1 Operating Systems

An OS is just a program that interfaces the hardware with the application programs (allows user programs to run efficiently on hardware). Its key functionalities are

- Provide hardware abstraction
- Manage hardware resources

Hardware abstraction means

- Creates a simpler, easier to use model of the computer
- Presents to the application a simple machine that appears to be dedicated to the software itself
- Example abstractions are: threads (processor), virtual address space (memory), files (disk), sockets (io devices like network), process (program, the code)

Resource management mean

- Orderly and controlled allocation of processors, memories, and io devices to programs wanting them
- This is because there are more processes than processors, and resources are shared between them all
- Allows running more than one program at once on a single core, each with its own abstract process

Multiplexing

- Time multiplexing means each process shares cpu time
- Space multiplexing means processes sharing memory

Computer Architecture

- Computer hardware consists of: cpu and memory
- Memory stores programs and data
- CPU does the computation. It has: registers, ALU, MMU.
- Registers is the fastest memory (register, cache, memory, disk). General purpose stores any data (rax, rbx, rcx, ...), special purpose stores data with special functionalities (pc, sp, mem, psw). Both are used for program execution.
- ALU for arithmetic logical unit, MMU for memory management unit

CPU instruction cycle, fetch-decode-execute

- Fetch, gets an instruction from memory using the program counter register (storing the mem address of the next instruction)
- Decode, understand what the instruction is
- Execute, cpu runs the instruction, may include copying between memory or registers, use alu to perform operation

Subroutines

- A function within a function
- A function contains local variables (stack) and registers
- To call a subroutine, we save the current state (registers) into a stack, push return address onto stack (the next instruction after the function call), push parameters onto stack or register, load pc with the address of the subroutine, and update stack pointer (and save the current sp to the stack).
- To exist a subroutine, goto return address from stack by setting pc to it, pop previous saved registers from stack into registers, and update state pointer (from stack).
- There are HW support for functions like call, ret, push, pop, instructions, and sp register.

Stack

- A region of memory storing info about subroutines
- A stack frame (region of stack memory) for each subroutine not yet finished
- A stack frame contains a return address at least
- Stack pointer sp register always prints to top element of the stack.
- Extra, the base pointer (bp) stores the top of the stack in this current stack frame (and the local variables are stored against the bp)

Asm quickstart

- Registers are `rax`, `rsp`
- Instructions like `pushq`, `movl`
- Note that the stack grows downwards toward lower memory address (the sp points to the highest memory address in the stackframe)

Memory boundaries

- HW support to enforce memory boundaries between processes via MMU and MEM register.
- The MMU and MEM special register allows the OS to prevent one application from modifying the memory of another through translating virtual memory to actual memory

Execution modes

- The Program Status Word (special register) provides hardware support for differentiating user mode and kernel
- A bit in the PSW will be set if the execution is in user mode
- Code running in user mode cannot issue privileged instructions, and can only access the part of memory allowed by OS
- Code in kernel modes can issue any instruction and access all memory. Only OS can run in kernel mode
- Privileged instructions are any instruction that do IO or affect the machine hardware.
- The OS kernel is the part of OS that runs in kernel mode. It contains the core of the OS with a set of functions that manage scheduling, resource allocation, and IO.

Execution under kernel mode

- Hardware (IO or timer), user software, and OS can trigger interrupts to communicate and hand over control to the OS.
- User programs can use system calls to communicate to the OS

Interrupt

- Hardware events that cause the CPU to pause execution.
- During an interrupt, the cpu enters kernel mode and executes the interrupt handler looked up from the interrupt vector (a lookup handler in OS memory) which are memory addresses of interrupt code (also in memory)
- During interrupt, the CPU sets PSW to kernel mode and sets PC with the address from the interrupt vector
- When the handle finishes, we continue execution of the interrupted application unless the OS killed the application or rescheduled it
- External interrupts are generated by external devices at unpredictable times. Clock interrupt tells the OS a certain amount of time passed. IO device interrupt tells the OS an IO operation completed
- Internal interrupts are caused by exceptions in instruction execution from CPU (the processor hands the responsibility to the OS). Error condition interrupts tells the OS that the current application did illegal actions and cannot be recovered (divide by zero). Temporary problem interrupts are error conditions that the OS can handle like bring the required page into memory.

System calls

- Allows user program to run privileged instructions using the OS kernel
- To do a syscall, the program puts the syscall number into a register that the OS expects.
- It executes a TRAP instruction that launches a syscall interrupt.
- The syscall interrupt handler in the kernel will use the syscall number to lookup the syscall handler in a table in OS memory.
- The syscall handler than runs.
- When the syscall finishes, control may be returned to the user program. The syscall could be blocking, so the OS may reschedule some other process to run during the syscall handler

Types of syscalls

- fork, open, read, write, mkdir, kill, time

2 Process and Threads

A process is

- A running program

- An abstraction by the OS, gives programs an interface of a dedicated, abstract machine to run one
- Program is code, static. Process is executing, dynamic
- Multiple process can be running the same program
- Contains: threads, address space, files, sockets

The process abstraction allows a single CPU to run multiple programs simultaneously (concurrency not parallelism). This is multiprocessing

- Allows multiple process to share running on a CPU
- CPU can only run one program at a time, so this is an illusion
- CPU switches between process to process every tens of millisecond
- Allows for efficiency and requires a scheduling algorithm for which process to run next and how long it gets to run

Four events can create a process

- System initialization
- Executing of a process creation syscall by a running process
- User request to create a new process
- Running a batch job (scheduled automated process that runs tasks repeatedly, cron jobs)

Typical events that terminates a process

- Normal exit (exit function), voluntary
- Error exit (exit function with non-zero), voluntary
- Fatal error (exceptions), involuntary
- Killed by another process, involuntary

The fork syscall will create a new process by a currently running process. It will create a clone of the currently running process (executing from the same location) with

- Different address spaces for process
- Same memory image (copied memory from parent process)
- Same values for the program counter and general purpose registers, and the same open handles

Fork will return 0 if from the child process, otherwise it returns the child process id. We usually follow the fork with another syscall from the exec family: execve will replace the currently running process with a new program, with new stack/heap/data-segments.

Process states

- Running, using the CPU
- Ready, runnable, but paused
- Blocked, unable to run until an external event (blocked by IO)

- Scheduler can move the state from running to ready by switching process, and ready to running by running this process
- Any IO will move the running process to blocked. When IO is available, block will go to ready.

The OS maintains information about a process in a process control block (PCB) data structure

- The process table has a list of PCBs
- PCB contains: process id, process state, parent process, memory management information, file descriptors, priority, CPU statistics
- The execution context (for execution by the CPU) like program counter, stack pointers are stored by threads instead.

Thread stores the execution context for a process. It represents

- Sequential execution of a set of instruction
- Contains a PC, SP, registers, stack

About threads

- To run a thread, CPU loads the execution context to the CPU's registers
- When not running, the execution context of threads is saved in OS memory into the thread control block, and the CPU registers will hold the execution context for another thread.
- Threads are executed inside a process, multiple threads can be running at the same time using multithreading

Thread control blocks stores the thread execution context. It stores: thread id, stack pointer, program counter, registers, thread states, pointer to the PCB of the parent process. The OS stores the list of TCB in its OS memory.

Address space with threads

- From low to high, code, data (global variables), heap that grows up, stack that grows down.
- Threads will share code, data, and heap. Each thread will have its own stack separated by free space

Why threads

- Multithreading can use multiple processors at the same time, to speed up the program
- Enables overlapping IO with other processes: one thread waits for IO, and another uses the CPU.

Multithread vs multiprocess

- Threads will share an address space, making sharing data between threads easier and faster.
- Threads are faster to make
- Switching between threads are faster
- Threads provides no isolation, one thread will break the entire process
- Process allows application to be parallelized beyond a single machine

- Process provides isolated execution environments
- Process has IPC overheads

3 IPC

Processes communicate using IPC to share resources/data, the mechanisms are: shared memory, pipes, files, sockets, signals.

But cooperating processes may interface with each other so we need to ensure an orderly execution. The focus is to understand and avoid race conditions in cooperating procceses.

Race conditions happen when

- Multiple threads/procceses access a shared resource
- The result depends on the timing of code execution, the outcome is not fixed and can change between runs

To avoid race conditions, define

- Critical region, a part of the program that resources are accessed/updated
- Mutual exclusion, guarantee that only one thread can be executing in the critical region at any time

We can identify the critical region as a part of code where a shared resource is accessed and updated at the same time, and is not atomic.

Features of achieve mutual exclusion

- No two processes can be simultaneously inside the critical regions (mutual exclusion definition)
- No assumptions can be made about the speeds or the number of cpus
- No process running outside the critical region can block other process (blocking other process from entering the critical region)
- No process should have to wait forever to enter the critical region (starvation definition)

The general idea is to use a lock to control access to the critical region. We acquire the lock when entering the critical region, and release the lock when exiting the critical region. And allow only the process/thread with the lock to be in the critical region.

A context switch is the method that OS uses to stop the execution of one thread and starts the execution of another thread.

The basic lock program is wouldnt work, since there may a context switch after waiting for the lock so another thread can also enter the critical region. Other methods for mutual exclusion are

- Busy waiting: strict alternation, TSL with busy waiting
- Blocking: mutex

3.1 Busy Waiting

The idea is

- When thread want to enter the critical region
- Check in a loop for lock
- Acquire lock after loop

Cons are: waste of cpu cycles, priority inversion problem, and the context switch problem. The first two cons are general for all busy waiting algorithms.

The priority inversion problem is

- Two processes with differing priority
- Scheduling means that high priority will always run when it is not blocked
- If low priority is in the critical region, and high priority is in the busy waiting
- High priority will run in a loop, and the OS will not let the low priority to run and exit the critical region, so the high priority will loop forever (starvation)

Strict alternation is a busy waiting approach in which two threads take turns running the critical region

```
int turn;

# Thread A, can only enter when turn == 0
while (turn != 0) {}
criticalRegion();
turn = 1;

# Thread B, can only enter when turn == 1
while (turn != 1) {}
criticalRegion();
turn = 0
```

Strict alternation has no context switch problem, but it introduces starvation, if thread A is busy waiting and thread B is not executing the critical region and keeping the turn in 1, prevent A to run.

Test and Set lock is hw support for locking, its instruction is (TSL register lock)

- It copies content of lock into register
- and stores a non-zero value in lock

This is atomic, so that these two operations are indivisible and the os cannot interrupt the TSL midway. To use TSL for busy waiting

- TSL register lock
- if register is zero, enter critical region
- if register is not zero, lock is used, so go back to the first step
- When leaving, set lock to zero

TSL solves both the starvation problem and the context switch problem.

Can also use the atomic compare and exchange.

3.2 Blocking

When thread wants to enter the critical region

- Check if entry is allowed through a lock
- If not, let another thread to use the CPU by blocking/yielding
- Otherwise, enter the critical region and set the lock.

Can use in place of busy waiting with the same algorithms: TSL, strict alternation.

Cons depend on the specifics of implementation, but it involves starvation, overhead of syscall, overhead of context switches.

Yielding mutex

```
mutex_lock:
TSL register, mutex
CMP register, 0
JZE ok
CALL thread_yield
JMP mutex_lock

ok:
# lock is acquired
RET

mutex_unlock:
MOV mutex, 0
RET
```

so we call `mutex_lock` when acquiring the lock, and `mutex_unlock` when releasing the lock.

Doesn't have the priority inversion problem.

A set of processes are deadlocked if each process in the set is waiting for an event that only another process in the set can cause. It may exist in all shared resource settings.

Example of deadlocks, both threads are waiting for each other's lock.

3.3 Summary

Two axes: busywaiting or blocking, and strict alternation or TSL.

4 CPU Scheduling

The scheduling units can be a single-threaded process or a thread. We will focus on single-threaded processes as the scheduling unit, but the policies are task-independent and can be used in all cases.

A context switch is when the OS changes the execution context (transiting from executing one thread to another).

- Thread context switch, switching between threads in the same context. The OS must save the current execution context to the TCB and load the execution context from the TCB.
- A process context switch switches between threads in different processes. The OS must do a thread context switch, loading state related to memory management using the PCB, and flushing the translation lookaside buffer (if virtual memory)
- The context switch always happens in kernel mode

Most processes alternate bursts of computing with IO requests

- CPU bound processes have long CPU bursts, spends most of the time computing
- IO bound process have short CPU bursts, and spends most of the time waiting

Transitions to context switches

- Process initiates an IO syscall and blocks
- Process finishes with a syscall
- IO interrupt

CPU Scheduling involves picking a process to run from a set of processes in the ready state. There are two kinds of scheduling

- Cooperative, non-preemptive scheduling. Pick a process and let it run until it blocks or voluntarily releases
- Preemptive scheduling, pick a process and run for a maximum amount of time (quantum), if it is still running at the end of the quantum, suspend it and context switch. Uses the clock timer interrupt to give control of the CPU to the OS scheduler to schedule
- Preemptive schedulers will result in more context switches because it can context switch during process execution.

Scheduling environment

- Batch system, a set of periodic jobs that need to run. The jobs don't require interaction with users. Non-preemptive scheduling or preemptive with long quantum are good choices, since there's no latency requirements
- Interactive systems, users expect fast response. Preemptive with short quantum is better, since there will be shorter waits between context switches.

Scheduling goals

- Fairness, processes get a fair share of the CPU. Comparable process (by priority) get comparable service/allocation
- Efficient use of resources. Spend most of the CPU time running user processes instead of context switching. And allowing IO bound jobs to use the CPU when they need it so they can keep the IO devices busy.

Specific goals

- Batch system, maximizing throughput (number of processes completed in a time unit), minimize turnaround time (the interval from process submission into CPU queue to task completion)
- Interactive systems, minimize response time (the time between issuing a command and getting a result, and any feedback)

4.1 Cooperative scheduling algorithms

First-come First-serve (FCFS) will run processes in the order they become ready. When the currently running process blocks, the process at the head of the queue is run next. Uses a FIFO queue where new ready processes are appended to the tail of the queue.

FCFS analysis

- Simple and easy to implement
- Minimal context switches
- Convoy effect, IO bounded process will have to wait for a long running CPU bounded process, making them to spend a long time waiting, reducing IO resource utilization.
- Longer turnaround time due to convoy effect.
- Starvation is possible if a ready process is entirely CPU bound forever.

Shortest job first (SJF), runs the job that has the least amount of work to do until its next IO or completion.

SJF analysis

- Simple and easy to implement. Minimal context switches
- No convoy effect, IO bounded process will have lower turnaround time. Optimal average turnaround time for a given set of jobs (CPU burst) that are available simultaneously
- Can't estimate the cpu burst time
- Starvation of CPU bounded processes, if IO bounded process are frequent

4.2 Preemptive scheduling

Round robin (RR)

- runs a process for a quantum
- Switch to the next job in the ready queue and place the current one in the back of the queue
- Repeat til all processes are finished
- If a process blocks, switch to the next job, and place the process in the blocked stack. When the process unblocks, puts it in the back of the queue

RR quantum

- If quantum is too long: RR is just FCFS, less context switching overhead, less responsive just like FCFS

- If quantum too short: lots of context switching overhead, more responsive
- Quantum length is a trade off between context switches and responsiveness

RR analysis

- Simple, easy to implement
- with a reasonable quantum, high responsive time
- Bad turnaround time, since each process completion is delayed
- Not fair, tends to favor CPU bounded processes over IO bounded processes. Since IO bounded processes uses only a tiny section of a quantum and has to wait at the back of the queue
- No starvation

Priority Scheduling

- Assigning a priority to each job, allocate the CPU to run the process with the highest priority that is ready
- Priority assigned either statically at the start of process, or dynamically adjusted throughout the execution
- To avoid starvation, decrease the priority of running process overtime after clock ticks (and preemptively switch the process if their priority drops below the next highest process)

Multi-level feedback queue

- Multiple queues with decreasing priority going down. Lower priority has higher quantums
- Processes start at the highest priority queue. If they've used their quantums (cumulatively), move them to the lower priority queue (and reset the cumulative quantum count). Otherwise, stay in the same queue.
- We run a scheduling algorithm (RR) in the highest priority non-empty queue. The scheduling algorithm is a parameter to the algorithm
- CPU bound processes will sink to the bottom overtime, while IO bound processes will float since they use their quantum less. This leads to good resource allocation and fewer context switches for the CPU bound process since IO bound processes will execute first with shorter quanta.
- New processes are executed before older processes. This increases response time
- Algorithm assumes nothing from the processes at the start, all priority is derived empirically

5 Memory Management

We need memory management for

- Support multiprocessing, as multiple processes must be loaded into memory to ensure a reasonable supply of ready processes to consume cpu time
- Security, isolation between processes and the OS memory

- Allows computers to run processes with memory requirements larger than what is physically available , either a single process requirement exceeds available, or the sum of all processes exceeds available

No memory abstraction

- Programs directly reference physical memory
- Memory instructions operate on physical memory addresses
- Multiprocessing can be achieved by having the running process in memory and all other processes on disk. On context switch, move the running process memory to disk and swap the newly running memory into ram.

Byte-addressable memory means that we can address each byte of memory individually. If the addressable unit is n-bytes, we can only address every n bytes.

Memory abstraction

- Provides logical address spaces each bound to their separate physical address space. The program instructions refer to logical addresses, while the OS manage physical memory. The hardware translates logical addresses to physical addresses
- Two approaches, dynamic relocation with base and limit registers, paged virtual memory

5.1 Dynamic relocation

Dynamic relocation is

- Program is loaded into consecutive physical memory when there is room
- Two registers in the CPU are used to denote the physical memory location of the process. Base register is the physical address where the program begins, and limit register stores the length of the program
- The values of the registers are used by the OS and stored in the PCB across context switches. They are loaded into the CPU by the OS from PCBs when context switching

To translate logical address to physical address

- First check that the logical address is below the limit register (otherwise, the logical address is not in the bounds of the process and error interrupt)
- Otherwise, physical address = logical address + base
- The CPU's MMU performs the translation on the HW level (the cpu will send the logical address for a memory access/write to the MMU, and the MMU will perform the operation with the base/limit registers on physical addresses)

Details

- Requires contiguous memory allocation
- OS need to keep track of memory usage to: allocating free memory to process, and deallocate memory when process is terminated/blocked or swapped to disk

Keep track of memory, use a linked list with node containing: hole or process, starting address, length of node, pointer to next node.

Consider allocating a free block of memory to a process. When the block is chosen, we will divide the hole into two pieces: one for the process and one for the unused memory (unless an exact fit). Some hole selection algorithms are

- First fit, allocate process to the first free block that fits the process. Fast and fewer searches
- Best fit, allocate to the smallest block that fits the process. Slower than first fit, creates small, useless free blocks since it always produces the smallest leftover hole
- Worst fit, allocate to the largest available free block. Produces the largest leftover memory hole. Slower than first fit

Consider deallocating contiguous memory. This is required when a process terminates or is swapped out of memory

- If a process is not running or blocked, it can be swapped out of memory to disk and then brought back into memory when it resumes
- Makes it possible for the total physical address space of all processes to exceed the real physical memory
- When deallocating, we update the linked list and merge nearby holes into the same node

External fragmentation

- Overtime, free memory might be broken into very small pieces such that the total available memory can satisfy another process, but they are not contiguous
- Solution is to deallocate and swap out all the processes and reallocate them (or defrag and move all processes to the start of the memory)

Limitation of dynamic relocation

- Entire process must be loaded into contiguous memory
- Process cannot be larger than physical memory
- When more space is needed, an entire process needs to be swapped to disk
- External fragmentation
- If the memory usage of a process increases over execution, the OS may need to move its physical address

5.2 Paged Virtual Memory

Method is

- Break up logical address space of a process into fixed sized pieces call Pages
- Break up physical memory space into fixed sized slots called Page Frames (Frames)
- Pages are mapped to frames: they don't have to be the same size, but generally they do
- Pages don't need to be stored in contiguous frames
- Not all pages have to be in physical memory at the same time to run a process. During the lifetime of a process, pages start out on disk and may move between RAM and disk multiple times. When a page moves into RAM and thus into a frame, it may be put into another frame than before

Page table

- Maintained in OS memory in PCB, one page table per process, with entries that maps logical pages indices to physical frames indices (or null if the physical frame is not in memory but on disk)
- The page table base register (PTBR) points to the start of the page table and is stored in the PCB

Using the page table conceptually

- Find the page of the logical address
- Divide the logical address into a page number and an offset (using mod page size)
- Translate the page number to the frame number using the page table, and get the address of the frame by multiplying the number by the frame size.
- The physical address is the frame address plus the offset

Structure of the logical address (and physical) in CPU

- The logical address of size m (for 2^m logical addresses) has the lower n bits for the page offset (for 2^n offsets), and $m - n$ bits for the page number (for 2^{m-n} pages).

MMU translation in practice

- Cpu computes the logical address
- MMU gets the first $m - n$ bits for the page number
- MMU translates the page number to the frame number
- Creates a physical address by copying the n lower bits of the offset and the higher bits as the frame number.
- The physical address size can be different to the page address size

Internal fragmentation

- When processes are divided into pages, the last page may have some leftover space (since the logical space is essentially never a multiple of the page size).
- Tradeoff between page table size and internal fragmentation. Larger page size, more internal fragmentation, but larger page table.

Page table entry contains

- The page number by the entry index
- The frame number
- Present/absent bit that indicates whether the page is in physical memory (1). Otherwise, a page fault is triggered
- Referenced bit, set to (1) if the page has been accessed/referenced at least once, set by MMU automatically
- Modified bit, set to (1) if the page has been written to, set by MMU automatically

Page fault

- If a process references a page not mapped to a frame, MMU raises a page fault using an internal interrupt
- OS will handle a page fault by: if no free frames, pick a page to evict and writes its content to disk only if the modified bit is set; fetches the page referenced from disk and load it into the frame; updates the page table; restarts the process at the instruction that caused the page fault.

Page replacement algorithm is to pick a page frame to evict (since each page is mapped to a frame). Optimal implies the fewest number of page faults, so we evict the page that will be accessed furthest in the FUTURE. (Optimal not practical, used as a comparison point)

First in First out will evict the page that has been in memory the longest.

- Maintain a FIFO queue of pages based on load time, page loaded the longest time ago at head of queue, most recent page at tail of queue
- Evict the head at a page fault
- Simple to implement, but may evict heavily used pages

Principle of locality

- Temporal locality, if a process accesses a particular memory address, it will likely reference the same address in the near future
- Spatial locality, if a process accesses a particular memory address, it will likely access nearby addresses in the near future
- We can design page replacement algorithms to optimal against this principle

Second chance algorithm

- Simple modification to FIFO
- Inspects the referenced bit of the oldest page. If 0, evict. If 1, the page is old but recently used, so we clear the referenced bit and move the page to the tail of the queue (second chance) and update the load time as if it is recently loaded, repeat until a page is found with 0

LRU

- Pages that has been accessed in the near past are likely to be accessed again in the near future
- Evict the least recently used page
- Maintain a list of all pages with the most recently used at the head. Choose the page at the tail to evict. This is hard to implement since we need to update the list on every page access

LRU Aging implementation

- Aging will maintain a bit counter for each page
- At each clock interrupt, shift all bits to the right (lower)
- Between clock interrupts, set the leading bit to one if the page has been accessed
- Pick the page with the lowest age to evict.

- Aging could evict a page that was not the least recently used, since overtime the age counter will all be zero so any usage difference longer than that will be ignored.
- Aging cannot differentiate between the times of references within a single clock tick

Translation lookaside buffer

- Without optimization, each memory reference requires two memory accesses (one in the page table, one in the data)
- TLB is a fixed capacity hardware cache in the MMU
- Implemented using associative circuity (all entries are looked up in parallel)
- Each TLB entry contains: tlb valid, page number, frame number, other bits
- the valid bit indicates whether the cached entry is valid or not

Using the TLB

- For a logical address, search for an entry for that page in the TLB. If hit, get the frame number directly using the TLB
- If miss from TLB (either doesn't exist of TLB invalid), use the page table to lookup the frame number. Update the TLB afterward and set the TLB valid to one.
- If TLB miss and it is full, an entry from the TLB is evicted and replaced with an entry for the page that was just looked up.
- If the page of the running process is swapped out, the corresponding entry in the TLB becomes invalid and it will be invalidated
- Context switch requires clearing the TLB (since it contains mapping only valid for the currently running process as it uses the mapping from the process's page table), this often leads to slower initial memory access after a context switch due to TLB misses. This is called a tlb flush.
- Larger page size implies more memory locations that can be indexed with a fixed number of pages, hence more tlb hits and better performance

6 Cryptography

Alice and Bob wishes to communicate securely over an insecure computer network. Alice and Bob can be: routers, browser and server, ssh client and ssh server, git and github.

Secure communication has three properties

- Confidentiality, a third party cannot read the message, only the sender and intended receiver should understand the communication contents
- Integrity, detects if the content of the communication has been tampered with, ensures that no one modifies the message
- Authentication, establish the identities of one or both of the endpoints to confirm that they are who they claim to be.

Cryptographic principles

- techniques like: encryption, hashing, MAC, digital signatures
- based on mathematics, problems that are considered to be computationally hard, factoring two large primes (RSA), solving discrete log (ElGamal)
- Cryptography is not absolute, there are no perfect security. It is always susceptible to brute force, the idea is to make a bruteforce attack take so long that it is infeasible to perform it during the useful lifetime of the data

6.1 Encryption

Encryption

- Provides confidentiality
- Take a plaintext message, encrypt it into a ciphertext in a way that only the authorized party can decrypt it back to plain text
- Two function, `c = encrypt(m, K_e)` and `m = decrypt(c, K_d)` where K_e is the encryption key and K_d is the decryption key.
- Content of c are hidden from anyone without the decryption key
- Kerckhoff's principle, the security of an encryption scheme only assumes that the decryption key is private, and does NOT assume that the algorithm is private

Two types of algorithms

- Symmetric encryption, same key for encryption and decryption, both keys are secret
- Asymmetric encryption (public key encryption), two different keys, public key to encrypt and private key to decrypt, only the private key must be kept secret

Symmetric encryption

- `c = encrypt(m, K)`, `m = decrypt(c, K)`
- Relies on the secret key K being securely exchanged between the two communicating parties
- Modern example is AES
- Main question is, how to securely exchange K

Symmetric encryption protocol: Alice wants to send a message to Bob

- Alice and Bob securely exchanges secret key K
- Alice computes ciphertext c by running encrypt
- Alice sends c to Bob
- Bob receives m by running decrypt using the same key
- Provides confidentiality provided that K is secret

AES

- Block cipher, breaks data into fixed-sized blocks, and separately encrypts and decrypts each block

- Two mode of operations: electronic codebook, cipher block chaining. A mode of operation determines how each block is treated and linked with other blocks.

Electronic codebook

- Each block is encrypted and decrypted independently using K
- Parallelizable
- Should never be used due to: deterministic encryption (same text always encodes to the same ciphertext), leaks information about the plaintext (2 identical plaintext blocks results in identical ciphertext blocks, thus two identical text always encodes to the same ciphertext)

Cipher block chaining

- Uses probabilistic encryption, through inserting randomness during encryption, so that the same plaintext encrypts to different ciphertext
- Uses an initialization vector that must be random, this can be publicly shared
- Encryption: first block XOR the plaintext with IV, encrypt, get ciphertext; other blocks XOR the previous ciphertext with plaintext, encrypt, get ciphertext
- Decryption: first block decrypt ciphertext, xor against iv, get plaintext; other block decrypts ciphertext, xor against previous ciphertext, get plaintext
- XOR is used to combine two messages with a high entropy while being reversible
- Encryption must be done sequentially, decryption can be done in parallel

Asymmetric encryption

- Public key encryption
- Each principal has a pair of public private key pair
- Encryption is done using the public key (which is visible to everyone), `c = encrypt(m, K_public)`
- Decryption is done using the private key (which is secret), `m = decrypt(m, K_private)`
- Example are RSA, ElGamal

Asymmetric encryption protocol, ALice send message to Bob

- Bob generates a key pair
- Bob posts the public key
- Alice uses Bob's public key and computes ciphertext c using encrypt
- Alice sends c to Bob
- Bob recovers m by decrypting using his private key
- No need to securely exchange keys
- Provides confidentiality if: private key is secret, public key is actually Bob's public key
- Problem that Asymmetric encryption is slow on long messages due to it computationally expensive

Hybrid encryption

- Symmetric is efficient for long messages but requires sharing a key. Asymmetric is slow on long messages but no need for key exchange
- Use asymmetric to securely exchange a shared secret key. Use symmetric to encrypt communication using the shared secret key

Hybrid (RSA key exchange) protocol using Asymmetric encryption

- Alice generates key pair
- Alice posts public key
- Bob generates a secret key and encrypts it using public key
- Bob sends ciphertext to Alice
- Alice recovers secret key using her private key
- Alice and Bob now shares a secret key, can now exchange messages using symmetric encryption like AES

6.2 Integrity

Message integrity

- Verifies that the message was not tampered with
- Reason is that attackers may modify messages in transit through MITM attacks
- Can use MAC and digital signatures

Cryptographic hash function

- A plain hash function that takes an arbitrary length input and produce a fixed length hash (digest)
- Two properties: collision resistance implies that it is hard to find two inputs that hash to the same digest; one way, hard to inverse the hash function
- Example functions like SHA2 (sha256, sha512), and SHA3, bcrypt

Message Authentication Code (MAC)

- Provides integrity to message, but not confidentiality
- Symmetric cryptography, requires a shared secret key
- Requires a MAC function, generally a cryptographic hash function that takes the secret key and a message and hash the concat (can also use some other concat methods to mix the message and secret key)

MAC process

- Alice and Bob have a shared secret key
- Alice generates message m
- Alice generate a tag t given the message m and a secret key K using the MAC function
- Alice sends both the message and the tag to Bob

- Bob generates the tag using the same secret key and message, and verifies that the tags are equal. If the tags are different, the message was tampered
- A third party cannot forge the tag of a new message only by knowing the message (since they also need the secret key)

Digital signatures

- Asymmetric cryptography, needs a private signing key and a public verification key
- Sign algorithm, use the private signing key to produce a signature on a message
- Verify algorithm, use the public verification key to verify the signature given the signature and the original message. If verified, the message is not tampered with and must be signed by the other party
- Provides integrity and non-repudiation (signer cannot deny signing the document, since only they have the private key). MAC is repudiative since the other party can also sign the message.

Digital signature protocol

- Bob want to verify Alice's message
- Alice generates a signature for a message
- Alice sends message and signature to Bob
- Bob accepts the message only if verification succeeds
- Goal is integrity not confidentiality

Since asymmetric cryptography is slow, for large messages, we sign on the hashed message instead. This hash function must be cryptographic otherwise trudy can use another message with the same hash and thus tamper the message.

6.3 Authentication

Authenticated encryption provides both confidentiality and integrity.

Encrypt then MAC protocol

- Alice uses encryption secret key to encrypt message
- Alice generates tag on the ciphertext using MAC secret key
- Alice sends ciphertext and tag
- Bob use ciphertext and MAC secret key to verify the tag
- Bob then decrypts the ciphertext into plaintext
- Examples: AES-GCM, AES-OCB, etc

MAC then encrypt is considered to be not as secure as encrypt then MAC.

Encrypt then MAC problems

- Symmetric, so still need to securely exchange two shared secret keys (encryption and MAC keys)

- Use an asymmetric encryption key exchange protocol (RSA key exchange) first to exchange secret keys

Different ordering

- Encrypt and MAC may leak the plaintext because MAC does not guarantee to not leak message
- MAC then encrypt issues
- If digital signature, should sign then encrypt since signing doesn't have the same problems as MAC. If encrypt then sign, the signer does not need to know the plaintext message, and thus we cannot establish that the message came from the signer (since a third party can strip the signature, and sign it themselves)

6.4 Digital Certificates

In asymmetric encryption key exchanges, the other party need to confirm that the public key is indeed from the expected person. Otherwise, a third party can insert their own public key and act as a middleman proxy reading the messages.

Digital Certificates

- Securely associate identities with public keys
- A digital signature on a binding of (identity, public key of the identity).
- Certificates are issued by certificate issuers (trusted third party). The issuer has their own public private key. A binding is an identity public-key pair,a certificate is a tuple of the binding and digital signature of the binding
- The issuer is using a digital signature to certify the binding relationship.

Authentication with certificate

- To verify Bob's identity and public key
- Bob creates a fixed message
- Bob signs the message with digital signature
- Bob sends the message, signature, and cert
- Alice verifies the cert to ensure that it came from the issuer, checks the binding in cert to find Bob's public key
- Verify the signature using Bob's public key

But this has the same issue: how can we verify the signer's public key?

Certificate authorities

- Entities that are explicitly trusted (trusted as in trusting their public keys)
- Sign certificates for others, certificates signed by them are also trusted
- Their public keys are contained in root certificates which are self-signed with their own public keys

- Root certificates are shipped with the OS and browser, so we have a list of certificate authority public-key pairs.

Can use a chain of certs all the way to the root. To verify a chain, use the root public key to verify the second cert and thus their public key, use the second cert's public key to verify the third, etc.

6.5 TLS

Not examinable.

Diffie Helman is a key exchange algorithm that doesn't require integrity to exchange a common key. Provides forward secrecy: reducing impact on existing sessions if the private keys are leaked after it is started.

7 Internet Introduction

Internet history

- The internet is the aggregation of many smaller networks, not a single network under a single point of control. We often connect to one of the local subnets when using the internet, and not the entire internet.
- 3 development phases: ARPANET, NSFNET, Internet

Complexity of the internet

- Million of nodes with no direct physical pairwise connections
- Need to tell data where to go, specify the physical signals sent, and share physical links between pairs
- Requires a modular way of handling these different tasks to increase cohesion (one layer for each task)

Packet concepts

- Named headers are structured fields with context
- Payload contains pure data
- Often made of a protocol stack, with each protocol adding their own metadata/headers to the payload. This forms a stack of protocols/headers: invitation, letter, post; or invitation, letter, email. Often we can change protocols without changing the payload content, sometimes we can't.

Network model

- The network is modeled as a stack of layers. This is just a conceptual model. The lowest layer is the most physical.
- Each layer offers services to the layer above it (the higher layer can use the service of the lower service). This is done through an interface
- Inter-layer exchanges are done according to a protocol on that layer

Services and protocols

- Service is a set of primitive operations that a layer provides to the layer above it.
- Protocol is a set of rules that govern the format and meaning of packets exchanged between peers in the same layer

Service connection

- Connection Oriented (TCP), requires a connection before use, need a negotiation handshake to setup the connection
- Connectionless (UDP), each message is self contained
- The choice of connections in the service affects the reliability, quality and cost of the service.

TCP/IP and OSI

- Both are standard set of layers
- TCP/IP model reflects what happens on the internet
- OSI model reflects the thought process that is followed when designing an ideal network or fixing a fault. Idealized, with a degree of flexibility in practice where protocols can cross layers.

OSI details

- A layer for a different abstraction, each layer performs a well defined function (the function should also be standardizable)
- Layer boundaries should be selected to minimize information flow between layers
- Number of layers large enough so distinct functions are in distinct layers, but small enough so it is not too complex

OSI model layers, with each one having their own protocols

- Application, data here
- Presentation
- Session
- Transport, tidy up end to end
- Network, move data from end to end
- Datalink, tidy up point to point
- Physical, move data point to point

End to end implies finding a path from one subnet to another subnet through the internet. Point to point implies sending data through a physical link.

TCP/IP

- Only includes the: application, transport, internet, link
- Still numbered using the OSI model

Using protocols

- Give data to the layer below it, which adds a header and metadata plus the data payload. Repeat for all layers below it creating a protocol stack
- When sending point to point, payload is maximally wrapped to the physical layer, sent across the physical layer, unwrapped to the layer desired for computation (link for link switch, network for routers, application for destination)

TCP/IP protocol stack

- Application: HTTP, FTP, SMTP, DNS
- Transport: TCP, DUP
- Network: IP
- Physical/Link: LAN

IP Narrow waist

- Narrow waist because: many application protocols above it, one IP protocol, and many link layer protocols below it
- Benefit of IP over everything: new physical networks supporting IP supports everything, application using IP runs over everything
- Not always the case, SMS and calls first ran over a non-IP network. A network can theoretically support multiple network layer protocols

Network architecture

- The design of the network: protocols and layers used to send messages.
- The narrow waist is part of the architecture of TCP/IP
- It considers the entire network. Is hard to change. But shouldn't be considered too hard to be perfect
- TCP/IP prevailed since they were experimenting when the internet was small and flexible. OSI was stuck on trying too hard.
- Network topology refers to the physical connections between nodes in a network

HTTP

- Many networks block everything except HTTP through firewalls
- New application protocols send data as HTTP instead
- Maybe a new narrow waist

8 Socket Programming

Socket concepts

- A method that user-space code use to send messages to kernel-space networking code
- The user networking program will create a socket and use it to send a message. Every process will send and receive network data through a socket.
- In a client server scenario, there will be two sockets, one on the client, one on the server

- An abstraction of the transport layer provided by the OS

Socket details

- A five tuple for the socket: protocol (transport layer protocol), local ip, local port, remote ip, remote port
- The port number is used to differentiate between different sockets on the same endpoint
- Often using source/dest instead of local/remote. Source/dest can be confusing since a socket is used for both sending and receiving, but the dest is always the server
- The five tuple is only for a full duplex socket

Berkeley sockets

- A popular socket interface that all major OSes use. A portable interface
- In unix, everything is a file, exported as an integer file descriptor. A socket is therefore also a file. The APIs exported by sockets are implemented through general file systemcalls (like connect, read, write, close) on the socket file descriptor.
- Note that some file systemcalls (like fseek) doesn't make sense on socket file descriptors

Client vs server

- The client is defined as the side that initiates the request. The server is defined as the side that takes the request.
- The client server model only exists on the application layer
- The server must: ensure connected to the internet, online, ready to accept a request
- The client must: ensure connected to the internet, send a request with a dest address to connect to

The steps to use a socket under this model is

- Server creates a socket: socket
- Server chooses a location to bind the socket to: bind
- Server allows incoming connections: listen
- Server waits for an incoming connections: accept
- The client creates a socket: socket
- The client makes the connection: connect
- Connection is established, server accept unblocked
- The client sends a message: write
- The server reads a message: read
- Either side can close the connection: close

Socket terminologies

- Socket, creates socket
- Bind, assiciate socket with local port

- Listen, annouce willingness to accept connections, give queue size
- Accept, passively establish an incoming connection
- Connect, actively establish a connection
- Send, Receive, close

Socket state machine

- IDLE
- Via passive accept request segment or active connect execution, transition to establishment pending
- If connection successful, transition to Established
- If passive disconnect segment received, or actively disconnect, transition to disconnect pending.
- If disconnection successful, transition to idle.

Server sockets

- The server code has two sockets: listen and conn
- The listening socket is a half socket that doesn't have a remote ip and remote port. Its goal is to listen for incoming requests and create a connection full socket for each request
- The connection socket is a socket that we can use to read and write, identified by a full 5 tuple

Blocking and non-blocking reads

- Because network data arrives in stages, when reading from the socket we must always check how much was read (which can even be less content than send via a write call from the other side)
- Blocking sockets waits until there is some data, so each read syscall will block until some data. The returned length must be at least 1
- Non-blocking sockets only reads the data that has already arrived, meaning that it can return 0 bytes even if there are more data coming
- Multithreading makes blocking sockets more appealing since they are easier to write and we can simply create more threads to do something when it is blocked

TLS

- Sockets carry plain text
- TLS is a layer on top of sockets/TCP that encrypts data. It provides the same service as TCP, but encrypted
- Requires an extra handshake at the begining to verify and setup cryptography
- It is easy to use TLS just for encryption, since many websites requires TLS. It is harder to verify the identity of the server, which is needed when you are providing sensitive information to the server.

9 DNS

Domain name server is a protocol that maps domain names to ip addresses

- Domain names are the naems we type
- Sockets requires ip addresses in numbers
- The DNS service maps human readable names to fixed length computer friendly ip addresses
- DNS is on the application layer, requires creating a socket to find the IP address with a name server (this requires the hardcoded IP address of the name servers)

Components of DNS

- Domain name space, an abstract tree structured name space that underpins a valid DNS name
- DNS database, each node in the name space tree contains a set of dns information in resource records (RR). The collection of them forms a distributed database
- Name servers are programs that hold information of a subset of the domain namespace tree and their RRs
- Resolvers, are client programs that extract dns information from name servers in response to DNS lookups

Domain name characteristic

- Not case sensitive
- Constituents separated by dots
- Each constituents have up to 63 characters
- Can have a total max 255 characters
- Can be internationalized into unicode characters

Domain name space tree

- Root of the tree are generic top level domains (.com or .net), and country top level domains (.au or .us)
- Each node under a TLD (or another node) represents another constituent under its parent constituents
- A dns path is a path down the tree, combining the constituents with dots

Top level domains

- The roots of nodes in the domain namespace tree. Can be general TLD and country TLD

Resource records

- Records an domain name entry, mapping from dns name to ip
- Notable types are: A, ipv4 address of a host; AAAA, ipv6 address of a host; MX, mail exchange record denoting the domain handing mails also with priority (number space constituent); NS, the name of the server handling this domain; CNAME, canonical domain name, the real domain name

- Each node in the domain namespace tree will contain a set of RR of its child domain names

RR table details

- A set of lines, each line is a RR associated with a domain name. A domain name can be represented absolutely by ending with a dot, or relatively against this nameserver's absolute domain name by not ending with a dot
- There are no inherit ordering of RR entries in the table, except that the absolute domain name entries should come before the relative domain name entries
- Duplicated entries in the RR entries means that the name server will return multiple results for a DNS lookup. For example, multiple A records for the same domain name means that the name server will return both ip addresses. The resolver will then try both ip address in order
- The name server can be setup to select duplicated RR randomly to do act as a poor load balancer, or it can return all for: backup, compatiability with both ipv4 and ipv6

Name server zones

- Name server zones are nodes (or node) in the domain namespace tree. Each zone will contain one or more nameservers that are authoritative for that zone. Zones do not overlap with each other on the nodes
- Name servers are hierarhical, in that the name servers at the root knows their child name servers
- The root name servers are the authoritative name servers for TLD names. These name servers are contacted a lot by local name servers if they don't have a cached result and will contain thousands of name servers.

Types of name servers

- TLD dns servers, for TLD domains and country TLD. For example, .com, .edu
- Other authoritative DNS servers, from organizations providing DNS servers against hostnames owned by the organization's servers. These can be provided by the organizations themselves for a thirdparty
- Local DNS server, from local ISPs, that handle DNS requests/queries via cache, or act as a proxy to query the DNS from the namespace hierarchy

Resolver

- A client that asks the local DNS for the domain to ip mapping. Resolvers are often clients on the machine (OS)
- If answer is known by local DNS, returns the answer
- Otherwise, the local DNS server will query against a root name server. Each name server will either return the ip of the domain name, or return the ip of the next subtree name server that the local DNS server will contact next.
- So the local DNS server will repeat this hierarhical quering against authoritative name servers until it gets a final ip address, which is returned back to the resolver

10 Email

Architecture

- User agent, a client that allows user to send and read emails
- Message transfer agents, software that handles the transportation of messages from source to dest
- The sender user agent uses SMTP to send the message to a receiver. MTA along the way passes the SMTP message to the receiver user agent. If the receiver has a mail box, the SMTP stops at the mailbox and the receiver user agents uses a delivery protocol to read from the mailbox

User agents

- Mail program that handles: compose, report, display, disposing emails
- Encapuslates message into SMTP, containing headers (which have user agent info), body (for message text)
- User must provide the message, destination, and other parameters
- Addressing scheme is user@address

SMTP Message format

- Header lines like: from and to. Headers are more free form text, while fields are data identified by their byte locations
- Blank line between header and body
- ASCII message body

SMTP

- Simple message transfer protocol
- An application protocol that uses TCP to transfer email messages from clients to servers, port 25
- Three phases, handshaking, transfering, closure
- A command and response interaction, where commands are in ASCII text, and response consists of a status code and and phrase

MIME, multipurpose internet mail extension

- Metadata that dictates message types other than text
- MIME headers: MIME-version, content-description (human readable description of content type), content-id (unique identifier), content-type (type of content)
- Content types are: text, image, audio, video, application, etc. Subtypes can be: text/plain, image/jpeg, application/json. The content-type header contains the content type and the subtype combined with a backslash

Message transfer and delivery

- SMTP is used to transfer mails between servers

- Delivery is how the receiver receives the mail, its protocol can be: local, POP3, IMAP, HTTP
- POP3 is post office protocol that allows downloading messages from a mailbox server. IMAP is internet mail access protocol that is more complex with more featuers, allows direct manipulation of stored messages on the server
- Local delivery implies that the receiver is permanently connected to the internet, and their user agent is on the same machine as the message transfer agent. This is rare
- Remote delivery means that the mailbox is on a separate server to the receiver user agent, thus the user agent need not to always be powered on and connected to the internet (just its mailbox MTA). The receiver user agent can periodically connect to its mailbox to download mails. Most phone/laptop is not a MTA and uses remote delivery

IMAP details

- Allows user agent to query an MTA
- Has concepts of users and sessions, and can keep user states across sessions
- Allow manipulation of online and offline messages and mailbox folders
- Authorized, uses the same command and respones ASCII interface as SMTP

11 HTTP

WWW terms

- Client, a browser that cna access web pages
- Server, daemon based content delivery of pages
- URL, a combination of the protocol, DNS name, and filename/path

The Architecture of WWW

- The web browser sends a http request to the server
- The web server responds with a http response containing html structured data
- The web browser parses the html and sends a http request for images/ads content in the html
- The web server responds with a http response containing these images/ads/content

HTTP context

- Hypertext transfer protocol, the protocol for the web
- Resources are located by the uniform resource location (URL)
- URLs are defined in the HTTP spec, and is an address for a response. Can be absolute or relative to a domain name
- URI (uniform resource identifiers) are very similar to URLs, but also include non-web identifiers like ISBN
- The format of URL is `protocol://user:password@host:port/path?query#fragment`

HTTP protocol overview

- Client initiates TCP connection to server on port 80
- Server accepts TCP connection from client
- HTTP messages are exchanged between the client and server
- TCP connection closed

HTTP connections

- HTTP 1.0 has a single-use connection that transfers only one specific information
- HTTP 1.1 has persistent connections, allowing multiple information transfers per connection, requires a header field

A non-persistent connection will: client server connect, client send HTTP request for html, server sends html, client receives html and finds that it needs some 10 images, server close, client separately create 10 connections to fetch the 10 images. A persistent connection does not have the server to close the first connection reducing latency from the first two roundtrip. Summary

- Non-persistent requires 2 response times per object (initiate TCP and first HTTP request) plus file transmission time. Requires OS overhead for each object via TCP connection. Browsers can parallelize TCP connections to fetch related objects
- Persistent connections has the server leave the connection open after sending response. Subsequent HTTP messages can be instantly sent by client over the first open connection, when it parses the server response
- Four possible configurations for multiple connections: sequential requests non-persistent; pipelined requests non-persistent; sequential requests persistent; pipelined requests persistent (fastest)
- Pipelining requests means to send all the resource requests before receiving their response. Sequential requests means to wait for the response before sending the next request

What happens when a link is selected from the browser

- Browser gets the URL
- Browser uses DNS to find server IP address
- Browser makes a TCP connection
- Browser sends HTTP request
- Server sends HTTP response containing web page
- Browser fetches other URLs from the same connection, and potentially opening more TCP connection with other servers containing resources
- Browser displays the page and updates it as content arrives
- Browser closes the TCP connections

HTTP request methods

- GET, POST

- Idempotent means that duplicated requests will have the same effect as a single request (DELETE, PUT, since deleting and updating multiple times is equivalent to doing it once). Does not mean PURE
- Safe means that the request is only for retrieving information, and shouldn't update the server state (GET is safe but POST is not)

HTTP request format

- First line is the request line: request method, relative URL, http version
- Next few lines are header lines, each line represents a header by: header name, colon, header value
- Two CRLF blank lines to indicate end of headers and message

HTTP response code (a number in ASCII)

- 1xx for information
- 2xx for request succeeded
- 3xx for redirection
- 4xx for client error
- 5xx for server error

HTTP response format

- First line is the status line: http version, response code, response code name
- Next few lines are header lines
- A single CRLF blank line to end header
- Data
- CRLF to end message

Compared to SMTP, it is not chatty since it consists mostly of a single request/response, instead of a back and forth command response pattern.

HTTPS

- A separate protocol for web servers
- Basically running HTTP over TLS instead of over TCP
- Uses port 443
- Everything is encrypted (including the URLs) namely the payloads

HTTP 2

- Compatible mostly with HTTP 1.1
- Aim is to decrease latency by: compressing HTTP headers, server push by sending both the html and the html images at the same time, multiplexing requests over a single TCP connection so that bytes for one image doesnt have to wait until the bytes for another to finish transmitting (loading multiple resources at the same time)

- Negotiation is done by both the client and server over the HTTP version
- Supported by a third of websites

HTTP 3

- Aim to decrease latency: runs over QUIC for more parallelism
- Application layer protocol essentially unchanged
- Supported by about a third of the website

12 Transport layer

The purpose of the transport layer is to tidy up the e2e (client to server) services provided by the ip layer. Specifically, it provide services needed by applications using the services provided by the network layer.

The application needs

- Data from one application is not mixed with that of another
- Data doesn't arrive faster than it is processed
- Some data are just a long stream of bytes in order
- Data arrives reliably (or we know that a packet is dropped)
- Data arrives in order

The network provides one and only one service

- Getting packets from one endpoint address to another endpoint address
- Based on a best effort principle: works most of the time, and sometimes even with multiple copies

The transport layer entities provides the implementation (hardward or software) for these protocols. They can exist in multiple locations on the host (in kernel, process, dedicated card).

The transport layer provides a logical communication channel between processes running on different hosts. They can be connection oriented or connectionless

- Connection-oriented: establish connection, data transfer, connection close (TCP)
- Connectionless: transfer data (UDP)

The transport layer also provides multiplexing (allowing multiple data transfers per link/network at the same time), and can provide a service on top of an unreliable network.

Specifically, transport layer provides

- Packets are groups of data transferred at once by the transport layer. Also: segments, packets, frames, messages
- Unreliable (connectionless) service provides no guarantees. This often provides multiplexing between different processes (allowing multiple connections at the same time)

- Reliable (connection oriented) service provides a perfect connection, barring privacy, isochrony (equal delays). It encapsulates acknowledgements, congestion control, lost packet retransmission, etc. This is provided to the higher layers
- Reliable transport service guarantees: no packet loss (no duplicated packets, only need to send and receive each packet once), and that packet integrity is preserved (checksum). Note in order packets is not required for reliable service. TCP has both, UDP only has integrity

The payloads encapsulated in segments are encapsulated in packets that are encapsulated in frames. Each layer will only work with an abstract representation of messages from the above layer.

Multiplexing and demultiplexing (MUXING and DEMUXING)

- Multiplexing is combining multiple distinct streams (from multiple applications) into a single shared streams
- Demultiplexing splits distinct streams out from a single shared stream
- Needed since each endpoint only has a single network connection out and in

Multiplexing addressing

- Requires specifying the process to connect to on both the application and the transport layer
- Use 16 bit port numbers to determine the application. A unix service xinetd will intercept all incoming request packets and spawns a service to handle the request per specific port.
- The 5-tuple socket contains the ports. Half sockets like tcp listen sockets and udp sockets (listen sockets) only has the local port

Port allocations

- Allocated by IANA
- Classified into three parts: well known ports (0-1023), registered ports or user ports (1024-49151), dynamic ports used for random ports (49152-65535)

Thus, MUXING will combine transport layer segments to multiple remote ports into a single stream. DEMUXING will split that single stream into segments to multiple ports.

13 User Datagram Protocol

UDP allows applications to transmit data over IP without connections

- Transmits in segments consisting of a header followed by the payload
- UDP header contains of both the source and dest ports. The UDP socket is a 3-tuple, but each packet is a 5-tuple
- Payload is handed to the process attached to the dest port on the destination endpoint

Note that UDP packets are seperated into datagrams, so as long as all your data fits into a single packet, you do not need to check for deliminators. This also means that each send() call corresponds to a recv() call, and that the bytes will not be split up into two different packets. THIS IS DIFFERENT TO TCP WHICH ARE BYTE STREAM BASED WITH NO DELIMITATORS.

UDP conversations

- Connectionless, each message is independent
- Can still carry conversations if on the application level: client sends first message from dynamic port to well known port, server replies from well known port to same dynamic port, client replies with the same port. The server can then guess that the dynamic port is handling the conversation. Each payload needs to attach a number indicating the ordering of the messages.

UDP header, 32 bits

- Source port
- Dest port
- UDP length
- UDP checksum (the checksum is over both the udp header and the ipv4 header)

UDP properties

- Easy to code, fewer roundtrips.
- Suitable for simple client requests: client sends a short request, waits for a response, if that does not occur client timeouts and resends. DNS for example
- Also suitable for real time services, since we dont care about lost packets. Can conceal packet lost using the "best guess" information.

UDP pros and cons

- Allows muxing and demuxing. No delay in waiting to recover lost packets
- No flow/congestion control, error control, or retransmission of lost/bad segments
- Use where applications requires a precise level of control over packet flow/error/timings, since they can implement the missing features themselves.

14 TCP

Network layer properties

- Gets packet from end to end
- Packets can be lost, arrive out of order, duplicated, arrive faster than we can process, don't say which application each packet is for, and being corrupted
- The transport layer provides a cleaner and reliable interface on top IP

TCP

- The transmission control protocol allows applications to transmit and receive a stream of bytes

- Application does not need to worry about: segmenting bytes into IP datagrams, bytes being dropped or duplicated, or arriving out of order
- TCP transport entity manages these TCP stream and interfaces with the IP layer. They accept user data streams, segment them into pieces, and sends each piece as a separate IP datagram. Recipient TCP entities will reconstruct the original byte stream from the IP datagrams

Primitive: core functions that TCP provides

- Listen, no packets, block until something tries to connect
- Connect, send Connection Request packet, actively attempts to establish a connection
- Send, sends data packet, send information
- Receive, no packets, block until data packets arrives
- Disconnect, send Disconnection Request packet, states that this side wants to leave the connection

TCP model

- Splits data into fix size segments into separate IP datagrams
- The remote combines the separate IP datagrams and delivers to the application the entire series of bytes
- The byte stream has no sense of message, no delimiter
- Both the sender and receiver creates sockets. Sockets are kernel data structures identified by a 5-tuple. There can be multiple connections to the same port as long as the sockets are different
- For a TCP connection to be active, the connection must be between a socket on the sender and a socket on the receiver

Features of TCP

- Full duplex, send data in both directions simultaneously
- End to end, between sender and receiver hosts
- Byte streams, no notation of messages. The application layer message boundaries (such as a single send) are not preserved, so that the receiver cannot tell where one "write()" ends.
- Buffer capable, where the TCP entity can buffer prior to sending or ceiveing. Reduces overhead but increases delay

TCP details

- Each TCP segments has 20-60 bytes of headers, plus zero or more data bytes.
- TCP entities can decide how large each segments can be. The two constraints are: smaller than the IP payload limit of 65515 bytes, a maximum transfer unit of 1500 bytes (ethernet packet limit).
- This maximum segment size DOES NOT INCLUDE the header, it is ONLY for the data.

- Uses a sliding window protocol for: reliable data delivery without overloading receiver, congestion control (reduce impact on the network layer)

TCP header

- Source port (16), dest port (16)
- Sequence number (32), Ack number (32) (for sliding window)
- TCP header length (in number of 32-bit words, not bytes) (4), some flags (8)
- Window size (16) (for dynamic sized sliding window)
- Checksum (16), urgent pointer (16)
- Options (0 or more 32-bit words)
- Data (optional)

The important thing is that most fields in the TCP headers are in units of 32-bits. This is because of performance reasons in parsing the header on early computers.

TCP header details

- Source and dest port. The source port is the socket port by whoever sends the packet, and the dest port is the socket port of whoever receives the packet.
- Sequence number. If SYN flag is set, the initial sequence number. If SYN flag is unset, the accumulated sequence number of the first data byte in this segment (from the start of the stream against the initial sequence number). This uses bytes to allow grouping packets together for retransmission, and detaching the packet/segment from the true byte units of the byte stream.
- There is an initial sequence number to prevent delayed packets to the same socket from a dead connection to arrive to another process
- Ack number. If ACK flag, the next sequence number byte the receiver expects
- Flags: SYN, ACK, RST, FIN
- Window size, the size of the receive window in bytes

If sending a long stream of bytes that overflows the sequence number, we simply let it loop around since the window size is smaller than the number of possible sequence numbers, so that it is impossible for the receiver buffer/window to have duplicated sequence numbers.

Connection establishment

- TCP is a connection orientated protocol running over a connectionless network layer
- Networks can lose, store, and duplicate packets, so this connection establishment can be complicated: congested networks may delay acks, IP can repeat transmissions, and any packet of which can arrive not at all or out of sequence
- The protocol can be represented as a time diagram, where time flows down and packets flow left and right.

The goals of a reliable connection establishment

- Ensures that one and only one connection is established, even if the setup packets are lost or retransmitted
- Establishes the initial sequence numbers for the sliding window

Three way handshake

- Avoids problems that can occur with duplicate sequence numbers
- Both sides exchange information about the sequence strategy, and agree on them before transmissing data
- Client will send a SYN packet with their SEQ number
- Server replies with a SYN/ACK packet with the client SEQ number as ACK and server's own SEQ number
- Client replies with a ACK packet with the server's SEQ number as ACK and client's own SEQ number
- At the end, both hosts will agree on their respective sequence numbers
- The client can immediate send the data after sending the final ACK, since the server can assume that the data packet is the ACK. This final ACK packet doesn't have to be a dedicated packet

When the client sends the SYN, it treats the connection as half-open. When the server receives the SYN and sends the SYN/ACK, it also treats the connection as half-open. The client treats a full connection when receives the SYN/ACk, and server treats a full connection when receives the ACK packet.

Three way handshake error handling

- Two simulatenous attempts of connection from both ends will lead to only one connection. Both sides will send a SYN packet, and reply with a SYN/ACK packet, thus completing the three way handshake since both sides will have the two sequence numbers.
- If the final ACK is dropped or the SYN/ACK is dropped, the client resends the SYN packet with the same sequence number, the server will then return SYN/ACK with the original server seq number.

TCP synchronization

- SYN bit is used to establish a connection. A connection request has SYN set and ACK unset, and a connection reply will have both set. The SYN is used for both connection request and connection accepted
- Sequence number is the first byte of this segment payload. We treat the initial establishment packets as having one byte of data
- Ack number is the next byte that the host expects.

TCP FIN

- FIN flag is to indicate a request to close a connection (A sends FIN implies that A doesn't want to send new data)
- FIN is directional. Once acked, new data cannot be sent from the sender to the receiver. Data can continue from the other direction, allowing the host to send the FIN without

waiting for a response. Note that the sender of the FIN can still retransmit unacked packets

- Typically requires four packets to close, one FIN and one ACK for each direction. Can be optimized with FIN, FIN/ACK, ACK. This doesn't usually happen since the server may want to keep sending data.

TCP reset

- The RST flag signifies a hard close. States that the sender is closing the connection and will not listen for any further messages.
- Sent in reply to a packet sent to a socket with no open connection.
- Can be used to close a connection, but should use FIN instead since it is more ordered.

Note that both the SYN and FIN packets count as one byte.

SYN flooding

- The server requires to store an initial sequence number per SYN request
- Attacker can make lots of SYN requests and not send the following ACK, causing the server to fill with sequence numbers
- Solution uses a SYN cookie, which is to overload the seq number to include: client address, port, and a timer. The server can then use the SYN cookie from the seq field in the client ACK to use as its seq field
- To prevent arbitrary client ACK seq fields, the server validates that the SYN cookie is correct for that specific host. This will have overhead so SYN cookies are only used when the server thinks that it is facing a SYN flooding attack

15 TCP Sliding Window

Sliding window provides

- Flow control (not sending too much data to client to process)
- Reliable delivery
- In order delivery

15.1 Flow Control

Buffers

- Sliding window is controlled by receiver. It determines the max data that the receiver can accept currently.
- Sender has a send buffer and receiver has a receive buffer. No guarantee that data is immediately sent or read from the respective buffer (client may wait until the buffer is large enough before sending the packet).
- The receiver will drop packets if they cannot be stored in their buffers. The sender will have to store the data if the window size is zero.
- The buffers are circular array buffers.

Protocol

- Sender sends data
- Receive receives data, reply with ACK and window size
- Sender can send a maximum of the window size of data
- If sender window size changes due to application, resend ACK with new window size. This is optional, however

Zero window probe. When the window is zero

- can send urgent data
- can send zero byte ack packets
- can send zero window probes, a zero byte segment that causes the receiver to re-annouce its window size. Prevents deadlock if the final ACK packet with new window size is lost

Sliding window. Note that send window and receive windows are implemenented on the respective buffers

- Send window is the data that the sender can currently send. Includes unacked packets and unsent data that can fit in the send window
- Receive window is the data that the receiver is willing to receive, represented in the window size of ACK
- When sender receives ACK with new window size, expand send window
- When receiver receives a packet, replies ACK with ack number on the earliest unreceived packet and new smaller window size (if the packet is out of order, ignore that in the window size. The window size is computed from the ack number).
- When receiver application reads packets, receiver sends window update ACK packet that updates the ack number and new window size. Alternatively let the sender use zero window probes.

Sliding window invariance

- Sender window packets that has been sent are called bytes in flight
- Last byte sent minus last byte acked must be smaller than the receiver window advertized

15.2 Segment loss

Segment loss

- If a data packet is not received, the receiver will ACK on the next received packet (out of order) with the ack number and window size unchanged
- The sender will resend packets when a timeout occurs with no ack number updates, or when there are three duplicated ack numbers causing fast retransmit

TCP originally used flow control loss handling from the link layer. This proved to be very bad and increases congestion. The two flow control methods are

- Go-Back-N, re-transmit everything from the first packet loss point to the full window size, receiver doesn't need to save out of order packets

- Selective repeat, only retransmit the lost packet. Since packets will arrive out of order, receiver must store these out of order packets and send them in order to the application
- The idea is that selective repeat is more complex, and only helps if loss is common. The belief is that link layers will reduce error/loss rates

Congestion collapse

- In 1980s, took tens of minutes to send a packet to the next building
- Router buffers were overflowing due to too many packets
- Senders were using go-back-n, so each packet loss causes N more packets to enter the system, further congesting the network
- Selective repeat (fast retransmit) solves this problem following the packet conservation principle: ensure that the total number of packets in flight is constant, only send a new packet when we know a packet is lost.

Fast retransmit combines selective repeat flow control with fast detection of packet loss. It states that we should send the last ack number packet (only that packet) when three dup ack numbers are received.

Deadlock

- If window size is zero and no packets are inflight, sender will not send any packets, and receiver will not ack any message. The window size may never update
- The receiver can notify the sender that the window size increased using a Window Update packet
- Sender can monitor the window size by sending a Zero Window Probe packet with zero payload on a timeout. The receiver will response with the new (maybe updated) window size

15.3 Congestion window

Congestion control

- When network is overloaded, congestion occurs that affects all layers with either packet losses or delays
- Lower layers (link and network) will attempt to reduce congestion, but tcp will affect congestion the most since it directly controls the data rate (like reducing the video resolution in real time via a protocol)
- Original tcp has the receiver choose a window size based on its buffer size. This removes congestion due to buffer overflow on the receiver receive buffer, but may cause congestion within the network (switches and routers) by having too many packets inflight.

Congestion control window

- An additional window that is dynamically adjusted based on netwokr performance to maximize transfer rates
- CWND (and the sliding window) determines the maximum packet inflight that the sender can send

- CWND maintained and used only by the sender. The sliding window is controlled by the receiver and is encoded in the tcp header. The CWND is not.

Slow start

- CWND initally set to the maximum segment size (MSS) with one packet
- For each packet acked, CWND adds MSS. This means that for each full CWND of packets acked, the CWND is doubled.
- Slow start grows until a packet loss or reaches a threshold value (slow start threshold)
- Typically the first slow start threshold is set to infinite

Additive increase (after slow start)

- Eventually too many packets will cause congestions and timeouts. When segment loss occurs, set threshold to half the current CWND and restart slow start (from CWND equals MSS)
- When reached slow start threshold, grow the CWND linearly by adding MSS only when a full window of packets are acked.
- Used in TCP tahoe

Multiplicitive decrease

- TCP reno optimizes the segment loss case
- A fast recovery when fast retransmit occurs, by starting from the new slow start threshold to avoid the slow start phase and going directly into additive increase

Large window connections

- Used by data centers where transfers are high speed and long distance. Tcp reno requires unrealistically small packet loss
- Requires new congestion control algorithms needed for bulk transfers over long distance
- IETF will not change due to things possibility breaking, companies thus implements their own standards on their own networks

16 Remote procedure call protocol design

Protocol design decisions

- How does an interaction look like: services exposed, packets sent, connection oriented or connectionless
- What data are communicated and how: what layer protocol is used
- What format is used: headers, fields, free-form data, valid data ranges
- Error handling

RPC

- Allows calling procedures on a remote server as if they are local to the client. Hides the networking part from the programmer

- Isn't a single protcol, but have multiple variant
- To hide networking, both client and server has a stub that acts like an interface. From the client perspective, all calls are local to the stub.
- Parameters and return values are marshalled and unmarshalled during RPC, converting memory data structure to a form that is transmitted and vice versa

17 IP Addresses

Network layer

- Role is to get data from end to end, with routers in the middle connected point to point so that it takes multiple hops (each hop is point to point)
- Must efficiently route traffic using routers (internetwork routing)
- Nodes/routers must be given names (ip addresses)
- Most network layer code are on routers. The data units sent between routers are called packets

Connectionless vs connection-oriented

- Connectionless like packet switching (IP). Minimum required service is to send a packet. Called a datagram network
- Connection-oriented like virtual circuit switching/network (ATM, MPLS). Usually acts as a single link in the IP network. Minimum required service: handshake, send data, teardown

Packet switching

- The internet is a packet switched network. Called store-and-forward packet switching since the router may have a backlog of packets to forward/switch
- Process is: host transmit to the nearest router
- Packet is buffered while it is arriving, verify checksum
- If valid, packet is stored in buffer until outgoing interface is free
- Router forwards the packet onto the next router in path
- Repeat

Forwarding tables

- Used to determine the path that a packet takes in forwarding
- Paths can change for packets in the same transport layer connection if table is updated
- For connectionless, maintain a forwarding/routing table that maps from packet destination to next router.
- For connection-oriented, a connection setups a fixed path to send all the packets. Each router has two indexed forwarding table: IN table containing source-virtual-number pairs, OUT table that contains dest-next-virtual-number pairs. To route, look at the packet host and connection number in IN, and use the index in the OUT table to forward to dest with a new VC number. Each VC number is local for each hop.

Pros and cons

- Datagram networks don't have to hold state information about connections. VC has problems in reboots
- Datagram network requries each packet having full source and dest information. VC circuit only requires a short VC number
- Routing in datagram is independent. Routing in VC is defined at connection setup
- Quality in datagram is bad, in VC is good
- Congestion control in datagram is bad, good in VC
- Link failures can be recovered easily in datagram, but hard in VC

IP design principles

- Something that works OK is better than an ideal standard in progress
- Keep it simple via Occam's Razor
- Be strict when sending and tolerant when receiving. Web browsers accepting invalid html
- Make clear choices in a standard
- Avoid static options and parameters, rather negotiate them at runtime
- Think about scalability
- Packet switching based on a best effort model, with no performance guarantee

IP properties

- Responsible for moving packets through network from end to end
- Multiple paths available in the network for redundancy
- Routing algorithms used to determine best path

IPv4 header

- Version
- IHL, header length in 32 bit words
- Differentiated service for QoS
- ECN, explicit congestion notificiation
- Total length including payload
- Identification/fragmentation offset, for sending fragmented packets
- Time to live, TTL, countdown of hops that is decreased at each hop, dropped when TTL is zero
- Protocol for transport layer service
- Source and destination address
- Option, rarely used

IPV4 address

- 32-bit number, 4 bytes
- Expressed in decimal, where each byte is shown as a decimal 0-255 separated by periods
- 0.0.0.0 is lowest, 255.255.255.255 is highest
- IP addresses are given to network interfaces and not hosts. A host with multiple interface cards have multiple IP addresses
- Supply of IPV4 address are basically exhausted

Address types

- Unicast, one destination, typical address
- Broadcast, send to everyone in network. Fixed by standard
- Multicast, send to a particular set of nodes, used for streaming videos of live events. Fixed by standard

Network and host address hierarchy

- IP address encodes both network and host number
- Network address in top bits (MSB), host address is lower bits. The network part is identical for hosts in the same networks.
- Allows scaling, as packets can be first sent to the network, and let the network router to determine the host
- IP addresses are assigned to networks in blocks defined by MSB bits. So each network corresponds to a contiguous block of IP address space, called a prefix.
- Prefixes are written as the lowest IP address possible followed by slash with the size in bits of the network portion
- Interface need to know the size of the network portion it is assigned to. It cannot tell the subnet mask given just the ip address. All interfaces in the same subnet will have the same prefix/subnet-mask.
- Trade-off between network bits and host bits
- A subnet mask is a binary mask of 1s that when AND against an ip address grants its network prefix. It is a one if that bit is part of the prefix.
- For routing, networks are assigned blocks of address, and subnet routers only need to maintain routes for host with that prefix. Only when the packet arrives at the destination subnet does the host portion need to be read

Route aggregation

- Performed automatically, usually halving the routing table size because all subnet hosts have the same prefix address
- The idea is to combine multiple dest ips into a single aggreggated prefix/address if they are sent to the same router
- If multiple prefix matches in the routing table, forward to the destination with the longest matching prefix.

Router networks

- The link between two routers is a network all by itself
- Often uses /31 prefixes, with the last bit indicating the left or the right router in the link

IPV6

- Designed to solve the problem of exhausting ipv4 addresses
- Other changes were made: simple header, improve security, further QoS support
- 128 bit addresses

IPV6 header

- Version, the number 6, in the same position as ipv4 version
- differential serviced
- flow label for virutal circuit id
- payload length
- next header (protocol tcp/udp)
- hop limit, same as time to live
- two 128 bit ipv6 source and destination address

IPV6 addresses

- 8 groups of 4 hex digits separated by colons
- Add square brackets around address if grouped with ports
- One group of consecutive zeros can be written as two colons
- Backwards compatiable with IPV4 using `::ffff:x.y.z.h`, ie, setting first few bits zero and ones for the bits before the 32 bits

IPV6 not yet widely deployed, with only around 30% global use. Likely to see continued growth overtime.

18 IP Routing

The routing algorithm decides the series of routers that an incoming packet should be transmitted on. It directly creates the forwarding table.

Good routing algorithm properties

- Correctness, finds a valid route between all pairs
- Simplicity
- Robustness, router crash shouldn't require a whole network reboot
- Stability, reaches equilbirium and stays there
- Fairness in capacity
- Efficient

- Flexibility in adding new policies

Delay vs bandwidth

- Either optimizing delay or network throughput
- Simplest approach is to minimize delay by reducing the number of hops a packet has to make. Tends to increase bandwidth and improves delay. Also can reduce the distance travelled by packets, but not guaranteed since a long link (crossing pacific) is a single hop
- Actual algorithms will weight each hops/link that is more flexible and represents preferences.

Routing algorithms

- Non-adaptive, static routing. Does not adapt to changes in network topology. Forwarding table calculated offline and uploaded to router at boot. Does not respond to failures in neighboring routers. Reasonable if there are no other choices for routing (always forwarding to the same address like your home router to ISP router).
- Adaptive, dynamic routing. Adapts to changes in topology and even traffic levels. Optimize some proeprty like hops. Can get information from adjacent routers, or from all routers in the network.

Flooding

- Simplest non-static routing
- Send packet to all neighbors who don't have it yet. In practice, sends packet to all neighbor, each node will drop a flooding packet if it receives duplicate by using an identifier
- Guarantees shortest distance and minimal delay
- Useful benchmark for speed of routing
- Extremely robust as it will find any valid path
- Highly inefficient generating lots of duplicated packets
- Have to have a way of discarding packets via TTL. Can be set of the diameter of the network

Adaptive routing considerations

- Needs something more efficient than flooding
- Should adapt to topology changes
- Given that we have the complete knowledge of network topology, need an algorithm to find an "optimal" route.

Optimality principle

- If router j is on the optimal path from i to k, then the optimal path from j to k is on the same path.
- If a better route existed from j to k then it would update the i to k path, contradicting our initial assumption

- Does not always apply in BGP as that also needs to consider other factors than just distance/hops

Sink tree

- Collorary from the optimality principle
- A set of optimal routes from all sources to a sink forms a tree rooted at the sink
- The routing algorithm on a router will find this sink tree with the sink at each router

Shortest path algorithm

- Translate to a labelled, weighted graph
- Dijkstra's algorithm find the shortest path between a sink and all sources, assuming all non-negative positive

Dijkstras

- At each step, nodes are labeled with their distances from source and the best known path
- Nodes are divded into three groups: unseen (no idea), open (visited its neighbor but not it), closed (visited it, knows best path)
- Moves nodes from unseen to open to closed
- Overtime, the labels in each node will be updated. Closed state implies the shortest path is found. Open state means a path is found

Link state routing

- A distributed algorithm using dijkstras. Replacing the original distance vector (more robust) routing that had slow convergence
- A family of protocols, variants of LS routing still used today.
- 5 step process:
- Discover immediate neighbors and learn their address
- Set distance or cost metric to each of its neighbors
- Construct a packet containing all its just learnt about neighbors
- Send packets to all other routers via flooding. Accept flooding packets from all other routers doing the same
- Compute shortest path to every other router after receiving enough paths to map out the network

LS routing specifics

- On boot router sends a HELLO packet to each of its neighbours. The router on the other end must reply with their unique ID. Used to discover neighbors
- Costs set automatically or manually. Can use the inverse bandwidth or delay (from ECHO packet). Many networks will manually choose preferred routes and design link costs that makes those routes the shortest (backwards to design costs manually).

Link state packet

- A router ID, sequence number, age, and a list of neighbours with their costs
- Building packet is easy, deciding when to build/send them (at intervals, neighbors changed) is hard.
- Flooding is used to send link state packets to all other routers. Precisely, reliable flooding (that requires ACK) to ensure that all routers received the packet.
- To avoid waste, routers will compare the new link state packet to the one it previously received from the same router id. If sequence number is not larger it discards it and not forward it as it is out-of-order or duplicated
- Sequence numbers are only 32-bits and wrap-around will occur, dropping all packets. Alternatively can be an issue if router restarts and its sequence number is reset to zero. Solution is the age field is decreased each second per received packet. When its age is zero, that packet is effectively removed so its sequence number check is removed

Border-Gate-Protocl (BGP) is not examined

19 IP control protocols

Data plane vs control plane

- Protocols don't really form a stack for the network layer (BGP uses TCP for updates). Better model is to think of different planes of protocols forming multiple stacks, with each plane achieving a different task, within the same layer
- Data plane is for forwarding in the network layer
- Control plane is for routing in the network layer
- Management plane is for setting BGP policies in the network layer
- The control plane uses the data plane to do its routing logic, it may also cross protocol layers.

Control protocols

- Protocols used by the network layer to manage its properties
- ICMP, DHCP, ARP (technically not internet layer)

19.1 ICMP

Internet control message protocol

- Are packets that are sent over the network layer directly, sent usually by hosts or routers. Doesn't use transport or application layers
- PING is an ICMP message, but ICMP handles more types
- Each packet has a message type: destination unreachable, time exceeded, parameter problem, source quench, redirect, echo, timestamp echo, router advertisement

Traceroute

- Exploits the time exceeded ICMP message

- Traceroute is a program that maps the series of hop routers with timings (latency) for a packet to reach a destination host
- Traceroute sends out ICMP PING packets to the at destination incrementing TTL. Counters will hit zero at successive routers, causing them to return a time exceeded ICMP packet, containing the IP address of that router in IP header. Repeat until host is reached.
- Use traceroute command. Each TTL is sent 3 times, displaying astrix if no response and the router IP if a time exceeded is returned.
- The router doesn't need to send the ICMP time exceeded packets, so some TTL will not return an address in the traceroute.
- Since routers have many ip address, it may send any of those in traceroute with no guarantees

19.2 DHCP

Dynamic host configuration protocol

- Internet layer requires that each interface have a unique ip address
- Could manually configure each host, but this is difficult, error prone, and slow to respond to new devices
- DHCP automatically allocates ip addresses (subnet mask, dns server, default gateway) within a subnet.
- Note that there may be security concerns on the subnet side as connecting any device will have DHCP issue an ip address. Also security concerns on the host side since the subnet can issue a malicious dns server

DHCP creates a mapping from a fixed link layer address (MAC address) to a dynamically allocated network address (IP address). This means the same device will always (mostly) be dynamically allocated the same address.

DHCP protocol

- Network has a DHCP server
- Host sends a DHCP discover packet to a router. Routers are configured to relay (forward) any of these packets to the DHCP server, even across different networks/subnets
- DHCP server receives the request and responds with a DHCP offer packet containing an available IP address.
- There are no source ip address to send the response back to. Instead, use the lower layer/network's address called MAC address and route via that back to the host.

DHCP dynamic allocations

- IP addresses are issued on a lease — the time til the ip address is reclaimed by the server and needs to be re-issued
- Hosts can request a renewal before the lease expires. Should do this as existing tcp connections will break when ip address expires/changes. This can hurt large file transfers using a single connection, but will not harm single use connections like HTTP.

- Also allocates a bunch of other parameters like default gateway, dns server address, time server address

MAC (Media access control) address

- Used to route the OFFER packet back to host
- Also called ethernet/wifi/physical address. They are not the physical layer address though.
- A globally unique identifier for the interface hardcoded by the manufacturer of the interface. Between 48 to 64 bits long.
- An address used in the data link layer

19.3 ARP

Address resolution protocol

- ARP is the link between the internet layer to the underlying network/link layer by translating ip addresses to mac addresses.
- Used by the link layer to determine which destination interface to send an ip packet to. The idea is: ip routing algorithm determines next hop router ip, sends packet to link layer, the network interface uses ARP to resolve the router mac address, and forwards the packet to that mac address.
- Steps: broadcast an ethernet/wifi packet asking how owns the target ip address; broadcast will arrive at every host on the entwork; owner will respond with its mac address.
- Note that ARP is completely operated on the link layer using mac addresses.
- Runs ARP a lot to find out how to communicate with the neighboring routers. Don't need to use ARP on every IP packets as we can time cache the mapping from the last ARP. Instead, repeat ARP when no replies within the link layer indicating changed IP.

ARP properties

- Simple, but not efficient
- Security issue: no auth; intermediate routers can cache responses; can be spoofed for man-in-the-middle attacks
- Namely, can intercept and spoof the ARP request message to associate the attacker's mac address with another host's ip address (often default gateway, dns, or target website)
- Accept security risk since ARP is only used within a network, where we likely owns all devices

Global scale of the internet

- Inter-networking is the physical connection between networks
- Global communcation are dependent on lots of undersea cables. Each cable contains optic fibre connections

20 NAT

IPV4 address scarcity

- IPv4 addresses are becoming scare, so we need methods to handle more clients/hosts
- IPV6 solves this problem, but a temporary stop gap solution is needed
- Private address are ipv4 addresses that are reused within private subnets only and cannot be advertized. This is applicable if hosts in a company only needs to have internal access among the same subnet. These addresses will be unique within a subnet but not unique across the entire internet
- Three different sizes of private addresses: 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8
- Goal is to use an application layer proxy (HTTP proxy) to access outside services. Instead, people simply used a NAT transport layer proxy router to proxy the requests

Network address translation

- Each home is assigned a public ip address
- Internally hosts are issued private ip addresses
- Internal ip address are used to communicate within the local area network LAN. These cannot be used on the public internet
- When packet is heading out to the public network, the NAT router translates the internal address to its public ip address. The port is also translated to a free port on the NAT router (network address and port translation)

NAT details

- A transport layer proxy that assumes TCP/UDP, in particular the location of the source and destination port fields in the header
- Replaces IP source address with own public ip address
- Replace source port with NAT router port from the entry in the NAT translation table. There are 65536 entries in the table (one for each port). Each entry contains the original private ip address and the original source port number and has the index of the router port
- Recompute IP and TCP checksum after translation
- When packet arrives from the internet, the NAT router looks up the original source port and address from the destination port in the packet in the TCP header in the translation table. Recomputes checksum of the packet and forwards to internal host.

NAT criticisms

- Breaks end to end connectivity, as an interface in the private network can only receive packets once it has sent packets out and the NAT router created a mapping. Can't run a server in the private network
- Violates the IP architectural model that states every IP address indicates a unique interface on the internet.
- Layering violation by snooping on the payload contents. Initially NAT only worked on TCP and UDP packets. It was changed to work for FTP messages by snooping on it

- Changes internet to be pseudo-connection oriented rather than connectionless, as the NAT maintains the connection states by its NAT translation table. This slows down the connection from private to public, and prevents any connection from public to private. If NAT router crashes all connections are lost, and TCP retransmission will fail to recover the connection since the translation entries are gone.
- Limited number of simulatenous connections since there are only 16bits for the ports.

Benefits of NAT

- Widely deployed particularly in homes and businesses. Also contains carrier grade NAT where ISP gives customers private address only
- Significant security advantage as packets can only be received once an outgoing connection is created. So the internal network hosts are shield from attacks from incoming unsolicited packets
- Not a firewall, since a firewall has precise control over the type of traffic that can be exchanged, while a NAT box can still forward unsolicited packets if the attacker guesses the port number of an existing connection
- Likely to remain in use even after IPV6 is widely deployed and there are no scarcity of IP addresses for its high security.

Universal plug and play

- Overcomes the weakness of NAT, by implementing the internet gateway device protocol that allows static port mappings to be created in the NAT to allow servers to run on the internal network
- This suffices the need for local hosts to be servers.
- Bad implementation can leak NAT translation tables

21 Debugging

Symptoms

- Error in one application or all
- One, many, all, none responses
- Very slow
- Access denied
- Partial response

Possible problem locations

- In space (locla host, server, router, NAT, firewall)
- In the network stack (physical, link, network, transport, application)
- Some combination doesn't make sense

Tools

- Network layer forwarding, ifonconfig, route, arp

- Network layer control, ping, traceroute
- Application layer DNS, dig
- Application layer, CURL
- Server, speedtest, downdetector
- Application layer, browser devtools
- Task manager

For exams, remember the functionalities available to check problems across space and across protocol stack