OS

- Purpose: Hardware Abstraction; Resource Management (Time & Space Multiplexing)

User Mode VS Kernel Mode

- *User Mode*: cannot issue privileged instructions; can access parts of memory allowed by kernel mode code
- *Kernel Mode*: can issue all instructions; access all memory
- *Privileged*: instructions/memory locations that can interfere with other processes

I/O Devices

- Device → Controller → Driver
- Accessed via system calls (I/O calls are privileged instructions)
- Can be done using:
  - Busy Waiting: Driver sits in tight loop; poll device (see if done/busy)
  - Interrupts: Device sends interrupt to driver when done
  - Direct Memory Access (DMA): chip controls flow of bits between memory and controller without CPU intervention; chip causes interrupt when done

System Calls

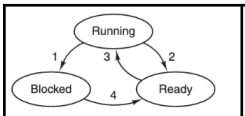
- *System Call*: allows user mode program/process to request a service from OS kernel
  - switches user→kernel mode; significant performance overhead
  - Process, File, Directory Management; kill process, get time

1. Put the system call number in a place where OS expects it (register)
2. Execute a TRAP instruction to switch from user mode to kernel mode and start execution at a fixed address within the kernel
3. The kernel code that starts following the TRAP examines the system-call number and dispatches to the correct system-call handle
4. System call handler runs
5. Once it has completed its work, control may be returned to the user program
6. The system call may block the caller, then the OS will look around to see if some other process can be run next

Inter-Process Communication

Process

- *Process*: abstraction of a running program; encapsulates all resources/information needed to run a program

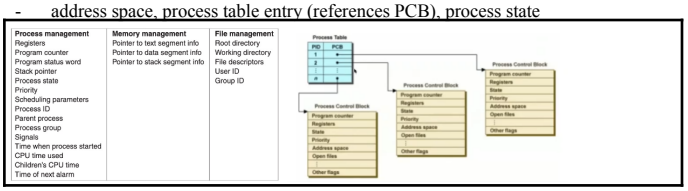


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Running: using CPU
Ready: runnable, but temporarily stopped
Blocked: unable to run until external event happens

Process Table

- linked list of PCBs (Process Control Blocks)
  - *PCB*: contains state required to resume a process
- lCurrent process runs on RAM
- Not-currently-running process: info stored in process table
- address space, process table entry (references PCB), process state



Threads

- *Thread*: sequential execution stream within the process; entities scheduled for execution by CPU
- PRO: parallel; shared address space; reduced overhead (easier create, destroy, switch, communicate); □ I/O
- CON: not help performance when CPU bound (more threads than CPU cores)

Per Process Items (shared among threads)	Per Thread
- Address space - Global Variables - Open files - Child processes - Pending alarms - Signals and signal handlers - Accounting information	- Program counter - Registers - Stack - State Threads can read, write or delete other thread's stacks

Multiprogramming

- allows multiple programs to share CPU and run in (pseudo)parallel
- but CPU can only run one program at a time – OS gives illusion of parallelism by switching between processes quickly
  - processes are sequential but execution is interleaved
  - programs cannot rely on timing assumptions

Race Conditions

- when multiple processes/threads read/write from common storage

```
stack //global shared variable

void  my_stack_function() {
  if (isEmpty(stack)) return;
  int s = pop(stack);

    //do something with s...

}
```

A B

Another potential execution transcript

1. A tests isEmpty(stack)⇒ false
2. B tests isEmpty(stack)⇒ false
3. A pops ⇒ stack is now empty
4. B pops – Exception?

Avoiding Race Conditions

- *Mutual Exclusion*: ensure that 1 process exists in the critical region at any given time
- *Critical Region (CR)*: part of program where shared resource/data is accessed
- Conditions to achieve good solution
  1. No two processes may be simultaneously inside their CRs
  2. No assumptions can be made about speed and number of CPUs
  3. No process running outside its CR can block other processes
  4. No process should have to wait forever to enter CR

Deadlocks

4 Conditions for deadlock

1. Mutual Exclusion: each resource can only be assigned to 1 process at a time
2. Hold-and-Wait: process currently holding resources can request new resources
3. No-Preemption: resources previously granted cannot be forcibly taken away
4. Circular Wait: circular list of 2 or more processes; each waiting for resource held by next member of chain

Dealing with Deadlocks

1. Ignore the problem
2. Detection (Resource allocation graph and graph algorithms) and recovery
3. Avoid by careful resource allocation
4. Prevention

Process Scheduling

- *Scheduling Algorithm*: determines which process to run next
- General Scheduling Goals
  - *Fairness*: giving each process a fair share of the CPU
  - *Policy Enforcement*: seeing that stated policy is carried out
  - *Balance*: keeping all parts of the system busy
- When to schedule: process created/exits/blocks, I/O interrupt, clock interrupt

Preemptive VS Non-preemptive Scheduling

- *Preemptive*: pick a process and let it run for a quantum (maximum fixed amount of time); clock interrupt at the end of time interval to give control to CPU back to scheduler
- *Non-preemptive*: pick process and let it run for as long as it wants until blocks/voluntarily releases to CPU
- Context Switching: happens every time OS switches process; expensive

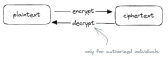
Scheduling Environments

Batch	Interactive	Real Time
-Don't require user interaction -A set of (potentially periodic) jobs that need to run	-Interactive users expect fast response times/responsiveness from the system	-Need to meet deadlines -Not really focused in this subject
Can be Preemptive or Non-Preemptive	Preemptive -by interrupting the process after a fixed time, it ensures that every process will produce a fast response -rather than having to wait, a very long time for one process to finish before moving onto the next one, which may lead to the user having to wait and thus not achieving fast interactivity	Can be Preemptive or Non-Preemptive -possibly schedule the process with fastest deadline first
-Throughput: maximise jobs per hour -Turnaround time: minimise time between submission and termination -CPU utilisation: keep CPU busy all the time	-Response Time: respond to requests quickly -Proportionality: meet users' expectations	-Meeting deadlines: avoid losing data -Predictability: avoid quality degradation in multimedia systems
-[Non-Pre] First-Come First Served (FCFS) -[Non-Pre] Shortest Job First (SJF) -[Pre] Shortest Remaining Time Next (SRT)	-[Pre] Round Robin	-[Pre] Priority Scheduling

Memory Management

- PRO: support multiprogramming; security (isolation between process and OS memory); enable running processes with large memory requirements
- *Memory Manager*: manages (part of) memory hierarchy; keep track of which parts of memory are in use; allocate when needed – deallocate when done
- *Swapping*: bringing in each process in its entirety, running it for a while

Cryptography



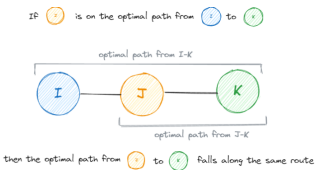
Cryptography: encrypt plaintext to ciphertext such that only aut
relies on keys being secret (not algorithm)
based on mathematically hard problems (RSA: factoring produc
solving discrete logs; AES: substitution-permutation network)

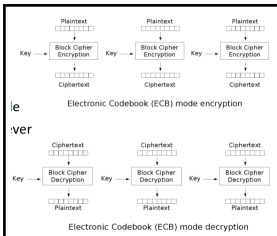
CIA

- *Confidentiality*: only sender and intended receiver should understand contents of transmitted message
- *Authentication*: establish identities of one or both endpoints
- *Integrity*: ensure that messages are not altered

Symmetric Cryptography

- Same key used for encryption and decryption
- *Advanced Encryption Standard (AES)*
  - Break data into blocks and encrypt each block
  - Modes of operation: ECB, CBC – determines how each block is treated/linked
  - *Electronic Codebook (ECB)*

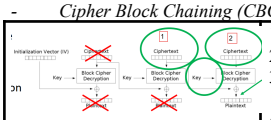




1. plaintext divided into fixed-sized blocks
2. blocks individually and independently encrypted using encryption key
3. each block decrypted individually

- PRO: simple & parallelizable; no race conditions
- CON: deterministic (identical blocks produce identical ciphertext); vulnerable to plaintext attacks; limited diffusion (changes to 1 block only affect that block)
- should never be used

- *Cipher Block Chaining (CBC)*



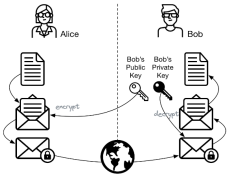
1. plaintext divided into fixed-sized blocks
2. create random Initialisation Vector (IV)
3. first block XOR-ed with IV to create ciphertext
4. subsequent ciphertexts created by XOR-ing with previous cipher block

- PRO: fixed problems with ECB; decryption can be parallel; loss/corruption of block/IV affects at most 2 blocks
- CON: encryption must be done sequentially

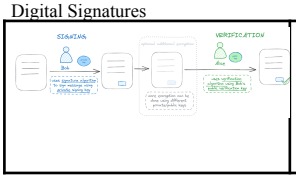
Symmetric Cryptography

- Two different keys for encryption and decryption
- CONS: slower, not suitable for large amounts of data/multiple blocks

- Alice generates her key pair (PublicKey, PrivateKey)
- She posts her PublicKey online
- Bob generates a SK with symmetric encryption
- Bob runs CipherText := Encrypt(PublicKey, SK)
- Bob sends the CipherText to Alice
- Alice runs SK := Decrypt(PrivateKey, CipherText)
- Now Alice and Bob share a SK and can exchange Symmetrically encrypted messages
- This is sort of how TLS (HTTPS) works
  - A lot of further detail, but in principle, TLS is a key exchange protocol



- Digital Signatures



- private key for encryption; public key for authentication/verification
- provides non-repudiation (cannot deny that Bob didn't sign the document)
- integrity (no tampering)
- authentication/verification

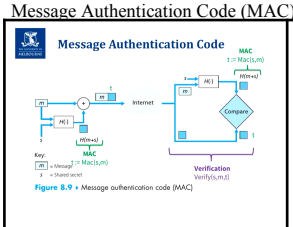
- Hash Digest

- For large documents instead of digital signatures
- Hash Function
- lossy compression function
- should look random
- collision resistant
- resulting has has constant length

$$\text{hard to find } x \neq x' \text{ such that } h(x) = h(x')$$

- diff with Encryption: does not need key; one-way

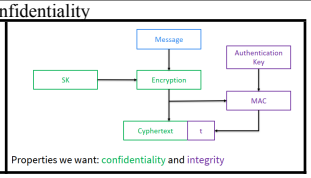
- Message Authentication Code (MAC)



- Provides Integrity
  - Sender use shared secret authentication key (s) and message to be verified (m) to generate a tap (t) using a MAC function (hash function)
  - Append tap with message to be sent to receiver
  - Receiver regenerates the tap (t') using s and m and compares to tap sent

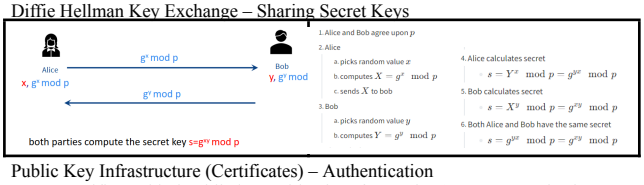
- Authenticated Encryption – Integrity + Confidentiality

- General construction: Encrypt-then-Mac:
  - C:=Encrypt(SK, m) Message m
  - t := Mac(S, C) Secret key SK of symmetric encryption; Authentication key s used to generate t
- Verify: if Verify(s,t,c) returns 0, do not decrypt
- Examples: AES-GCM, AES-OCB, AES-CCM



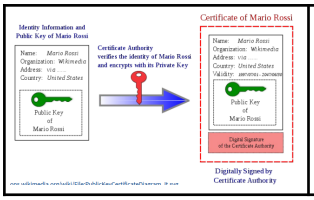
Properties we want: confidentiality and integrity

- Diffie Hellman Key Exchange – Sharing Secret Keys



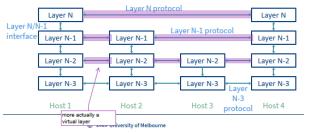
1. Alice and Bob agree upon p
2. Alice
   a. picks random value x
   b. computes $X = g^x \mod p$
   c. sends X to bob
3. Bob
   a. picks random value y
   b. computes $Y = g^y \mod p$
4. Alice calculates secret
   - $s = Y^x \mod p = g^{yx} \mod p$
5. Bob calculates secret
   - $s = X^y \mod p = g^{xy} \mod p$
6. Both Alice and Bob have the same secret
   - $s = g^{yx} \mod p = g^{xy} \mod p$

- Public Key Infrastructure (Certificates) – Authentication
  - Certificate: bind public key to identity of an entity (person, organisation, website etc.); signed by trusted entities (Certificate Authorities: explicitly trusted; predefined in OS; root cert is self-signed)

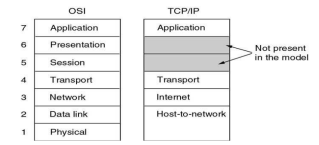


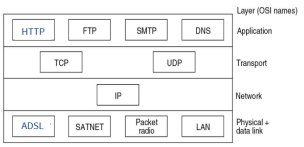
Consists of: Version Number, Serial Number, Signature Algorithm ID, Issuer Name, Validity Period (not before and not after), Subject name, Subject Public Key Info (Public Key Algorithm and Subject Public Key) , Issuer Unique Identifier (optional), Subject Unique Identifier (optional), Extensions (optional), Certificate Signing Algorithm, Certificate Signature, subject alternative names

Network Protocols and Service Models



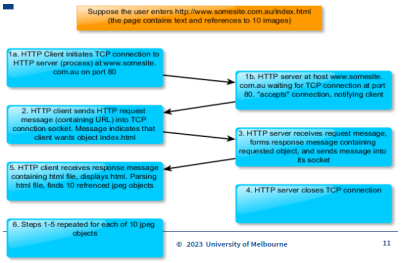
- Each layer
  - offers services to layers above it
  - uses service from layers below it
- Inter-layer exchanges are conducted according to a protocol
- *Service*: set of primitives/operations that a layer provides
- *Protocol*: rules that govern the format and meaning of packets exchanges by peers within a layer





TCP/IP is what happens on the internet; OSI is idealised model

HTTP



HTTP Method	Safe	Idempotent	Cacheable
GET	Yes	Yes	Yes
HEAD	Yes	Yes	Yes
POST	No	No	Yes/No
PUT	No	Yes	No
DELETE	No	Yes	No
CONNECT	No	No	No
OPTIONS	Yes	Yes	No
TRACE	Yes	Yes	No
PATCH	No	No	No

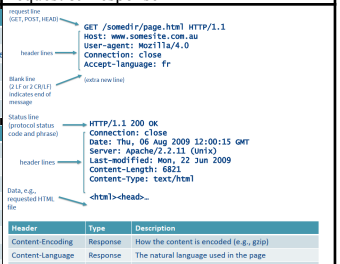
- Non-Persistent: requires 2 response times (initiate TCP; initial HTTP request) per object + file transmission time; OS overhead for each TCP connection
- Persistent: server leaves connection open after sending response
- Idempotent: multiple identical requests have same effect
- Safe: only for information retrieval, should not change state

Codes&Headers

Code	Meaning	Examples
1xx	Information	100 – server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content pres
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 try again later

Header	Type	Description
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The compression formats the client can handle
Accept-Language	Request	The natural languages the client can handle
If-Modified-Since	Request	Time and data to check freshness
If-None-Match	Request	Previously sent tags to check freshness
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Referer	Request	The previous URL from which the request came
Cookie	Request	Previously set cookie sent back to the server
Set-Cookie	Response	Cookie for the client to store
Server	Response	Information about the server

Request & Response



Header	Type	Description
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Content-Range	Response	Identifies a portion of the page's content
Last-Modified	Response	Time and date the page was last changed
Expires	Response	Time and date when the pages stops being valid
Location	Response	Tells the client where to send its request
Accept-Ranges	Response	Indicates the server will accept byte range requests
Date	Both	Date and time the message was sent
Range	Both	Identifies a portion of a page
Cache-Control	Both	Directives for how to treat cache
Etag	Both	Tag for the contents of the page
Upgrade	Both	The protocol the sender wants to switch to

DNS

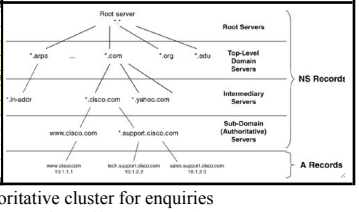
Domain Name System (4 elements): Domain name space,name servers,DNS database,resolvers
If you want to look up an address:
1. You contact the resolver
2. Resolver will contact multiple name servers
3. The name servers will tell you something about the database
4. The structure of the database is captured by the DNS on the domain name space

Domain Name characteristics: not case sensitive,up to 63 chars per constituent,255 chars per path,can be ASCII/UNICODE,

Resource Records

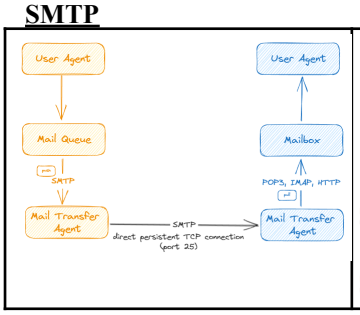
Type	Meaning	Value
SOA	Start of authority	Parameters for this zone
A	IPv4 address of a host	32-Bit integer
AAAA	IPv6 address of a host	128-Bit integer
MX	Mail exchange	Priority, domain willing to accept ema
NS	Name server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
SPF	Sender policy framework	Text encoding of mail sending policy
SRV	Service	Host that provides it
TXT	Text	Descriptive ASCII text

Name Servers



- Root Name Server: 13 globally; authoritative cluster for enquiries
- Top-level Domain Servers: (com, edu, gov, int, mil, net, org, …)
- Authoritative Servers: maintained by organisation/service provider; takes domain and subdomain→returns correct IP address
- Local DNS Server: each ISP/company/university; returns cached value if exists; acts as proxy, forwards requests up hierarchy

SMTP



- Direct, persistent TCP connection
- 7-bit ASCII
- all message's objects is in 1 message
- very chatty protocols
  - multiple transactions
  - unlike HTTPS (big pushes)

Headers the user usually sees

To:	Cc:	Bcc:
From:	Date:	Subject:
Message-Id:	In-Reply-To:	References:
Reply-To:	Sender:	Return-Path:
Keywords:	Received:	

Headers the user usually doesn't see.

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Note: many back-and-forth exchanges.

Slow on modern networks, where latency is larger than serialization delay.

Modern approach of one header has lower delay.

Multipurpose Internet Mail Extensions: Limited to English and not for other languages or audio/img
MIME has 5 additional message headers:
- MIME-version: identifies the MIME VERSION
- Content-Description: human readable describing contents
- Content-Id: unique identifier
- Content-Transfer-Encoding: how body is wrapped for transmission
- Content-Type: type and format of content

Type	Example subtypes	Description
text	plain, html, xml, css	Text in various formats
image	gif, jpeg, tiff	Pictures
audio	basic, mpeg, mp4	Sounds
video	mpeg, mp4, quicktime	Movies
model	vrml	3D model
application	octet-stream, pdf, javascript, zip	Data produced by applications
message	http, rfc822	Encapsulated message
multipart	mixed, alternative, parallel, digest	Combination of multiple types

Post Office Protocol(POP3 for delivery): hard to access email from multiple devices,designed to keep mail until it is fetched, and then once it is fetched by a client, it is deleted from the server. That means you can't easily download it to the next device. The client then arranges it into folders, performs searching etc. If you configure it to use on two client computers, then putting it into a folder on one won't do the same on the other. Email cannot be reread..
Internet Message Access Protocol:IMAP keeps user state across sessions;Retain mailbox contents online (server) and allow manipulation of online and offline messages and mailbox folders;Implications of server infrastructure to support high volume of IMAP users. This implies storage projections by the provider, and hence limitations,designed so that mail stays on the server, and the server has folders;The client can download email, and then delete it once it has been displayed (or cache it for offline viewing, of course, but the definitive copy is still on the server).

TCP

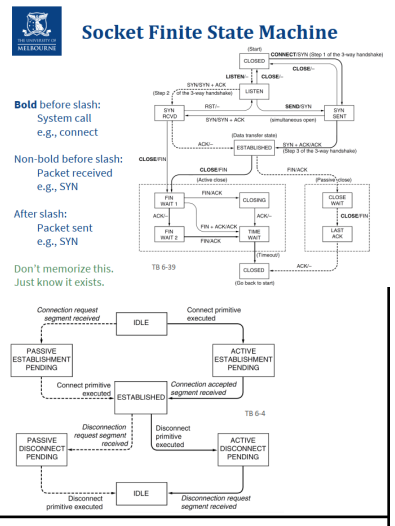
- Transport layer: tidy up end-to-end (lost/duplicated/corrupted packets)
- stream-oriented (segmenting into IP datagrams), reliable (dropped/duplicated) and in-order protocol
- *full duplex*: data in both directions, simultaneously
- *end-to-end*: exact pairs of senders and receivers
- *byte streams*: not message streams; message boundaries not preserved (no way of knowing sender has ended message)
- *buffer capable*: TCP entity can choose to buffer prior to sending or not; buffering reduces overhead (□headers) but □delay
- *segments*: 20-60 byte header + zero or more data bytes (TCP decides size of segments)
- 5-tuple address: sender IP, receiver IP, sender port#, receiver port#, protocol

Primitives

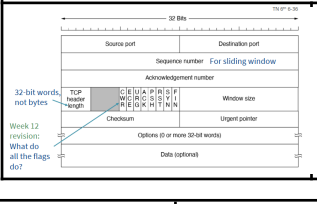
Primitive	Packet Sent	Meaning
LISTEN	(none)	Block until something tries to connect
CONNECT	CONNECTION REQ	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	[none]	Block until DATA packet arrives
DISCONNECT	DISCONNECTION REQ	This sides wants to release the connection

State	Simplified name	Description
CLOSED	Idle	No connection is active or pending
LISTEN	Pass. est.	The server is waiting for an incoming call
SYN RCVD	Pass. est.	A connection request has arrived; wait for ACK
SYN SENT	Act. est.	The application has started to open a connection
ESTABLISHED	Established	The normal data transfer state
FIN WAIT 1	Act. disc.	The application has said it is finished
FIN WAIT 2	Act. disc.	The other side has agreed to release
TIME WAIT	Act. disc.	Wait for all packets to die off
CLOSING	Act. disc.	Both sides have tried to close simultaneously
CLOSE WAIT	Pass. disc.	The other side has initiated a release
LAST ACK	Pass. disc.	Wait for all packets to die off

Sockets

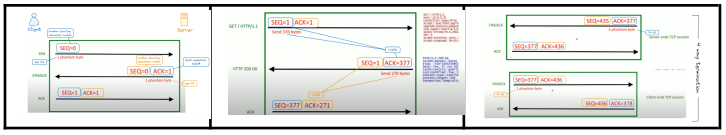


Header

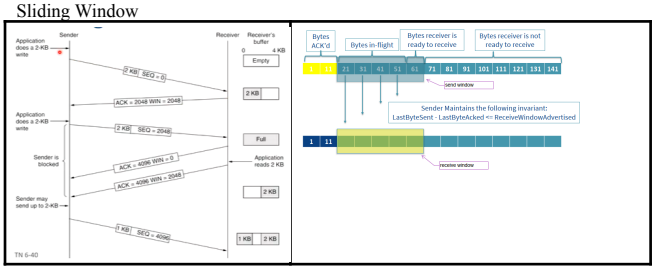


Name	Description
Source port	Sending port
Destination port	Receiving port
Sequence Number	If **SYN**=1: initial sequence number if **SYN**=0: is accumulated sequence number of the first data byte of this segment
Acknowledgement number	If **ACK**=1: next sequence number that the sender of the **ACK** is expecting
Data offset	Size of the TCP Header (20-60 bytes = 5-15 x 32-bit)
Flags	Single bit flags (**SYN, ACK, RST, FIN,** etc.)
Window size	Size of receive window – how much data the sender of this segment is willing to receive

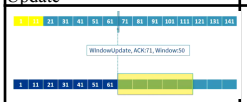
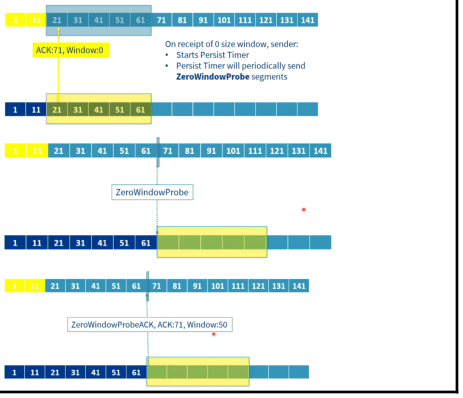
3 way handshake	SEQ and ACK incrementing	4 way termination



Sliding Window



Deadlock
- Full Window: sender window full and doesn't receive ACKs; receiver window full, doesn't receive packets
- Lost ACK: receiver waiting for a packet that will never be retransmitted

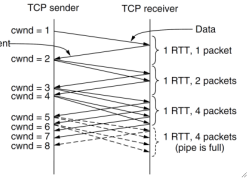
Solution 1: Send a Window Update	Solution 2: Zero Window Probe
	
data sent to application→window moves→window update	

Multi-threaded server
- Clearly a server needs to be able to handle concurrent connections from multiple clients
- This can be achieved through the usage of a multi-threaded server
- Typically, you want to have a multithreaded server where each thread is responsible for one TCP connection and we have another thread, which is the dispatching thread.
  1. When we're doing an accept and somebody's trying to contact the web, it will create a new thread or wakes up a sleeping thread
  2. The thread will go and surf the webpage, or surf the remote procedure call
  3. The dispatcher thread will start another accept and wait until another accept comes

QUIC: reliable but out of order, over UDP.Multiple byte streams instead of just one(if a packet is lost, only one stream has to wait for retransmission),TLS handshake is combined with TCP handshake (fewer RTT), QUIC has a connection ID, and uses that to re-establish the connection if an IP address changes. That means that it doesn't need to re-download data, and if there were "ssh over QUIC"(not named by the 5-tuple (2 x (IP + port) + protocol))
QUIC sockets
- TCP has two types of sockets. Listening sockets (after listen()/accept()) and connected sockets.
- QUIC has three types: Listening sockets (just like TCP listening sockets), stream sockets (basically like regular (i.e., connected) TCP sockets, able to carry one stream of data), and a new type, "connection sockets".
- Connection sockets carry meta-data rather than data of the connection.

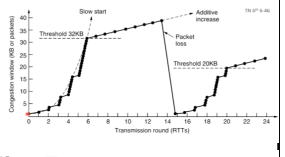
Port allocations
- Port numbers can range from 0-65535 (16 bits)
- Allocated by Internet Assigned Numbers Authority (IANA)
- Ports are classified into 3 segments:
  - Well Known Ports (0-1023)
    - 21 FTP
    - 22 SSH
    - 23 Telnet
    - 25 SMTP
    - 80 HTTP
    - 10 POP3
    - 179 BGP
    - 443 HTTPS
  - Registered Ports (1024-49151)
    - Also called "user ports" but still registered with IANA
  - Dynamic Ports (49152-65535)
    - Used when you want to connect to a host on a well
    - Also known as ephemeral ports (ephemeral → not

Congestion Window (CWND)
- additional window dynamically adjusted
- based on **network performance**
- controlled by the sender
  - knows more information: what is sent and what is received
- how it works
  1. start with small segment size
  2. if get full window of acknowledgements
     - double congestion window size
  3. grows until
     - timeout
     - threshold (SSthresh) reached

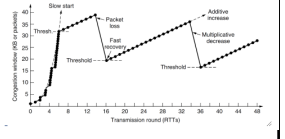


  4. if packet loss
     - SSthresh = CWND/2, start slow start again

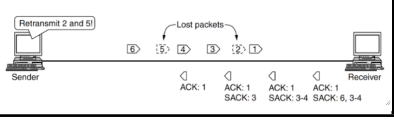
Incremental Congestion Control (Additive Increase)
5. once SSthresh is reached
- slow down growth
- Additive increase: CWND += 1MSS for each complete window
- until packet loss (4.)



Fast Recovery
4. if packet loss
- start from new SSthresh instead of original starting value
- avoid slow start phase and go straight to Incremental Congestion Co



SACK (Selective Acknowledgements)
- normal TCP packet loss
  - receiver can only ACK last contiguous packet
  - sender has to retransmit all packets from that point onward
- SACK packet loss
  - TCP header is a little longer
  - receiver can ACK non-contiguous packets
  - receiver can indicate precisely which packets were successful
    - up to 3 ranges of bytes can be specified (TCP header is still limited)
  - sender only retransmit specific missing packets
- avoid unnecessary retransmissions and timeouts



Window
- W increases once per window
  - Approximation: Each packet that arrives increases W by 1/W
- When a loss occurs (with probability p), W is halved
  - With probability p, W ← W/2
- The average increase in window size is $\frac{(1-p)}{W} - p\frac{W}{2}$
- To be in equilibrium (balance), the average increase must be 0
  - $W \approx \sqrt{2/p}$

more losses decreases window size

Rate and Fairness
- The window is sent ≤ once per round trip time (RTT), T
- Rate is $\frac{W}{T} \approx \frac{1}{T}\sqrt{2/p}$
1. For a given packet loss rate, longer RTTs get less rate
   - "RTT unfairness"
2. If RTT is small, TCP forces the packet loss rate to be high

- This formula is very approximate
  - Window only responds to one packet loss per RTT
  - Packet losses are clustered

packet loss rate is dominated by TCP, not buffer