How to share a link?Multiplexing,De-multiplexing,Synchronous Time-division Multiplexing, Time slots/data transmitted in predetermined slots. **Circuit-switched networks**:Traditionally used for telephone networks (ATM),Pre-allocate resources per connection,Pre-determine route on connection setup,Fixed bandwidth requirements per-connection;**Packet-switched networks**:Internet,No pre-determined allocation of resources (coming though),No connection setup,no pre-determined route More fault tolerance,Efficient utilization in the face of bursty traffic patterns .**TCP/IP PROTOCOL STACK**: Application Layer,Transport Layer,Network Layer,Link Layer. **How do you know when a packet has been lost?** Ultimately sender uses timers to decide when to retransmit • **But how long should the timer be?** • Too long: inefficient (large delays, poor use of bandwidth) • Too short: may retransmit unnecessarily (causing extra traffic) **End to end protocols** • Move from host-to-host to process-to-process communication model • TCP – provide abstraction of reliable in-order byte stream on top of IP protocol, sliding window, ACKs

SWITCHING, FORWARDING, AND ROUTING: Bridge used for transmitting frames between lans **CLASS-BASED ADDRESSING**:total 32 digits, Class A starts with 0, B starts with 1 0,14 digits for network and 16 for host. Class C starts with 1 1 0,21 network. 8 host. Class D (1110) for multicast, Class E (1111) experimental. Host bits all set to 0: network address • Host bits all set to 1: broadcast address. IP FORWARDING TABLES • Router needs to know where to forward a packet • Forwarding table contains: • List of network names and next hop routers • Local networks have entries specifying which interface • Link-local hosts can be delivered with Layer-2 forwarding • E.g. www.ucsd.edu address is 132.239.180.101 • Class B address – class + network is 132.239 • Lookup 132.239 in forwarding table • Prefix – part of address that really matters for routing **SUBNETTING** • Individual networks may be composed of several LANs • Only want traffic destined to local hosts on physical network • Routers need a way to know which hosts on which LAN • Networks can be arbitrarily decomposed into subnets • Each subnet is simply a prefix of the host address portion • Subnet prefix can be of any length, specified with **netmask.** **Subnet Address** Every (sub)network has an address and a netmask • Netmask tells which bits of the network address is important • Convention suggests it be a proper prefix • Netmask written as an all-ones IP address • E.g., Class B netmask is 255.255.0.0 • Sometimes expressed in terms of number of 1s, e.g., /16 • Need to size subnet appropriately for each LAN • Only have remaining bits to specify host addresses. **CIDR:** e.g. 10.95.1.2 contained within 10.0.0.0/8: • 10.0.0.0 is network and remainder (95.1.2) is host • Pro: Finer grained allocation, aggregation • Con: More expensive lookup: **longest prefix match** **UDP**: Provides unreliable message delivery between processes • Source port filled in by OS as message is sent • Destination port identifies UDP delivery queue at endpoint • Connectionless (no state about who talks to whom). UDP includes optional protection against errors • Checksum intended as an end-to-end check on delivery • So it covers data, UDP header, and IP pseudoheader (history) TCP:Reliable bi-directional bytestream between processes • Uses a sliding window protocol for efficient transfer • Connection-oriented • Conversation between two endpoints with beginning and end • Flow control (last lecture) • Prevents sender from over-running receiver buffers • (tell sender how much buffer is left at receiver) • Congestion control (later in term) • Prevents sender from over-running network capacity. **HTTP: HTTP** is a text oriented protocol.HTTP is a request/response protocol. The first line (START LINE) indicates whether this is a request message or a response message.Request Messages define • The operation (called method) to be performed • The web page the operation should be performed on • The version of HTTP being used.Response: Also begins with a single START LINE. • The version of HTTP being used • A three-digit status code • Text string giving the reason for the response.

NETWORKED STORAGE AND REMOTE PROCEDURE CALLS (RPC): NFS FILE HANDLES (FH): Opaque identifier provider to client from server • Includes all info needed to identify file/object on server. It's a trick: "store" server state at the client! • Generation #optional, depending on the underlying file system. **TANSTANFL**• With local FS, read sees data from "most recent" write, even if performed by different process • "Read/write coherence", linearizability • Achieve the same with NFS? • Perform all reads & writes synchronously to server • Huge cost: high latency, low scalability • And what if the server doesn't return? • Options: hang indefinitely, return ERROR. **SHOULD SERVER MAINTAIN PER-CLIENT STATE?** Stateful: Pros • Smaller requests • Simpler req processing • Better cache coherence, file locking, etc. Cons • Per-client state limits scalability • Fault-tolerance on state required for correctness. Stateless: Pros • Easy server crash recovery • No open/close needed • Better scalability • Cons • Each request must be fully self-describing • Consistency is harder. EXPLORING THE **CONSISTENCY TRADEOFFS**: Write-to-read semantics too expensive • Give up caching, require server-side state, or … • Close-to-open "session" semantics • Ensure an ordering, but only between application close and open, not all writes and reads. • If B opens after A closes, will see A's writes • But if two clients open at same time? No guarantees • And what gets written? "Last writer wins". **NFS CACHE CONSISTENCY**: Recall challenge: Potential concurrent writers • Cache validation: • Get file's last modification time from server: getattr(fh) • Both when first open file, then poll every 3-60 seconds • If server's last modification time has changed, flush dirty blocks and invalidate cache • When reading a block • Validate: (current time – last validation time < threshold) • If valid, serve from cache. Otherwise, refresh from server. **Some problems:** "Mixed reads" across version • Assumes synchronized clocks.• Writes specified by offset. **LOCK:** Problem: What if the client crashes? • Solution: Keep-alive timer: Recover lock on timeout • Problem: what if client alive but network route failed? • Client thinks it has lock, server gives lock to other: "Split brain". **Lease**: Client requests a lease • May be implicit, distinct from file locking • Issued lease has file version number for cache coherence • Server determines if lease can be granted • Read leases may be granted concurrently • Write leases are granted exclusively • If conflict exists, server may send eviction notices • Evicted write lease must write back • Evicted read leases must flush/disable caching • Client acknowledges when completed. **BOUNDED LEASE TERM SIMPLIFIES RECOVERY**: Before lease expires, client must renew lease • Client fails while holding a lease? • Server waits until the lease expires, then unilaterally reclaims • If client fails during eviction, server waits then reclaims • Server fails while leases outstanding? On recovery, • Wait lease period + clock skew before issuing new leases • Absorb renewal requests and/or writes for evicted leases.

R**PC(REMOTE PROCEDURE CALL): RPC's Goal**: To make communication appear like a local procedure call: transparency for procedure calls. **RPC ISSUES:** Heterogeneity • Client needs to rendezvous with the server • Server must dispatch to the required function • What if server is different type of machine? • Failure• Performance • Procedure call takes ≈ 10 cycles ≈ 3 ns • RPC in a data center takes ≈ 10µs (103× slower) • In the wide area, typically 106× slower. **How could it go wrong?**1. Client may crash and reboot 2. Packets may be dropped • Some individual packet loss in the Internet • Broken routing results in many lost packets 3. Server may crash and reboot 4. Network or server might just be very slow. **AT-LEAST-ONCE SCHEME**. Simplest scheme for handling failures 1. Client stub waits for a response, for a while • Response takes the form of an acknowledgement message from the server stub 2. If no response arrives after a fixed timeout time period, then client stub re-sends the request • Repeat the above a few times • Still no response? Return an error to the application. **SO IS AT-LEAST-ONCE EVER OKAY?**. Yes: If they are read-only operations with no side effects •Yes: If the application has its own functionality to cope with duplication and reordering. **AT-MOST-ONCE SCHEME:** Idea: server RPC code detects duplicate requests • Returns previous reply instead of re-running handler • How to detect a duplicate request? • Test: Server sees same function, same arguments twice • No! Sometimes applications legitimately submit the same function with same augments, twice in a row. **How to ensure that the xid is unique?** 1. Combine a unique client ID with the current time of day 2. Combine unique client ID with a sequence number • Suppose the client crashes and restarts. Can it reuse the same client ID? 3. Big random number. **CONCURRENT REQUESTS**: Problem: How to handle a duplicate request while the original is still executing? • Server doesn't know reply yet. Also, we don't want to run the procedure twice • Idea: Add a pending flag per executing RPC • Server waits for the procedure to finish, or ignores. **SERVER CRASH AND RESTART:** Problem: Server may crash and restart • Does server need to write its tables to disk? • Yes! On server crash and restart: • If old[], seen[] tables are only in memory: • Server will forget, accept duplicate requests.

VIRTUALIZATION AND CLOUD PLATFORMS: Concurrency: • Progress of multiple elements of the set overlap in time • **Parallelism:** • Progress on elements of the set occur at the same time. **Threads** share the same address space. Thread context switching can be done entirely independent of the operating system. 2 Process switching is generally (somewhat) more expensive as it involves getting the OS in the loop. 3 Creating and destroying threads is much cheaper than doing so for processes. **Why use threads:** Avoid needless blocking: a single-threaded process will block when doing I/O; in a multi-threaded process, the operating system can switch the CPU to another thread in that process. Exploit parallelism: the threads in a multi-threaded process can be scheduled to run in parallel on a multiprocessor or multicore processor. Avoid process switching: structure large applications not as a collection of processes, but through multiple threads. **Model & Characteristics:** Multithreading: Parallelism, blocking system calls; Single-threaded process: No parallelism, blocking system calls; Finite-state machine: Parallelism, nonblocking system calls. **Virtualizatio**n is important: Hardware changes faster than software Ease of portability and code migration Isolation of failing or attacked components. **Ways of virtualization** (a) Process VM, (b) Native VMM, (c) Hosted VMM. **Differences** (a) Separate set of instructions, an interpreter/emulator, running atop an OS. (b) Low-level instructions, along with bare-bones minimal operating system (c) Low-level instructions, but delegating most work to a full-fledged OS.

NAMING, DNS, AND CHORD: MAPPING BETWEEN IDENTIFIERS • Domain Name System (DNS) • Given a host name, provide the IP address • Given an IP address, provide the host name • Address Resolution Protocol (ARP) • Given an IP address, provide the MAC address • To enable communication within the Local Area Network • Dynamic Host Configuration Protocol (DHCP) • Automates host boot-up process • Given a MAC address, assign a unique IP address • … and tell host other stuff about the Local Area Network. **BROADCAST: ADDRESS RESOLUTION PROTOCOL (ARP)** • IP forwarding tables: one entry per network not host • Thus, routes designed to get packets to proper network • Network needs to take over from there to get to proper host • Address resolution protocol (ARP) translates IP addresses to link-level addresses • Broadcast request over network for IP➔link-level mapping • Maintain local cache (with timeout) **DHCP** Broadcast-based LAN protocol algorithm • Host broadcasts "DHCP discover" on LAN • DHCP server responds with "DHCP offer" message • Host requests IP address: "DHCP request" message • DHCP server sends address: "DHCP ack" message w/IP address • Easy to have fewer addresses than hosts and to renumber network (use new addresses) • What if host goes away (how to get address back?) • Address is a "lease" not a "grant", has a timeout • Host may have different IP addresses at different times? **A distributed system architecture**: • No centralized control • Nodes are roughly symmetric in function • Large number of unreliable nodes. **P2P ADOPTION:** Client-to-client (legal, illegal) file sharing •2. Digital currency: no natural single owner (Bitcoin) 3. Voice/video telephony: user to user anyway. **DHS hostname vs IP address:** DNS host name • Mnemonic name appreciated by humans • Variable length, full alphabet of characters • Provides little (if any) information about location • IP address • Numerical address appreciated by routers • Fixed length, decimal number • Hierarchical address space, related to host location

DNS: GOALS AND NON-GOALS: A wide-area distributed database • Goals: Scalability; decentralized maintenance • Robustness • Global scope(Names mean the same thing everywhere)Distributed updates/queries • Good performance • But don't need strong consistency properties. **DNS**: Hierarchical name space divided into contiguous sections called zones • Zones are distributed over a collection of DNS servers • Hierarchy of DNS servers: • Root servers (identity hardwired into other servers) •Top-level domain (TLD) servers • Authoritative DNS servers • Performing the translations: • Local DNS servers located near clients • Resolver software running on clients. DNS is hierarchical: Hierarchy of namespace matches hierarchy of servers • Set of nameservers answers queries for names within zone • Nameservers store names and links to other servers in tree. **Top-level domain (TLD) servers** • Responsible for com, org, net, edu, etc, and all top-level country domains: uk, fr, ca, jp • Network Solutions maintains servers for com TLD • Educause non-profit for edu TLD • Authoritative DNS servers • An organization's DNS servers, providing authoritative information for that organization • May be maintained by organization itself, or ISP. **LOCAL NAME SERVERS** • Do not strictly belong to hierarchy • Each ISP (or company, or university) has one • Also called default or caching name server • When host makes DNS query, query is sent to its local DNS server • Acts as proxy, forwards query into hierarchy • Does work for the client **DNS is a distributed database storing resource records. DNS CACHING** • Performing all these queries takes time • And all this before actual communication takes place • Caching can greatly reduce overhead • The top-level servers very rarely change • Popular sites visited often • Local DNS server often has the information cached •All DNS servers cache responses to queries • Responses include a time-to-live (TTL) field • Server deletes cached entry after TTL expires. **LDAP: LDAP OVERVIEW** • LDAP is a hierarchical, extensible, searchable data store • Used in many networked services • Login (ieng6) • Active Directory • Many UCSD services • UCSD Wireless • Mobile networks • Every time you make a call on AT&T, for example • Permissions in Linux.

Time: WHAT MAKES TIME SYNCHRONIZATION HARD? 1. Quartz oscillator sensitive to temperature, age, vibration, radiation • Accuracy one part per million (one second of clock drift over 12 days) 2. The internet is: • Asynchronous: arbitrary message delays • Best-effort: messages don't always arrive. **Time synchronization • Cristian's algorithm • Berkeley algorithm • NTP**. **CLOCK SYNCHRONIZATION: TAKE-AWAY POINTS** • Clocks on different systems will always behave differently • Disagreement between machines can result in undesirable behavior • NTP, Berkeley clock synchronization • Rely on timestamps to estimate network delays • 100s μs−ms accuracy • Clocks never exactly synchronized • Often inadequate for distributed systems • Often need to reason about the order of events • Might need precision on the order of ns. **Lamport clock: TOTALLY-ORDERED MULTICAST (CORRECT VERSION)** 1. On receiving an event from client, broadcast to others (including yourself) 2. On receiving or processing an event: a) Add it to your local queue b) Broadcast an acknowledgement message to every process (including yourself) only from head of queue 3. When you receive an acknowledgement: • Mark corresponding event acknowledged in your queue 4. Remove and process events everyone has ack'ed from head of queue. **Cons**: Does **totally-ordered multicast** solve the problem of multisite replication in general? • Not by a long shot! **TAKE-AWAY POINTS:** LAMPORT CLOCKS • Can totally-order events in a distributed system: that's useful! • But: while by construction, a → b implies C(a) < C(b), • The converse is not necessarily true: • C(a) < C(b) does not imply a → b (possibly, a || b) **Can't use Lamport clock timestamps to infer causal relationships between events.** **Vector Clock:** Vector clock timestamps tell us about causal event relationships. **Two events a, z Lamport clocks: C(a) < C(z) Conclusion: None Vector clocks: V(a) < V(z) Conclusion: a → … → z. VC APPLICATION: CAUSALLY-ORDERED BULLETIN BOARD SYSTEM** • Distributed bulletin board application • Each

post → multicast of the post to all other users • Want: No user to see a reply before the corresponding original message post • Deliver message only after all messages that causally precede it have been delivered • Otherwise, the user would see a reply to a message they could not find.
CACHING, CONTENT-DISTRIBUTION NETWORKS, AND OVERLAY NETWORKS: WHY WEB CACHING? • Motivation for placing content closer to client: • User gets better response time • Content providers get happier users • Network gets reduced load • Why does caching work? Exploits locality of reference • How well does caching work? • Very well, up to a limit • Large overlap in content • But many unique requests. **CACHING WITH REVERSE PROXIES** • Cache data close to origin server → decrease server load • Typically done by content providers • Client thinks it is talking to the origin server (the server with content) • Does not work for dynamic content. **CACHING WITH FORWARD PROXIES** • Cache close to clients → less network traffic, less latency • Typically done by ISPs or corporate LANs • Client configured to send HTTP requests to forward proxy • Reduces traffic on ISP-1's access link, origin server, and backbone ISP. **CONTENT DISTRIBUTION NETWORKS** • Proactive content replication • CDN replicates the content • On many servers spread throughout the Internet • Updating the replicas REPLICA SELECTION: GOALS • Live server • Lowest load• Closest • Best performance. **MAPPING SYSTEM** • Equivalence classes of IP addresses • IP addresses experiencing similar performance • Quantify how well they connect to each other • Collect and combine measurements • Ping, traceroute, BGP routes, server logs • Network latency, loss, throughput, and connectivity. **ROUTING CLIENT REQUESTS WITH THE MAP** • Map each IP class to a preferred server cluster • Based on performance, cluster health, etc. • Updated roughly every minute • Short, 60-sec DNS TTLs in Akamai regional DNS accomplish this • Map client request to a server in the cluster • Load balancer selects a specific server**. ADAPTING TO FAILURES** • Failing hard drive on a server • Suspends after finishing "in progress" requests • Failed server • Another server takes over for the IP address • Low-level map updated quickly (load balancer) • Failed cluster, or network path • High-level map updated quickly (ping/traceroute). **TAKE-AWAY POINTS: CDNS** • Content distribution is hard • Many, diverse, changing objects • Clients distributed all over the world • Moving content to the client is key • Reduces latency, improves throughput, reliability • Content distribution solutions evolved: • Load balancing, reactive caching, to • Proactive content distribution networks. **HOSTING: MULTIPLE MACHINES PER SITE** • Problem: Overloaded popular web site • Replicate the site across multiple machines • Helps to handle the load • Want to direct client to a particular replica. Why? • Balance load across server replicas • Solution #1: Manual selection by clients Solution #2: Single IP address, multiple machines Solution #3: Multiple IP addresses, multiple machines. **SUMMARY** Load-balancer approach • No geographical diversity ✗ • TCP connection issue ✗ • Does not reduce network traffic✗ • DNS redirection • No TCP connection issues ✓ • Simple round-robin server selection • May be less responsive ✗ • Does not reduce network traffic✗. **FACTORS OF VARIABLE RESPONSE TIME**: • Shared Resources (Local) • Global Resource Sharing • Daemons• Maintenance Activities • Queueing• Power Limits• Garbage Collection• Energy Management**. CONCLUSION** • Variability of latency exists in large-scale services. • Live with variable request latency • Infeasible to eliminate variable request latency in large-scale services. • The importance of these techniques will only increase. (scale of services getting larger) • Techniques for living with variable latency turn out to increase system utilization. **AVAILABILITY METRICS** • Mean time between failures (MTBF) • Mean time to repair (MTTR) • Availability = (MTBF – MTTR)/MTBF. **HARVEST AND YIELD** • yield = queries completed/queries offered • harvest = data available/complete data. **DQ PRINCIPLE** • Data per query * queries per second → constant • At high levels of utilization, can increase queries per second by reducing the amount of input for each response • Adding nodes or software optimizations changes the constant **GRACEFUL DEGRADATION** • Cost-based admission control • Search engine denies expensive query (in terms of D) • Rejecting one expensive query may allow multiple cheaper ones to complete • Priority-based admission control • Stock trade requests given different priority relative to, e.g., stock quotes • Reduced data freshness • Reduce required data movement under load by allowing certain data to become out of date (again stock quotes or perhaps book inventory).
PRIMARY-BACKUP: GOALS • Mechanism: **Replicate** and separate servers • Goal #1: Provide a highly reliable service • Despite some server and network failures • Continue operation after failure • Goal #2: Servers should behave just like a single, more reliable server. **STATE MACHINE REPLICATION** • Any server is essentially a state machine• Need an op to be executed on all replicas, or none at all • Key assumption: Operations are deterministic. **PRIMARY-BACKUP (P-B) APPROACH** • Nominate one server the primary, call the other the backup • Clients send all operations (get, put) to current primary • The primary orders clients' operations • Should be only one primary at a time. 1. Primary logs the operation locally 2. Primary sends operation to backup and waits for ack • Backup performs or just adds it to its log 3. Primary performs op and acks to the client • After backup has applied the operation and ack'ed **CHALLENGES** • Network and server failures • Network **partitions** • Within each network partition, near-perfect communication between servers • Between network partitions, no communication between servers. **MONITORING SERVER LIVENESS** • Each replica periodically pings the view server • View server declares replica dead if it missed N pings in a row • Considers the replica alive after a single ping • Can a replica be alive but declared "dead" by view server? • Yes, in the case of network failure or partition. **SUMMARY OF RULES** 1. View i's primary must have been primary/backup in view i−1 2. A non-backup must reject forwarded requests • Backup accepts forwarded requests only if they are in its idea of the current view 3. A non-primary must reject direct client requests 4. Every operation must be before or after state transfer. **PRIMARY-BACKUP: SUMMARY** • First step in our goal of making stateful replicas fault-tolerant • Allows replicas to provide continuous service despite persistent net and machine failure • Finds repeated application in practical systems. **Safety and liveness OFTEN A TRADEOFF** • "Good" and "bad" are application-specific • Safety is very important in banking transactions • May take some time to confirm a transaction • Liveness is very important in social networking sites• See updates right away. **Two-phase commit SINGLE-SERVER: ACID** • Atomicity: all parts of the transaction execute or none (A's decreases and B's balance increases) • Consistency: the transaction only commits if it preserves invariants (A's balance never goes below 0) • Isolation: the transaction executes as if it executed by itself (even if C is accessing A's account, that will not interfere with this transaction) • Durability: the transaction's effects are not lost after it executes (updates to the balances will remain forever). **TWO-PHASE COMMIT (2PC) • Goal**: General purpose, distributed agreement on some action, with failures • Different entities play different roles in the action. **REASONING ABOUT ATOMIC COMMIT** • Why is this correct? • Neither can commit unless both agreed to commit • What about performance? 1. Timeout: I'm up, but didn't receive a message I expected • Maybe other node crashed, maybe network broken 2. Reboot: Node crashed, is rebooting, must clean up. **REASONING ABOUT THE SERVER TERMINATION PROTOCOL** • What are the liveness and safety properties? • Safety: if servers don't crash, all processes will reach the same decision • Liveness: if failures are eventually repaired, then every participant will eventually reach a decision • Can resolve some timeout situations with guaranteed correctness • Sometimes however A and B must block • Due to failure of the TC or network to the TC • But what will happen if TC, A, or B **crash and reboot? HOW TO HANDLE CRASH AND REBOOT?** • Can't back out of commit if already decided • TC crashes just after sending "commit!" • A or B crash just after sending "yes" • If all nodes knew their state before crash, we could use the termination protocol… • Use write-ahead log to record "commit!" and "yes" to disk. **RECOVERY PROTOCOL WITH NON-VOLATILE STATE** • If everyone rebooted and is reachable, TC can just check for commit record on disk and resend action • TC: If no commit record on disk, abort • You didn't send any "commit!" messages • A, B: If no yes record on disk, abort • You didn't vote "yes" so TC couldn't have committed • A, B: If yes record on disk, execute termination protocol • This might block **TWO-PHASE COMMIT** • This recovery protocol with non-volatile logging is called Two-Phase Commit (2PC) • Safety: All hosts that decide reach the same decision • No commit unless everyone says "yes" • Liveness: If no failures and all say "yes" then commit • But if failures then 2PC might block • TC must be up to decide • Doesn't tolerate faults well: must wait for repair.
Raft Protocol :follower: Respond to RPCs from candidates and leaders. • Convert to candidate if election timeout elapses without either: • Receiving valid AppendEntries RPC, or • Granting vote to candidate **Candidate**: Increment currentTerm, vote for self • Reset election timeout • Send RequestVote RPCs to all other servers, wait for either: • Votes received from majority of servers: become leader • AppendEntries RPC received from new leader: step down • Election timeout elapses without election resolution: increment term, start new election • Discover higher term: step down **Leader**: Initialize nextIndex for each to last log index + 1 • Send initial empty AppendEntries RPCs (heartbeat) to each follower; repeat during idle periods to prevent election timeouts • Accept commands from clients, append new entries to local log • Whenever last log index ≥ nextIndex for a follower, send AppendEntries RPC with log entries starting at nextIndex, update nextIndex if successful • If AppendEntries fails because of log inconsistency, decrement nextIndex and retry • Mark log entries committed if stored on a majority of servers and at least one entry from current term is stored on a majority of servers • Step down if currentTerm changes
Security: HTTP Overview: 1.HTTP-secure 2.HTTP with TLS(transport layer security) 3.Started as secure sockets layer in 1996, now TLS 1.3. 4.Provides confidentiality, integrity, authenticity(CIA). **Keys:** 1.Bit String generated by Cryptographically Secure Pseudo Random Number Generators. 2. Created using entropy in your computer. 3.keys need to be certain size to secure. **Cipher Functions(like ROTB) Hash Functions** Asymmetric key Cryptography aka Public Key Cryptography. (private key+public key). **Goals of the TLS handshake** 1.Create key for message encrypting 2. Open and verify messaging channel. 3.Authenticate server (and sometimes client). **Ciphersuite**s: asymmetric- asymmetric-symmetric-hash **Why not asymmetric all the time? Low effiency, time consuming too much**. **Potential Problems with HTTPS**: Invalid Certificate, Expired, Wrong domain, self-signed. **Side effects of HTTPS**: 1.people don't actually know what the lock means 2. report from Phishlabs that 47% of phishing sites use a certificate 3. focus onus onto the user
Final Prepare: **Replication vs partitioning**: Replication is easy, partitioning is hard. The challenge with replication is consistency. The challenge for partitioning is re-partitioning Re-partitioning usually needs downtime. **Average latency** is the average amount of it takes to complete a single transaction. While less latency is always better, average latency is fine because it's predictable. **Tail latency** is not predictable. **round robin load balancing** has one major advantage, it is extremely simple to implement , but it needs to be understood that it does have a number of potentially important drawbacks. These come from the very DNS hierarchy that it uses to perform its load balancing This problem can result in unpredictability and even corrupt the DNS tables of all but the most advanced network load balancers. This means that servers that have failed continue to receive requests for providing content to users despite the fact that they are down and therefore no longer available**. Link state** • Tell everyone what you know about your neighbors • **Distance vector** • Tell your neighbors what you know about everyone. Frame:Header+Payload+Trailer, Header usually contains addressing information. **FIXED-LENGTH FRAMES** • Easy to manage for receiver • Well understood buffering requirements • Introduces inefficiencies for variable length payloads • May waste space (padding) for small payloads • Larger payloads need to be fragmented across many frames • Very common inside switches • Requires explicit design tradeoff **LENGTH-BASED FRAMING** • To avoid overhead, we'd like variable length frames • Each frame declares how long it is • E.g. DECNet DDCMP • What's the issue with explicit length field? • Must correctly read the length field (bad if corrupted) • Need to decode while receiving **SENTINAL/DELIMITER BASED FRAMING** • Allow for variable length frames • Idea: mark start/end of frame with special "marker" • Byte pattern, bit pattern, signal pattern • But… must make sure marker doesn't appear in data • Two solutions • Special non-data physical-layer symbol • Impact on efficiency (can't use symbol for data) of code • Stuffing • Dynamically remove marker bit patterns from data stream • Receiver "unstuffs" data stream to reconstruct original data. In computing, an **idempotent** operation is one that has no additional effect if it is called more than once with the same input parameters. For example, removing an item from a set can be considered an idempotent operation on the set.(The operation to write data to a file can be defined (i) as in Unix where each write is applied at the read-write pointer, in which case the operation is not idempotent; or (ii) as in several file servers where the write operation is applied to a specified sequence of locations, in which case, the operation is idempotent because it can be repeated any number of times with the same effect. The operation to append data to a file is not idempotent, because the file is extended each time this operation is performed. The question of the relationship between idempotence and server state requires some careful clarification. It is a necessary condition of idempotence that the effect of an operation is independent of previous operations. Effects can be conveyed from one operation to the next by means of a server state such as a read-write pointer or a bank balance. Therefore it is a necessary condition of idempotence that the effects of an operation should not depend on server state. Note however, that the idempotent file write operation does change the state of a file.) **Briefly describe how Akamai enables distribution of live and streaming content?** 1. Through CDN: Afterwards, Akamai had application delivery networks that can accelerate entire web or IP-based applications, media delivery networks that provide HD- quality delivery of live and on-demand media, and EdgeComputing networks that deploy and execute entire Java J2EE applications in a distributed fashion 2. The real-time mapping: The real-time mapping part of the system creates the actual maps used by the Akamai platform to direct end users (identified by the IP addresses of end users and their name servers) to the best Akamai edge servers to respond to their requests. This part of the system also selects intermediates for tiered distribution and the overlay network. **What information does the Akamai CDN use to map requests to resources? How (briefly) does it implement that mapping?** Historic and real-time data.The real-time mapping part of the system creates the actual maps used by the Akamai platform to direct end users (identified by the IP addresses of end users and their name servers) to the best Akamai edge servers to respond to their requests. This part of the system also selects intermediates for tiered distribution and the overlay network. This assignment happens in two main steps: Map to cluster and Map to server. When hardware and network faults are identified (such as a failing hard drive on a server), the failed edge servers are ―suspended and will finish up processing in-progress requests but will not be sent any additional end users until the fault has been resolved. The mapping system itself is a fault-tolerant distributed platform that is able to survive failures of multiple data centers without interruption. The scoring and map-to-cluster portions of the system run in multiple independent sites and leader-elect based on the current health status of each site. The map-to-server portions of the system run on multiple machines within each target cluster. All portions of the system, including monitoring agents and DNS servers, communicate via a multi-path overlay transport that is able to tolerate network faults. **We can not set back the clock because some application could timestamp events under the assumption that clocks always advance**. In our case we have the errant clock, let's call it E, and the hardware clock H (which is supposed to advance at a perfect rate). Let's now construct a software clock such that after 13 seconds we can replace the errant clock with the software clock in good conditions. Let's denote the software clock with S, then: S = c(E − Tskew) + Tskew, where Tskew = 17 : 53 : 28 and c is to be found. We know that S = Tskew + (13 − 7) = Tskew + 6 when E = Tskew + 13, so: Tskew + 6 = c(Tskew + 13 − Tskew) + Tskew, and c = 6/13. We obtain the formula: S = (6/13)(E − Tskew) + Tskew (when Tskew ≤ E ≤ Tskew + 13).