

# Contents

<b>1</b>	<b>Computer Networks and the Internet</b>	<b>3</b>
1.1	Internet overview . . . . .	3
1.1.1	Devices . . . . .	3
1.1.2	Communication Links . . . . .	3
1.1.3	Packet switches . . . . .	3
1.2	Service view of the internet . . . . .	3
1.2.1	Provider of services to apps . . . . .	3
1.2.2	Programming interface to apps . . . . .	3
1.3	Protocols . . . . .	3
1.3.1	Definition . . . . .	3
1.3.2	Required . . . . .	3
1.4	Network edge . . . . .	4
1.4.1	Access networks . . . . .	4
1.4.2	Physical media . . . . .	4
1.5	Network core . . . . .	4
1.5.1	Interconnected routers . . . . .	4
1.5.2	Delay, loss and throughput in Packet switched networks	4
1.5.3	Queing, delay, loss . . . . .	7
1.5.4	Sources of delay . . . . .	8
1.5.5	Packet loss . . . . .	9
1.6	Protocol Layers . . . . .	11
1.6.1	iso/osi Reference Model . . . . .	12
1.7	Network Security . . . . .	12
1.7.1	Malware . . . . .	12
1.7.2	Packet "sniffing" & IP Spoofing . . . . .	12
<b>2</b>	<b>Application layer</b>	<b>13</b>
2.1	Web and HTTP . . . . .	13
2.1.1	Client-Server Architecture . . . . .	13
2.1.2	Process Communicating . . . . .	13
2.1.3	Process . . . . .	13
2.1.4	Sockets . . . . .	14
2.1.5	Protocols . . . . .	14
2.1.6	HTTP . . . . .	16
2.1.7	User-server state: Cookies . . . . .	17
2.1.8	Web caches (proxy server) . . . . .	18
2.1.9	conditional GET . . . . .	18
2.2	FTP . . . . .	18

2.3	Electronic mail: SMTP, POP3, IMAP . . . . .	18
2.3.1	POP3 . . . . .	19
2.3.2	IMAP . . . . .	19
2.4	DNS . . . . .	19
2.4.1	DNS Services . . . . .	20
2.4.2	DNS Records . . . . .	20
2.4.3	Attacking DNS . . . . .	21
2.5	Principles of network applications . . . . .	22
2.5.1	Server/client . . . . .	22
2.5.2	P2P . . . . .	22
2.6	P2P Apps . . . . .	22
<b>3</b>	<b>Transport Layer</b>	<b>22</b>
3.1	Transport Services & Protocols . . . . .	22
3.2	Multiplexing and Demultiplexing . . . . .	23
3.2.1	Port . . . . .	23
3.2.2	Socket . . . . .	23
3.3	Demultiplexing . . . . .	24
3.3.1	Connectionless Dmuxing . . . . .	24
3.3.2	Connection-oriented Dmux . . . . .	24
3.4	Connectionless Transport UDP . . . . .	25
3.4.1	Why UDP? . . . . .	26
3.5	Principles of Reliable Data Transfer . . . . .	26
3.5.1	RDT: Getting Started . . . . .	27
3.5.2	Dependency between event & state . . . . .	27
3.5.3	RDT 1.0 . . . . .	27
3.5.4	RDT 2.0 . . . . .	27
3.5.5	RDT 2.1 . . . . .	29
3.5.6	RDT 2.2: A NAK-free protocol . . . . .	30
3.5.7	RDT 3.0: Channels with errors and loss . . . . .	30
3.6	Pipelined Protocols . . . . .	32
3.6.1	Got-back-N . . . . .	32
3.6.2	Selective repeat . . . . .	32
3.7	TCP . . . . .	32
3.7.1	Overview . . . . .	32
3.7.2	TCP seq Number Acks . . . . .	32
3.7.3	Round Trip Time, Timeout . . . . .	32
3.7.4	Retransmission . . . . .	32
3.7.5	Flow Control . . . . .	32
3.7.6	Connection Management . . . . .	32

3.7.7 Principles of Congestion Control . . . . .	32
--	----

# 1 Computer Networks and the Internet

## 1.1 Internet overview

### 1.1.1 Devices

- host = end system, runs apps

### 1.1.2 Communication Links

- fiber, copper, radio, satellite

### 1.1.3 Packet switches

- routers and switches

## 1.2 Service view of the internet

### 1.2.1 Provider of services to apps

- Web, VoIP, email, games, eCommerce, social net

### 1.2.2 Programming interface to apps

- hooks
- service options (postal)

## 1.3 Protocols

### 1.3.1 Definition

A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event.

### 1.3.2 Required

- format
- order of messages

## 1.4 Network edge

### 1.4.1 Access networks

- Wired, wireless comms links

How does one connect an edge to a router?

- Frequency division multiplexing
- Cable network is shared
- HFC: hybrid fiber coax
- fiber homes -> ISP router
- DSL
- Ethernet
- WLAN: IEEE<sub>802.11</sub>

### 1.4.2 Physical media

- Guided (wires)
- Unguided (radio)
- Physical link (transmitter, { $x$ }, receiver)
- Bit (propagates between)

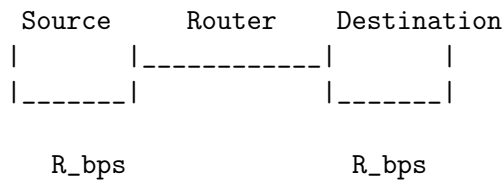
## 1.5 Network core

### 1.5.1 Interconnected routers

- mesh of interconnected routing packets transmitted at full link capacity

### 1.5.2 Delay, loss and throughput in Packet switched networks

- store and forward

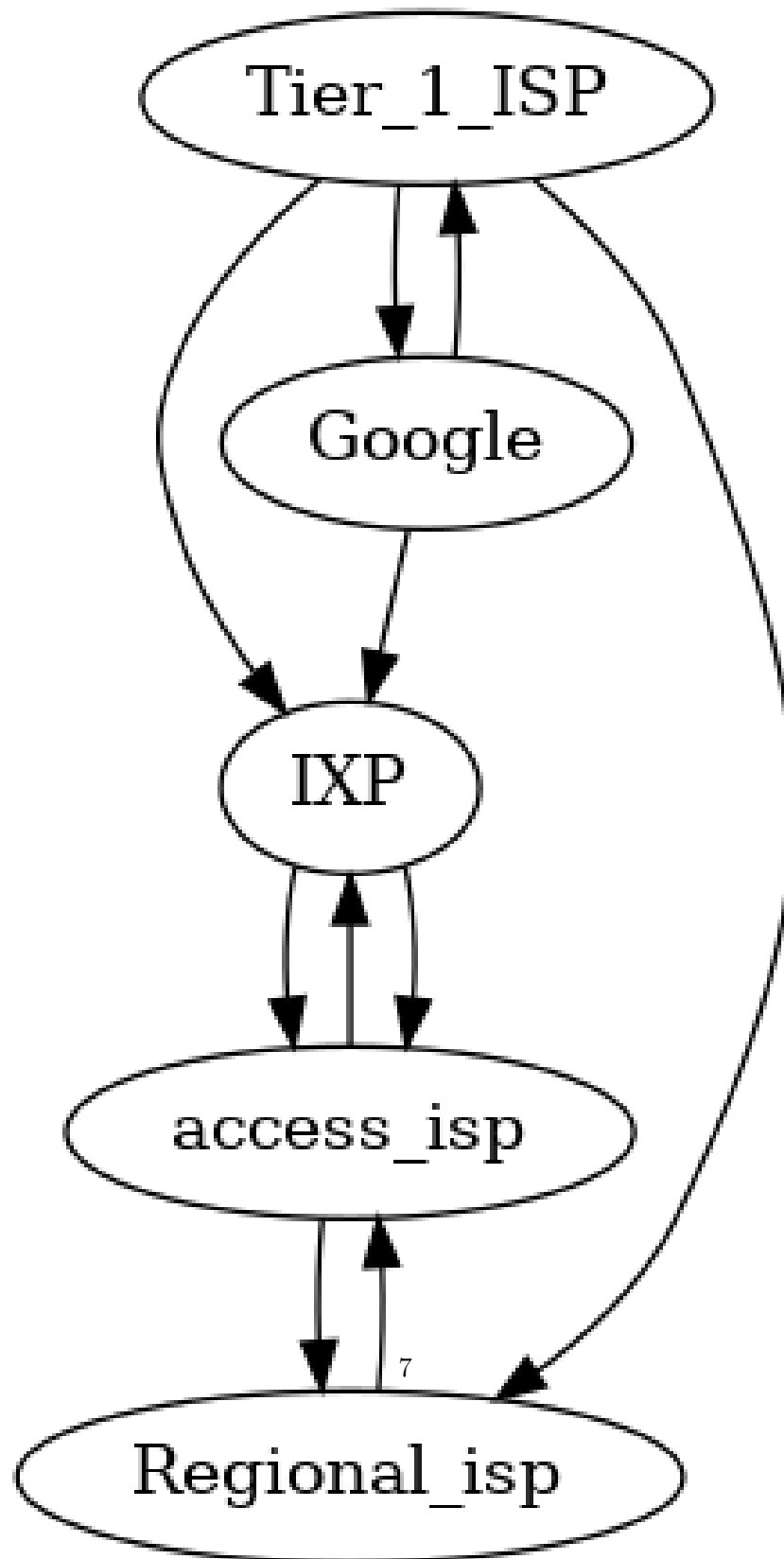


L bits per packet

- End to end delay (assumes zero propagation delay) =  $2L / R$

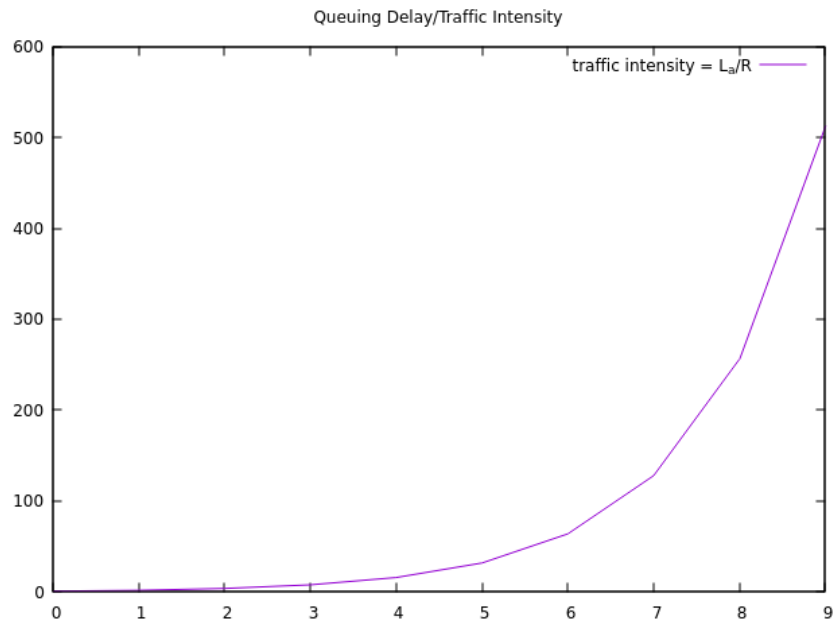


### 1.5.3 Queuing, delay, loss



#### 1.5.4 Sources of delay

- transmission
- nodal processing
- queuing



traffic intensity = $L_a/R$	Avg. queuing delay
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9

$L_a/R \sim 0$  : avg. q delay small

$L_a/R \leq 1$  : avg. q delay large

$L_a/R > 1$  : more work arriving than can be serviced average delay infinite



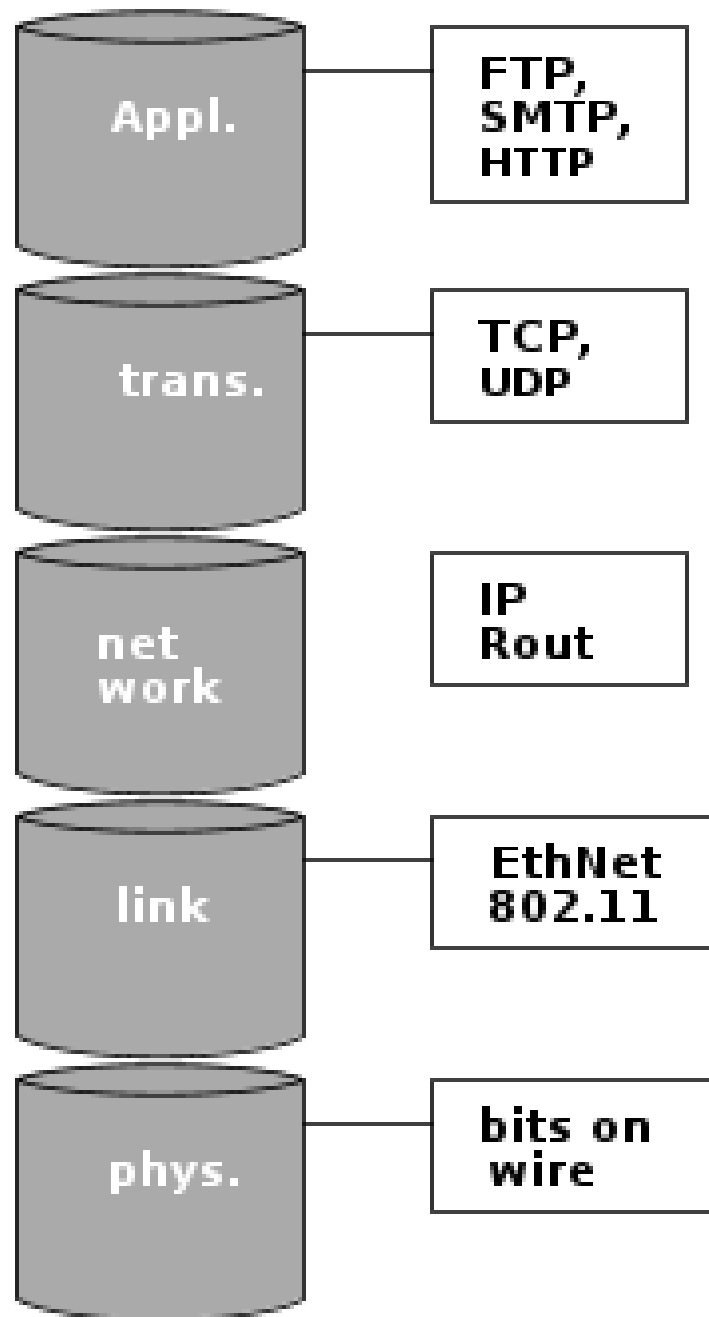
### 1.5.5 Packet loss

Buffer has finit capacity

packet  $\rightarrow$  full queue = dropout

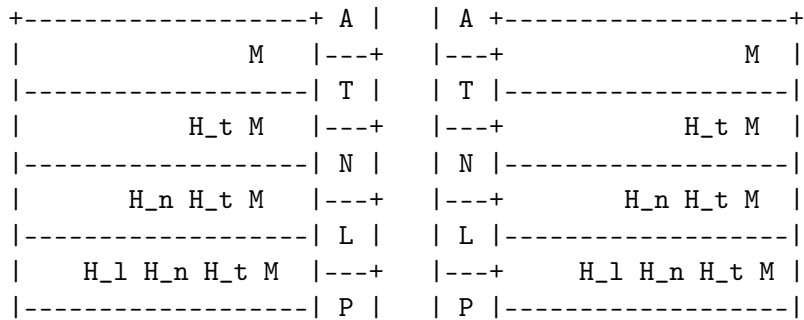


## 1.6 Protocol Layers



### 1.6.1 iso/osi Reference Model

Encapsulation



Source → Switch → Router → Destination

## 1.7 Network Security

Internet was originally designed to be used by mutually trusting users attached to a transparent network.

### 1.7.1 Malware

- **Virus:** self-replicating infection by receiving/executing object (email attachment)
- **Worm:** self-replicating infection by passively receiving object that gets itself executed
- **Spyware:** record keystrokes, web sites visited, upload info to

Infected host can be enrolled in botnet, used for spam. DDOS attacks.

- **DDOS attacks:** make resources unavailable to legitimate traffic by overwhelming with bogus traffic

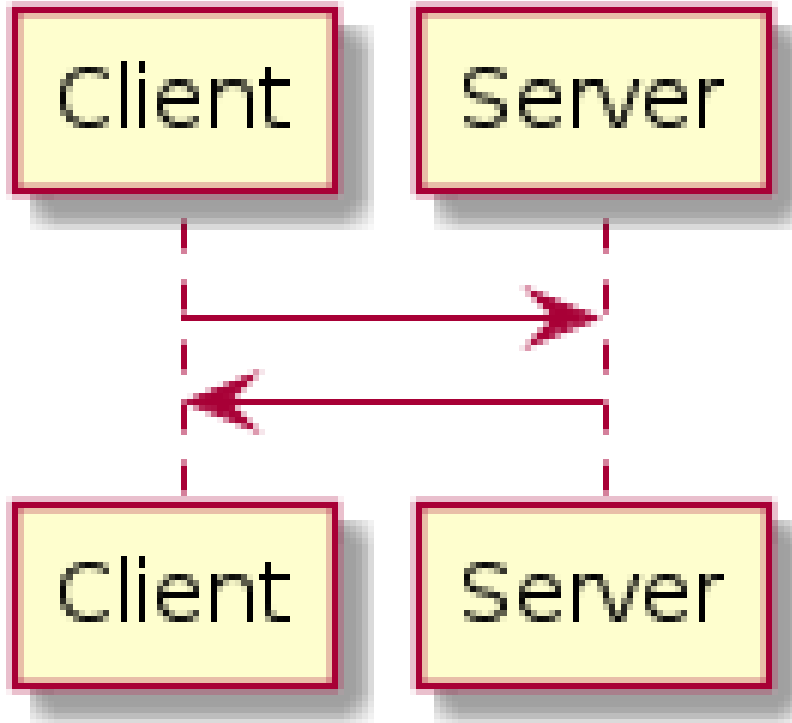
### 1.7.2 Packet "sniffing" & IP Spoofing

- Broadcast media
- Promiscuous network interface reads/records all packets passing by
- Send packet with false source address

## 2 Application layer

### 2.1 Web and HTTP

#### 2.1.1 Client-Server Architecture



Client	Server
Communicates with server	Always on host
May be intermittantly connected	Permanent IP
May have dynamic IP	Data centers for scaling
Do not communicate directly with each other	

#### 2.1.2 Process Communicating

Client process initiates comms, server process waits for contact

#### 2.1.3 Process

- Running within a host

- Withing same host, two processes communicating using inter-process communiation (defined by OS)
- Processes in different hosts communicated by exchanging messages

#### 2.1.4 Sockets

v	^
+---v-----+	+---^-----+
_V_V_V_V_	_V_V_V_V_

#### Addressing Processes

- to receive messages, process must have ID
- IP = 32 bit
- identified = IP + Port
- HTTP server: 80
- Mail server: 25

#### Application Layer Protocol

Defines:

- dypes of messages exchanged (eg: request, response)
- msg syntax (fields and delineation)
- msg semantics

#### 2.1.5 Protocols

Open protocols:

- defined in RFC
- allows for interoperatbility
- eg: http, smtp

Proprietary protocols:

- eg. sype

Transport service for an app

- 100% reliable?
- can tolerate loss
- low latency
- multimedia, minimum throughput
- "elastic apps" whatever throughput
- encryption data integration

Different apps need different architecture to accommodate all user requirements

### **TCP (Transmission Control Protocol)**

- Reliable transport send → receive
- Flow control (doesn't overwhelm receiver)
- Congestion control (throttle sender when network overloaded)
- Does not provide: timing, minimum throughput guarantee
- Connection oriented: setup required between client & server process

### **UDP (User Datagram Protocol)**

- unreliable data transfer between sending and receiving
- does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup

<b>App</b>	<b>App layer protocol</b>	<b>underlying transport protocol</b>
email	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming	http [rtp 1889]	TCP or UDP
VoIP	SIP, RTP, proprietary	TCP or UDP

### 2.1.6 HTTP

HTTP is a stateless protocol, server maintains no info about previous client requests **uses TCP**

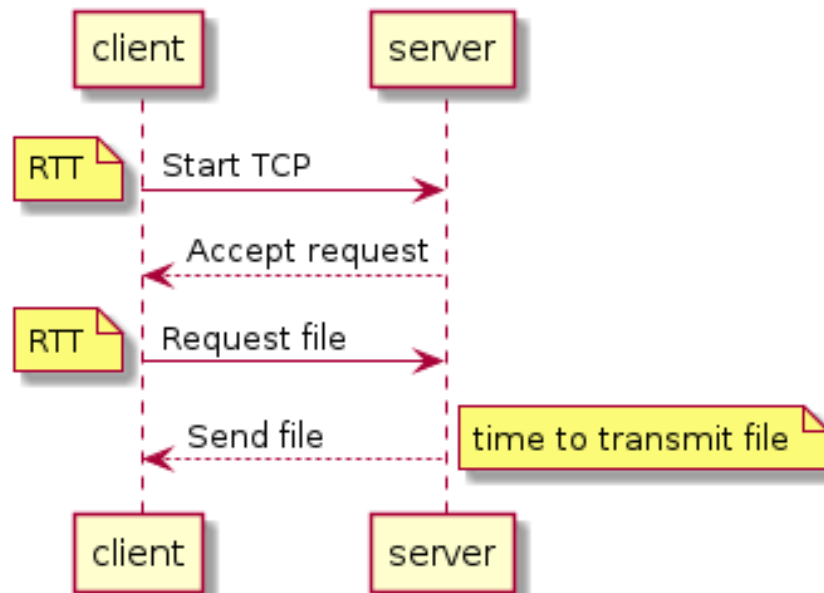
- client initiates
- server accepts
- http messages (application-layer protocol messages) exchanged between browser (http client) and web server (http server)
- TCP connection closed

#### **non-persistent**

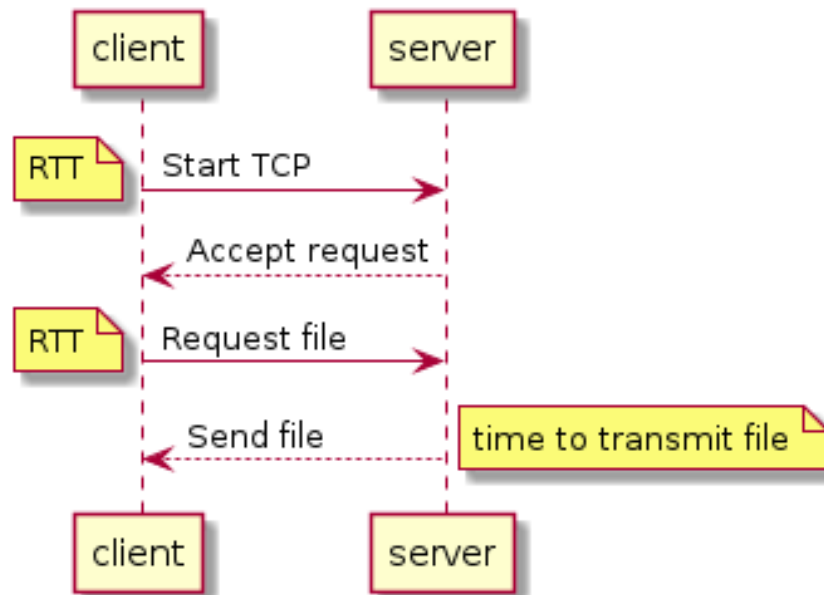
- At most one object sent over TCP connection
- connection then closed
- downloading multiple objects required multiple connections

#### **persistent**

- multiple objects can be sent over single TCP connection between client, server







### 2.1.7 User-server state: Cookies

Four components:

1. Cookie header line of http response message
2. cookie header line in next http request message
3. cookie file kept on user's host, managed by browser
4. backend database at website

**uses**

- authentication
- shop cart
- recommend
- user session state (webmail)

**cookies and privacy**

- permit site to learn about users

### 2.1.8 Web caches (proxy server)

- acts as both client and server
- typically installed by ISP (uni, company, residential)
- reduce response time
- reduce traffic on access link

### 2.1.9 conditional GET

- **goal:** don't send object if cache has up to date cached version
- **cache:** specify date of cached copy in http request
- **server:** response contains no object if cached copy up to date

## 2.2 FTP

File Transfer Protocol

- to/from remote host
- client/server model
  - initiated by client
  - server: remote host
- Ftp: RFC 959
- Ftp server: port 21

## 2.3 Electronic mail: SMTP, POP3, IMAP

- User agent
- mail server
- SMTP

Uses **TCP on port 25** Three phases:

- handshaking
- Transfer
- closure

### 2.3.1 POP3

- download & delete
- cannot re-read file after client change
- stateless across sessions
- download-and-keep copies and different clients

### 2.3.2 IMAP

- keeps messages in one place:server
- allows user to organize messages in folders
- keeps user state accross sessions:
  - names of folders and mappings between message IDs and folder name

## 2.4 DNS

Domain Name System

- Names map IP to readable format
- Distributed database implemented in hierarchy of many name servers
- application-layer protocol: hosts, name servers communicate to resolve names (address/name translation)
  - Note: core interenet function implemented as application-layer protocol complexity at network's edge
- **Root DNS Servers** over 400 worldwide (2016 numbers)
- **Top-level domain (TLD) servers** for each of {com, org, ned, edu, gov, ie, at, jp, etc) there is a server or server cluster
- **Authoratative DNS servers** publicly accessible records that map the names of the host companies to IP addreses

### 2.4.1 DNS Services

- Hostname to IP address translation
- host aliasing
  - canonical alias names
- mail server aliasing
- load distribution
  - replicated web servers: many IP addresses correspond to one name

#### Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance
- doesn't scale

### 2.4.2 DNS Records

DNS: distributed db storing resource records (RR)

RR Format: (name, value, type, ttl)

**type=A**

- name is hostname
- value is IP

**type=NS**

- name is domain
- value is hostname of authoritative name server for this domain

**type=CNAME**

- name is alias for some "canonical" real name

- www.ibm.com
  - servereast.backup2.ibm.com
  - value is canonical name

**type=MX**

- value is name of mail server associated with name

### **2.4.3 Attacking DNS**

#### **DDoS attacks**

- Bombard servers with traffic
  - Not successful to date
  - traffic filtered
  - local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD (top level domain) servers
  - potentially more dangerous

#### **Redirect attacks**

- man in the middle attacks
  - intercept queries
- DNS poisoning
  - send bogus replies to DNS server, which caches

#### **Exploit DNS for DDoS**

- send queries with spoofed source address: target IP
- requires amplification

## 2.5 Principles of network applications

### 2.5.1 Server/client

- send one copy  $F/u_s$
- send  $N$  copies  $NF/u_s$

Client must download file copy

- $d_{\min} = \min$  client dl rate
- min client download time:  $F/d_{\min}$

#### Distribution time

$$D_{c-s} > \max\{NF/u_s, F/d_{\min}\}$$

### 2.5.2 P2P

Max upload rate:  $u_s + \sum u_i$

distribution time (*increases linearly in  $N$* ):

$$D_{p2p} > \max \{ F/u_s, F/d_{\min}, NF/(u_s + \sum u_i) \}$$

## 2.6 P2P Apps

Commonly used to distribute software

Distributed hash table (DHT)

DHT: a distributed P2P database

## 3 Transport Layer

### 3.1 Transport Services & Protocols

- Provide logical communication between app processes running on different hosts
- Transport protocols run in and systems
  - Send side: breaks app messages into segments, passes to network layer
  - receiver side: reassemble segments into messages, passes to app layer
- More than one transport protocol available to apps
  - Internet: TCP & UDP

## **3.2 Multiplexing and 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

### **3.2.1 Port**

Simply a number used by a particular software to identify its data coming from the internet

### **3.2.2 Socket**

IP Address + Port num. Used by another computer to send data to software on a particular machine

- IP = Machine
- Port = Software

### 3.3 Demultiplexing

- Host receives IP datagrams
  - Each datagram has source IP address, destination IP address
  - Each datagram carries one transport-layer segment
- Host uses IP address & port numbers to direct segment to appropriate socket

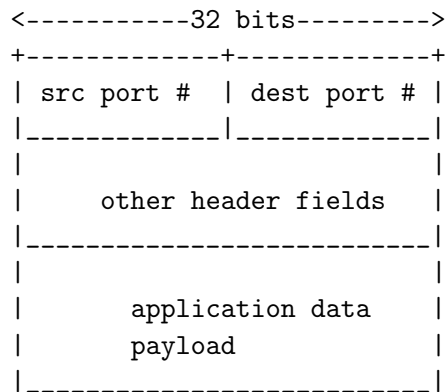


Figure 1: TCP/UDP segment format

#### 3.3.1 Connectionless Dmuxing

When host receives UDP segment:

- checks destination port number in segment
- directs UDP segment to socket with that port number

IP datagrams with some destination port number, but different source IP and/or source port numbers will be directed to same socket at destination.

#### 3.3.2 Connection-oriented Dmux

TCP socket identified by 4-tuple:

- Source IP address



- Source port number
- Destination IP address
- Dest port number

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 different socket for each request

### 3.4 Connectionless Transport UDP

- No handshaking between UDP, sender, receiver
- each UDP segment handled independently of others

UDP uses:

- streaming multimedia apps (loss tolerant, rate sensitive)
- DNS
- best effort service

UDP segments may be

- lost
- delivered out-of-order to app

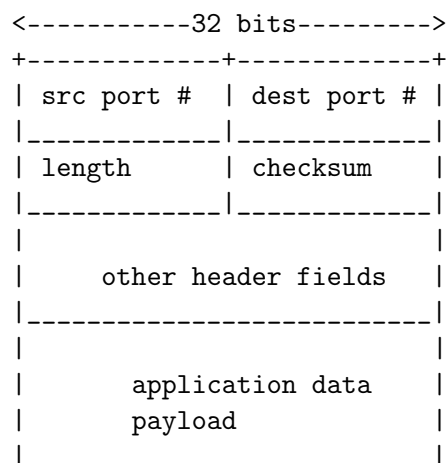
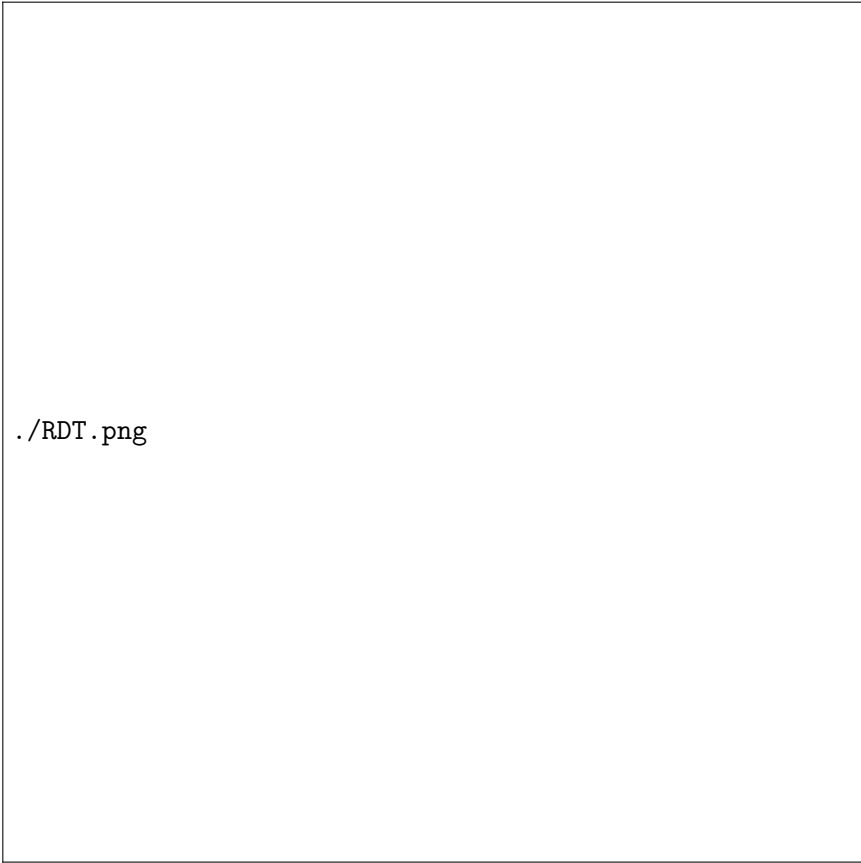


Figure 2: UDP segment format

#### 3.4.1 Why UDP?

- No connection establishment (which can add delay)
- Simple: no connection state at sender, receiver
- Small header size
- No congestion control: UDP can blas away as fast as desired

### 3.5 Principles of Reliable Data Transfer



./RDT.png

Figure 3: Reliable Data Transfer (RDT)

### 3.5.1 RDT: Getting Started

Relies on four functions

`rdt_send()`

`deliver_data()`

`udt_send()`

`rdt_rcv()`

### 3.5.2 Dependency between event & state

#### 3.5.3 RDT 1.0

Reliable data transfer over a reliable channel. Underlying channel is perfectly reliable

- no bit errors
- no loss of packets

Separate FSMs for sender, receiver:

- sender sends data into underlying channel
- receiver reads data from underlying channel

#### 3.5.4 RDT 2.0

Channel with bit errors: underlying channel may flip bits in packet

- checksum to detect bit

**Question:** 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



`./eventStateDependency.png`

Figure 4: Event  $\rightarrow$  State dependency

- Sender retransmits pkt on receipt of NAK

**FATAL FLAW** ACK/NAK can be corrupted

- sender doesn't know what happened at receiver
- can't just retransmit possible duplicate

Handling duplicates:

- Sender retransmits current pkt if ACK/NAK corrupted
- Sender adds sequence number to each pkt
- Receiver discards (doesn't deliver up) duplicate pkt

Stop and wait:

- sender sends one packet, then waits for receiver to respond

### 3.5.5 RDT 2.1

Sender:

- Seq number added to pkt
- Two sequence numbers (0,1) will suffice
- Must check if ACK/NAK corrupted
- Twice as many states
  - states must "remember" whether "expected" pkt should have sequence number of 0 or 1

Receiver:

- Must check if received packet is duplicate
  - State indicates wheter 0 or 1 expected pkt sequence number

Note: receiver can not(!) know if its last ACK/NAK received okay at sender

### 3.5.6 RDT 2.2: A NAK-free protocol

+Same functionality as **RDT 2.1**, using ACKs only

- Instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must explicitly include sequence number of packet being ACKed
- Duplicate ACK at sender results in same action as NAK: retransmit current pkt

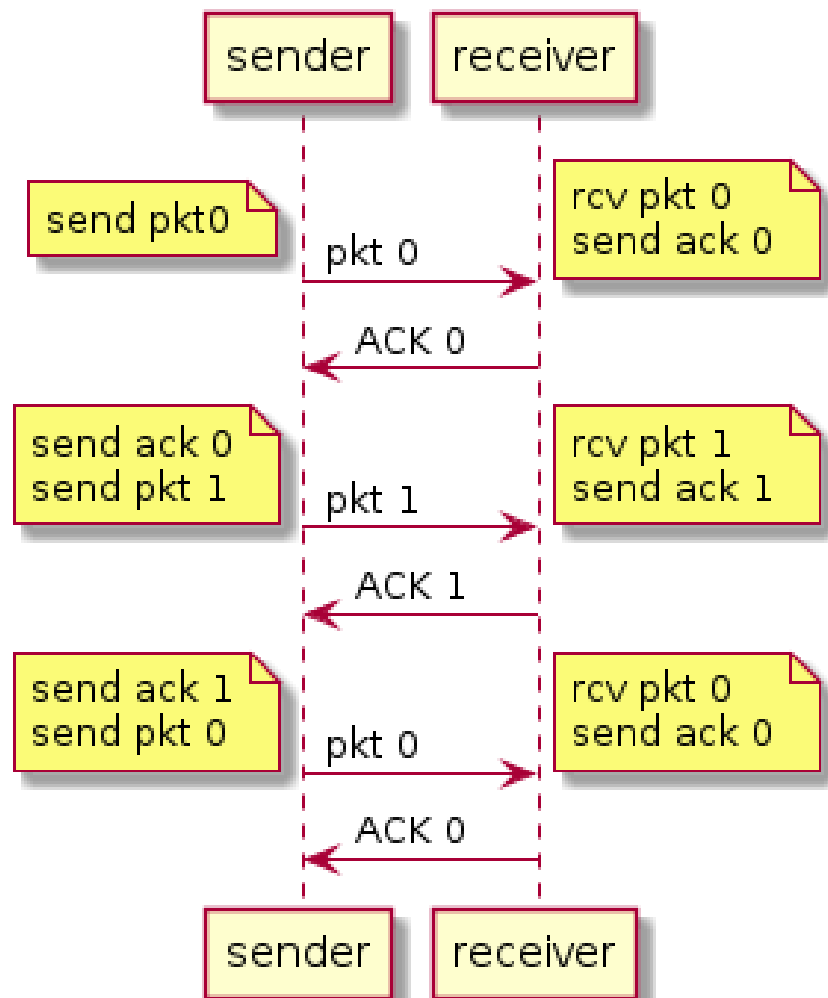
### 3.5.7 RDT 3.0: Channels with errors and loss

**New assumption:** underlying channel can also lose packets (data, ACKs)

- checksum, sequence number, ACKs, retransmission will be of help ... but not enough

**Approach:** Sender waits reasonable amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost)
  - retransmission will be duplicate, but sequence numbers already handles this
  - receiver must specific sequence number of pkt being ACK
- Requires countdown timer



## 3.6 Pipelined Protocols

### 3.6.1 Got-back-N

### 3.6.2 Selective repeat

## 3.7 TCP

### 3.7.1 Overview

### 3.7.2 TCP seq Number Acks

### 3.7.3 Round Trip Time, Timeout

### 3.7.4 Retransmission

### 3.7.5 Flow Control

### 3.7.6 Connection Management

### 3.7.7 Principles of Congestion Control