Week01

Examples of OS:
Windows, macOS, Unix, Linux, Android
Graphical User Interface (GUI) or a command line shell is **not** an operating system

Resources managed by OS Kernel: One or more processors, Main memory, Disks, Printers, Keyboard, Mouse, Display, Network interfaces, I/O devices

Operating system sort of gives applications a layer of interaction between hardware and software.
Functionality:
1. Provide an abstraction of hardware resources
   - Provide abstractions to application programs
   - Manages pieces of complex system
   - Provide orderly controlled allocation of resources
2. Manage these resource and allocate them to operate different programs

Program is static and **process** is dynamic: C code is static, it's a program. When run it, it's a process
Processes: a program in execution
- Process is associated with an address place and also associated with other set of resources
- Process can be seen as a container which holds all information needed to run a program

At any time the process number can be 0,1,2,3,.., and the processes are independent unless you create dependency on them
The state of process consists of the code or text of the program, the value of all variables, both in memory and in registers, the address of the current instruction, and probably other data as well.
Process memories segments:
- Text (program code, usually read only)
- Data (constant strings, global variables)
- Stacy (local variables)

Memory hierarchy:

Typical access time	type	typical capacity
1 nsec	registers	<1kb
2 nsec	cache	4 MB
10 nsec	main memory (RAM)	1 - 8 GB
10 msec	magnetic disk	1 - 4 TB

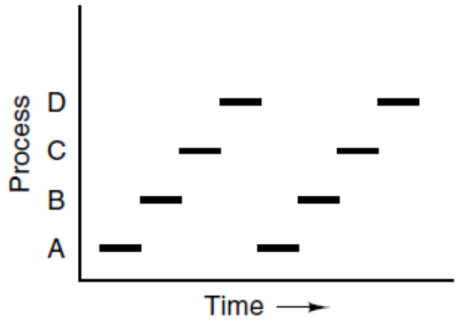
Processor / CPU (central processing unit):
Fetches instructions from register (memory); executes them
Registers keep some process state information
Two models of execution:
- Kernel (all instructions can be executed)
- User (only a subset)

A CPU can only execute a single process one time

User vs system processes:
- Most processes that a typical user deals with are created by that user (by invoking a program, either from the command line or via a GUI)
- However, some services provided by the OS may also be implemented as separate processes. A typical example is a print daemon.

Multiple processes can run in 'parallel'

- Conceptually each process has its own virtual CPU.
- In reality, multiple processes will share a CPU, each running for a small period of time in turn. This is called **multiprogramming**.
- When a process needs an I/O, it doesn't really need the CPU, then the other process can gets the CPU share
- **Multiprogramming**: increases system **efficiency**. When one process needs to wait for e.g. data from disk or keyboard input, another process can make use of the CPU, is useful even when the machine has two or more CPUs, since (for the foreseeable future) the number of processes will sometimes exceed the number of CPUs.

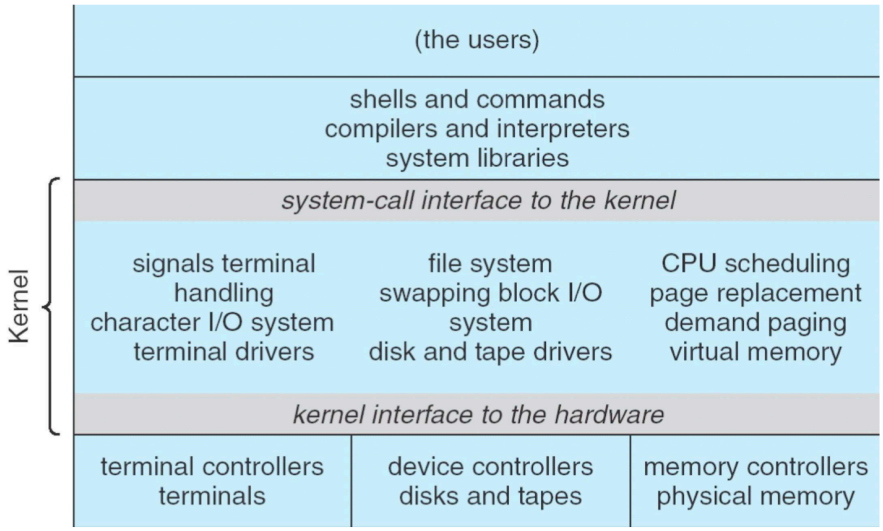


Each process has its own address space to store the state so that we can determine which process can make share of the CPU next.

Kernel:

- If several processes are to be active at the "same" time, something has to ensure that they do not get in each other's way or steal other process's variables. That something is the privileged part of the operating system, usually called the kernel.
- The kernel also provides services such as "read N bytes from this file". These services are used by application programs, utilities, and by the non-privileged parts of the operating system.
- The kernel is not itself a process because it cannot be terminated.

Unix System Structure



User-kernel distinction:

Most CPUs have two modes (some have more). The **program status word** (PSW) register gives the current mode.

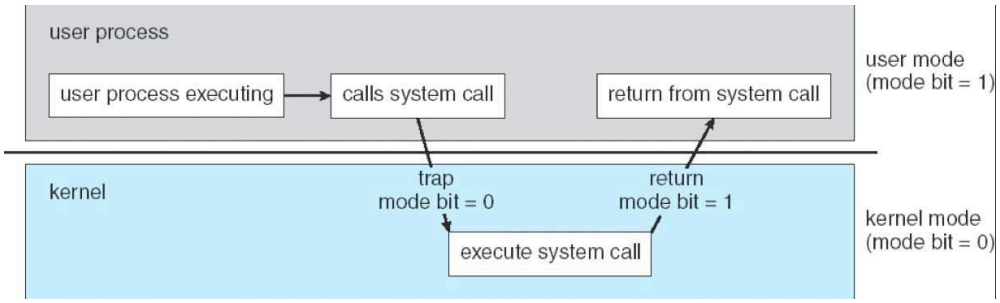
- Code running in **user mode** cannot issue privileged instructions, and can access only the parts of memory allowed it by kernel mode code.
  e.g. if the user mode process can access all instructions, the interruption handling may be ruined, as the process can tell CPU not to take any interruptions. The process may have too much control of system. The access of address, memory or disk is also not permitted.
- Code running in **kernel mode** (also called system mode or supervisor mode) can issue all instructions, and can access all memory.

Instructions and memory locations whose use could interfere with other processes (e.g. by accessing I/O devices) are **privileged**.

The user mode / kernel mode distinction is the foundation needed by the kernel for the building of its security mechanisms.

Having two modes allows designers to run user programs in user mode and thus deny them access to critical instruction

Transition from user to kernel mode



When user process its executing and needs something they do a system call (usually provided by the library). Mode bit (PSW) provided by hardware provides the ability to distinguish when the system is running user code or kernel code.

- From user to kernel mode:
  - Interrupts
  - Exceptions
  - System calls
- From kernel to user mode:
  - New process starts
  - Return from interrupt or system call

System calls

- Usually *package* things then batch them up in kernel mode, as the switch is expensive
- Exist to allow user programs to ask the kernel to execute *privileged instructions* and access *privileged memory* locations on their behalf. Since the OS checks those requests before executing them, this scheme preserves system integrity and security.
- To make system calls more convenient to use, a system call will typically do several privileged things in carrying out one logical operation, e.g. reading N bytes from a file on disk.
- To the application programmer, a system call is a call to a privileged function.
- Example system calls on Unix open, read, write, close, fork, exec, exit, wait
- Typical system class (OS independent list)
  - Process control, e.g.. load/execute; create and terminate a process; get/set process attributes
  - File management e.g.. create/delete/open/close/read from a file; get/set file attributes
  - Device management e.g.. request/release a device; read/write from a device
  - Information maintenance e.g.. get/set time or date
  - Communication e.g.. create/delete communication connections.
- The printf function in C library do write() system call function to handle.

Related Question: To a programmer, a system call looks like any other call to a library procedure. Is it important that a programmer knows which library procedures result in system calls? Under what circumstances and why?
As far as program logic is concerned, it does not matter whether a call to a library procedure results in a system call. But if performance is an issue, if a task can be accomplished without a system call the program will run faster. Every system call involves overhead time in switching from the user context to the kernel context. Furthermore, on a multiuser system the operating system may schedule another process to run when a system call completes, further slowing the progress in real time of a calling process.

Source Control

Version control can keep track of changes to do in files and go back into histories.
Audit all the changes have been made in group works.
Reason for using version control:

- collaboration: allows collaboratively working on the files,
- revision history: keeps track of all the changes and by who,
- audit changes,
- backup files

Cloning a repository copies this directory from the remote repository, which contains a full history of all the files in the repository
Use new branches to create your own branch to test code without effecting other's codes

Git stores changes remotely by syncing repositories
Git steps:
- Working directory – top level folder with single version of the files
- Staging area – file in .git that records files to be added to next commit
- .git directory – configuration and repository database

Storage:
- It takes only snapshots: if the file has changed, the file will be covered.
- If the file hasn't changed, it will mark it hasn't changed, same as previous one (aka referenced).
- It will create a new copy (snapshot) of the file instead of store the changes.

Git database:
- Just a key-value database
- put a value in database returns a SHA1 hash (40 hex-character identifier) [provide integrity]
  - The value is stored as a *blob* in the objects folders
  - The SHA1 hash is the key
- Use git status to see this

Git – Add - **Staging**
- Create a readme.txt file with "Hello World" in it
- Staging is the process of marking the file as to be included in the next commit
- git add is the staging command, staging readme.txt:
  - Adds a key-value for the file contents (create the pair)
  - Adds the file path to the staging index

Git – **Commit**
- The commit itself is a git object
- Hash all things in staging area since checkout
- Creates a git tree object of the changes, and wraps them in a commit.
- The tree object contains Unix directory entries
  – Permissions, path, blob key

Log: see the previous commits with messages
Checkout: Checks out – switches the working tree to a particular commit/branch (generally the head, but can be a past commit)

Git init	initialises a repository
.git folder	is where everything is stored
git status	to see the database of git
git add <filename>	to stage the file
git commit - - message "<message>" between - -	commit the added file, there's no space
git log	to display log of activity, gives a commit has code
git checkout <…>	<…> can be used to check out the <> is <…> from the last step
git checkout -b <new-branch-name>	create a new branch from the commit.
git clone	sets the remote references in the git config file
git push	perform two steps fetch and merge
git pull	transfers your local commits to the remote repo

- Tags can be used to add more informative names to commits, i.e. releases 1.0
- A release is also often stored on a branch, this allows bug fixing to continue without impact on future development
- Git doesn't solve inherent conflicts, two people editing the same file can still create a problem. General workflow is to commit/push often, with small changes.
- Master → Dev → Feature
- Not uncommon for developers to be unable to merge onto Dev: maybe restricted, or require code review prior to senior staff merging branches
- Binary files can be inefficient to store in git
- Git automatically packs file changes using delta compression
- Binary files typically exhibit large changes (JPEG recompression)
- Increases the size of the clone operation
  Forks – complete separate copy of repository
- Link remains to parent
- Useful for large changes, keep development work out of main repository
- Remains even if original repository is removed

Git(Hub) – Continuous Integration
- Snippets of code running after push (e.g., test cases)
- Important part of software development
- Continuously build, test, and deploy iterative code changes
- Reduce the chance of maintaining test-failing code

- Build processes that depend on external repositories are fragile
  - A user deletes the repository -> build process breaks
  - Risk of unknown code changes in builds
- Better to fork the repository
  - Higher support costs – need to pull changes
- Security
- Be careful what you store in public repositories
  - Never store credentials/aws keys

Week 02

Process creation:
1. System initialisation.
2. Execution of a process creation system call by a running process.
3. A user request to create a new process.
4. Initiation of a batch job.

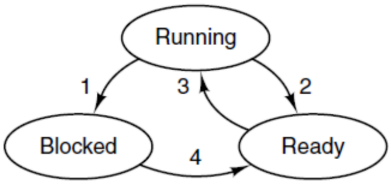
fork(): Creates a new process, copy of the parent process, returns a pid value (each process has a process id)
execve(): is used after a fork to replace one of the two processes virtual memory space with a new program

Process termination:
- Normal exit (voluntary)
- Error exit (voluntary)
- Fatal error (involuntary): e.g. division by 0, try to visit privileged memory
- Killed by another process (involuntary)

exit() terminates a process
A parent may wait for a child process to terminate: wait provides the process id of a terminated child so that the parent can tell which child terminated; wait3 allows the parent to collect performance statistics about the child

Process state
- Running (actually using the CPU at that instant).
- Ready (runnable; temporarily stopped while another process is running).
- Blocked (unable to run until some external event happens). E.g. waiting for an I/O

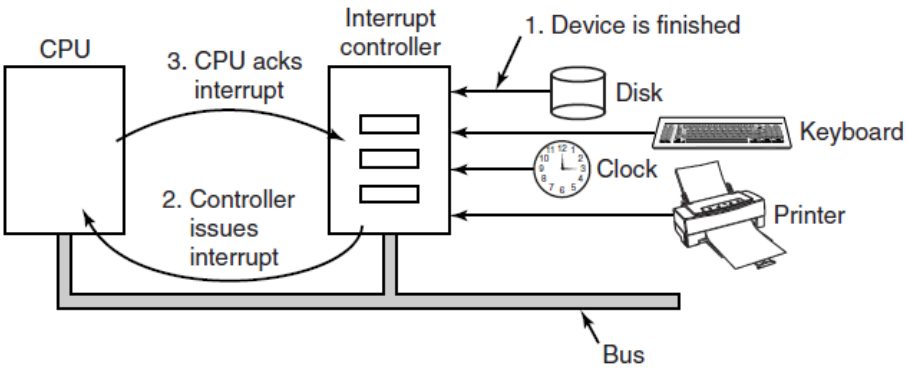
Process states:
- Process blocks for input
- Scheduler picks another process
- Scheduler picks this process
- Input becomes available

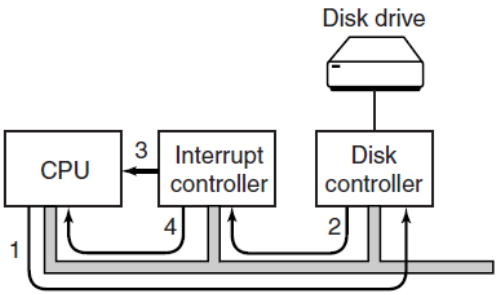


Process state
The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

Process Table
- One entry per process
- Contains state information to resume a process
- Fields include information about:
  - process management (e.g., registers, program counter, program status word)
  - memory management
  - file management

Interrupts





The steps in starting an I/O device and getting an interrupt.
The cpu may running another independent process, however the CPU has to acknowledge the interrupt has occurred, e.g. handle it later or handle it straight away. So the cpu need to remember and copy the state of this process and then load the interrupt handler. All happen in kernel mode.

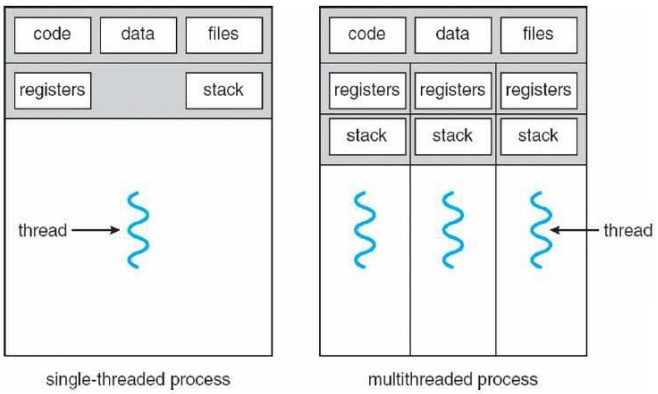
- When a hardware device needs attention from the CPU, e.g. because it has finished carrying out its current command and is ready to receive its next command, it generates a signal to interrupt the CPU.
- Asynchronous with the currently executing process (won't terminate the running process, call back when finishing)
- When an interrupt occurs, the CPU's hardware takes the values in the program counter and program status word registers (and, on some kinds of machines, the stack pointer register), and saves them in privileged memory locations reserved for this purpose.
- It then replaces them with new values.
- The replacement PSW will put the CPU into kernel mode. The replacement PC will cause execution to resume at the start of the interrupt handler, code that is part of the kernel.

Interrupt vector: address of the interrupt handler
The **interrupt handler** must
- save the rest of the status of the current process,
- service the interrupt,
- restore what it saved, and
- execute are turn from interruptor similar instruction to restore whatever the hardware saved when the interrupt occurred (i.e. the PC, the PSW).

Pseudo-interrupts:
- True interrupts come from hardware devices outside the CPU, pseudo-interrupts from the CPU itself.
- User programs may generate pseudo-interrupts inadvertently, e.g. by executing divide-by-zero: such events are usually called exceptions. Some exceptions cause process termination. This should be specially designed to catch! Otherwise the process would be shut by OS.
- Users can generate pseudo-interrupts intentionally by executing a special instruction for system calls (e.g., CTRL-C)
- Catch interrupts with trap command

If we use fork to create new process, it's a complete clone of the first one, however if the variables changed in the child process, it's not be seen by the parent process. Sometimes we want to work in parallel way. Thread is a kind of 'lightweight process'
Thread: a sequential execution stream within the process.
Threads are the basic unit of CPU utilization – including the program counter, register set and stack space.
Threads can communicate with each other without invoking the kernel – threads share global variables and dynamic memory.
Threads can run in different speed (one thread get CPU more than others), it can do independent tasks
Shared by threads: address space and memory – code and data sections; contents of memory (global variables, heap); open files; child processes; signal and signal handlers;
Threads own copy: program counter; registers; stack(local variables, function call stack); state (running/waiting etc.).
Global variables are shared across threads.
- thread **switches** could occur at any point
- thus, another thread could modify shared data at any time
- consequently, there is a need to synchronize threads– if not, problems could arise.

Threads vs. processes
A process has one container but may have more than one thread, and each thread can perform computations (almost) independently of the other threads in the process.
Having threads is much more efficient than having processes, as operating system is not included in creating threads, creation of threads is involved in process.
There is reduced overhead than when using multiple processes
- less time to create a new thread;
- less time to terminate;
- less time to switch between threads;
- less time to communicate between threads.

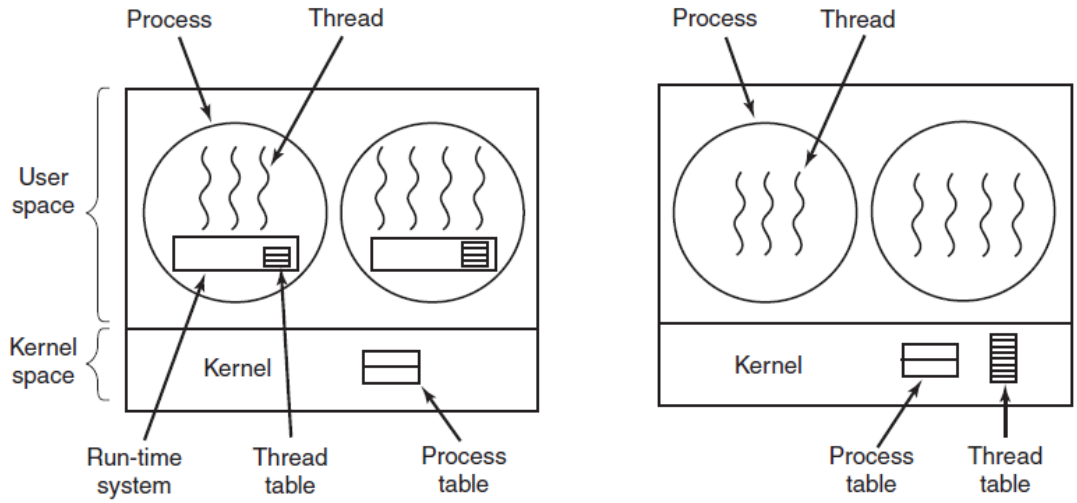


Pthreads
A POSIX standard API for thread creation and synchronization.
- all functions start with pthread
- include pthread.h
- all threads have an id of type pthread

Commands:	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

A system call may be blocking or nonblocking depending on its internal semantics.
`pthread_create` is nonblocking because it creates and starts the thread, and returns immediately. This is required in order to be able to start multiple threads and have them perform work in parallel. This would be impossible if it blocked until the completion of the spawned thread.

User Kernel Thread



A **user-level** thread package (left). The CPU may doesn't know which thread is being executed, e.g. if one thread is waiting for I/O or waiting on the disk, the whole process is blocked by OS even though some other threads doesn't need to wait for that.
A threads package managed by the **kernel** (right). The threads can be managed by OS

Related question: In the lectures, a multi-threaded text editor was shown. If the only way to read from a file is the normal blocking read system call, do you think user-level threads or kernel-level threads are being used for the text editor? Why?
A worker thread will block when it has to read a file from the disk. If user-level threads are being used, this action will block the entire process, destroying the value of multithreading. Thus it is essential that kernel threads are used to permit some threads to block without affecting the others.

Multi-threading is extremely powerful, but it comes with a significant challenge: Concurrency
If you are running multiple threads with a shared memory/data store modeling how they interact becomes critical, otherwise the following can happen:
- Race conditions – where the output is dependent on the sequence/timing of events
- Deadlock – each thread is waiting for another thread to complete a task

Requires locks, synchronization, and careful analysis

Interprocess communication
- Multiple processes improves efficiency,
- Why concurrency? increase efficiency (parallelism) through cooperation
- Exchange information between processes/ threads (have excess to same file)
- Processes can interfere with each other
- Proper sequencing and order (the order of execution is hard to predict)
- Ensure system integrity and predictable behavior

Race condition
- Multiple processes have access to shared object
- All can read and modify it
- Race condition arises when output depends on the order of operations
- Hard to debug

Deal with race condition:
- prohibit access to shared object at the same time
- Identify **critical region/ section** of the program

Requirements to avoid race conditions:
- No two processes may be simultaneously inside their critical regions.
- No assumptions may be made about speeds or the number of CPUs.
- No process running outside its critical region may block other processes.
- No process should have to wait forever to enter its critical region.

Techniques for Avoiding Race Conditions
Methods
- Disabling Interrupts
- Strict Alternation
- Test and Set Lock
- Sleep and Wakeup
- Other: Semaphores, Monitors
- Message passing

Implementation:
- Busy Waiting
- Blocking

Strict alternation with Busy Waiting

Thread A

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

Thread B

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

Test and Set Lock (TSL)
Most CPUs come with a test and set lock instruction
TSL RX, LOCK
Set RX to LOCK
Test Lock, if it is 0, set lock to 1 //this is called atomic operation
RX can be used to decide whether to enter critical region or not: Compare RX and LOCK

Busy Waiting
When a process wants to enter a critical section
- it checks if the entry is allowed
- If not, the process executes a loop, checking if it is allowed to enter a critical section

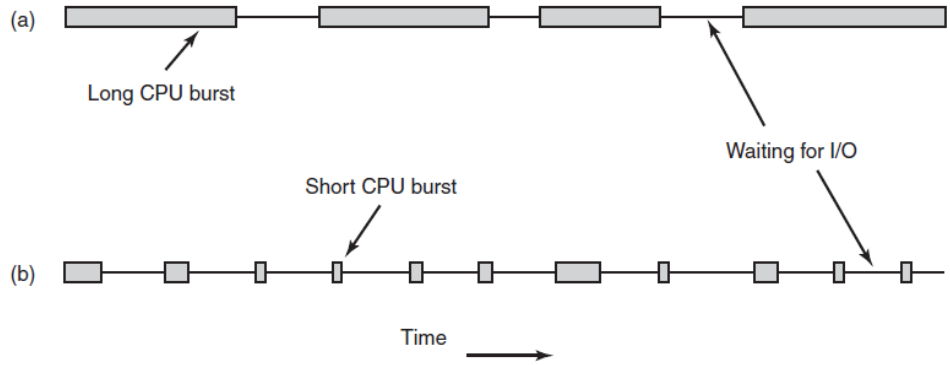
Advantage: generally easy to design and debug.
Cons:
- Waste of CPU
- Priority Inversion Problem
  - Low and high priority processes
  - Low priority process may starve

Related Question: In the lectures we saw that the priority inversion problem can happen with processes. Can the **priority inversion problem** happen with **user-level** threads? Why or why not?
Consider a setting where low-priority process and high-priority process have access to a shared resource or object. The priority inversion problem can occur when a low-priority process is in its critical region (i.e., accessing the shared object) and suddenly a high-priority process becomes ready and is scheduled. If the high-priority process uses busy waiting to get access to the shared object, it will run forever. With user-level threads, it cannot happen that a low-priority thread is suddenly preempted to allow a high-priority thread run. There is no preemption. With kernel-level threads this problem can arise.

Blocking
– Attempt to enter a critical section
– If critical section available, enter it
– If not, register interest in the critical section and block
– When the critical section becomes available, the OS will unblock a process waiting for the critical section, if one exists
- Example: Sleep() and Wakeup()
  Using blocking constructs improves the CPU utilization

Deadlock: A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause
May exist in many shared resource settings; not just memory
Deal with deadlock
- Ignore the problem
- Detection(e.g., graph algorithms) and recovery (Draw the graph and find cycles)
- Avoid by careful resource allocation
- Prevention

Process scheduler
Scheduler picks which process to run and for how long How: based on a scheduling algorithm
Decide when to have process creation, process exit, process blocks, interrupt.
Scheduler input: processes in ready state (kept in run queue)
Schedule is hard:
- Different workloads
- Different environments
- The scheduler has only a limited information about the processes



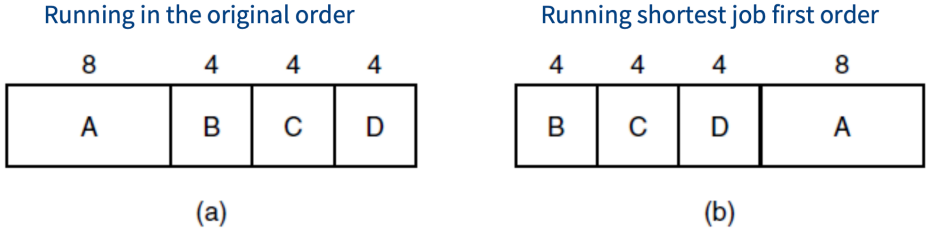
Bursts of CPU usage alternate with periods of waiting for I/O.
(a) A CPU-bound process
(b) an I/O-bound process.

Related Question: Can a measure of whether a process is likely to be CPU bound or I/O bound be determined by analyzing source code? How can this be determined at run time?
In simple cases it may be possible to see if I/O will be limiting by looking at source code. For instance a program that reads all its input files into buffers at the start will probably not be I/O bound, but a problem that reads and writes incrementally to a number of different files (such as a compiler) is likely to be I/O bound. If the operating system provides a facility such as the UNIX ps command that can tell you the amount of CPU time used by a program, you can compare this with the total time to complete execution of the program. This is, of course, most meaningful on a system where you are the only user.

Environments:
1. Batch (huge jobs e.g., periodic analytic tasks)
2. Interactive (e.g., user facing)
3. Real time (e.g., need to meet deadlines)

Scheduler objectives
- Fairness
  – all processes get fair share of the CPU
- Throughput
  – number of processes that complete per unit time, the higher the better
- Turnaround Time
  – time from process start to its completion
- Response Time
  – time from when a request was first submitted until first response is produced Algorithm

Turnaround time calculation:



What is the average turnaround time of (a)? (b)?

(a) (8 + 12 + 16 + 20)/4 = 14

(b) (4 + 8 + 12 + 20)/4 = 11

Algorithm categories:
- Non-preemptive
  – Process runs until it finishes or blocks
- Preemptive
  – Process can be suspended after some time interval (indicated by a clock interrupt)

Process/context switch
Context switching takes time (expensive) involves saving and reloading process states:
- saving and loading registers and memory maps (program counter, stack pointer etc)
- updating various memory tables and lists

Batch system algoritms:
- First-Come First-Served
- Shortest Process First
- Shortest Remaining Time

Next Interactive system algorithms:
- Round-Robin Scheduling
- Priority Scheduling
- Multiple Queues
- Guaranteed Scheduling
- Lottery Scheduling
- Fair-Share Scheduling

First-come first-served:
Advantage: simple and fair
Disadvantage: if the late arrived process just have little thing to finish will wait for long time, may result longer turnaround time. E.g. a CPU bound process comes later than an I/O bound process

Shortest job first:
Advantage: the turnaround time is less.
Cons: the remaining time is hard to estimate

Round-Robin scheduling:
The batch system algorithm assumes we know how long the process gonna take
But it's hard to estimate
The scheduler sets a timer to generate an interrupt after a particular amount of time, *quantum*
The amount of time that a process can run without interruption
每一个都给一个quantum的时间去运行, quantum at least longer than context switch, it's highly system dependent

Priority scheduling
- Important jobs always run before less important jobs.
- For example: system processes are more important than user processes in some cases.
- Static vs. dynamic prioritization
- Potential problem: starvation; possible that low priority jobs will never execute, if more important jobs continually arrive.
- Combine with round-robin algorithms, create queues with different priority and run with different level of quantum from highest priority queue to lowest ones.
- Cons: the low priority may never get chance to run if higher priority process continuous comes

Multiprocessors:
- per CPU or shared queue of processes
- load balancing

Week03

Memory requirements: fast, cheap, large, non-volatile
Reality: Different types of memory with different properties; Higher speed, smaller capacity, greater cost (refer to the memory hierarchy in Week 1)

Goals of memory management:
- The processes may want more memory than they were given, then the memory management creates an illusion that all the processes have enough memory they need in the fastest hierarchy.
- Make sure the least fluently accessed allocations of process are in the slowest hierarchy.

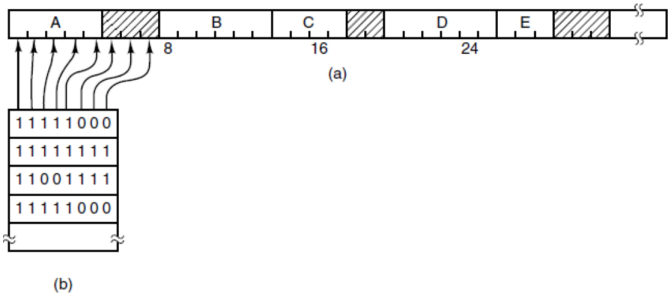
Functionality of memory manager:
- to allocate memory to processes when they require it
- to deallocate when finished
- to protect memory against unauthorized accesses
- to simulate the appearance of a bigger main memory (RAM) by moving data automatically between main memory and disk
- to keep track of which parts of memory are free and which parts are allocated (and to which process)

If there is no memory abstraction:
All the user program can access all physical memory: cause **security issue**, may goes into memory where it shouldn't

Base register specify the beginning of the address space of the process.
Limit register store the maximum size of the address given to a process
If a process is accessing an address, the address will be compared with the base register and limit register. If the address is outside of given addresses, it cannot be access by the process.
Advantage: simplicity, since only two values need to be maintained.
Disadvantage: all address space of the process needs to be in memory in a consecutive block.

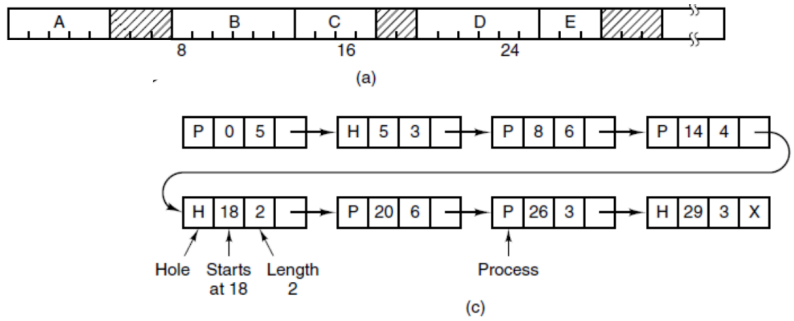
Swapping:Memory allocation changes as processes come into memory and leave it.

Memory Management with Bitmap



(a) A part of memory with five processes and three holes.The **tickmarks** show the memory allocation units. The shaded regions (0 in the bitmap) are free
(b) The corresponding bitmap

Memory Management with Bitmap



(a) A part of memory with five processes and three holes.The **tickmarks** show the memory allocation units. The shaded regions (0 in the bitmap) are free
(c) The same information as a linked-list

Memory Management with linked lists

Four neighbor combinations for the terminating process, *X*.

Memory management algorithms: first fit, next fit, best fit, worst fit, quick fit

First fit: Find the first available memory with space more than or equal to the size of memory needed. The operating system doesn't search for appropriate partition but just allocate the job to the nearest memory partition available with sufficient size.
Advantages: It is fast in processing. As the processor allocates the nearest available memory partition to the job, it is very fast in execution.
Disadvantages: It wastes a lot of memory. The processor ignores if the size of partition allocated to the job is very large as compared to the size of job or not. It just allocates the memory. As a result, a lot of memory is wasted and many jobs may not get space in the memory, and would have to wait for another job to complete.

Next fit is a modified version of 'first fit'. It begins as the first fit to find a free partition but when called next time it starts searching from where it left off, not from the beginning. This policy makes use of a roving pointer. The pointer moves along the memory chain to search for a next fit. This helps in, to avoid the usage of memory always from the head (beginning) of the free block chain.
Advantages:
- First fit tends to allocate memory parts at the beginning of the memory, which may lead to more internal fragments at the beginning. Next fit tries to address this problem by starting the search for the free portion of parts not from the start of the memory, but from where it ends last time.
- Next fit is a very fast searching algorithm and is also comparatively faster than First Fit and Best Fit Memory Management Algorithms.

Best fit: The allocator places a process in the smallest block of unallocated memory in which it will fit.
Advantage: Memory utilization is much better than first fit as it searches the smallest free partition first available.
Disadvantage: It is slower and may even tend to fill up memory with tiny useless holes

Worst fit: The memory manager places a process in the largest block of unallocated memory available. The idea is that this placement will create the largest hold after the allocations, thus increasing the possibility that, compared to best fit, another process can use the remaining space.
Advantage: Reduces the rate of production of small gaps
Disadvantage: If a process requiring larger memory arrives at a later stage then it cannot be accommodated as the largest hole is already split and occupied

Quick Fit relies on lazy coalescing. If you free a block of size n, you are very likely, in the near future, to allocate a block of size n, because what you really did was free an object of type A (sizeof(A) == n) and you are therefore fairly likely in the near future to allocate a new object of type A. So instead of returning the block to the general heap, and possibly coalescing it with nearby blocks, you just hang onto it, keeping it in a list of blocks of size n. The next time you allocate an object of size n, the allocator looks on the free list[n] to see if there are any blocks laying aroud to be reallocated, and if one is, your allocation is essentially instantaneous. Only if the list is empty do you revert to one of the slower algorithms. In QuickFit, coalescing is done before you decide to ask the operating system for more storage. First you run a coalesce pass, then see if you now have a big enough block. If you don't, then you get more space from the system.

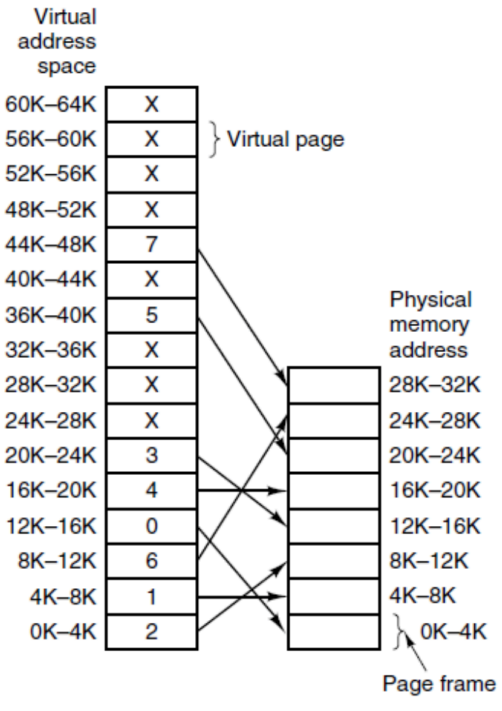
Fragment: the space between two memories. If we want to use it, we may need to cut a process into several parts and need more base and limit registers which make the system too complex.

Virtual memory:
- There is a need to run programs that are too large to fit in memory
- Observations:
  – all the code and data of a program does not have to be in main memory when the program is running
  – this code and data does not have to be stored in contiguous locations.
- Virtual memory systems allow programs to continue making both assumptions: each program has its own address space, broken up into chunks called pages

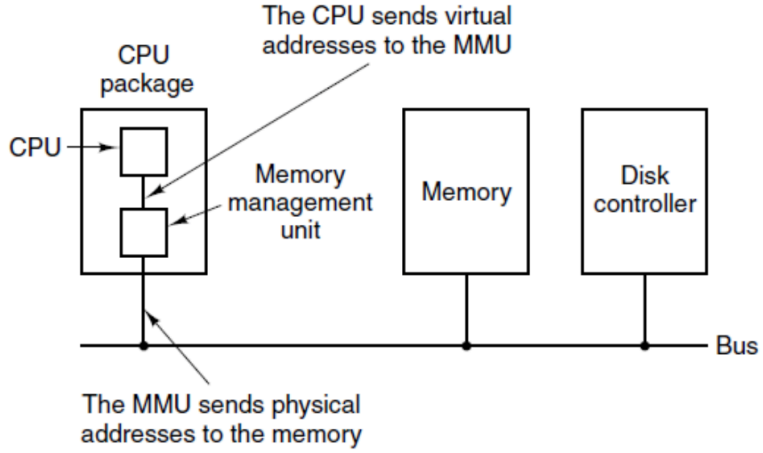
Virtual address spaces
- The virtual address space of a machine is the set of addresses that programs on that machine may generate.
- Each process has its own virtual address space.
- The virtual address space may be bigger than its physical address space.
- virtual address = virtual page number * page size + offset

Paging:
- A paged system divides both virtual and physical address spaces into fixed size pages.
- *Virtual page can be mapped onto any physical page.* As the job of a physical page is to hold a virtual page, physical pages are also called **page frames**.
- During the lifetime of a process, pages in its virtual address space all start out on disk, and may move between disk and memory any number of times.
- When a virtual page is moved to disk and back again, it may be put in a different page frame than before.

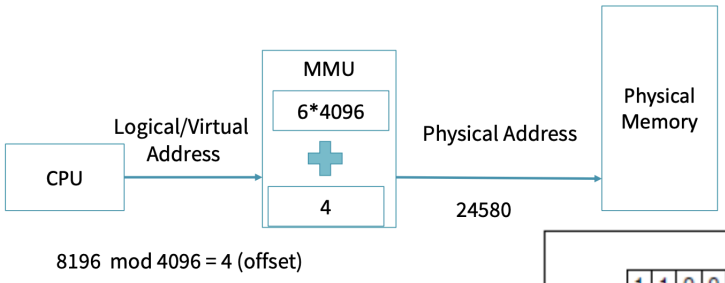
The relation between virtual addresses and physical memory addresses is given by the page table. Every page begins on a multiple of 4096 and ends 4095 addresses higher, so 4K–8K really means 4096–8191 and 8K to 12K means 8192–12287



Memory Management Unit (MMU)

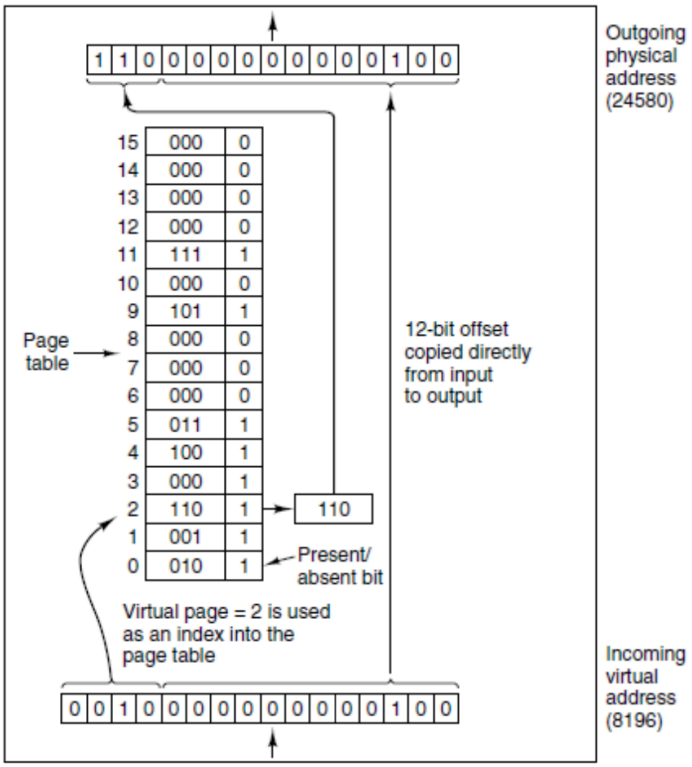


mapping to a physical address using MMU
The mapping is generated by MMU. All actual memory access is done with physical address value.Which physical address does virtual address 8196? (From picture in previous page)
Virtual page 2 mapped to physical page 6

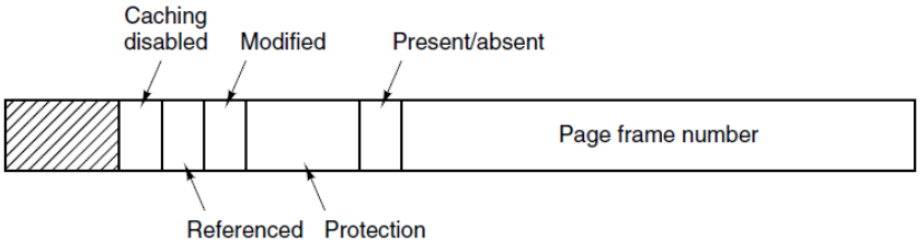


Page table maps virtual addresses to physical addresses

16 4-KB pages



Operations of paging:



- The MMU must check that the selected page table entry has the **present bit** set to one (true) and that the permissions permit the requested memory access.
- If both conditions are met, it will construct the physical address using the physical page number field of the PTE (page table entries). (If not: exception:)
- It will then set the referenced bit, and if the access was a write, it will also set the modified bit.

A **page fault** (sometimes called #PF, PF or hard fault) is a type of exception raised by computer hardware when a running program accesses a memory page that is not currently mapped by the memory management unit (MMU) into the virtual address space of a process

Speeding up paging: Major issues faced:
1. The mapping from virtual address to physical address must be fast.
2. If the virtual address space is large, the page table will be large.

Translation Lookaside Buffer (TLB)

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

- Without optimization, each memory reference by the program in a paging system would require two memory accesses: one to access the PTE and one to access the data.
- To avoid this performance penalty, paged computers have a translation lookaside buffer (TLB) in the MMU.
- The TLB serves as a cache, holding copies of **recently accessed page table entries** (PTEs).
- The TLB is implemented using associative circuitry that is much faster than main memory.
- In binary, division by powers of two is trivial, but division by other numbers is very difficult, taking more than a dozen CPU cycles even in good implementations. However, you cannot search the TLB for a virtual page number until it has been computed. Therefore non-power-of-two page sizes would slow down every memory reference by a large factor.

Related Question: You are given the following data about a virtual memory system:
(a) The TLB can hold 1024 entries and can be accessed in 1 clock cycle (1 nsec).
(b) A page table entry can be found in 100 clock cycles or 100 nsec.
(c) The average page replacement time is 6 msec.
If page references are handled by the TLB 99% of the time, and only 0.01% lead to a page fault, what is the effective address-translation time?
Note that a page table entry can be found either in the TLB or in the page table. Hence, 100 clock cycles accounts for the total time of the event of finding a PTE. Similarly, the page replacement event means that the address is translated due to a page fault. Since a page fault occurs due to the page table entry not being found, it also includes the TLB and page table lookup times.
The chance of a hit is 0.99 for the TLB, 0.0099 for the page and 0.0001 for a page fault (i.e., only 1 in 10,000 references will a page fault). The effective address translation time in nsec is
$0.99\times1+0.0099\times100+0.0001\times(6\times10^6) \approx 602$ clock cycles.
that the effective address translation time is quite high because it is dominated by the page replacement time even when page faults only occur once in 10,000 references.

Page fault handling
- If the page fault is caused by the permissions being violated, the page fault handler will usually terminate the process.
- Otherwise, the OS must:
  - suspend the process,
  - free up a page frame (if there are none available now),
  - load the required virtual page from swap space into a free page frame,
  - cause the MMU to map the virtual page onto the physical page, and
  - restart the process at the same instruction

Memory Replacement Algorithms
- Decide what to remove to make space
  – Which process's page to remove? Local (when OS only evict the pages that are allocated with processes) vs. global (can evict something from another processes memory)
  – Which page to remove?

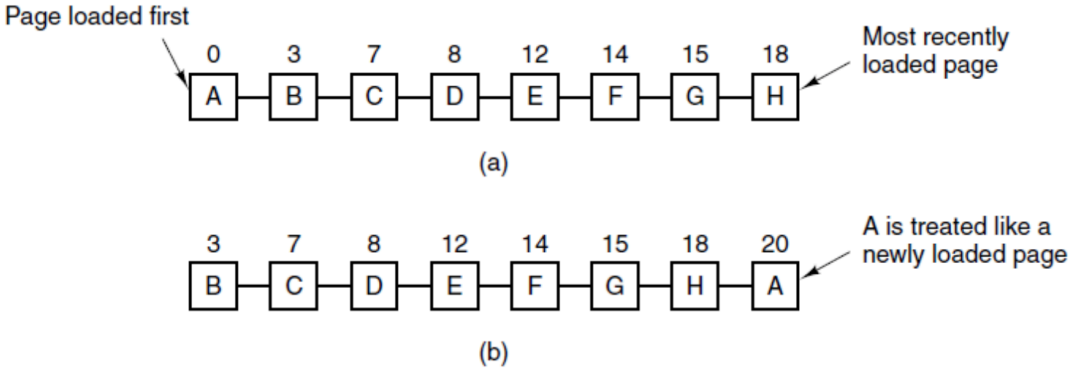
For the page decided to move, we need to know if any data it stores has been modified
- Discard the page, if not modified
- Write to disk, if modified

Page Replacement Algorithms
- Optimal algorithm
- Not recently used algorithm
- First-in, first-out (FIFO) algorithm
- Second-chance algorithm
- Clock algorithm
- Least recently used (LRU) algorithm
- Working set algorithm
- WSClock algorithm

Not Recently Used Algorithm
- At page fault, system inspects pages
- Categories of pages based on the current values of their R and M bits:

Class 0: not referenced, not modified.
Class 1: not referenced, modified.
Class 2: referenced, not modified.
Class 3: referenced, modified.

Second-Chance Algorithm



Operation of second chance. (a) Pages sorted in FIFO order. (b) Page list if a page fault occurs at time 20 and A has its R bit set. The numbers above the pages are their load times.
It checks to see if its referenced bit is set. If it is not set, the page is swapped out. Otherwise, the referenced bit is cleared, the page is inserted at the back of the queue (as if it were a new page) and this process is repeated. This can also be thought of as a circular queue. If all the pages have their referenced bit set, on the second encounter of the first page in the list, that page will be swapped out, as it now has its referenced bit cleared. If all the pages have their reference bit cleared, then second chance algorithm degenerates into pure FIFO.

Temporal and spatial locality
- Working set: pages currently used by a process
- If the working set is too small, thrashing occurs: frequent page faults
- Locality: Data access is not uniform throughout memory
- Access may be clustered around certain areas/time
  – Consecutive locations of code
  – Within a data structure

Related question: Working set page replacement algorithm will first try to evict pages that are not in the working set before evicting pages in the working set. Under what type of access locality does a working set algorithm perform best (i.e., minimise number of page faults) and why?
Temporal locality: If the program tends to access a set of k pages more frequently than others then keeping these pages in the working set will minimise the number of page faults on average. Spatial locality can also help if the accesses that the program is making are to different parts of the same page. Note that pages in a working set of a program can change over the course of program execution.

Cryptography application
Secure communication
Authentication and verification
Integrity

Encryption
– Hiding data (plain text) from everyone except those holding the decryption key
– Output is a cipher text
Decryption
– Recovering the original data from the cipher text using the key
– Output is a plain text

Modern **cryptography** is based on mathematics
- Problems that are considered to be hard, or are well studied
  - Factorizing the product of 2 large primes (RSA)
  - Solving discrete logs (ElGamal)
  - AES – substitution-permutation network

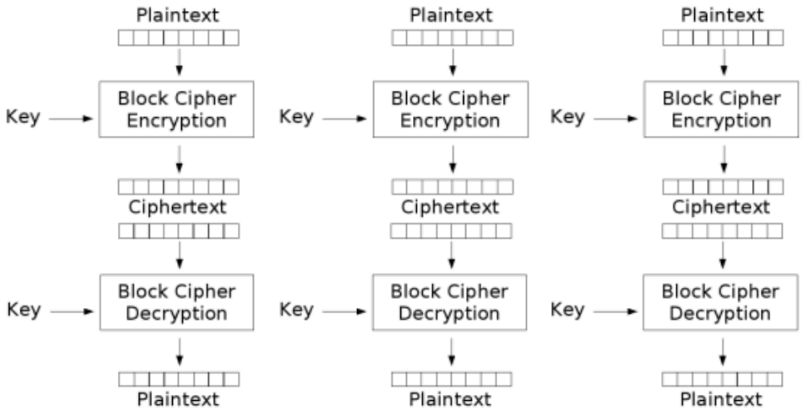
Information theoretic crypto: one time pad
– m ⊕ x
– where m is a message, x is a secret key, ⊕ is XOR, so it's seen as symmetric

Security of cryptography
Cryptography is not absolute
– there is no perfect security
– Always susceptible to brute force attack
- Trying every possible key value
– The challenge is to make the brute force attack take so long as to be infeasible to perform within the useful lifetime of the data

Symmetric Cryptography
– Same key is used for encryption and decryption
- Modern example is **AES** (Advanced Encryption Standard)
- CipherText := Encrypt(SecretKey, Message)
- Message := Decrypt(SecretKey, CipherText)
- If you want to use it to share a message with someone you have to find some way of securely exchanging the secret key
- Useful for keeping your own data secure
  – Full disk encryption

- AES breaks data into blocks and encrypts each block
- AES has different modes of operation
  – ECB, CBC, CTR, etc.
  – Determines how each block is treated, and / or linked
- AES-NI instruction set extension on Intel

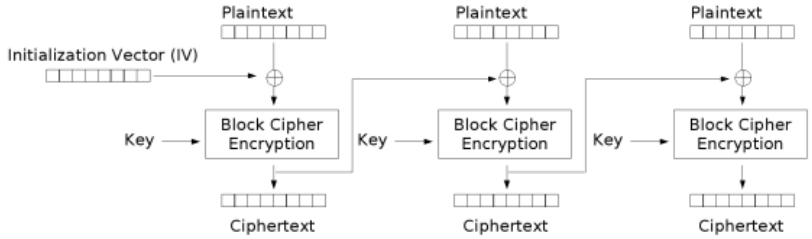
ECB - Electronic Codebook
- Simple, parallelisable
- Does not provide diffusion, may not even provide confidentiality
- Generally should never be used
- The problem is that the same plaintext encrypts to the same cipher text
- Repeated patterns will be evident in the output, result a lower confidentiality



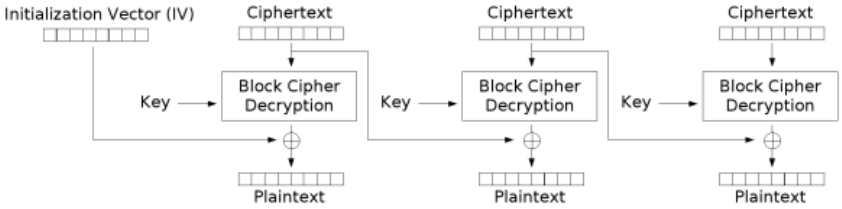
Electronic Codebook (ECB) mode decryption

Probabilistic encryption
- Secure notion of encryption
- If you are encrypting the same message for multiple times, you are very likely to get different cipher text
- Enc(SK, ..., “msg”) != Enc(SK, ..., “msg”)

CBC
– Cipher Block Chaining
- IV (initialization vector) – not secret, but must be **random** and **not reused**
- Encryption must be done sequentially
- Decryption can be done in parallel
- Loss or corruption of a ciphertext block or IV affects correct decryption of at most **two** blocks



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Asymmetric Cryptography
- Consists of two related keys
  – PublicKey – as per the name, can be made public
  – Private/SecretKey – must be kept secret by the owner/user
- CipherText := Encrypt(PublicKey, message)
- message := Decrypt(PrivateKey, Cipher Text)
- Asymmetric Cryptography (Public Key Crypto)
  – Two keys, encrypt with one, decrypt with the other
- More commonly called public key cryptography
- At the heart of modern security – **Digital signatures**
  – TLS (Transport Layer Security)
  – Secure Messaging
  – End-to-end Encryption (WhatsApp, Signal, etc.)
- No need to meet or securely exchange any keys

Asymmetric & Symmetric
- Asymmetric Cryptography is slower than Symmetric
- It often isn’t suitable for encrypting large amounts of data or even multiple blocks ,Asymmetric cryptography is slower than symmetric (usually due to two reasons: larger key size and operations themselves, e.g., exponentiation of large numbers for asymmetric cryptography and bit manipulation in symmetric). It often is not suitable for encrypting large amounts of data or even multiple blocks.
- Often used together with Symmetric Cryptography as a way of exchanging a joint secret key

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

Public Key Signatures

- Provides a **link** between a **key holder** and a **message/document/file**
- Importantly, is considered to provide non-repudiation
  – Cannot deny having signed the document
- Provides **integrity** (against tampering)
- Generate **SignKey** (kept **private**) and **VerifKey** (**Public**)
- s := Sign(SignKey, m)
- Anyone can check if verification succeeds: – Verify(VerifyKey, s, m)?

What happens with **large documents**?

– Use a Cryptographic Hash Function

- Takes a near arbitrary length input and outputs a fixed length digest
- SHA256, SHA512, (MD5, SHA1 – now deprecated)
  – Sign the Hash Digest – which can be recalculated by the verifier: s' := Sign(SignKey, h) where h = Hash(m)

Cryptographic Hashing

- Hash function H is a lossy compression function
  – Collision:H(x)=H(x')for some inputs x≠x'
- H(x) should look "random"
  – Every bit (almost) equally likely to be 0 or 1
- A cryptographic hash function must have certain properties:
  – **Collision resistance**: hard to find x ≠ x' such that h(x)=h(x')
- Hashing vs. encryption
- Applications: password storage(?), software **integrity**

Key Sizes

- Recall our security is based on a computationally hard problem, taking an infeasibly long time to solve using brute force
- As such, our security parameter, which determines the degree of security we achieve, is the key size
  – Normally expressed in bits
- Different key lengths are required for symmetric and asymmetric
  – Reflects the different mathematically hard problems they are based on
  – Asymmetric is generally longer than Symmetric, current minimums:
  - RSA – 2048 bits (3072*), AES – 128 bits
  - Previously much shorter – 512 bits for RSA in the 90's, 56 bits for DES

Randomness

- Cryptography is dependent on randomness
  – Key Generation
  – IV generation for ciphers
  – Padding
- Pseudorandom generators
- Randomness must never be reused or discoverable
- Generating good randomness is hard
  – Must be unpredictable
  – Dependent on good OS implementations
- Compromising randomness generation is a known attack
  – NSA Dual EC DRBG
  – Juniper Networks

Week04

Secure Communication
Confidentiality: when someone observe something, they cannot extract that the message is
Authentication: if you are talking to someone, you are sure you are talking to the right person
Integrity:how can you sure about the messages hasn't been tampered with and the data hasn't been changed
Adversary controls Wi-Fi, DNS, routers, can create its own websites, can listen to any packet, modify packets in transit, inject its own packets into the network
Objective is to provide **secure private** communication between two end-points, with **integrity checks** to ensure data does not change in transit, and **authentication** to establish identities of one or both of the end-points.

2 problems for 'key change' from last week (use asymmetric cryptography to exchange secret key then use the secret key to send messages):
1. How does Alice know ciphertext has not been modified?
2. How does Alice know PK_B is Bob's public key?

CBC – Cipher Block Chaining Tampering I (previous lecture)
Attacker can: reorder ciphertext or flip bits
Every possible ciphertext corresponds to some valid plaintext

Hashing should not be used for authentication method:
- we cannot hash cipher text because it's public, and the intermediate person can also get the cipher text, so it's useless
- If we hash the same message twice, the intermediate person will receive two same hash results, which may be extract by probability — low confidentiality

Message Authentication Code (MAC)
- Detect if the message has been tampered with
- s: MAC's secret key; m: message
- t := Mac(s,m); b := Verify(s,m,t) where b is 0/1 indicating successful verification
- Verifies **integrity** of a message using a secret key
- Security: Adversary cannot create (m', t') such that Verify(s,m',t') returns b = 1 for m' it has not seen

CBC-MAC based on AES encryption,
difference from CBC: only sending the last cipher text but not sending cipher text for every block
only works for fix variable messages, be careful with variable length messages because we need to encode the length, as it is easy to extend the cipher text, or just plug and play with CBC
HMAC: Industry standard and used widely in practice
Generate a MAC tag t := Hash ((s ⊕ opad) ‖ Hash ((s ⊕ ipad) ‖ m))
ipad and opad are fixed constants used for padding

Authenticated Encryption
- **Confidentiality** and **integrity** of messages exchanged between Alice and Bob
- General construction: Encrypt-then-Mac:
  – c:=Encrypt(SK,m) Message m, Secret key SK of symmetric encryption
  – t:=Mac(s,c) s: MAC's secret key
- Never use same MAC's secret key and symmetric encryption key
- Verify: if Verify(s,t,c) returns 0, do not decrypt
- Examples: AES-GCM, AES-OCB, AES-CCM

Related Question: In the lectures we saw that authenticated encryption can be achieved by using secure semantic encryption and message authentication code via Encrypt-then-authenticate paradigm: given m the output is set to (c, t) where c := Enc(s1 , m), t := MAC(s2 , c) and s1 and s2 are the secret keys of the corresponding schemes. Consider a different candidate for authenticated encryption where (c,t′) are set as follows: c as above and t′ = MAC(s2, m). Does this scheme provide secure authenticated encryption? If it does, show that if you can break it, then you can break either the underlying encryption or the underlying MAC scheme. If it does not, provide a counter example.

MACs do not aim to provide confidentiality of messages they authenticate. For example consider MAC2(s2,m) that returns as its tag a tuple (t,m) where t := MAC(s2 , m). Here, MAC2 is a valid message authentication code, as breaking it would result in breaking MAC, which is assumed to be secure.
Now consider using MAC2 together with a secure encryption Enc as suggested in the question. This would set (c, t′) to a tuple (c, (t, m)), as a result, trivially violating confidentiality requirement of authenticated en- cryption.

Diffie-Hellman Key Exchange
Introduction
- Fundamental to protocols such as HTTPS, Secure Shell (SSH), Internet Protocol Security (IPsec), Simple Mail Transfer Protocol Secure (SMTPS), and other protocols that rely on Transport Layer Security (TLS).
- Agree on a shared key
- Provides perfect forward secrecy: exposure of long term keys does not compromise security of past sessions
- Sends information in a way that allows both parties to calculate a shared key without having to ever explicitly communicate the shared key

Steps
- Generate some public information:
  – A large prime p
  – A generator g (primitive root modulo p)
- Alice picks a random value x and computes $X=g^x \bmod p$
  – Sends X to Bob
- Bob picks a random value y and computes $Y=g^y \bmod p$
  – Send Y to Alice
- Alice calculates the secret $s=Y^x \bmod p = g^{yx} \bmod p$
- Bob calculates the secret $s=X^y \bmod p = g^{xy} \bmod p$
- $g^{yx} \bmod p = g^{xy} \bmod p$

Security consideration
- At the end of the process we have a shared secret, both parties compute the secret key $s=g^{xy}$, the component parts of which we have never openly communicated,
- Solving the discrete log (in the particular group we operate) is considered a hard problem
- As such, it is considered infeasible to recover the x from g^x
- Provided the two parties discard their secrets, even if one of them loses their private key, it will not allow past communication to be decrypted
- Secret key should look indistinguishable from random
- DH key exchange relies on Decisional DH

Human-in-the-middle Attack: aka Man-in-the-Middle (MITM) attack
Attack: network traffic between Alice and Bob goes via Eve who:
- Trick Alice into thinking it is communicating with Bob and vice versa by modifying the messages (impersonation)
- Silently snoops on the messages sent between Alice and Bob (eavesdropping)
None of the attack can be against by DH secure, because sending g^x, g^y can be token by man-in-middle

Purpose of certificates

- Securely *associate identities with cryptographic public keys*
  – Name, domain, organization, etc.
  – e.g., for SSL/TLS and HTTPS for authenticated and encrypted web browsing, code signing, client authentications
- The certificate itself is signed (Certificate Signature) – often by a third party

Example

PK is public key, SK is signing key here

- certificate = (PK_{Alice}, Alice, ...)
- Signing: s = Sign (SK_{Issuer}, certificate), certificate contains d and other details such as issuer, algorithms, validity date
- Certificate: Issuer, Signature s, certificate = (Issuer, PK_{Alice}, Alice, ...)
- Verification: Verify (PK_{Issuer}, s, certificate)

Certificate format

X.509 the most common standard format
Consists of: Version Number, Serial Number, Signature Algorithm ID, Issuer Name, Validity Period (not before and not after), Subject name, Subject Public Key Info (Public Key Algorithm and Subject Public Key) , Issuer Unique Identifier (optional) Subject Unique Identifier (optional), Extensions (optional), Certificate Signing Algorithm, Certificate Signature, subject alternative names

Certificate Hierarchies

- The signer is vouching for the identity behind the public/private key pair
- Certificates can be chained: A signs B's Certificate, B signs C's certificate and C runs a website or issues digital signatures
- Trust flows down the hierarchy, if you trust A, then you trust all certificates A signs, and in turn, all certificates signed with certificates signed by A, i.e. trusting A implies trusting C
- Limiting purpose is critical (BasicConstraints - id-ce 19)

Chained Signing & Verification: Example

- PK_A, SK_A, PK_B, SK_B, verification/signing keys of a digital signature
- A signs B's certificate, B signs C's certificate
  – A: s_B = Sign (SK_A, $certficate_B$), $certificate_B$ contains PK_B
  – B: s_C =Sign(SK_B, $certficate_C$), $certficate_C$ contains Charlie's domain address www.foobar
- Alice wants to verify www.foobar is Charlie's using $certficate_C$:
  – Verify(PK_B, s_C , $certficate_C$)
  – Verify(PK_A, s_B , $certficate_B$)

Trust anchors and certificate authorities

- Trust anchors are entities that are explicitly trusted
  – Most commonly found as **root certificates**
- They are the points from which all other trust is derived
- **Certificate Authorities** (CA) are the most common trust anchors
  - Sign certificates for others
  - Root certificates are shipped, pre-loaded, with your Operating System/Browser
  - Root certificate is **self-signed**
  - Sub-CA–intermediate CA that is not a root, but has been signed by a root
  - Cross-signing–Sub-CA signed by multiple root CAs (compatibility/robustness)

Trusting root certificates

- Machine will contain 50+ root certificates
  – issued by governments, companies

Certificate Issuance

- **Domain Validation** (DV)
  - Most common
  - Ties a certificate to a domain and checks the requester has some control over the domain
  - Validation via email/DNS/URL – possible weakness
- **Organization Validation** (OV)
  – Ties a certificate to a domain and a legal entity (e.g., an operating business)
- **Extended Validation** (EV)
  - Establishes legal entity, jurisdiction, and presence of authorized officer (e.g., physical address, phone #)
  - Offline process+expensive

Certificate Issuance - problems

- DV certificates do **not** establish a link between the domain and a real world entity
  – LetsEncrypt has issued 14,000 Certificates containing the word “paypal”
  – WoSign incorrectly issued certificates for GitHub
- Even EV certificates are not immune
  – Symmantec issued an EV certificate for google.com

Certificate Revocation

- Revocation occurs when a certificate is:
  – Mistakenly issued (Let’s Encrypt, February 2020)
  – Private key is compromised
- Performed via Certificate Revocation Lists and OCSP (Online Certificate Status Protocol) [CRLite, Mozilla]
  - by the certification revocation list (a huge list maintained by CA), it’s hard and take too much time, that’s why CA has expiration date to revoke certificate easier
  - (efficient) the web-browser would ask the CA is this certificate been revoked, however the CA may know someone is asking for the certificate, the CA has to be someway online to accept this kind of request
  – so here is **Performance vs privacy**
- When root certificates are revoked all certificates below that become untrusted
  – Example: DigiNotar, CNNIC, WoSign
  – Hence cross-signing root certificates
- Only useful if aware of incorrectly issued certificate

Certificate Transparency (CT)

- Open framework for monitoring and auditing certificates
- Intended to provide a way of monitoring certificates that have been issued
- Detect:
  – mis-issued certificates
  – maliciously acquired certificates
  – rogue CAs
- Uses a cryptographic **append-only** log to record the issuance of certificates
  – Merkle Trees: consistency and membership verification
- Server should send SCT (signed certificate timestamp)
  – SCT is issued by CT log to prove inclusion
- Browser may choose to decline certificates not on CT log
- SCT is an extension in X.509

SSL(Secure Socket Layer)/TLS(Transport Layer Security)

Protocol for secure communication over Internet Transport Layer Security protocol

- Supported by all popular web browsers , web servers, internet commerce sites
- HTTPS: implementation of TLS over HTTP

TLS Basics
- Handshake protocol:
  – Uses public-key cryptography to establish several shared secret keys between the client and the server
  – An initial negotiation between client and server that establishes the parameters of their subsequent interactions within TLS
- Record protocol:
  – Uses the secret keys established in the hand shake protocol to protect confidentiality, integrity, and authenticity of data exchange between the client and the server

Handshake Protocol
Runs between a client and a server
For example, client = Web browser, server = website
Negotiate version of the protocol and the set of cryptographic algorithms to be used:
- Interoperability between different implementations

Authenticate server and client (optional)
- Use digital certificates to learn each other's public keys and verify each other's identity
- Often only the server is authenticated

Use public keys to establish a shared secret

Digital certificate
The fundamental job of a certificate is to bind a public key to the name of a principal (individual, company, etc.). That is, a certificate is used to prove ownership of a public key by a given entity. The certificate will include details about the owner (the 'subject'), information about the public key, and the digital signature of the certifying authority.

ClientHello
Client (in plaintext):
• Protocol version
• Cryptographic algorithms
• random nonce (important, because the nothing is encrypted in this step, and someone may plan attack later on, the random nonce creates a new communication to prevent reuse to interaction)

serverHello
Server (in plaintext) :
• Highest protocol version supported by both the client and the server
• Strongest cryptographic suite selected from those offered by the client

ServerKeyExchange
Server (in plaintext) :
• Highest protocol version supported by both the client and the server
• Strongest cryptographic suite selected from those offered by the client
Server sends its public-key certificate containing either its public key, or its Diffie-Hellman public key g^y (depending on chosen crypto suite)

ClientKeyExchange
The client generates secret key material and sends it to the server encrypted with the server's public key or g^x if DH

TLS Handshake
[over a reliable network connection]
- Client sends ClientHello to server asking for secure connection, listing its supported cipher suites
- Server responds with ServerHello and selects one of the cipher suites presented that it supports, also includes its certificate, and can request the client send its certificate (mutual authentication)
- Client confirms validity of certificate
- Client generates session key
  – Either directly by picking a random key and encrypting it with the public key of the server, or
  – By running the Diffie-Hellman Key Exchange protocol that provides better security
- Handshake concludes and both parties share a key that is then used for encrypting/decrypting messages

Beyond TLS
- Certificates and Certificate Authorities provide authentication
- TLS provides private communication with integrity guarantees
- It's on this basis that the public are told to check for the padlock to know their communication is secure and protected between their web browser and the server, and is with the genuine server

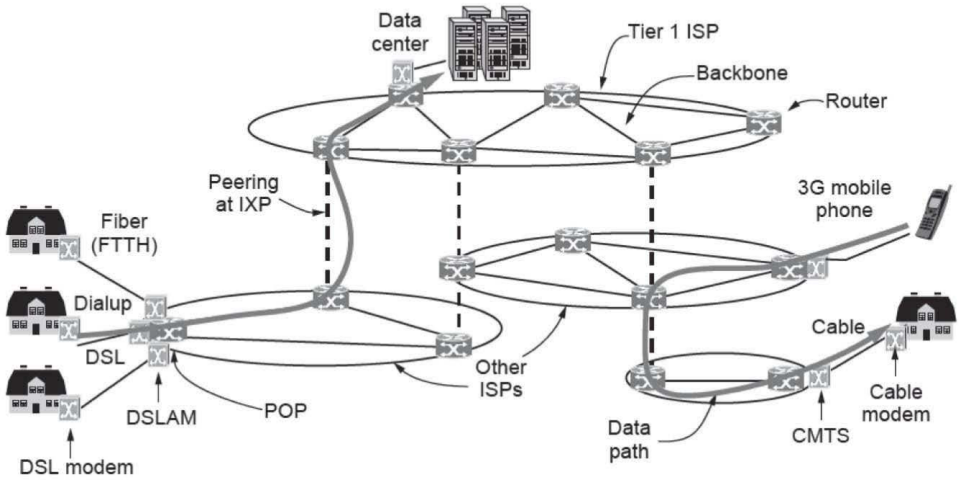
Local TLS Interception
- Create own local Certificate Authority and install certificate into root stores
- Local proxy can intercept TLS connections
  – Acts as a human-in-the-middle (MITM)
  – Dynamically issues fake certificate using local CA (append themselves into the root CAs list)
  – Virtually invisible to the user
- Common practice for Anti-Virus/Web protection software
  – Avast, Kaspersky, ESET
  – Necessary to monitor and analyze TLS traffic
- Not only trusting tools for privacy, also security
- When acting as MITM, the local proxy is performing the certificate validation not the browser
- Too often this has shown to be flawed:
  - Susceptibility to FREAK
  - Not supporting modern features (OCSP stapling)
  - Protocol downgrade attacks
- **Worst case** Superfish/eDellRoot
  - **Same root certificate** installed on all machines – **same private key** that was locally stored - that means if two person use same software are using same public/private keys, they can decrypt each other's cipher text
  - Once compromised anyone could impersonate websites/code signing
  - Superfish also had faulty certificate checking
  - Proxy was Komodia: "Our advanced SSL hijacker SDK is a brand new technology that allows you to access data that was encrypted using SSL and perform on the fly SSL decryption."

Server-side problems
- Web servers increasingly using TLS proxies or Load Balancers e.g. cloudfare
  When cloudflare gets traffic from client how would cloudflare know where to send it if it's encrypted? So it needs to decrypt it. For the purposes of knowing which server to put into also for the load balance purpose. After that, the traffic needs to be re-encrypted.
  – Protects against hacking and DDoS
  – Prevalent on cloud services
- Proxy has certificates for many different sites
- Must trust proxy for privacy
- Fundamentally breaks concept of end-to-end encryption
- Proxies analyze traffic

Cloudbleed (2017)
- Cloudflare – large CDN and TLS proxy service
- Error in HTML parser led to memory being leaked into returned pages (unbalanced tags!)
- This in turn led to that data being crawled by Google/Bing/etc.
- Potential breaches of authentication tokens, passwords, private messages, etc.
- Many big name sites use Cloudflare: Uber, Yelp, OkCupid, FitBit
  – Impact was limited to those using only certain Cloudflare services

Week06

The Internet



- The Internet is composed of the aggregation of many smaller networks – not a single network or under a single point of control.
- Historically, the Internet developed in 3 distinct phases
  – ARPANET (1960's-early1970's)
  – NSFNET(1970's-early1980's)
  – Internet (1980's-present)
- . . . the rise and rise of social media, Web 2.0+ (present)

Brief history

- It is necessary to understand at least some of the history of the internet in order to understand how the current set of standards came to be
- As with most IT projects there were opposing views, designs, and teams. Ultimately, the group that focussed on implementation over standardization won out
- As we shall discuss throughout the lectures, the design decisions taken 40+ years ago have left us with a number of security issues today
- **ARPANET** (1969-1990)
  – Advanced Research Projects Agency Network funded by the Department of Defense (ARPA was part of DoD)
  – Initially started with just 4 sites: UCLA, SRI International, University of California Santa Barbara, University of Utah
  – TCP/IP(Transmission Control Protocol/Internet Protocol) was developed at ARPANET
  – Misconception that it was designed to survive a nuclear attack – robustness was a necessity in a world of unreliable communication links (https://www.internetsociety.org/internet/history-internet/brief-history-internet/)
  - Paul Baran, co-inventor of packet switching, did have that goal
- **International Network Working Group** (1972)
  – Chaired by Vint Cerf (one of the father's of the internet)
  – Proposed a packed switched datagram based network standard
  – Submitted to International Telegraph and Telephone Consultative Committee (CCITT, now ITU-T) for standardization, but rejected
  – Cerf resigned as chairman in 1975; went to work with Bob Kahn at ARPA
  – Cerf & Kahn had already published the foundations of TCP/IP in 1974 in their paper "A Protocol for Packet Network Intercommunication"
  – Cerf & Kahn developed internet protocols in a restricted environment (ARPANET) that they could control

- **OSI Model**
  – Remaining members of INWG regrouped under the International Standards Organization (ISO) working group to design the OSI (open system for internet)
  – This left a bitter rivalry between those at ARPANET and the OSI working group
  – First plenary meeting in1978, published as an international standard in 1984
  – OSI looked liked the dominant player – even the US Department of Defense recommended moving away from TCP/IP to OSI
  – By the late 80's the slow development of the OSI model was leading to increased frustration
- **NSFNet** (National Science Foundation) created in 1986 to provide researchers access to supercomputer sites in the USA
- By the late 80's commercial internet service providers started to appear
- Also in the 80's CERN developed their TCP/IP based network, which would ultimately lead to the creation of the World Wide Web
- TCP/IP became the protocol stack of choice and eventually the OSI became little more than theoretical abstraction

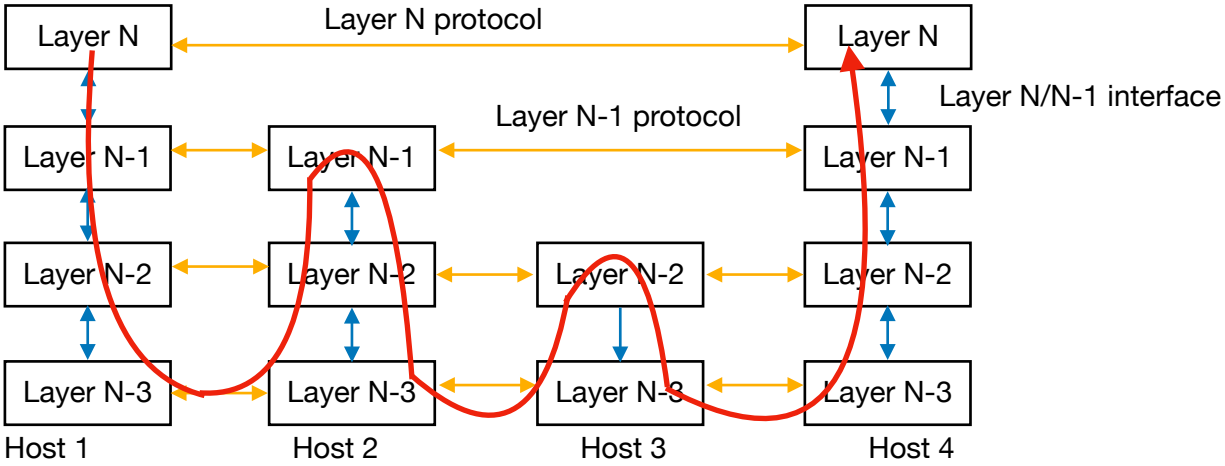
- Many of the underlying protocols were designed without consideration for an adversary on the network
  – Security has been retrofitted, with many insecure protocols still widely used (DNS)
- The rivalry between the two groups led to TCP/IP working groups rejecting OSI concepts out of principle
  – The so called "palace revolt" of 1992 in which the limitations of IPv4 were raised, but the proposed OSI solution was rejected and the leaders of the working group voted out for having suggested it. It wasn't until 1996 that IPv6 was proposed and is still not in widespread use
  – However, one of the important TCP/IP protocols (IS-IS routing) was developed by the OSI and adopted by the TCP/IP community

Related question: List two advantages and two disadvantages of having international standards for network protocols.
One advantage is that if everyone uses the standard, everyone can talk to everyone. Another advantage is that widespread use of any standard will give it economies of scale, as with VLSI chips. A disadvantage is that the political compromises necessary to achieve standardization frequently lead to poor standards. Another disadvantage is that once a standard has been widely adopted, it is difficult to change, even if new and better techniques or methods are discovered. Also, by the time it has been accepted, it may be obsolete.

The outcome
- Two protocol stacks:
  – TCP/IP effectively standardized post implementation
  – OSI standardized pre-implementation, but not widely implemented
- Why do we need a model?
  – Interoperability – Open, ideally not proprietary
  – A reference model to develop and validate against independently
  – Since networks are multi-dimensional, a reference model can serve to simplify the design process.
  – It's engineering best practice to have an abstract reference model, and a reference model and corresponding implementations are always required for validation purposes

Network Models

- Model the network as a stack of layers. Just a model
- Each layer offers services to layers above it.
- Inter-layer exchanges are conducted according to a protocol.
- The lowest layer is called physical layer
- Not all nodes need to have a top layer, but must have a lowest(physical) layer



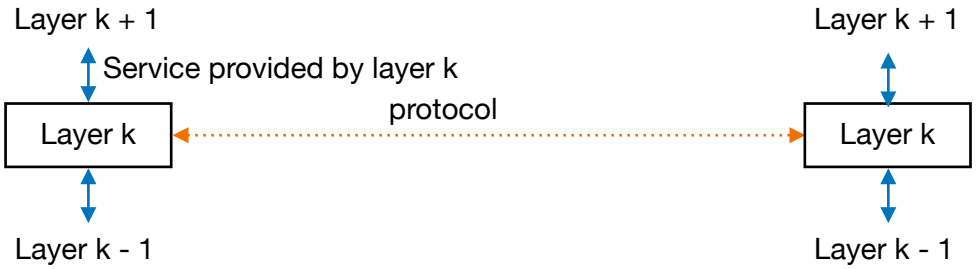
The real information flow to 'peer', the information doesn't need to be sent to the top layer each time, it depends on the layer's decision

Layer Protocols: between peers talking to each other

Layer interface: between different layers

The lowest layer is physical layer which are fiber things

Services to protocols relationship



- Service = set of primitives that a layer provides to a layer above it
  – interfaces between layers
- Protocol = rules which govern the format and meaning of packets that are exchanged by peers within a layer
  – packets sent between peer entities

Related question: Suppose the algorithms used to implement the operations at layer k are changed. Do the implementations of the operations at layers k – 1 and k + 1 need to change accordingly?

No, but it is complicated.

On one hand, the algorithms do not matter as long as layer k still provides the same services to layer k + 1, and still only uses services provided by layer k – 1. The role of layer k – 1 is to provide services to layer k. All the implementations at layer k are based on the services (or API's) from the layer k – 1. As a result, it is not necessary to change the layer k – 1 implementations. Since the role of layer k is to provide services to the layer k + 1, as long as the API's of the services have not been changed, there is no need for the layer k + 1 to change its implementations.

However, the change in algorithm at layer k may change which services are used layer k – 1 and how often they are made, and may change the costs of the services it provides to layer k + 1. As a result, these layers may choose to re-optimise the way they implement their services.

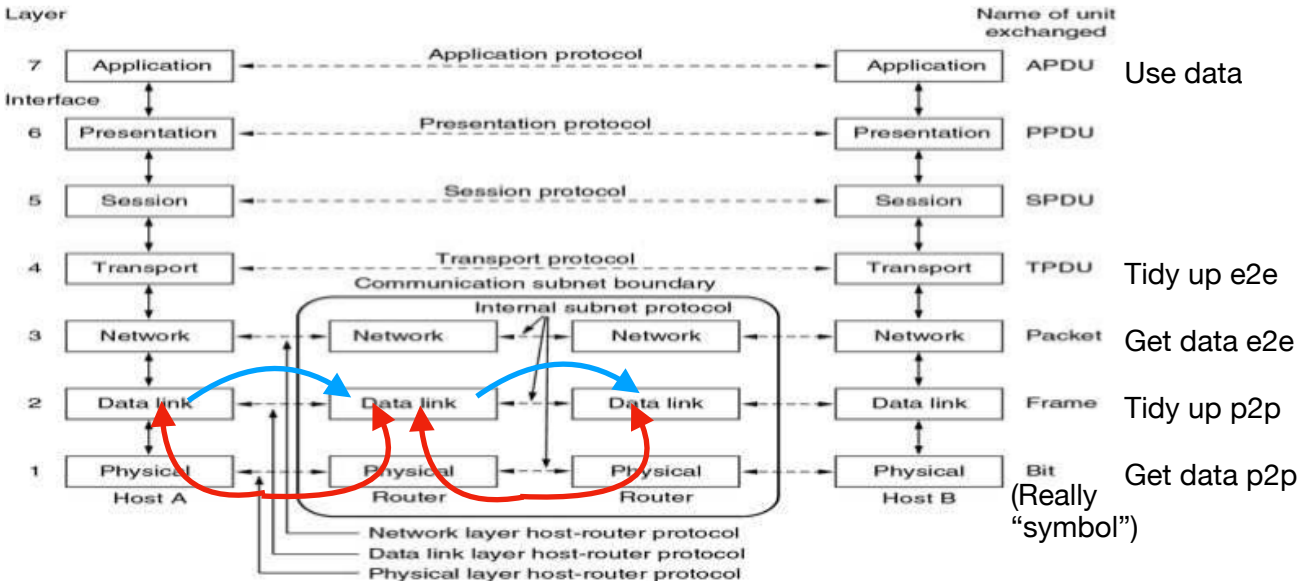
Related Question: Suppose there is a change in the service (set of operations) provided by layer k. How does this impact services at layers k – 1 and k + 1?
There is no impact at layer k – 1, but operations in k + 1 have to be reimplemented.

Connection-oriented and Connectionless services
- Connection Oriented (TCP: Transmission Control Protocol):
  – connect, use (exchange data), disconnect
  – the packet will contain “this is packet ”
  – negotiation inherent in connection setup similar to telephone service
  advantage: if a package got lost, and you find package NO.5, package NO.7, then you know package NO.6 is missing, but cannot discover in UDP as the package are independent
  Disadvantage: the extra delay of setting up a connection can be large depends on the use time
- Connectionless (UDP: User Diagram Protocol):
  – Use message routed through intermediate nodes
  – similar to postal service or text message
  – each packet is independent, so can’t tell packet after that you shouldn’t come after, you should come before sort of things.
- The choice of service type affects the reliability, quality and cost of the service itself.

TCP/IP vs OSI
- The TCP/IP model reflects what happens on the internet
- The OSI model helps reflect the **thought process** that should be followed when designing a network or diagnosing a fault
  – It remains at the core of a number accreditation schemes
- View the OSI model as **idealized**, but with a degree of **flexibility**
  – When we map protocols surrounding TCP/IP to the OSI model, don’t be surprised to see protocols straddle layers or for there to be ambiguity as to which layer a protocol belongs to

Open Systems Interconnection (OSI) reference model
- A layer should be created where a different abstraction is needed.
- Each layer should perform a well defined function. The function of each layer should be chosen with a view toward defining internationally standardized protocols.
- The layer boundaries should be chosen to minimize the information flow across the interfaces.
- The number of layers should be large enough that distinct functions need not to be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.

OSI model



physical layer can be anything that carry information from one to another physically: cable, even pigeon

The other 2~6 layers are abstract

Data link is not a end-to-end protocol, it's a point-to-point protocol

If the data has error — not consistent or missing, it will ask peer layer data link to resend, not asking the physical layer, repeat red track until no errors to complete blue transfer

Repeat point to point with some degree of reliability and separating bit to frame

Network layer is the only layer that works both end-to-end and point-to-point.

Because it's essentially to find a path to the data, through other intermediate nodes.

It's point-to-point to intermediate nodes but end to end to the goal node.

It names all the network entities

Transport layer is purely end to end, similar to data link layer, it's ensure reliable transport from one end to the other, also ensuring not send too fast for the network, just like data link layer ensure not too fast from two point to point systems

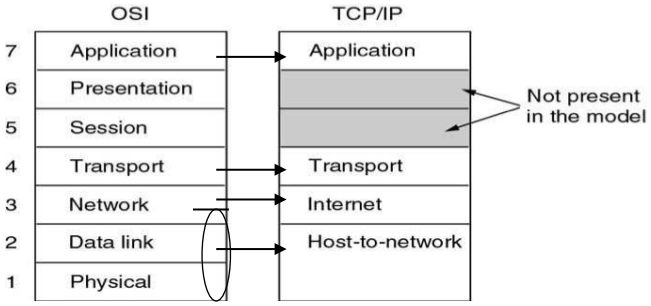
Session layer is not in tcp/ip protocol stack, not widely used

Presentation layer choosing character sets, compressing data

Application layer: includes everything about how we interpret the packages, e.g. http, ftp, file downloads, smtp mail protocol

Related Question: The web protocol HTTP was designed to run over TCP. Google has developed (and uses) a new version of HTTP, HTTP/3, that runs over a new transport layer protocol, QUIC (Quick UDP Internet Connections), which provides new services used by HTTP/3.
(a) Which layers are affected by this change? **(b)** What would have happened if there had been a change to QUIC with no change to HTTP? What assumptions are you making? **(c)** What would have happened if HTTP had been updated to use the new services without switching to QUIC? What assumptions are you making? **(d)** Google controls both the servers and the web browsers (Chrome). How would this make the change easier? What assumptions are you making?
(a) OSI Layers 4, 5, 6 and 7. However, layers 5 and 6 are "empty" in the Internet protocol stack (their functionality is handled at layer 7), and so only transport and application layers are affected.
(b) This would have worked OK, since QIC provides all of the services that TCP does.
(c) This would not have worked, since the changes to HTTP use services that are not provided by TCP.
(d) Because the transport layer and application layer are end-to-end protocols, controlling the end points is enough to implement a new protocol. There is no need to change elements inside the network. However, Chrome runs as a user application, and so Google implemented QUIC in user space, which is possibly less efficient than an implementation in the kernel.

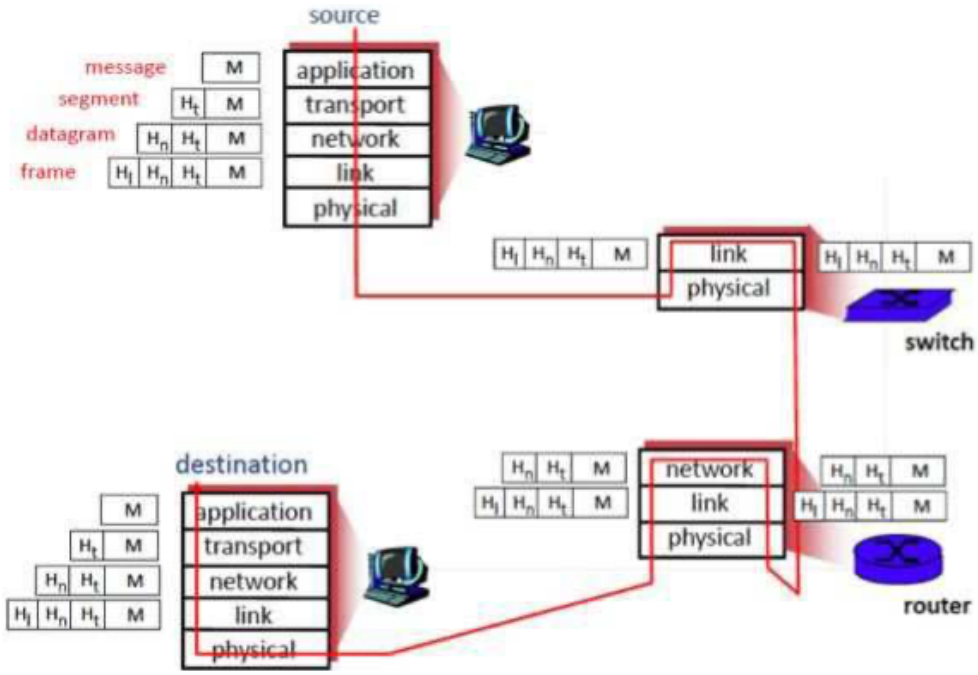
TCP/IP model



Internet layer is top half of network layer
Host-to-network layer often called ‘layer 2’
The bottom half of the network layer is also called ‘layer 2’ sometimes

- TCP/IP – Transmission Control Protocol/Internet Protocol – was designed to be independent of data link and physical layers (Cerf & Kahn, 1974)

Using protocols



Week 07

World Wide Web – A Short History
- Sir Tim Berners-Lee
  - 1984 return to CERN(TCP/IP installed)
  - Saw many online databases with different access mechanisms (FTP, Gopher, ...)
  - 1989 wrote the proposal “a large hypertext database with typed links” (No takers)
  - by 1990, had design and built: HTTP, HTML, httpd, WorldWideWeb (browser)
  - 1992 left for MIT, after CERN IT Head described it as a misallocation of resources
- Hypertext
  – Ted Nelson coined the term in1963
  – Creation and use of linked content
- The vision was that HTTP would be the “glue” between data on different existing protocols
  – e.g., FTP (file transfer protocol) – many files available for download
- GOPHER – distributed database developed at U. Minnesota in 1991
  – Hierarchical file structure
  – More suited for text interfaces – lower network overhead – February 1993, charging for server
- May 1994 first International WWW Conference (at CERN)
- September 1994 W3C formed (DARPA & European Community) – Standardization of web technologies – royalty free
- Browser wars 1994-1998 (Microsoft vs. Netscape)
- 1999 – 2001 .com boom
- 2002+ Ubiquitous web
- Web 2.0 – semantic web, social media

WWW - Components
- Client - typically a browser based access to pages
- Server - daemon based content delivery of pages, continuously running server which waits for requests, and serves the content of request
- URL ≈ Protocol + DNS Name (host name) + file name

Architecture:
The web browser sends a HTTP request to web server, web server returns a HTTP response to the web server

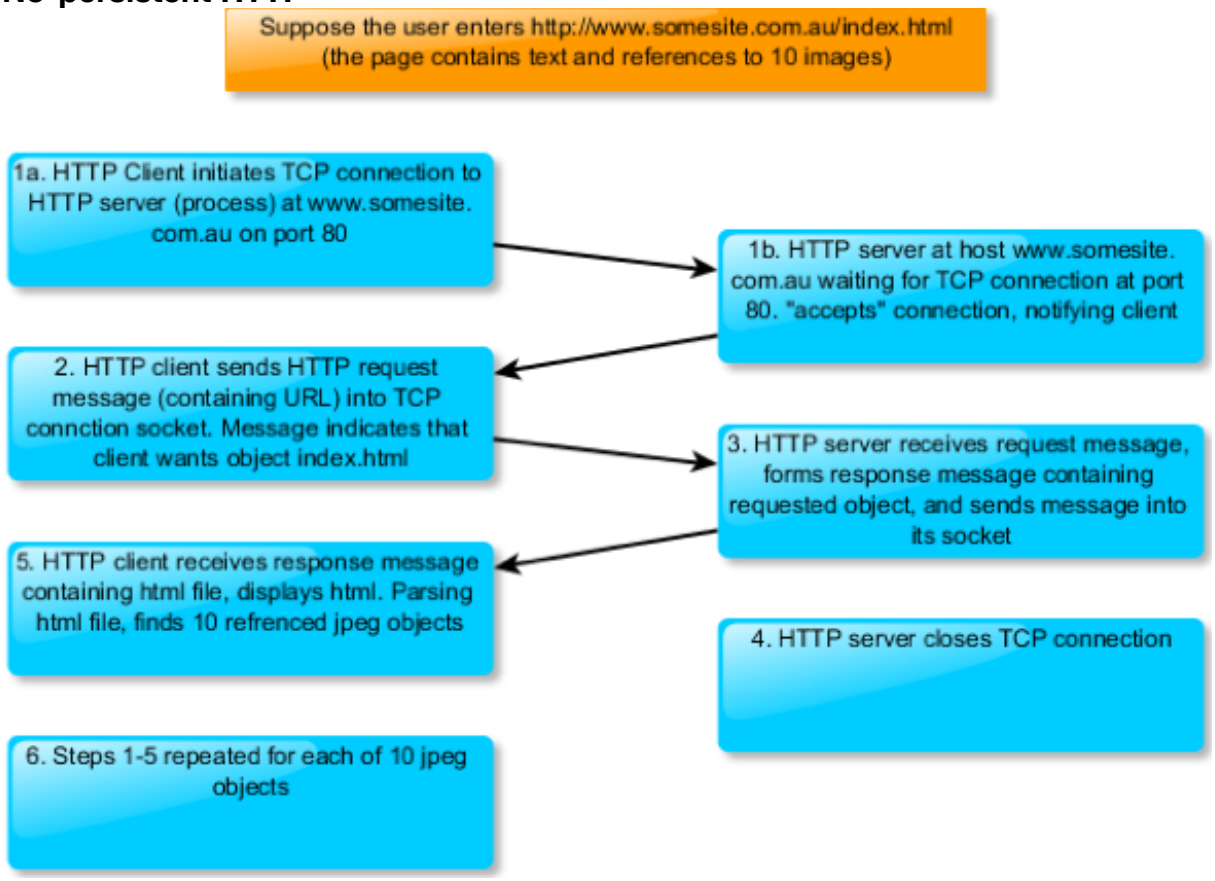
HTTP - Overview
- HyperText Transfer Protocol
  – Defined everything needed for the web
- TCP/IP Model vs OSI Model
  – Application layer (except compression/encoding - Presentation)
- Resources are referenced by URLs

URL/URI
- Uniform Resource Locator
  - Sir Tim called it the “universal resource locator”
  - Defined in original HTTP specification
  - An address for a resource
  - Can be relative “./nextpage.html”or absolute “http://www.google.com”
- Separate specification by W3C in 1998 for URI
  – Uniform Resource Identifier
  – scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]
  – abc://username:password@example.com:123/path/data?key=value#fragid1

HTTP – Protocol Overview

- Overview:
  - Client initiates TCP connection (creates socket) to server, port 80
  - Server accepts TCP connection from client
  - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
  - TCP connection closed
- Connections:
  - HTTP1.0 – single use connection
  - HTTP1.1 – persistent connections, additional headers
  - HTTP/2 – 2015 – Further speed improvements (origins in SPDY)
  - HTTP/3(draft; in use) – Allow more parallelism in data loading (QUIC)

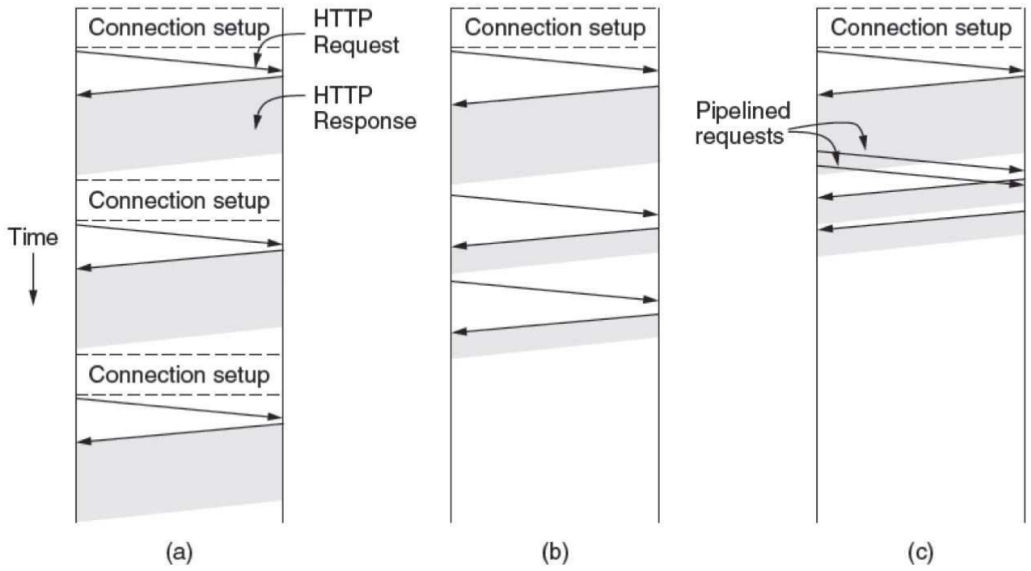
No-persistent HTTP



Persistent vs. Non-persistent

- Non-persistent:
  - requires 2 “response times” (one to initiate TCP connection and one for initial HTTP request) per object + file transmission time
  - OS overhead for each TCP connection
  - browsers often open parallel TCP connections to fetch referenced objects
- Persistent:
  - Multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection.
  - server leaves connection open after sending response
  - subsequent HTTP messages between same client/server sent over open connection
  - client sends requests as soon as it encounters a referenced object, reducing overall response time
- Persistent cannot open parallel connections, so it’s a speed tradeoff

HTTP Request Connection



HTTP with (a) multiple connections and sequential requests.
(b) A persistent connection and sequential requests.
(c) A persistent connection and pipelined requests.

HTTP – Summary of key steps

- Steps that occur when a link is selected:
  - Browser determines the URL
  - Browser asks DNS for the IP address of the server (ResolvingURL)
  - DNS replies
  - The browser makes a TCP connection
  - Sends HTTP request for the page
  - Server sends the page as HTTP response
  - Browser fetches other URLs as needed
  - The browser displays the page (progressively, as content arrives)
  - The TCP connections are released

Request Methods: HTTP

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

- Idempotent – multiple identical requests have same effect
- Safe – Only for information retrieval, should not change state

HTTP Request Example

```
Request line (GET, POST, HEAD) ->  GET /somedir/page.html HTTP/1.1
                                [  Host: www.somesite.com.au
Header lines                    [  user-agent: Mozilla/4.0
                                [  connection: close
                                [  Accept-language: fr
Blank line                         (extra new blank line)
(2LF or 2 CR/LF)
 indicates end of message
```

Related Question: Consider the following string of ASCII characters that were captured by Wireshark when the browser sent an HTTP GET message (i.e., this is the actual content of an HTTP GET message). The characters ⟨cr⟩⟨lf⟩ are carriage return and line-feed characters Answer the following questions, indicating where in the HTTP GET message below you find the answer.

GET /people/index.html HTTP/1.1<cr><lf>
Host: cis.unimelb.edu.au<cr><lf>
Connection: keep-alive<cr><lf>
Cache-Control: max-age=0<cr><lf>
Upgrade-Insecure-Requests: 1<cr><lf>
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.186 Safari/537.36<cr><lf>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
image/webp,image/apng,*/*;q=0.8<cr><lf>
Accept-Encoding: gzip, deflate<cr><lf>
Accept-Language: en-AU,en;q=0.9<cr><lf>
<cr><lf>

(a) What is the URL of the document requested by the browser?
(b) What version of HTTP is the browser running?
(c) Does the browser request a non-persistent or a persistent connection?
(d) What is the IP address of the host on which the browser is running?
(e) What type of browser initiates this message? Why is the browser type needed in an HTTP request message?

(a) The document request was http://cis.unimelb.edu.au/people/index.html, assuming https was not used. The Host : field indicates the server's name and /people/index.html indicates the file name.
(b) The browser is running HTTP version 1.1, as indicated just before the first ⟨cr⟩⟨lf⟩ pair.
(c) The browser is requesting a persistent connection, as indicated by the Connection: keep- alive
(d) This is a trick question. This information is not contained in an HTTP message anywhere. So there is no way to tell this from looking at the exchange of HTTP messages alone. One would need information from the IP datagrams (that carried the TCP segment that carried the HTTP GET request) to answer this question.
(e) Google Chrome, but that isn't obvious because it identifies itself as four different browsers. The browser type information is needed by the server to send different versions of the same object to different types of browsers

HTTP Response Code

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

HTTP - Response

Status line(protocol status code and phrase)		HTTP/1.1 200 OK
Header lines	[	Connection close
	[	Date: Thu, 06 Aug 2009 12:00:15 GMT
	[	Server: Apache/2.2.11 (unix)
	[	Last-modified: Mon, 22 Jun 2009
	[	Content-length: 6821
	[	Content-Type : text/html
	[	
Data, e.g. requested HTML file		<html><head>…

The Date: header line indicates the time and date when the HTTP response was created and sent by the server.

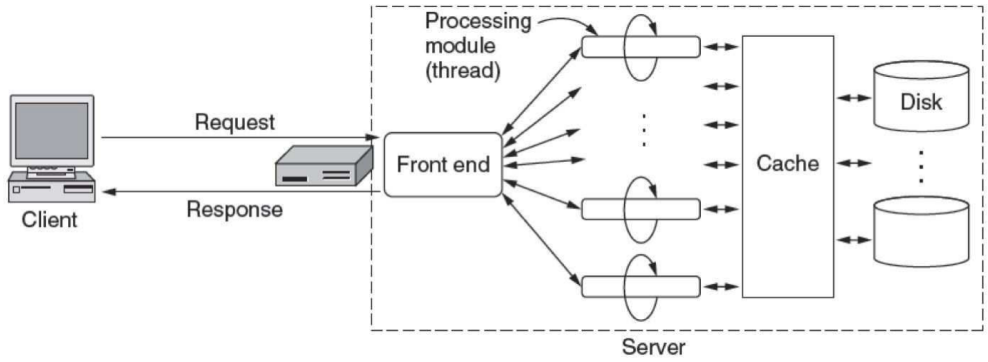
HTTP - headers:
in PPT 2021-S1-WK7-LEC1-HTTP.pdf page 20

Related Question: FTP, the File Transfer Protocol, is one of the oldest protocols still in use in the Internet. Instead of asking for a file in a single request packet, like HTTP does, FTP has a TCP control channel separate from the data connections, and has the client issue a sequence of commands like “USER” (set user credentials)“CWD” (cd), “LIST” (ls), “RETR” (get) and “STOR” (put).
(a) What advantages are there in this approach?
(b) What drawbacks are there in this approach?
(c) What is the use of the USER command?
(a) This is useful when files may be changed to different locations, or files may be added by individual users;
there is no need to add a link to a centrally-owned link farm.
When a human is looking for a file, it is useful to be able to get a directory listing each time the directory is changed, and change response based on that.
(b) A drawback of this approach is that it may require many round-trip-times (RTTs) to get to the desired directory. This is a problem in modern networks, where the propagation delay (time it takes the first bit to get from the sender to the receiver) can be much larger than the serialization delay (the time between sending the first bit and sending the last bit).
(c) This allows a user to log in. This allows data to be password-protected so that users have access to the same files, with the same permissions, they would have access to if they were logged in to the host.

Client side processing
- Plugins/Extensions - integrated software module which executes inside the browser,
  – direct access to online context
- Helper - separate program which can be instantiated by the browser, but can only access local cache of file content
  – application/pdf
  – application/msword

Server side processing – static page
- 5 step process:
  – Accept TCP Connection from client (browser)
  – Identify the file requested
  – Get the specified file from the local storage (disk,RAM,...)
  – Send the file to the client
  – Release the TCP connection

Multi-threaded Web Server

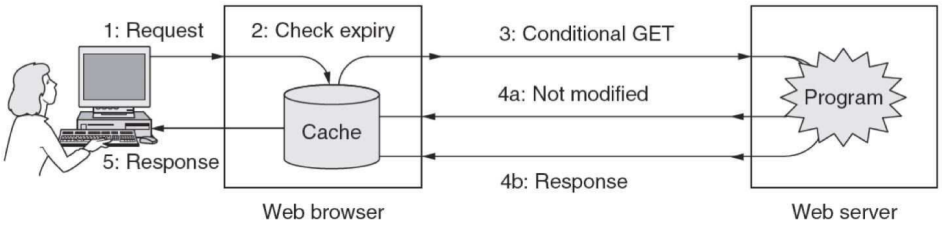


- A multithreaded Web server with a front end and processing modules.

Multi-threaded Web Server – dynamic

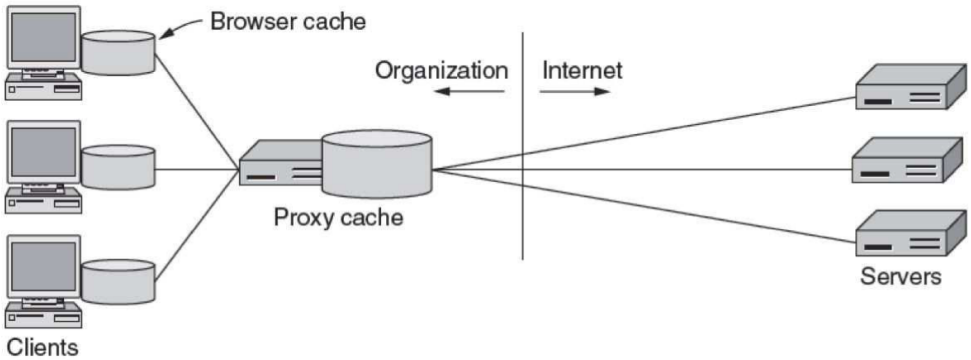
- A processing module performs a series of steps:
  - Resolve name of Web page requested.
  - Perform access control on the Web page.
  - Check the cache.
  - Fetch requested page from disk or run program
  - Determine the rest of the response
  - Return the response to the client.
  - Make an entry in the server log.

Web Cache



Web proxy

- Used for caching, security and IP address sharing
- The browser sends all HTTP requests to the proxy. The proxy returns objects in its cache or else the proxy requests object from origin server, then returns object to client.
- Note: the proxy server acts as both client and server.



Cookies
- The network stores no state about web sessions
- Cookies can place small amount (<4Kb) of information on the users computer and re-use deterministically (RFC 2109)
- Cookies have 5 fields
  – domain, path, content, expiry, security
- How to keep state – maintain state at sender/receiver over multiple transactions; http messages carry "state"
- Questionable mechanism for tracking users (invisibly perhaps) and learning about user behavior
  – e.g.,competitor snooping, undesirable content etc.
- Tracking with cookies is well known
- Tracking companies have expanded beyond simple cookies
  – Plug-in, browser fingerprinting

Related Question: Consider an e-commerce site that wants to keep a purchase record for each of its customers. Describe how this can be done with cookies.
When the user first visits the site, the server creates a unique identification number, creates an entry in its back-end database, and returns this identification number as a cookie number. This cookie number is stored on the user's host and is managed by the browser. During each subsequent visit (and purchase), the browser sends the cookie number back to the site. Thus the site knows when this user (more precisely, this browser) is visiting the site.

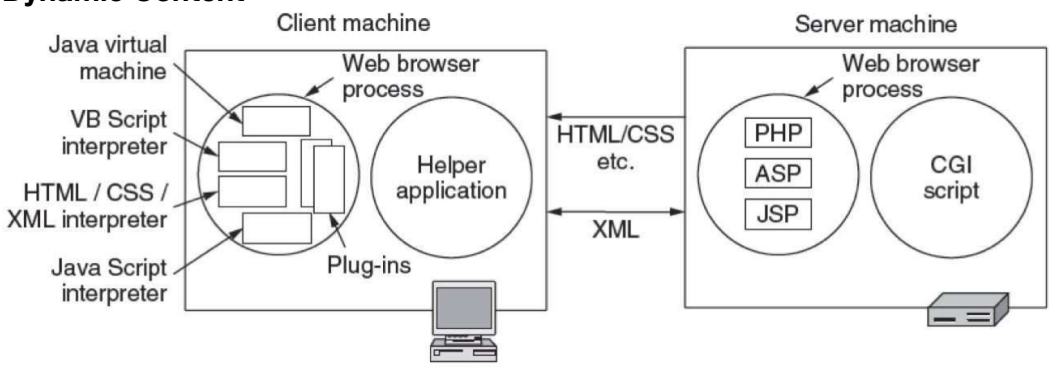
Static web documents
- HTML - Hypertext Markup Language
  – a simple language designed to encode both content and presentational information
  – Plain text encoding, with browser based rendering
  – Restricted to ISO-8859 Latin-1 character set (internationalization not introduced until XHTML with UTF encodings)
- Web Page Components
  – Structural divisions:
  - Head <head> ... </head>
  - Body <body> ... </body>

  – Syntactically Restricted Tag Sets
  – Attributes & Values <img src="pic1.gif" alt="pic1.gif">

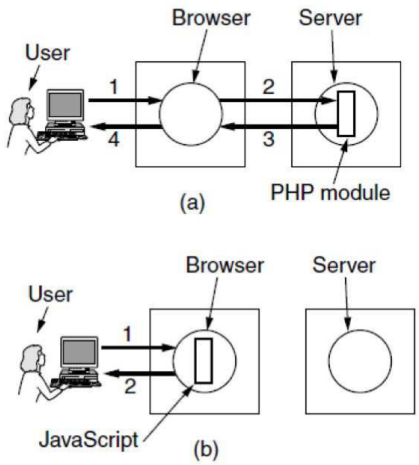
HTML 1.0 only support Hyperlinks, Images, Lists

Beyond HTML
- HTML was originally an instance of SGML
  – standard generalized markup language
- People wanted an HTML-like language to describe data that is not hypertext – but SGML is too general / "heavy"
- XML (Extensible Markup Language) & XSL (Extensible Stylesheet Language)
  – Primary feature: separation of content and presentational markup
  – Stringent validation requirements
- XHTML
  – Essentially an expression of HTML 4.0 as valid XML
  – Major differences to HTML 4.0 are the requirements for conformance, case folding, wellformedness, attribute specification, nesting and embedding, and inclusion of a document type identifier

Dynamic Content



Client-side Scripting



•Technologies for producing interactive web applications include:
•JavaScript
•Java Applets - compiled Java code (platform independent)
•ActiveX - compiled code for Windows
•AJAX
– HTML and CSS: present information as pages.
– DOM: change parts of pages while they are viewed.
– XML: let programs exchange data with the server.
– An asynchronous way to send and retrieve XML data.
– JavaScript as a language to bind all this together

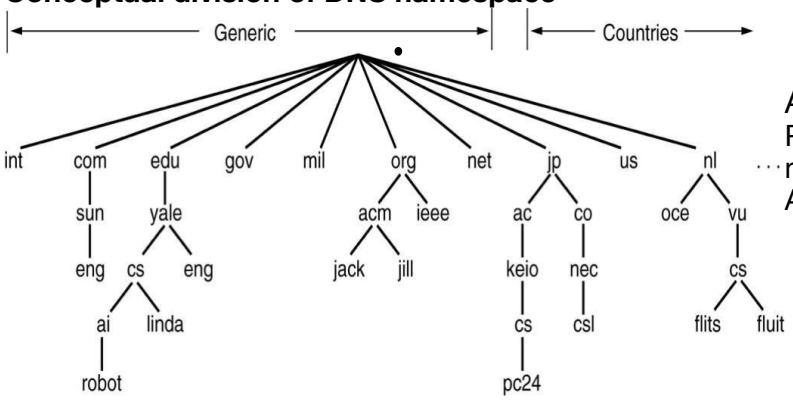
Domain Name System - DNS

- Remember back to URLs
  – We use DNS to resolve the URL to an absolute location
- Briefly mentioned IP addresses before
  – For now, just consider them to be unique numerical identifiers
  - 8.8.8.8 Google Public DNS server
  - 203.2.218.208 – abc.com.au
- Note: Conceptually an IP addresses should uniquely identify a socket/jack (or wireless interface) on a computer
  – Often not the case today
- DNS is essentially the technology behind mapping host.domain.com to an IP address.
- Four elements comprise the DNS:
  – **Domain name space**: DNS uses a tree-structured name space to identify resources on the Internet.
  – **DNS database**: Each node/leaf in the name space tree names a set of information that is contained in a resource record (RR). Leaf is the actual complete domain name mapping to actual ip address. Each of the mapping is stored in RR. The collection of all RRs is organized into a *distributed* database.
  – **Name servers**: Server programs/devices running the program that hold information about one portion of the domain name tree structure and the associated RRs. Which means each name server contains part of the database. One name server is going to contact other name server/s to find where the data is in this distributed database
  – **Resolvers**: These are programs that query/extract information from name servers in response to client requests.

Domain name characteristics

- Domain names:
  – are case insensitive
  – can have up to 63 characters per constituent (word between two '.')
  – can have up to 255 chars per path
  – can be internationalized (since 1999) – caused security problems
    have websites with similar/same looking symbol, confusion with existing site
- Naming conventions usually follow either organizational or physical boundaries e.g.,
  – au.ibm.com / uk.ibm.com (for email)
  – ibm.com.au / ibm.co.uk (for web)
- Absolute domain names ends in a '.' (Root of tree)
- Relative domain names end in a constituent e.g., .com

Conceptual division of DNS namespace



At top of the domain is '.'
People use .com.au because of marketing issues, to show we are in Australia not US.

Top-level domains

• The same followed within country TLDs
But exceptions...
• abc.net.au is not a network provider
• Many new starting 2014
  – .accenture
  – .calvinklein

Domain	Intended use	Start date	Restricted?
com	Commercial	1985	No
edu	Educational institutions	1985	Yes
gov	Government	1985	Yes
int	International organizations	1988	Yes
mil	Military	1985	Yes
net	Network providers	1985	No
org	Non-profit organizations	1985	No

Resource Records

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

```
; Authoritative data for cs.vu.nl
cs.vu.nl.       86400   IN  SOA     star boss (9527,7200,7200,241920,86400)
cs.vu.nl.       86400   IN  MX      1 zephyr
cs.vu.nl.       86400   IN  MX      2 top
cs.vu.nl.       86400   IN  NS      star

star            86400   IN  A       130.37.56.205
zephyr          86400   IN  A       130.37.20.10
top             86400   IN  A       130.37.20.11
www             86400   IN  CNAME   star.cs.vu.nl
ftp             86400   IN  CNAME   zephyr.cs.vu.nl

flits           86400   IN  A       130.37.16.112
flits           86400   IN  A       192.31.231.165
flits           86400   IN  MX      1 flits
flits           86400   IN  MX      2 zephyr
flits           86400   IN  MX      3 top

rowboat                 IN  A       130.37.56.201
                        IN  MX      1 rowboat
                        IN  MX      2 zephyr

little-sister           IN  A       130.37.62.23

laserjet                IN  A       192.31.231.216
```

The domains are absolute
2 mail services
Zephyr and top
Name server: star

Canonical name: star
ftp server is zephyr, means if you find ftp, go ask zephyr
If you look for flits, you may get one of the two addresses

Inserting records into DNS

- Example: new start-up Network Utopia
- Register name networkuptopia.com at DNS registrar (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into com TLD (top level domain) server:
    - (networkutopia.com, dns1.networkutopia.com, NS)
    - (dns1.networkutopia.com, 212.212.212.1, A)
  - create authoritative server:
    - Type A record for www.networkuptopia.com;
    - Type MX record for networkutopia.com

Example of tools

- Using DNS query tools:

– nslookup – dig – host

Name server zones

- Zones:
  - DNS namespace is divided into overlapping zones. The name servers are authoritative for that zone.
    - usually two name servers for a zone
  - Name servers are arranged in a hierarchical manner extending from a set of root servers
- Root name servers:
  - The root servers form the authoritative cluster for enquiries. The root servers are contacted by a local name server that can not resolve name.
    - if you don't know who to ask for answer? Go ask one of these root servers, they may don't know the answer but they must know who to ask.
  - There are 13 "root names servers" globally
  - the root servers are unique, a "root server" may be a cluster of geographically dispersed servers
    - F-ROOT 252 sites; J-ROOT 162 sites

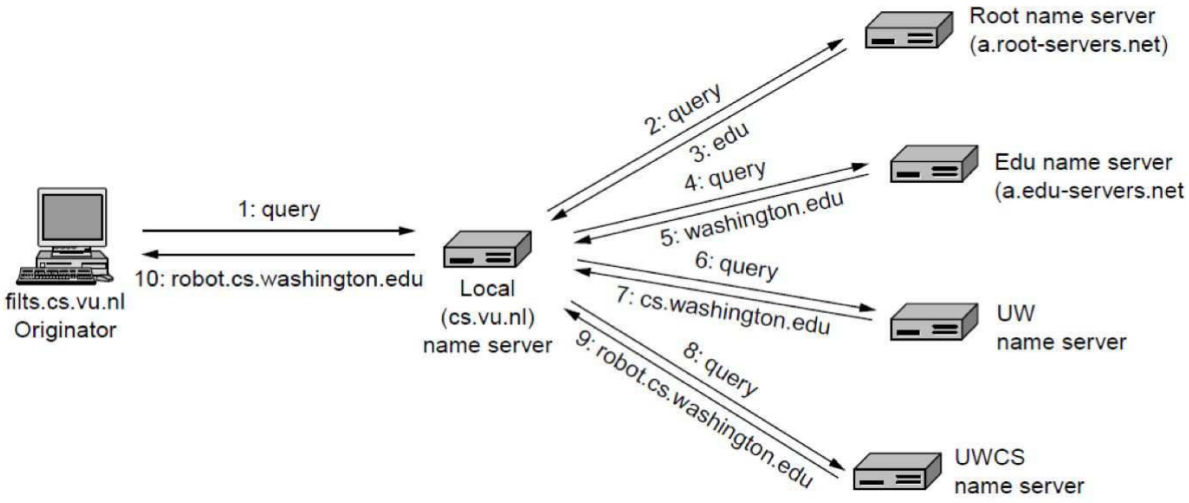
Types of name servers

- Top-level domain DNS servers: responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, au, jp.
  - Examples include: Network Solutions maintains servers for com; and Educause for edu
- Authoritative DNS servers: organizations DNS servers, providing authoritative hostname to IP mappings for organizations servers (e.g., Web, mail).
  - Can be maintained by the organization itself or service provider.
- Local DNS server: Typically, each ISP (residential ISP, company, university) has a "default name server" which handles DNS queries
  - Returns cached value if one exists
  - Otherwise, acts as proxy, and forwards the request up the query hierarchy

Resolving a query

- A resolver client asks the local DNS for the domain to IP mapping:
  - if answer is known by the local DNS, then it sends the answer.
  - if answer is not known, then the local DNS queries up the hierarchy to the top level (root) DNS for the domain and then relays the answer to the resolver client.
- Essentially, this is a recursive query mode. Queries are subject to timers to avoid longer than necessary response times.

Example
It doesn't really reply edu, Washington.edu, it gives the ip address of edu and washington.edu etc



HOSTS File
- Hard-coding mappings
  – Unix: /etc/hosts
  – Windows: C:\Windows\System32\drivers\etc\hosts
- Example:
  127.0.0.1 localhost
  127.0.1.1 username-VirtualBox
  # ad blocking
  0.0.0.0 pagead2.googlesyndication.com
  0.0.0.0 static.adsafeprotected.com
- Local DNS Providers

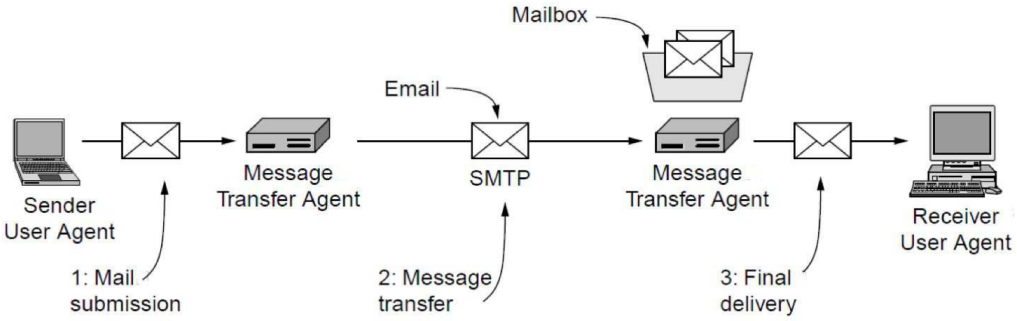
DNS Security
- No security in original design
  – DNS spoofing: is an attack in which altered DNS records are used to redirect online traffic to a fraudulent website that resembles its intended destination
  – DNS flooding: send a small request to DNS server, but reply with a big big amount of information as response. but it may say send this to somebody else. There's a large number of hosts spread in the world, and they all send same request to the DNS server, and they send all the big response back to some poor person who have denial service attack
- Solutions (not fully implemented yet)
  – DNSSEC
  – Root signing

Question: List one motivation for a host to send an IP packet with the wrong source IP address. List two ways that this can adversely affect the legitimate owner of that IP address. A host launching a denial-of-service attack may send packets with a spoofed source address that corresponds to another host, in order to evade detection. The legitimate owner may be blamed for the attack (and perhaps also blocked from sending legitimate traffic to the victim destination), and may also receive unwanted return traffic (e.g., SYN-ACK or RST packets). (Advanced answer: A valid reason for doing it is "network tomography", which is trying to measure properties of links within the network.)

Email services and architecture

- Email has a long heritage (since 1960s)
- In this time, evolutionary steps in infrastructure and standards have been taken.
- Standards for Internet-enabled email are based on 2 RFC's
  - RFC821 (transmission)
  - RFC822 (message format)
  - RFC2821 and RFC2822 (revised versions of earlier RFCs)
- Architecture and Services
  - User agents (UA's/MUA's)
    allow user to read and send email
  - Message transfer agents (MTA's)
    transport messages from source - destination

Email services and architecture



User agent (mail program)

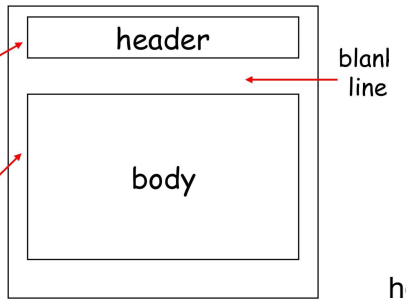
- Basic functions: – compose, report, display, dispose
- Envelope and contents: – encapsulation of transport related information
- Header – user agent control info
- Body – for human recipient
- User must provide message, destination, optional other parameters
- Addressing scheme user@dns-address

Mail Message Format

SMTP: protocol for exchanging email messages

RFC 822: standard for text message format:

- header lines, e.g.,
  To:
  From:
  Subject:
- body
  the "message", ASCII characters only



here, ASCII characters means the 7-bit characters

SMTP – Simple Message Transfer Protocol

- SMTP uses TCP to reliably transfer email message from client to server, default port 25
- Typically direct transfer: sending server to receiving server
- Three phases of transfer (1) handshaking (greeting), (2) transfer of messages, and (3) closure
- Command/response interaction: commands in ASCII text and response consists of status code and phrase
- Messages must be in 7-bit ASCII

MIME – Multipurpose Internet Mail Extensions
Which basically give us attachments, and give us the ways the body it self can be re-encoded to support for unicode content.

- In the early days of email, messages were in English and used only ASCII - RFC 822 reflects these simple constraints. In time, the limitations of RFC822 became clear:
  – other language requirements
  – alternative message content type (audio/images)
- MIME has 5 additional message headers:
  – MIME-Version: identifies the MIME version
  – Content-Description: human readable describing contents
  – Content-Id: unique identifier
  – Content-Transfer-Encoding: how body is wrapped for transmission
  – Content-Type: type and format of content

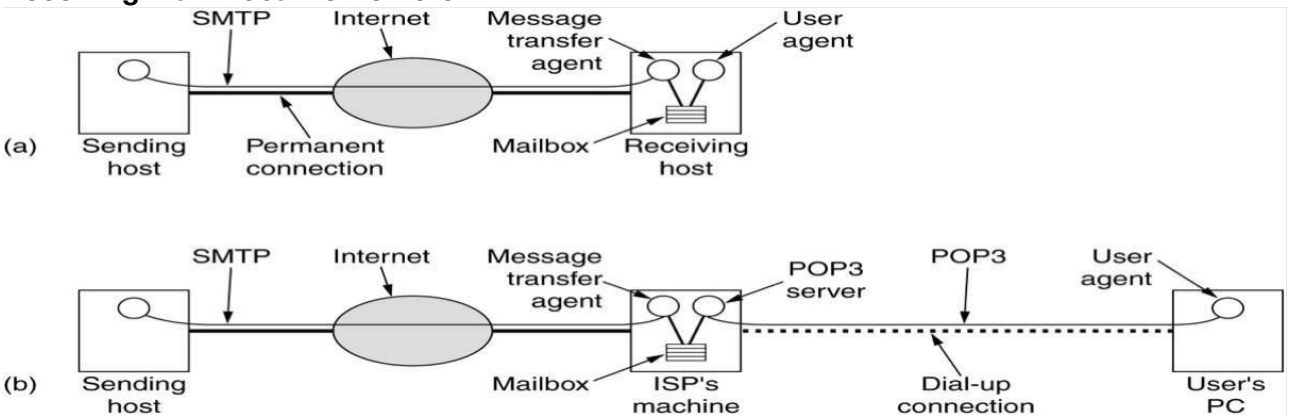
MIME - Content-Types

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

Message Transfer & Access

- Transfer
  – SMTP: delivery/storage to receiver's server
- Delivery
  – Local
  – POP3: Post Office Protocol; authorization (agent – server) and download
  – IMAP: Internet Mail Access Protocol; more features (more complex); provides for the manipulation of stored messages on server
  – HTTP: gmail, Hotmail, Yahoo! Mail, etc.

Receiving Mail: Local vs Remote



Possibly Intermittent connection

- a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent (now rare).
- b) Current case: notebook/PC/phone is not an MTA.

POP3 – Post Office Protocol
- Three states of a POP3 transaction
  – Authorisation
  – Transactions
  – Update
- Syntax
  – USER/PASS
  – LIST
  – RETR/DELE
  – QUIT(update)
- Issue: "**download and delete**" mode does not allow messages to be re-read.
  - if you download something to your PC, t is deleted from the server

IMAP – Internet Message Access Protocol
- IMAP keeps user state across sessions.
  – Retain mailbox contents online (server) and allow manipulation of online and offline messages and mailbox folders
  - just reading doesn't delete content
  – Implications of server infrastructure to support high volume of IMAP users. This implies storage projections by the provider, and hence limitations.

IMAP vs POP3:
IMAP is better on user's point of view but it requires more resources
The best protocols/standards are the best for their time. Once we have another computer power to support IMAP, then IMAP is better than POP. If we don't have computer power, it's better to have POP than having nothing

Streaming
- 40% of internet download traffic is streaming (video, audio)
  – Not necessarily 40% of core traffic, due to caching
- WebSockets (ws://... and wss://...)
  – Multiplexing full-duplex channels over one TCP connection
  – Everything on TCP port 80 or 443 (HTTP/HTTPS) to pass firewalls
  – Unlike HTTP, doesn't require client to keep requesting. -> streaming
- RTP + RTCP/RTSP (Real Time [Control/Streaming] Protocol)
  – RTP uses a playback buffer to deliver packets with the same spacing as they were sent, undoing network jitter
  – RTCP Monitors delays, adapts video coding rate to available capacity
  – RTSP provides play/record/pause services
- Real Time Messaging Protocol (RTMP, used by Flash)

Week08

Transport service primitives

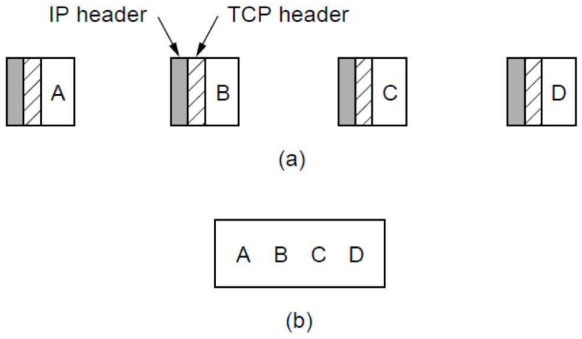
Primitive	Package	Meaning
LISTEN	(None)	Block until something tries to connect
CONNECT	CONNECTION REQUEST	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(None)	Block until DATA packet arrives
DISCONNECT	DISCONNECT REQUEST	The sids wants to release the connection

Connection establishment issues

- Remember that TCP is a connection orientated protocol running over a connectionless network layer (IP)
- When networks can lose, store and duplicate packets, connection establishment can be complicated
  – congested networks can happen when we have queue of packets or send over the same link may delay acknowledgements
  – incurring repeated multiple transmissions
  – any of which may not arrive at all or out of sequence – delayed duplicates

TCP - Overview

- The Transmission Control Protocol provides a protocol by which applications can transmit IP datagrams within a connection-oriented framework, thus increasing reliability.
  - datagram means the packet is rooted independently through the network, not a part of the sequence, TCP sends sequence and make sure everything is in the order, IP just sends each packet as single items
  – TCP transport entity manages TCP streams and interfaces to the IP layer
  – TCP entity accepts user data streams, and segments them into pieces < 64Kb (often 1460 bytes in order to fit the IP and TCP headers into a single Ethernet frame), and sends each piece as a separate IP datagram
- Recipient(receiver) TCP entities reconstruct the original byte streams from the encapsulation, so labels are needed to reorder and avoid duplicates

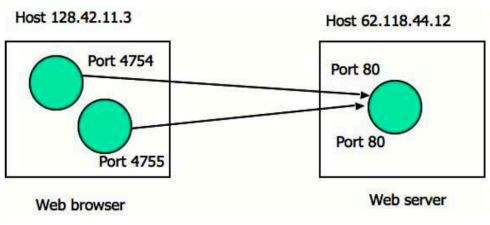


•Example:
– (a)Four 512-byte segments sent as separate IP datagrams, The thing other than headers is payloads
– (b)The 2048 bytes of data delivered to the application in a single READ call. If you READ, it just return whatever in the packet, maybe just A,B, or nothing. If set to blocking mode, it will say here is all I got so far. The user of tcp may tell whether this is all packets or not

TCP – Service Model

The sender and receiver both create **sockets**
– A kernel data structure, named by the 5-tuple of IP address and port number of sender and receiver, and the protocol
– For TCP service to be activated, connections must be explicitly established between a socket at a sending host (src-host, src-port) and a socket at a receiving host (dest-host, dest-port), this means we can have 1 socket for listening for connections, and multiple TCP entities can try to connect it and each produce an different socket

Example:
– 2 sockets on port 80



Features of TCP connections

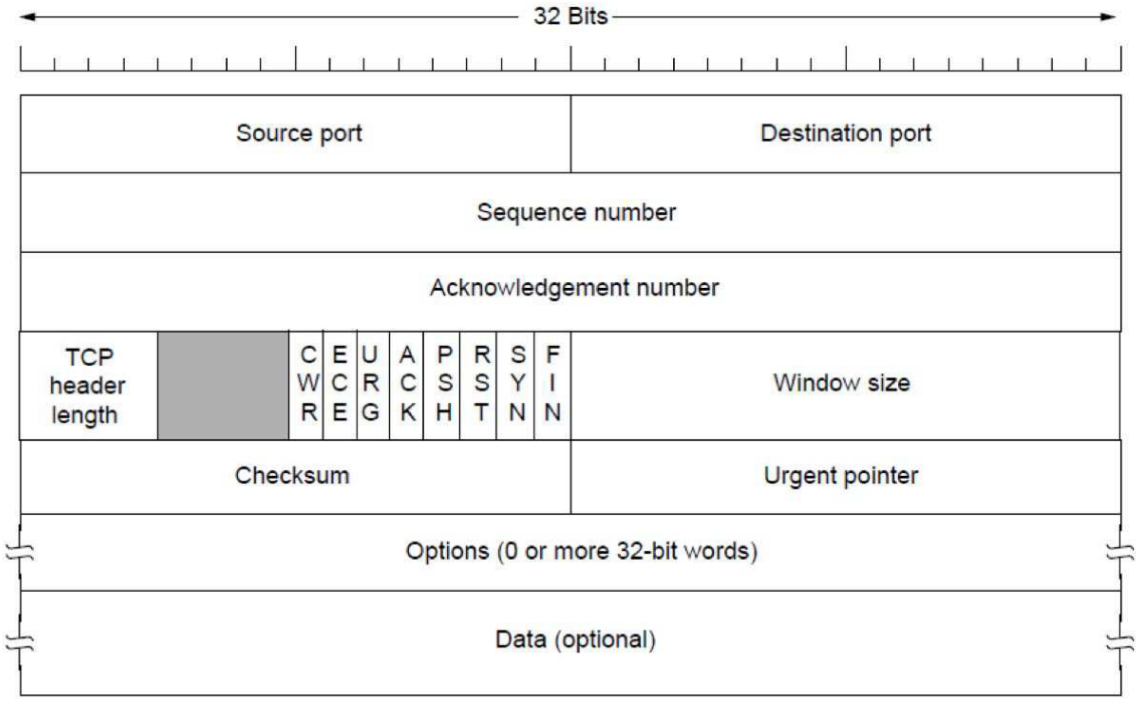
- TCP connections are:
  – Full duplex - data in both directions simultaneously, with data or acknowledgements, which provides liability because the sender will know it doesn't need to send again
  – End to end - exact pairs of senders and receivers
  – Byte streams, not message streams - message boundaries are not preserved
  – Buffer capable - TCP entity can choose to buffer prior to sending or not depending on the context
    - PUSH flag - indicates a transmission is not to be delayed, and should interrupt the receiving application
    - URGENT flag - indicates that transmission should be sent immediately (priority above data in progress), and that the receiver should send it to the application out-of-band (not part of its byte stream)

Related Question: Leslie designed a new application layer protocol to run on top of TCP for file system management. The protocol was going to be very efficient: a message would consist of a single letter command followed by a filename. Two of the commands are ”C” to copy the file from the server to the client, and ”D” to delete the file. Leslie wrote the client and got Alex to write the server. Leslie has three files on the server, called ”ate” and ”BlindDate” and ”Blind”, and wants to download them both. However, the command sequence ”CBlindDate” and ”Cate” gave an error.
(a) What could the error have been?
(b) Could Alex have made any other interpretation of the protocol that would have given a different error?
This question is about the fact that TCP provides a byte stream, and does not tell the receiver the locations of the boundaries of the data sent by individual send or write calls. This has to be done by the higher layer protocols, and Leslie's does not do it.
(a) Two possible errors are: “File BlindDateCate was not found”, and “File ate not found”.
(b) The problem is that TCP provides a byte stream, and Leslie's protocol doesn't specify any delimiters between messages. The first error would occur if the server kept reading data and ran to two commands in together. The second would occur if the server assumed that a file name ends when a command is encountered; in this case, a D. The messages would be parsed as CBlind|Date|Cate. The first message would cause Blind to be downloaded, and ate to be deleted, and the second would be unable to find ate.

TCP Properties

- Data is exchanged between TCP entities in segments
  – each has a 20–60 byte **header**, plus zero or more data bytes, zero bytes for pure acknowledgement with just a header with acknowledge flag
- TCP entities decide how large segments should be, given two constraints:
  – IP payload < 65, 515 byte
  – Maximum Transfer Unit (MTU) - generally 1500 bytes
  - normally a timeout, we wait until the time until the first data receive, allow us be responsible for data received and efficient
- Sliding window protocol
  – Initial use: reliable data delivery without overloading the receiver
  – Now also tied closely with congestion control

TCP - Header

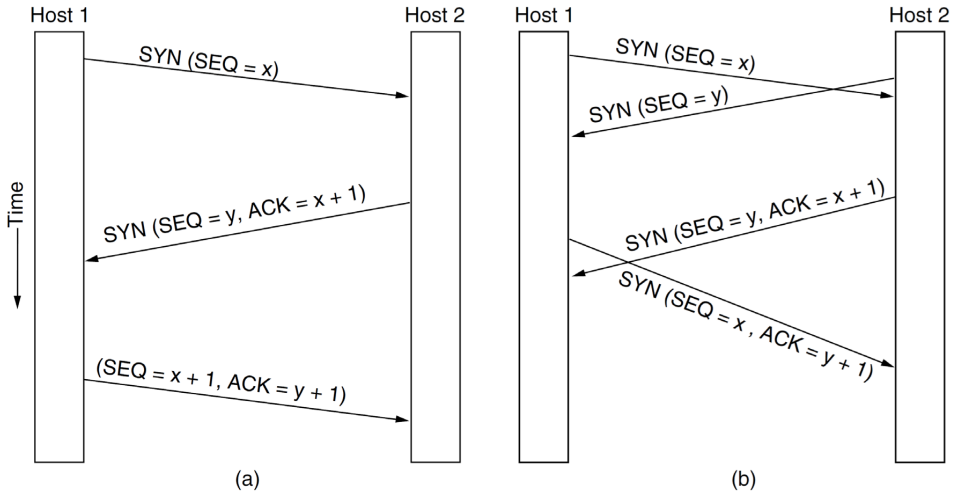


The sequence number is for sliding window
FIN: finish, end of connection SYN: start a new connection
RST: reset, finish but don't know what it is in the first place or it never work
PSH: push packets URG: urgent packets
ACK: whether or not this acknowledgement number is actually acknowledging the data
CWR/ECE: for congestion control

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

Three-way handshake

- Goals of reliable connection establishment:
  – Ensure one (and only one) connection is established, even if some set-up packets get lost
  – Establish initial sequence numbers for sliding window
- Three-way handshake:
  – A proposed solution, which avoids problems that can occur when both sides allocate same sequence numbers by accident (e.g. after host/router crash) (cf. Tomlinson, 1975).
  – Sender and receivers exchange information about which sequencing strategy each will use, and agree on it before transmitting segments



- Normal operation,
- Simultaneous connection attempts.
  – Two simultaneous connection attempts results in only one connection (uniquely identified by end points).
- At end, Host 1 and Host 2 have agreed on respective sequence numbers

Synchronisation

- SYN is used for synchronization during connection establishment
  – Sending SYN or FIN causes sequence number to be incremented by 1
- Sequence Number – first byte of this segments payload
  – Offset by a random number – initial value is arbitrary, offset will be reflected in both Sequence and Acknowledgement numbers
- Acknowledgement Number – next byte the sender expects to receive
  – Bytes received without gaps – a missing segment will stop this incrementing, even if later segments have been received

TCP Synchronization Recap

- SYN bit is used to establish a connection
  – Connection request has SYN=1, ACK=0
  – Connection reply has SYN=1, ACK=1
- SYN is used in both CONNECTION_REQUEST and CONNECTION_ACCEPTED, ACK bit distinguishes between the two
- After connection setup:
  – Sequence Number –first byte of this segment payload (1 + data sent
    - Offset by a random number – initial value is arbitrary, offset will be reflected in both Sequence and Acknowledgement numbers

  – Acknowledgement Number – next byte the sender expects to receive (data successfully received + 1)
    - Bytes received without gaps – a missing segment will stop this incrementing, even if later segments have been received

TCP Retransmission
TCP is a reliable protocol, but it doesn't mean the packets are delivered correctly. If the packet gets lost, it will retransmit. It will continue retransmitting until the packets correctly received or until the connection is cut off.

- Each segment sent has an associated retransmission timer (RTO)
  – Initialized with a default value and updated based on network performance (path and congestion)
  – If the timer expires before an ACK is received the segment is resent
- Receiver receives segment with a sequence number higher than expected (i.e. segment has been lost)
  – Receiver sends ACK with sequence number it is expecting (i.e. the next byte it expects – also implies data it has received)
  – This is a duplicate of the previously sent acknowledgement (DupACK)
  – After receiving 3 DupACKs the sender resends the lost segment, this is known as fast retransmission

TCP Closing

- The FIN flag is used to signify a request to close a connection
- Each FIN is directional, once acknowledged no further data can be sent from the sender to the receiver
  – Data can continue to flow in the other direction, you can keep sending acknowledgements, but no new data. (Size 0)
  – E.g. client could send FIN after making request, but before receiving the response
  – Sender of FIN will still retransmit unacknowledged segments
- Typically requires 4 segments to close, 1 FIN and 1 ACK for each direction
  – Can be optimized: Host A send FIN request, Host B responds with ACK of Host A FIN request, and sends FIN request of its own, Host A send ACK of Host B's FIN request and connection is closed
- The RST flag is used to signify a hard close of a connection
  – Basically states the sender is closing the connection and will not listen for any further messages, sometimes used when blocking a port
  – Sent in reply to a packet sent to a 5-tuple with no open connection
  - e.g., to invalid data being sent or a crashed process that left a remote socket open, that the OS is now cleaning up
- Can be used to close a connection, but FIN is greatly preferred because it is an orderly shutdown of the connection, as opposed to a reset

QUIC

- Google's QUIC has shown that there is demand for a new transport protocol
  – Why is it only now being deployed?
  – Why does it run on top of UDP?
- Although the transport layer is supposed to be "end to end", there are many "middle-boxes" that inspect and modify transport layer protocols
  – Firewalls, NAT intrusion detection systems, load balancers
- These all accept TCP and UDP, but typically drop all other transport layer protocols.
- Stream Control Transport Protocol is a carefully designed protocol that achieves many of the goals of QUIC, but it hasn't been widely deployed because of these middle-boxes.
- Difference between TCP/UDP: there will be multiple flows in connection, each has its own sequence number, no need to wait until a flow is finished

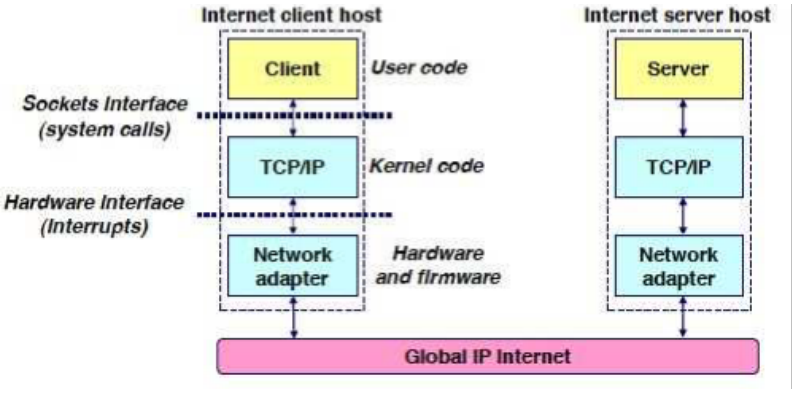
SYN Flooding

– Popular attack in the 90's to denial of service a server
– Remember back to the arbitrary (random) initial Sequence number
– This requires the server to remember an initial Sequence number for each received SYN request
– An attacker would make initial SYN requests then not send the appropriate ACK, causing the server to gradually fill up its queue with sequence numbers for now defunct connections
– One solution was SYN Cookies
  • Rather than store the sequence number it is derived from connection information and a timer that creates a stateless SYN queue using cryptographic hashing
  • Incurs performance cost in validating SYN Cookies, but preferable to being unresponsive – typically only enabled when under attack

Sockets

Maps the network interface, the ip stream comes in to file descriptors, typically one per process

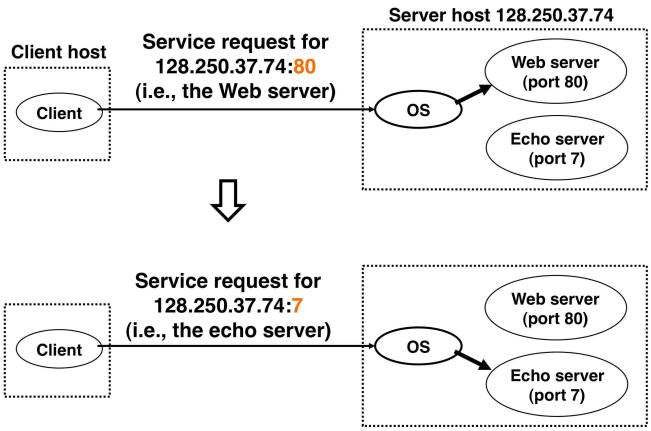
• A message going from one host to another must cross the underlying network.
• A process sends and receives through a socket
  – the "doorway" leading in/out of the application



Using sockets

• The "address" of a socket is the 5-tuple:
– Protocol
– source-IP
– source-port number
– destination-IP
– destination-port number (for multiplexing)

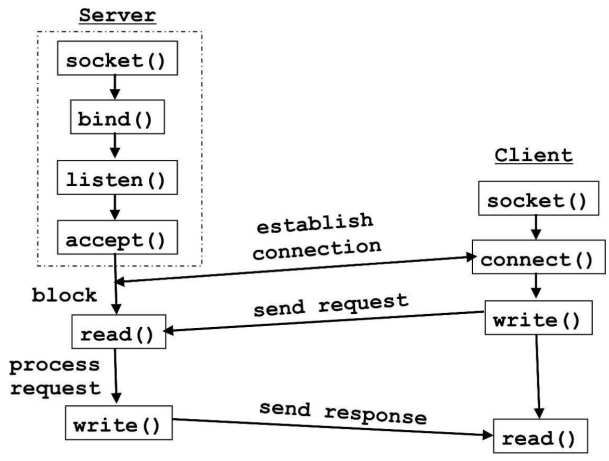
Multiplexing: distinguish from multiple inputs
This 5 tuple defines a flow of packets, a socket is almost identified with a flow of packets



Berkeley Sockets

• Socket interface
  – originally provided in Berkeley UNIX
  – later adopted by all popular operating systems
  – simplifies porting applications to different OSes
• In UNIX, everything is like a file
  – all input is like reading a file
  – all output is like writing a file
  – file is "addressed" by an integer file descriptor
• API implemented as system calls:
  – examples include connect(), read(), write(), close()

Using sockets



socket(): creat a socket
bind(): associate with a ip address
listen(): to see if sb is going to say sth
accept(): ready to accept sth

connect(): connect to the ip address the port

Socket Primitives

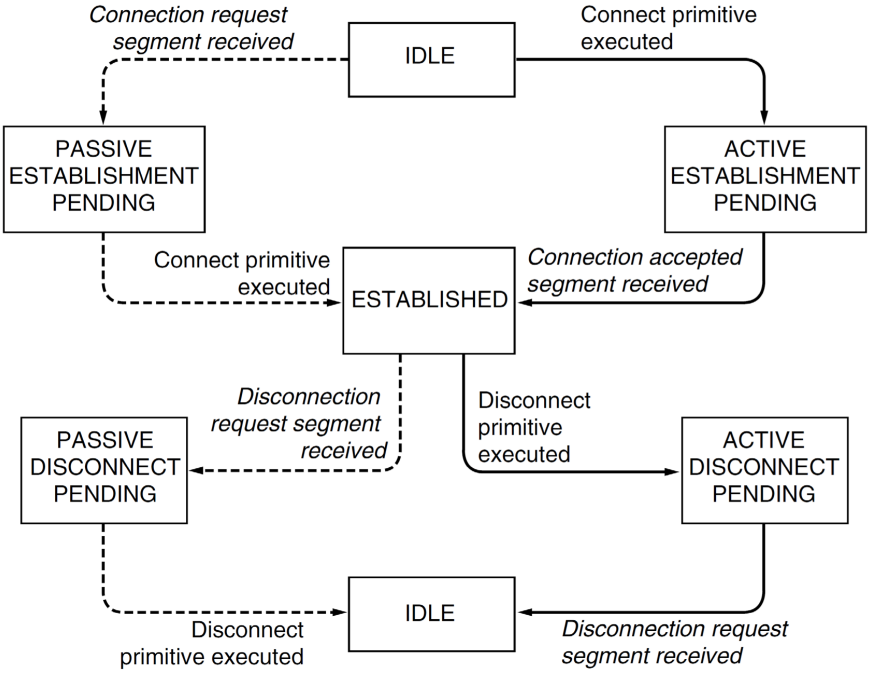
State	Description
SOCKET	Creates a new communication endpoint
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection (block until then)
CONNECT	Actively attempt to establish a connection
SEND	Send some data over a connection (write())
RECEIVE	Receive *some* data from the connection (read())
CLOSE	Release the connection

Bind, listen and accept are used for passive open, connect is used for active open
Listen sets up a buffer to queue incoming SYN packets (connect requests) and then returns
Accept blocks until a SYN packet has been received, and then generates and returns a full 5-tuple socket.
Accept and connect deal with 5-tuple identified sockets.
Return values: bind, listen, connect: an error code (0 on success, -1 on error). accept: Non-negative file descriptor of the new socket on success, -1 on error.

Related Question: What flexibility would be lost if there was a single call to bind+listen+accept?
The listen call reserves a port for this application. It may be useful to reserve the port before the application is ready to accept calls. A bind call followed by closing the socket can also be used to check whether or not a port is available.

Related Question: A process on host 1 has been assigned port p, and a process on host 2 has been assigned port q.
(a) Is it possible for there to be two or more TCP connections between these two ports at the same time?
(b) Is it possible for there to be more than one TCP connection on port p of host 1 at a time?
(a) No. A connection is identified only by its sockets. Thus, (1, p) → (2, q) is the only possible connection between those two ports.
(b) Yes. There can (in principle) be trillions of connections using port p on host 1, as long as they all have different remote IP addresses or different remote ports. (To arrive at "trillions", note that an IPv4 address is 32 bits and the port number is 16 bits.) In practice, there are unlikely to be more than a few thousand.

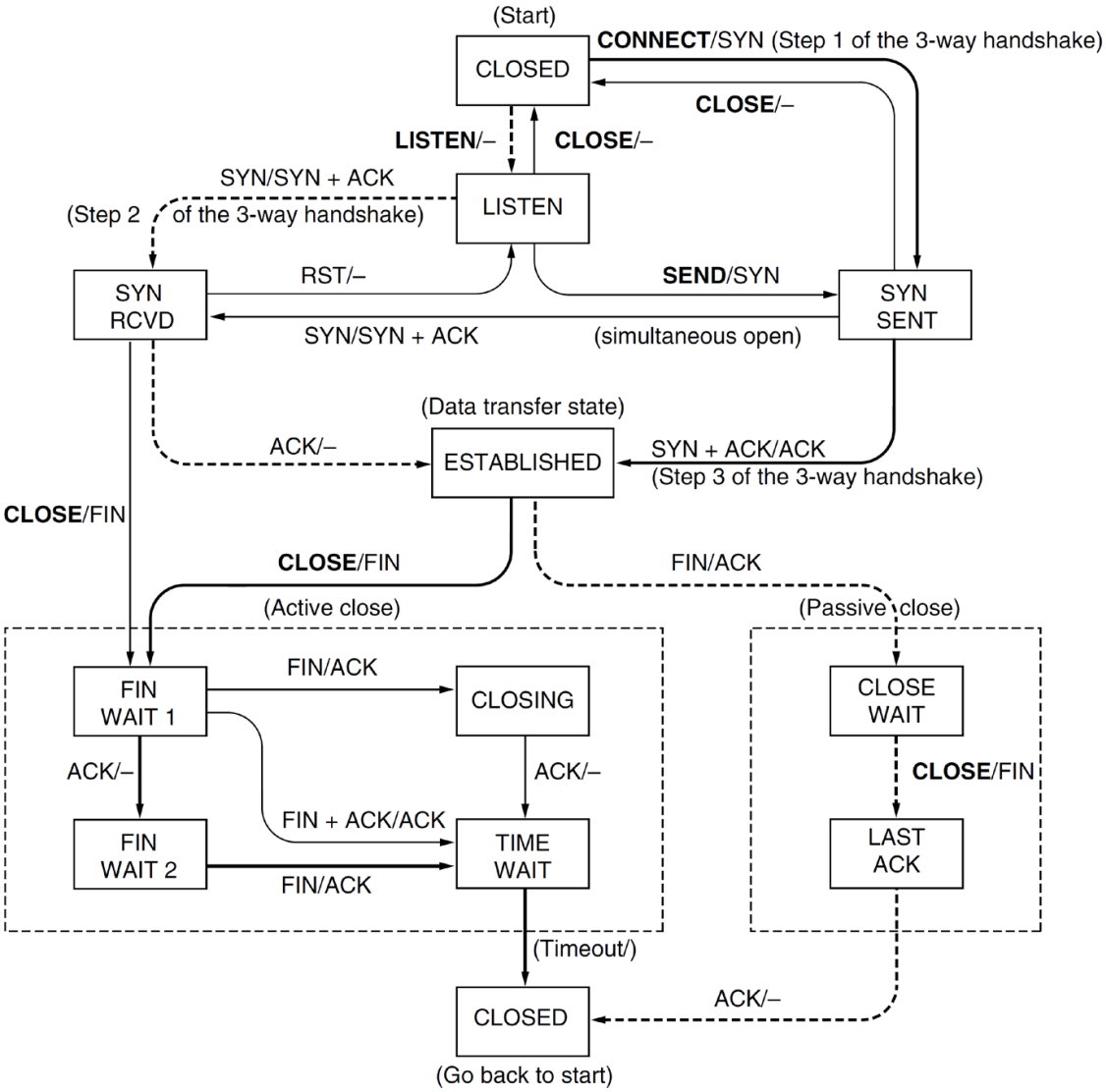
Simplified state diagram for connection management



Complete list of Socket States

State	Simplified name	Description
CLOSED	Idle	No connection is active or pending
LISTEN	Pass. est.	The server is waiting for an incoming call
SYN RCVD	Pass. est.	A connection request has arrived; wait for ACK
SYN SENT	Act. est.	The application has started to open a connection
ESTABLISHED	Established	The normal data transfer state
FIN WAIT1	Act. disc	The application has said it is finished
FIN WAIT2	Act. disc	The other side has agreed to release
TIME WAIT	Act. disc	Wait for all packets to die off
CLOSING	Act. disc	Both sides have tried to close simultaneously
CLOSE WAIT	Pass. disc	The other side has initiated a release
LAST ACK	Pass. disc.	Wait for all packets to die off

Socket Finite State Machine
Bold before slash: System call e.g., connect
Non-bold before slash: Packet received e.g., SYN
After slash: Packet sent e.g., SYN



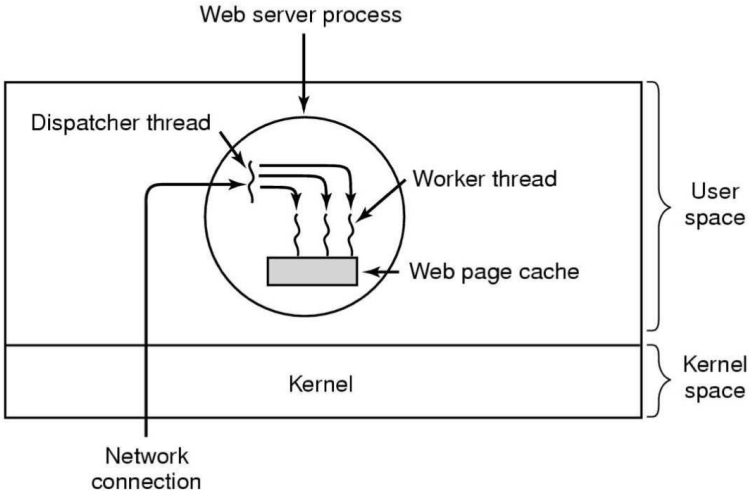
Sockets in C

- **Headers**
  #include <arpa/inet.h>
  #include <sys/socket.h>
  #include <netinet/in.h>
- **Variables**
  int listenfd = 0, connfd = 0; //file descriptor
  char sendBuff[1025];
  struct sockaddr_in serv_addr;
- **Create a socket**
  listenfd = socket(AF_INET, SOCK_STREAM, 0); //create socket
  memset(&serv_addr, 0, sizeof(serv_addr)); //initialise server address
  memset(sendBuff, 0, sizeof(sendBuff)); //initialise send buffer
  serv_addr.sin_family = AF_INET; //Type of address – internet IP
  serv_addr.sin_addr.s_addr = htonl(INADDR_ANY); //Listen on ANY IP Addr
  serv_addr.sin_port = htons(5000); //Listen on port 5000
- **Bind and listen**
  bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
  listen(listenfd, 10); // maximum number of client connections to queue
- **Accept, send, close**
  connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);
  snprintf(sendBuff, sizeof(sendBuff), "Hello World!"); write(connfd, sendBuff, strlen(sendBuff));
  close(connfd);
- **Wait until one of several files is ready to read / write**
  - select (), pselect (), poll ()
- **Connect (client)**
  connfd = socket(AF_INET, SOCK_STREAM, 0); //create socket
  connect(connfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))
- Receive (client)
  while ((n = read(connfd, recvBuff, sizeof(recvBuff)-1)) > 0) {
  //process received buffer
  }
  – If read is set to be blocking, it waits until there is data
  • This loop reads the whole connection
  – If non-blocking, this just reads data that has arrived
  • More may come after a delay

Related Question: Given a TCP packet in buffer buf, write C functions to:
• return 1 if the SYN flag is set and 0 otherwise;
• return a pointer to the first byte of the TCP payload.
(a) int syn(unsigned char *buf) { return (buf[13] & 2) != 0; }
(b) int payload(unsigned char *buf) { return buf + (buf[12] >> 4)*4; }

Multi-threaded Web Server

- Clearly a web server needs to be able to handle concurrent connections from multiple clients
- This can be achieved through the usage of a multi-threaded web server



Multi-threaded web server

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

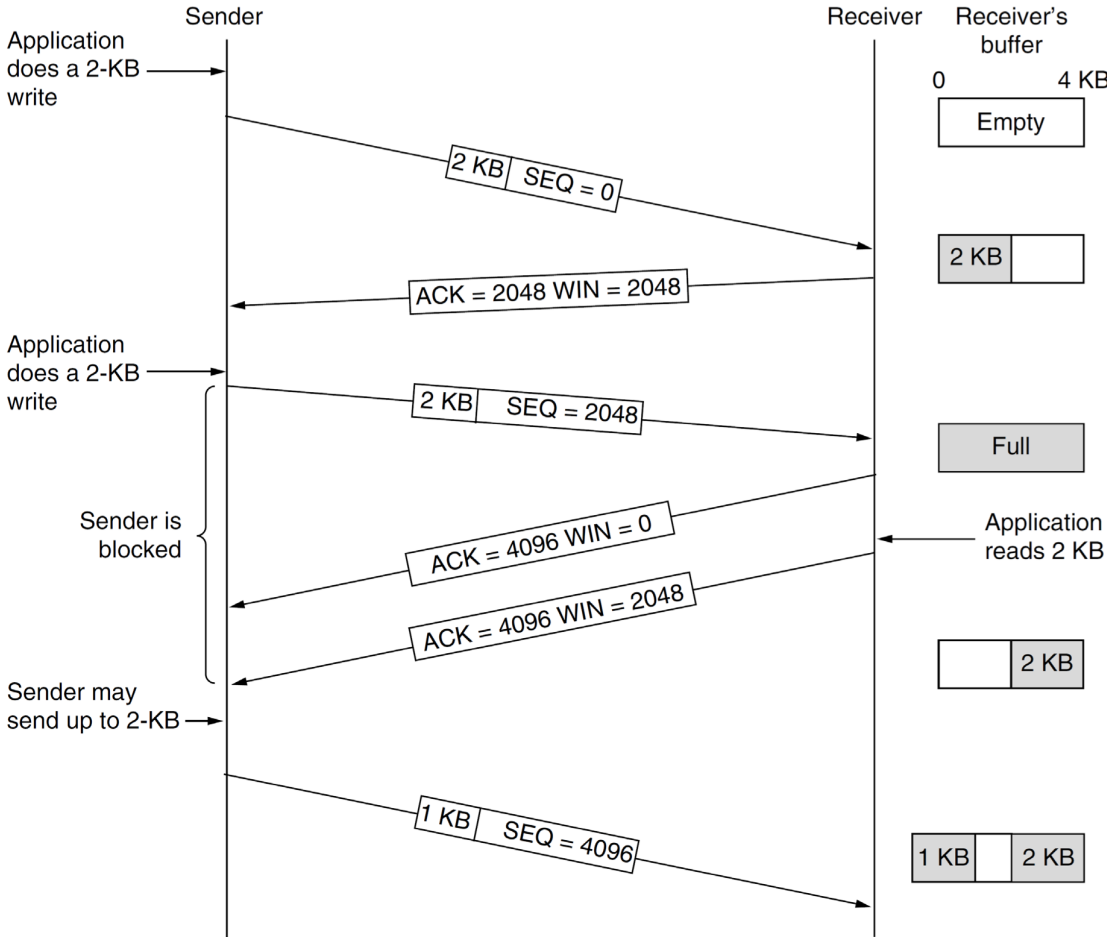
(b)

High level outline of code for previous slide:
– Dispatcher thread
– Worker thread

TCP Sliding Window

- Sliding window is controlled by receiver
- Determines amount of data the receiver is able to accept
  – Sender and receiver maintain buffers to send and receive data independently of the application
  – No guarantee that data is immediately sent or read from the respective buffers

TCP Sliding Window



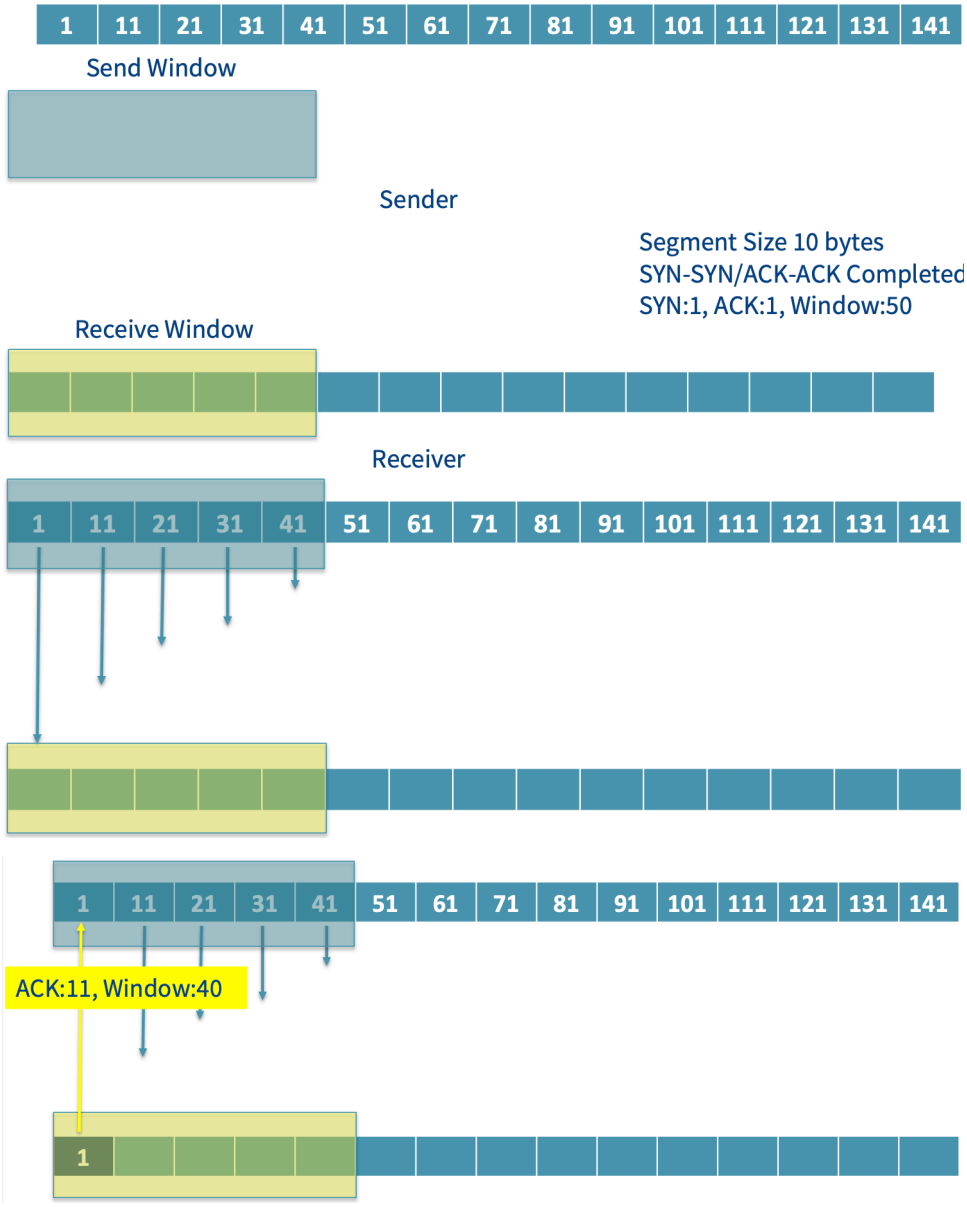
Related Question: Consider a data stream consisting of 1000-byte packets, starting with sequence number 0. Consider a window of 4000 bytes. Imagine that the packets starting at bytes 0, 3000, 6000 are lost, and that the (first) acknowledgment of the packet starting at byte 4000 was lost. Assume that the higher layers read the entire buffer contents once the buffer is full. Draw a sequence like the picture above, with time going down, and the flow of packets shown by sloping lines. (For exam practice, repeat this question with losses of different packets, or with a different sized buffer.)

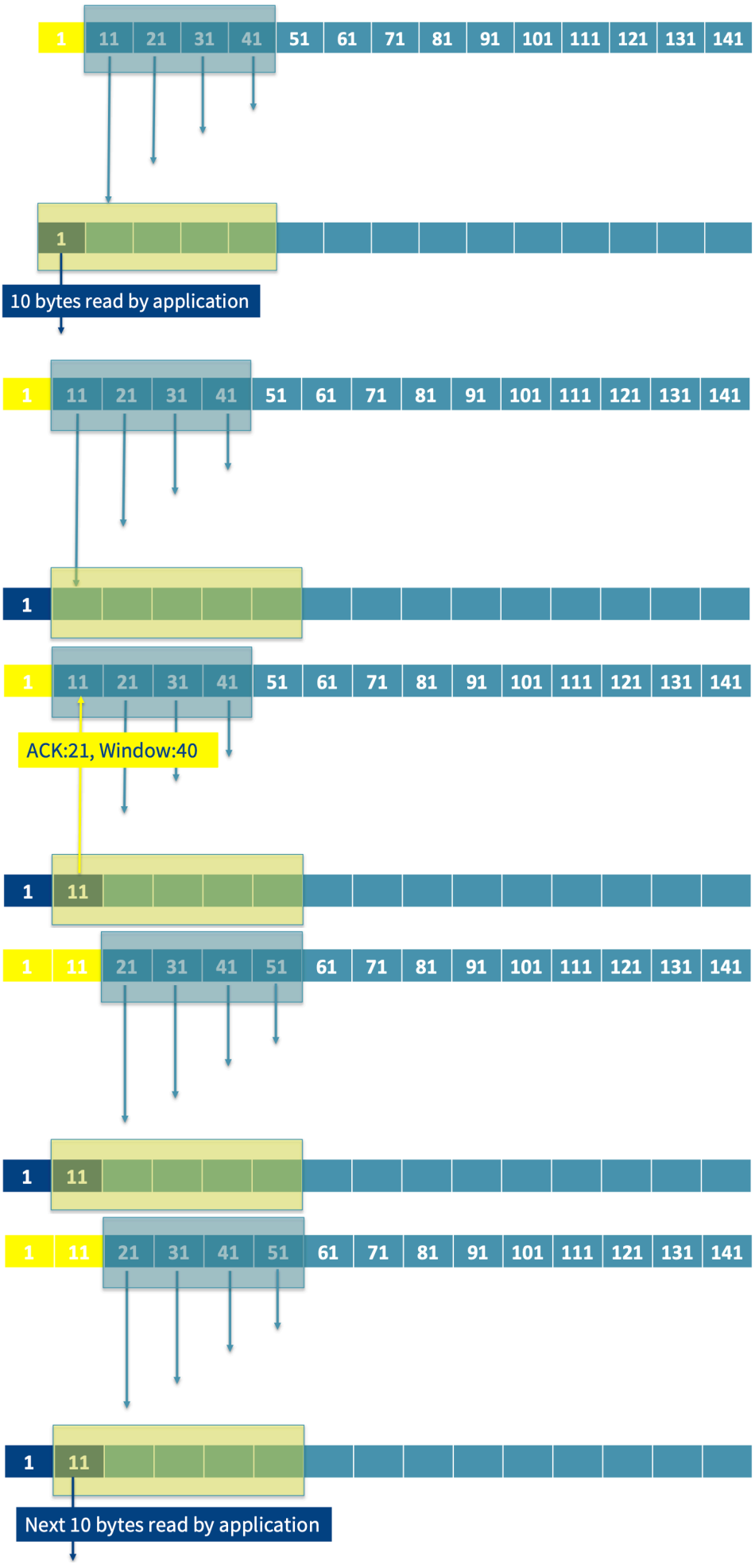
TCP Sliding Window

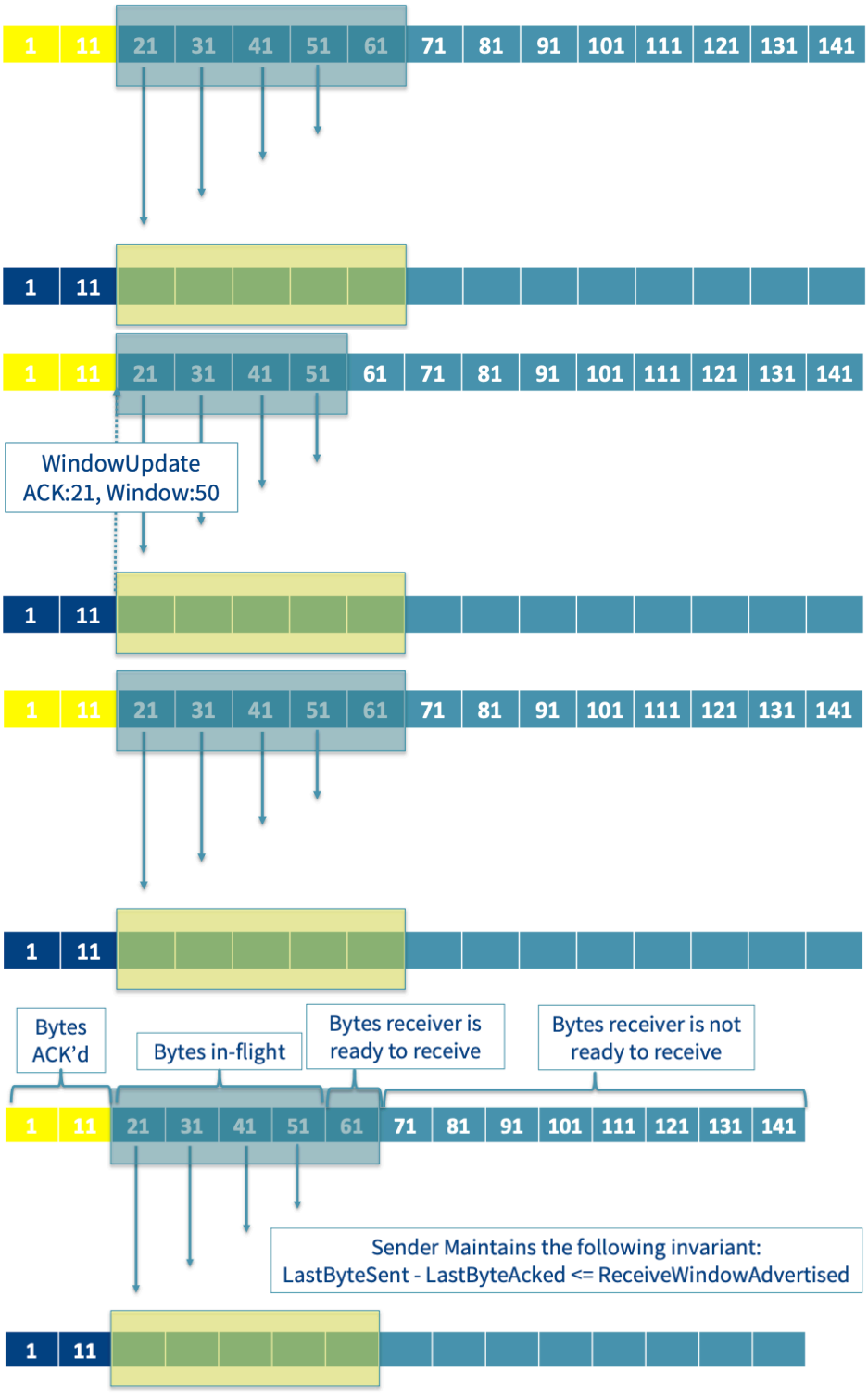
- When the window is 0 the sender should not send any data
  – Can send URGENT data (because it's out-of-band, does not goes into buffer)
  – Can send "zero window probe": 0 byte segment that causes the receiver to re-announce the next expected byte and window size (window probe) this is designed to prevent deadlock
  The deadlock here: Neither is going to send data saying now they are having a none zero window. If we can never send data when the window was 0, then if they were both 0 we have no way to break the deadlock as neither of them are able to send data telling them the situation has changed. "Here is no data, please tel me have you get space in your window yet". We will do that typically if the application read from the receive buffer, you will be able to send "zero window probe" packet
- Senders may delay sending data, e.g., instead of sending the 2kiB immediately, could wait for a further 2kiB to fill the 4kiB receive window

- Send window
  – What data the sender is able to send – unacknowledged segments and unsent data that will fit into the receive window
- Receive window
  – Amount of data the receiver is willing to receive – window size in ACK
- Other windows are maintained for congestion control

Example



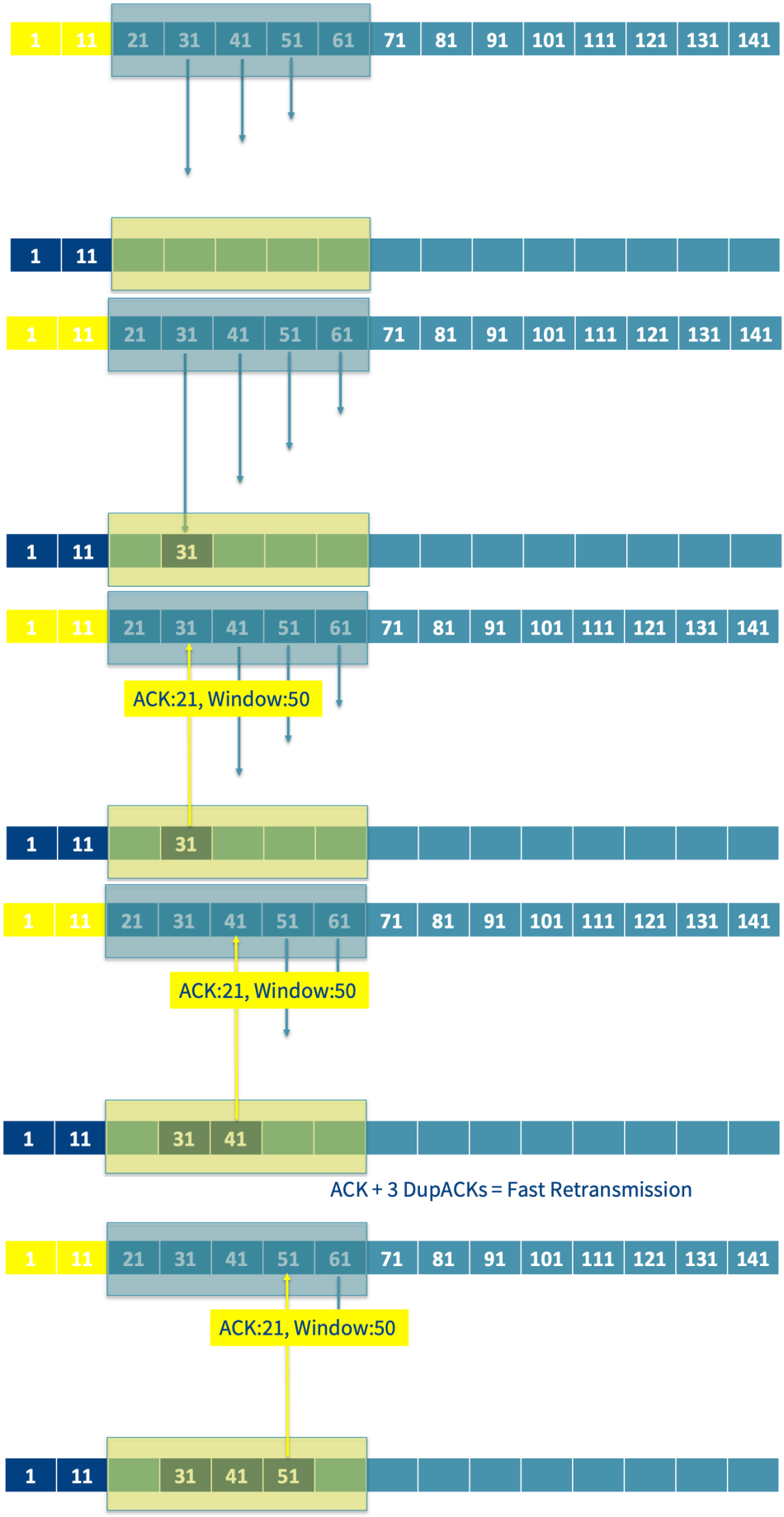

1 11 21 31 41 51 61 71 81 91 101 111 121 131 141
1
10 bytes read by application
1 11 21 31 41 51 61 71 81 91 101 111 121 131 141
1
1 11 21 31 41 51 61 71 81 91 101 111 121 131 141
ACK:21, Window:40
1 11
1 11 21 31 41 51 61 71 81 91 101 111 121 131 141
1 11
1 11 21 31 41 51 61 71 81 91 101 111 121 131 141
1 11
Next 10 bytes read by application


1 11 21 31 41 51 61 71 81 91 101 111 121 131 141
WindowUpdate
ACK:21, Window:50
Bytes
ACK'd
Bytes in-flight
Bytes receiver is
ready to receive
Bytes receiver is not
ready to receive
Sender Maintains the following invariant:
LastByteSent - LastByteAcked <= ReceiveWindowAdvertised

Related Question: TCP's sliding window is responsible for providing both in-order delivery of data and reliable delivery. (a) Imagine you wanted a service that provided reliable delivery but not in-order delivery without changing the TCP sender. Could you make the sliding window "better" (e.g., send fewer packets, reduce the delay in making data available to the application) with this weaker service guarantee?
(b) Imagine you wanted in-order delivery only, without guaranteeing reliable delivery. Could you make the sliding window "better" in this case?
(c) If you can also modify the sender in the second case, what is the most "efficient" implementation, where "efficiency" is measured in packets sent?
(a) You could allow the packets to be read as soon as they are received. The receiver would still need to keep track of which packets it has sent to the higher layers to ensure that it doesn't accidentally delivery data twice; "reliable delivery" includes a guarantee that data isn't delivered more than once. This would also require telling the higher layer where in the data stream the packet belongs, since it can't assume that the nth byte read is the nth byte of the file.
(b) You could simply not ask for retransmissions, and acknowledge each packet as soon as it is received. The left hand side of the receive window would advance to the packet most recently sent to the application layer; any packet before the window would have to be deleted to avoid out-of-order delivery.
(c) The most "efficient" protocol would simply discard all data at the sender, and send no data. Of course this is absurd. Most unreliable transport protocols, like IP and UDP, promise "best effort" service, which means they actually try to get data across even though they don't guarantee it.

Week09

TCP Sliding Window – Segment Loss



Original flow + loss control

- Flow+loss control had existed on single point-to-point links for a long time
- TCP originally used the experience from the link layer
- Caused bad design decision.

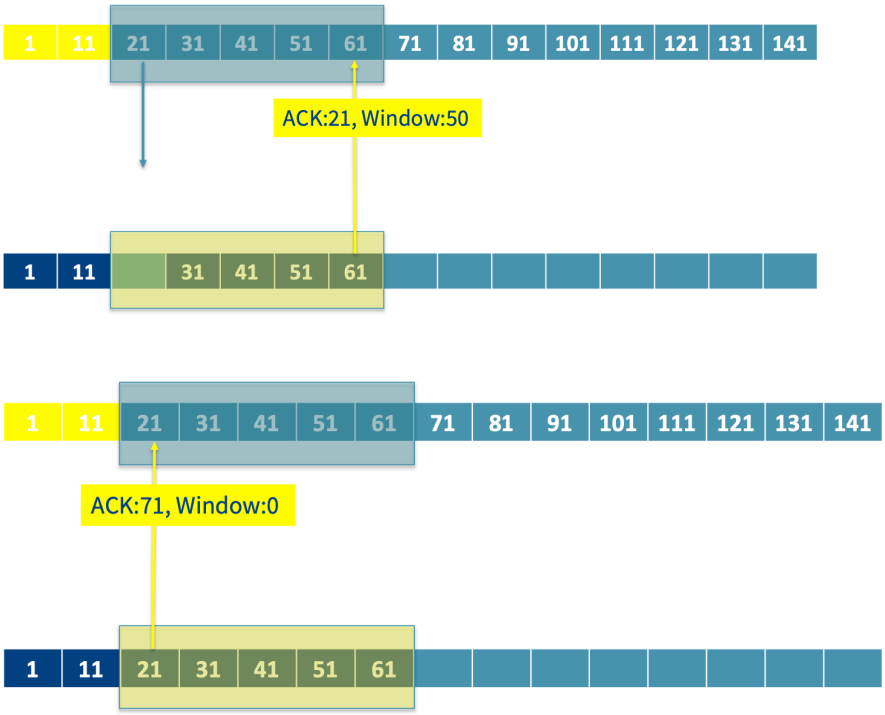
Two types of flow control

- "Go-back-N" (3.4.3 of Kurose and Ross)
  – When a packet is lost, go back to the point where it was transmitted, and (re)transmit everything from that point onward
  – Advantage: Receiver doesn't need to store/reorder packets
- "Selective Repeat" (3.4.4 of Kurose and Ross)
  – Only retransmit the lost packet
  – Packets arrive out of order. Receiver must store out-of-order packets to send them in-order to the application
- Selective repeat: more complex, only helps if loss is common
  – Link layer fixes errors. Errors will be rare – right?

Congestion collapse

- In the late 1980s, the internet had "congestion collapse"
  – It took tens of minutes to send a packet to the next building
- Van Jacobson diagnosed and solved the problem
- Router buffers were overflowing, causing high loss
- Senders were doing go-back-N, so that every packet loss caused N more packets to enter the system
- Solution: Selective repeat ("fast retransmit")
  – "Packet conservation" principle: don't send packet into the network until a packet come out from the network

TCP sliding window - Fast retransmit

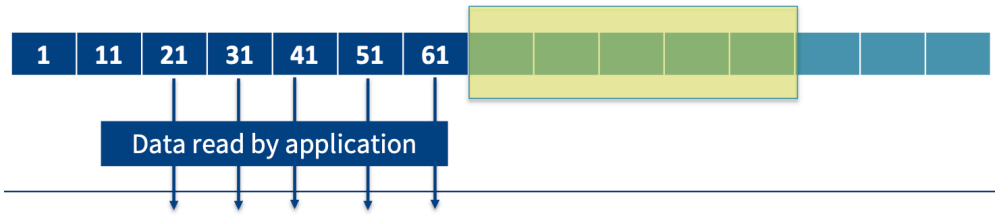


Potential for deadlock

Potential for deadlock

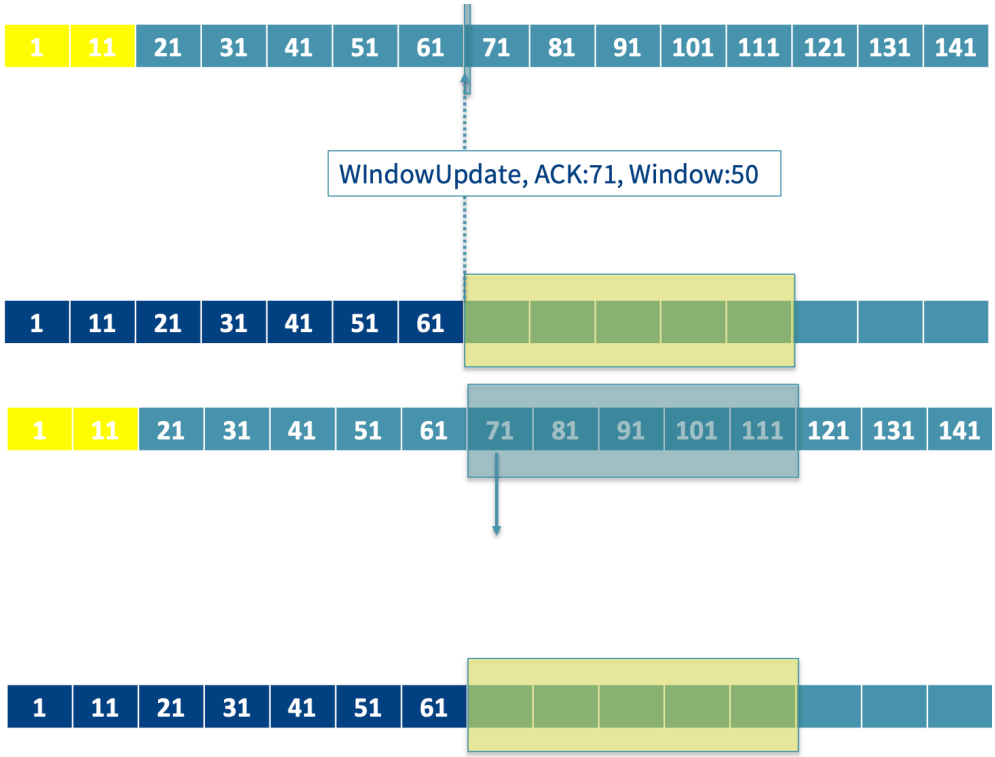
- Sender won't send any more data (window size is 0)
- Receiver won't receive anything, so won't send any ACKs to increase window size

Avoid Deadlock



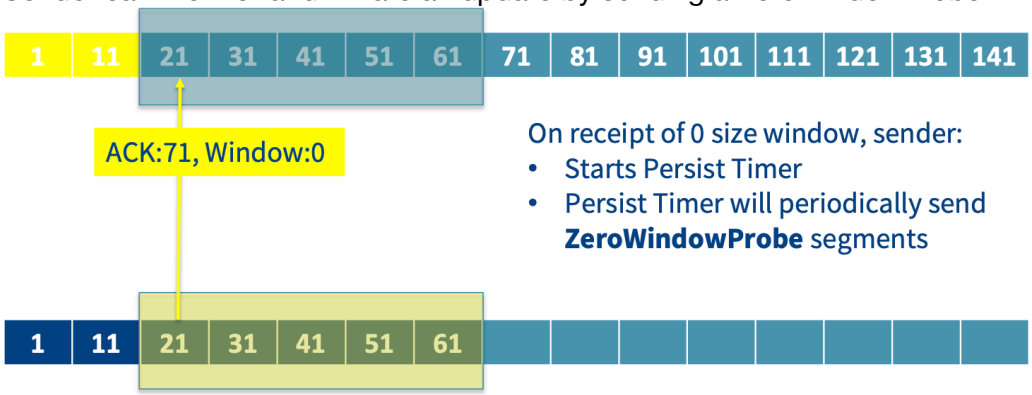
TCP Sliding Window – Window Update

Receiver can initiate an update by sending a WindowUpdate

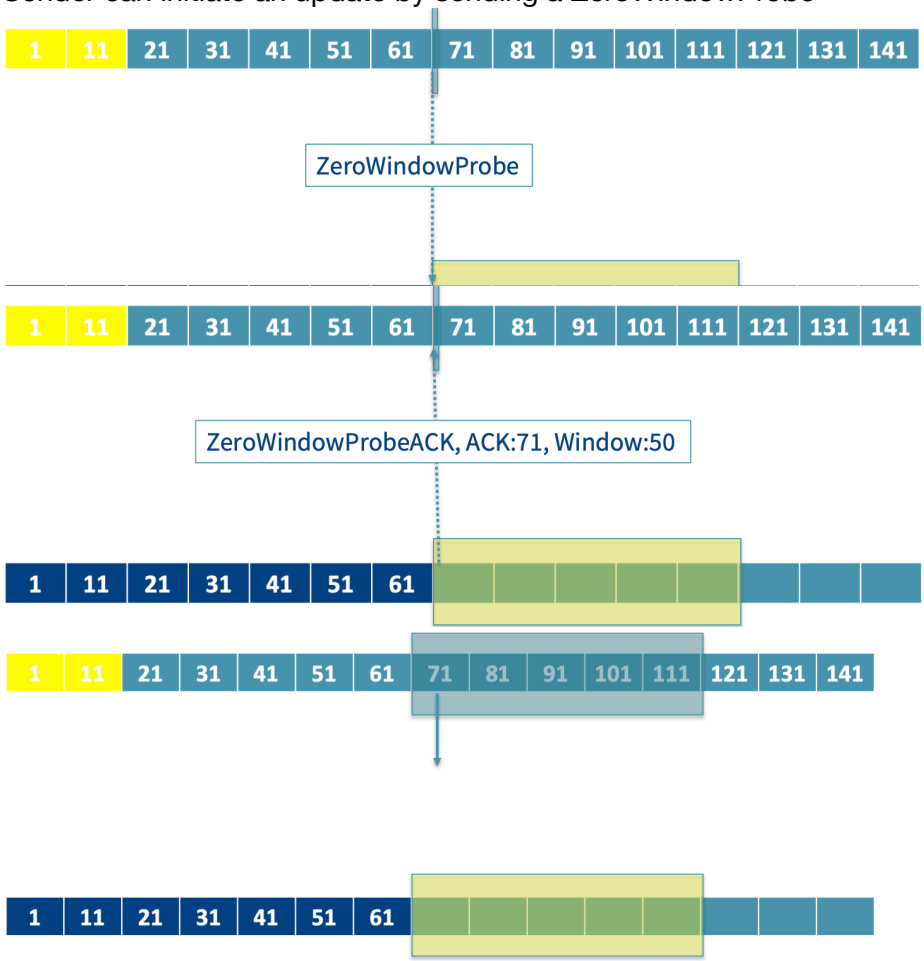


TCP Sliding Window – Persist Timer

Sender can monitor and initiate an update by sending a ZeroWindowProbe



Sender can initiate an update by sending a ZeroWindowProbe



TCP Congestion Control

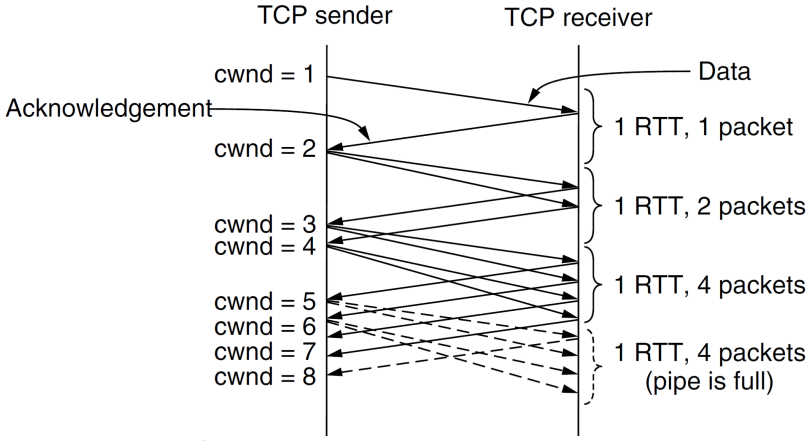
- When networks are overloaded, congestion occurs, potentially affecting all layers (the higher layers needs to wait data from lower layers, lower layers get overloaded)
- Although lower layers (link and network) attempt to ameliorate congestion, in reality TCP affects congestion most significantly because TCP offers methods to transparently reduce the data rate, and hence reduce congestion itself
  – Real-time control protocol also helps by changing video compression
- Original TCP (before Jacobson)
  – Initially, the receiver chooses a window based on its buffer size
  – if the sender is constrained to this size, then congestion problems will not occur due to buffer overflow at the receiver itself, but may still occur due to congestion within the network

Congestion Control Window

- Jacobson introduce CWND, the congestion window, in the same software update that introduce selective repeat.
- Additional window that is dynamically adjusted based on network performance to aid efficient transfer. If the congestion seems to be high, the window will reduce. If the congestion is low, the window will increase.
- The congestion window size is maintained by the sender, unlike the sliding window that is controlled by the receiver
  – Also only used at the sender.
  – No changes to packet formats to send additional field
    - Was a big consideration. Now a huge one.

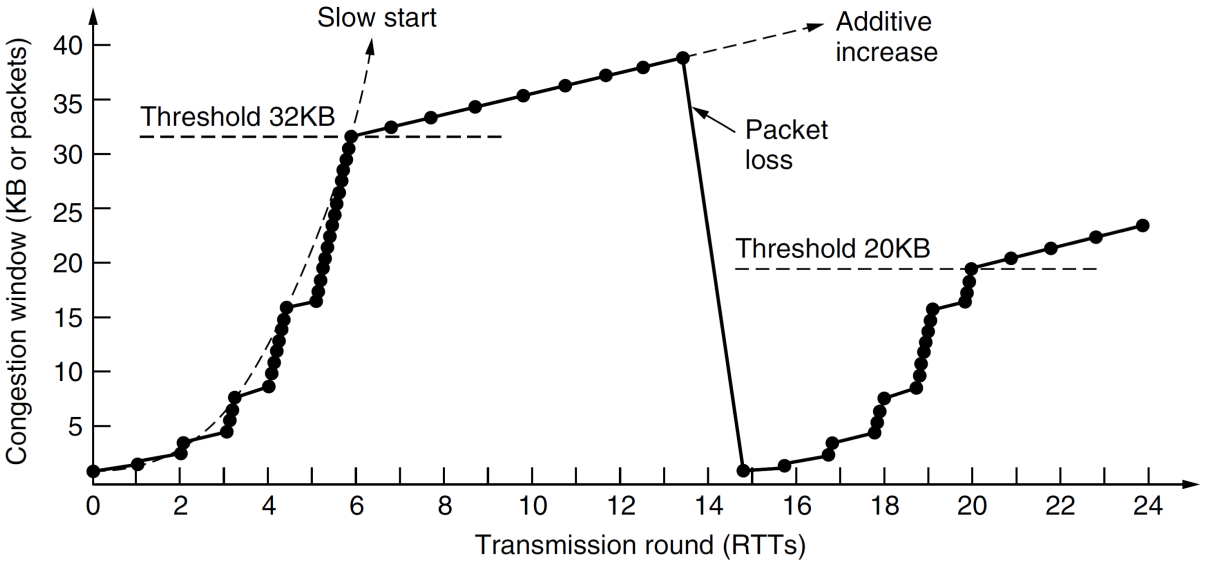
Incremental congestion control

- **Slow-start algorithm**
  – Initial rate is slow
  – Rate grows exponentially from there
- At first, CWND <- 2*maximum segment size
  – Sender transmits 2 segments
- If this segment is acknowledged, CWND++
  – Sender transmits 3 segments
- As each new segment is acknowledged, the congestion window is increased by one maximum segment size



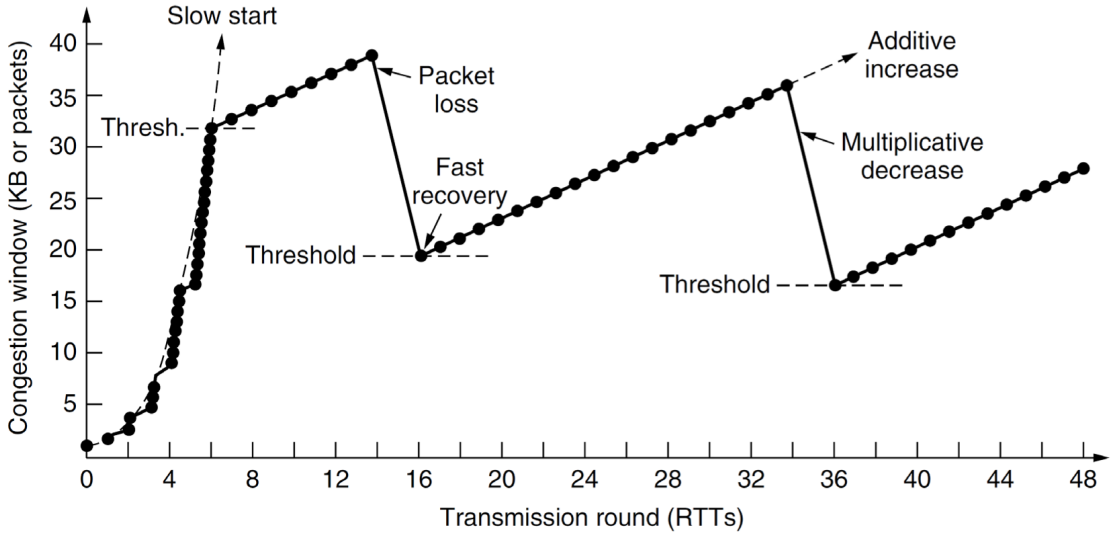
- Each full window of acknowledgements doubles the congestion window - which grows until
  – a timeout
  – It reaches a threshold, SSthresh

- Eventually too many segments would be placed onto the network causing congestion and timeouts
- Slow-start maintains a threshold: Slow Start Threshold **ssthresh**
- Slow start keeps increasing the size of the window until a timeout occurs or the threshold is reached
- If segment loss occurs the ssthresh is set to half the **current** congestion window size, and the process start again
- Once the threshold is reached the growth is slowed to linear, by adding 1 MSS to the congestion window for each successful ACK, known as additive increase
- Can also react to known lost segments via fast retransmission
- Known as TCP Tahoe (1988) after BSD release name



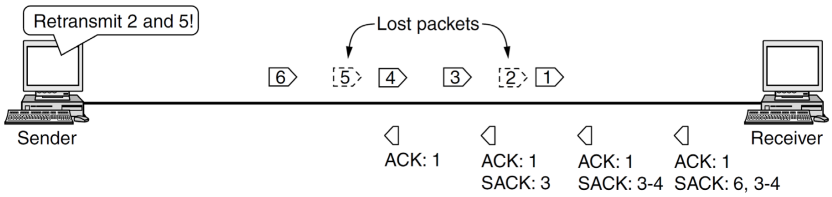
Congestion control optimisation

- Further optimisations have been made
  – Fast recovery using duplicate acknowledge counts (if three duplicated acknowledgments are received, apply fast recovery mechanism. Three duplicate acknowledgements is 4 acknowledgements in total. Duplicate acknowledgement: the acknowledgement has been acknowledged, and it doesn't change the window size and it doesn't send any data)
  – Starts from new ssthresh instead of original starting value, effectively avoiding the slow start phase and going straight to additive increase.
  – so the only thing TCP looks at is loss packet, not network congestion. If there's no packet loss, speed UUUUP!



Further Optimisations

- SACK (Selective Acknowledgements) – provides greater ability to track segments in-flight, by allowing up to 3 ranges of bytes received to be specified



- ECN Explicit Congestion Notification (ECE & CWR bits set during SYN)
  – lets a router say "I'm not gonna drop a packet, that will be a waste, but nonetheless, I'm getting congestion, please slow down". So instead of dropping a packet to indicate congestion, it can set a bit to indicate congestion. The sender will respond as if it lost the packet.
  – purpose: reduce the amount of wasted packets. If you say I'm getting congested before running out of buffer space, you don't need to send the packet halfway through the network just get dropped last link
  – Allows IP Layer to indicate congestion is occurring without dropping the segment by setting an ECN flag
  – Receiver indicates this to sender via ECE (ECN Echo) flag
  – Sender acknowledges this by setting the Congestion Window Reduced Flag (CWR), reacts as if a segment has been lost, without actually having lost it

Macroscopic model
- These packet-level rules affect many things we care about
  – Fairness between flows
  – Response to long round-trip times(RTTs)
  – Response to random packet loss
- It is useful to have an algebraic expression relating these quantities

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

Rate, fairness
- The window is sent ≤ once per round trip time (RTT), T
- Rate of TCP is W/T ≈ 1/T * sqrt(2/p)

1. For a given packet loss rate, longer RTTs get less rate
   – "RTT unfairness"
2. If RTT is small, TCP forces the packet loss rate to be high

- This formula is very approximate
  – Window only responds to one packet loss per RTT
  – Packet losses are clustered
- However, it gives two important insights

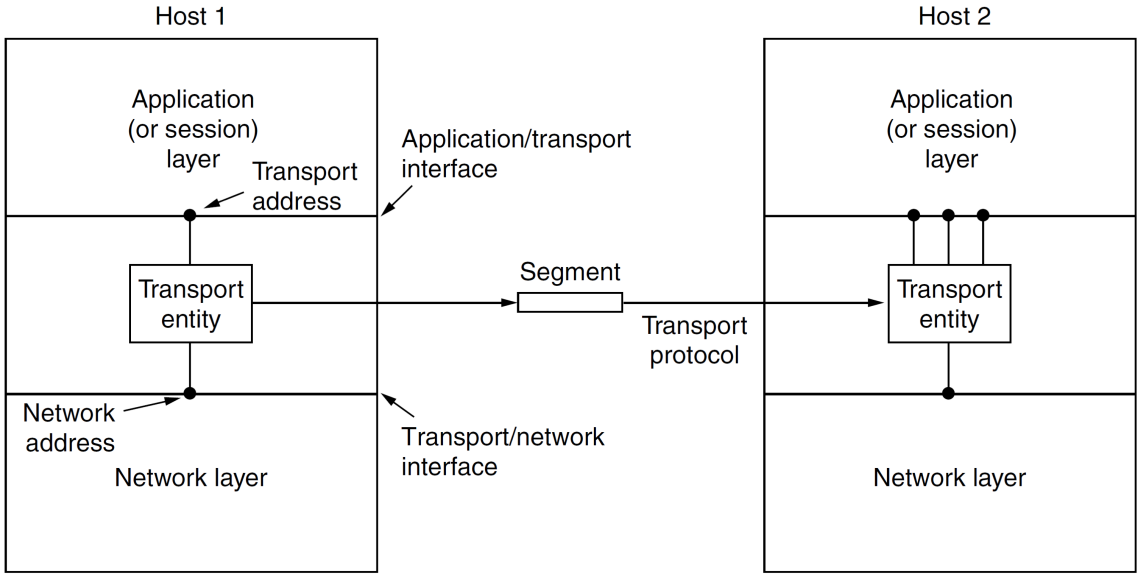
Presentation Layer
- OSI layer 6 to provide:
  – Encryption
  – Compression
  – Data conversion (e.g., mapping CR/LF to LF, .doc to .docx)
  – Mapping between character sets (ASCII/EBCDIC, now UTF-8/BIG5/...)
- These services haven't vanished: done by applications
- Why does IETF consider them "Application Layer"?
  – The protocol to negotiate encryption etc. is quite simple and separate from the algorithms
  – There aren't simple "common services" needed by all applications
  – The application is not in the kernel, and so much more flexible
  – "Layering violations": look at application layer, violates layering but efficient
- Closest thing to presentation layer: Real time protocol (RTP)

Session Layer
- OSI Layer 5 to provide
  – Authentication
  – Authorization
  – Session restoration
    - continue a failed download
    - log back in to same point in an online purchase
- Examples:
  – Remote procedure call (RPC)
  – Point-to-point tunneling protocol (PPTP)
  – Password (/Extensible) Authentication Protocol (PAP/EAP)
- Often used between protocols called* layer 2 and layer 3
  – *Layers are funny. Ethernet is always called "layer 2", but has many properties of layer 3, and even some of layer 4.

Transport layer
Role: provide services needed by applications, using services available by the network layer.

- Application needs:
  - Data is a stream of bytes
  - Data from one application is not mixed with that for another
  - Data arrives reliably (or we know when a packet has been lost)
  - Data arrives in order
  - Data doesn't arrive faster than we can handle
- Network provides:
  - Get packets from host to host…
  - …sometimes multiple copies
- The Transport layer services provide interfaces between the Application layer and the Network/ Internet layer.
- The Transport layer entities (the hardware or software which actually does the work e.g. OS kernel, processes, NIC) can exist in multiple locations. E2E from one location to another
- Services provide a "logical" communication channel between processes running on different hosts: if two processes on same host, it its considered to be 2 end points by transport layers
  – Connection-oriented
    - = Connection establishment, data transfer, connection release (TCP)
    - Like phone call
  – Connectionless: data transfer (UDP)
    - Like text messages

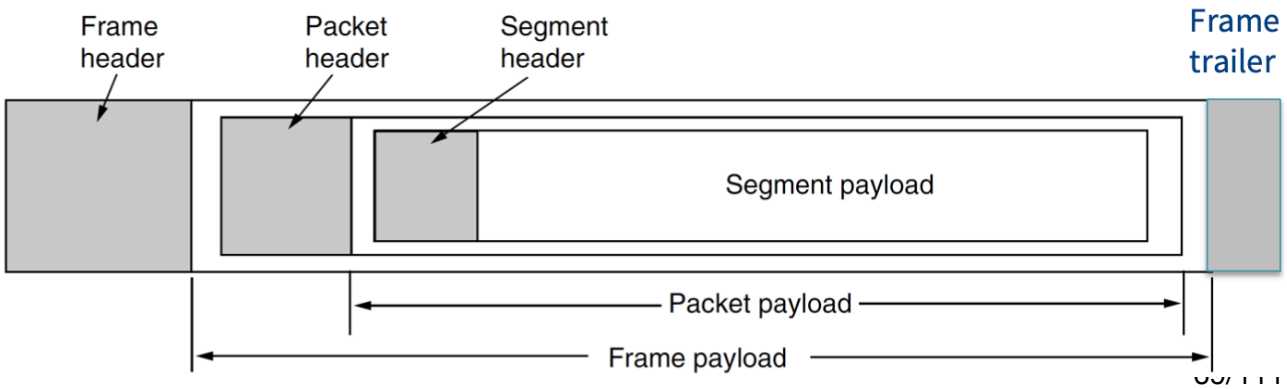
Transport entity illustrated (Tanenbaum)



Connection-oriented transport services (can) provide a reliable service on top of an unreliable network.

Transport layer encapsulation

- Abstract representation of messages sent to and from transport entities
- Encapsulation of segments (transport layer units) in packets (network/internet layer units) in frames (data/link layer units) segment payload may have a application header, UDP header



Transport layer services

- Terminology (not universal):
  - Segments – sent at the transport layer
  - Packets – sent at the internet/network layer
  - Frames – sent at the link/data link layer
- In the case of a **reliable** connection orientated service
  - Provides a notional “perfect” connection between two nodes
    - Doesn’t provide privacy, isochrony (preserving delay between packets)
  - Hides acknowledgements, congestion control, lost packets
  - This service is provided to the higher layers
- In the case of a unreliable connectionless orientated service (UDP is one of it)
  - Provides multiplexing between different processes

Related Question: Is it possible for an application to get reliable data transfer even when the application runs over UDP? If so, how?
Yes. The applican can implement its own retransmission scheme (transport layer functionality) on top of UDP. This is what is done by QUIC, which is a reliable transport layer that runs on top of UDP. In this context, the application (or QUIC) would be treating UDP as a network-layer protocol. It is common to find layers mishmashed like this, because it makes network administration easier. Another example is ”virtual LANs” (VLANs), which sit on top of the network layer (i.e., they send IP packets), but provide the same services to the higher layers as ethernet does.
One case in which this is useful is for using authentication protocols that expect to run on ethernet. Someone connected to one VLAN may be able to access resources (say a printer) on that VLAN, whereas someone else on the same physical LAN but a different VLAN could not. Similarly, someone could print to the printer from a different city, as long as they are on the same VLAN.
Another case in which this is useful is if a router is connected to many physical LAN segments, and an IP interfaces used to connect to another campus. By running a VLAN over the IP interface, the router configuration can be made more symmetrical, which makes it easier to maintain.
There are even “virtual wires”, in which IP packets carry bit streams, which represent actual ethernet frames.

Transport Layer Addressing

- Specification of the remote process to “connect to” is required at both the application and transport layers.
- Addressing in the Transport layer is typically done using port numbers (e.g. port 80: web server).
  - cf. Unix/etc/services, www.iana.org (well known ports) a process server intercepts inbound connections and spawns requested server and attaches inbound connection
    - cf. Unix /etc/(x)inetd
- Full address is a 5-tuple
  - (source IP address, source port, destination IP address, destination port, protocol)

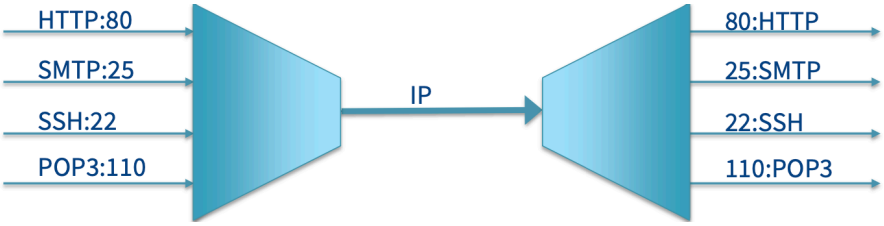
Related Question: Both UDP and TCP use port numbers to identify the destination entity when delivering a message. Give two reasons for why these protocols invented a new abstract ID (port numbers), instead of using process IDs, which already existed when these protocols were designed?
Here are three reasons. First, process IDs are OS-specific. Using process IDs would have made these protocols OS-dependent. Second, a single process may establish multiple channels of communications. A single process ID (per process) as the destination identifier cannot be used to distinguish between these channels. Third, having processes listen on well known ports is easy, but well-known process IDs are impossible.

Port allocations

- Port numbers can range from 0-65535 (16 bits)
- Allocated by Internet Assigned Numbers Authority (IANA) – (http://www.iana.org/assignments/port-numbers)
- Ports are classified into 3 segments:
  - Well Known Ports (0-1023)
    - 21 FTP
    - 22 SSH (secure shell)
    - 23 Telnet
    - 25 SMTP (super mail transport protocol)
    - 80 HTTP
    - 110 POP3 (mail)
    - 119 NNTP (like Facebook 40 yrs ago)
  - Registered Ports (1024-49151)
    - Also called "user ports" but still registered with IANA or similar body
  - Dynamic Ports (49152-65535)

Multiplexing /Demultiplexing

- Shortened to MUXING and DEMUXING
  - Multiplexing – combining multiple distinct streams into a single shared stream
  - Demultiplexing – splitting distinct streams out from a single shared stream



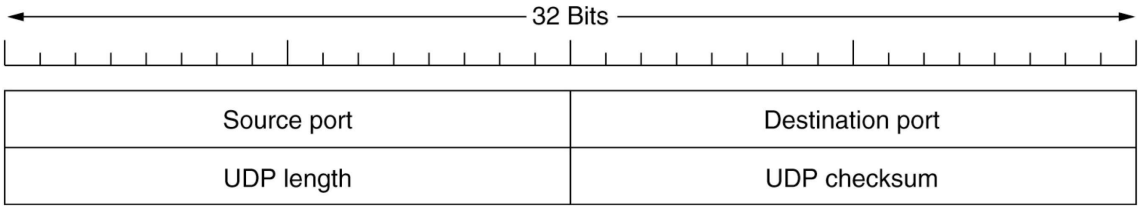
UDP – User Datagram Protocol

- The User Datagram Protocol provides a protocol whereby applications can transmit encapsulated IP datagrams without establishing a connection.
  - UDP transmits in segments consisting of a header followed by the payload
- UDP headers contain source and destination ports, payload is handed to the process which is attached to the particular port at the destination (using BIND primitive or similar)
- The main advantage of using UDP over raw IP is the ability to specify ports for source and destination pairs.
- Note: both source and destination ports are required - destination allows initial routing for incoming segments, source allows reply routing for outgoing segments.
- **Strengths and weaknesses** of UDP:
  - Strengths: multiplexing/de-multiplexing; **no delay** waiting to recover lost packets
  - Weaknesses: No flow control, error control or retransmission of bad segments
  - Conclusion: where applications require a precise level of control over packet flow/error/timing, UDP is a good choice. If you want someone do these control for you, you better use TCP

Related Questions: Suppose a process in Host C has a UDP socket with port number 6789. Suppose both Host A and Host B each send a UDP segment to Host C with destination port number 6789. Will both of these segments be directed to the same socket at Host C? If so, how will the process at Host C know that these two segments originated from two different hosts?
Yes, both segments will be directed to the same socket. This is because UDP receivers only use listening sockets, which are identified by a 3-tuple (protocol, local IP address, local port), instead of the 5-tuple used for a connection-oriented socket.
For each received segment, at the socket interface, the operating system will provide the process with the sender's IP address and port number to determine the origins of the individual segments. If the sender also has a UDP listening socket on its sending port, then the original receiver can send it a reply by sending a UDP packet with the original sender's IP address and port.

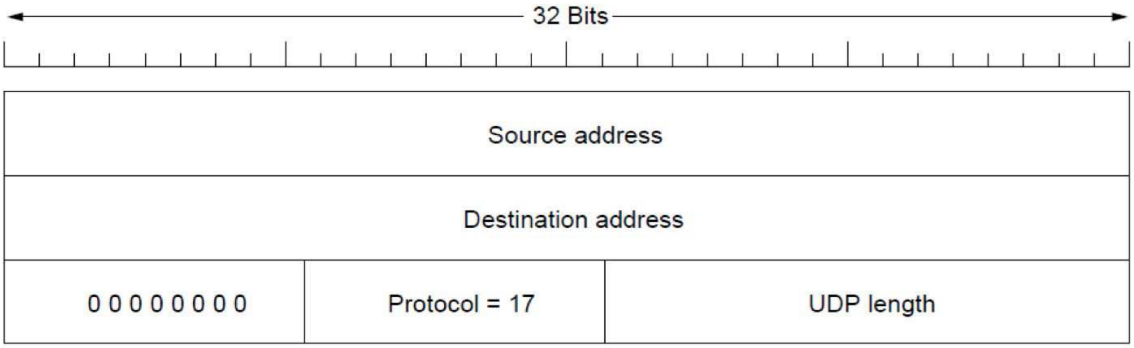
UDP header

- UDP header



  - UDP checksum tells if the packet has been corrupted, it's not needed if transporting over a reliable link protocol. Only used when part of the segment is lost part is received
  - UDP length: can be work out by the IP packet size told by IP packet
  - The two address are most important to support multiplexing
- The IPv4 pseudo-header included in the UDP checksum.

UDP – User Datagram Protocol

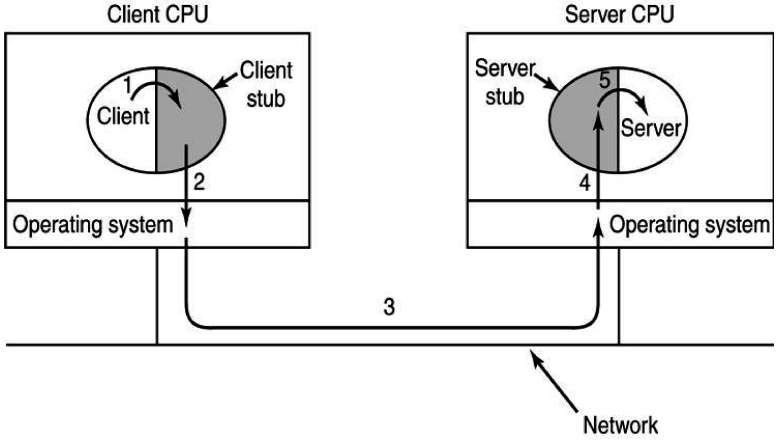


- Simple and efficient
- Suitable for some client – server settings
  - Clients sends a short request to the server, expects a short response
  - If that does not occur (request or response is lost) client timeouts and resends
  - Simple to code, and fewer messages, one in each direction
  - DNS is a good example
- Also suitable for real-time services (e.g., VoIP)
  - If a packet is lost, we don't want to wait for it to be resent
  - Loss concealment: fill in the time with our "best guess" sound

Related Question: Why does UDP exist? Would it not have been enough to just let the user processes send raw IP packets?
IP packets contain IP addresses, which specify a destination machine. Once such a packet arrived, how would the network handler know which process to give it to? UDP packets contain a destination port number. This information is essential so they can be delivered to the correct process.
UDP is sometimes called a "process-to-process", as distinct from a "host-to-host" protocol like IP. (The terms "host-to-host" and "process-to-process" are not very standard, unlike "point-to-point" and "end-to-end" which are widely used.)

Remote Procedure Calls

- RPC – Remote Procedure Calls
  – Allow calling procedures on a remote server as if they are local to the client (e.g. lookup a database)
  – Hides the networking aspects from the programmer
- RPC isn't a single protocol/API. Dozens of variants exist.
- How it works abstractly:
  – Client process on Machine A calls procedure on Machine B
  – Process on machine A is suspended, whilst execution of the procedure takes place on Machine B
  – MachineB responds with result to Machine A, which then continues processing
- To hide the networking, the client and server must be bound to respective stubs
  – Client stub – operates in the client address space (part of client's process, often from library)
  – Server stub – operates in the server address space
- From the perspective of the client and server processes all the calls are local
- Parameters can be passed and returned
  – Marshalling – convert the in-memory data structure to a form that can be stored or transmitted (a portable format that standardized by client and server stub)
  – Unmarshalling – covert the stored or transmitted data into an in-memory data structure



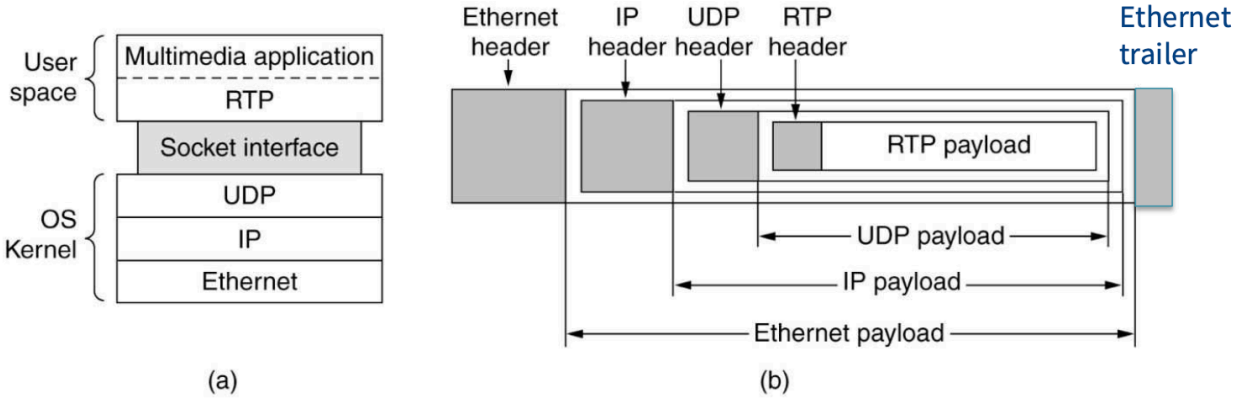
- Conceptually simple, but many **challenges** exist
  – Cannot pass pointers easily – client and server are in different address spaces (cannot pass a point of a tree/list, but can put the whole tree)
    - Possible to marshal and unmarshal underlying value and create a pointer in each address space
      – Does not work for complex data structures
  – Weakly typed languages like C can present problems
    - e.g. unknown array sizes
  – Unable to deduce parameter types
  – Global variables are not shared
- UDP can be a good choice for RPC
  – Requires some additional scaffolding
    - Resending after timeout if no reply is received
      – a reply constitutes an acknowledgement of the request
    - Handling large parameter sizes that need to be split across multiple UDP segments
  – Caution must be used if operation is not idempotent
    - e.g., incrementing a bank balance
- TCP can be used for non-idempotent operations

RTP – Streaming and VOIP

- Real-Time Transport Protocol (RTP)
- Which layer is RTP at?
  - Runs in user space, uses UDP from the transport layer -> Application layer
  - Generic protocol that provides services to applications -> Transport layer
  - (Neither – Presentation layer!)
- RTP multiplexes several streams into a single stream of UDP

UDP Example Use

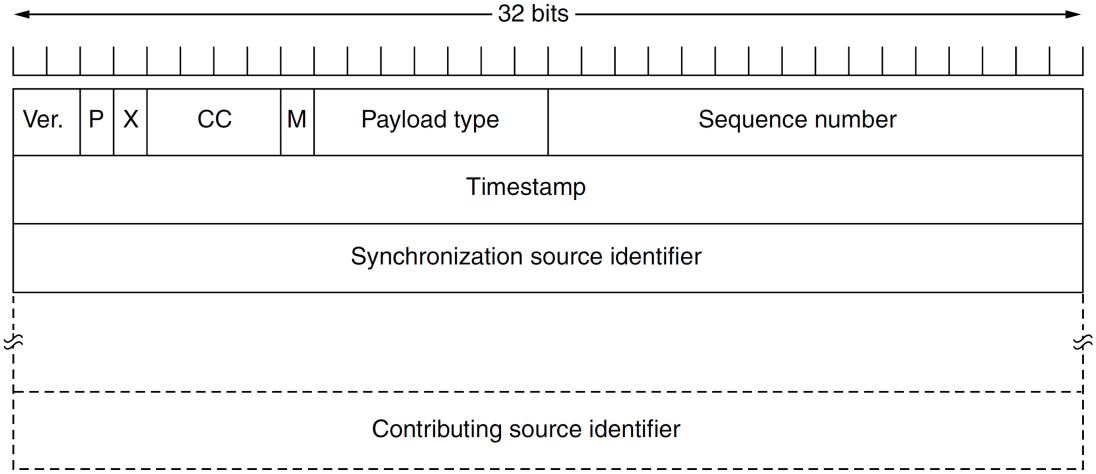
The position of real-time protocol in the protocol stack



Packet nesting

RTP Header

- Payload type – encoding used (MP3, etc.) – can vary each time

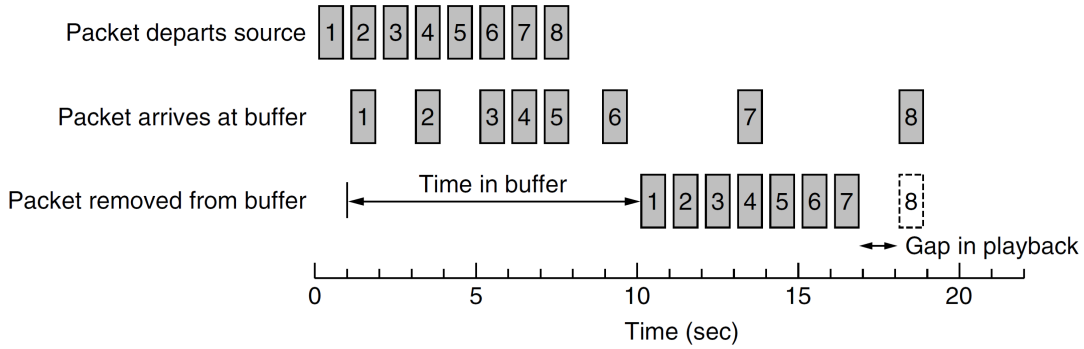


- Sequence Number – counter incremented on each packet
- Timestamp – Source controlled relative to start of the stream, the packets may come before we actually use them, so we need to know how long to wait

Real-time Transport Control Protocol (RTCP)

- **Control protocol** for RTP
  – Handles feedback, synchronization, and UI
- Feedback to source
  – Delay, jitter, bandwidth, congestion
  – Used by encoder to adaptively encode to suit network conditions
  – In multicast settings, feedback is limited to small percentage of media bandwidth
- Synchronization – Where different streams use different clocks/have different drift
- UI – naming sources to show who is on a conference call
- (Another network model: “Control plane” is a stack parallel to the “data plane” stack.)

RTP Playback



- Jitter – variation in delay of packets
  – Buffer at receiver to counter it
- Packet 8 too late, can wait or skip, depending on application
- Size of buffer is also application specific (VOIP = small buffer)

- Memcached Reflected DDoS Attacks
  – Distributed memory object caching – speeds up dynamic websites by caching database queries
  – Should never been configured externally facing
- Small UDP request made to memcached server with fake source IP
- Memcached responds with up to 50,000 times the data
  – 203 byte request results in 100MB response

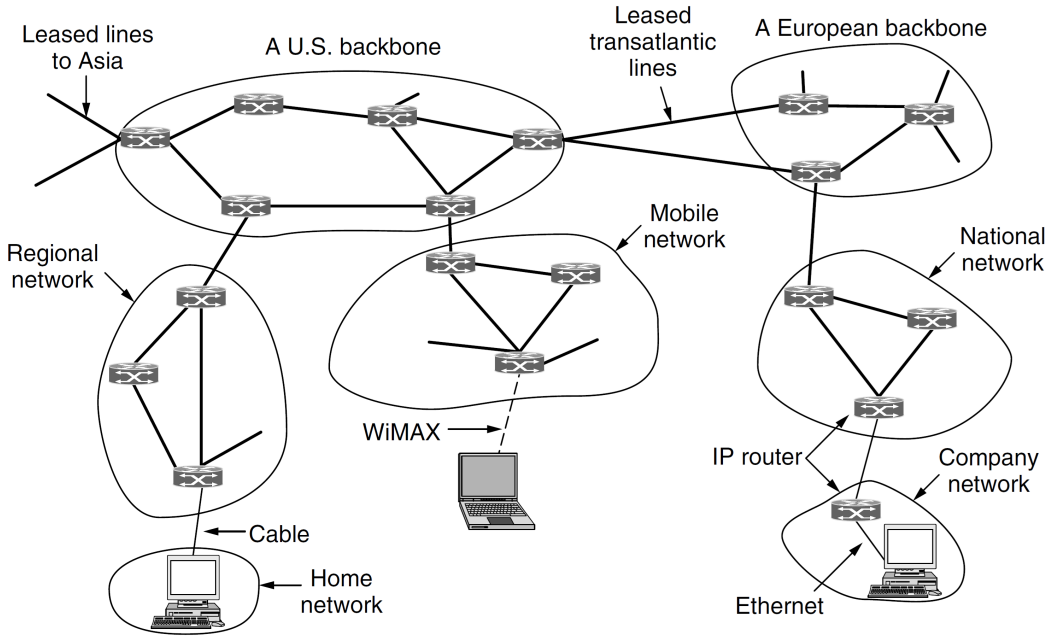
Week10

Internet (network) Layer

- Role: get data from the source all the way to the destination
  - May not be in a single hop (point-to-point link)
- Traffic must be routed efficiently
  - This is performed by network devices called routers
- Nodes must be given names (addresses)
- “Internet” is a network of networks
  - “Internet Layer” is a sublayer at the top of the network layer
- In an internet, the source and destination may be in different networks. A “hop” is a whole network. (Note the ‘i’ in internet is small)

Internetworking and Routing

• Connecting multiple networks, a national backbone connects with other national backbones. It’s multi-point to multi-point

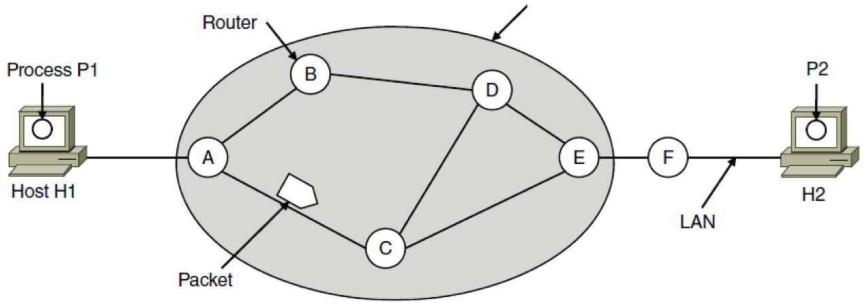


Network Layer

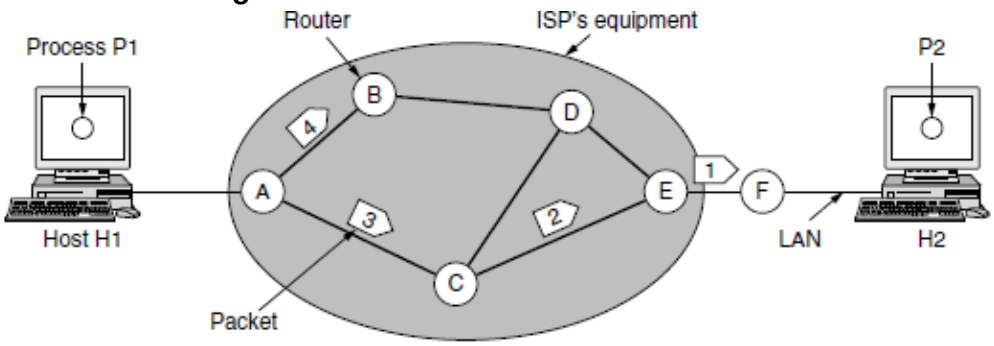
- Most Network Layer code runs on routers
- We will refer to the protocol data units as packets
- What types of services does the Network Layer provide?
  - Connectionless
    - Packet switching (Internet Protocol - IP)
    - Minimum required service: “send packet”
    - Called “datagram” network
  - Connection-oriented
    - (virtual) Circuit Switching
      - Asynchronous Transfer Mode – ATM
      - MultiProtocol Label Switching MPLS
    - Called “virtual circuit” network
    - These usually act as a single “link” of an IP network

Store-and-Forward Packet Switching

- The internet is a packet switched network
- Host H1 wants to send a packet to H2
  1. Transmits it to the nearest router (A)
  2. The packet is buffered while it is arriving, and the checksum is verified
  3. If valid, the packet is stored until the outgoing interface is free
  4. The router forwards the packet onto the next router in the path
  5. Repeat 2-4



Packet forwarding – Connectionless



A's table (initially)

Dest.	Line
A	–
B	B
C	C
D	B
E	C
F	C

A's table (later)

A	–
B	B
C	C
D	B
E	B
F	B

C's table

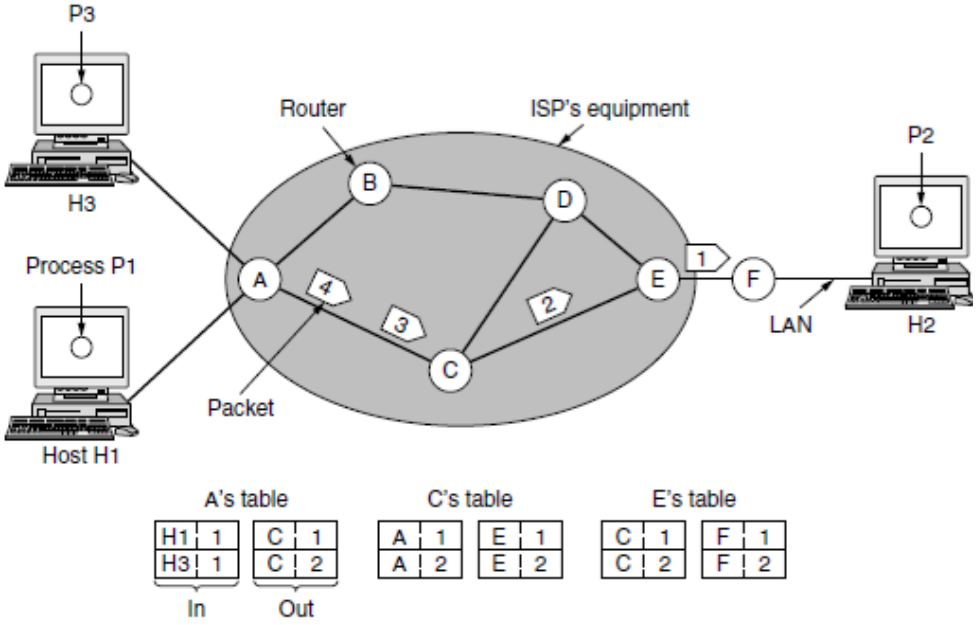
A	A
B	A
C	–
D	E
E	E
F	E

E's table

A	C
B	D
C	C
D	D
E	–
F	F

Paths can change for packets in the same transport layer connection (A's table before and after)
The tables are called Forwarding tables (also called routing tables)

Packet forwarding – Connection-oriented



Usually sending everything on same path.
Benefit: there are millions of hosts on the Internet, so we need a lot of bits to specify the source destination
But the number of virtual sockets running over a given link could easily just be hundreds, so we have much smaller identifiers for which socket we are on, so we have smaller packets.
Forwarding table:
"In": connection ID
"Out": next hop and new connection number
H1,H3 are two different sockets, so 1 for each. When the first socket comes to C, numbered with 1. When the second come to C, number it with 2.
Connection number is local to a hop

Internet Layer

• Connection-oriented vs Connectionless

Issue	Datagram Network	Virtual Circuit
Type	Connectionless	Connection-oriented
Addressing	(-) Each packet has full source and destination	(+) Each packet contains a *short* VC number
State	+ Routers do *not* hold state information about connections	- Each VC requires router table space. - Router *reboots* a problem.
Routing	Each packet independently	Defined at set-up
Quality of Service	- Difficult	+ Easy if enough resources
Congestion control	- Difficult	+ Easy if enough resources
Link failure recover	+ Simple	- Extra work

Related Question: Datagram networks route each packet as a separate unit, independent of all others. Virtual- circuit networks do not have to do this, since each data packet follows a predetermined route. Does this observation mean that virtual-circuit networks do not need the capability to route isolated packets from an arbitrary source to an arbitrary destination? Explain your answer.
Virtual circuit networks most certainly need this capability in order to route connection setup packets from an arbitrary source to an arbitrary destination.

MultiProtocol Label Switching

- Widely deployed Virtual Circuit (connection-oriented) Network Layer Protocol (below the internet sublayer)
  - MPLS network is one IP hop
- Primary purpose is Quality of Service
  - Prioritising traffic
  - Service Level Agreements for network performance
  - Reliable connectivity with known parameters
- Popular with businesses that want to connect multiple sites and phone companies carrying voice traffic
- Expensive: Price roughly 20-100 times more per Mbps than a standard internet connection. (Cost is much more similar.)
  - https://www.networkworld.com/article/2222196/why-does-mpls-cost- so-much-more-than-internet-connectivity-.html

Quality of Service

- Why is Quality of Service (QoS) important?
  - Not all services are equally important or robust to network delay
    - VoIP(Voice over IP) vs file downloads (VoIP is more robust), VoIP is more robust
    - VPN connections vs web browsing, VPN is more robust
- Within your own network, or within a single administered network (ISP), services can be prioritised
  - Own network - typically explicitly (prioritisation)
  - Shared network - typically implicitly (ISP traffic shaping)
- In the case of explicit prioritisation, the Differentiated Services header can be used to define classes of traffic
- Useful in an office building with a network that carries both internet and telephony traffic

Network Layer

- Be aware that
  - connection-oriented services exist, and
  - the concept of Quality of Service is important
- Our focus will be on the connectionless Internet Protocol that forms the backbone of the internet

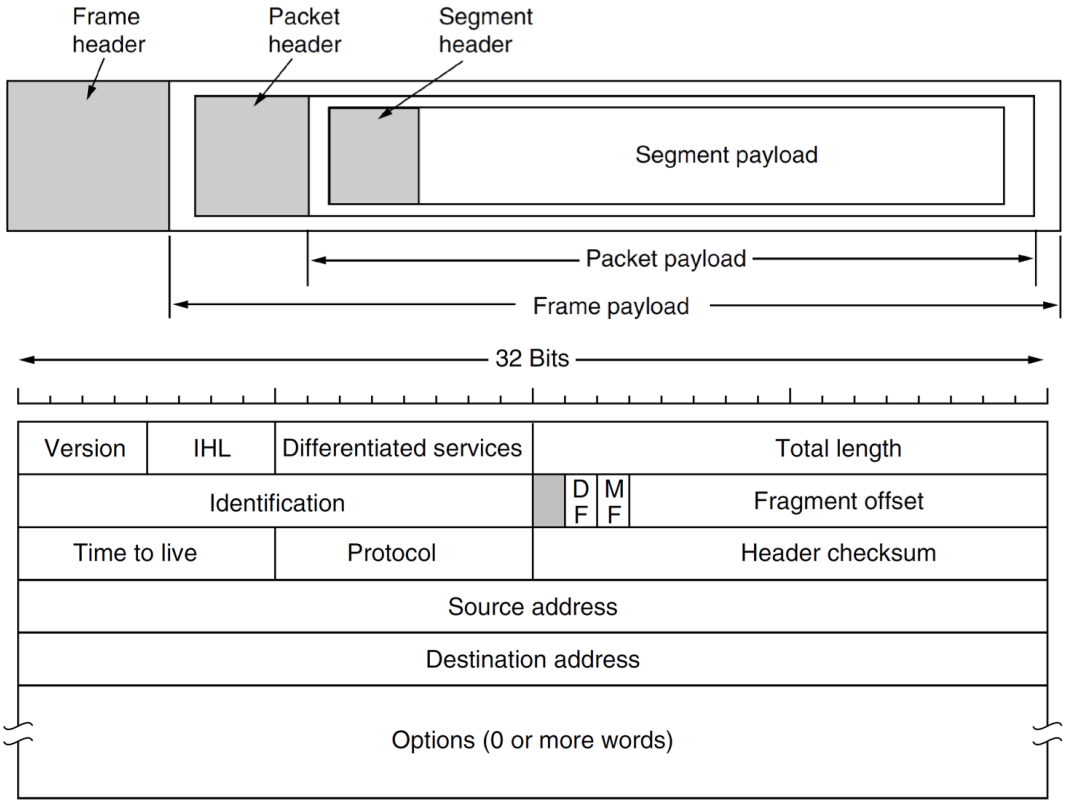
Internet Protocol

- Designed with a number of principles in mind, including:
  - Something that works OK is better than an ideal standard “in progress”
    - Standard of “OK” has gone up as the Internet has become vital
  - Keep it simple, simple as possible but not simpler – Occam’s Razor
  - Be strict when sending and tolerant when receiving: if you are unsure whether send this or not then don’t send it. If received something and it doesn’t meet standard, don’t throw out, have a guess about what is intended
    - E.g., web browsers handle pages with invalid HTML
  - Make clear choices – don’t have different approaches in a standard
  - Avoid static options and parameters – negotiate them at runtime
  - (New principle: Think about scalability)
- “Best effort”, not guaranteed performance, the higher layer guarantee whether resend or not
- Responsible for moving the packets through the various networks from source to destination host
- Multiple paths through the network – Important for redundancy
- Routing algorithms are used to determine best path
- Nothing guaranteed – just “best effort”

IP Version 4 Protocol (IPv4)

Field	Usage
Version	Protocol version 4 (this field is also in IPv6)
IHL	Header length in 32 bit words; min 5, max 15
Differentiated services	6 bits for service class, 2 bits for congestion control (ECN)
Total length	Including payload, max 65,525
Identification, DF, MF, Fragment Offset	Used in the handling of fragmentation
Time to live (TTL)	Countdown of hops, at zero packet is discarded
Protocol	Transport layer service (TCP/UDP/SCTP/DCCP/etc.)
Source and Destination	IPv4 address
Options	Rarely used and poorly supported

The packet header is also known as **IP header**



Related Question: The IP packet header includes a time-to-live field that is decremented by each router along the path. Why is the time-to-live field necessary?
A packet may get stuck in a forwarding loop (e.g., due to a router configuration mistake). By decrementing the TTL field at each hop, and discarding the packet when the TTL reaches 0, the network prevents the packet from cycling in a loop indefinitely. Otherwise, the packet would consume excessive resources, or even escape the loop eventually and reach the destination much later (running the risk that the packet is mistakenly viewed as part of a more recent transmission with the same IP addresses and TCP/UDP port numbers).

IPv4 Addresses

- 32-bit number
- Expressed in decimal notation, each byte is shown as a decimal, separated by a period, 172.22.44.10 (0xAC162C0A)
- 0.0.0.0 is lowest, 255.255.255.255 is highest
- Overall IP allocation responsibility of Internet Corporation for Assigned Names and Numbers (ICANN) by delegation to IANA and Regional Internet Registries (RIR's)
- IP addresses are given to interfaces not hosts, i.e. a host with multiple network cards will have multiple IP addresses
  - No two interfaces can have the same address, but it is possible for an interface to have multiple addresses (such as a link-local address and a global address), or for an interface not to be connected to the Internet and have no IP address, such as if it is an MPLS interface.
- Supply of IPv4 addresses has basically been exhausted

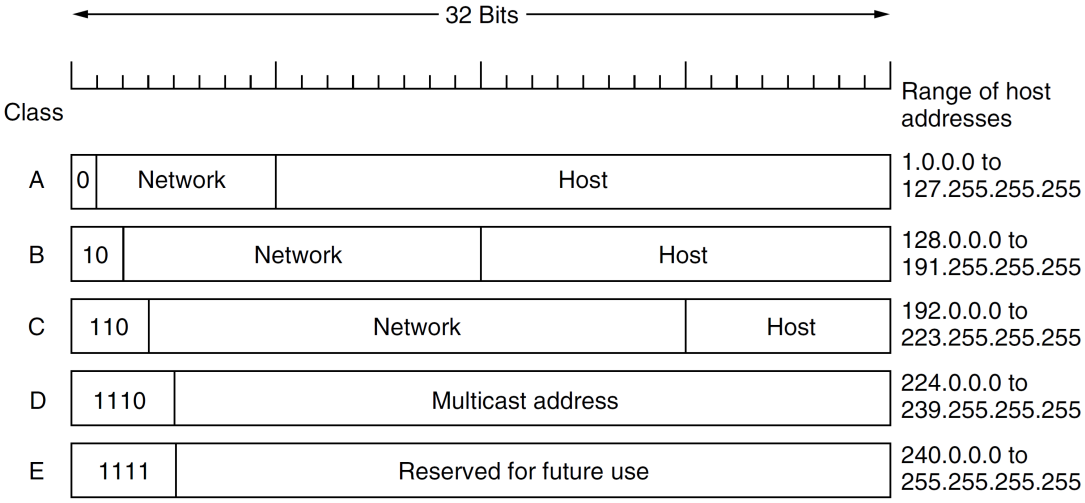
Types of address

- Unicast: One destination ("normal" address)
- Broadcast: Send to everyone
  - Unicast and Broadcast were thought to be the only types
- Multicast: Send to a particular set of nodes
  – Used for streaming video of live events
- Anycast: Send to any one of a set of addresses
  – Used for database queries, like DNS, NTP
- Geocast: Send to all users in a geographic area
  – "Location aware"services
  – Send ad to those in. store
  – Send warning to those near a hazard
  - Not widely used. Just to show that the above aren't all

IPv4 Addresses – Classes

(no longer used, but spoken of)

- Originally IP addresses were allocated based on classes
- Routing was performed based on the class, which could be derived from the first part of the address



IP Addresses – CIDR

- Classes simplify routing
  – size of "network" field is implicit in the address , the 0, 10, 110 represents 1 byte, 2 byte, 3 byte for the network address
- Wasteful. Networks often much bigger than needed
  – Network with 260 nodes must be class B with 65,536 addresses
- Classless InterDomain Routing
  – Each interface/route explicitly specifies which bits are the "network" field
  – Network with 260 nodes only needs 9 bits for "host" field, not the 16 bits
  —>512 addresses
  —>Can have 128 times as many of these networks as class B networks
- Hierarchical - encodes the network and host number
  – Network in top bits
  – Host in bottom bits
- Assigned to networks in blocks, the network part will be the same for all hosts on that network
  – A network corresponds to a contiguous(adjacent) block of IP address space, called a prefix
  – Prefixes are written as the lowest IP address followed by a slash and the size of the network portion
  - 192.0.2.0/24 means 24 bits for the network address, the 0 before slash will be host address up to 255.
  - 192.0.2.0/23 means only 23 bits of these other network address, that means all the bits in the first byte, all the bits in the second byte and only the most significant 7 bits in third byte will be the network address, the least significant bits of the third byte are part of the host identifier.

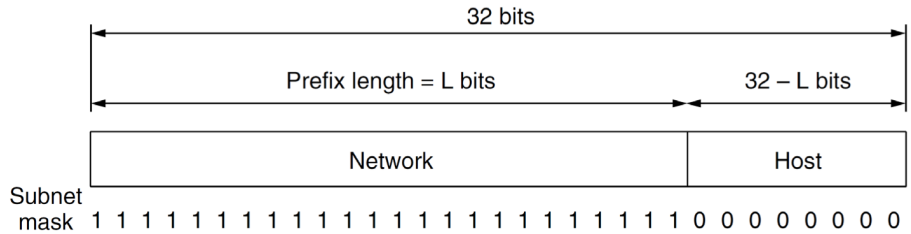
Related Question: A router has the following (CIDR) entries in its routing table:

Address/mask	Next hop
135.46.56.0/22	Interface 0
135.46.60.0/22	Interface 1
192.53.40.0/23	Router 1
default	Router 2

For each of the following IP addresses, where does the router send the packet if a packet with that address arrives? (a) 135.46.63.10 (b) (for exam study: 135.46.57.14)
(c) (for exam study: 135.46.52.2) (d) (for exam study: 192.53.40.7) (e) (for exam study: 192.53.56.7)
(a) Interface 1 (b) Interface 0 (c) Router 2 (d) Router 1 (e) Router 2

IP Addresses - Blocks

- In the case of 192.0.2.0/24
  – 24 bits are for the network 192.0.2.0
  – Leaving 8 bits for hosts – up to 256 addresses
- In the case of 10.0.0.0/8 (reserved private block)
  – 8 bits are for the network 10.0.0.0
  – Leaving 24 bits for hosts – up to 16,777,216 addresses
- Can also be written as a subnet mask, a binary mask of 1's
  – In the case of /24: the subnet mask is 255.255.255.0
  - in case of /23 the subnet mask is 255.255.254.0
  - in case of /22 will be 255.255.252.0



Related Question: A network on the Internet has a subnet mask of 255.255.240.0. What is the maximum number of hosts it can handle?
The binary representation of the subnet mask consists of 20 ones followed by zeroes. This means that the mask is 20 bits long, so the network part is 20 bits. The remaining 12 bits are for the host, so 4096 (2^{12}) host addresses exist. (Technically, the first and last addresses are reserved as the network address and broadcast address, and so 4094 addresses can be used for actual hosts.)

Related Question: A large number of consecutive IP addresses are available starting at 198.16.0.0. Suppose that four organizations, A, B, C, and D, request 4000, 2000, 4000, and 8000 addresses, respectively, and in that order. Each request is given the lowest-numbered subnet available for it. For each of these, give the first IP address assigned, the last IP address assigned, and the mask in the w.x.y.z/s notation.

A: 198.16.0.0 - 198.16.15.255 written as 198.16.0.0/20
B: 198.16.16.0 - 198.16.23.255 written as 198.16.16.0/21
C: 198.16.32.0 - 198.16.47.255 written as 198.16.32.0/20
D: 198.16.64.0 - 198.16.95.255 written as 198.16.64.0/19

(a) 4000 is slightly under 2^{12}. A /20 network has 2^{12} hosts (because host represents the number of devices, always adds up to 32. Bit 20 network is midway of 3rd octet) 198.16.0000|0000.00000000 → 198.16.0.0/20, 198.16.0.0 - 198.16.15.255 Start address has host bits 0, end address has host bits 1

(b) 2000 is just under 2^{11}. A /21 network has 2^{11} hosts. Now, 0000 prefix is used, so we cannot have 00000 or 00001 198.16.00010|000.00000000 → 198.16.16.0/21, 198.16.16.0 - 198.16.23.255

(c) 4000 is just under 2^{12}. A /20 network 2^{12} hosts. Now, 0000 prefix is used, a part of 0001 is used 198.16.0010|0000.00000000 → 198.16.32.0/20, 198.16.32.0 - 198.16.47.255

(d) 8000 is just under 2^{13}. A /19 network 2^{13} host Now, 000 and 001 prefix are partially used 198.16.010|00000.00000000 → 198.16.0.0/19, 198.16.64.0 - 198.16.95.255

IP Addresses – Prefixing

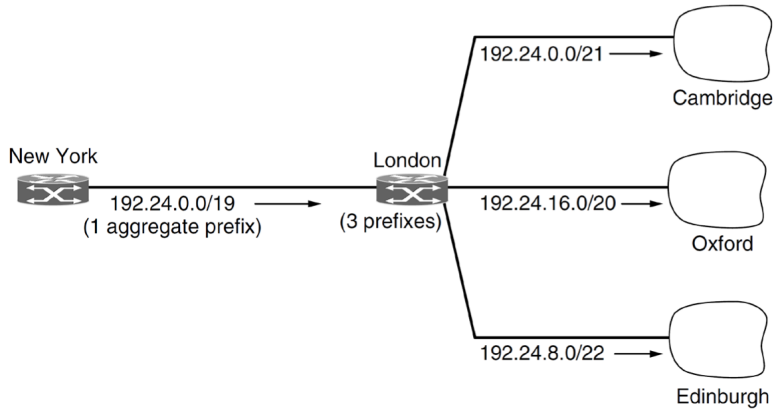
- Network number = network mask (bitwise-AND) IP address
- This is crucial for efficient routing on the internet
  – Since networks are assigned in blocks, intermediary routers need only maintain routes for the prefixes, not every individual host
  – Only when the packet arrives at the destination network does the host portion need to be read

And finally...

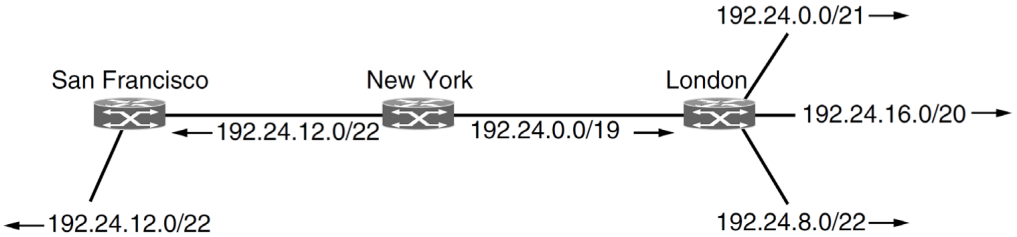
- One of the disadvantages of a hierarchical address space is that it can be quite wasteful if not carefully assigned, i.e. large parts of the address space unused if blocks badly assigned
- Combined with the scarcity (exhaustion) of available IP addresses, they have become a valuable commodity
- Not intended to be sold, should really be returned to the assigning body for reallocation
- Early adopters of IPv6 were able to sell their IPv4 address space at a premium

IP Addresses – Route aggregation

- Aggregation is performed automatically
- Currently it roughly halves the size of the routing table

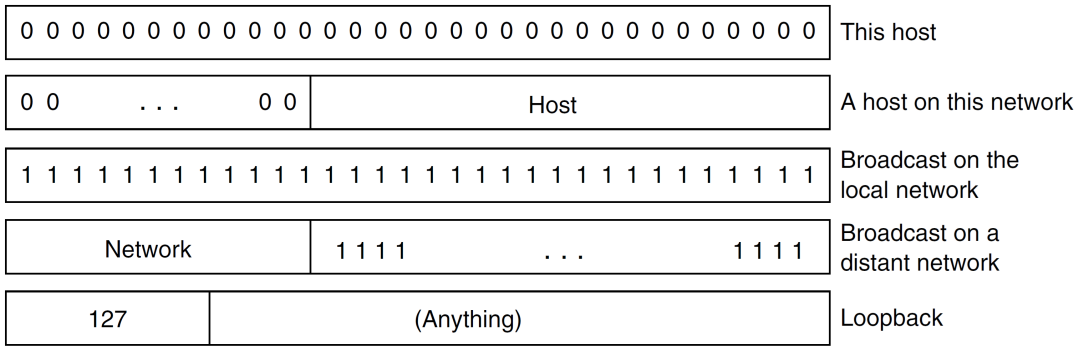


- Prefixes can overlap, in which case the longest matching prefix is selected



Special IP Addresses

Private address range	Prefix & Mask	Available Addresses
10.0.0.0 – 10.255.255.255	10.0.0.0/8 (255.0.0.0)	16,777,216
172.16.0.0 – 172.31.255.255	172.16.0.0/12 (255.240.0.0)	1,048,576
192.168.0.0 – 192.168.255.255	192.168.0.0/16 (255.255.0.0)	65,536
Link local / zero config		
169.254.0.0 – 169.254.255.255	169.254.0.0/16 (255.255.0.0)	65,536

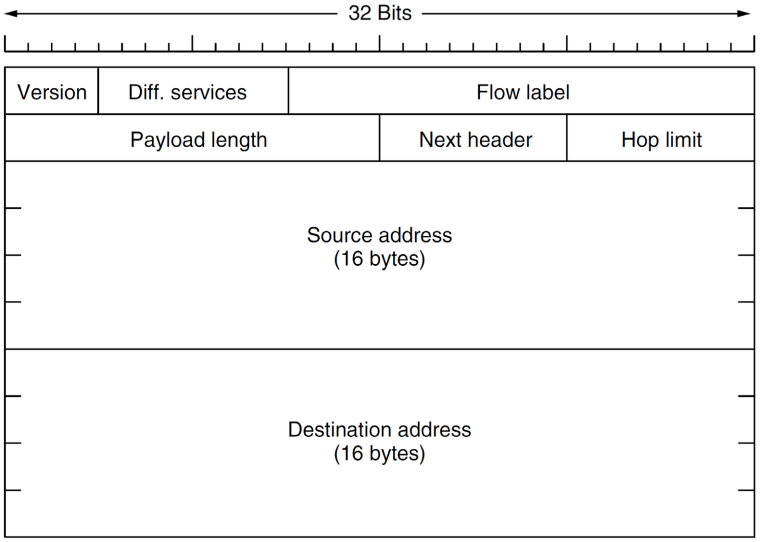


IPv6

- Designed over 20 years ago to address the problem of exhausting the IPv4 address space
- Whilst solving that problem some other changes were made
  - Simpler header – allows faster processing
  - Improved security – now back-ported to IPv4
  - Further Quality of Service support
- IPv6 addresses are 128 bits
  - Unlikely ever to run out...
  - …unless new wasteful allocation schemes are used

IPv6 Header

Field	Usage
Version	6
Differentiated services	6 bits for service class, 2 bits for congestion control (ECN)
Flow label	Pseudo-Virtual Circuit identifier
Payload length	Bytes after the 40 byte header
Next header	Used to specify additional headers or Protocol (TCP/UDP)
Hop limit	Same as TTL (Time To Live)
Source and Destination	16 bytes IPv6 addresses



IPv6 Addressing

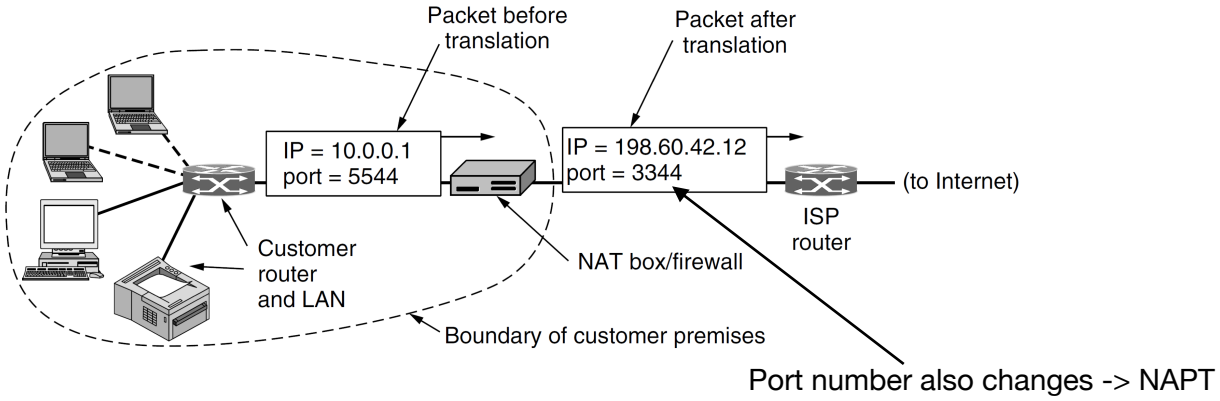
- Written as 8 groups of (up to) 4 hex digits
  - 8000:0000:0000:0000:0123:4567:89AB:CDEF
- Can be optimized by stripping one group of consecutive 0's
  - 8000::123:4567:89AB:CDEF
- Backwards compatibility with IPv4 is achieved with
  - ::ffff:192.31.2.46 (note the mix of hex with decimal)
- Still not widely deployed
  - Google measures IPv6 requests – still around 34% globally, (~24% in Australia)
- Fairly widely supported – likely to see a sudden growth in the next few years

IPv4 address scarcity

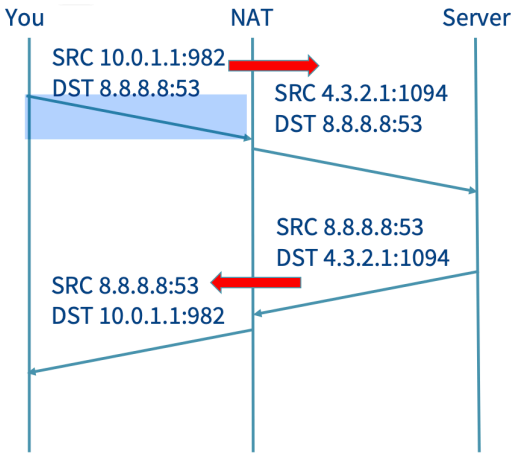
- As IP addresses became scarce, methods for handling many more clients were developed
- Whilst IPv6 would solve the problem a stop gap was needed
- Private addresses
  – Many hosts in a company only need internal access
  – “Private” subnets 192.168.0.0/16, 172.16.0.0/12, 10.0.0/8
  – Can be reused: unique address within organization, not globally
- Intention: “Application layer proxies” to access outside services

Network Address Translation (NAT)

- Each customer/home is assigned one public IP address
  – Businesses might be issued a few
- Internally hosts/interfaces are issued Private IP addresses
  – Recall 10.0.0.0/8-10.255.255.255 (as an example)
- Internal IP addresses are used for communicating among hosts in the Local Area Network (LAN)
- They must never be used on the public internet
- When a packet is heading out of the network (to the ISP) the internal address is translated to the public IP address
- Limitation: if the network is large behind NAT box, we might have more than 64,000 TCP connections in total that want to go through the NAT box because we only have 2^16 TCP connections from this IP address, when they do this they need to retranslate the IP address and port number and recalculate the checksum and modify the headers.



How NAT works:
– Assumes TCP/UDP (some exceptions), in particular the locations of source and destination port fields
– NAT box replaces IP source address (10.x.y.z) with public IP address
– TCP source port replaced with index of entry in NAT translation table
  • One of 65,536 entries (16 bits – same as TCP port field)
  • Each entry contains original IP address (private IP) and original source port number
– IP and TCP checksums are recalculated
– When a packet arrives from the internet at the NAT box it looks up the destination port from the TCP header in the translation table
  • Retrieves original source port and source IP address, updates headers and checksums and sends to the internal host



Criticisms of NAT
– Breaks end-to-end connectivity: An interface in the private network can only receive packets once it has sent packets out and created a mapping (some exceptions) So the NAT boxes gonna create a mapping for me between my ip address port and outgoing address and port
  • we have this TCP connection but no longer from one point to another point but one point to a middle box to another point.
  • Different from the application proxy, where the application proxy is TCP connection goes from one point to a proxy and it knows it's going to a proxy, the layer of hierarchy is still respected even though there's a proxy in the middle we are talking to. With these middle boxes I feel I'm talking to the web server but somebody else is messing with my packets in between
– "Layering violation" by assuming nature of payload contents – initially only worked for TCP and UDP. Must snoop on FTP messages
– Violates IP architectural model that states every interface on the internet has a unique IP address (millions of interfaces connecting to the internet have 10.0.0.1)
  • We no longer have unique mapping between ip address and interface
– Changes internet from connectionless to pseudo-connection-oriented
  • NAT maintains connection state, if it crashes all connections that are using the net are lost
– Limits number of outgoing connection, since port numbers are 16 bits.

• Despite criticisms, it is widely deployed, particularly in homes and small businesses
  – Carrier grade NAT: ISP only gives customers private addresses. So what NAT does is convert form one private address space to a different address space and all the costumers addresses are private to the ISP. The ISP will have a handful of real ISP addresses but it uses those addresses performs a NAT on them to go through to individual clients
• Significant security advantage
  – Since packets can only be received once an outgoing connection has been created, the internal network is greatly shielded from attacks from incoming unsolicited packets
  – NAT should not replace firewalls, and NAT is not a firewall
• Likely to remain in use even after IPv6 is widely deployed and there is no longer a scarcity of IP addresses

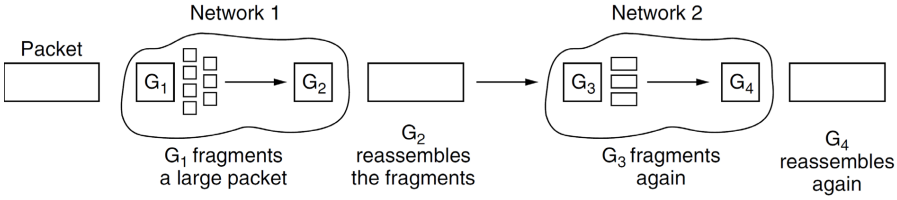
Fragmentation

- Recall that IP packets have a maximum size of 65,535
  - determined by the Total Length header field being 16 bits
- However, most network links cannot handle such large sizes
- All networks have a maximum size for packets, due to:
  - hardware (may have physical buffer in particular size)
  - OS (limit based on NO. of bit uses to store packets)
  - protocols (limit on NO. of bits)
  - standards compliance
  - desire to reduce transmissions due to errors (random bit errors)
  - desire for efficiency in communication channel (reduce delay)
- Nature of layered protocol stack means lower layer potentially needs to be able to fragment larger packets

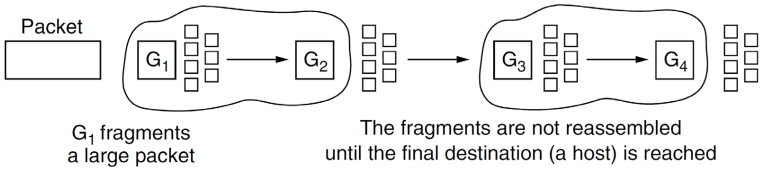
- More important: the most restrictive link on a packet's path may be on a link the sender is not connected to
  - Can't just pass info up the protocol stack in the sender
- Fragmentation (division of packets into fragments) allows network gateways to meet size constraints
- Hosts want to transmit large packets, since it reduces workload for them
- Creates a problem when that packet transits other networks that may not support such a large packet size
- Common maximum sizes for different network technology
  - 1500 bytes for Ethernet (non-standard extension to 9000)
  - 2304 bytes for 802.11.
- For example, sending packets between two devices on the same WiFi network could use larger packets than sending between WiFi and Ethernet device

- **MTU** – **Maximum Transmission Unit**: Maximum size for that network or protocol
- Path MTU: Maximum size for the path through the network
- Why not just set the Path MTU at the sender?
  - Connectionless network, with dynamic routing – both route and link
  - MTU can change after the packet has been sent
  - In keeping with design goals of TCP/IP (keep it simple) the easiest solution seemed to be to allow routers to break large packets into fragments to be sent individually along the network

- Problem: breaking a large packet into smaller fragments is easy, putting them back together again is a much harder task
- Two approaches:
  - Transparent Fragmentation – reassembly is performed at next router; subsequent routers are unaware fragmentation has taken place
  - Nontransparent Fragmentation – reassembly is performed at the destination host



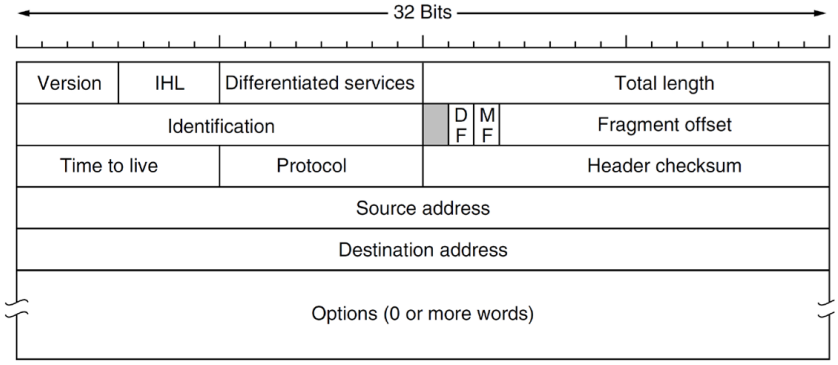
(a)Transparent



(b) Nontransparent (used by IP)

Fragmentation and IP Headers

- Recall the following IP Headers

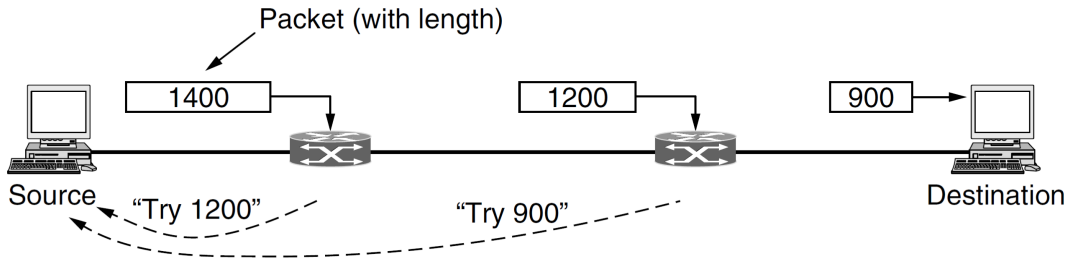


  - Identification – used to identify a packet
  - Flags(DF=Don't Fragment and MF=MoreFragments)
  - Fragment offset – offset in 8 byte blocks which means the offset value is multiple of 8 bytes, marks how the fragment should be ordered (use offset-value/8)
    - 13 bits – max offset (2^13 – 1) * 8 = 65,528,
- If a packet is fragmented:
  - Identification stays the same for all fragments
  - MF=1 for all fragments, except the last
  - Fragment offset – appropriately set for each fragment
- Fragment offset allows the receiving host to reconstruct out-of-order fragments in a buffer – similar to TCP Segments
- Fragment offset, and therefore fragmentation size, must be on an 8 byte boundary
  – Cannot send single byte fragments (except the last)
- If we have a payload of 1700 bytes. MTU=1500 bytes, ID=1:
  – 1st Fragment : ID = 1, DF=0, MF=1, FO=0
  – 2nd Fragment: ID = 1, DF=0, MF=0 , FO=185
    - (185*8=1480 = 1500 - 20 byte header) where payload = MTU - header

Fragmentation

- Simple approach, but some downsides:
  – Overhead from fragmentation (20 byte header for each fragment) is incurred from the point of fragmentation all the way to the host
  – If a single fragment is lost the entire packet has to be resent
  – Overhead on hosts in performing reassembly higher than expected
- Alternative approach is Path MTU discovery
  – Each packet is sent with the DF bit set – don't fragment
  – If a router cannot handle the packet size it sends an ICMP (Internet Control Message Protocol) to the sender host telling it to fragment its packets to a smaller size

Path MTU Discovery



- May cause initial packets to be dropped, but host can learn optimal size quickly and reduce subsequent fragmentation
  - The ip will be told the MTU of the path, then tell TCP passes up the stack and TCP can give segments small enough, so they don't need to be fragmented
- Fragmentation may still have to occur between hosts, unless upper layers can be informed of the size restriction
  – This is one reason why TCP/IP are typically implemented together so they can share such information
  – UDP relies on PMTUs discovered by TCP (because UDP don't have connections)

IPv4 vs. IPv6 Fragmentation

- IPv4 allows for either nontransparent fragmentation, or path MTU discovery
  – IPv4 minimum accept size 576 bytes
- IPv6 expects hosts to discover the optimal path MTU
  – routers will not perform fragmentation in IPv6
  – IPv6 minimum accept size 1280 bytes
  - Caution:
  – ICMP messages are sometimes dropped by networks, causing Path MTU discovery to fail.
  – In such circumstances a connection will work for low volume, fails at high volume – if in doubt send at the minimum accept size

Subnets

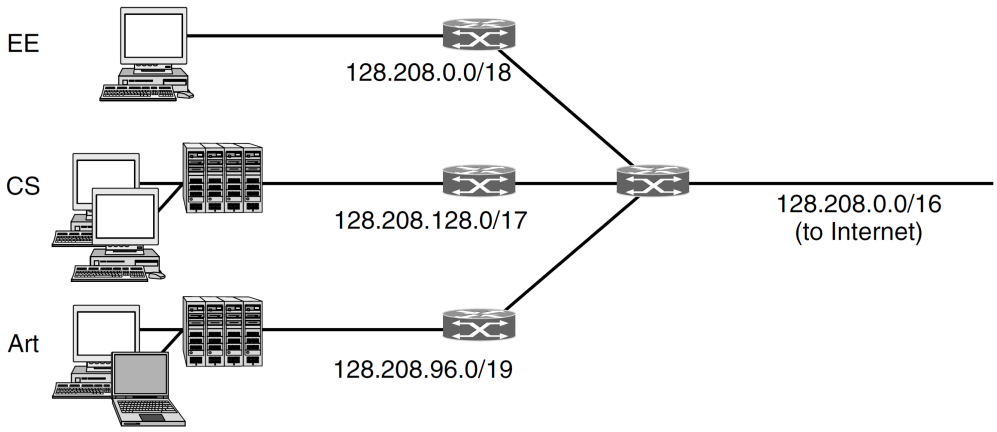
Network address	Host address	
Whole network address	Prefix of Host address	/
Subnet address = Whole network address + prefix of Host address		

- Recall: prefixes in IP addressing indicates different destination networks
- The same approach can be used internally within an organisation to maximise the use of their assigned IP prefix
- Subnetting allows networks to be split into several parts for internal use whilst acting like a single network for external use
- Subnet masks are written the same way as network masks:
  – "dotted decimal" (e.g. 255.255.255.128) or
  – "slash"notation(e.g./25)

Related Question:A router has just received routes to the following new IP networks: 57.6.96.0/21, 57.6.104.0/21, 57.6.112.0/21, and 57.6.120.0/21. If all of them use the same outgoing line, can they be aggregated? If so, to what? If not, why not?
They can be aggregated to 57.6.96.0/19 (the network parts of the addresses that use the same outgoing line are the same up to 19 bits). (Advanced answer: It could also be aggregated to a larger subnet containing 57.6.96.0/19, provided that didn't clash with other aggregates.)

- Example: A university with a /16 prefix could subnet its network as follows:
  – Computer Science/17 (half of allocation)
  – Electrical Engineering/18 (quarter of allocation)
  – Arts/19 (1/8 of allocation)
- Splits don't need to be even, but bits must be aligned to allow hosts portion to be used

Computer Science:	10000000	11010000	1|xxxxxxx	xxxxxxxx
Electrical Eng.:	10000000	11010000	00|xxxxxx	xxxxxxxx
Art:	10000000	11010000	011|xxxxx	xxxxxxxx



When a packet arrives from the internet, the router can use the subnet masks (bitwise AND) to find which subnet it should send the packet to, without knowing all hosts on the subnet

Network	Prefix	Network Address (binary)
EE	128.208.0.0/18	10000000.11010000.00000000.00000000
CS	128.208.128.0/17	10000000.11010000.10000000.00000000
Arts	128.208.96.0/19	10000000.11010000.01100000.00000000

Network	Prefix	Subnet Mask	Binary Subnet Mask
EE	128.208.0.0/18	255.255.192.0	11111111.11111111.11000000.00000000
CS	128.208.128.0/17	255.255.128.0	11111111.11111111.10000000.00000000
Arts	128.208.96.0/19	255.255.224.0	11111111.11111111.11100000.00000000

Example, packet comes in for 128.208.2.151, 10000000.11010000.00000010.10010111

Incoming	10000000.11010000.00000010.10010111
AMND CS Subnet Mask	11111111.11111111.10000000.00000000
Result	10000000.11010000.00000000.00000000
CS Network	10000000.11010000.10000000.00000000

Incoming	10000000.11010000.00000010.10010111
AMND CS Subnet Mask	11111111.11111111.11100000.00000000
Result	10000000.11010000.00000000.00000000
CS Network	10000000.11010000.01100000.00000000

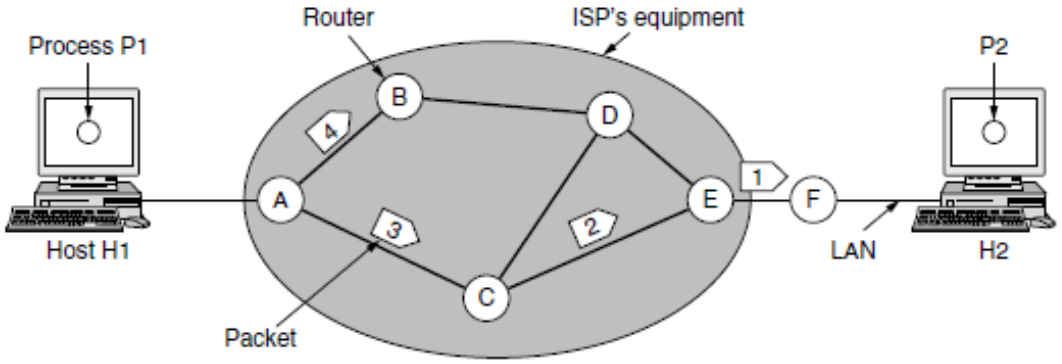
Incoming	10000000.11010000.00000010.10010111
AMND CS Subnet Mask	11111111.11111111.11000000.00000000
Result	10000000.11010000.00000000.00000000
CS Network	10000000.11010000.00000000.00000000

Match!

- Future changes can be made without any external impact
  – No need to request additional IP address allocation
  – Routing on the internet does not change, only internally

Week11

Packet forwarding



A's table (initially)

A	–
B	B
C	C
D	B
E	C
F	C

Dest. Line

A's table (later)

A	–
B	B
C	C
D	B
E	B
F	B

C's table

A	A
B	A
C	–
D	E
E	E
F	E

E's table

A	C
B	D
C	C
D	D
E	–
F	F

- Each router has a forwarding table (or routing table).
- This maps destination addresses to outgoing interfaces.
- Upon receiving a packet:
  – inspect the destination IP address in the header
  – index into the table
  – determine the outgoing interface
  – forward the packet out that interface
- Then, the next router in the path repeats the process
- The packet travels along the path to the destination.

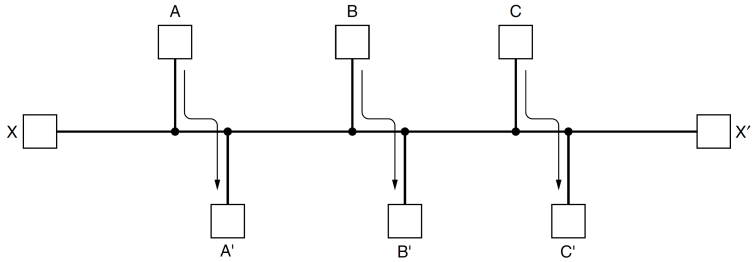
Routing

- How do these forwarding tables get created?
- The routing algorithm decides which output line an incoming packet should be transmitted on
- Combination of
  – an algorithm local to each router
  – a protocol to gather the network information needed by the algorithm

Routing Algorithm

- Properties of a good routing algorithm
  – Correctness – finds a valid route between all pairs of nodes
  – Simplicity - make faster to run, less buggy, easy to understand
  – Robustness – a router crash should not require a 'network' reboot
  – Stability – a stable algorithm reaches equilibrium and stays there, may let the router don't know where to send the packet or get the packet loops
  – Fairness
  – Efficiency
  – Flexibility to implement policies

Fairness vs. Efficiency
If there is enough traffic between A and A', B and B', and C and C', to saturate the horizontal link, what is the most efficient course of action for handling traffic between X and X'?



Make X to X' without stopping at all, then 3k - 2|x| assuming k is the capacity, (-2|x|) because the data goes two-ways between X and X'

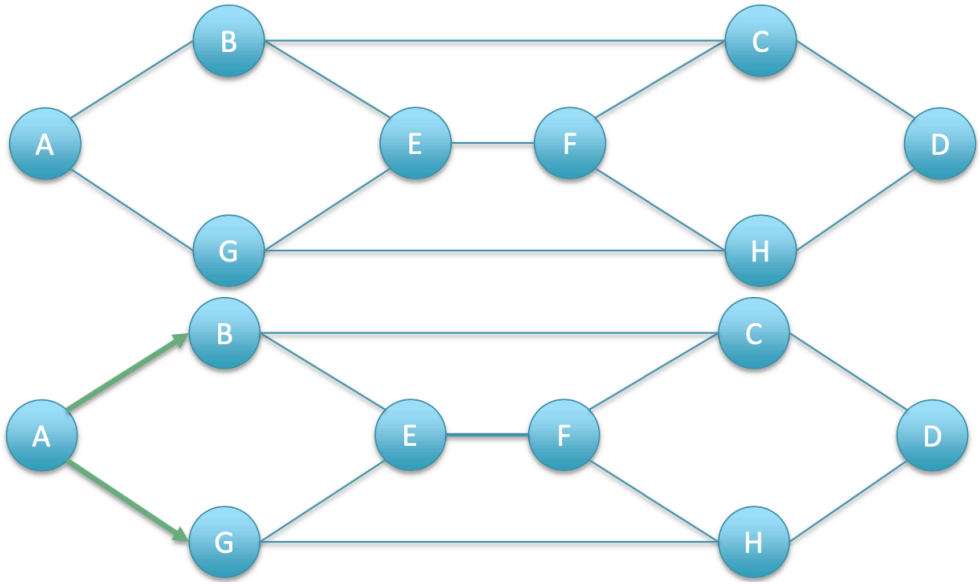
Delay vs. Bandwidth
- What is being optimised?
  - Mean packet delay?
  - Max network throughput?
- Simplest approach
  - Minimise the number of hops(links) a packet has to make
    - Tends to reduce per packet bandwidth and improve delay
    - Hopefully also reduces the distance travelled – but not guaranteed
      - Crossing the Pacific is one IP hop
  - Actual algorithms give a cost to each link
    - More flexible, but still cannot express all routing preferences

Routing Algorithm
- Non-adaptive (static routing)
  - Does not adapt to the network topology
  - Calculated 'offline' and uploaded to the router at boot
  - Does not respond to failure
  - Reasonable where there is a clear or implicit choice
    - Think about your home router, a static route out of your network is perfectly reasonable, since it is the only choice

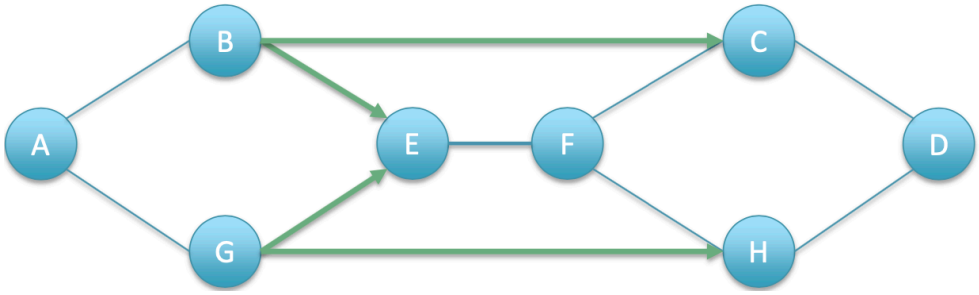
- Adaptive
  - Dynamic routing, adapts to changes in topology and potentially even traffic levels
    - May cause 'route flapping' where the routing of hops from one to the other, seldom works well in practice
      - Can be solved by time-day, instead of responding to the traffic levels, make something depends on time of day, make dependence reflect your measurements of traffic levels.
  - Optimise some property: distance, hops, estimated transit time, etc.
  - May get information from adjacent routers, or all routers in the network

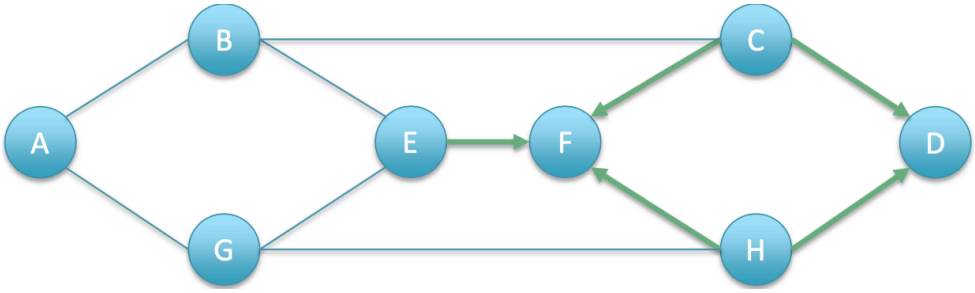
Flooding
- Simplest adaptive routing
– Guarantees shortest distance and minimal delay
– Useful benchmark in terms of speed
– Extremely robust – if there is a path it will find it
– Highly inefficient – generates many duplicate packets
– Have to have a way of discarding packets (TTL)
- If unknown can be set to diameter of network

Flooding Example:
Send a packet from A to D



E receives two copies
E forwards one copy, must keep track of packets it has forwarded



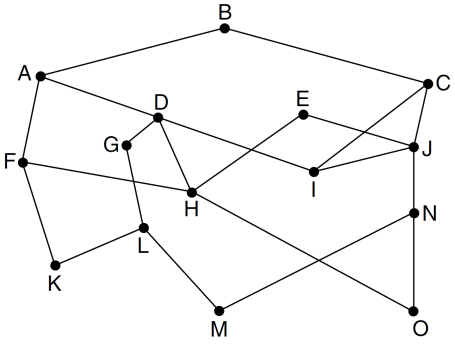


Adaptive Routing
- We need something more efficient than flooding
- Should adapt to network topology and changes
- If we have complete knowledge of the network "topology" (topology is what is connected to what) how can we determine an optimal route?

Optimality Principle (Bellman 1957)
I———J———K
- If router J is on the optimal path from router I to K, then the optimal path from J to K also falls along the same route.
  – If a better route existed for J to K it would be combined with the one from I to J, which would contradict our initial condition that our route from I to K was optimal
- When we study BGP, we'll see that this doesn't always apply

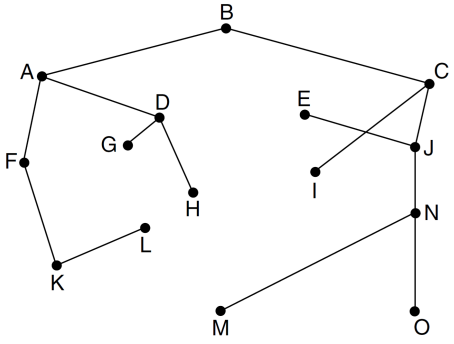
Sink Tree

• The optimality principle means that a set of optimal routes from all sources form a tree rooted at the destination



(a)

network



(b)

sink tree for router B

Shortest Path Algorithm

- View as a labelled graph
  - Label weight based on delay, distance, cost, etc.
- A number of algorithms, most famous is Dijkstra's algorithm
  - Finds the shortest path between a sink and all sources or vice versa
- At each step, each node is labelled with its distance (sum of costs on edges) from the source, and the best known path
  - Distances must be non-negative

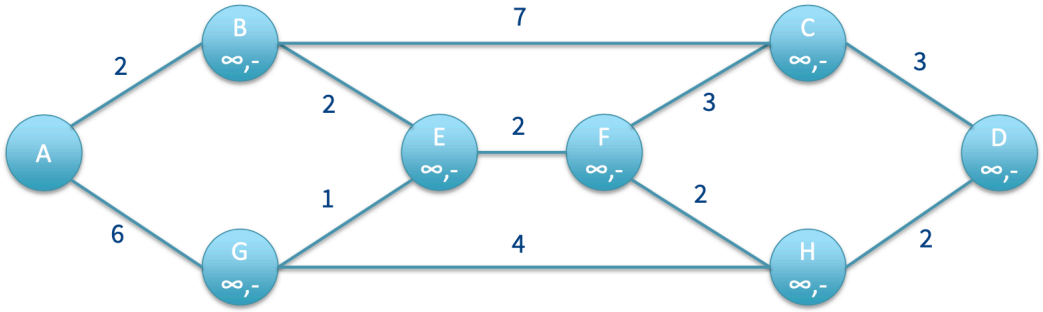
Dijkstra's Algorithm

- Divide nodes into three groups: "unseen", "open", "closed"
  - unseen: Not a neighbour of any node we have processed
  - Open: We have "visited" a neighbour, but not it. We know a path
  - Closed: We have visited it. We know the best path to it
- The algorithm moves nodes: unsee->open->closed
- Initially no paths are known so all nodes have a value of infinity ("unseen")
- As the algorithm proceeds, labels will be updated
  - Tentative ("open") – initial state
  - Permanent ("closed") – once a shortest path is found labelis permanent and won't change again
- Visit the source node: "Open" all of its neighbours and set their labels as the distance to them.
- Repeat until all nodes visited, or destination found:

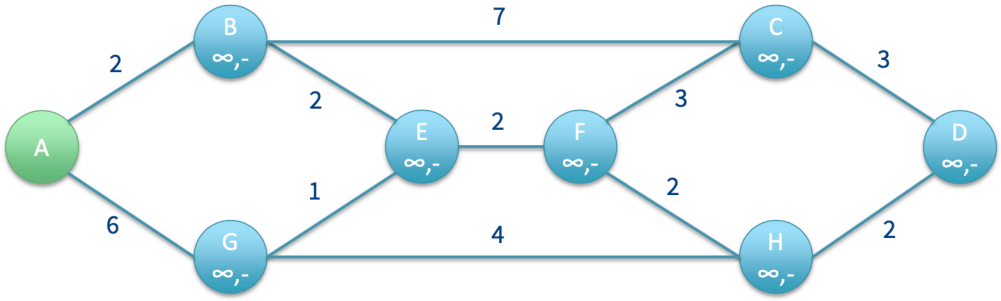
1. Examine adjacent nodes to the "working node", calculate distance, update labels if improved
2. Examine all tentative/open nodes in the graph and pick the one with the lowest distance and "visit" it
   - make that permanent/closed,
   - mark it as the "working node"
   - Go to step 1. (i.e., "Open" all of its neighbours and calculate the distance to them via this path. If it is less than the neighbour's current label set the label to the lower value.)

Example:

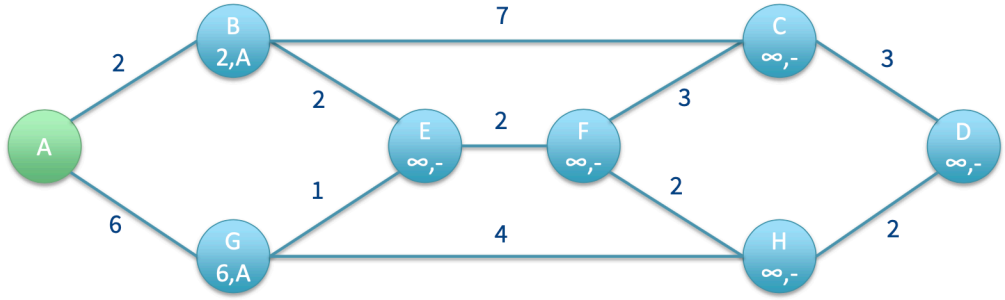
Send a packet from A to D



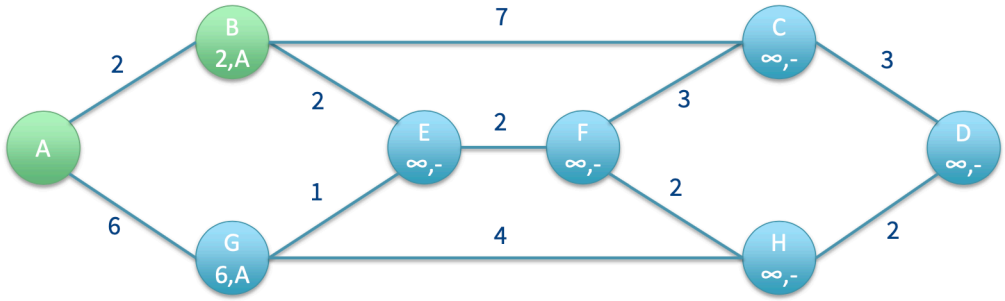
Make A permanet



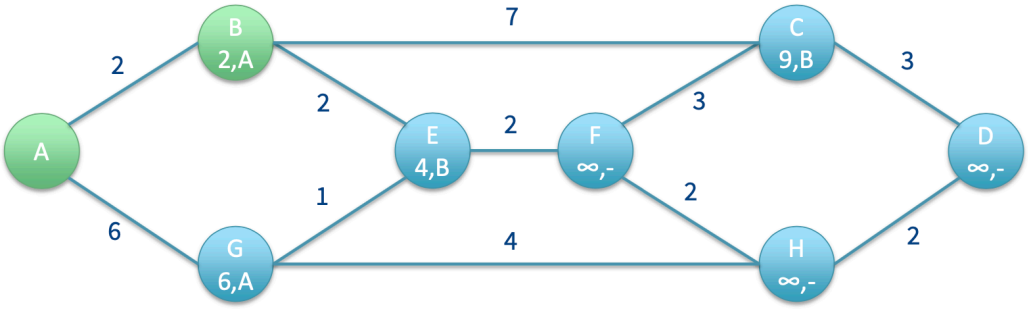
Open adjacent nodes (B,G): recalculate their weights



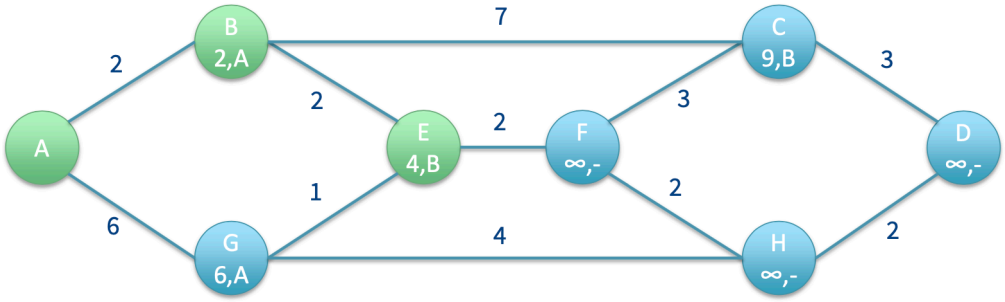
Select lowest distance tentative node (A) and make permanent



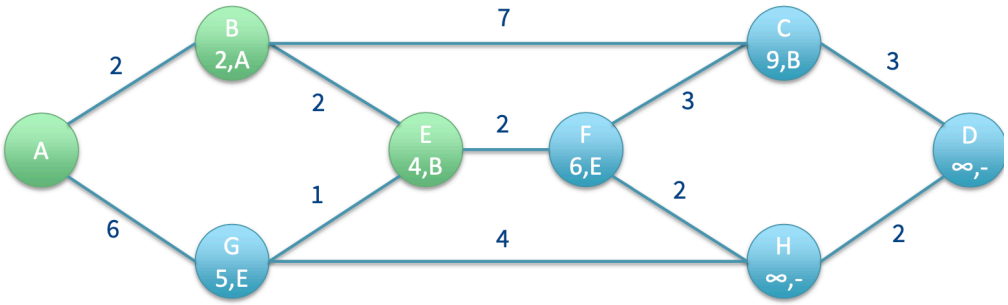
Recalculate distance for adjacent nodes (C,E)



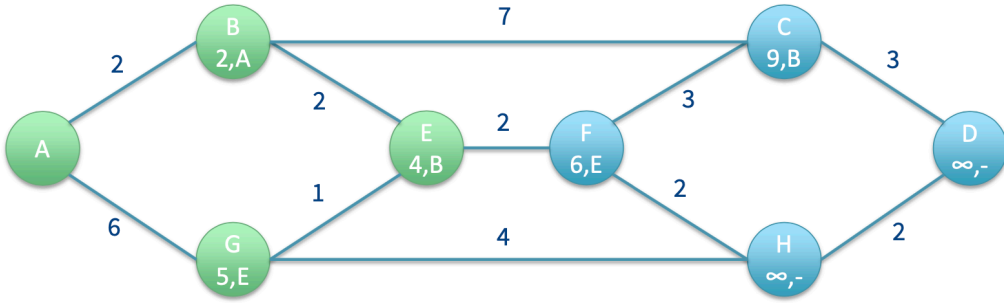
Select lowest distance tentative node (E) and make permanent



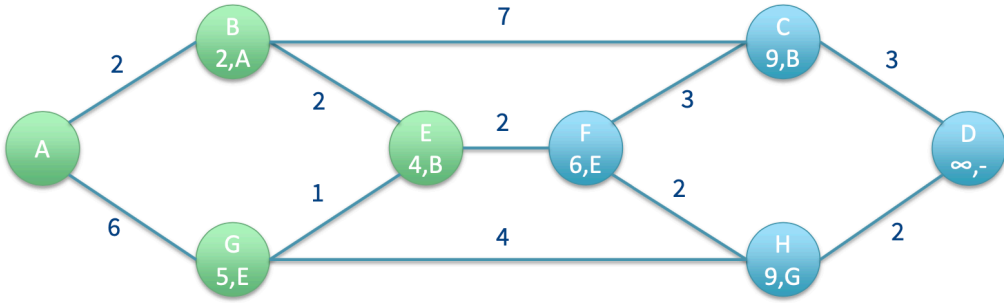
Recalculate distance for adjacent nodes (F, G). G's cost lower



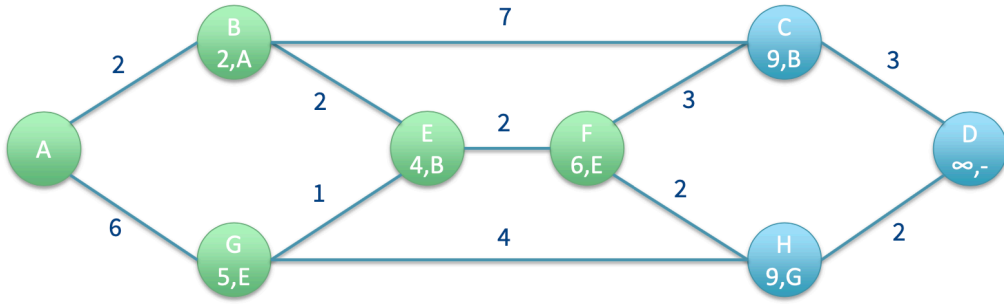
Select lowest distance tentative node (G) and make permanent



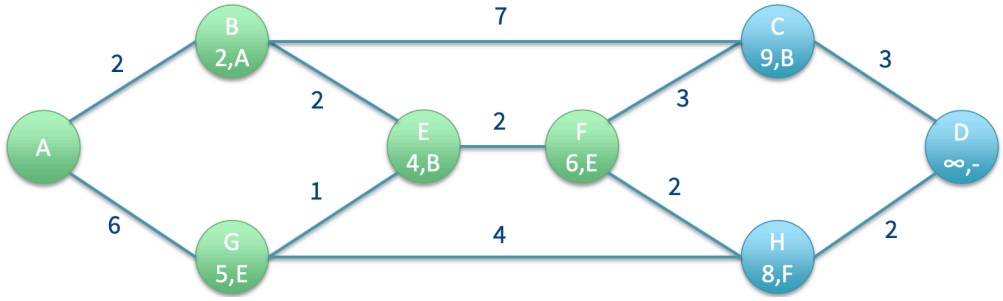
Recalculate distance for adjacent nodes (H)



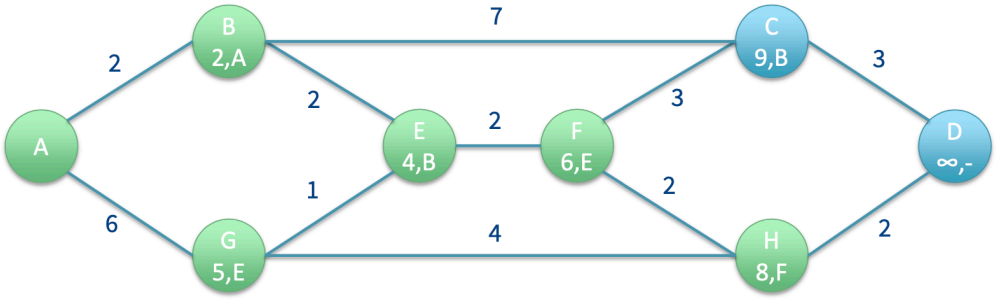
Select lowest distance tentative node (F) and make permanent



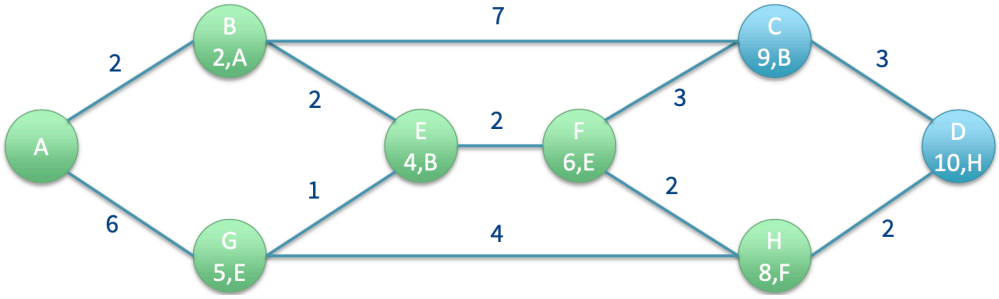
Recalculate distance for adjacent nodes (H drops, C stays)



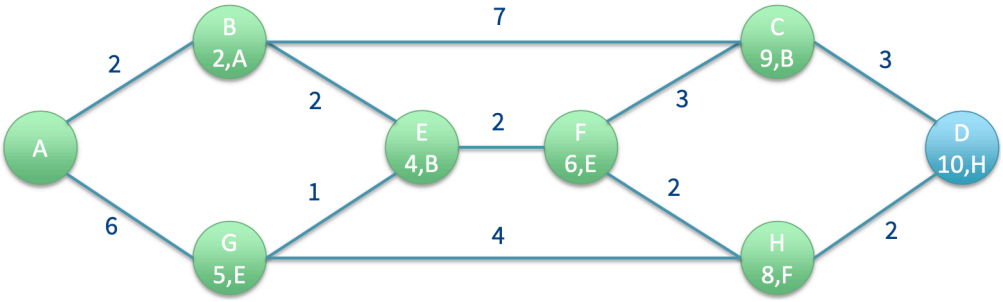
Select lowest distance tentative node (H) and make permanent



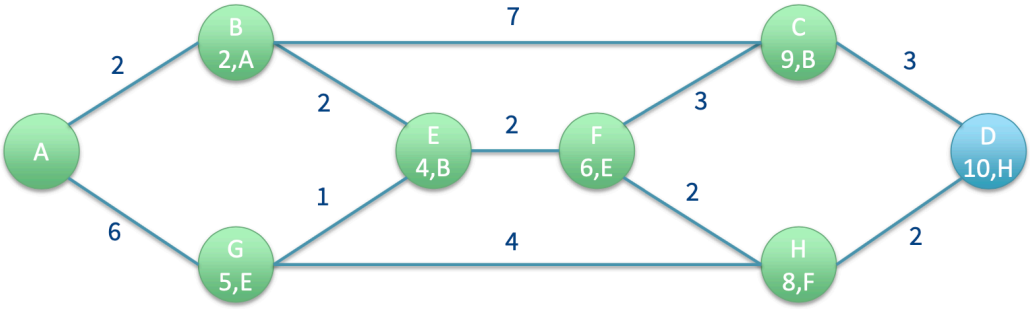
Recalculate distance for adjacent nodes (D)



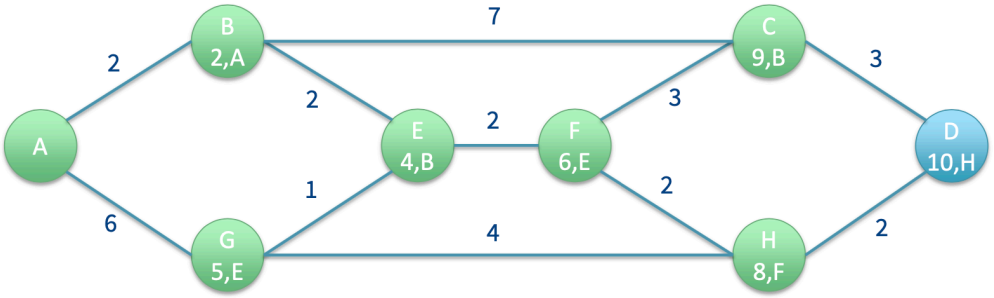
Select lowest distance tentative node (C) and make permanent



Recalculate distance for adjacent nodes (D, stays)



We can now stop because the lowest cost tentative node is the destination

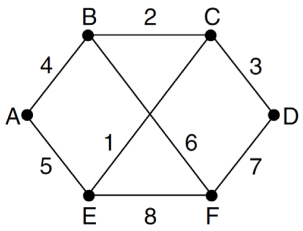


Link State Routing

- "Distributed" algorithm that replaced Distance Vector Routing ("Bellman-Ford"), which converged slowly
- Variants of Link State Routing used today
- Relatively simple, 5 step process each router must follow: (Share info 1-4; centralized alg 5)
  1. Discover its neighbours and learn their network address
  2. Set the distance or cost metric to each of its neighbours
  3. Construct a packet containing all it has just learned
  4. Send the packet to, and receive packets from, all other routers
  5. Compute the shortest path to every other router

- To discover neighbours, a router on boot sends out a HELLO packet on each interface. The router on the other end must reply with its unique ID
  – Slightly more complicated on LANs
- Cost can be set automatically or manually
  – Common technique is 1/bandwidth: 1 Gbps = 1, 100Mbps = 10
  – Could use delay as well – calculated using an ECHO packet
  – Many networks manually choose preferred routes and then look for link costs to make those routes the shortest. "Traffic engineering" .
    - The operator choose these routes and it finds link cost that give these routes when everything is going OK, but operator doesn't want 'if this link goes down, then this is the optimized set of routes, if that link goes down then that is…' by using a routing algorithm most of time I like to work roughly like this. If you can't do that then figure out something gonna work.

- A Link State packet consists of ID, sequence number, age, and a list of neighbours and their respective costs



(a)

Link State Packets

A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

(b)

- Building the packet is easy, deciding *when* to build them is difficult
  – At intervals?
  – When a change occurs – link disconnect?
- To send packets to all other routers, flooding is used
  – To be precise, reliable flooding – which uses acknowledgements to guarantee every other router receives the packet
  – To avoid waste, when a router receives a Link State Packet it compares the sequence number to the one it previously received (if any). If the sequence number is not larger it discards it and does not forward on the flood
  – Sequence numbers are 32 bits to avoid wrap-around
  – Still a problem if a router crashes and restarts its sequence number from 0
    - Solution is the age field, which is reduced by 1 each second, when the age hits zero the information is discarded

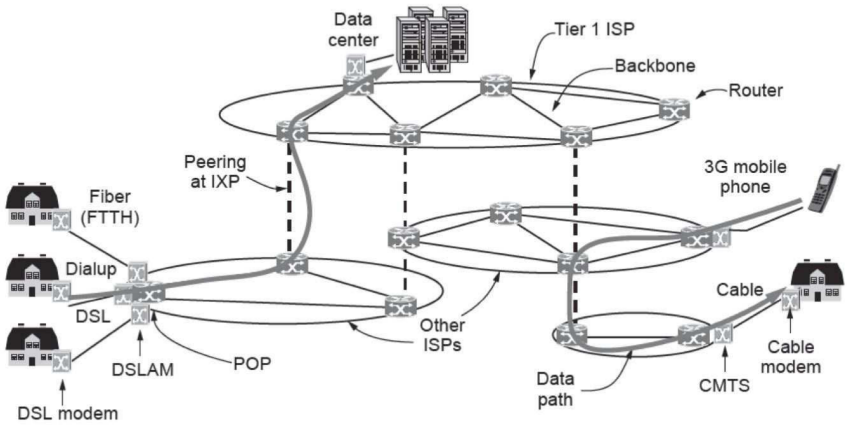
Distance vector routing

- Link state routing distributes the topology
  – Everyone then performs centralised routing
- Distance vector routing uses a true distributed algorithm
  – Nodes announce the distance from themselves to each destination
- Instead of announcing the topology like in link state routing
  – When they receive a new announcement of others' distances, the update their own estimates, and make a new announcement
  – Bellman Ford algorithm (Same Bellman as Bellman's principle)

Related Question: The Open Shortest Path First (OSPF) link-state routing protocl specifies that dijkstra's algorithm be used to calculate the shortest paths once all the link states are known. (a) What could possibly go wrong if a different shortest path algorithm were used?
(b) Why is that unlikely to cause problems in practice?
(a) In principle, a different shortest path algorithm could find a different path, and in principle, having different routes used by different routers could cause "black holes" and routing loops.
(b) In practice, a different shortest path algorithm could only find a different path if it had the same cost as the one found by Dijkstra's algorithm. (Otherwise, one of them wouldn't be the shortest.) This means that we can't have a routing loop, unless one of the link costs is 0. Similarly, each hop will bring the packet nearer to the destination whichever shortest path a router chooses, by Bellman's principle.

Border Gateway Protocol (BGP)

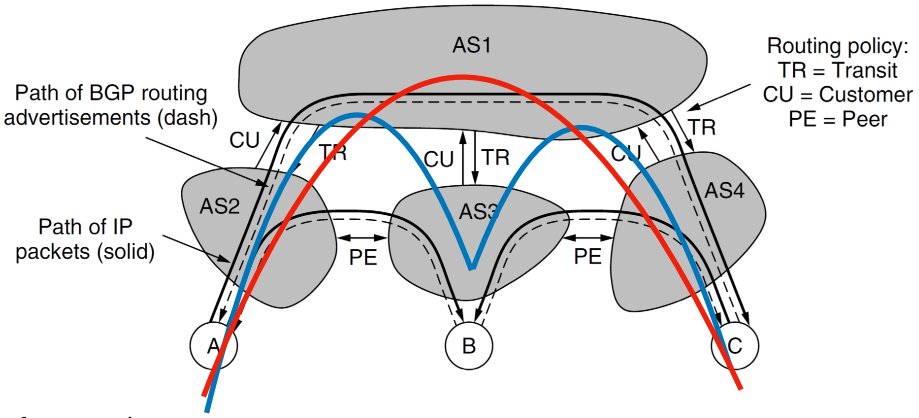
• Recall that the internet is constructed by internetworking independently administered networks
– Autonomous Systems (AS) - collections of routers under the same administrative control
– Bigger than the "network" part of IP address
• Each network will have
– A protocol for internal routing (usually based on linked state)
– A protocol for external routing between ASes
  • Must be the same for all ASes
  • BGP



- In contrast to the internal routing, BGP needs to consider politics as well
  – Companies not willing to have their network used by others
  – ISPs not wanting other ISPs' traffic on their networks
  – Not carrying commercial traffic on academic networks
  – Use one provider over another because they are cheaper
  – Don't send traffic through certain companies or countries
- Can't always say one route is "better" than another
  – Better in some respects, worse in others
  – Bellman's optimality principle doesn't always apply

Border Gateway Protocol (BGP)

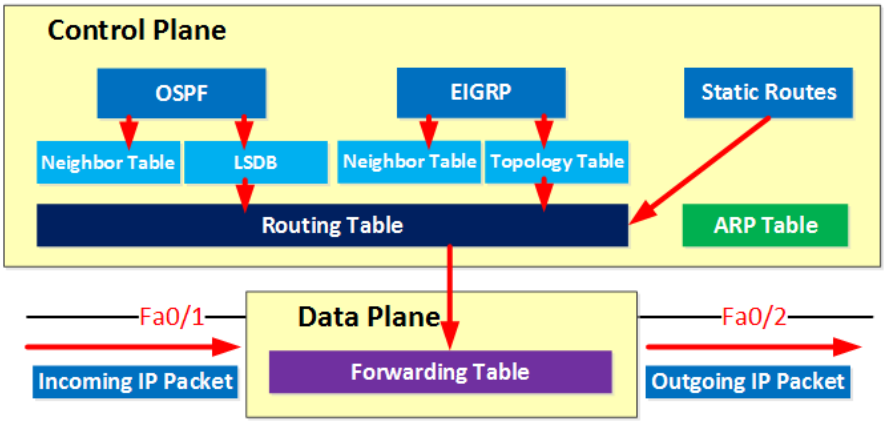
- Typically based on
  – customer/provider: I pay you for "transit" of traffic I send or receive
  – or peering agreements we carry each others' traffic without charge
- Provider advertises routes for the entire internet
  – Customer only advertises routes for their network to avoid transiting other traffic
- "Valley free" routes (AS3 is the valley here), use the red path instead of blue



- BGP is also a prime target for attack
  – A malicious AS can advertise routes for networks at extremely low cost, causing traffic to re-routed through that AS
  - 2017 Russian AS advertised routes for Google, Apple, Facebook, Microsoft, Twitter, etc.
  - 2008 Pakistan attempted to block YouTube, but inadvertently blocked it for the entire internet
  – Sometimes an innocent mistake, but also an effective way to divert traffic for monitoring or disruption

Data plane vs control plane

- Protocols don't really form a single stack
  – BGP (network layer) uses TCP (transport layer) for updates
- A better model is to think of different "planes" of protocols
  – Data plane (network layer: forwarding)
  – Control plane (network layer: choosing routes)
  – [Management plane (network layer: setting BGP policies)]



Internet Control Protocols

- Protocols that are used at the internet layer to manage functionality
– ICMP – Internet Control Message Protocol
– DHCP – Dynamic Host Configuration Protocol
– ARP – Address Resolution Protocol (not strictly internet layer)

Internet Control Message Protocol

• You've probably all used ICMP messages (PING), but they are not limited to just sending out echo requests

Message Type	Description
Destination Unreachable	Packed could not be delivered
Time exceeded	Time to live field hit 0
Parameter problem	Invalid header field
Source quench (no more use)	Choke packet
Redirect	Teach a router about geography
Echo and echo reply	Check if a machine is alive
Timestamp request/reply	Same as Echo, but with timestamp
Router advertisement/solicitation	Find a nearby router

Related Question: The ping command is used to detect if the interface associated with an IP address is "alive": connected and active. It sends an ICMP "echo request" packet to that host, to which the host replies with an ICMP "echo reply" packet. The sender also typically records the delay between sending the request and receiving the reply, giving an indication of the round trip delay.

Given that most systems use interrupts to process arriving packets, is it possible for a host to reply even if the host has crashed?

Some hosts give lower priority to ICMP packets than to TCP and UDP packets. How could this distort the results of ping?

(a) Yes.

(b) It would report higher RTTs than were experienced by data traffic.

Traceroute

- Exploits the Time exceed message
  - Recall TTL is hop count (routers the packet traverses)
  - Trace route sends out packets to the same destination, each with an incremented TTL
  - Counters will hit zero at successive routers, causing the router to return a Time exceeded message, revealing the IP address of the router
  - Sender can use this information to determine path and timings of the route a packet will take

Related Question: Traceroute is often implemented using ICMP echo request packets as the probes. (a) Is it possible to use other types packets? Why or why not? (b) What advantages and disadvantages are there in using other types of packets?

(a) Yes, any type of IP packet should be discarded when its TTL field expires, and should trigger the sending of an ICMP packet in response.

(b) Packets such as TCP packets and UDP packets may be less likely to be blocked by middleboxes. Also, if the goal is to find the path taken by a TCP packet, a TCP probe packet is more likely to take the same path as a real TCP packet.

However, a TCP packet may perhaps cause a higher load on the final receiving host.

Traceroute example

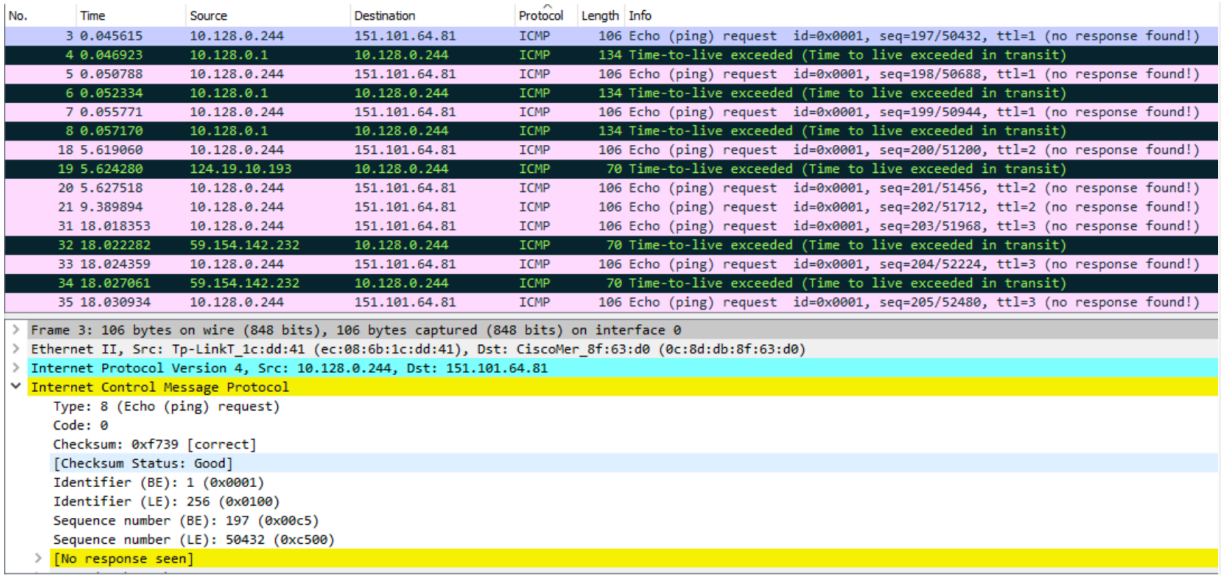
- tracert on windows, traceroute on ubuntu
- For TTL as 1, the first packet took 3 milliseconds to get a reply, the second packet took 2 milliseconds and third took 5 ms, the address we get is 10.128.0.1
- For TTL as 2, only first packet go reply, the ip address we get is 124.19.10.193
- For TTL = 8, the packet is sent successfully

```
tracert bbc.com

Tracing route to bbc.com [151.101.0.81]
over a maximum of 30 hops:

  1     3 ms     2 ms     5 ms  10.128.0.1
  2     4 ms     *        *     124.19.10.193
  3     6 ms     2 ms     4 ms  59.154.142.234
  4    11 ms     3 ms     8 ms  165.228.132.193
  5     6 ms     5 ms     6 ms  bundle-ether11.exi-core10.melbourne.telstra.net [203.50.11.113]
  6     5 ms     4 ms     3 ms  bundle-ether1.lon-edge901.melbourne.telstra.net [203.50.11.108]
  7     5 ms     3 ms     8 ms  fas2867546.lnk.telstra.net [110.145.207.178]
  8     5 ms     8 ms     5 ms  151.101.0.81

Trace complete.
```



DHCP – Dynamic Host Configuration Protocol

- The internet layer requires each host to have a unique IP address
- We could manually configure each host (certain networks do this)
  – Difficult to administer, error prone
  – Slow to respond to new devices
- DHCP is the automated way of handling IP address allocation
- Security concerns – connecting any device will issue an IP address (can apply restrictions)
- Network has a DHCP server for issuing IP addresses
- Host sends out a DHCP DISCOVER packet
  – Routers can be configured to relay these to the DHCP server if not directly connected to the same network
- DHCP Server receives the request and responds with a DHCP OFFER packet containing an available IP address
- IP addresses are typically issued on a lease – time after which the IP address will be reclaimed by the server and re-issued
  - the shorter the time, the sooner the server will be able to reclaim the address and allocate to someone else, the longer the time, the less overhead varies, less processing , less traffic
  – Hosts can request a renewal before the lease expires
- Used to set a number of parameters
  – Default gateway, DNS servers address, time servers (use to synchronize)

Network booting: every time updates, no need to push into local hard drive. The kernel is determined by network. Drawback: the whole kernel is needed to be sent, so it's slower

MAC Address

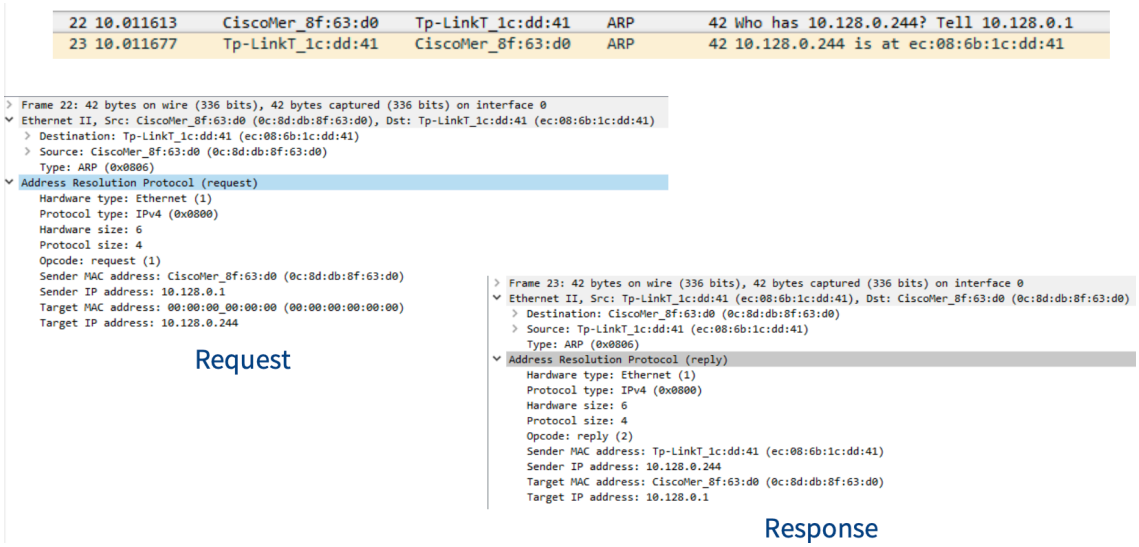
- If we don't have an IP address how does the DHCP server know where to send the response to?
- The answer is a MAC (Media Access Control) Address (link layer address, address within local subnet)
  – Can think of it as a globally unique identifier for the interface
  – Usually hard-coded by the manufacturer
  – Between 48 and 64 bits long
- Addressing used at the Host-to-network/data link layer
  – Sometimes called physical address

Related Question: A DHCP server knows the MAC address of the host that is requesting configuration. This means that the server can give the host the same IP address each time the host requests one. (a) What are the advantages of this static assignment over allocating independent of the MAC address? (b) What are the advantages of this over statically configuring the hosts? (c) What are the disadvantages of this compared to statically configuring the hosts?
(a) This improves security by restricting the service to hosts whose users have registered their MAC addresses.
It also allows other permissions to be based on the IP address, since this is now constant for each device.
(b) It causes all of the configuration to be centralized in one place. This makes management much easier.
(c) It increases the boot time, it requires a DHCP server to be administered, and it very slightly increases network traffic.

Address Resolution Protocol

- ARP is the link between the **internet layer** and the **physical network layer**
- It allows a host to translate an IP address into a physical address (MAC)
- Broadcasts an Ethernet packet asking who owns the target IP address
- Broadcast arrives at every host on the network, the owner will respond with its MAC address
- The low level sending is done via MAC addresses, so this protocol is run a lot, even to find out how to communicate with the nearest router

ARP in Action



Proxy ARP: says there is another computer delegate the task of replying ARP messages. It will answer the IP address so that not need to bother MAC address
Benefit: If you have an appliance that is sleeping, should be listening to the network and respond these ARP packets. So the computer can stay sleep and get up once it gets packets directly to it. All the other computers don't need even to listen to the ARPs, the computers that are idle can stay sleep.

ARP

- Incredibly simple
- Not particularly efficient
- Security nightmare
  - No authentication
  - Caching of responses, even when not directly requested
  - ARP spoofing is the gateway attack for most man-in-the-middle attacks
  - Provides a way of intercepting and spoofing ARP messages to associate the attackers MAC address with another hosts IP address (i.e. default gateway, DNS server, website)

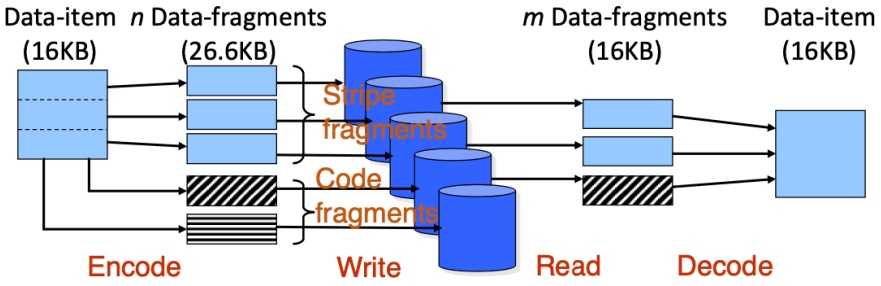
Internetworking

- We have spoken a lot about internetworking without looking at the global scale of the internet.
- The global communication we have come to take for granted are dependent on a network of undersea cables

Guest lecture

Automating erasure coding decisions
We don't want to lose data, we want robust and durable, so we make multiple data
Make the data separate into n fragments, m of them belong to the original, (n-m) from fancy math way, when read the data just read m of them

- Degree of data redundancy
- Manner in which redundancy achieved
- General characterization: {*m*-of-*n, +encryption}*
- *N/m is smaller than m/1*
- *We can tolerate two wrongs*



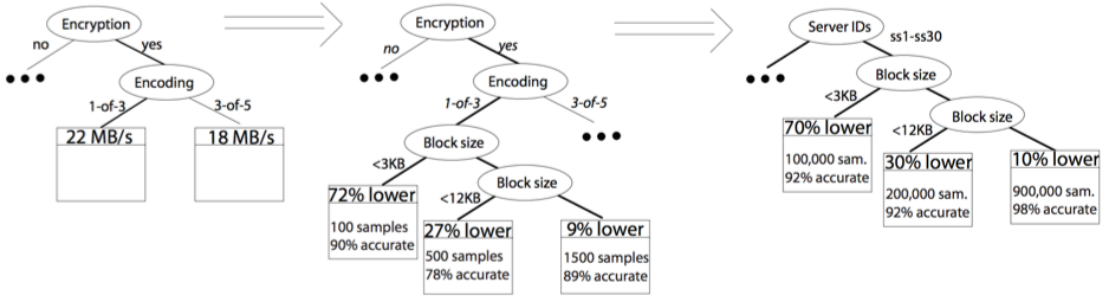
How do we automate this?
Before, a human admin had to understand *n, m, workload*
This was (and still is) complicated stuff

	Availability	Confidentiality	CPU	Net/storage
n	↑		↑	↑
m	↓		↑↓	↓
encryption		↑	↑	
workload			↑↓	↑↓

- Reads vs. writes: reads like parallelism, writes waste bandwidth
- Latency vs. throughput
- Capacity-bound or IOPS-bound
- Speed of repair
- CPU cost of encoding
- Caching behavior

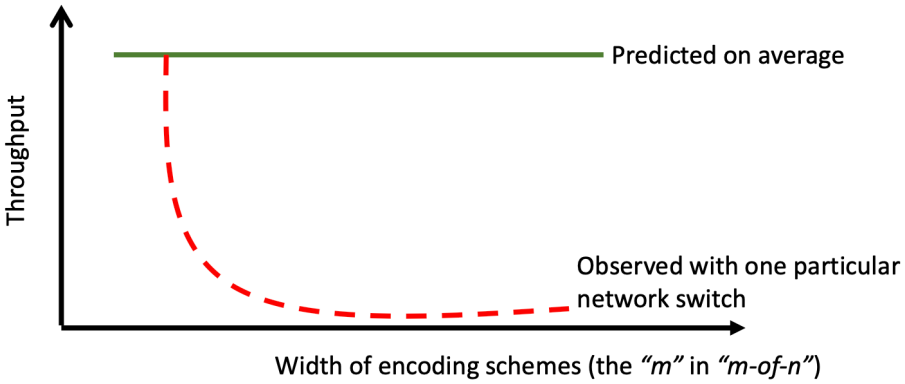
So let's use **ML**

- Pre-deep learning era, using trees and forests
- Hope to discover non-linear relationships based on observations only



In 2008 we were happy this kind of modeling worked at all

Early signs that operationalizing(get things to work) was going to be hard



Heads up: this was the **TCP in-cast problem**
- Before the TCP in-cast papers started appearing later in 2008*
- Throughput collapse due to overloading of switches' buffers

Every-time read from source simultaneously, everything reaches the switch at the same time, it overloads the buffer starts to dropping stuff.

The system is wrong not the model
Most of the time spent fixing TCP problem

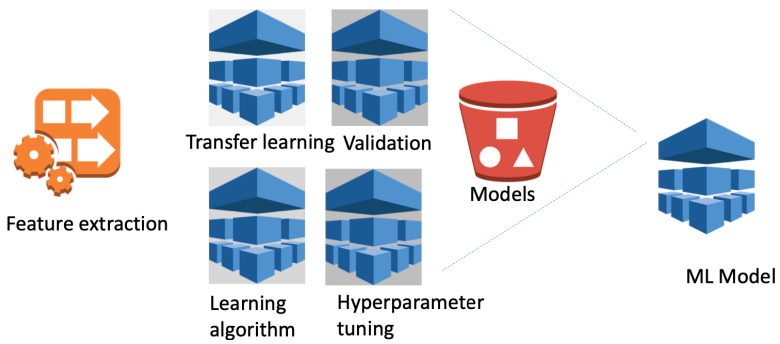
Past 10 years: big data, new hardware AND experience with operationalizing
Difficulty: Coding < debugging < operationalizing

ML Pipeline: for self-managing storage:
Encoding: what m, n going to choose
Forecasting workload:
Hardware trends forecasting:
Placement prediction: how to layout the data in data center across the owned servers?

ML Pipelines: For supply-chain decisions
Supply chain forecasting: forecast the flows in system
Demand forecasting:
Regional space availability models:
Transportation models:

Billions of predictions daily Fleets of thousands of servers
Example When things go wrong: if there is an arriving new container with no space, ML made predictions but when in reality there is no space, re-routed is needed.

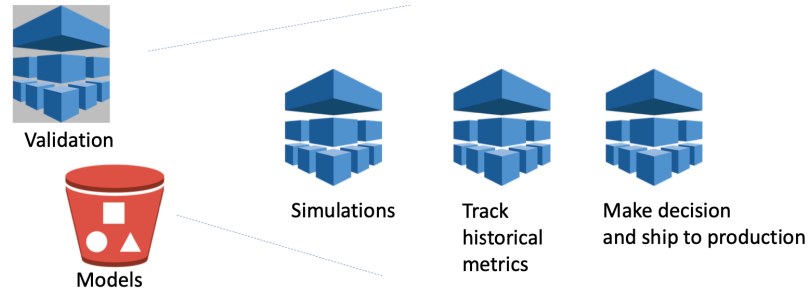
"ML Model" is a **subsystem** on its own



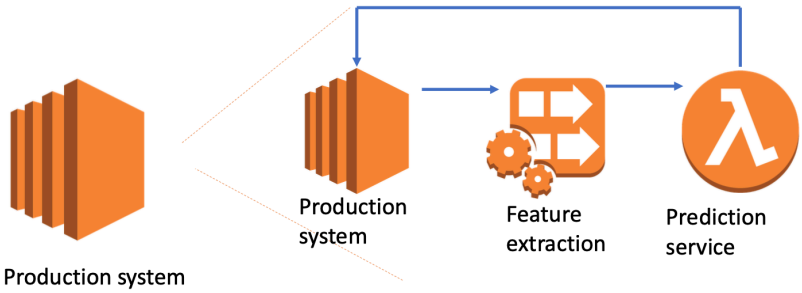
Operationalizing ”ML Model”

- How are you storing and versioning your models?
- Is model creation reproducible?
- How do you debug a model?
- How do you tie a trained model back to the data that produced it?
- And not just the data – you may need the exact version of your own code and all the dependent libraries and infrastructure. What if there was an update to Spark that affects your training, can you still reproduce your previous experiment?
- How do you know model is not obsolete? When do you retrain?

Which model to ship to production system?



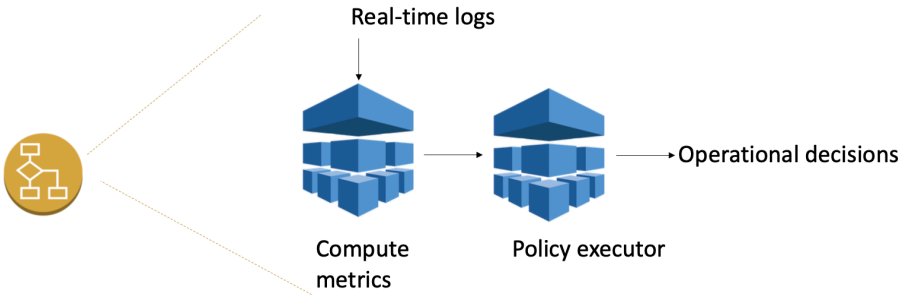
”Production system” has several parts



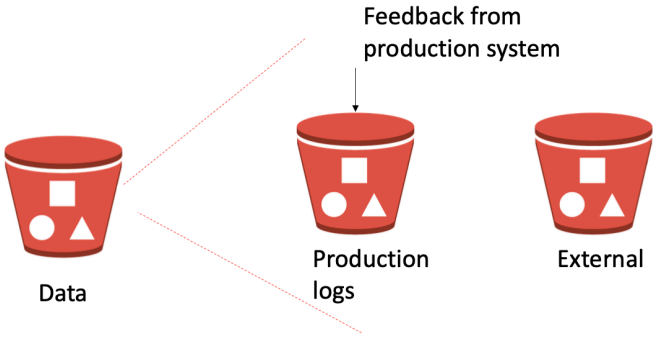
Operationalizing ”production system”

- So how should you deploy models to production?
- ML models change are usually continuously being improved, so your system needs to be able to handle that.
- The same way, and with the same care, you deploy anything else. With offline validation, staged roll-outs, A/B testing, canaries, whatever is right for your situation.
- Having a separate deployment process for ML models is a classic route to making your system impossible to maintain.

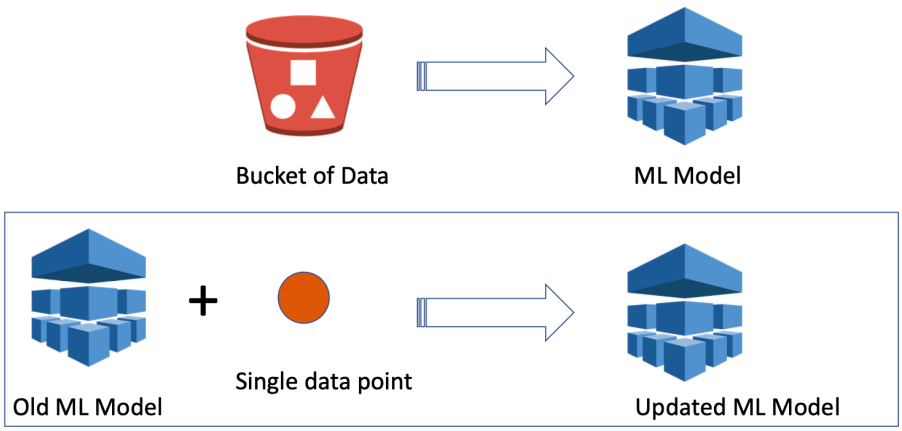
Monitor everything



And then there is the data



Incremental learning (aka ”learn all the time”)

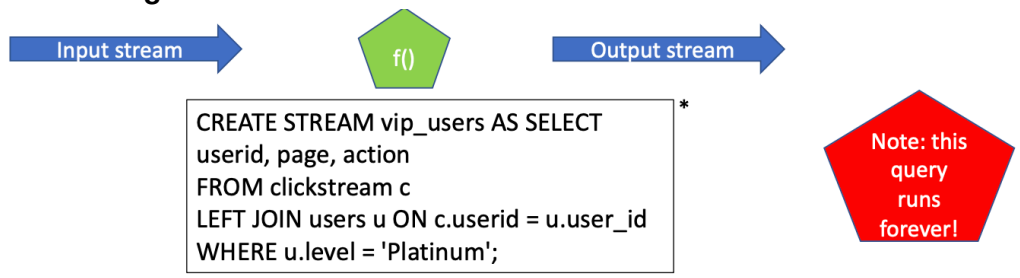


Systems component of incremental learning
• Batch-based IO model -> streaming IO model

Why streaming-based IO model?
• Some data is just streaming/never complete/unbounded in nature
• Competitive pressure to learn from every event as soon as it happens
• Who needs it?
  • ~35% of Fortune 500 are exploring streaming solutions •
  • Fraud detection
  • High-frequency trading
  • Ads & clicks

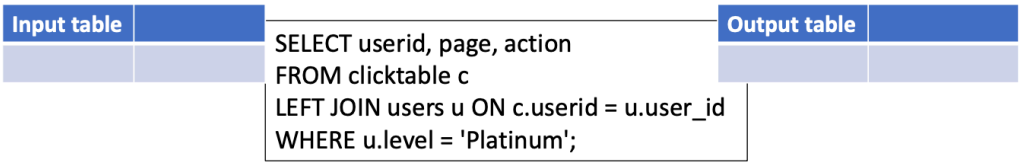
What do we really mean by streaming again? In the systems context:
• Push vs. pull
In the ML context:
• Low latency (data analysis time - data arrival time)
• Data recency: recent data/observations matter disproportionally
• Retraining model after each event
• Reparametrize frequency
• Structural model changes frequency
• ”after each observation << every 24 hours << weekly << monthly”

Be aware: **streaming code** is different



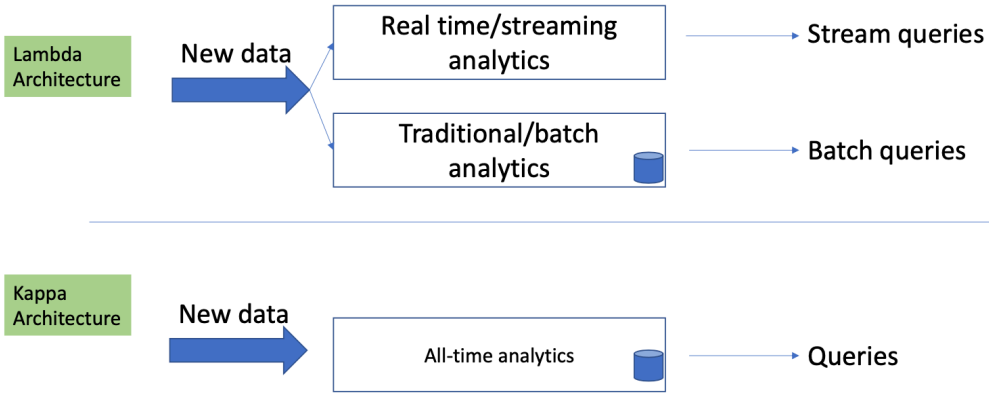
Streaming queries

Batch queries



Streaming IO challenges (and advantages)
- Optimized for ingest AND data processing
- ms-seconds latency range, billions of events per day
- Relaxed consistency models
- Natural robustness to data loss (exactly-once not needed)

Streaming IO challenges: towards a unified architecture



Rethinking **hardware** for continuous training
- How should hardware look like when training is continuous?
- Latency vs. throughput tradeoffs

Algorithmic component
- Incremental algorithms != one-shot algorithms

One shot

```
jarvis(S)
   // S is the set of points
   pointOnHull = leftmost point in S // which is guaranteed to be part of the CH(S)
   i = 0
   repeat
      P[i] = pointOnHull
      endpoint = S[0]      // initial endpoint for a candidate edge on the hull
      for j from 1 to |S|
         if (endpoint == pointOnHull) or (S[j] is on left of line from P[i] to endpoint)
            endpoint = S[j]   // found greater left turn, update endpoint
      i = i+1
      pointOnHull = endpoint
   until endpoint == P[0]      // wrapped around to first hull point
```

Incremental:

Algorithmic component

- Neural networks don't even have a way to change structure incrementally
- Even changing weights incrementally can lead to "catastrophic forgetting"

Who will solve this problem first?

- Open question: systems community or theory community?

Table 1. The basic ID3 tree construction algorithm.

1. If all the instances are from exactly one class, then the decision tree is an answer node containing that class name.
2. Otherwise,
   (a) Define a_{best} to be an attribute with the lowest E-score.
   (b) For each value $v_{best,i}$ of a_{best}, grow a branch from a_{best} to a decision tree constructed recursively from all those instances with value $v_{best,i}$ of attribute a_{best}.

Table 2. The basic ID4 tree update procedure.

1. For each possible test attribute at the current node, update the count of positive or negative instances for the value of that attribute in the training instance.
2. If all the instances observed at the current node are positive (negative), then the decision tree at the current node is an answ node containing a ''+'' (''−'') to indicate a positive (negative) instance.
3. Otherwise,
   (a) If the current node is an answer node, then change it to a decision node containing an attribute test with the lowest E-score.
   (b) Otherwise, if the current decision node contains an attribute test that does not have the lowest E-score, then
      i. Change the attribute test to one with the lowest E-score.
      ii. Discard all existing subtrees below the decision node.
   (c) Recursively update the decision tree below the current decision node along the branch of the value of the current test attribute that occurs in the instance description. Grow the branch if necessary.

Architecture: Transfer learning & HPO

- Transfer skills learned from old tasks
  - Addresses the warm-start problem for models
- Avoid “catastrophic forgetting” of past
- Extends Bayesian optimization methods
- Subtle: what to transfer for erasure coding example?
  - Do we have any a priori beliefs about how things work?
  - Yes: we can use queueing theory* to form first beliefs and pass them around

Architecture: change detection

Approximate Convex Hull of Data Streams

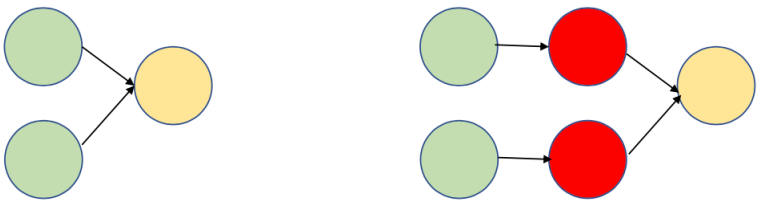
Avrim Blum*[1], Vladimir Braverman†[2], Ananya Kumar‡[3], Harry Lang§[4], and Lin F. Yang¶[5]

1 TTI-Chicago, Chicago, United States
avrim@ttic.edu
2 Johns Hopkins University, Baltimore, United States
vova@cs.jhu.edu
3 Carnegie Mellon University, Pittsburgh, United States
skywalker94@gmail.com
4 Johns Hopkins University, Baltimore, United States
hlang8@math.jhu.edu
5 Princeton University, Princeton, United States
lin.yang@princeton.edu

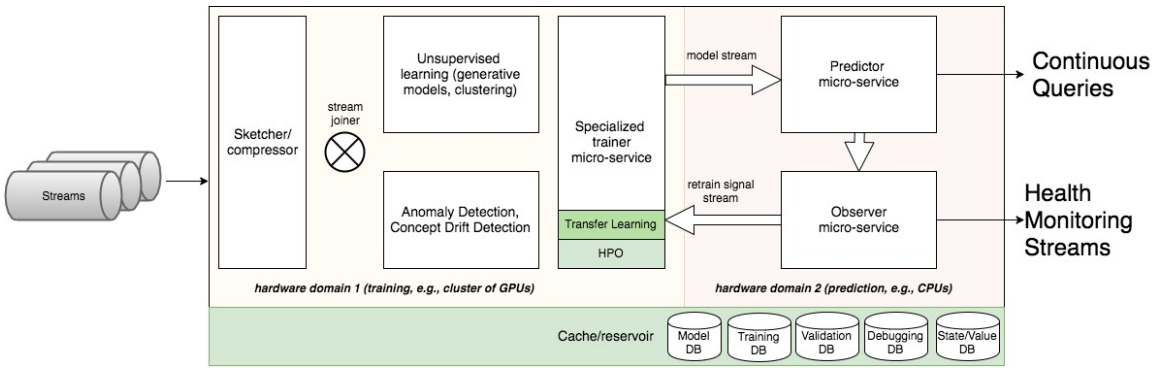
Abstract

Given a finite set of points $P \subseteq \mathbb{R}^d$, we would like to find a small subset $S \subseteq P$ such that the convex hull of S approximately contains P. More formally, every point in P is within distance ϵ from the convex hull of S. Such a subset S is called an ϵ-hull. Computing an ϵ-hull is an important problem in computational geometry, machine learning, and approximation algorithms.

In many real world applications, the set P is too large to fit in memory. We consider the streaming model where the algorithm receives the points of P sequentially and strives to use a minimal amount of memory. Existing streaming algorithms for computing an ϵ-hull require $O(\epsilon^{(1-d)/2})$ space, which is optimal for a worst-case input. However, this ignores the structure of the data. The minimal size of an ϵ-hull of P, which we denote by OPT, can be much smaller. A natural question is whether a streaming algorithm can compute an ϵ-hull using only $O(\text{OPT})$ space.



This is not allowed



Change detection

- How do we know model is still performing well?
- Can we detect when workload has sufficiently changed to warrant retraining?
- Unsupervised learning: when labels do not exist, monitor properties of the workload, e.g., histogram comparisons
- Supervised learning: if labels exist, monitor prediction quality and properties of label distribution

Cache/reservoir

• Goal: automatically manage storage for ML models
• Given 1TB of storage space, automatically manage storage and carve out space for continuous retraining and validation
• What is the equivalent of LRU/LFU here?
• Keep data based on season (Christmas, School holidays, etc)?

Where do we go from here?

- Transition over past 10 years has been more than just big data and novel hardware.
- Coding < Debugging < Operationalizing
- Systems and algorithmic opportunities in research and industry • Streaming systems • Incremental learning algorithms • Storage management for ML
- Watch this space:
  - Amazon Sagemaker: fully managed ML service
  - Amazon Kinesis: real-time analytics over data streams