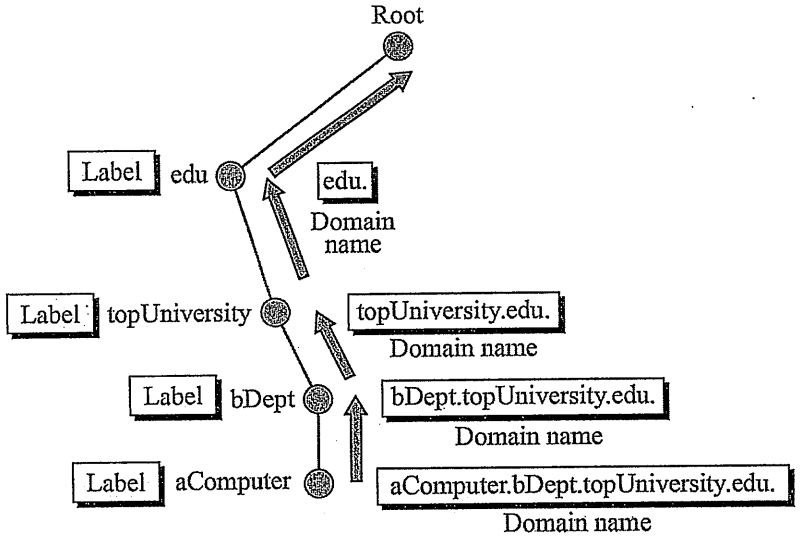
Domain Name System (DNS)

- Distributed database implemented in hierarchy of many name servers
- Application-layer protocol
  - Hosts, name servers communicate to resolve names (address/name translation)
- DNS services
  - Hostname to IP address translation e.g. www.rte.ie or rte.ie
  - Host aliasing
  - Canonical, alias names
  - Mail server aliasing
  - Load distribution
  - Replicated Web servers
- Many IP addresses correspond to one name

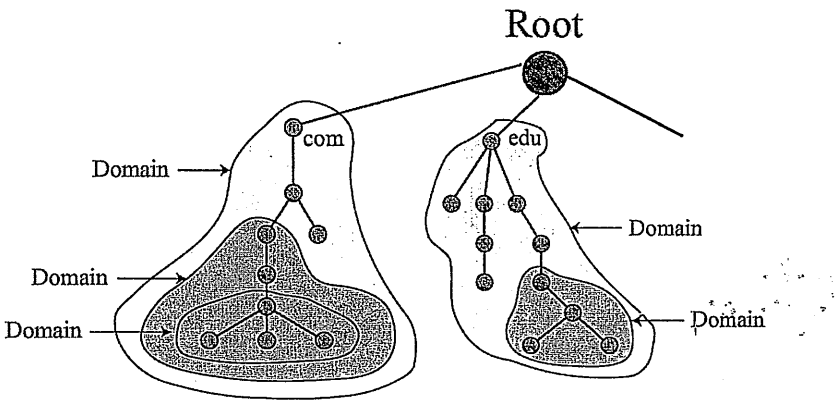
Domain Name Space



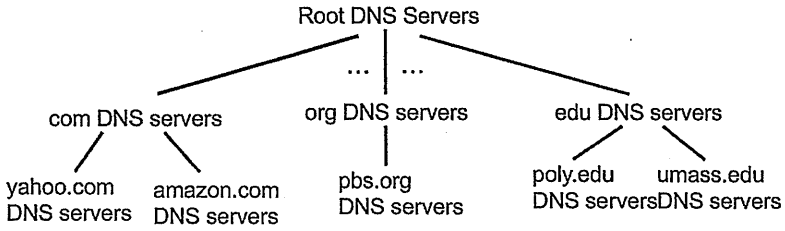
Domain Name and Labels



Domains



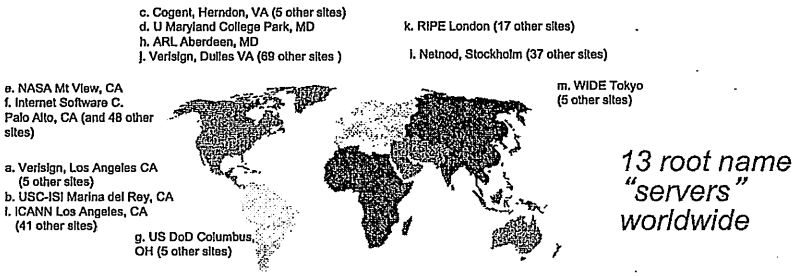
DNS: A Distributed Hierarchical Database



- Client wants IP for www.amazon.com
- Client queries *root* server to find .com DNS server
- Client queries .com DNS server to get amazon.com DNS server
- Client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: Root Name Servers

- Contacted by local name server that can not resolve name
- Root name server
  - Return list of IP addr for responsible TLD servers
  - www.digwebinterface.com



13 root name "servers" worldwide

TLD and Authoritative Servers

- Top-level Domain (TLD) Servers
  - Responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g. uk, fr, ca, jp
    - Network Solutions maintains servers for .com TLD
    - Educause for .edu TLD
- Authoritative DNS Servers
  - Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts

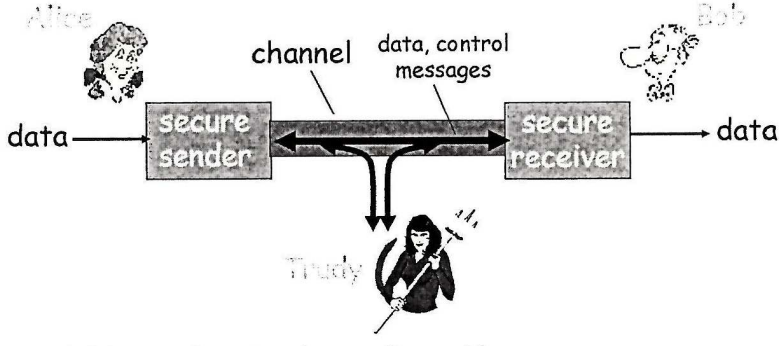
Network Security

- Introduction
- Symmetric-Key Cryptography
- Asymmetric-Key Cryptography
- Digital Signatures, X.509 Certs & PKI
- Authentication Protocols
- HTTPS – SSL/TLS
- IPsec
- DNSSEC

What is Network Security?

- Confidentiality
  - Only sender, intended receiver should "understand" message contents
    - Sender encrypts message
    - Receiver decrypts message
- Authentication
  - sender, receiver want to confirm identity of each other
- Message Integrity
  - Sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

Friends and Enemies



- Well-known in network security world
- Bob, Alice want to communicate "securely"
- Trudy (intruder) may intercept, delete, add messages

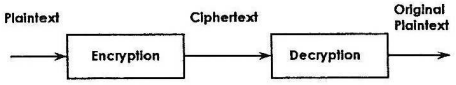
Who might Bob and Alice be?

- Well, real-life Bobs and Alices!
- Web browser/server for electronic transactions
  - e.g. on-line purchases
- On-line banking client/server
- DNS servers
- Routers exchanging routing table updates

What can the bad guys do?

- Passive Attack
  - Eavesdrop or intercept messages
- Active Attack
  - Actively insert messages into connection
- Impersonation
  - Fake (spoof) source address in packet (or any field in packet)
- Hijacking
  - "Take over" ongoing connection by removing sender or receiver, inserting himself in place

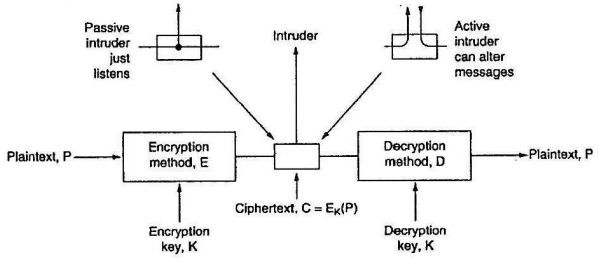
Cryptography



- Original data to be transferred is called Plaintext or Cleartext
  - Encrypted version is called Ciphertext
- Plaintext is denoted *P*, whereas ciphertext is denoted *C*
  - Encryption function *E* operates on *P* to produce *C*
    - $E(P) = C$
- In the reverse process
  - Decryption function *D* operates on *C* to produce *P*
    - $D(C) = P$
- Following identity must also hold true for the cryptosystem to function correctly
  - $D(E(P)) = P$
    - Must be inversable (the two functions).

Cryptographic Keys

- All modern encryption algorithms use a key denoted by *K*
- The key can take on many possible values
  - Range of possible values is called the key space



- The encryption and decryption functions now become
  - $E_K(P) = C$ and $D_K(C) = P$

Substitution Ciphers

- Each letter or a group of letters is replaced by another letter or group of letters to disguise it
- Caesar Cipher - Mono-alphabetical Substitution
  - In this system the alphabet is written out twice

a b c d e f g h i j k l m n o p q r s t u v w x y z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

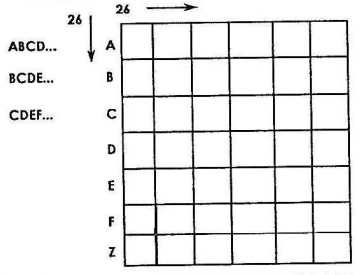
- Q: What is the key?
  - 3 (shift by 3)
- To send a secret message
  - Letters of the message are taken one by one and the letters appearing below are written instead
- The message "send spears" would be enciphered as
  - "VHQG VSHDUV"

Substitution Ciphers II

- Attacks
  - Identify commonly occurring characters
    e, t, o, a, n, i
  - Commonly occurring bigrams/digrams
    th, he, in, er
  - Domain specific buzz words
    System, login, Password, money etc.
- Substitution ciphers preserve the order of the text symbols but disguise them

The Vigenère Cipher

- Some protection from the above can be gained by using a poly-alphabetical cipher
  - create a matrix of 26 different alphabets



- Now pick a key e.g. AFGHANISTANBANANISTAN
  - Use row A to encrypt first letter of plaintext, row F the second time around etc...

Exercise: How can we break this cipher?

Transposition Ciphers

- Transposition ciphers reorder the symbols
  - Do not disguise them

M	E	G	A	B	U	C	K
7	4	5	1	2	8	3	6
p	l	e	a	s	e	t	r
a	n	s	f	e	r	o	n
e	m	i	l	l	i	o	n
d	o	l	l	a	r	s	t
o	m	y	s	w	i	s	s
b	a	n	k	a	c	c	o
u	n	t	s	i	x	t	w
o	t	w	o	a	b	c	d

Plaintext

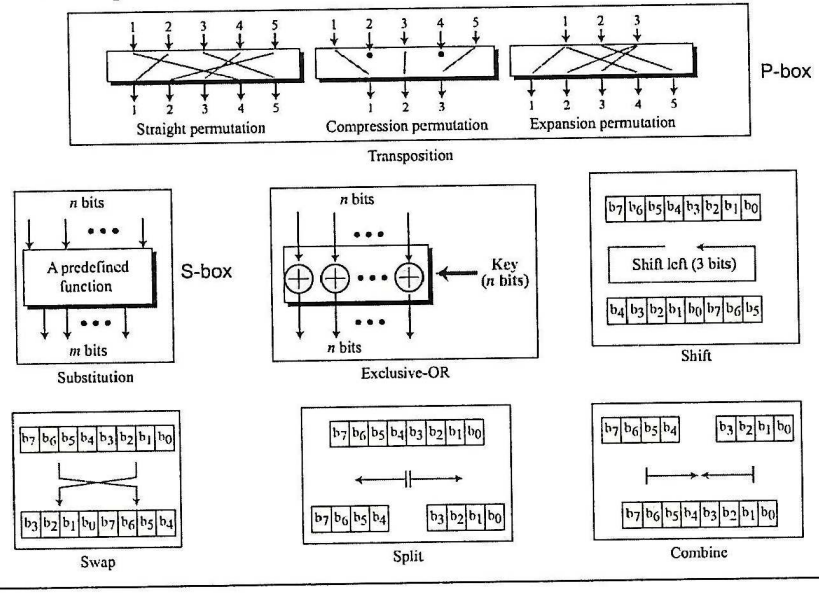
pleasetransferonemilliondollarsto
myswissbankaccountsixtwotwo

Ciphertext

AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUOERIRICXB

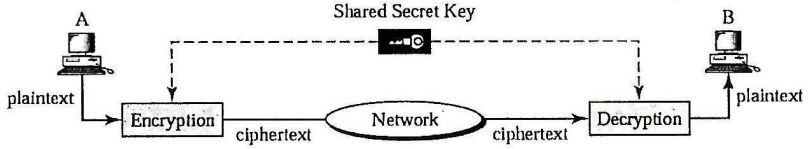
- The plaintext is written horizontally in rows
- Ciphertext is read out in columns
  - Starting with the column whose key is the lowest

Components of a Modern Block Cipher



Symmetric-Key Encryption

- Based on the sender and the receiver of a message knowing and using
  - The same (Secret) key
  - Cryptosystem



- Sender uses the secret key to encrypt the message
  - Receiver uses the same secret key to decrypt the message

Q. How do Bob and Alice agree on a key value?

Key Management

- Main problem is getting the sender and receiver to agree on a secret key without anyone else finding out
  - Especially if there is no cryptosystem in place to begin with
- Key management is one of the fundamental issues that has to be addressed in symmetric key cryptosystems
- Examples of symmetric key algorithms are
  - DES, Triple DES, IDEA, AES

DES can be brute forced
AES used today

Data Encryption Standard (DES)

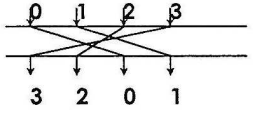
- In January 1977 a standard encryption method was adopted by the U.S. gov
  - Origins lie in an internal IBM project codenamed Lucifer
- Though the algorithm used is complex
  - It is easily implemented in hardware
  - Software implementations are also widely available
- DES is a Block Cipher
  - Operates on a single chunk of data at a time
    - 64 bits (8 bytes)

- Produces a 64 bit output

- The key length is 56 bits
  - Often expressed as a 8 character string with the extra bits used as a parity check

DES

- Algorithm has 19 distinct stages
- First stage re-orders the bits of the 64-bit input block by applying a fixed permutation (P-box)

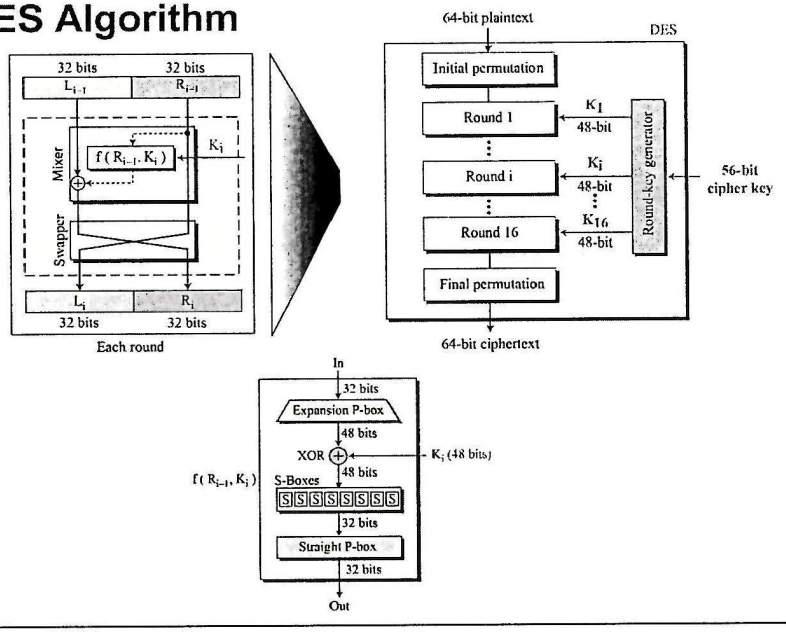


Transposition of 4 bits (example only)

- Last stage is the exact inverse of this permutation
- Stage penultimate to the last one
  - Exchanges the leftmost 32 bits with the rightmost 32 bits
- Remaining 16 stages are called Rounds
  - Functionally identical but take as an input a quantity computed from the key and the round number

- Round key is 48 bits

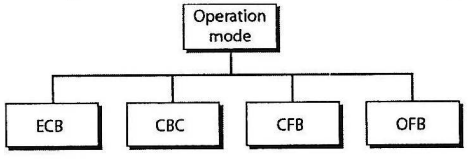
DES Algorithm



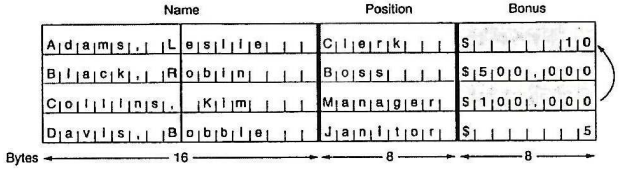
Cracking DES

- 56 bits is a short key
- Brute force attack (try every key)
  - 2^{56} encryption to try all keys
  - Special chips can check 4 million keys/sec
  - $1 million DES cracking machine could break it in a few hours
- Jan'99 Challenge III won in 22 hrs and 15 mins using a supercomputer and 100,000 Internet nodes
  - Tested 245 billion keys per second
- Improvement - Triple DES or 3DES
  - Makes use of two or three keys (112 or 168 bits).
  - Uses EDE or DED mode

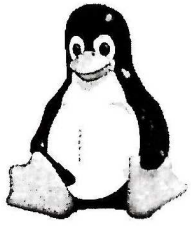
Modes of Operation for Block Ciphers



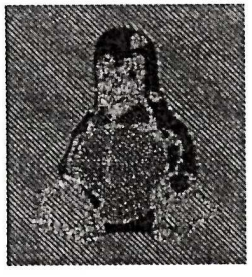
- In the previous discussion of DES known as Electronic Code Book (ECB) mode
- Still allows for an passive intruder to replicate the information
  - e.g. Repeat request for withdrawal of $1bn



Electronic Code Book Mode



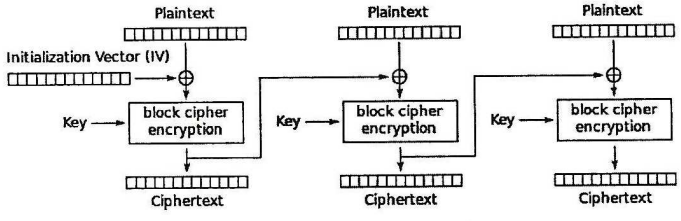
Original Image



Encrypted with ECB Mode

Cipher Block Chaining Mode (CBC)

- In CBC mode each block of plaintext is XORed with previous ciphertext block before being encrypted
- Each ciphertext block depends on all plaintext blocks processed up to that point



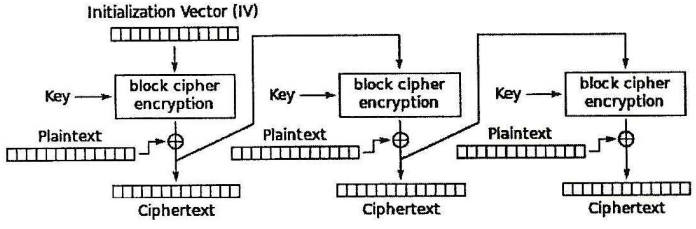
Cipher Block Chaining (CBC) mode encryption

• To make each message unique

encrypt a character at the time

Cipher Feedback Mode (CFB)

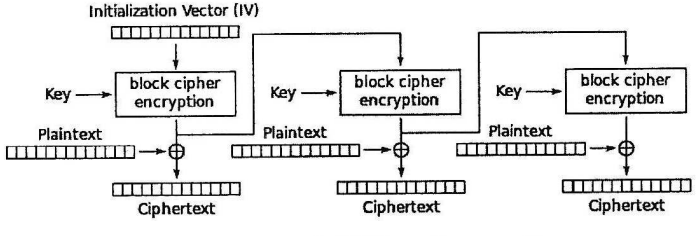
- The CFB mode is a close relative of CBC
  - Makes a block cipher into a self-synchronizing "Stream Cipher"



Cipher Feedback (CFB) mode encryption

Output Feedback Mode (OFB)

- OFB mode generates a keystream of blocks, which are then XORed with the plaintext blocks to get the ciphertext

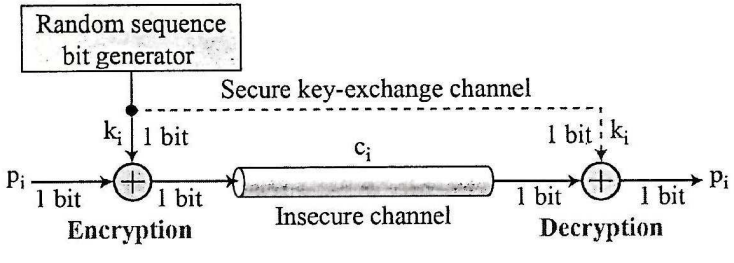


Output Feedback (OFB) mode encryption

in theory most secure cipher : don't use it in practise

The Vernam Cipher

- Simplest and most secure stream cipher is called the One-time Pad
- Chooses a key stream (k) that is randomly chosen for each encipherment -- makes use of XOR operator

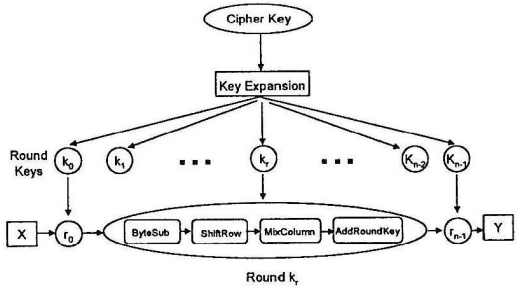


K>=P : can't recycle the key

Advanced Encryption Standard (AES)

- In 1997 NIST announced a call for proposals to develop a new Advanced Encryption Standard
  - After a long vetting process five algorithms were short listed
    - Rijndael (Rhine-doll) was eventually chosen as the new AES
- Symmetric cipher with variable key and block sizes of 128, 192 and 256 bits
  - Most common mode is 128 bit key and block size
  - Support for fast encryption and decryption in s/w - 700 Mbps
    - Can be implemented efficiently in smartcards
- A device that could check a 10^{18} AES keys/s would in theory require about 3×10^{51} years to exhaust the 256-bit key space
- Brute force decryption taking 1 sec on DES, takes 149 trillion years for AES

AES



- The cipher consists of between 10 or 14 rounds (N_r)
  - Depending on the key length (N_k) and the block length (N_b)
- A plaintext block X undergoes n rounds of operations to produce an output block Y
  - Each operation is based on the value of the n^{th} round key
- Round keys are derived from the Cipher key by first expanding the key
  - Then selecting parts of the expanded key for each round

Asymmetric-Key Cryptosystems

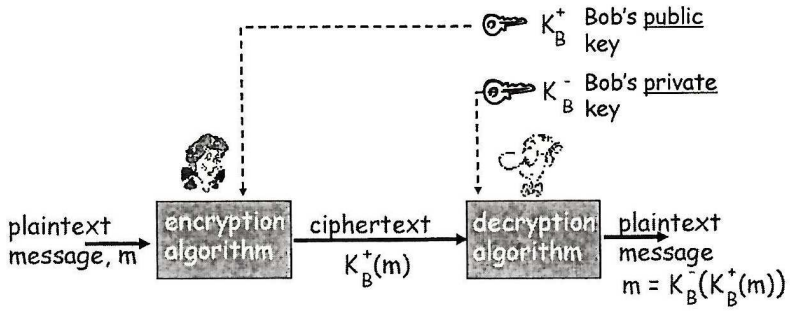
- Public key cryptography was invented by Diffie and Hellman in 1976
  - Solves the key management problem associated with symmetric key cryptosystems
- In public key cryptography each person generates a pair of keys
  - The Public key and the private key
- Public key is published and widely distributed
  - While the private key is kept secret
- Examples of public-key cryptographic algorithms are
  - RSA, Diffie-Hellman, ElGamal, ECC

Properties of Asymmetric-Key Cryptosystems

- Must be computationally easy to encipher or decipher a message given the appropriate key
- Must be computationally infeasible to derive the private key from the public key
- Need for exchanging secret keys is eliminated
  - All secure communications now only involves Public keys

Public-Key Cryptography

- Each user in a public-key system selects his own private key (K^-) and his own public key (K^+)



- When user Alice wants to send an encrypted message to Bob
  - She looks up his Public key (K_B^+) in a Public directory
  - Or obtains it by some other means

RSA

- De-facto standard algorithm for implementing asymmetric-key cryptography
  - Named after Rivest, Shamir and Adleman who developed it in 1978 at MIT
- It's security is based on the difficulty of factoring very large numbers
  - One-way "Trapdoor Function"
- Example: Prime Factoring
  - Easy to calculate product of 2 large prime numbers
  - Difficult to calculate the prime factors from product

Modular Arithmetic I

- Most number sets we are used to are infinite e.g. set of real numbers
  - However most cryptographic algorithm are based on arithmetic with a finite set of numbers
- Consider the hours of a clock
  - 1*h*, 2*h*, 3*h*, ..., 11*h*, 12*h*, 1*h*, 2*h*, 3*h*, ..., 11*h*, 12*h*, 1*h*, 2*h*, 3*h*, ...

Modulo Operation

- Let *a*, *r*, *m* ∈ ℤ (where ℤ is the set of all integers) and *m* > 0
  - $a \equiv r \bmod m$
- *a* is said to be congruent to *r* mod m if *m* divides *a* - *r*
  - *m* is called the modulus and *r* is called the remainder

Modular Arithmetic II

- It is always possible to write $a \in \mathbb{Z}$ such that
  - $a = q * m + r \quad 0 \le r < m$
- Since $a - r = q * m$ (*m* divides $a - r$) we can now write
  - $a \equiv r \bmod m$
- Example : Let $a = 42$, $m = 9$
  - $42 = 4 * 9 + 6$
    - $42 \equiv 6 \bmod 9$
- Q: $-11 \equiv x \pmod 7$
  - $-11 \equiv 3 \bmod 7$
  - $-11 - 3 = -2 * 7$

Multiplicative Inverse

- The integers modulo n, denoted $\mathbb{Z}_n$ is the set of integers {0, 1, 2, … n-1}
  - Addition, subtraction and multiplication are performed modulo n
    - $\mathbb{Z}_n$ is referred to as an *Integer Ring*
- $\mathbb{Z}_{25}$ = {0, 1, 2, …, 24}
  - 13 + 16 = 4
    - $29 \equiv 4 \pmod{25}$
- The multiplicative inverse of a modulo n is an integer $x \in \mathbb{Z}_n$ such that
  - $ax \equiv 1 \pmod{n}$
- The multiplicative inverse only exits for an element $a \in \mathbb{Z}_n$ iff
  - gcd(a, n) = 1

Q: Does the multiplicative inverse of 15 exist in $\mathbb{Z}_{26}$?

Extended Euclidean Algorithm (EEA)

- Division of a by b modulo n is a product of a and b^{-1} modulo n
  - $b$la is equivalent to $a * b^{-1} \pmod{n}$
- Q: What is 4^{-1} modulo 11?
  - $x \equiv 4^{-1} \pmod{11}$
    - $4 * x \equiv 1 \pmod{11}$
- The modular multiplicative inverse of a modulo m can be found with the Extended Euclidean Algorithm
  - $s * r_0 + t * r_1 = \gcd(r_0, r_1)$
  - $s * r_0 + t * r_1 = 1$
  - $t = r_1^{-1} \bmod r_0$
- gcd (11, 4) = 1
  - 11 = 2 * 4 + 3
  - 4 = 1 * 3 + 1
  - 3 = 3 * 1 + 0
- Back substitution
  - 1 = 4 – 1 * 3
  - = 4 – 1 * (11 – 2 * 4)
  - = 3 * 4 – 1 * 11
  - = (-1) * 11 + (3) * 4
  - $t = 3$
    - $4 * 3 \equiv 1 \pmod{11}$

$s * r_0 + t * r_1 = 1$

$s * r_0 + t * r_1 \equiv 1 \pmod{r_0}$

$t = r_1^{-1} \pmod{r_0}$

Exercise: compute $15^{-1} \bmod 26$

Fermat's Little Theorem

- Let a be an integer and p be a prime, then
  - $a^p \equiv a \pmod{p}$
  - $a^{p-1} \equiv 1 \pmod{p}$
  - $a * a^{p-2} \equiv 1 \pmod{p}$
    - $a^{-1} \equiv a^{p-2} \pmod{p}$
- Thus we have a way for inverting an integer a modulo a prime
- Example a = 4 and p = 11
  - $a = 4$ and $p = 11$
    - $4^9 = 262144 \equiv 3 \bmod 11$
- Can also be used for basic primality testing!!
  - Exercise: Is 221 a prime number?

Euler's Totient function $\phi(n)$

- Number of positive integers less than n and relatively prime to n
- Example: $\phi(10) = 4$
  - 1, 3, 7, 9 are relatively prime to 10
    - Relatively prime means gcd(a, b) = 1
- Example: $\phi(21) = 12$
  - 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20 are relatively prime to 21
- Q: What is $\phi(7)$ and $\phi(11)$?
  - $\phi(m) = m - 1$ when m is prime!

RSA Algorithm

- Choose two large distinct primes p and q
- Compute the product (modulus)
  - $n = p * q$
- Randomly choose an encryption key e, less than n that has no common factors with $\phi(n)$
  - e and $\phi(n)$ are relatively prime [GCD of these two is 1, therefore we have an inverse of E (here)]
  - e is invertible iff $gcd(e, \phi(n)) = 1$
- Finally compute the decryption key, d such that
  - $e * d \equiv 1 \pmod{\phi(n)}$
  - $d \equiv e^{-1} \bmod \phi(n)$
  - $d \equiv e^{-1} \bmod (p - 1)(q - 1)$

∴ we could break the private key if we could factorize $f(n)$ i.e. get p and q.

RSA Usage

- The numbers e and n are the public key
  - The number d is the private key
- Break the plaintext message into a number of blocks
  - Represent each block as an integer
- Encryption
  - CiphertextBlock = $(\text{PlaintextBlock})^e \bmod n$
- Decryption
  - PlaintextBlock = $(\text{CiphertextBlock})^d \bmod n$
- Key Sizes
  - 1024, 2048, 3072, 7680 bits

• Recommended key size from 2015 onward is 3072 bits

RSA with Workable Numbers

- Let $p = 3$ and $q = 11$
- Using $n = p * q = 33$
  - $\phi(n) = (p - 1) * (q - 1) = 20$
- Choose $e = 3$ (e and $\phi(n)$ have no common factors)
- Solving $e * d \equiv 1 \bmod 20$ and $d < 20$
  - $d \equiv e^{-1} \bmod 20$

- $d = 7$ (How to get this?)

RSA Example

Plaintext (P)			Ciphertext (C)		After decryption	
Symbolic	Numeric	P^3	$P^3 \pmod{33}$	C^7	$C^7 \pmod{33}$	Symbolic
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	05	E
		Sender's computation		Receiver's computation		

- Since the primes chosen in this example are small, P must be less than 33
  - Each block can only contain a single character
- If p and $q \approx 2^{512}$, we would have $n \approx 2^{1024}$

An Important Property of RSA

The following property will be very useful later

$$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$$

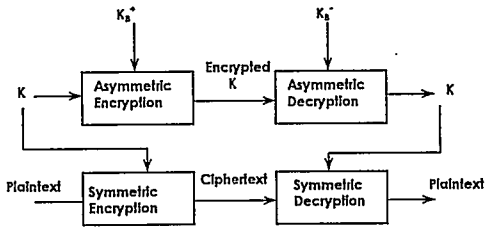
use public key first, followed by private key

use private key first, followed by public key

result is the same!

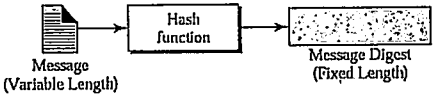
Hybrid Schemes

- Asymmetric-key algorithms are not a replacement for symmetric-key algorithms such as DES or AES
  - Rather they supplement DES or any other fast bulk encryption cipher



- Above example shows
  - How we can use a public key algorithm to securely transfer a session key (*K*), and

Msg Auth & Integrity

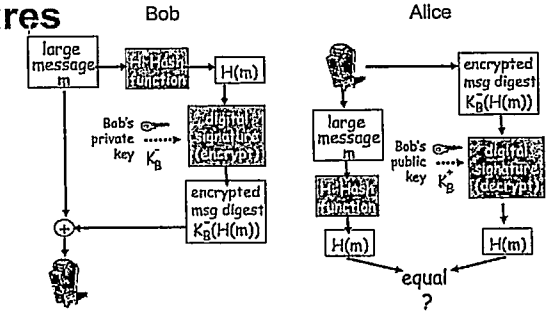


- Bob receives a msg from Alice, wants to ensure
  - Msg originally came from Alice - *Authentication*
  - Msg not changed since sent by Alice – *Integrity*

Message Digest/Cryptographic Hash

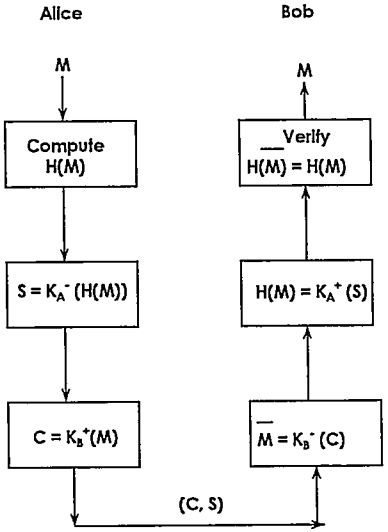
- A message digest is a strong digital fingerprint of a message
- Takes input *m*, produces fixed length value, *H(m)*
  - 128/160/256 bits
- Computationally infeasible to find two different messages, *x* and *y* such that
- Examples of message digest algorithms are
  - MD2, MD4, MD5, SHA-1 and SHA-2

Digital Signatures



- Cryptographic technique analogous to hand-written signatures
- Sender (Bob) digitally signs the document establishing he is the document owner/creator
- Recipient (Alice) can prove to someone that Bob, and no one else (including Alice) must have signed document

Enveloped and Signed Data



Key Management

- When Alice obtains Bob's public-key (from a website, e-mail etc.), how does she know it is Bob's public key, not Trudy's?
- Public key cryptography is based on the idea that
  - An individual will generate a key pair
  - Keep one component secret and publish the other component (public key)
- Other users on the network
  - Must be able to retrieve this public key and associate the user's identity with it
- One way to form this association is to

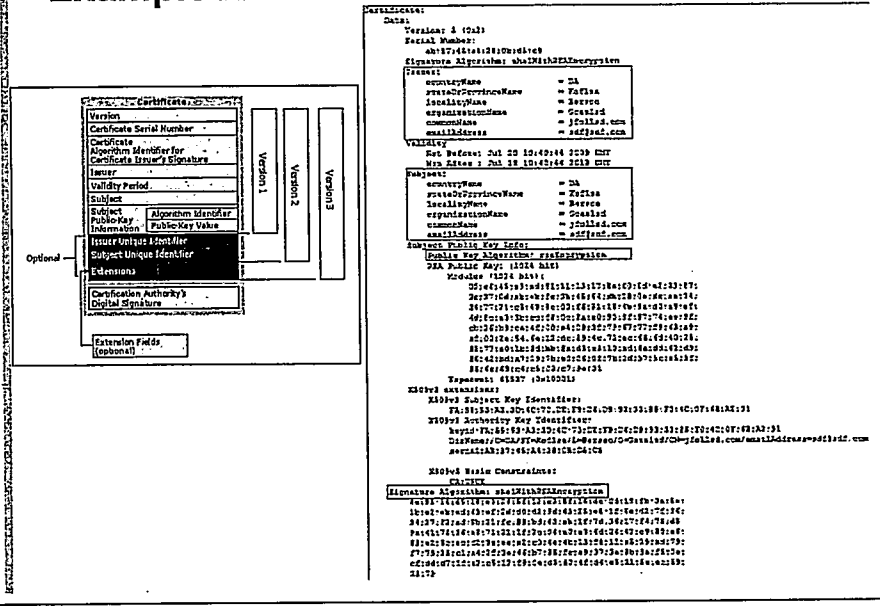
X.509 Certificates

- The TTP will construct a message referred to as a Certificate

Subject (Identity of User)	Public Key	Validity Period	Issuer (Identity of TTP)	Other fields	Signature of TTP

- The cert contains a number of fields
  - Identity of the user
  - Public key of the user
  - .....
- Assumes that every user in the system is equipped with the public-key of the TTP
  - Allows one to verify the digital signature on the certificate
    - Guaranteeing that the public key is associated with the named user

Example X.509 Certificate



1

Electronic Payment Systems

- Overview
- eCash
- Bitcoin
- Micropayments

2

Development of Money

Definition: *"something generally accepted as a medium of exchange, a measure of value, or a means of payment."*

Monetary History

ABSTRACTION

- Barter (direct exchange of goods)
- Medium of exchange (arrowheads, salt)
- Coins (gold, silver)
- Tokens (paper)
- Notational money (bank accounts)
- Dematerialized schemes (pure information)

} NEED BANKS

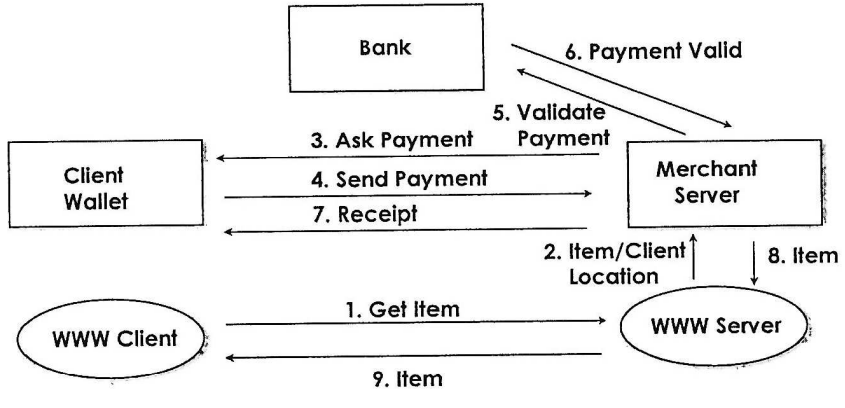
3

Types of Money

	TOKEN	NOTATIONAL	HYBRID
FIDUCIARY	• CASH • GOVERNMENT BEARER BOND	• ACCOUNT WITH CENTRAL BANK	• GOVERNMENT CHECK
SCRIPTURAL	• CERTIFIED CHECK • TRAVELER'S CHECK	• BANK ACCOUNT • FREQUENT FLYER MILES	• PERSONAL CHECK • GIFT CERTIFICATE

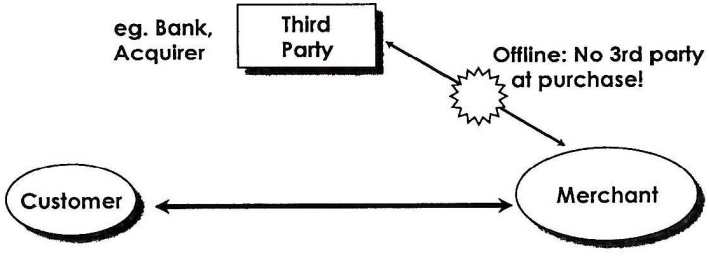
4

Generic Web Payment Example



On-line/Off-line

- In an on-line payment a third party is involved at the time of purchase
- Transaction takes longer
- Third party can be bottleneck



Anonymity vs Audit Trail

- Audit trail provides detailed log of all payments
  - Helps prevent fraud
  - Banks like it!
- Anonymity protects identity of buyer
  - Full anonymity
  - Limited anonymity
    - Collaboration could yield identity
  - Anonymous to merchant
  - Privacy
    - Payment details hidden from outsiders

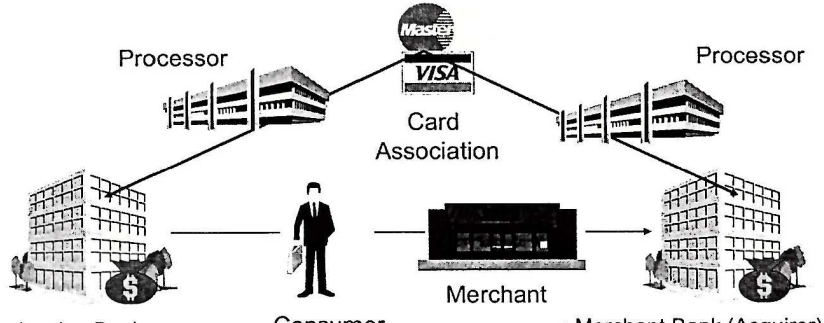
Payment Methods

Macropayments
- > $1
- Strong Crypto

Micropayments
- < $1
- Lightweight Crypto

- **Credit/Debit Cards**
  - International acceptability
  - No fee for buyer
- **Cash**
  - Small amounts
  - Person-to-person
  - No (low) transaction fee
  - No need for bank account
- **Cheque/EFT**
  - Potentially large amounts
  - Person-to-person
  - Vital for B2B transactions
- **Micro**
  - Very small payments
  - Possibly 10c or less
  - Information goods
    - » e.g. Payment for stock quote $0.001

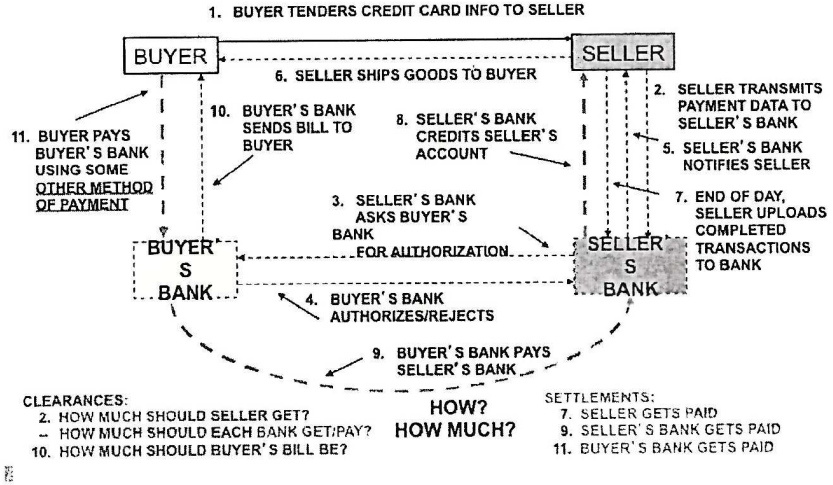
Card Payment - Participants



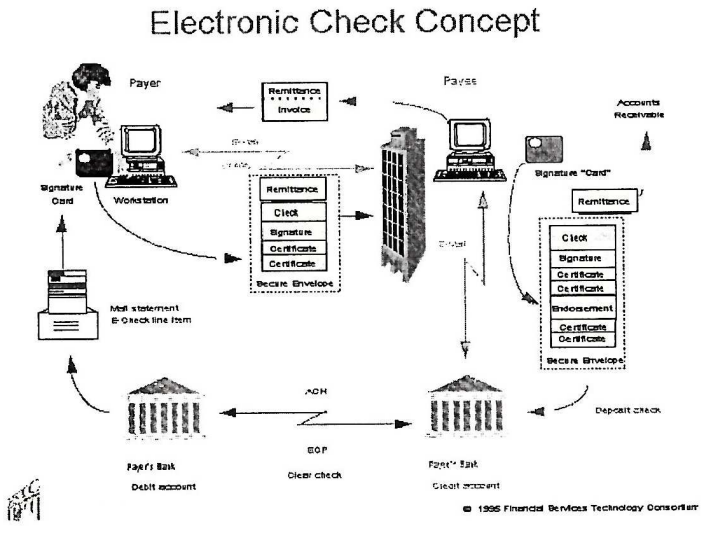
- Issuing Bank
- Issues card
- Extends credit
- Assumes risk of card
- Cardholder reporting

- Merchant Bank (Acquirer)
- Sets up merchant
- Extends credit
- Assumes risk of merchant
- Funds merchant

Card Payment - Transaction



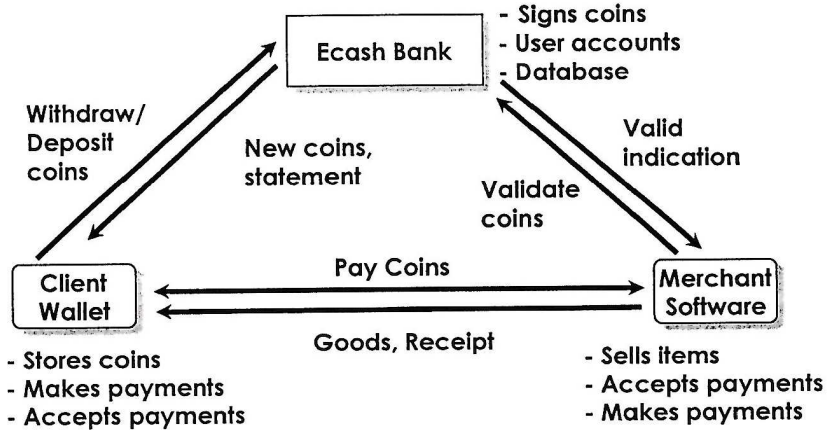
Electronic Cheques



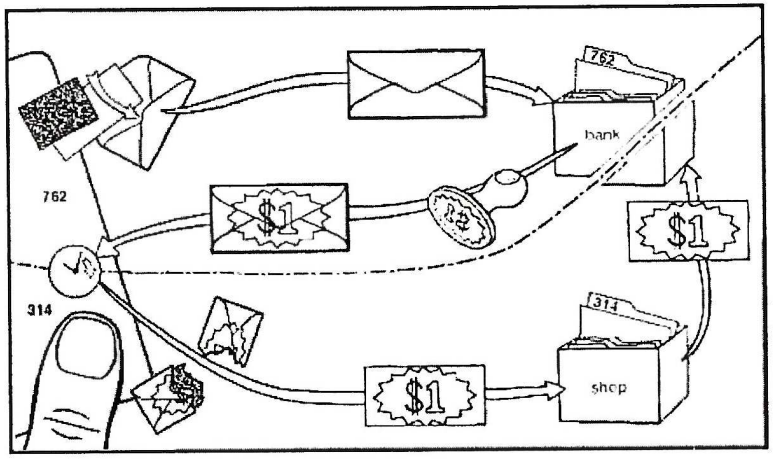
eCash

- Fully anonymous digital cash
  - Pieces of data representing real monetary value
  - Digitally signed by bank
  - Problems
- DigiCash - 1990
  - David Chaum, "the father of digital cash"
- Information, hard goods etc.
- Strong security, good privacy

eCash Payment Model



Anonymous Digital Cash



Blind Signature Protocol

- Let m be the coin's serial number, r the blinding factor, e and n the bank's public key exponents
- Sender raises r to the banks public key exponent e, and computes the product of serial number and the blinding factor
  - $m' \equiv m \cdot r^e \pmod{n}$
- Bank signs the blinded serial number with its private key
  - $s' \equiv (m')^d \bmod n$
- Returns the coin to the user who removes blinding factor
  - $s \equiv s' \cdot r^{-1} \pmod{n}$
- User now has a coin signed with the banks private key
  - $s \equiv (m')^d \cdot r^{-1} \pmod{n}$
  - $= (m \cdot r^e)^d \cdot r^{-1} \pmod{n}$
  - $= m^d \cdot r^{ed} \cdot r^{-1} \pmod{n}$
  - $= m^d \cdot r \cdot r^{-1} \pmod{n}$
  - $= m^d \pmod{n}$

Bitcoin

- Decentralized, Peer-to-Peer (P2P) Electronic Cash System
  - Invented in 2008 by "Satoshi Nakamoto"
- Bitcoin makes everyone collectively the bank!!
  - No longer any single organization in charge of the currency
- Think about the enormous control a central bank has over the money supply
  - Bitcoin introduces a pretty huge change to this business model
- Makes use of the "proof-of-work" concept to prevent double spending in the Bitcoin network
  - Bitcoin miners are rewarded for solving the proof-of-work problem with newly minted bitcoins or transaction fees

Bitcoin – Version 1

- Suppose Alice wants to give Bob a Bitcoin
  - Alice writes down the message "I, Alice, am giving Bob one Bitcoin"
  - Digitally signs the message using her private key
    - Announces the signed string of bits to the entire world

Q: What is the problem with this version of the protocol?
Alice could keep sending Bob the same signed msg over and over!

- We need a way of making Bitcoins unique
  - Need a label or serial number
  - Alice would sign the message "I, Alice, am giving Bob one Bitcoin, with serial number 8740348"

Bitcoin – Version 2

- Make *everyone* collectively the bank
  - Everyone keeps a complete record of which Bitcoins belong to which person
    - i.e. a shared public ledger showing all Bitcoin transactions
- Suppose Alice wants to transfer a Bitcoin to Bob
  - Signs the message "I, Alice, am giving Bob one Bitcoin, with serial number 1234567"
  - Bob uses his copy of the Blockchain to check that the Bitcoin is Alice's to give
    - Broadcasts both Alice's msg and his acceptance of the transaction to the entire network
      - Everyone updates their copy of the Blockchain

Q: What is the problem with this version of the protocol?

Bitcoin – Version 3

- When Alice sends Bob a Bitcoin
  - Bob should not try to verify the transaction alone
- Broadcast the transaction to the entire network of Bitcoin users
  - Ask them to help determine whether the transaction is legitimate
- Q: Can Alice double spend in this version of network-based protocol?

Proof-of-Work (PoW)

- Involves a combination of two ideas
  - Make it computationally costly for network users to validate transactions
- As people on the network hear a message
  - Each adds it to a queue of pending transactions that they have been told about, but which have not yet been approved
  - A network user named David might have the following queue of pending transactions
    - I, Tom, am giving Sue one Bitcoin, with serial number 1201174
    - I, Alice, am giving Bob one Bitcoin, with serial number 1234567
    - .....

Hash Collisions

- David checks his copy of the Blockchain, and can see that each transaction is valid
  - Would like to help out by broadcasting news of that validity to the entire network
- As part of the validation protocol David is required to solve a hard computational puzzle – the "Proof-of-Work"
- David has to find a nonce x such that when we append x to the list of transactions l and hash the combination, the output hash begins with a long run of 0s
- The puzzle can be made more or less difficult by varying the number of zeroes
  - A simple puzzle might require four 0s at the start of the hash
    - A more difficult puzzle might require 15 consecutive zeros

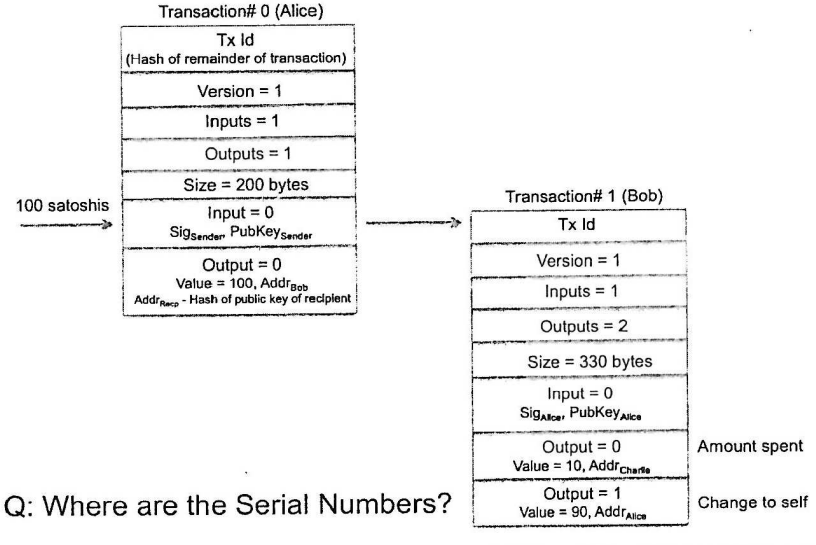
PoW Example

- For example, if we use *l* = "Hello, world!" and the nonce $x = 0$
  - h("Hello, world!0") = 1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64
    - $x = 0$, is a failure, since the output does not begin with any 0s
- We can keep trying different values for the nonce, $x = 1, 2, 3, \ldots$ Finally, at $x = 4250$ we obtain
  - h("Hello, world!4250") = 0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9
- If we want the output hash value to begin with 10 zeroes
  - Then on average, we need to try $16^{10} \approx 10^{12}$ different values for x before we find a suitable nonce

Bitcoin Miners

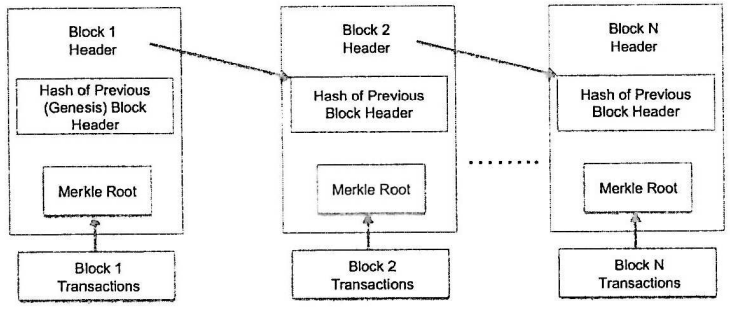
- Suppose David is lucky and finds a suitable nonce x
- Broadcasts the block of transactions he is approving to the network, together with the value for x
  - Other participants in the network can verify that x is a valid solution to the proof-of-work puzzle
    - Update their Blockchain to include the new block of transactions
- This validation process is called *mining*
  - For each block of transactions validated, the successful miner receives a bitcoin reward

Bitcoin Transactions

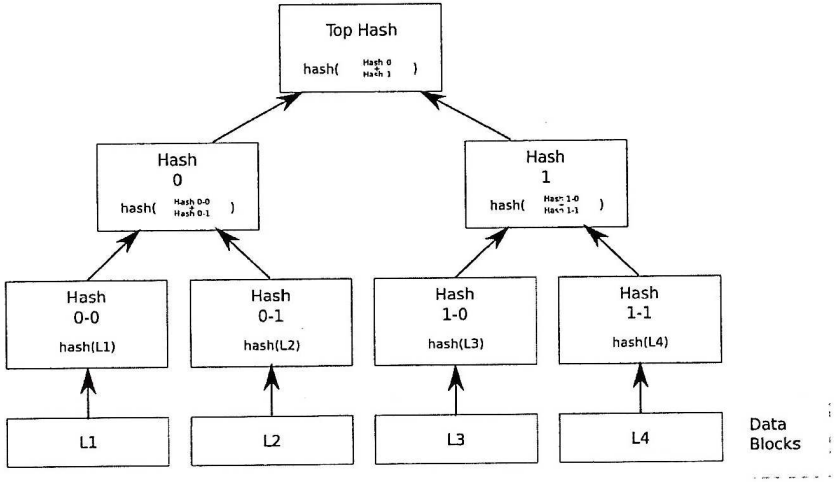


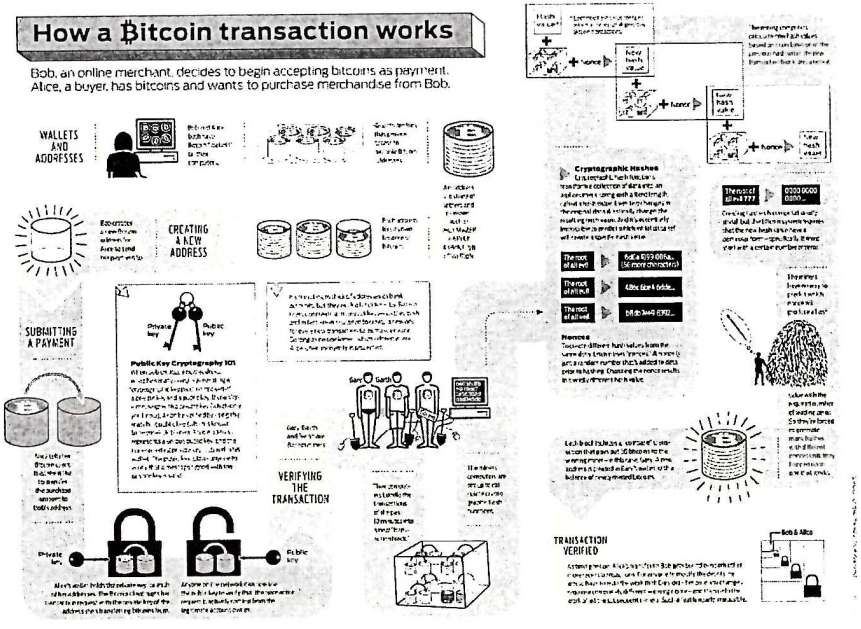
Q: Where are the Serial Numbers?

Block Chain



Merkle Tree





Micropayments

- Repeated small payments for low value information
- Macropayment Problems
  - Minimum price set by transaction processing costs
  - Maximum number of transactions/second
    - Efficiency limits of strong cryptographic protocols
- Micropayments Solution
  - Very small per-transaction cost (sub-cent)
  - Efficiency by slightly relaxing security
  - Some fraud (few cents) is OK
- Systems
  - Millicent, PayWord, MicroMint, Subscrip

Micropayments Enable

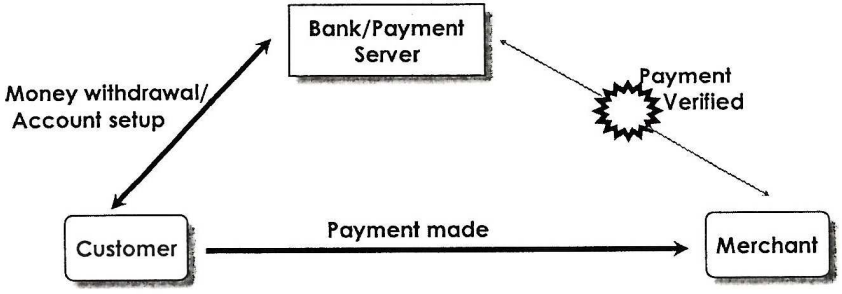
- No minimum price for information and services
  - New Internet opportunities
- Quality information due to financial reward

To buy information
- Articles and Web pages
- Stock quotes and DB queries
- Cartoons and clip art
- Music and videos

To buy software
- Java applets
- Apps
- Software add-ons
- Games

To bill access
- To applications
- For services
- Education
- To shared resources

Micropayment Purchase



On-line verification with 3rd party removed