DNS(domain name system) was created as a solution to make IP addresses human-readable for users.**Hierarchy** ①Names are hierarchical: Domains get more specific from right to left.②Authority is hierarchical: Each level has a responsible party (.edu, berkeley.edu, etc).The DNS root is controlled by ICANNTop Level Domains (TLDs) are controlled by over 1500 authorities, such as Educause for edu domains and Verisign for .net/.com domains.A zone corresponds to an administrative authority responsible for a contiguous portion of the authority. An example of a zone is *.berkeley.edu which controls all domains ending in berkeley.edu.③Infrastructure is hierarchical: the DNS system is composed of many name servers which each are responsible for one part of the hierarchy.**Name Lookup** ①A client looks up a domain by querying their resolving name server (usually run by ISP) ② The resolving name server runs a recursive query (actually iterative) by repeatedly doing the following:Get a request from the current server (starting at the root);If the server knows the answer, return the answer.;Otherwise, move onto the next server and return the result of that query**.There are several main classes of name servers:**①Root server knows all the TLD servers②TLD server knows about a particular TLD (such as .edu)③Authoritative servers know information about their zone (such as *.berkeley.edu) and map domain names to IPs.**The DNS Protocol** C/python socket API: result = gethostbyname("hostname.com"): deprecated but still common, limited to IPv4 error = getaddrinfo("hostname.com", NULL, NULL, &result): more modern, not limited to IPv4 Standard DNS server: BIND (berkeley internet name domain server) basically a daemon/server process listens on port 53 (UDP) Messages may be either a query or response (QR bit in header 0 or 1 respectively). Data is stored in resource records (RRs) that are a tuple of (type, name, value, ttl, class). type is A, NS, etc. name is the domain name value is the IP address ttl is how long the record is valid for class is used for other network types (not really used in practice)**Step by step** Client queries resolving name server;Resolving name server queries root server, requesting an A record;Response: list of NS records corresponding to TLD/authoritative servers, as well as an additional A record (IP for name server we should ask next);Repeat until the desired authoritative server is contacted, and an A record is returned**Registering a domain** Companies can purchase/request IP blocks from ISP;Register domain with a registrar;Run 2 authoritative name servers for the domain (often handled by registrar/external service);Registrar will insert pairs of NS and A records into the TLD name servers**Reverse lookups** Using the PTR record, we can convert an IP address to a hostname.;name = dot-quad IP address listed backwards (138.110.1.200 -> 200.1.110.138);name is followed by .in-addr.arpa**Record Types** A: "address record" - maps hostname to IP address AAAA: Same as A, but for ipv6 NS: "nameserver" - maps domain to DNS server CNAME: "canonical name" - way for aliasing from one hostname to another hostname DNAME: maps an entire subtree to another subtree MX: "mail exchanger": redirects to another mail server TXT: human-readable information, often used to prove ownership of domain SRV: used for arbitrary services (servicename.transportprotocol.hostname)**Availability, Scalability, Performance** DNS should be:Highly available: accessible at all times (otherwise the internet breaks down);Highly scalable: most devices on the internet will use DNS;Highly performant: lookups should be fast and take little bandwidth;How do we do this? Just add more servers.Domains have at least two name servers each.Have multiple servers per domain such that if one domain goes down, others are still available.Have multiple root servers (currently, there are 13), and make each of these root servers a network of physical servers around the world. (For example, the E root has over 300 servers with the same IP address using anycast)Caching: Increases performance by reducing the number of requests/iterative queries being made. Caches can be introduced at any layer, including the host.**Example**Let's say our local host wants to access the domain ischool.berkeley.edu.Our host will send a recursive query to the resolving name server (cdns01.comcast.net), asking for the A record for ischool.b.e.The resolving nameserver will check the cache, and if present, return the result.If cache entry not present, the resolving name server queries the root server requesting the A record for ischool.b.e.The root server sends back the DNS tuples (NS, edu, k.edu-servers.net), and (A, k.edu-servers.net, aaa.aaa.aaa.aaa).The resolving nameserver queries k.edu-servers.net requesting the A record for ischool.b.e.The TLD server sends back NS and A records for berkeley.edu.The resolving nameserver queries adns1.berkeley.edu.Berkeley's DNS server sends back the desired A record, along with a TTL. **Web** In 1989, Tim Berners-Lee set out to solve a problem: there was a lot of information being stored digitally, and no way to find or access much of it. He created "Information Management: A Proposal" in which the concept of the "web" was first established.This proposal had several key parts:not based on a hierarchy.allows remote access across networks.heterogeneity: different systems can access the same data.non-centralization: ability for existing systems to be linked together without a central control.access to existing data: ability to get data from existing databases to reduce overhead of adapting new system**Why was this so successful?**Very flexible; didn't force any changes on existing data, systems, or networksMany systems were built for networks in the first place.Integrated interface for scattered information.Practical solution to a specific problem.Early form of open-source software (free for anyone to use).No over-specification: websites can be structured in many ways.No central authority or single underlying system: anyone can add their own systems easily.Ability to quickly navigate between different sources**Basics**A way to represent content with links: HTML A client program to access content: web browsers A way to reference content: URLs A way to host content: serversA protocol to transfer content between servers and clients: HTTP URL Syntax scheme://host:port/path/resource?query#fragment Scheme: protocol (https, ftp, smtp…) Host: DNS hostname or IP address Port: 80 for http, 443 for https Path: traditional filesystem hierarchy Resource: desired resource Query: search terms Fragment: subpart of a resource**HTTP**Note: The following information refers to the HTTP 1.0 standard. HTTP 2 is also commonly supported(about 44% adoption), and HTTP 3 is upcoming (5%, mostly Google/Facebook), but they are significant departures in terms of implementation.**Main idea**:Client-server architecture Client connects to server via TCP on port 80 Stateless protocol**HTTP Request** Plaintext, separated with CRLF (CR = Carriage Return, ASCII 13, LF = Line Feed, ASCII 10) Request Line: Method Resource Protocol Method: GET, HEAD, POST… Resource: what needs to be fetched Protocol version: HTTP/1.1 or HTTP/1.0 Request Headers: provide additional information Body: separated with a blank line; used for submitting data **HTTP Status** Status Line: Protocol Status Reason Protocol: HTTP/1.1 or HTTP/1.0 Status: status code (200, etc) Reason: human-readable message **HTTP Methods** GET: request to download (body on response only) POST: send data from client to server (body often present in both request and response) HEAD: same as GET except no body is needed in the response (used to check for existence)**Status Codes** 1xx: informational (not defined)2xx: successful200: OK3xx: redirection301: moved permanently304: not modified4xx: client error400: bad request401: unauthorized404: not found5xx: server error500: internal server error **Caching** Web caching takes advantage of temporal locality: if something is accessed, it'll probably be accessed soon. This is true because the most popular content is accessed far more frequently than non-popular content. Caching is implemented via two headers: Cache-Control: max-age=(seconds) - 1.1 Expires: (absolute time of expiry) - 1.0We can also specify the following to force skip caches:Cache-Control: no-cache - 1.1Pragma: no-cache - 1.0Additional settings:If-Modified-Since: (date)If a resource has changed since date, respond with latest version. Otherwise, respond with 304 (not modified) Proxy servers make requests on behalf of clients. This creates an extra layer of caching that can server multiple clients more quickly.Reverse proxies are caches close to the servers.Forward proxies are caches close to the clients. (typically done by ISPs)**CDNs** Content Delivery Networks provide caching and replication as a service. CDNs are large-scale distributed storage infrastructure that create new domain names for customers. The content provider then rewrites content to reference the new domains instead of the original ones.typically aliased using CNAMEs to make domain names still human-readable Pull:CDN acts like a cache content provider gives CDN an origin URLwhen a client requests it from CDN:if cached, serveif not cached, pull from origineasier to implement (less work for content provider)Push:Content provider uploads content to CDN, who serves it like a normal serverprovides more control over content**HTTP Performance** The primary bottleneck is RTT, not transmission delay. Using standard TCP, downloading many small objects takes 2 RTTs per object, which adds up to a lot of time.**Some optimizations can be made:**Concurrent requests: make many requests in parallel need to share bandwidth between all concurrent requestsPersistent connections: maintain TCP connection across multiple requestscan be combined with concurrent requests default for HTTP 1.1 Pipelined connections: send multiple requests all at once can combine small requests into one large request not used in practice, due to bugs and head-of-line blocking (remaining connections all need to wait for a slow connection in the middle).there are **three types of HTTP cache**s:**Private caches** are associated with a specific end client connecting to the server (e.g. the cache in your own browser). Now, if the same user requests the same resource a second time, they can fetch the resource from their local cache. However, private caches are not shared between users.**Proxy caches** are in the network (not on the end host), and are controlled by the network operator, not the application provider. These caches can be shared between lots of users, so a user requesting a resource for the first time might get the data from the proxy cache instead of the origin server.**Managed caches** are in the network, and are controlled by the application provider. Note that managed cache servers are deployed separately, and are not the original server that generated the content. Because these caches are controlled by the application provider, this gives the application more control.**Ethernet** Shared Media In a radio network, nodes use a shared medium (the electromagnetic spectrum). As such, transmissions from different nodes might collide with one another, so we need a multiple access protocol to allocate the medium between users. Some common approaches for doing this include:**Frequency Division Multiplexing**: divide medium by frequency. This can be wasteful since frequencies are likely to be idle often.**Time Division Multiplexing**: divide medium by time. Each sender gets a fixed time slot to send data. This has similar drawbacks to FDM.**Polling protocols**: a turn-taking scheme where a coordinator gets to decide who gets to send data when (how Bluetooth works)**Token-passing**: a turn-taking scheme where a virtual token is passed around, and only the holder can transmit Random access: see **ALOHAnet** Additive Links Online Hawaii Area: a first attempt at wireless connections across the Hawaiian islands (1968, Norman Abramson) In ALOHAnet, a hub node transmitted its own frequency, and all remote nodes transmitted on the same frequency using a random access scheme. Pure ALOHA random access #If a remote has a packet, just send it.When the hub gets a packet, it sends an ack.If two remote sites transmitted at the same time, a collision will occur and the hub will not send an ack If the remote doesn't get the expected ack, wait a random amount of time and resend.**Etherne**In 1972, Bob Metcalfe was trying to connect hundreds of Xerox computers in the same building. It needed to be fast, maximially distributed, and cheap.The main idea was to connect all of the machines onto the same cable, and use it as a shared medium.**Carrier Sense Multiple Access (CSMA)**CSMA is an improvement over ALOHA: instead of nodes sending data first, CSMA nodes listen to the network first and start transmitting when it's quieter.By itself, this doesn't completely avoid collisions due to propagation delay.**CSMA/CD**Main idea: listen while you talk. If a node detects another packet being sent at the same time, stop sending since the packet has already collided. This is collision detection (CD).In addition, use a randomized binary exponential backoff: if retransmit after collision also collides, wait twice as long; continue doubling for every collision.**Addresses and Service Types** #On the ethernet shared medium, everyone will receive transmitted data. As such, ethernet has flat addresses: no routing or aggregation is required.Addresses are 48 bits (6 bytes) shown as six 2-digit hex numbers with colons. The general structure is:2 bits of flags22 bits identifying manufacturer (company/org)24 bits identifying device Addresses are typically permanently stored in network interface hardware, and are mostly unique (there are more devices than there are addresses).The broadcast address is all ones (FF:FF:FF:FF:FF:FF). Data sent to this address are received by everyone. This allows trivial implementation of broadcast.Multicast (sending to all members within a group) is also trivially implemented by setting the first bit to 1.However, classic ethernet does not support anycast (single address being shared by multiple devices).**Switched Ethernet** #In modern ethernet implementations, shared media is rarely used. Instead, switches exist between nodes that remove the possibility of collision.The main idea of switched ethernet is to flood all packets, such that everyone gets it just like in classic ethernet.**Summary: MAC (L2) vs IP (L3)** #**MAC addresses**:hard coded by device manufacturers not aggregation friendly ("flat" addresses with no hierarchy)topology independent: same even when host moves main purpose: packet transfer within the same L2 network require no network configuration **IP addresses**:dynamically configured and assigned by network operators + DHCP have hierarchical structure topology dependent: depends on where host is attached main purpose: packet transfer to destination subnet **The IP/MAC split solves the bootstrap problem**, in which the fixed behavior of MAC makes a convenient first assignment that IP can build off of for assigning the first hop.**ARP** #Address Resolution Protocol: converts IP addresses to corresponding ethernet addresses.ARP runs directly on top of L2 (between L2 and L3, which is IP). In general, the host broadcasts a query asking who has a particular IP address. The desired host then responds via unicast with its ethernet address.Hosts typically cache results in an ARP table (Neighbor table), which is refreshed occasionally**.ARP Example**If H1 wants to send a packet to the IP address 10.0.0.2:Check the prefix to see if it's on the same subnet. (it is)H1 broadcasts an ARP request to all hosts on the subnet.H2 answers H1's request with its

MAC address (using unicast)H1 receives the answer, sends the packet, and caches the entry in its ARP table.If H1 wants to send a packet to 10.1.0.3:Check the prefix and see that the subnet is different.Since the subnet is different, broadcast an ARP request for the router (10.0.0.254) instead.R1 answers H1's request with its MAC address.H1 receives the answer, and sends the packet with destination IP 10.1.0.3, but destination MAC of R1.**DHCP**Although ethernet addresses are hard-coded into the hardware, IP addresses are adaptable depending on the context. Typically, the routers know what IP addresses to assign to hosts, but how do hosts know this?One method is to manually assign static addresses to each host. However, portable devices like phones or laptops may move around multiple times a day, and needing to reassign them every time we move is very annoying!The solution is DHCP (Dynamic Host Configuration Protocol), which provides a way for hosts to query the network for local configuration information (IP address, netmask, default gateway, local DNS resolver).DHCP servers are added to the network, either as standalone or as a part of a router, which listen to UDP port 67.The DHCP server leases IP addresses to hosts. If the lease is not renewed, the IP address will be returned to the pool and can be assigned to another host.**TLS: Secure Bytestreams**Secure BytestreamsTCP by itself is insecure against network attackers. Someone on the network (e.g. a malicious router, an attacker sniffing packets on a wire) could read or even modify your TCP packets while they're in transit.Also, with TCP, you might connect to an attacker instead of the real server. Suppose you want to connect to a bank website, and you do a DNS lookup for www.bank.com. The attacker (e.g. someone who hacked into the resolver or a router) changes the DNS response so that it maps www.bank.com to the attacker's IP address, 6.6.6.6. Now, when you form a TCP connection to the bank website, you're talking to the attacker. You might end up sending your bank password to the attacker!To address these security issues, we add a new protocol, Transport Layer Security (TLS), on top of TCP.TLS can be thought of as a Layer 4.5 protocol, sitting in between TCP and application protocols like HTTP. (We use a weird number like 4.5 because the obsolete Layers 5 and 6 have nothing to do with security.) TLS relies on the bytestream abstraction of TCP, so it doesn't think about individual packets or packet loss/reordering. TLS provides the exact same bytestream abstraction to applications as TCP does, but the bytestream is now secure against network attackers. This is why HTTP and HTTPS are semantically identical protocols. The only difference is that HTTPS runs over the secure bytestream of TLS-over-TCP, while HTTP runs over raw TCP with no TLS.To distinguish between HTTPS and HTTP, we use Port 80 for HTTP connections, and Port 443 for HTTPS connections. Servers can force users to use HTTPS by replying to all Port 80 requests with a redirect to use Port 443 instead.**TLS Handshake**At a high level, TLS uses cryptography to encrypt messages sent over the bytestream. TLS also uses other cryptographic protocols (message authentication codes) to prevent attackers from changing messages as they're sent over the network.In order to encrypt traffic, TLS must start with an additional handshake to exchange keys and verify the identity of the server (e.g. real bank, not someone impersonating the bank).Because TLS is built on top of TCP, the TCP three-way handshake first proceeds as normal. This creates an (insecure) bytestream, allowing all future messages, including the TLS handshake, to proceed without thinking about individual packets.
End to End OperationSuppose we have the following scenario:①Host H1 boots up②Fetches small file from H5③Goes idle④Fetches two small files from H2 Here's what will happen:DHCP to get configurationUDP Discover => broadcastUDP Offer from H4 <- H1UDP Request => broadcastUDP ack <- H1ARP for DNS serverARP request for H3 => broadcastH3 ARP response <- unicast to H1 Resolve H5 UDP DNS request for H5.com => H3 UDP DNS response <- H1 ARP for R1 ARP request for R1 => broadcast ARP response from R1 <- H1 TCP connection to H5 TCP SYN => H5 TCP SYNACK <- H1 TCP ACK => H5 HTTP request to H5 TCP HTTP GET => H5 TCP ACK <- H1 HTTP response <- H1 ACK => H5 (after download completes and connection becomes idle for a while) FIN => H5 TCP disconnect from H5 ACK <- H1 FIN <- H1 ACK => H5 The rest of the steps are extremely similar to the above. Resolve H2 ARP for H2 TCP to H2 HTTP to H2 HTTP to H2 (2) TCP disconnect from H\2 F, telephone networks use circuit switching and the modern internet uses packet switching. F, transport layer is layer 4. F, the outermost header is the application headers (HTTP etc). T, the only increase/decrease mode that is fair is AIMD. F, UDP does provide checksums. F, HTTP runs over TCP, not UDP. T, DNS runs over UDP since it needs to be responsive. T, DHCP runs over UDP since it's broadcast. F, ARP runs on L2 so no UDP is used. T, ARP requests are broadcast. F, ARP responses are unicast. F, ACK packets can be combined with HTTP requests (which are payloads). F, MX is a mail record that redirects a domain to another domain's mail server. F, removing timers means TCP wouldn't be reliable in the case of timeouts. T, SDN allows additional control of networking within cloud providers; traditional networking relies on vendor code within hardware devices. T, OpenFlow gives access to the forwarding plane. F, cellular networks store user state on the data plane to route packets to the correct carrier. F, Google's global WAN consists of many different interconnected regions and zones.Mux and demux from/to application processes is **provided by both UDP and TCP.ECN** enables a sender to respond to congestion before router queues are full.**How does HTTP compensate for the fact that it is a stateless protocol?**Cookies**.In SDN, what is the interface between the NOS and switches?**OpenFlowWhen you take your laptop to campus and want to access the www.chocolics.com website, which of the answers above best describe how the laptop determines the following (assuming no information resides in caches, no proxies are being used, etc.)? (List one option for each of the following.)1. Laptop IP address Solution: DHCP 2. www.chocolics.com's IP address Solution: DNS 3. Laptop's first-hop router IP address Solution: DHCP 4. Laptop MAC address Solution: Hardware 5. The MAC address of www.chocolics.com's server, assuming the server is not on your laptop's subnet Solution: No Need 6. Laptop's first-hop router MAC address Solution: ARP