# 1. Data representation and cache

## Temporal locality:

Temporal locality refers to the phenomenon in computer science where recently accessed data is likely to be accessed again in the near future. This concept is used in computer architecture and memory management to optimize performance by keeping recently accessed data in faster forms of memory, such as cache.

accessing the same variables over time

## Spacial locality:

Spatial locality refers to the phenomenon in computer science where data that is stored physically close to each other in memory is likely to be accessed together. This concept is used in computer architecture and memory management to optimize performance by grouping related data together in memory, such as keeping all the variables of a single function together in memory. This way, when one piece of data is accessed, the adjacent data is likely to be used soon as well and can be quickly accessed as well.

accessing things close to each other in memory.

## Data representation:

In a 32-bit system, a long variable is typically 32 bits in size, while a float variable is also 32 bits in size. The main difference between the two is the way they store and represent numerical values.

A long variable is used to store integers (whole numbers), while a float variable is used to store floating-point numbers (numbers with decimal points). The range of values that can be stored in a long variable is typically larger than the range of values that can be stored in a float variable, but floating-point numbers can be more precise than integers.

## Inaccuracy in floating point numbers:

In the IEEE floating point format, the inaccuracies of calculations become larger as the numbers get larger due to the limited number of bits available to represent the mantissa (or significand) of the number. This means that the larger the number, the less precise the representation of that number in the format.

This is because the mantissa is represented as a fixed-point number, with a fixed number of bits to the left and right of the radix point. As the number gets larger, more bits are required to represent the mantissa, but since the number of bits is fixed, the representation becomes less precise.

As a programmer, it is important to be aware of this limitation and understand that floating point numbers may not be accurate to their full precision when working with large numbers. This can lead to rounding errors and other inaccuracies in calculations. It is also important to use appropriate precision and rounding techniques to minimize these inaccuracies.

In a 32-bit system, an int variable is typically 32 bits in size, the same as a long variable. The main difference between the two is the way they are defined and used.

An int variable is used to store integers (whole numbers) and it's the most common data type for representing integers in most programming languages. The range of values that can be stored in an int variable depends on the specific implementation, but it is usually between -2,147,483,648 and 2,147,483,647.

long variable is also used to store integers, but the range of values that can be stored in a long variable is typically larger than the range of values that can be stored in an int variable, but it's not always the case, it depends on the specific platform and compiler.

## Stride:

How long there is between entries you access. So if you have good spatial locality you have a low stride, best/lowest is 1. Accessing a matrix in c is best if you go through a whole row at at a time, getting next entry, meaning stride=1. If you go column-wise, your stride is equal to the length of the rows, as you move one row's length every time you change the column.

**(eksamens spørgsmål) How many bits are used to represent a ‘char‘, a ‘long‘ and a ‘float‘ in the normal 64-bit Intel architecture?**

- 'char': 8 bits
- 'long': 64 bits
- 'float': 32 bits

**(eksamens spørgsmål) How many bits are used to represent a ‘char‘, a ‘long‘ and a ‘float‘ in the normal 32-bit Intel architecture?**

- 'char': 8 bits
- 'long': 32 bits
- 'float': 32 bits

**(Eksamens spørgsmål) Why is it not possible to represent 1 10 (one tenth) in the IEEE floating point format? What is the consequence of this?**
1/10 cannot be represented exactly in the IEEE floating-point format because it is a repeating decimal, meaning that the digits to the right of the decimal point repeat indefinitely. Since the mantissa has a finite number of bits, it cannot represent an infinite number of repeating digits.

The consequence of this is that when a program uses the IEEE floating-point format to represent 1/10, the value it actually stores will be a close approximation of 1/10, but not exactly equal to it. This can lead to errors and inaccuracies in calculations that involve 1/10 and other numbers that cannot be represented exactly in the IEEE floating-point format.

This phenomenon is called the "floating-point precision error" and it affects all floating-point computations. Programs that need to perform high precision computation, such as scientific simulations and financial computations, use arbitrary precision libraries or other methods to perform these calculations.

# 2. Operating Systems

### Which functions always involve a system call (assuming no error occurs)?

- Reading and writing to files: Functions such as fopen(), fread(), (fwrite() **only** if the buffer needs to be flushed, so not always), and fclose() all involve system calls to the operating system's file system to perform these operations.

- Allocating memory: Functions such as (malloc() **only(So not always)** if there is not enough space in the heap)  and calloc() involve system calls to the operating system's memory management system to request and allocate memory.

- Creating and managing processes: Functions such as fork() and exec() involve system calls to the operating system's process management system to create and run new processes.

- Interacting with network resources: Functions such as socket() and connect() involve system calls to the operating system's network stack to create and manage network connections.

- Interacting with the hardware: Functions such as ioctl() and mmap() involve system calls to the operating system's device drivers to interact with the hardware.

### What is a part of the kernel in a standard Unix system?

- Process Management: The kernel manages the creation, execution and termination of processes. It also manages the scheduling of processes and assigns the CPU time to different processes.

- Memory Management: The kernel manages the physical and virtual memory of the system and provides memory allocation and deallocation services to the processes.

- File System: The kernel provides an interface for the file system, it manages the file system, and it controls the access to the file system by the processes.

- Network Stack: The kernel provides the network stack, which is responsible for managing the network communication and protocols.

- Device Drivers: The kernel provides the device drivers, which are responsible for managing the communication between the hardware and the kernel.

- Security: The kernel provides security mechanisms such as access control and authentication to protect the system and user's data.

## Which operations should be accessible only to the kernel?

Operations that should be accessible only to the kernel are those that involve direct manipulation of system resources, such as memory management, process management, and device driver access. These operations are considered privileged and should only be executed by the kernel, which is the core part of the operating system that has complete control over the system. Examples of such operations include:

- Memory allocation and deallocation
- Interrupt handling
- Task scheduling
- File system management
- Network stack management
- Access to hardware devices
- Creation and deletion of processes
- Control and manipulation of system resources such as CPU, memory, and I/O devices.

Allowing user-level programs to perform these operations could potentially lead to system instability or security vulnerabilities.

In summary, the kernel is the part of an operating system that controls all the resources of a computer and provides the interface between the hardware and the software. So, any operation that could affect the stability and the security of the system should be executed by the kernel.

## Which are states that a Unix process can be in:

- running
- sleeping (waiting for an event or signal)
- stopped (paused)
- zombie (terminated but waiting for parent process to acknowledge)
- uninterruptible sleep (waiting for a hardware event)

## Consider a demand-paged system with the following time-measured utilisations: CPU utilisation 10% Paging disk 97.7% Other I/O devices 5%. What would likely improve CPU utilisation?

- Install a faster paging disk.
- Install more main memory.

## What does Fork() copy?:

- register contents
- process memory
- file descriptors (pointere til åbne filer)

## Why are virtuel memory useful?

Virtual memory is useful because it allows a computer to efficiently use more memory than is physically available. It does this by temporarily transferring data from RAM to a hard disk, which has a much larger storage capacity. This process is transparent to the user and allows the computer to run programs that would not be able to fit into RAM.

dditionally, virtual memory allows multiple programs to run simultaneously by allocating a separate portion of memory to each program, preventing them from interfering with each other. This improves system stability and security by isolating processes from one another.

## Dirty bit:

The dirty bit is a flag used in virtual memory management. It is set by the operating system to indicate that a page of memory has been modified and the changes have not yet been written to disk. This means that if the system were to lose power or crash, the changes on that page of memory would be lost. The purpose of the dirty bit is to track which pages of memory need to be written to disk before the system can safely shut down or before the page can be reused for another purpose. This process is called "paging out" or "swapping out", it's done by a kernel component called "swapping daemon" or "pager" which periodically scans memory looking for dirty pages and writes them to disk.

## Threads running as part of the same process have different:

- Instruction counter
- register contents
- stacks

## Threads running as part of the same process have the same:

- Virtual memory space
- Open files
- Heaps

## Threads (T: Threads, P: Processors, N: Assignments) :

- T =P is likely efficient when each chunk of N ÷P data pieces takes the same amount of time, i.e. when the computation is load balanced.

- T > P could be effective when the difference pieces of data vary widely in how long they take to process, as we can schedule the excess threads on processors that would otherwise be idle. It might also be effective if the processing is heavily dependent on waiting for slow IO operations, as a form of latency hiding.

- T =N is probably only effective for fairly low N, or when the time taken to process one piece of data is high enough to make thread creation cost negligible

## What is a race condition?

A race condition occurs when two or more threads access shared data simultaneously and the outcome of the program depends on the order in which the threads access the data. This can lead to unexpected or inconsistent results. To avoid race conditions, proper synchronization techniques such as locks or semaphores should be used to ensure that only one thread is accessing shared data at a time.

## What is a deadlock?

A deadlock is a situation in which two or more threads are unable to proceed because each is waiting for one of the others to do something. A deadlock typically occurs when a thread requests a resource that is held by another thread, while the second thread is waiting for a resource held by the first thread. Because neither thread can proceed, the program can become unresponsive or "deadlocked". To avoid deadlocks, it's important to ensure that resources are acquired and released in a consistent, predictable order, and to use synchronization techniques such as locks or semaphores to manage access to shared resources.

Virtuel address

- Oversæt adresse til binær
- Oversæt page size til bytes
- **Offset**: Kig på hvor stort page table is - ex page size 64 bytes = 2^6 which means offset is 6 bits.
- **VPN:** Is the address without the offset.
- **index**: How many bits do you need to represent the set in TLB.
  Ex: 4 sets (0,1,2,3) you´ll need 2 bits.
- **TLB Tag** is the rest without offset and index.

## Internal vs. external fragmentation

Internal fragmentation and external fragmentation are both types of memory fragmentation that can occur when allocating and deallocating memory.

**Internal fragmentation** occurs when a block of memory is allocated that is larger than the requested size. For example, if a program requests a block of memory with a size of 100 bytes and the memory manager allocates a block of memory with a size of 128 bytes, there will be 28 bytes of internal fragmentation. This is because the program only needs 100 bytes of memory, but the memory manager has allocated 128 bytes.

**External fragmentation** occurs when there are small, scattered blocks of free memory that are not contiguous, making it difficult to allocate larger blocks of memory. This can happen when a program repeatedly allocates and deallocates small blocks of memory, leaving behind many small, scattered free blocks.

An example of external fragmentation is when a program requests a block of memory with a size of 1000 bytes, but there is only a single block of free memory with a size of 512 bytes and another block of free memory with a size of 128 bytes. The memory manager cannot allocate the requested 1000 bytes of memory because the two free blocks are not contiguous.

**In general,** external fragmentation is more of a concern in dynamic memory allocation, where memory is allocated and deallocated frequently. Internal fragmentation is more of a concern in fixed-size memory allocation, where all blocks are the same size.

## Semaphore

A semaphore is a synchronization object that controls access to a common resource in a parallel programming environment. It is typically used to protect shared data structures from concurrent modification, and it uses a counter to keep track of the number of threads that can access the resource at any given time. When a thread requests access to the resource, the semaphore decrements the counter. If the counter is zero, the thread is blocked until the semaphore is signaled, at which point the counter is incremented and the thread is granted access to the resource.
**explain, in pseudocode or prose, how semaphores can be implemented in terms of mutexes and integer variables. Show how to initialise a semaphore with initial value v,**

**how to implement the P operation, and how to implement the V operation. Could your implementation be made more efficient by also using condition variables?**

A semaphore can be implemented using a mutex (short for "mutual exclusion") and an integer variable to keep track of the current value of the semaphore. The mutex is used to protect access to the semaphore value, while the integer variable stores the current value of the semaphore.

To initialize a semaphore with an initial value of v, the following pseudocode can be used:

```
initialize_semaphore(semaphore s, int v) {
    s.value = v;
    initialize_mutex(s.mutex);
}
```

The P (proberen) operation, also known as "wait" or "acquire", decrements the value of the semaphore and blocks the calling thread if the value becomes negative. The pseudocode for the P operation is as follows:

```
P(semaphore s) {
    lock_mutex(s.mutex);
    s.value--;
    if (s.value < 0) {
        block_thread();
    }
    unlock_mutex(s.mutex);
}
```

The V (verhogen) operation, also known as "signal" or "release", increments the value of the semaphore and unblocks a waiting thread if the value becomes non-negative. The pseudocode for the V operation is as follows:

```
V(semaphore s) {
    lock_mutex(s.mutex);
    s.value++;
    if (s.value <= 0) {
        unblock_thread();
    }
    unlock_mutex(s.mutex);
}
```

It is possible to make this implementation more efficient by also using condition variables. A condition variable is a synchronization object that allows a thread to wait for a specific condition to become true. Using a condition variable, the implementation of P and V could be simplified. The thread would wait on the condition variable instead of blocking, and signal the condition variable instead of unblocking a thread. This would allow for more fine-grained control over thread synchronization and potentially reduce the number of context switches.

## What does automatically mean in semaphores:

In the context of semaphores, "atomically" means that the operation is performed as a single, indivisible action. This means that the operation cannot be interrupted or divided into smaller parts, and it must be completed in its entirety or not at all.

In the case of semaphores, an atomic operation is one that modifies the semaphore's value and/or changes the state of a process that is blocked on the semaphore. The most common example is the semaphore's "wait" and "signal" operations.

A "wait" operation (also known as "acquire" or "P" operation) decrements the semaphore's value atomically, and if the value becomes negative, the process that is executing the wait operation is blocked.

A "signal" operation (also known as "release" or "V" operation) increments the semaphore's value atomically, and if there are any processes blocked on the semaphore, one of them is unblocked.

The atomic nature of these operations ensures that the semaphore's value is always consistent, and that no race conditions can occur between multiple processes that are accessing the semaphore.

**(exam question) How might TLB interactions make context-switching between different processes be less efficient than context-switching between threads within the same process?**
TLB (Translation Lookaside Buffer) is a cache that stores recently used virtual-to-physical memory translations, so that the kernel does not have to perform a page table walk for each memory access.

When context switching between different processes, the kernel must flush the TLB to ensure that the new process's page table entries are used instead of the old process's page table entries. This is necessary to ensure that the new process has the correct permissions and mappings to the physical memory.

Flushing the TLB requires the kernel to invalidate all of the entries in the TLB, which can be a relatively slow operation. The more entries in the TLB, the more time it takes to flush it.

On the other hand, when context switching between threads within the same process, the kernel does not have to flush the TLB, since all the threads within the same process share the same page tables. Because of this, context switching between threads is faster than context switching between different processes.

Additionally, if a process has a large number of threads and a large working set, the TLB may not be able to hold all the translations, so the kernel will have to access the page table more often, which will increase the overhead of context switching.

In summary, context switching between different processes requires flushing the TLB, which can be a relatively slow operation and make context switching less efficient than context switching between threads within the same process.

## Heap:

malloc: alokere plads til memory - Malloc(8) betyder at payload skal være minimum 8
realoc: alokere plads - realloc (0x400b010, 20)
free(): free plads

## Heap:

- Is the immediate coalescing? freed blocks are merged together.
- (realloc, 8) - means payload should be 8 -> 8/4 = 2 with means there should be 2 blocks between header and footer.
- Reallocated blocks are allocated. Freed blocks are free.
- Check all header and footers last three bits.

## Which operations always involves transferring control to the kernel?

There are several operations that involve transferring control to the kernel, including:

- System calls: When a program needs to perform an operation that requires the kernel's assistance, it makes a system call. This transfers control to the kernel, which performs the requested operation and then returns control to the program. Examples of system calls include reading from or writing to a file, creating a new process, and allocating memory.

- Interrupts: Interrupts are used to signal to the kernel that a hardware device needs attention. When an interrupt occurs, control is transferred to the kernel, which handles the interrupt and then returns control to the program.

- Exceptions: Exceptions are used to signal to the kernel that an error has occurred in a program. When an exception is thrown, control is transferred to the kernel, which handles the exception and then returns control to the program.

- Scheduling: When the kernel needs to schedule a new process to run, it transfers control to the scheduler, which selects the next process to run and then transfers control to that process.

- Synchronization: When a program needs to synchronize access to shared resources, it may use kernel-provided synchronization primitives such as semaphores, mutexes, or monitors. These operations also involve transferring control to the kernel.
- Reading or writing a file: When a program needs to read or write to a file, it makes a system call to the kernel's file system driver.

- Creating or destroying a process: When a program needs to create or destroy a process, it makes a system call to the kernel's process management subsystem.
- Allocating or deallocating memory: When a program needs to allocate or deallocate memory, it makes a system call to the kernel's memory management subsystem.
- Opening or closing a socket: When a program needs to open or close a socket for network communication, it makes a system call to the kernel's socket management subsystem.
- Sending or receiving a signal: When a program needs to send or receive a signal, it makes a system call to the kernel's signal management subsystem.
- Mounting or unmounting a file system: When a program needs to mount or unmount a file system, it makes a system call to the kernel's file system management subsystem.
- Accessing a device driver: When a program needs to access a device such as disk, network, or GPU, it makes a system call to the kernel's device driver.
- Creating or destroying a thread: When a program needs to create or destroy a thread, it makes a system call to the kernel's thread management subsystem.
- Setting or getting time and date: When a program needs to set or get the system time and date, it makes a system call to the kernel's time management subsystem.
- Managing process priorities: When a program needs to manage process priorities, it makes a system call to the kernel's process management subsystem.

In general, any operation that requires the kernel's assistance, such as accessing system resources, managing memory, or performing I/O, will involve transferring control to the kernel.

## Which of the following components of a Unix system run in kernel mode?

The following are some of the components of a Unix system that typically run in kernel mode:

- **Memory management:** The kernel's memory management subsystem is responsible for allocating and deallocating memory, as well as managing virtual memory.

- **Process management**: The kernel's process management subsystem is responsible for creating, destroying, and scheduling processes.

- **File system management**: The kernel's file system management subsystem is responsible for managing the file system, including mounting and unmounting file systems, and providing access to files.

- **Network stack:** The kernel's network stack is responsible for managing network communication, including creating and managing sockets, handling network packets, and routing.

- **Device drivers**: The kernel includes device drivers, which are responsible for communicating with and controlling hardware devices such as disk drives, network interfaces, and GPUs.

- **Interrupt management:** The kernel's interrupt management subsystem is responsible for handling hardware interrupts, which occur when a device needs the attention of the kernel.

- **Security:** The kernel's security subsystem is responsible for managing access to system resources, such as files and devices, and enforcing security policies.

- **Scheduling:** The kernel's scheduling subsystem is responsible for determining which process should run next, and allocating CPU time to processes.

### Fork() and wait

1. Less than -1 : Meaning wait for any child process whose process group ID is equal to the absolute value of pid.
2. Equal to -1 : Meaning wait for any child process.
3. Equal to 0 : Meaning wait for any child process whose process group ID is equal to that of the calling process.
4. Greater than 0 : Meaning wait for the child whose process ID is equal to the value of pid.

# 3. Computer Networks

### Djikstras algorithm described shortly:

Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge weights, producing a shortest path tree. This means that the algorithm finds the shortest path from one particular source node to all other nodes in the graph. The algorithm repeatedly selects the node with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller.

### Distance vector algorithm described shortly:

Distance vector routing is a method used by routers in a computer network to determine the best path for forwarding packets. In distance vector routing, each router maintains a table of the shortest distances to every other network, as well as the next hop on the path to that

network. The basic function of distance vector routing is for each router to share its table of distances with its neighbors, and for each router to update its own table based on the information received from its neighbors. This process is known as distance vector algorithm or Bellman-Ford algorithm. The algorithm is iterative and each router continues to update its distance vector until the values in the vector stabilize and no further updates are made.

## The 5 Different layers shortly described:

The OSI (Open Systems Interconnection) model is a framework that is used to understand and describe the different layers of a computer network. The OSI model consists of 7 different layers, each with its own specific functions and responsibilities.

- **Application Layer:** The highest layer of the OSI model, the application layer is responsible for providing the interface between the network and the applications that run on it. This layer is responsible for providing services to the user such as email, file transfer, and web access.

- **Transport Layer:** The transport layer is responsible for providing end-to-end communication between devices on the network. It is responsible for providing reliable communication, flow control, and error recovery.

- **Network Layer:** The network layer is responsible for providing routing and switching services. It is responsible for determining the best path for data to travel from one device to another and for managing the flow of data across the network.

- **Link Layer:** The link layer is responsible for providing communication between devices on the same network segment. It is responsible for providing services such as error detection, flow control, and addressing.

- **Physical Layer:** The lowest layer of the OSI model, the physical layer is responsible for providing the physical connection between devices on the network. This layer is responsible for the transmission of bits over the physical medium such as copper or fiber-optic cables.

## The 5 Different layers shortly described and which protocols apply to each of them:

**Physical Layer:** This layer deals with the physical connection between devices, such as cables and switches. Protocols that apply to this layer include Ethernet and Wi-Fi.

**Data Link Layer:** This layer provides a reliable link between devices on a network. Protocols that apply to this layer include MAC (Media Access Control) and LLC (Logical Link Control).

**Network Layer:** This layer is responsible for routing and forwarding packets through a network. Protocols that apply to this layer include IP (Internet Protocol) and ICMP (Internet Control Message Protocol).

**Transport Layer:** This layer provides end-to-end communication between applications. Protocols that apply to this layer include TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

**Application Layer:** This layer interacts directly with the application and end-user, and provides services such as email, file transfer, and web access. Protocols that apply to this layer include HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), and SMTP (Simple Mail Transfer Protocol).

**(Exam question) Assume you were to connect to a website via HTTP, and that several weeks later you were to connect to the same website. Between the two connections the website has shifted to using a CDN that locates data physically closer to your location. What changes (if any) would you expect in the HTTP requests and responses? Justify your answer and state any assumptions you have made.**

When connecting to a website via HTTP, the changes in the requests and responses will depend on how the website's CDN is configured and how it interacts with the client. However, assuming that the CDN is properly configured and optimized for the client's location, there should be some changes in the HTTP requests and responses when connecting to the website after it has shifted to using a CDN.

In terms of the HTTP requests, the IP address of the server that is being requested may change, as the CDN will route the client to a server that is physically closer to their location. Additionally, the requests may contain information about the client's location, such as the IP address or geolocation data, which can be used by the CDN to make routing decisions.

In terms of the HTTP responses, the most significant change that you would expect is that the response time should be faster due to the data being physically closer to the client. The CDN will use the client's location to route the request to a server that is physically closer to them, reducing the distance the data has to travel, which in turn reduces the time it takes for the response to be received by the client. Additionally, CDN may use other techniques such as caching to improve the performance further.

It's also worth noting that this is assuming that the CDN is properly configured and that the client is connecting to the website via CDN-enabled URL, if the client is connecting via the original URL the changes in the HTTP requests and responses may not occur.

## Describe CDN in network: content distribution network

Content Distribution Networks (CDN s) are a system of distributed servers that are deployed in multiple data centers around the world. The goal of a CDN is to provide faster delivery of content to users by caching a copy of the content on servers that are closer to the user's location.

CDN s can be used to speed up the delivery of a wide variety of content, including web pages, images, videos, audio and video streaming, software downloads and more. They work by intercepting requests for content from users and redirecting them to the nearest server that has a cached copy of the content. This reduces the distance that the data needs to travel, which can result in faster delivery times.

CDN s also provide other benefits such as:

- Reducing the load on the origin server, which can improve its performance and availability.
- Protecting the origin server from DDoS attacks.
- a way to serve content to users in areas where the origin server's network infrastructure is poor.
- Providing a way to serve large files by breaking them into smaller chunks and serving them from multiple servers

CDN providers offer a variety of services, including caching, load balancing, and distributed denial-of-service (DDoS) protection. They also offer analytics, reporting, and customization options to help optimize the performance of your content delivery. CDN providers also offer a range of pricing options, including pay-as-you-go and subscription-based models, making it easier for businesses of all sizes to use a CDN to improve their content delivery.

## CDN - Bring Home:

A second design philosophy, taken by Limelight and many other CDN companies, is to bring the ISPs home by building large clusters at a smaller number (for example, tens) of key locations and connecting these clusters using a private high-speed network.

## CDN - Enter Deep.

One philosophy, pioneered by Akamai, is to enter deep into the access networks of Internet Service Providers, by deploying server clusters in access ISPs all over the world. (Access networks are described in Section 1.3.)

## Congestion control vs. flow control

- **Flow control** is a mechanism used to prevent a fast sender from overwhelming a slow receiver. It ensures that the receiver can handle the data being sent to it, by regulating the rate at which the sender sends data. Flow control is mainly used at the data link layer. ( only has to do with the receiver). It is done by having the receiver send a window of how much it can receive with its messages.

- **Congestion control**, on the other hand, is a mechanism used to prevent network congestion by regulating the rate at which data is sent into the network. Congestion occurs when too much data is sent into the network, causing network resources such as bandwidth and buffer space to become exhausted. Congestion control is mainly used at the transport layer. (Har mere at gøre med den fysiske “tykkelse” på ledingen

og hvor meget der kan sendes.) This is done by fx sing the slowstart, congestion control and so on.

In summary, flow control is used to prevent a sender from overwhelming a receiver, while congestion control is used to prevent network resources from becoming exhausted.

**(Exam question) How does flow control in TCP help with congestion control? Why is flow control not enough to deal with congestion control and conversely congestion control not enough to deal with flow control?**

Flow control in TCP helps with congestion control by allowing the receiver to limit the amount of data that the sender can send, preventing the sender from overwhelming the network. Congestion control adjusts the sending rate of the sender based on network conditions to prevent packet loss and delays. Both flow control and congestion control are necessary for efficient and stable network communication.

## Describe Go-Back-N:

Go-Back-N is a type of error control protocol used in computer networks to ensure the reliable transfer of data between devices. It is a type of sliding window protocol, which means that it uses a fixed-size window to control the flow of data between devices.

In Go-Back-N, the sender sends multiple packets of data in sequence, and the receiver acknowledges receipt of each packet. If the sender does not receive an acknowledgement for a packet within a certain period of time, it will assume that the packet was lost and retransmit it. If the receiver receives a packet out of order, it will discard it and request a retransmission of that packet.

Once the sender receives an acknowledgement for a packet, it will slide the window forward to the next unacknowledged packet, allowing it to continue sending data. Go-Back-N is efficient in terms of network bandwidth usage, as it does not require the sender to wait for an acknowledgement for every packet sent, but if a packet is lost, all packets sent after it will be retransmited which leads to waste of bandwidth.

## Describe selectivxe repeat:

Selective Repeat is a type of error control protocol used in computer networks to ensure the reliable transfer of data between devices. It is a variation of the sliding window protocol, like Go-Back-N and it is used in situations where the network has a high bit error rate.

In Selective Repeat, the sender sends multiple packets of data in sequence, and the receiver acknowledges receipt of each packet. However, unlike Go-Back-N, if the receiver receives a packet out of order, it will buffer it and request a retransmission of only the missing packets. This allows the receiver to continue processing the correctly received packets, reducing the delay caused by retransmitting all packets.

The sender will keep track of the acknowledged packets, and retransmit only the missing packets that have not been acknowledged by the receiver. This allows for more efficient use of network bandwidth, as only the missing packets are retransmitted, rather than all packets as with Go-Back-N.

Selective Repeat protocol is more efficient than Go-Back-N in terms of network bandwidth usage and reducing retransmissions but it requires a larger buffer in the receiver side to store the out-of-order packets.

## BitTorrent:

BitTorrent is a peer-to-peer file sharing protocol that allows users to share and download large files in a decentralized manner. Instead of relying on a central server to host and distribute a file, BitTorrent allows users to download small pieces of the file from multiple sources, called "peers," at the same time. This allows for faster and more efficient file sharing, as well as reducing the load on any one server or network.

The BitTorrent protocol uses a system of "torrent" files to facilitate the sharing and downloading process. A torrent file is a small file that contains information about the larger file(s) being shared, including the file name, size, and the location of a "tracker" server. The tracker server keeps track of which peers have which pieces of the file and helps connect new downloaders with existing seeders.

Once a user has downloaded a torrent file, they can open it using a BitTorrent client, such as uTorrent or BitTorrent. The client connects to the tracker server and begins downloading pieces of the file from other peers. As the client downloads pieces of the file, it also begins to upload those same pieces to other peers, effectively becoming a "seeder" itself.

The key feature of BitTorrent is that it allows for efficient file sharing by breaking large files into smaller pieces and downloading those pieces in parallel from multiple sources. This allows for faster download times and helps to distribute the load across multiple users rather than relying on a single server.

## DNS:

DNS er en forkortelse for Domain Name System. Systemet oversætter domænenavne til internettets IP-adresser via såkaldte navneservere. Alle placeringer på internettet har en talkode, som kaldes en IP-adresse. IP-adressen gør computere i stand til at finde en specifik placering på internettet.

DNS  dns stands for Domain Name System. It is a hierarchical, decentralized system for resolving domain names, such as www.example.com, into IP addresses. The IP address is the numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication. DNS allows users to access websites and other resources by typing domain names instead of IP addresses. DNS servers are responsible for resolving domain names to IP addresses and caching the results for a certain period of time. (If there is no caching, then ill have to find it every time) These servers are organized in a

hierarchical structure, with the highest level servers being the root servers. The lower level servers are responsible for specific domain names or subdomains. DNS is a critical service that enables the internet to work as it does today by allowing easy to remember domain names instead of IP addresses.

## The hierarchical structure of IP addresses:

An IP address is considered hierarchical because it is divided into different fields that represent different levels of the network hierarchy. The most common format of an IP address is IPv4, which is a 32-bit address divided into four octets. Each octet represents a different level of the network hierarchy:

- The first octet represents the network address, also known as the "class" of the IP address
- The second and third octets represent the subnet address
- The fourth octet represents the host address

This hierarchical structure allows for efficient routing and management of IP addresses.

The hierarchical structure of IP addresses solves several problems:

- Routing scalability: The hierarchical structure of IP addresses allows routers to make forwarding decisions based on the network address rather than the entire IP address. This means that routers only need to store the routing information for the networks they are directly connected to, rather than for every individual host. This improves routing scalability and reduces the amount of memory and processing power required for routers.

- Address aggregation: The hierarchical structure of IP addresses allows for address aggregation, which is the process of combining multiple subnets into a single routing entry. This reduces the number of routing entries required and reduces the size of routing tables, which in turn reduces the amount of memory and processing power required for routers.

- Hierarchical addressing: The hierarchical structure of IP addresses allows for a hierarchical addressing scheme, which is a way of allocating IP addresses that reflects the topological structure of the network. This allows for efficient routing and management of IP addresses, and makes it easier to add and remove devices from the network.

Overall, the hierarchical structure of IP addresses allows for a more efficient and scalable use of IP addresses, making it easier to manage and route traffic on the internet.

## Octet:

An octet is a unit of digital information that consists of eight bits. In the context of IP addresses, an octet refers to one of the four 8-bit fields that make up an IPv4 address. Each

octet is represented by a decimal number between 0 and 255, and is separated by a period (.) in the standard notation for an IPv4 address. For example, the IP address "192.168.1.1" has four octets, with the values "192", "168", "1", and "1" respectively. In IPv6 addresses, an octet is represented by a 16-bit field (2 bytes) instead of 8-bit field.

## General dealing with a client request:

a client sends a query to its local DNS server and receives a response back. Much can go on under the covers, invisible to the DNS clients, as the hierarchical DNS servers communicate with each other to either recursively or iteratively resolve the client's DNS query. (KR s. 177)

**Fra eksamen: DNS is a distributed, hierarchical Database. Explain what this means, and how a client request is dealt with by the DNS system. For this answer assume there is no DNS caching:**

DNS information is not stored in one location, but a collection of servers worldwide. These exist at different abstraction layers with root servers at the top serving all below. Top-level domains server specific domain names (eg. .com, .dk). Individual organisations have their own DNS servers at the bottom. This makes the system more robust by removing a central failure point, and quicker to respond by having servers closer to worldwide requests.

When a client requests a domain lookup they send a request to their local DNS server. Assuming no cache, the local DNS will not be able to help directly, but will then make a request to the root DNS server. This will parse the domain name to identify which top level domain can parse the request. The root will reply to the local DNS with a TLD DNS server address, and the local DNS sends a request to this. The TLD server parses the authoritative DNS within the domain and sends this address to the local DNS. The Local DNS sends a request to the authoritative DNS and retrieves the IP of the domain, which is sent to the Local DNS. It can finally reply to the client with the IP.

**Fra eksamen: What role does DNS caching play in DNS lookup? Does this benefit iterative or recursive queries more? Justify your answer.**

DNS caching allows the different levels of DNS servers to save previous DNS search results. This means that if it gets a request again it can immediately respond without having to make further queries. This speeds up DNS replies. It also means far fewer DNS requests being sent over the network.

The recursive version benefits most from it because it doesn't return before it has found the server, meaning that the whole road to the server is cashed, while in the iterative only one step is cashed.

The DNS protocol runs over UDP and uses port 53.
DNS Caching K&R page 130.

## Iterative DNS or Recursive DNS:

Iterative DNS and Recursive DNS are two different methods of resolving domain names to IP addresses.

**Iterative DNS** is also known as "non-recursive" DNS. In this method, a DNS client (such as a web browser) sends a query to a DNS server asking for the IP address of a specific domain name. If the DNS server doesn't have the answer, it returns a referral to another DNS server that the client can query. The client then sends a new query to the referred server and continues this process until it reaches a server that has the answer or a server that can't provide any more referrals. This process is known as "iterating" through the DNS hierarchy.

**Recursive DNS,** on the other hand, is a "recursive" method of resolving domain names. In this method, the DNS client sends a query to a DNS server and the server takes on the responsibility of finding the answer on behalf of the client. The server may query other DNS servers in the process, but it doesn't return referrals to the client. Instead, it follows the referrals itself, until it reaches a server that has the answer. The server then returns the answer to the client.

**In summary**, Iterative DNS is a method in which the client queries a DNS server, and the server gives referrals if it does not have the answer, whereas Recursive DNS is a method in which the DNS server takes the responsibility of finding the answer for the client and don't give referrals. The recursive DNS is more efficient in terms of network and client resources, but the iterative DNS is more secure because it does not rely on a single DNS server.

**DNS and CDN:**

Most CDNs take advantage of DNS to intercept and redirect requests; an interesting discussion of such a use of the DNS is [Vixie 2009]. Let’s consider a simple example to illustrate how the DNS is typically involved. Suppose a content provider, NetCinema, employs the third-party CDN company, KingCDN, to distribute its videos to its customers. On the NetCinema Web pages, each of its videos is assigned a URL that includes the string “video” and a unique identifier for the video itself; for example, Transformers 7 might be assigned http://video.netcinema.com/6Y7B23V. Six steps then occur, as shown in Figure 2.25:

1. The user visits the Web page at NetCinema.
2. When the user clicks on the link http://video.netcinema.com/6Y7B23V, the user’s host sends a DNS query for video.netcinema.com.
3. The user’s Local DNS Server (LDNS) relays the DNS query to an authoritative DNS server for NetCinema, which observes the string “video” in the hostname video.netcinema.com. To “hand over” the DNS query to KingCDN, instead of returning an IP address, the NetCinema
authoritative DNS server returns to the LDNS a hostname in the KingCDN’s domain, for example, a1105.kingcdn.com.

4. From this point on, the DNS query enters into KingCDN's private DNS infrastructure. The user's LDNS then sends a second query, now for a1105.kingcdn.com, and KingCDN's DNS system eventually returns the IP addresses of a KingCDN content server to the LDNS. It is thus here, within the KingCDN's DNS system, that the CDN server from which the client will receive its content is specified.
5. The LDNS forwards the IP address of the content-serving CDN node to the user's host.
6. Once the client receives the IP address for a KingCDN content server, it establishes a direct TCP connection with the server at that IP address and issues an HTTP GET request for the video. If DASH is used, the server will first send to the client a manifest file with a list of URLs, one for each version of the video, and the client will dynamically select chunks from the different versions.

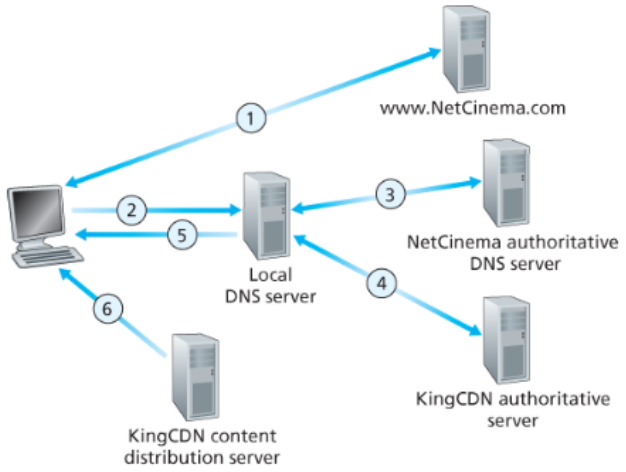



**Figure 2.25 DNS redirects a user's request to a CDN server**

## Count to Infinity problem:

The count-to-infinity problem is a problem that can occur in distributed systems, particularly in distance-vector routing algorithms. It is a problem that can happen when two or more nodes in a network are trying to determine the best path to a destination.

The problem arises when two nodes, A and B, both have an incorrect distance to a destination, C. Each node assumes that the other node has the correct distance and starts incrementing its own distance in an attempt to converge on the correct value. However, since both nodes have incorrect distances, they will continue to increment their distances indefinitely, causing their distance values to grow larger and larger. This is referred to as "counting to infinity."

The problem can be mitigated by using a metric such as Bellman-Ford or Dijkstra algorithm, which incorporate a mechanism such as a maximum hop count or a timeout to prevent the distance values from growing indefinitely. The idea behind these algorithms is to prevent the nodes from counting to infinity by introducing a maximum value for the distance, when this

maximum value is reached the node will consider that the information is incorrect and discard it.

In summary, The count-to-infinity problem is a problem that can occur in distance-vector routing algorithms when two or more nodes in a network are trying to determine the best path to a destination, and the nodes keep incrementing the distance values indefinitely, leading them to become larger and larger, referred to as counting to infinity. This problem can be mitigated by using a different routing algorithm that incorporates a mechanism to prevent the distance values from growing indefinitely.

## Man in the middle attack:

A man-in-the-middle (MITM) attack is a type of cyber attack in which an attacker intercepts and alters communications between two parties without their knowledge or consent. The attacker acts as a "man in the middle" by intercepting and manipulating the communication between the two parties, allowing them to steal sensitive information, inject malware, or perform other malicious actions.

In a MITM attack, the attacker can intercept the communication by either physically accessing the network or by using various techniques such as IP spoofing, DNS spoofing, or ARP spoofing.

Once the attacker has intercepted the communication, they can use various techniques to manipulate it, such as:

- Eavesdropping: the attacker can read and record the intercepted communication
- Modifying: the attacker can alter the intercepted communication to inject malware or steal sensitive information
- Injecting: the attacker can inject new communication into the intercepted communication to trick the parties into revealing sensitive information

MITM attacks can happen in various forms and scenarios such as:

- WiFi MITM: in this scenario, the attacker sets up a rogue wireless access point and tricks users into connecting to it, allowing the attacker to intercept and alter their communication.
- Web MITM: in this scenario, the attacker intercepts the communication between a user and a web server by using a technique called SSL stripping.

MITM attacks can be prevented by using methods such as:

- Encryption: encrypting the communication can prevent the attacker from reading and altering it
- Authentication: using strong authentication methods can prevent the attacker from impersonating one of the parties
- Network segmentation: segmenting the network and limiting the scope of the attack can prevent the attacker from accessing sensitive information.

In summary, Man-in-the-middle (MITM) is a type of cyber attack in which an attacker intercepts and alters the communication between two parties without their knowledge or

consent. The attacker acts as a "man in the middle" by intercepting and manipulating the communication, allowing them to steal sensitive information, inject malware or perform other malicious actions. The attack can be prevented by using methods such as encryption, authentication and network segmentation.

Nonce:
A nonce is a number that is used only once. In cryptography, a nonce is a random or unique number that is generated for each encryption operation. The nonce is included as part of the input to the encryption algorithm, along with the plaintext and a key. The nonce provides an extra level of security by ensuring that the same plaintext and key will not produce the same ciphertext even if the same encryption algorithm is used.

In some cases, the nonce is used to make sure that the data that is exchanged between the sender and receiver is unique and has not been tampered with. The nonce is sent along with the message and the recipient can check if it is unique or not, if the nonce is not unique then the message is dropped.

Nonces are widely used in various cryptographic protocols like in the WPA2 wireless security protocol and in the AES-GCM encryption algorithm.

## Playback attack:

A playback attack is a type of security attack where an attacker intercepts and records a valid communication, such as an authentication token or a session ID, and then replays it at a later time to gain unauthorized access to a system or service.

## Replay attack:

A replay attack is a type of network security attack in which an attacker intercepts and records a legitimate network packet and then retransmits it at a later time. The goal of a replay attack is to gain unauthorized access to a network or system by replaying a previously captured packet that contains valid authentication or session information.

Replay attacks can occur at different layers of the OSI model, but most commonly occur at the transport layer, where they can exploit the stateless nature of the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP).

For example, an attacker could intercept a packet that contains a valid username and password during an authentication process, and then replay that packet at a later time to gain unauthorized access to a network or system.

Replay attacks can be prevented by using methods such as:

- Timestamps: adding a timestamp to the packet to ensure that it can only be used for a limited time.

- Sequence numbers: assigning a unique number to each packet to prevent the replaying of a previously used packet.
- Cryptographic techniques: encrypting the packet to prevent the attacker from reading and replaying it.

In summary, a replay attack is a type of network security attack in which an attacker intercepts and records a legitimate network packet and then retransmits it at a later time to gain unauthorized access to a network or system. This attack can be mitigated by using methods such as timestamps, sequence numbers, and cryptographic techniques.

## How can you use encryption to make the protocol more secure? Explain which kind of encryption there is.

Encryption is a technique that can be used to make a protocol more secure by protecting the confidentiality and integrity of the data being transmitted. There are several types of encryption that can be used, each with their own strengths and weaknesses. Some of the main types of encryption include:

- Symmetric encryption: This type of encryption involves using a shared secret key to encrypt and decrypt the data. The same key is used to encrypt the data before it is transmitted and decrypt it upon receipt. Examples of symmetric encryption algorithms include AES, DES, and Blowfish.

- Asymmetric encryption: This type of encryption involves using a public and private key pair to encrypt and decrypt the data. The public key is used to encrypt the data, while the private key is used to decrypt it. Examples of asymmetric encryption algorithms include RSA, Diffie-Hellman, and Elliptic Curve Cryptography (ECC).

- Hash functions: This type of encryption is not used to encrypt the data but to ensure integrity of the data by creating a unique digital signature. When a message is hashed it produces a fixed-length output that will change if the original data is changed. Examples of hash functions are SHA-256, SHA-3 and MD5

- Stream ciphers: This type of encryption uses a pseudorandom stream of data, also known as a keystream, to encrypt the data. Examples of stream ciphers are RC4 and Salsa20

In summary, encryption is a technique that can be used to make a protocol more secure by protecting the confidentiality and integrity of the data being transmitted. There are different types of encryption that can be used, such as symmetric encryption, asymmetric encryption, hash functions and stream ciphers each with their own strengths and weaknesses. It's important to choose the right encryption algorithm and its implementation to ensure the best security for the specific use case.

## Which keys are there in network, and what do they do?

There are several different types of keys that can be used in a network, and their specific functions can vary depending on the context and the type of network being used. However, some common types of keys used in networks include:

**Encryption keys**: These keys are used to encrypt and decrypt data as it is transmitted over a network. They can be used to secure data from unauthorized access, such as by encrypting login credentials or other sensitive information.

**Authentication keys**: These keys are used to authenticate devices or users on a network. They can be used to ensure that only authorized devices or users can access the network or specific resources on the network.

**Session keys:** These keys are used to establish and maintain a secure session between devices or users on a network. They can be used to encrypt and decrypt data for a specific session, and are typically discarded or changed after the session is completed.

**Key management keys**: These keys are used to manage and distribute other types of keys on a network. They can be used to encrypt and decrypt keys for distribution, or to authenticate devices or users that are requesting keys.

**Digital signature keys**: These keys are used to create and verify digital signatures. They can be used to ensure the authenticity and integrity of data transmitted over a network.

It's worth noting that depending on the type of network, protocols, and security standards the keys may be used in different ways, and some may not be necessary in some cases.

## Non-cryptographic hash function:

- If a hash function is non-cryptographic, it means that it is fairly simple to reverse engineer the hash. It can be used only to check integrity of a message, since we can send a hash of the contents along the message, so the receiver can check if the message is altered(by purpose or error) (checksum)

## Cryptographic hash function:

- Is very hard to be reverse engineered. And can then be used to send secret data.

A cryptographic hash function has several properties that are different or stronger than those of a regular hash function. These properties are needed to make the cryptographic hash function suitable for use in various cryptographic applications such as digital signature, message authentication code (MAC) and password hashing.

- **Collision-Resistance**: It should be computationally infeasible to find two different input messages that produce the same output hash value, also known as a "collision".

- **Preimage-Resistance:** Given an output hash value, it should be computationally infeasible to find any input message that would produce that output hash value, also known as a "preimage".

- **Puzzle-friendliness**: It should be computationally hard to find any two input messages that have a difference of only one bit and produce very different output hash values.

- **Hiding property:** It should be computationally infeasible to determine anything about the input message by looking at its output hash value.

- **Avalanche effect**: A small change in the input message should cause a significant change in the output hash value.

- **Second preimage resistance:** Given an input message, it should be computationally infeasible to find a different input message that will produce the same output hash value.

- **Computational efficiency:** The hash function should be computationally efficient, meaning it should be able to process large input messages quickly.

- **Deterministic:** The output hash value should be deterministic, meaning that the same input message will always produce the same output hash value.

These properties are needed to protect the integrity and confidentiality of the data in various cryptographic applications, and to make it difficult for an attacker to find any weaknesses in the hash function.

## LRU replacement policy:

In an LRU scheme, the block replaced is the one that has been unused for the longest time. The set-associative example on page 397 uses LRU, which is why we replaced Memory(0) instead of Memory(6).

## TCP vs. UDP:

1. Connection-oriented vs Connectionless: TCP is a connection-oriented protocol, which means that a connection must be established between the sender and receiver before any data can be sent. UDP, on the other hand, is connectionless, which means that data can be sent without establishing a connection first.

2. Reliability: TCP provides a reliable connection, which means that data sent using TCP is guaranteed to reach the receiver. In contrast, UDP does not guarantee that data will reach the receiver, and packets can be lost or delivered out of order.

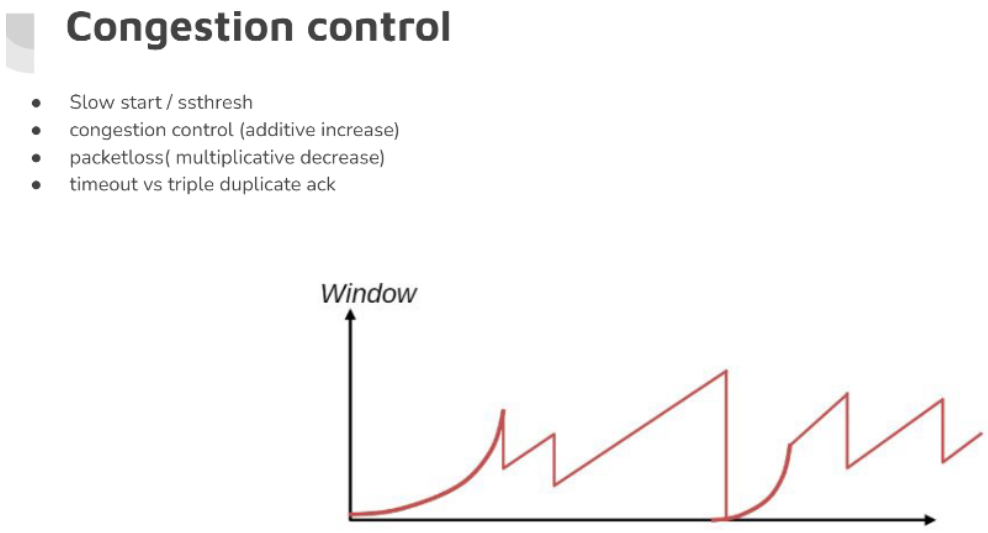
3. Flow Control: TCP uses flow control mechanisms to prevent the sender from overwhelming the receiver. UDP does not have flow control mechanisms, so the sender can potentially overwhelm the receiver.

4. Error Checking: TCP has error checking and correction mechanisms built in to ensure data integrity. UDP has minimal error checking, and any errors must be handled at the application level.

5. Overhead: TCP has more overhead than UDP because it establishes and maintains connections, performs error checking, and ensures data integrity. This can make TCP slower than UDP for some applications.

6. Use case: TCP is typically used for applications that require a reliable, ordered delivery of data, such as web browsing, email, and file transfer. UDP is typically used for applications that do not require a reliable, ordered delivery of data, such as streaming video and audio, online gaming, and DNS queries.

## Graph TCP:




Every step in the reno protocol:

- slow start
- triple duplicate loss
- Fast recovery ssthresh = control window/2
  ((new control window = control window/2) + 3)
- congestion control
- triple duplicate loss
- Fast recovery Fast recovery ssthresh = control window/2
  ((new control window = control window/2) + 3)
- congestion control
- timeout (control window = control window/2)
- slow start

- Reaches ssthresh
- congestion control

## asymmetric, public-key encryption on every communication:

**(Eksam question) Suppose there are k TCP connection ongoing over a shared link with bandwidth R. In addition to the k TCP connections, another TCP connection is initiated over this shared link. What is the bandwidth available to the new TCP connection and why?**
The bandwidth available to the new TCP connection would be affected by the other k ongoing TCP connections. The total bandwidth of the shared link, R, would be divided among the k+1 connections, which could lead to reduced bandwidth for the new connection.

In a shared link, the available bandwidth is divided among all the active connections using a technique called bandwidth sharing. The most common technique used for bandwidth sharing is the Transmission Control Protocol (TCP) congestion control algorithm, which uses a technique called "fair share" to divide the bandwidth among connections.

In fair share algorithm each connection is assigned a fair share of the available bandwidth, which is determined by the number of connections and the available bandwidth. In this case, with k ongoing TCP connections, the fair share of each connection would be R/k and the new connection would get (R/(k+1)) as its fair share.

It's worth noting that this is a simplified explanation and the actual bandwidth available to the new connection would also depend on the specific implementation of the congestion control algorithm used, as well as other factors such as packet loss and network delays.

AIMD stands for "Additive Increase, Multiplicative Decrease." It is a congestion control algorithm used in computer networks to control the rate at which data is sent to the network. In the AIMD phase, the sender increases its transmission rate (additive increase) as long as there is no congestion on the network. However, if congestion is detected, the sender reduces its transmission rate (multiplicative decrease) to avoid overwhelming the network. The rate at which the sender increases and decreases its transmission rate is typically based on a set of predefined parameters, such as the round-trip time (RTT) of the network or the number of lost packets. AIMD is widely used in various transport protocols such as TCP, and is considered as a standard congestion control algorithm.(Citer til Section 3.7.1 in KR)

**(Exam question) If instead of TCP, UDP had been used for the k connections, how much bandwidth would be available to the new UDP connection and why?**
If the k connections were using User Datagram Protocol (UDP) instead of Transmission Control Protocol (TCP), the bandwidth available to the new UDP connection would be different.

Unlike TCP, UDP does not have any built-in flow control or congestion control mechanisms. It is a connectionless and unreliable protocol, which means that it does not guarantee that packets will be delivered in order or that they will be delivered at all.

Because of this, in a shared link with k ongoing UDP connections, the new UDP connection would have access to the full bandwidth of the link, R. This is because UDP does not have any mechanism to control the rate at which data is sent or to manage the overall traffic in the network. Each connection would be able to send data at its own rate, potentially overwhelming the network and causing congestion.

It's worth noting that this scenario would not be an ideal one, as the lack of flow control and congestion control in UDP can lead to poor network performance, high packet loss and unpredictable delays. In a real-world scenario, using a different protocol like TCP or using a different algorithm for congestion control would be used to manage the network traffic more efficiently.

**(eksamen spørgsmål) Consider a network application that uses asymmetric, public-key encryption on every communication. What would you expect the shortcomings of this system to be? Suggest some potential improvements, and what effect(s) you would then expect to see?**
One potential shortcoming of a network application that uses asymmetric, public-key encryption on every communication is that it can be relatively computationally expensive, particularly for resource-constrained devices such as mobile devices or Internet of Things (IoT) devices. This can result in increased latency and reduced battery life. A potential improvement to address this issue would be to use a combination of asymmetric and symmetric encryption. In this approach, the client and server would use asymmetric encryption to establish a shared symmetric key, and then use that key for all subsequent communications. This would allow the initial key exchange to be done using the more computationally expensive asymmetric encryption, while still allowing for the bulk of the communications to be done using the more efficient symmetric encryption. Another potential shortcoming of a network application that uses asymmetric, public-key encryption on every communication is that it may be vulnerable to man-in-the-middle (MitM) attacks. A potential improvement to address this issue would be to implement certificate pinning. In this approach, the client would have a pre-configured list of trusted root certificate authorities (CA), and would only accept certificates from those CAs. This would prevent an attacker from being able to use a compromised CA to issue a fraudulent certificate and intercept the communication. Overall, using a combination of asymmetric and symmetric encryption and implementing certificate pinning would be expected to improve the performance and security of the system.

**(eksamens spørgsmål) A nonce is a common solution to a playback attack. What is a playback attack, and how does the nonce solve this problem?**
A playback attack is a type of network attack in which an attacker intercepts and records a valid authentication or encryption message, and then replays it at a later time to gain unauthorized access to a system.

A nonce (short for "number used once") is a unique value that is generated for each authentication or encryption message, and is included in the message as part of the authentication or encryption process. Because the nonce is generated anew for each message, it can only be used once, making it difficult for an attacker to replay a previously intercepted message.

When a message is received, the nonce is checked against a database of previously used nonces. If the nonce has already been used, the message is rejected, thus preventing a replay attack.

In summary, a nonce is a solution to a playback attack by ensuring that the message can only be used once, and that the message is coming from the expected sender, thus making it difficult for an attacker to replay a recorded message.

## Dijkstra og DV:

forwarding table: vælg en og sig af dem den ender hvilken vej skal der gås for at komme ud til noder.
Husk at bruge programmet.

## CDN vs. Peer-to-peer: (peer to peer)

A Content Delivery Network (CDN) is a system of distributed servers that are used to deliver web content to users based on their geographic location. The goal of a CDN is to reduce latency and improve the performance of web content delivery. This is accomplished by caching frequently requested content on servers located in strategic locations around the world, so that users can access the content from a server that is geographically closer to them.

Peer-to-peer (P2P) is a decentralized type of network, where each node (peer) acts as both a client and a server. In a P2P network, files and other resources are shared directly between users, rather than being stored on a central server.

The main difference between CDN and P2P is the way content is delivered. CDN uses a centralized system where content is stored on servers that are strategically placed, while P2P uses a decentralized system where content is shared directly between users. CDN is typically used for delivering static content such as images and videos, while P2P is typically used for sharing files such as music and movies. CDN is mainly used by large companies and websites with high traffic, while P2P is mainly used by smaller groups of people for sharing files among themselves.

## Describe the properties that are needed to make a cryptographic hash function (com- pared to normal hash functions).

Cryptographic hash functions have several properties that make them suitable for use in cryptography:

1. Deterministic: Given the same input, a cryptographic hash function will always produce the same output.
2. Preimage resistant: Given the output, it is computationally infeasible to find any input that would produce that output.
3. Second preimage resistant: Given an input, it is computationally infeasible to find any other input that would produce the same output as the original input.
4. Collision resistant: It is computationally infeasible to find any two distinct inputs that produce the same output.
5. Fast computation: A good cryptographic hash function should be able to produce a hash output quickly.
6. Output length fixed: A good cryptographic hash function should have a fixed length output.
7. Avalanche effect: Small changes to the input should result in significant changes to the output.

## Mac (not hierarchical)

There is 2^28 different mac addresses.

MAC (Media Access Control) addresses are unique identifiers assigned to network interfaces, such as Ethernet cards, Wi-Fi adapters, and so on. They are used to identify devices at the Data Link Layer of the OSI model, which is responsible for providing a reliable link between devices on a local area network (LAN).

The main purpose of MAC addresses is to provide a unique identifier for each device on a LAN, so that data can be properly delivered to the correct device. When a device on a LAN wants to send data to another device, it uses the destination device's MAC address to identify it. The data is then transmitted over the LAN in a special format called an Ethernet frame, which includes both the source and destination MAC addresses.

MAC addresses are not hierarchical because they are not divided into fields that represent different levels of the network hierarchy. Instead, they are fixed-length identifiers that are assigned to devices by their manufacturers. This makes them unique on the LAN, but they are not unique globally, as there's no concept of "network" or "subnet" on the MAC address.

MAC addresses are also not hierarchical because they are not used for routing purposes. Unlike IP addresses, which are used to route data between networks, MAC addresses are used only for identifying devices on a single LAN. The data is then delivered to the correct

device based on the MAC address, but it is not used for routing the data between different networks.

In summary, the purpose of MAC addresses is to identify devices on a LAN and to ensure that data is delivered to the correct device. They are not hierarchical in nature and are not used for routing purposes.

# 4 Machine architecture:

## Identify loops, conditionals, and function calls:

- **Loops:** Start by taking everything that can branch to somewhere new, and choose the ones that jomps back again. (jal doesnt alway loop)
- **Conditionals:** If statements like: bge osv
- **Function calls**: ret is a funcktion call.

```
L4:
   ret                exit function (jalr x0, x1, 0)
```

## Identify registers which are used for pointers:

- Look for sw and lw.

## Identify registers contains functions arguments and what are their type:

- Go through the code and find variables that is used but hasn't been assigned a value.

## Data cache:

- **Offset**: 2^x = blocksize  (hvor x er antallet af bits i byte offset.)
- **Block size:** 2^offset = block size
- **kb** = kibibyte
- **Antallet af blokke**: $\frac{størrelsen\ på\ hele\ cachen}{størrelsen\ på\ én\ blok}$
- **En -vejs: Antallet a sets** = 2^(antallet af index bits)
- **x -vejs: Antallet a index bit** = log2($\frac{antallet\ af\ af\ cache\ blocks}{x}$
- **Antallet af index bits** = log_2(antallet af sets)
- **Associativiteten:** $\frac{Antallet\ af\ blokke}{antallet\ af\ sets}$
- **Total size of cache:** $Antal\ blokke\ \ *\ \ størrelsen\ på\ én\ blok$
- **Byte**: 8 bits

## Cache mapping - How to solve this type of task:

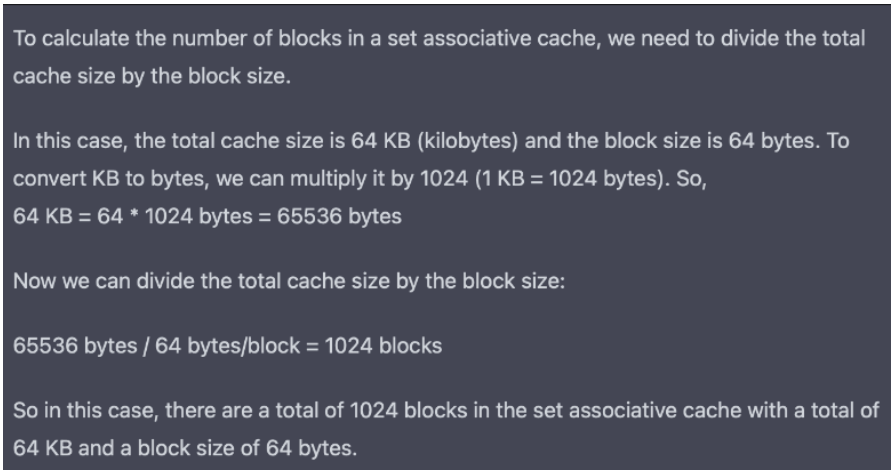
1. first you find the offset bits( the right-most bits of the address) and in this case, since this is a one-word block, we need 0 offset bits since (2^0 = 1)

2. then we see how many index bits we need(the next-to right-most address bits). Since we have 16 blocks we have 4 index bits because (2^4 = 16)
3. The remaining bits will then by the tag. In this case it will be 28 bits

For hver af de givet adresser vil man så finde index, offset og tag.

Index fortæller hvor henne i cachen vi skal ligge addressen hen i. Det bliver et miss hvis der ikke befinder sig noget. Hvis der ligger noget i det index men med et andet tag vil tagget blive overskrevet og det vil stadig være et miss. Hvis index og tag matcher så vil det være et hit og vi går videre.

## Cache:

- How many bytes is the total size of the cache?
    - Amount of blocks * the size of each block
- Calculate offset: block size = 2^x
    exs. (16bytes = 2^4 -> offset = 4)
- Calculate index: the amount of blocks/2 (if 2 set associative ect.)
- tag = everything without the offset and index.
- Number of blocks = (total cache size/block size)
    - Example with different cache and block representation:
- 


## Directmapped vs. set associative vs. fully associative:

In a **direct-mapped cache**, each block of memory can be stored in only one specific location in the cache. In a **set-associative cache**, each block of memory can be stored in any one of a group of cache locations. In a **fully-associative cache**, each block of memory can be stored in any cache location.

The main difference between the three is the number of options the cache has for mapping memory addresses to cache locations. A direct-mapped cache has the least amount of flexibility, while a fully-associative cache has the most.

In the context of cache memory, flexibility refers to the number of options the cache has for mapping memory addresses to cache locations. **A direct-mapped** cache has the least flexibility because each block of memory can be stored in only one specific location in the cache. This means that if that location is already occupied by another block of memory, a conflict occurs and the cache must choose which block to evict.

On the other hand, **a fully-associative** cache has the most flexibility because each block of memory can be stored in any cache location. This means that if a memory block needs to be stored and all the cache locations are already occupied, the cache can choose any location to evict, reducing the chances of conflict.

**A set-associative cache** has more flexibility than a direct-mapped cache but less than a fully-associative cache because each block of memory can be stored in any one of a group of cache locations. Downsides: this means the whole cache must be searched through to find a match or not.

## Write-back vs. write-through:

"Write-back" and "write-through" are two different cache write policies.

**"Write-back" caching** is a technique where changes made to the cache data are not immediately written to the main memory. Instead, the changes are temporarily stored in the cache and are periodically written back to the main memory in a process called "flushing". This method allows for faster writes since it reduces the number of writes to the main memory. However, it also increases the chances of data loss if there is a power failure or if the system crashes before the cache can be flushed.

**"Write-through" caching** is a technique where changes made to the cache data are immediately written to the main memory. This method ensures that data is always consistent between the cache and main memory and eliminates the risk of data loss due to power failure or system crash. However, it can slow down the write operation, since it requires an additional write to the main memory, which can be a bottleneck.

In summary, write-back is faster but can lead to data loss, while write-through is slower but more consistent and safe.

**(Exam question) The 5-step pipeline is expected to execute the program sequence faster than the 3-step pipeline because it can perform more instructions per clock cycle.**
A pipeline is a series of stages that are used to execute instructions in parallel. Each instruction goes through the pipeline stages one at a time, with each stage performing a specific operation. The more stages in a pipeline, the more instructions can be executed in parallel, increasing the overall performance of the pipeline.

In the case of the 5-step pipeline, it is able to perform less work in each step, which allows for a shorter longest signal path. This means that the pipeline can operate at a higher clock frequency without encountering the same timing constraints as the 3-step pipeline. Therefore, the 5-step pipeline can execute instructions 25% faster than the 3-step pipeline, which makes it execute the program sequence faster.

It is also worth noting that the performance gain from having more stages in a pipeline is not linear. The more stages there are, the more difficult it becomes to keep all stages busy and to keep the pipeline full. Therefore, the performance gain from having more stages in a pipeline will taper off as the number of stages increases.

## How to calculate latency?

To calculate latency in a pipeline, you can measure the time it takes for a data packet to travel from the input of the pipeline to the output. This can be done by inserting timestamp information at the input of the pipeline and comparing it with the timestamp information at the output. The difference between the two timestamps is the latency of the pipeline.

Another way to calculate latency in a pipeline is to measure the time it takes for a specific task or operation to be completed within the pipeline. For example, if a pipeline processes images, you could measure the time it takes for an image to be read from storage, processed, and written to a new location.

5.