Multicast: Repeated Unicast(easy impl. And deploy, keeps track of receivers grp, and does scale due to bandwidth usage). **IP mcast**(make copy of data at branching router, at most one copy of data per link, require router support, not used, grp of any size identified by class D IP and member located anywhere and join and leave grp on will, mcast is more dynamic and no idea about host location, host join grp on the basis of apps, routing is difficult in mcast, ucast has a prefix to announce and use shortest path for routing, mcast has a dissenmination tree from the sender to all receiver).**IGMP**("signaling or **quering**" protocol to establish(to know which grp they are interested in), maintain(up to date grp members), remove groups on a subnet, router grab traffic from internet and throws into LAN and interested grp pick the traffic, random TTL for querying, send leave grp msg if i am the most recent joiner otherwise leave quietly). **Mcast Routing(**build a distribution tree to reach all subnets(called **leaves** of the tree) of grp members, now use ucast to build the path for destination subnets or leaf nodes, **Source-Trees**:one tree for each sender, low delay becoz of shortest path, distributing traffic using more links but more routing states(#sources * #groups), **Shared Tree**:One tree connecting all receivers, longer delay, less #links, less routing states(#grps)). **DVMRP(**Basic distance vector protocol, sender flood to the networks and see who wants mcast pkt and prune receiver, every does that and make their tree, data driven, simple and robust, state & initial transmission overhead. reverse path check and reverse path broadcasting to use find shortest path for receiver, Reverse Path Multicast: Don't forward traffic onto networks with no receivers, prune(S,G)msg if no member in the LAN subtree, Grafting msg explicit reinstate the sub-tree**). Mcast OSPF(**Receiver flood link info & mcast address(announcing their membership). Every router is annotated by mcast add. Sender can run algo to compute the tree now. membership driven,computation overhead, cannot compute tree till we dont have source**).**

Core Based Tree(each grp has both mcast add. id and unicast address, send join-request to core using unicast to join the grp and core send a join-ack msg and add an interface in tree to that node, when core or any node before core receive pkt from source: that node convert that unicast pkt to multicast and send to the branches of the tree.Core should be in center of all receiver. Keep alive to check adjacent CBT, Quit-Request to parent to cut a branch if not member exist on that branch, if non-core parent fails: try join request again or flush the tree branch and force them to rejoin.**). PIM (**Protocol Independent Multicast: rely on underlying ucast routing protocol. Dense mode and sparse mode(support both shared and source tree). PIM hase a Rendezvous Point(RP) instead of core, encapsulate mcast pkt as ucast pkt(from src to RP) called register pkt, RP de-encapsulate the pkt. At the time of high data rate receive may ask src to give data directly using a join request. All receiver use hash base function to identify PR**).**

IPv6: TCP (works on segments) provide End-to-End unicast, Connection-oriented, Full duplex, reliable, in-order, byte-stream(Applications read from and write to a byte-stream but does transmit bytes, it convert to or receive as pkts form), with flow and congestion control. **Automatic Repeat reQuest** (ARQ:sender set a time for ack, if timer exp. And ack have not come, retransmit). **Stop-and-Wait**(Throughput = one pkt size / RTT). **Sliding Window(**Pipelining: send multiple outstanding segments (up to a limit), advance the window as the ACKs arrive, The max number of un-ACKed segments allowed is called window size, **) AdvertisedWindow or window(**contain buffer size of receiver and sender should not send data more than that**). Congestion(**Finite link bandwidth and buffer size at router. **Knee(**point after throughput increases very slow – Delay increases fast**). Cliff(**point after throughput starts to decrease very fast to zero (congestion collapse) – Delay approaches infinity**)). Performance(**Throughput = window size * pkt size / RTT, set both params correctly**) Setting Retransmission Timer(**Don't take sample of T(n) if retransmitted, Double Timeout after timeout, Reset Timeout for new packet when receiving ACK**).adjust window size(**cwnd: congestion window, flow_win: flow window, ssthresh: threshold size, use win = min(flow_win, cwnd)**). TCPTahoe(**Slow start:reach the knee point, cwnd=1 and cwnd has been doubled(1,2,4,8..) over one RTT when receive ack till cwnd < ssthresh. Else cwnd += 1/cwnd and on packet loss: ssthresh = cwnd/2; cwnd = 1**). TCP Reno(**for fast retransmission: Resend a segment after 3 duplicate ACKs(we know out of 4 only one got lost becoz we got 3 acks) and for fast recovery, set cwnd to cwnd/2 But when RTO expires still do cwnd = 1. No need to slow start again, cwnd oscillates around the optimal window size, problem: Flows with short duration don't get the benefit of high bandwidth and Flows with long delay respond/adapt too slow.**) TCP CUBIC(**Increase window size faster when it is far away from previous congestion point (Wmax), but slows down significantly when it is getting close to it.**)**

BBR: We want to fully utilize the bandwidth means high throughput and minimize delay or round trip latency. bandwidth delay product BW*RTT is the capacity of the pipe, ideal window size = BW*RTT/ segment size, BW and RTT are both dynamic moving targets. **BBR**(Based on measurement of throughput and delay rather than packet loss, Try to achieve high throughput without causing long queuing delay, deploy only sender side, Optimal point: BDP = (max BW) * (min RTT), BBR keep routers queue empty so that delay stays low. RTT calculated by ack.) Periodically send more bytes in flight to probe if more bandwidth has become available. Periodically send less bytes in flight to see if minimal RTT remains the same.

DNS: names can be flat, hierarchical, fixed or variable. DNS is variable length, mnemonic, tied to organizations. Before DNS, centrally managed file HOSTS.TXT (/etc/hosts) and It couldn't scale.

Components(hierarchical name space(Follow the hierarchy of organizations), distributed database(Provided by individual domain owners), Local resolvers(Provided by ISPs), client-server access protocol(on UDP port 53)). **Zones(** the entire DNS database is divided to a hierarchy of zones, zone: a continuous sub-space and domain can contain domains at different level, Each zone is controlled by an administrative authority with its own name server). **Recursive vs iterative query**. **Resource record** contains Name, value, TTL, type, class.**Performance**(Replication, caching at local server). **Infra RR**(NS Resource Record:provide name of zones authority server, A Resource Record: associated with NS record),**Affects DNS Availability(**s/w, n/w failure, Scheduled maintenance tasks, Configuration errors**).**

RON: IP scales well but Suffers slow outage detection and recovery, Detect badly performing paths, Efficiently leverage redundant paths, Multi-home small customers, Express sophisticated routing policy / metrics. **Overlay(**easily deployed, Internet focus on scalability, Keep functionality between active peers**).**

Routing(Probe between nodes, determine path, Store probe results in performance DB, Route along app-specific(link state) best path consistent with routing info stored in DB and Data handled by application specific mean we have different metric we have multiple table according to metric. throughput = sqrt(1.5**)**/rtt.p p=loss**) RON results(**Probe-based outage detection effective, Single-hop indirect routing works well, Scaling is explicitly not our forte but big enough, RONs improve packet delivery reliability**)**

Ethernet: CSMA: carrier sense multiple access and CD: collision detection(compare transmitted signals with received). **Why success(**east to maintain and inexpensive and good performance under light traffic). **Self learning(**Entry in switch table: (MAC Address, outgoing Interface, Time Stamp), Switch learns which hosts can be reached through which interfaces, Plug-and-play and Host mobility(self learn the new location of host), but Flooding causes duplicates and loops). **Spanning tree:(**Trim the topology into a spanning tree, then run the flood-based self-learning**).Ethernet Bridging(**Flat addressing, self-learning, flooding, forwarding along the tree**). IP Routing(**Hierarchical addressing, subnet and host configuration,Forwarding along shortest paths**). Seattle(**Switches run link-state routing with only their own connectivity info, Switches use hosts' MAC addresses for routing(zero conf. And comptible),Switches resolve end-hosts' location and address via hashing, Switches detect the arrival/departure of hosts, small table O(H) not O(S*H)**)**

DCN: Topology(edge, aggregation, core switches), **Fat** tree topology(Half port connect to lower layer and half to upper layer per switch), **PortLand(**Scalable layer-2 routing, single IP subnet, fat tree topo, design(Fabric manager(centralized process, maintain network conf., reduce cost of conf.), Positional pseudo MAC addresses(encodes the location of the host 48 bit(pod, position, port,vmid),),Proxy-based ARP, Distributed Location Discovery, Loop free forwarding, Fault tolerant routing), Loop Free Forwarding, fabric manager maintains the link states(up/down)in matrix)

DHT:search difficulty, Scaling, Adaptability(machine can go and come), security and privacy. Gnutella(flood the search req), Kazza(hybrid of Gnutella and Napster, has supernode to flood). DHT(Each node has an id = hash(ip), node maintain:Successor,Predecessor,Finger table, **Join Operation**(stabilize and notify msg to neighbors)). why(Distributed object lookup, Decentralized file system,Application layer multicast, Persistent storage)

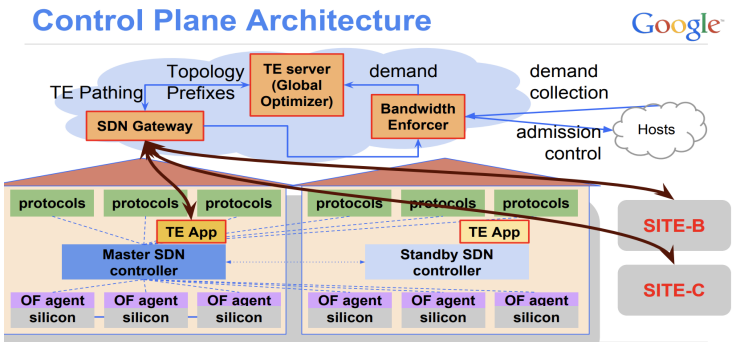
ALM:**Narada**(Richer overlay topology that includes all group members, but not a complete graph, Construct per-source trees using well known multicast routing algorithms (reverse path checking) components: Mesh Management and Optimization, Forwarding tree construction). Over time, add or drop virtual links to

improve the mesh quality, Keep the node degree under a threshold, Probe the members by probability of existence. Node i **adds** link i-j if utility(j) is greater than a threshold, Node i **drops** link i-j if consensus cost (j) is less than a threshold(ensure Stability and Partition Avoidance), **routing** is Similar to DVMRP.

Bittorrent:Multiple Dissemination Trees(using all links making multiple tree: ACB, ABC for chunk of data).

Torrent file(hashes of pisces and tacker(bootstrap new peers)). File sharing(tracker picks random peers and further select on the basis of data rate and verify hash after downloading parts). Choosing peer(choke and unchoke,Regular unchoke: a few peers with the largest upload rate to A, Optimistic unchoke: a randomly selected peer)

SDN: simple pkt forwarding h/w, networking OS, App (API bw OS and apps), or (we can virtualized slice os and apps) **Openflow(**southbound: interface bw os and hw, central OpenFlow Controller, Each forwarding element runs an OpenFlow Agent which implements southbound iface, openflow switch send packet header to controller and controller knows topology and tell the route**). Centralised traffic engineering. SND gives additional 30% throughput and decoupled sw and hw.**



HTTP-CDN:

scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment].

Proxy(Cache and serve web objects from proxies., shorter delay, reduced workload, less traffic, done at different locations, transparent or non-transparent to the client.**). Supporting caching: conditional GET(**: don't send object if proxy has up-to-date cached version, check using sending a request to server with if modified since with date if server says modified then send server response to client else cache response**). CDN(**Content providers, ISPs, and end users., CDN is another party to scale content distribution, Spread load to many surrogate servers, Shorter delay perceived by users., Less cost for ISPs as well, Absorb denial-of-service attack traffic, • Caching and management of the contents.**)**

IPv6 - Addr len 32 bits to 128 bits. IP options moved out of the base header. Header Checksum removed. Type of Service field removed. Fragmentation/Reassembly fields moved to options. Length field excludes IPv6 header. Time to Live->Hop Limit. Protocol->Next Header. Added Flow Label field. Transition From IPv4 To IPv6 : tunneling encapsulates a packet inside another header between two routers. **NAT**: Network Address Translation - Local network uses just one IP address(All datagrams leaving local network have same single source public IP address: 138.76.29.7, different source port numbers, Datagrams with source or destination in this network have private 10.0.0/24 address for source, destination). NAT Working(WAN side add(One IP and port), LAN side addr(local ip and port)) **Disadv**: limits the total number of connections supportable, Cannot run services from inside a NAT box. Reduced robustnes

NDN - Two types of packets Interest packet(content name, selecter, nonce) and data packet(content name, signature, signed info, data). Names are generated by apps, opaque to networks. **Model and naming**:(Every packet has its unique name Hierarchical names to identify relationship between data and facilitate name aggregation. The signature binds the name, content, and key). **Basic Operations - PUBLISHER** Announce name prefix(es). Name and sign data packet. Answer interests .**CONSUMER** Express interest packets for data by name, Receive data, verify signature, decrypt if necessary. **ROUTER** Route and forward Interest/Data based on names instead of addresses. **Address Independence and Location Independence.** (Content can be supplied by anyone from anywhere, as long as the names match, and the signature verifies. Fault tolerance Load balancing Mobility Data centric Security). Secure the content, not the container nor the channel. Force app developers to think about security from the beginning