Describe why network performance (bandwidth and latency) is reduced when you use Tor

Tor, short for The Onion Router, is designed to provide anonymity and privacy to users by routing their internet traffic through a series of volunteer-operated servers.

1. **Limited Bandwidth of Volunteer Nodes**: Tor relies on volunteers to operate nodes, and these nodes typically have limited bandwidth compared to commercial internet service providers. As a result, the overall bandwidth available for Tor users is constrained by the capacity of the slowest relay in the circuit.
2. **Encryption and Decryption Overhead**: The multiple layers of encryption and decryption that occur as data passes through each relay in the Tor network can introduce additional processing overhead. This cryptographic processing contributes to latency, as each relay needs to perform these operations before forwarding the data.
3. **Randomized Path Selection** Tor uses a randomized path selection mechanism for routing traffic through its network. While this randomness enhances anonymity, it can result in suboptimal routes, potentially causing delays and increasing latency.
4. **Exit Node Bottlenecks**: The exit node is the final relay in the Tor network before the traffic reaches its destination on the open internet. The bandwidth of the exit node can be a bottleneck, limiting the overall throughput of the Tor connection.
5. **UDP Traffic Handling**: Tor is primarily designed for TCP traffic, and handling UDP traffic introduces additional complexities and potential performance issues.

Describe how Tor and the Dark Web work

When you use Tor:

- Your connection is encrypted and routed through a series of volunteer-operated servers called nodes: the user contacts a "trusted" node to know the relays of the network and composes a path composed by at least 3 nodes (the first node is called guard and it's the only node which knows the sender)
- Each node in the Tor network only knows the IP address of the previous and next nodes in the chain, enhancing user privacy.
- The final node (exit node) connects to the destination server on your behalf, maintaining your anonymity.

The handshake protocol works using Authenticated Diffie-Hellman, using the public key of the node we're trying to connect to, also preventing Man-In-The-Middle (MITM) attacks. The key can be provided by trusted directory servers, or in the case of hidden services, it is actually encoded in the domain name itself (.onion v3).

1. **What happens if all the intermediate servers are malicious, are you safe?**

   If exit nodes are compromised, together with Guards (initial nodes), traffic correlation is possible and so the privacy is compromised. Since the same entity can observe both incoming and outgoing traffic, it knows both who is sending data and which site is receiving it.

   In the case of only intermediate nodes compromised, they do not know anything about the sender and the receiver, so it is perfectly fine as long as they are not guards or exit nodes.

2. **If you use TOR to visit CNN, is that the Dark Web?**

   No. This is just using Tor as a way to protect traffic from the sight of governments, ISPs and many other actors on the Internet. But the CNN is a classic website which is not inside Tor itself, and will be actually reached by the exit node.

   The Dark Web is a way to refer to Hidden Services. They are actually services that are reachable only from Tor, addressed using a .onion domain name. It is a way to protect the site and not only the client who connects to it. It's the case for sensible or illegal websites that must preserve their identity.

3. **How do you find hidden websites?**

   They can be found on other websites or search engines specialized in that, such as the Hidden Wiki, or someone can directly know their onion address.

   A hidden service registers itself to a directory server, attaching its public key and a set of introduction points.

How to connect to a Hidden Service on Tor

A hidden service registers itself to a directory server, attaching its public key and a set of introduction points.

A client asks the directory service the list of introduction points. The client picks a random node as a rendezvous point, and sends both the point and a random string to the introduction points.

The Hidden Service receives all of that, and if it is ok with that, it contacts the chosen rendezvous point to initiate the connection and finally communicates with the client.

No one knows who is talking to, and also all these links are using a **circuit** by themselves (a link of 3 or more nodes, randomly picked by the protocol).

Describe the notion of Proof Of Work in Bitcoin / Bitcoin consensus protocol

Bitcoin uses a consensus protocol based on Proof of Work. The nodes agree on one of the latest blocks in the blockchain, implicitly agreeing on all of the ones before that one. That's the case because inside the header of a block there is the Hash of the previous one. Keep going with this hash field for all previous blocks and we can see that all of them are implicitly agreed upon.

A miner is a fundamental actor in this context: it spends computational resources to find a random nonce that, inserted in a specific field in the block header, makes the final hash of the block lower than a certain threshold which determines the difficulty of finding the correct nonce, adjusted by the protocol.

Since the header of a block contains the hash of the previous one, after a number of blocks, usually 5, a block is considered finalized because changing its content will invalidate the hash of all the ones that refer to it as the predecessor (directly or indirectly) and so the nonce. Recalculating the nonce in a timely manner is considered infeasible.

Every other node will receive the block from one of the miners (the fastest one for that specific block) and will proceed validating it: all transactions will be validated and also the correctness of the nonce with respect to the criteria we've mentioned earlier.

Describe Failure Detectors and their taxonomy

	Strong Accurate	*Weak Accurate*	*Event. Strong Acc*	*Event. Weak Acc.*
Strong Complete	**Perfect**	**Strong**	**♦ Perfect**	**♦ Strong**
Weak Complete	Θ	**Weak**	**♦** Θ	**♦ Weak**

Completeness: if a process crashes, I'll know it after some time (<u>always eventual</u>)

- **Strong:** everyone knows all the failures

  $\forall\sigma \ \ \forall p \ \in crashed(\sigma) \ \forall q \in up(\sigma) \ \ \exists t: \forall t' \ > \ t \ \ , \ \ \ p \in D_q(t', \sigma)$

  For each run, for each crashed process P, **every** alive process Q after time T will know about P crash.

- **Weak**: someone knows about some crashes

  $\forall\sigma \ \ \forall p \ \in crashed(\sigma) \ \ \exists q \in up(\sigma) \ \ \exists t: \forall t' \ > \ t \ \ , \ \ \ p \in D_q(t', \sigma)$

  For each run, for each crashed process P, **exists an alive** process Q that after time T knows about P crash.

Accuracy: reduce false positives

- **Strong**: given 2 live processes, they never suspect each other
  $\forall\sigma \ \ \forall p, q \in up(\sigma) \ \forall t \ , \ \ p \ \notin D_q(\sigma, t)$

- **Weak**: at least 1 process, is never suspected by **anyone** to be crashed
  $\forall\sigma \ \ \exists p \in up(\sigma) \ \ \forall q \ \in up(\sigma) \ \ \forall t \ , \ \ p \ \notin D_q(\sigma, t)$

Eventual accuracy is the same, but not $\forall t$ but $\exists t : \forall t' \ > \ t$

Show how to make Paxos live with a Strong failure detector

Tip: discuss the fact that the failure detector is not always perfect but after some time (by definition)

A perfect failure detector is a specialization of a Strong failure detector.

A strong failure detector, by definition, is eventually right. In fact, only after a certain time t, every process will know that a process P has crashed some time before.

A perfect FD is also the specialization of a Strong accurate FD.

A strong accurate FD never reports a live process as crashed.

Leader Election: every process elects as a leader the process with the lower ID among the ones which it thinks are alive right now. The “election” is repeated every time the failure detector changes state.

1. Since eventually the failure detector state will converge, so does the elected leader
2. Having a single alive leader makes Paxos live

Does the common leader exist? Suppose a process picks a process P as its leader.

- **Can P be crashed?** Yes, but if it is, the FD will eventually change state and report P as crashed, thus changing the elected leader locally. The process repeats for a process Q (ID(Q) > ID(P)). In the worst case, the set of alive processes will contain only myself, so it’s guaranteed that it’s never empty.
- **Can P be considered crashed by someone else?** If P is alive, it cannot happen because the FD is strongly accurate, which means that an alive process is never reported as crashed by anyone.
- **Can process Q pick P' as leader?**
    - If ID(P’) > ID(P), that’s impossible for the point above
    - Otherwise, I am considering P’ as crashed. Since FD is strongly accurate, it is actually crashed, so Q will eventually converge to P.

Propose an implementation of a Failure Detector and discuss its properties in a synchronous system

Heartbeat protocol, under the assumption that all messages are delivered in time Δt at most.

1. Every process p, sends to everyone an Alive message every Δt time
2. Every other process $q \neq p$, if it doesn't receive any Alive message after Δt time from p, then $p \in D_q(\sigma, t')$

Strong Completeness

Under the assumption of a synchronous system in which messages are always delivered in a time bounded way without losing them, it is Strong Complete.

When p crashes, no one will ever receive a heartbeat message from it. After time Δt, other processes will report it as crashes. Also, this action is done by every process, so every process knows about the crash of anyone else

Strong Accuracy (not eventual)

Under the same assumption, a process must send the Alive message every fixed amount of time.

Since no message is lost, and it is time bound, every process will receive the Alive message from everyone before it can be considered crashed. So no one ever reports an alive process as crashed.

Unfortunately, in an asynchronous system, messages can be lost and are not time bound.

Describe briefly the Bit-Torrent system

BitTorrent is a peer-to-peer system used to distribute files efficiently all over the world.

When a user wants to download a certain file, first of all he must get a Torrent file, which is a descriptor that contains many information about the file to download, including:

- URL of the Tracker (= server that keeps info about the community at a global state level)
- Name, length and hash of each part of the file, usually 256KB each;
    - Splitting the file into several small parts is used to improve parallelism
    - Having their hash is used to avoid Pollution (having wrong parts of the file circulating around the network)
    - Newer versions use a Merkle tree to verify the hash of the parts of the file: this way, the data integrity of a single chunk can be verified using a logarithmic number of hashes, with respect to the total number of chunks.
    - Still, we have pollution if an entire wrong Torrent file is got by the user

Then, the Tracker returns a list of some peers who have the file ready to be uploaded.

Which parts of the file should I download first?

1. The Rarest first may be an approach, because it improves the redundancy of a rare part of the file and also it's safer to abort a download as soon as possible because of a part of a file is missing instead of realizing it only at the end of the process
2. Randomly: parts to download are chosen in a random way

End-game mode

At the very end of the download, in this mode, a piece of the file may be too slow to download because of several reasons, such as low bandwidth of the peer who is uploading.

In End-game mode, the remaining part is splitted more times and its download is parallelized, downloading from multiple peers.

Choking for Peers

In order to avoid saturating some peers and to stimulate a collaborative approach, where one peer also uploads and does not just perform downloads, **unchoking algorithms are used**.

First, a peer A unchokes B randomly, otherwise a new member of the network would have no chances to download anything and so it won't be able to upload either.

Another algorithm is **Tit for Tat**: A unchokes B if and only if B starts uploading other pieces to A (that A needs, of course).

"A" should not unchoke too many peers otherwise it'll have no bandwidth left to download what it actually needs.

Describe the CAP theorem

The CAP theorem states that it is impossible to have all of the following characteristics in a distributed system, simultaneously:

1. **Consistency**: a read of a variable always returns the latest written value
2. **Availability**: the system always responds with a valid value for a requested variable
3. **Partition Tolerance**: the system keeps working even if 2 partitions cannot communicate anymore between each other.

This is not possible because suppose to have a system partitioned in A and B.

At time t-1, the latest write for X is 0.

At time t, A and B cannot communicate with each other anymore.

At time t+1, A receives a write(x, 1)

At time t+2, B receives a read(x)

If B replies with 0, Consistency is lost.

If B replies with an error, Availability is lost.

B cannot reply with 1 because A and B cannot communicate, so it cannot be aware of the latest write.

CA-: banking systems

-AP: streaming services, CDNs, and so on

Availability is usually the most important feature for a distributed system.

Describe briefly the Akamai system

The idea behind Akamai is to serve static content on a website from a server geographically near the user who requests that content.

The implementation is based on the following steps:

- A website which wants to adopt this system will change the URL of the static content such as images to a URL managed by Akamai. cnn.com/static/image1.jpg may become something like cnn.static.akamai.com/image1.jpg
- A browser receives that URL and has to resolve cnn.static.akamai.com in a real IP address, using DNS
- The authoritative server responsible for the *.static.akamai.com domains will return the IP of the server(s) near the user, so that the latency will be lower when actually downloading the file and showing it to the final user in the webpage

This system loses Consistency (in the context of the CAP theorem), which is not so important in systems like CDNs.

What is Atomic/Sequential Consistency

Or other questions about Shared-Memory systems, which is a new topic

In Shared-Memory systems there are several types of possible memory consistency, each one of them with different properties and that can be implemented in different ways.

Two of them are the Atomic Consistency (aka Linearizability) and the Sequential Consistency.

Specifically:

- Atomic Consistency: in this type of consistency the properties are:
    a. Any read to a location must return the value written by the most recent write as per a global time reference
    b. All operations appear to be executed atomically
    c. All processes see the same sequence of reads and writes

  In this type of consistency every read and write can be saw as a sequence of <invocation, response> and so given a sequence SEQ of <inv, resp> it is said that SEQ is linearizable if exists a SEQ' of adjacent pairs <inv, resp> such that:

    1. For any variable v, the projection of SEQ' on v (called SEQ'v) is such that every read returns the most recent write
    2. If the resp of OP1 occurred before the invocation of OP2 then OP1 appears before OP2 in SEQ'

  This means that with this type of consistency we try to find a sequence of operations SEQ' that is the de-overlapped version of SEQ that is needed to let every process have the same vision of the events.

  The Linearizability can be implemented thanks to Total Order Broadcast

- Sequential Consistency: this is a weaker version of consistency wrt Linearizability in which the result of the computation is as if all the operations were executed in SOME order. The internal order of each process must be respected.

  Sequential Consistency is always possible when there is Linearizability (by respecting the same order) and it can be implemented with the Total Order Broadcast (without the broadcast of the reads)

Describe how a CDN works

A Content Delivery Network is a system which is used to improve the performance of serving static contents in different geographical locations.

It is a system that, from the point of view of a website's owner, improves the speed of the loading of the page, in a much cheaper way than renting servers geographically distributed.

A CDN works by storing and caching different static contents, distributing across the globe if they are heavily requested by users, and load balancing the requests.

Also, it can improve the performance of the dynamic content because many requests for static content don't need to be sent to the actual origin server because they are already cached by the CDN, so that it has more computational resources to compute dynamic content.

Some CDNs also offer the possibility to serve dynamic content executing small scripts on edge servers, located near the final user.

Akamai is a CDN service.

Describe how the DNS system works

Domain Name System is a hierarchical and distributed naming system used for translating human-friendly domain names into IP addresses.

1. The user enters a URL, so the client must resolve the domain name into an IP address (di.uniroma1.it)
2. The local DNS resolver receives the request and replies with the records associated, if it already has them in cache
3. Otherwise, the root DNS server is involved. It stores information about the TLDs servers, such as .com, .it and so on (.it)
4. The TLD DNS server will respond with the IP of the authoritative nameserver for the requested domain (.uniroma1.it)
5. The request is sent by the client to the authoritative nameserver, which will finally reply with the IP address(es) for the requested domain.

It can also reply with other information, based on the type of requested DNS records, such as A -> IPv4 address to visit, AAA -> IPv6 address, CNAME -> alias domain name, TXT -> text with various possible meanings, and so on.

Describe the four ways in which we can simulate a strong failure detector using a weak one

- θ to P

  In this case we have that in the θ there is only a single process that detects the crashed process and so we need some information sharing.

  Consider the following algorithm in which every process P do the following:

  initially: output = Ø

  when p changes:

  suspectsp <- Dp

  send(p, suspectsp) to everybody

  when msg is received by (p, Sp)

  output = (ouptputp U Sq) - {q}

  Broadcast the info so that everyone knows everything

Seminar: Pump & Dump in cryptos

In an unregulated market such as cryptocurrencies, market manipulation techniques are possible. Doing that on the stock market or other regulated markets is illegal.

The scheme involves artificially inflating the price of a coin (pump) and then quickly selling it as soon as possible, making its price drastically drop (dump).

The coordination of the pump action is usually conducted on Telegram or other privacy-oriented messaging systems by a team of people. Since they are the ones who perform the manipulation, they decide the coin to Pump. This way they can (and will) buy a huge amount of that coin, slowly from time to time to remain hidden from the public.

At a precise moment, they will release the name of the coin to pump to the members of the Telegram channel who will instantly buy large amounts of it, making the price go up.

The administrators, who bought the coin a long time before, will be the first ones to sell and the ones who make the greatest profit. Unfortunately, members are almost always the ones who get scammed because it's nearly impossible to sell right before the dump. Remember that the pump and dump may last for only very few minutes.

The members are acquired by stimulating the FOMO of the public and saying that everyone will earn a lot of money because only the outsiders will lose everything, which is false but members do not know that until they experience it first-hand.