P4

a) http://gaia.cs.umass.edu/cs453/index.html - from first line and Host field.
b) 1.1 - next to HTTP
c) persistent - from Connection:keep-alive
d) IP address of the client is not in the request.
e) A Netscape browser initiates this request, this is found in the User-Agent field. Browser type is needed so the server knows what kind of content the client is able to render.

P5

a) Yes the server was able to successfully find the document based on the 200 response code. The reply was provided at 12:39:45 GMT.

b) December 10, 2005 at 18:27:46 GMT

c) 3874

d) <!doc is the first 5 characters of the returned document. Each character is a byte. <= 00111100, ! = 00100001, $d = 01100100$, $o = 01101111$, $c = 01100011$

P6

a) The mechanism for signaling between the client and the server that a persistent connection is being closed is sending a Connection header including the connection-token "close". Both the client and the server may signal the close of a connection.

b) HTTP does not provide any encryption services.

c) The specification says a client should not maintain more than 2 simultaneous connections with a given server. A proxy can maintain 2*N connections with another server where N is the number of simultaneously active users.

d) Yes it is possible that one side closes the connection while the other side is transmitting data. The specification states that clients, servers and proxies should be able to recover from asynchronous close events. If the request was idempotent the client should automatically retry. If it is non idempotent the application should verify the status of the request that was asynchronously closed.

P7

The sum of all the round trip times $\sum_{i=0}^{n} RTT_i$. There are also additional RTTs for establishing a TCP connection with each DNS server, and an RTT for establishing a TCP connection with the web server holding the HTML object.

P8

a) With non-persistent HTTP and no parallel connections we still have the n round trips for DNS plus an additional 8 round trips for each object. So there are $n + 9$ round trips for the application level requests. There are an additional n round trips to establish TCP connections for the DNS requests, and an additional 9 round trips to establish TCP connections for the HTTP requests. So the total time is $2n + 18$ round trips.

b) In this case the DNS lookups still need to happen serially so that will be 2n round trips accounting for the round trips required to make TCP connections. For the HTTP requests we first request the HTML file requiring 2 round trips (TCP connection and HTTP request). Then open up 5 TCP connections in parallel which can be done in the time it takes to make 1 round trip. Then the next 5 objects can be received in the time it takes to make 1 round trip. Next the remaining 3 objects can be received in the time it takes to make one round trip plus another round trip prior to the HTTP requests open up 3 more TCP connections in parallel. So the effective time $2n + 4$ round trips.

c) With persistent HTTP we'll still need to query all n DNS servers for a total of $2n$ round trips accounting for TCP connections. Then for HTTP, we can open up a single TCP connection which takes 1 round trip, then query the 9 objects serially over HTTP taking 9 round trips. So the total time is $2n+10$ round trips.

P18

a) A whois database stores registered users or assignees of an Internet resource such as domain name, IP address block, or an autonomous system. Some of the information it returns in response to a query are the contact information, name servers, and created date.

b) I used whois.registrarsafe.com to obtain a DNS server facebook.com, which is a.ns.facebook.com. I also used whois.markmonitor.com to obtain a DNS server for amazon.com which is ns1.p31.dynect.net.

c) My local DNS is 8.8.8.8 which is a google DNS. When I ran a type A query

for google.com on my local DNS it returned the IP address 172.217.4.78 for google.com.
When I ran a type A query for google.com against the facebook DNS and the amazon DNS I got the same response "** server can't find google.com: REFUSED".
When I ran an NS query against my local DNS for google.com it returned 4 nameservers for google.com - ns1.google.com, ns2.google.com, ns3.google.com, and ns4.google.com.
When I ran NS queries against the facebook and amazon DNS servers for google.com I got the same response "** server can't find google.com: REFUSED".
When I ran an MX query against my local DNS for google.com it returned a list of "mail exhanger" fields, for example "google.com mail exchanger = 10 aspmx.l.google.com."
When I ran MX queries against the facebook and amazon DNS servers for google.com I got the same response "** server can't find google.com: REFUSED".

d) Querying my local DNS for the A records of amazon.com returns 3 IP addresses - 176.32.98.166, 205.251.242.103, and 176.32.103.205. A query for ncsu.edu returns one IP address 152.1.27.202.

e) The range returned by a query to the ARIN whois database for ncsu.edu is 152.1.0.0 - 152.1.255.255.

f) A hacker can use the nslookup tool on a domain such as ncsu.edu to get an IP address for that domain. Then use that IP address to query a whois database to get the full range of IP addresses used by that organization and points of contact. The hacker can then launch an attack on IP addresses in that range and try to extort the organization by contacting the points of contact.

g) The whois database is public because it holds domain owners accountable, verifies ownership of domain names and IP addresses, and facilitates domain transfer. If a domain is being used for malicious reasons you can track down the individual responsible. It also allows you to check if a domain is avaiable and contact the person who owns the domain if you want to try to acquire it.

P22

For P2P the server must upload all of the bits at least once, the distribution is not complete until the slowest peer is done downloading, and the entire system must upload NF bits using the total upload capacity of the server and peers combined. The distribution time will be at least the maximum of these 3 values. For N=10 and u=300 Kbps we have $max(5000, 7500, 4545.45) = 7500$ secs.

The max time will be 7500 secs for higher upload rates as well.
For N=100 and u=300 Kbps we have $max(5000, 7500, 25000) = 25000$ secs. For N=100 and u=700 Kbps we have $max(5000, 7500, 15000) = 15000$ secs. For N=100 and u=2 Mbps we have $max(5000, 7500, 6522) = 7500$ secs.
For N=1000 and u=300 Kbps we have $max(5000, 7500, 45455) = 45455$ secs. For N=1000 and u=700 Kbps we have $max(5000, 7500, 20548) = 20548$ secs. For N=1000 and u=2 Mbps we have $max(5000, 7500, 7389) = 7500$ secs.

	N=10	N=100	N=1000
u=300 Kbps	7500	25000	45455
u=700 Kbps	7500	15000	20548
u=2 Mbps	7500	7500	7500

For client server, the server must transmit NF bits in total which is 150 Gbits. The time it takes the server to upload these bits is $150 \cdot 10^9/30 \cdot 10^6 = 5000$ secs. The minimum distribution time for each peer to download the file is $15 \cdot 10^9/2 \cdot 10^6 = 7500$ secs. The distribution time is the maximum between the server upload time and slowest peer download time, so it is 7500 secs in this case. This is a lower bound but we'll assume the server can achieve this.
The slowest peer download time remains 7500 secs as N grows, so the distribution time for 100 peers is 50000 secs, and for 1000 peers it is 500000 secs. The upload rate of the peers doesn't matter in the client server architecture.

	N=10	N=100	N=1000
u=300 Kbps	7500	50000	500000
u=700 Kbps	7500	50000	500000
u=2 Mbps	7500	50000	500000

P23

a) Since $\frac{u_s}{N} \leq d_{min}$, the bottleneck will be the server uploading to the peers rather than the slowest download rate among the peers. We can rewrite the inequality as $\frac{1}{d_{min}} \leq \frac{N}{u_s}$, multiplying both sides by F gives $\frac{F}{d_{min}} \leq \frac{NF}{u_s}$. The maximum among the 2 terms will be the lower bound on the distribution time of the system, so we have $\frac{NF}{u_s}$. This lower bound can be achieved if all of the N peers are downloading from the server in parallel, and the full bandwidth of the server is dedicated to uploading to the peers.

b) Since $\frac{u_s}{N} \geq d_{min}$, the bottleneck will be the slowest download rate among the peers rather than the server uploading to the peers. We can rewrite the inequality as $\frac{1}{d_{min}} \geq \frac{N}{u_s}$, multiplying both sides by F gives $\frac{F}{d_{min}} \geq \frac{NF}{u_s}$. The maximum among the 2 terms will be the lower bound on the distribution time of the system, so we have $\frac{F}{d_{min}}$. This lower bound can be achieved if the average

download rate for all of the peers is d_{min}. Besides the peer with d_{min}, not all peers need to have a dedicated port as long as when each peer that had to wait starts to download it downloads at a higher rate to make up for idle time so that its average download rate is d_{min}. The peer with d_{min} needs a dedicated port the entire time.

c) The minimum distribution time is given by $max(\frac{NF}{u_s}, \frac{F}{d_{min}})$ because the larger of the 2 will be a bottleneck, giving a lower bound on the distrbution time. If the time it takes to upload all the bits to all peers is greater than the time it takes for the slowest peer to download the bits then the greater download speed will not matter, it will have to conform to the upload speed. A similar argument applies to the converse. In order to meet the lower bound, if the upload time is greater then the server needs to upload to all peers in parallel. If the download time is greater then the slowest peer needs to have a dedicated port for the duration of the process.

P26

a) Yes his claim is possible, his free riding just slows down the collective upload rate of the system. So it will take longer for him and everyone else to download the file.
b) If the collection of computers he uses in the computer lab are not free riders then they will add to the collective upload rate of the system. This will speed up the downloads for everyone, including free riders like Bob's personal computer.

Wireshark Lab

1) Both my browser and the server are running HTTP 1.1.
2) The Accept-Language field in the request header lists en-US and en.
3) The IP address of my computer is 192.168.1.67 and the IP address of the gaia.cs.umass.edu server is 128.119.245.12.
4) The status code returned from the server to my browser is 200.
5) The Last-Modified field in the response says Mon, 27 May 2019 05:59:01 GMT. The Date of the response is Mon, 27 May 2019 19:09:31 GMT.
6) The Content-Length field in the response is 128 bytes. The total length of the response is 552 bytes.
7) There is some data in the packet content window preceding the headers in the packet listing window for the HTTP response, but it is not readable. It appears seemingly random characters.
The screenshot for these answers 1-7 is in the figure *Wireshark 1*

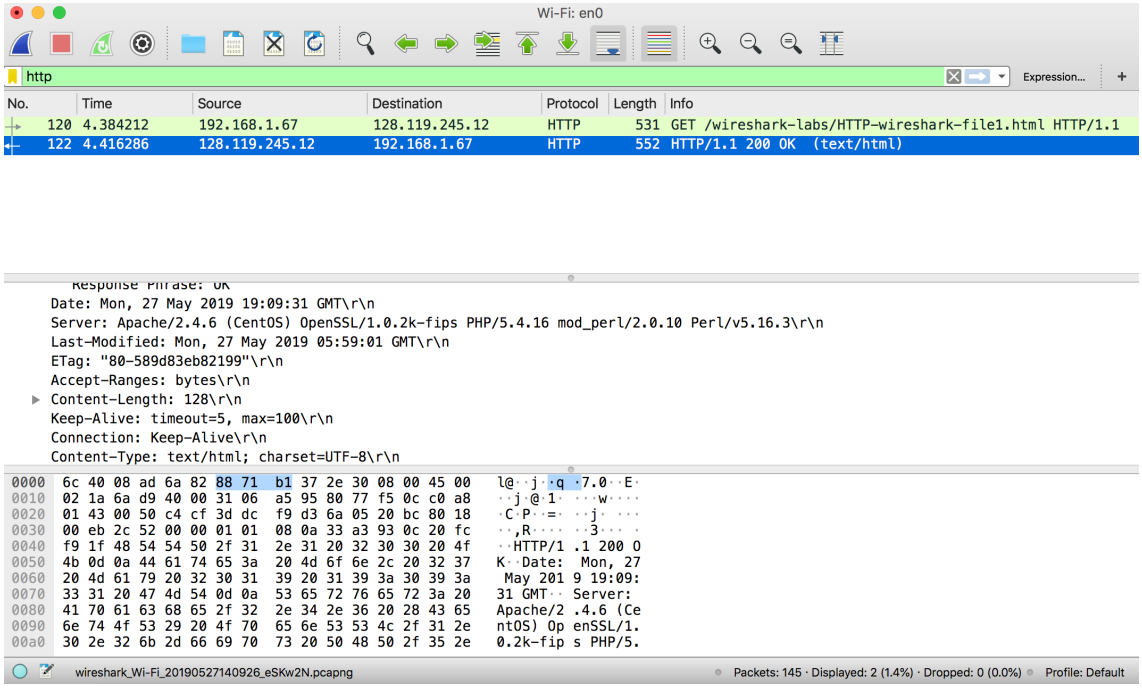

Figure 1: Wireshark 1

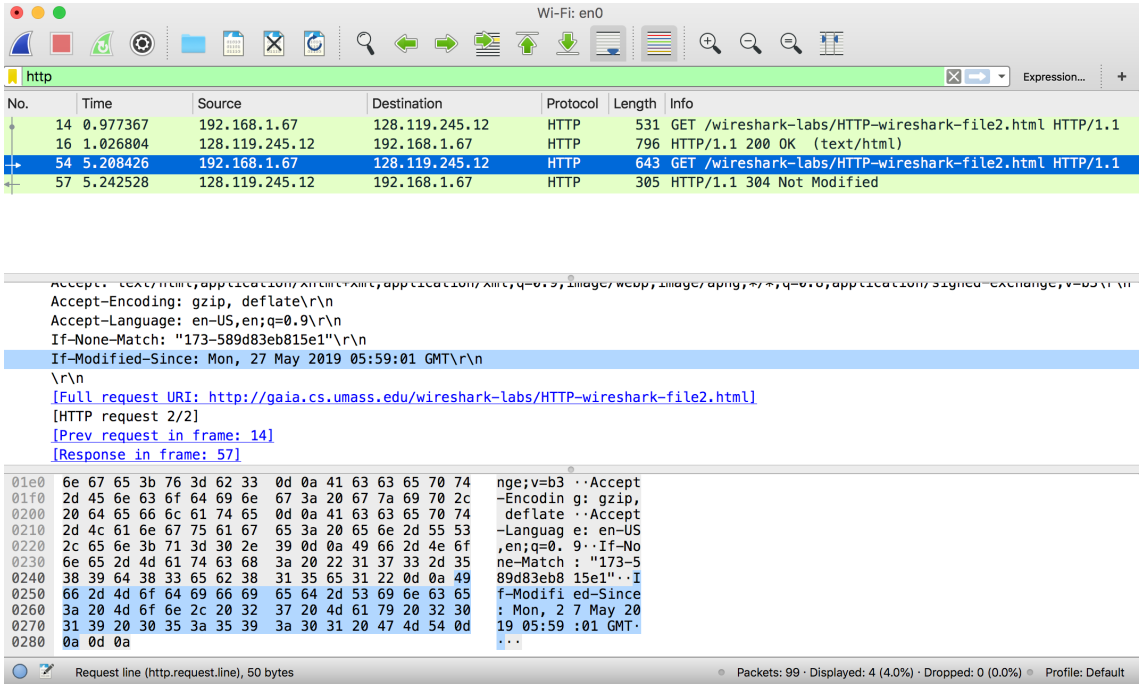

Figure 2: Wireshark 2

8) No there is no IF-MODIFIED-SINCE field in the first GET request.
9) Yes the server explicitly returned the contents of the file, I can tell from the 200 response code.
10) Yes there is an IF-MODIFIED-SINCE field in the second request, its value is Mon, 27 May 2019 05:59:01 GMT.
11) The response code and phrase for the second response is 304 Not Modified. This means a proxy or web cache returned the file rather than the server itself. The proxy will check if the file has been modified since the IF-MODIFIED-SINCE value, and if it hasn't been modified the proxy will return the cached file.
The screenshot for these answers 8-11 is in the figure *Wireshark 2*

12) My browser sent one HTTP GET request message. The packer number for the GET message is 95.
13) The packet number for the status code and phrase of the response is 97.
14) 200 OK
15) 4 TCP segments were needed to carry the HTTP response and text.
The screenshot for these answers 12-15 is in the figure *Wireshark 3*

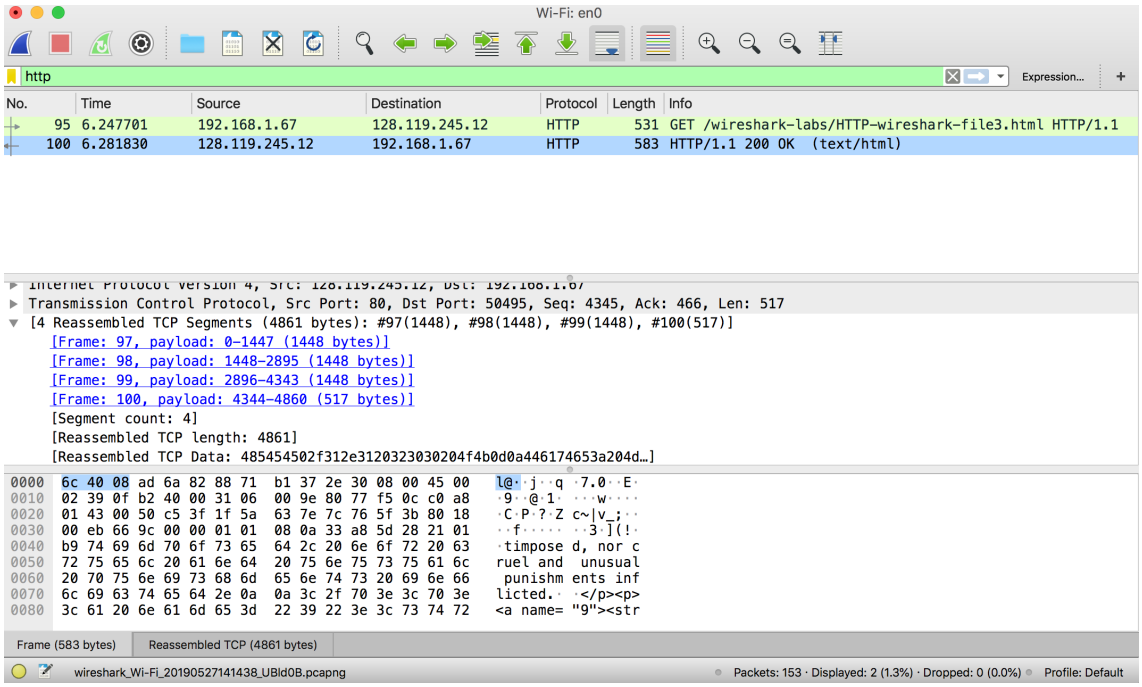

Figure 3: Wireshark 3

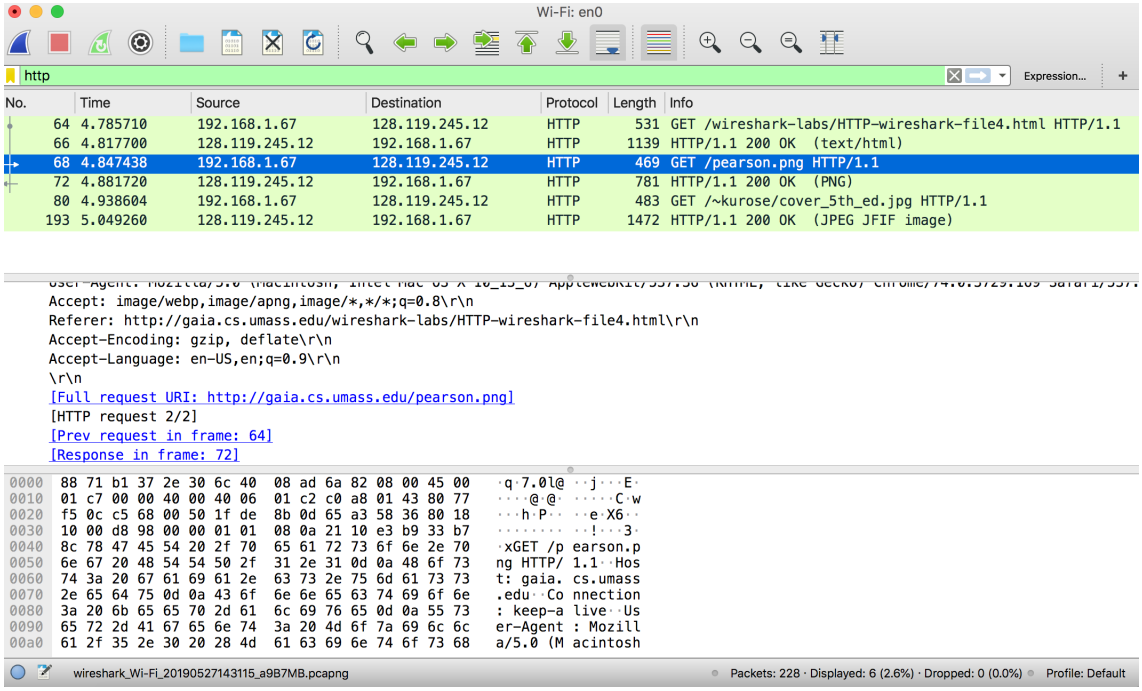

Figure 4: Wireshark 4

16) My browser sent 3 HTTP GET requests. They were sent to http://128.119.245.12/wireshark-labs/HTTP-wireshark-file4.html, http://128.119.245.12/pearson.png, and http://128.119.245.12/ kurose/cover
17) It looks like they were sent in parallel because the request for pearson.png says "HTTP Request 2/2" and "Prev request in frame: 64" referring to the request for the html file which was request 1/2. The request for the cover png says "HTTP Request 1/1" and doesn't make a reference to a previous request. This seems to indicate they weren't all part of a pipeline of requests, and the cover png executed at the same time the pearson.png executed.
The screenshot for these answers 16-17 is in the figure *Wireshark 4*

18) The server's response to the intial HTTP GET is 401 Unauthorized.
19) In the second GET request there is now an Authorization field with the value being "Basic " followed by the credentials in base64 encoding.
The screenshot for these answers 18-19 is in the figure *Wireshark 5*

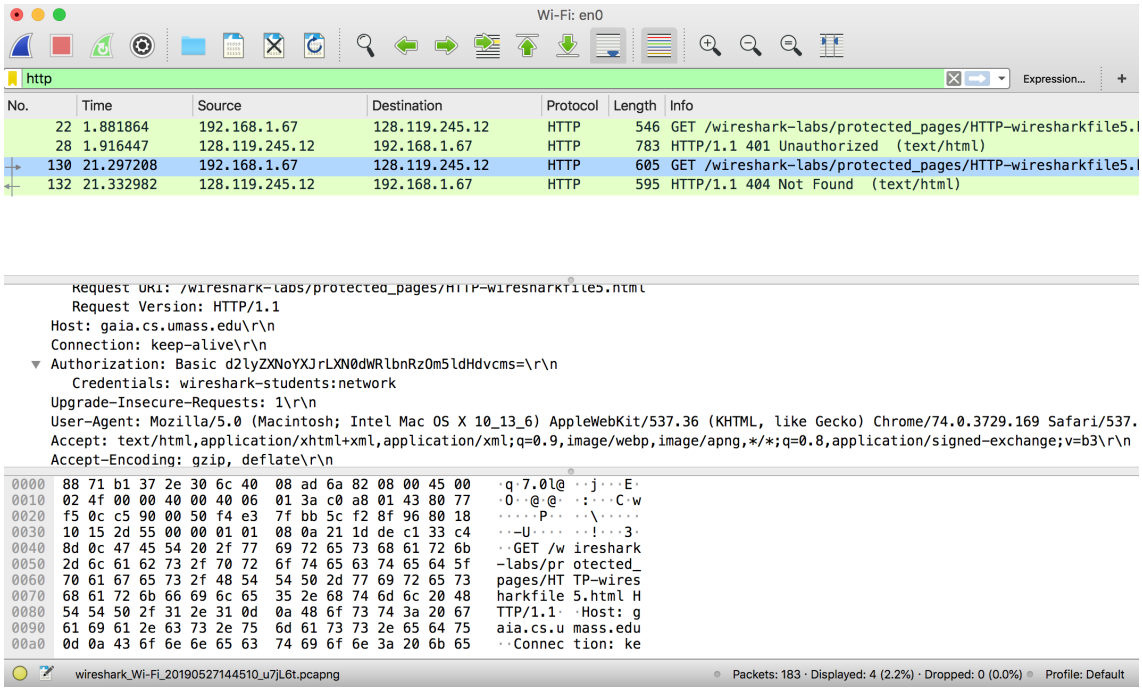

Figure 5: Wireshark 5

Internet Indirection Infrastructure

1) i3 decouples the act of sending from the act of receiving. i3 nodes forward packets to receivers that are interested in the ID associated with that packet. A receiver will contact an i3 node to insert a trigger consisting of (id, addr). The i3 node will insert that trigger in the appropriate i3 node handling that id. A sender will send packets consisting of (id, data), and these packets will be sent to the i3 node handling that id. The i3 node will forward the packets to receivers that have inserted a trigger for that exact id, or if at least k bits match in the id.
The communication primitives of i3 are mobility, multicast, and anycast. Mobility allows for the sender and receiver to move across the network and be assigned a new IP address while still maintaining a connection to the i3 service. When a receiver moves it just needs to update its trigger to reflect its new IP address. A sender doesn't need to do anything when it moves. Multicast allows multiple receivers to receive the same incoming packet to an i3 node. When an i3 node receives a packet, it checks all its triggers, and sends the packet to all triggers with a matching id. Anycast ensures that a packet is delivered to exactly one receiver in a group. A group of receivers will register triggers with the k most significant bits matching. The remaining m-k bits are then selected to ensure that only one member of the group receives certain packets sent by the sender.
ID stack generalization allows for a stack of identifiers to be sent with a packet instead of a single ID or used in a trigger instead of a single IP address. When an i3 node receives a packet with a stack of IDs, it checks if there is a trigger matching the first ID, if not it pops the first ID off the stack and continues the search. For the first ID it finds matching a trigger, that ID is replaced by the trigger's ID stack at the top of the stack. If the ID now at the top of the stack is an IP address, then the ID stack and data are forwarded to that IP address, where the application running on that IP address is expected to forward that packet to the next ID on that stack after processing the data. If the ID now at the top of the stack is another ID, then the packet is forwarded to the i3 node handling that ID.

2) The applications of i3 described in the paper are service composition, heterogeneous multicast, server selection, and large scale multicast. Service composition allows for data to be preprocessed before reaching a receiver. An intermediate device like a transcoder can insert a trigger which matches the first id on the stack of a packet, and the receiver can insert a trigger which matched the second id on the stack of the packet. The data will first go to the transcoder and then be inserted in an i3 node matching the second id, at which point it will be forwarded to the receiver.
Heterogeneous multicast allows for sending packets to many receivers that may or may not need some preprocessing on the data. A receiver that needs data transcoded can insert a trigger with an id stack with the transcoder at the top

of the stack, the data first goes to the transcoder and eventually reaches the receiver. A receiver that doesn't need preprocessing can insert a simple trigger without a stack and directly receive the raw data.
Server selection allows for a group of servers to create triggers with the first k bits of the id identical and the remaining m-k bits strategically designed so that a certain sender's data goes to a certain receiver in the group, for example the receiver closest to the sender.
Large scale multicast allows for creating a hierarchy of i3 nodes with the top of the hierarchy having multiple triggers forwarding to different IDs but the same ID in the first coordinate. This can load balance packets so that a single i3 node isn't handling the multicast of every packet with a particular ID.
Another potential application could be in an IOT application where a device is configured to send messages to an i3 node. You can then have receivers subscribing to certain devices by creating triggers and dropping their subscriptions by updating their triggers if they don't require the data anymore. The devices wouldn't have to be configured to send data to any particular receiver.

3) A public trigger is known by all end-hosts in the system and is used to allow an end host to contact another end host. Private triggers are chosen by a few end hosts and are short lived. Private triggers can improve routing by end hosts storing their private triggers on i3 nodes geographically close to avoid excessive triangle routing.
Private triggers improve security in that they are not known by all hosts, they are only stored on a single i3 node. For multicasting to work, i3 allows identical trigger IDs, but for a hacker to get traffic routed to him, he would have to use brute force to guess the ID or hack into the i3 node itself. Also the IDs can periodically change in case a hacker discovers an ID.

The Akamai Network

1) There are many challenges including:

- Peering point congestion - congestion in the interior of the network where network exchange traffic happens. These areas are not as well invested in as the edges of the Internet.
- Inefficient routing protocols - BGP does not factor in network topology, latencies, or real time congestion.
- Unreliable networks - natural disasters, misconfiguration, cable cuts.
- Inefficient communication protocols - TCP does not perform well on links with high latency and packet loss. Requires many extra acknowledgements between peers beyond the actual data being transferred.

- Scalability - difficult to know when and how much to scale an application, and to scale network bandwidth.
- Updating old or suboptimal applications and clients.

2) The main components of Akamai's delivery network are:

- Mapping system - Maps a URL from a client browser to an IP address of an edge server that can provide the requested content.
- Edge server platform - global deployment of 1000s of edge servers responsible for providing content to end users.
- Transport system - reliable connection from edge servers to origin servers.
- Communications and control system - Distributes status information, control messages and configuration updates.
- Data collection and analysis system - collects and processes data from sources such as server logs, client logs, and network information. Can be used for monitoring, reporting, billing, etc.
- Management portal - Allows enterprises to control/configure how content and apps are served to end user, and provides visibility such as reports.

3) The main architectural solutions for streaming and content delivery are:

- Distributing server clusters all over the world rather than a smaller amount larger datacenters. Necessary because of the fragmentation of the internet to reach a majority of users. Also increases availability to recover from servers failing.
- An edge based platform with servers in thousands of locations described above can handle demand for video which can reach 100 Tbps.
- Global monitoring infrastructure to gather metrics on streams and simulating streaming to users.
- For content delivery a tiered system is used to minimize requests and connections to the origin server. Edge clusters first try to serve the content, if they can't they check with parent clusters holding additional content. This reduces requests to origin servers and prevents origin servers from having to maintain thousands of connections to edge servers.
- For streaming, once the stream is captured and encoded it is sent to a cluster of servers called entrypoints. From there the stream is sent to reflectors where the stream can be replicated and sent to a massive amount of edge servers, which sent the stream to users. Uses a publish-subscribe model where edge servers subscribe to streams the entrypoints have. Reflectors also determine the best path between the entrypoint and edge cluster, can use multiple disjoint paths replicating the stream to recover loss on some links that were low quality.

4) The main difference between application delivery and content delivery is the content is static and can be cached, while applications are dynamic and must produce a unique response in real time, which means their responses cannot be cached.
The main techniques Akamai uses to optimize performance of applications are:

- Creating an overlay network of Akamai servers to deliver the app from origin to end user. This allows for the data to be transported using a custom transport protocol that takes real time network data into account in order to find optimal paths and changing the window size. Also it can send the data over multiple paths to reduce packet loss. It can also prefetch objects from applications so they are available in an edge cache for the end user.

- Edge computing is another method for optimizing application performance. This actually executes the application on an edge server close to the user rather than the origin to speed up communications.

5) The main implications are that applications serving from static databases, performing data collection or performing some transformations on user input can be deployed on an edge server close to the user rather than an origin server. This will greatly speed up communication between the user and the application. Applications that rely heavily on a transactional database will still need to run on the origin, but edge computing could make application designers think about dividing their applications into components which can be run on the edge and components that must run on the origin.

6) The main parts of Akamai's mapping system are:

- The first part is the scoring system. This creates a current topological map of the state of connectivity across the Internet. This is done by dividing the Internet up into groups of IP addresses and observing how well they connect with one another using real time and historical data. This score is used to help select an edge cluster for an end user to connect to.

- The second part is real time mapping which creates maps that can be used to identify the best edge cluster for a user to connect to. First a user is mapped to a cluster, and then the user is mapped to an individual machine in that cluster. These processes involve a lot of real time analytics on latency, trace routes, logs, etc.