Distributed Systems

Satvik Gupta

January 25, 2023

Distributed Systems

These notes are by no means comprehensive or complete

Nodes on the internet are identified by IP addresses. Data is passed from one router to another, moving each packet closer to its destination, in the hope that it will ultimately be delivered.

Each router must regularly be supplied with up-to-date routing tables. Individual IP addresses are grouped into prefixes. Autonomous Systems (AS) own these prefixes. Routing tables between ASes are maintained using Border Gateway Protocol (BGP).

Autonomous Systems

An Autonomous System can best be defined as a collection of IP routing prefixes, that are under the control of 1 or more network operators, on behalf of a single entity or domain, that **present a single,common and clearly defined routing policy to the Internet**.

- Each AS has a number (ASN).
  - There may be ASes without ASNs too. It is possible that an organization is running BGP using private AS numbers given by an ISP.
  - The ISP must have an officially registered ASN. The multiple private AS numbers are supported by the ISP.
  - The Internet, however, can only see the routing policy of the ISP. It's the ISP's duty to include the routing policies of the private AS numbers in its routing policy.

Transit vs Peering

Peering

Peering is when two ASes allow free-flow of traffic between them and their downstream customers. Peering relations are free of charge. Peers cannot see each other's upstreams.

Transit

Transit is when an AS (provider) agrees to:

- Direct the traffic of another AS(customer) to the internet
- Direct traffic from the Internet to the customer AS.

A transit fee is charged by the provider to the customer.

- The transit provider gives the customer a list of routes that belong to other ASes/ISPs that the provider is connected to. The transit client can send/receive traffic from those routes through the provider.
- The transit provider also advertises its customers' routes to its connections, so they know that traffic to those routes needs to go via the transit provider.

The transit provider may themselves use other transit networks too.

A transit-free network is a network based only on peering. These are generally the top-level ASes. They provide transit to smaller ASes, like ISPs.

IMPORTANT

A TRANSIT PROVIDER WILL NOT ANNOUNCE A PEER ROUTE TO OTHER PEERS, OR TO NETWORKS IT BUYS TRANSIT FROM

- If it did, it would be providing free transit over its network to its peers.
- Or worse, buying transit and giving it away for free to peers.

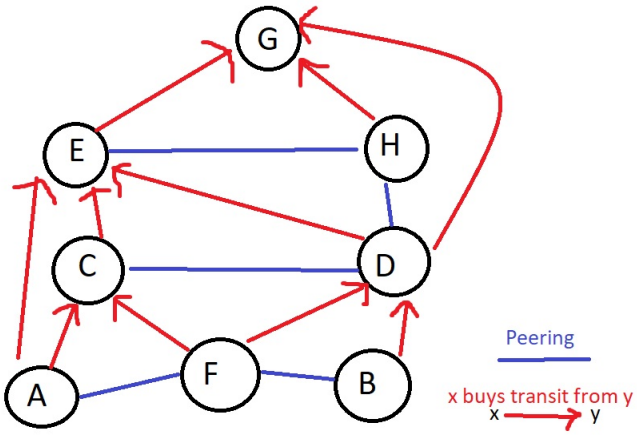

Figure 1: Transit and Peering Example Diagram

In the above represented network,

- G can see all networks since E,H and D buy transit from it.
- A can see F and customers of F, but it cannot see B through F.
- C can see B through D, but not through F.
  - D will want to tell B that it has access to C, to provide more incentive to B to buy its transit. After that, it will be obligated to let C know that it has access to B, since B bought transit from D. Therefore, C will see B through D.
  - F will not let C know it has access to B. If it did, F would have to pay for C->F->B transit.
- A can see B through C, but not through F.
- Traffic from C->H flows through E, but not through D.

BGP Hijacking

BGP was designed without security - it is based on trust. If an AS says it controls certain IPs, all its peers will believe it, and route app traffic to those IPs, to that AS. Security extensions and third-party validations exist, but they're not widely used.

An AS may announce an IP it doesn't own, or claim to have a shorter path to it than is already available - even if that "shortest" path doesn't actually exist.

Generally, ISPs filter BGP traffic. They allow BGP advertisements from their downstream networks to contain only valid IP space (i.e, IP addresses that the downstream network is known to possess). However, hackings have occurred because this isn't always true.

If an AS is hacked, this can be exploited. However, this will be quickly found out and reversed.

OSI Model

Open Systems Interconnection.

Widely referenced in academic literature, not often used exactly in industry. For example, the TCP/IP protocol doesn't fit neatly on top of the OSI model.

From lowest to highest, the 7 layers are:

1. **Physical Layer**

   Deals in bits. Handles transmission and reception of raw bit streams over a physical medium.

2. **Data Link**

   Deals in frames. Transmission of data frames b/w 2 nodes. Handles frame sync, errors, QoS, physical addressing, etc.

   - Physical Addressing, for e.g, using MAC addresses.
   - NOT Network Addressing.

3. **Network Layer**

   Deals in packets. Handles structuring and managing a multi-node network, including addressing, routing,etc.

4. **Transport Layer**

   Deals in segments/datagrams.

   Handles reliable transmission of data segments b/w points on a network. Handles segmentation, acknowledgment, multiplexing, handshakes, etc.

   - Segmentation is dividing large data into smaller sizes in order to match packet size imposed by network layer. This is known as MTU (Max Transmission Unit).

5. **Session Layer**

   Manages communication sessions, i.e, continuous exchange of information in the form of multiple back and forth transmissions btw 2 nodes. Creates the setup and controls the connection. Performs teardown. DNS is often put in this layer.

   Logon, name lookup, log off occur here.

   Authentication in FTP is built into session layer.

6. **Presentation Layer**

   Translates data b/w a network and an application. Includes character encoding, data compression and encryption/decryption.

   AKA Syntax layer.

   TLS/SSL is generally considered to be in this layer.

7. **Application Layer** High level protocols, such as HTTP or FTP. Generally include file sharing, message handling, DB access, etc.

The Internet Protocol Suite

The Internet Protocol Suite (TCP/IP) is different than the OSI model. Many layers are similar, but their differences start after the network layer.

TCP/IP layers don't fit neatly into OSI layers.

Layers in TCP/IP suite are:

1. Physical
2. Data Link
3. Network
4. Transport

5. Application

TCP/IP doesn't differentiate between session,presentation and application layer.

This is why it doesn't fit correctly into OSI.

For example, TLS is built on top of transport layer such as TCP. But, applications generally use it as if it was a transport layer.

Some mentions of TCP/IP don't even contain a physical layer. The Data Link and Physical layers are merged into a common *"Link"* layer. It is unspecified as if the Link layer does the job of the physical layer, or if TCP/IP assumes that physical hardware already exists underneath.

Application Based Multicasting

Nodes organize themselves into an overlay network that is used to spread information. Network routes aren't involved in group membership.

Starting a multicast

- Node wanting to start a multicast generates a multicast id *mid*.
- It looks up *succ*(*mid*) - the node responsible for that key. This node is promoted and becomes the root of the multicast.
- If P wants to join the multicast, it executes *lookup*(*mid*). This will send a message, from P to *succ*(*mid*), that P is requesting to join the *mid* multicast.
- This request message is sent via routing. While routing, let's say the message comes to node Q.
  - If Q has never seen a join request for *mid* before,
    - It will become a forwarder for *mid*.
    - P becomes a child of Q, and Q forwards P's join request to root.
  - If Q has seen a join request for *mid* before,
    - This means Q will already be a forwarder for *mid*.
    - P will become a child of Q, but its join request doesn't need to be forwarded to root anymore, as Q is already a member of the multicast tree.
- P is a forwarder for *mid* by definition.
- Messaging is done by sending a multicast message to root via *lookup*(*mid*).After that,root sends the message along the multicast tree.

Lamport Clocks

- If event a happens before event b, it is denoted by $a \rightarrow b$
- If a and b occur in the same process, they occur in the order they were observed.

  If

  $$T(a) \rightarrow T(b)$$

  then

  $$a \rightarrow b$$

- If a is the event of sending a message, and b is the event of the message being received (by another process),then $a \rightarrow b$.
- If a and b occur in 2 processes that never exchange any messages, then

  $$a \rightarrow b$$

  and

  $$b \rightarrow a$$

  are both false. This basically means that nothing can be said, and nothing needs to be said, about when the events happened, or which one happened first.

Lamport Algorithm

- Each process keeps an internal clock, that is incremented between 2 consecutive events.
- Let's say P sends a message that is received by the receiving process Q.
  - The event of sending the message is a.
  - The event of the message being received is b
- The message must also include the time when it was sent, i.e, $T(a)$.
- When the message is received, Q will set its internal clock to:

  $$max(T(a) + d, current\_clock)$$

  Generally, d is set to 1.

Totally Ordered Multicasting

Totally ordered multicasting is when events must occur in the same order on *all* nodes.

For example, consider the following scenario.

- A bank has 2 branches, each with their own copy of the database. One branch is in Mumbai, another in Delhi.
- A particular customer has 1000 in his account.
- Two events occur (nearly) simultaneously in real time:
  1. Bank HQ at Mumbai issues a 1% increase in amounts of customers, by interest or because of any other reason.
  2. The customer in Delhi deposits 100 in his account.
- In Mumbai, $1 \rightarrow 2$.

  The customer's final bank balance = 1110
- In Delhi, $2 \rightarrow 1$.

  The customer's final bank balance = 1111

This is an incosistency. The bank will be okay with either amount in the customer's account, but the amount on all nodes should ***match*** with each other.

How to solve this?

We solve this using totally ordered multicasting. The order of events must be the same on all nodes, even if the order is different than the real-world order of events. It should just be the same.

- Lamport Clocks are used for this.
- When a process receives a message, it's stored in a queue ordered by timestamp.
- The receiver multicasts an acknowledgement to all nodes.
- Since lamport clocks are being used, $T(ack) > T(msg)$.
- A message will only be delivered to the underlying application if it is at the head of the queue and has been acknowledged by *all* other nodes.

This ensures that all nodes have the same copy of the queue.

Replica Management

Replicas are created for servers, to reduce latency and to provide resilience. They need to managed efficiently and consistency needs to be maintained across all replicas.

Content Distribution

What data should we propagate?

- Propagate only notificiation of an update. Replica servers will request the actual data from the main server whenever they need it.
  - This is used when writes>reads.
  - Needs a properly implemented invalidation protocol.
- Transfer data from one copy to another.
  - Transfers can be aggregated.
  - Used when reads » writes.
- Propagate the update operation.
  - Don't transfer the data.
  - Tell each replica what update to perform.
  - Send only parameters for the updates.

Push vs Pull Semantics

Push

- Server Based.
- Updates are propagated to replicas without request.
- Server has to keep a list of client replicas as well as their caches.

Pull

- Client Based.
- Client has to explicitly request data.

Hybrid This approach is based on leases.

- Server issues leases to clients.
- While the lease is active, the server will push updates.
- After the lease has expired, the client has to pull updates, or request a new lease from the server.

Expiration Times for the leases can be based on:

- **Age** - If an object hasn't changed in a long time, it probably won't change in the future. We can provide a longer lease for such an object.
- **Renewal Frequency Based** - If a client requests an object more frequently, the expiration time for that client, for that object, will be higher.
- **State Based** - If a server has high load, it will grant leases with shorter expiry time.

Epidemic Protocols

- Used to implement eventual consistency.
- Propagate updates to all nodes in as few messages as possible.

Replicas can be of 3 types in this protocol.

- **Inactive** - Holds the update, and is willing and able to spread it.
- **Susceptible** - Hasn't received the update yet.
- **Removed** - Holds the update, but can't (or won't) spread it.

Anti-Entropy