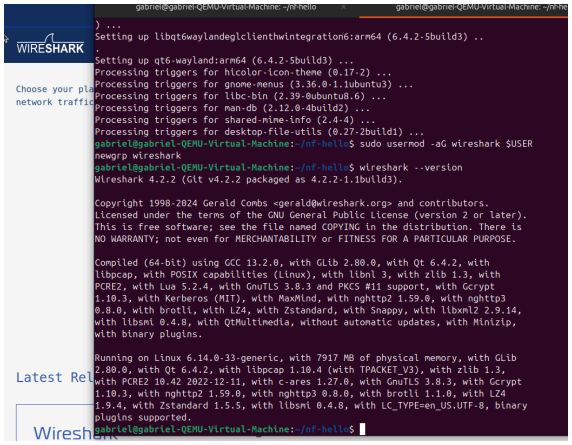
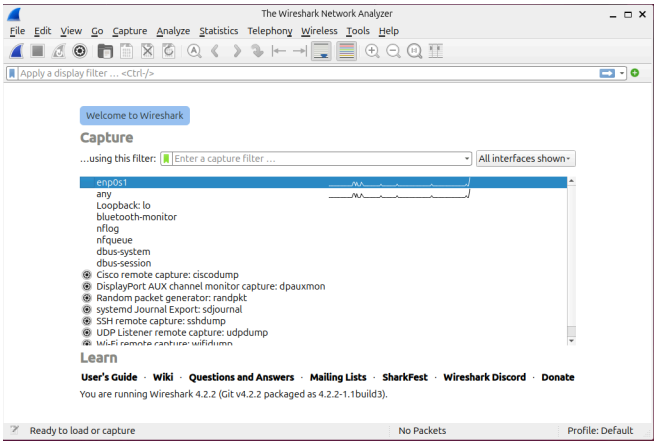
Computer Networks Homework 2

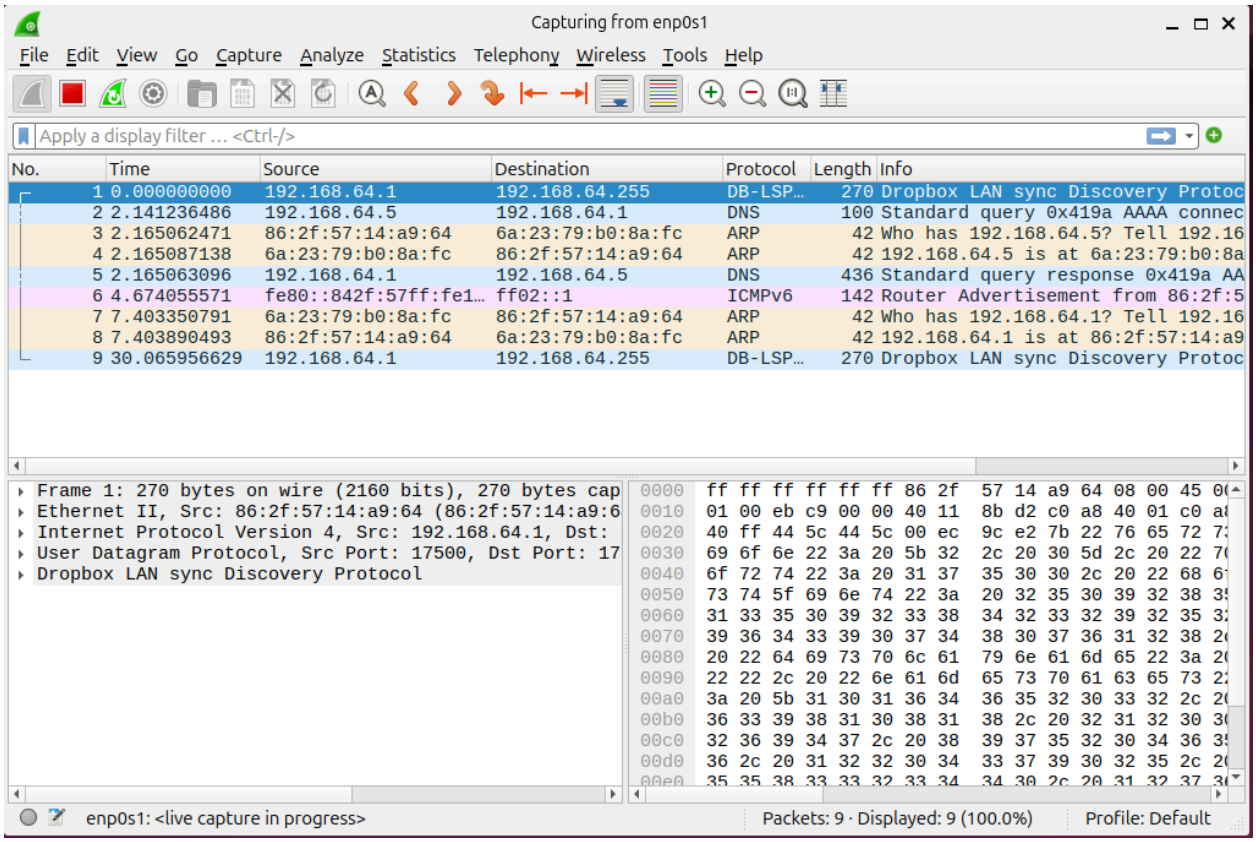
Gabriel Urbaitis
CS 585 10-19-25

1.

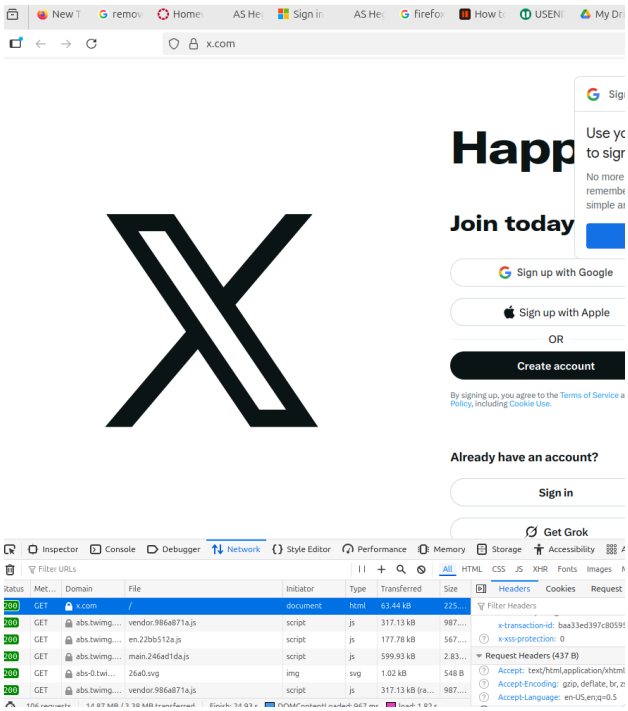




2.



3.
1(a) Accept-Language: en-US,en;q=0.5

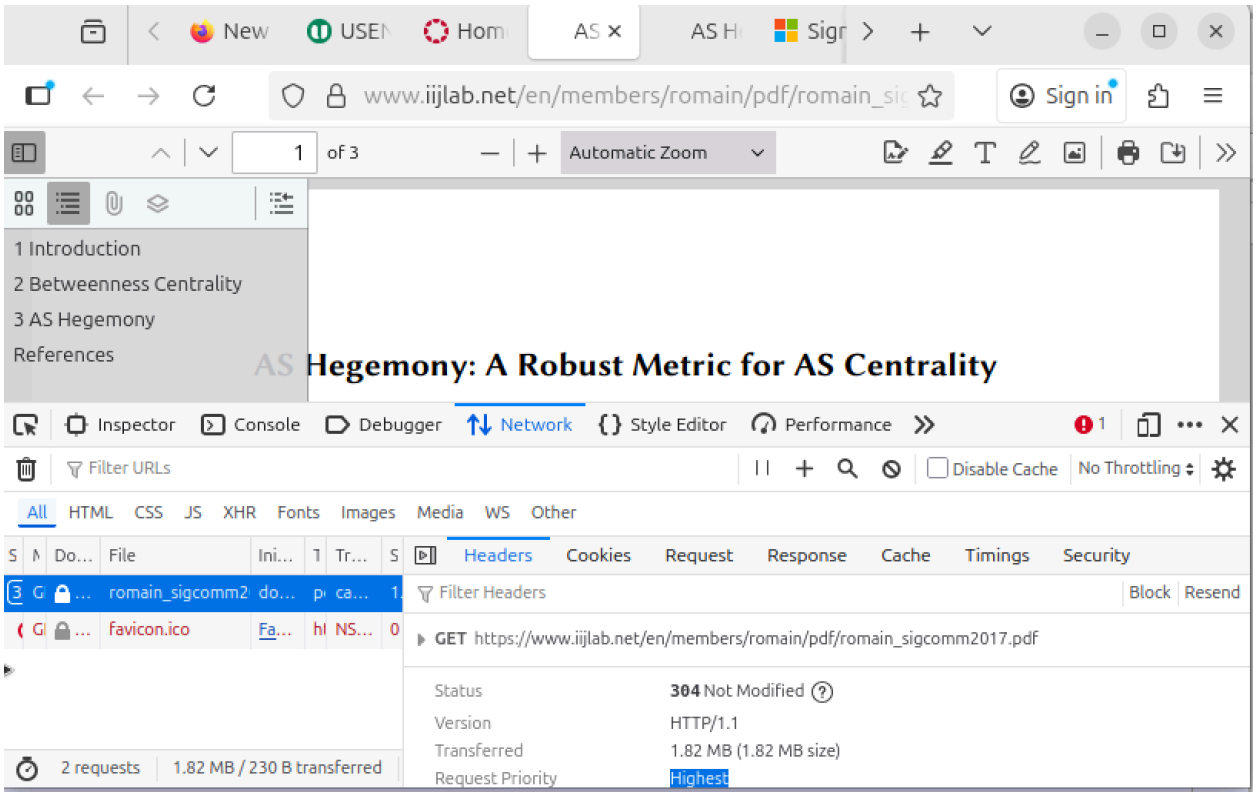


b-g are found the exact same way
1(b) Accept-Language: en-US,en;q=0.5
1(c) Accept-Language: en-US,en;q=0.5
1(d) Accept-Language: en-US,en;q=0.5
1(e) Accept-Language: en-US,en;q=0.5
1(f) Accept-Language: en-US,en;q=0.5
1(g) Accept-Language: en-US,en;q=0.5

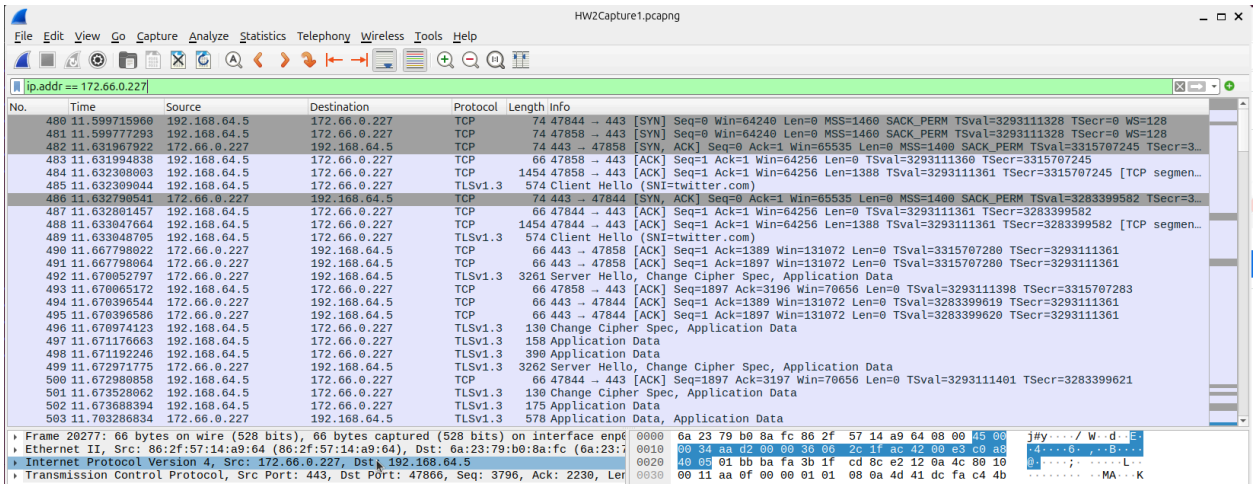
Question 2 is with 4 and 5 for each one, as they can be answered from individual Wireshark screenshots

3(a) 200
3(b) 200
3(c) 200
3(d) 200
3(e) 200
3(f) 200

a-f are found the exact same way as (g)



3(g) 304 Not Modified



2(a) My IP: 192.168.64.5, URL IP: 172.66.0.227

4(a) – x.com (Twitter) — Sequence of Events

Packets 481 – 503 show the complete HTTPS session setup for x.com (SNI: twitter.com, IP 172.66.0.227).

1. TCP 3-way handshake – Packet 481 is the SYN from my host (192.168.64.5 → 172.66.0.227, port 443).

Packet 486 is the SYN-ACK from the server, and the next packet is my ACK, completing the handshake.

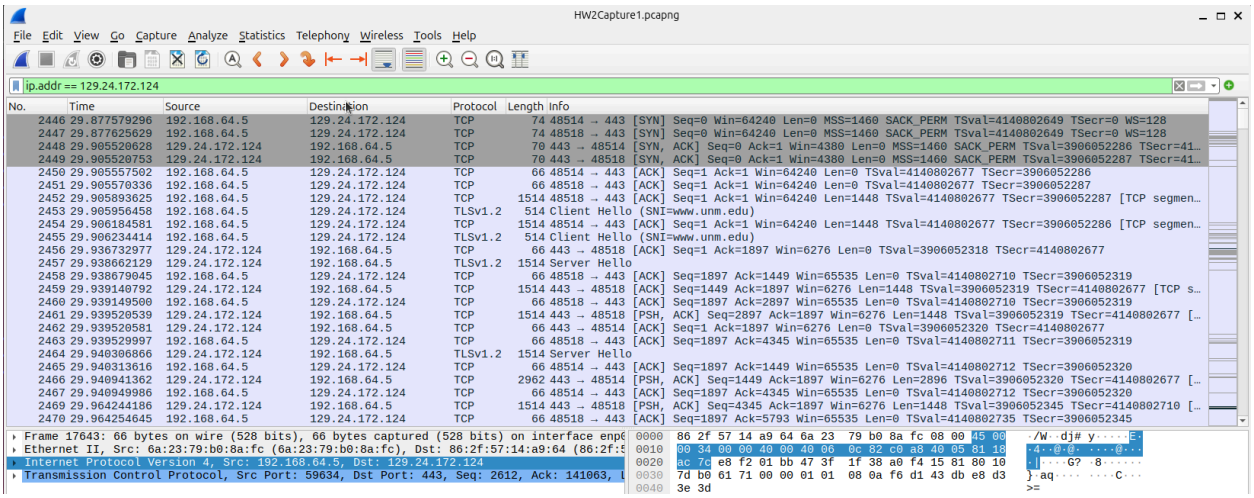
2. TLS 1.3 handshake – Packet 485 contains the Client Hello, with the Server Name Indication field showing twitter.com.

Packets 490 – 493 show the Server Hello, key exchange, and cipher negotiation.

3. Change Cipher Spec / Finished – Packets 494 – 496 indicate encryption is established.

4. Application Data exchange – Packets 497 – 503 carry the encrypted HTTPS traffic for the X.com home page request and its response.

5(a) Protocols: TCP, TLSv1.3 Ports: 47866 (client ephemeral port), 443 (HTTPS server port)



2(b) My IP: 192.168.64.5, URL IP: 129.24.172.124
4(b) – cs.unm.edu — Sequence of Events

Packets 2446 – 2469 show the HTTPS session setup for cs.unm.edu (129.24.172.124).

1. TCP 3-way handshake – Packet 2446 is the SYN from my host (192.168.64.5 → 129.24.172.124, port 443).

Packet 2449 is the SYN-ACK reply from the server, and the next ACK completes the handshake.

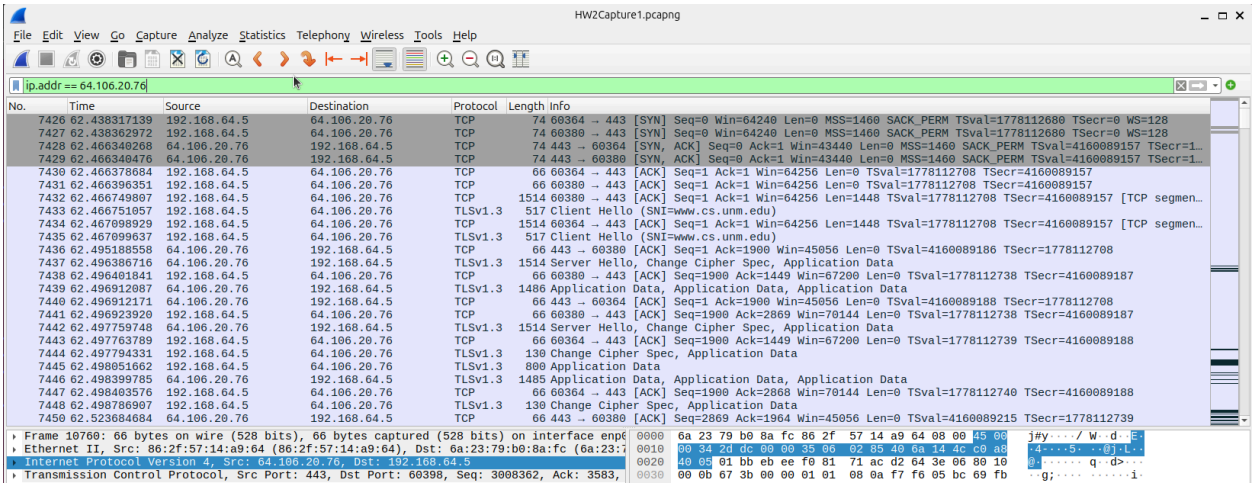
2. TLS 1.2 handshake – Packet 2451 contains the Client Hello, with SNI www.unm.edu.

Packets 2456 – 2458 show the Server Hello, certificate exchange, and cipher suite negotiation.

3. Change Cipher Spec / Finished – Packets 2459 – 2461 indicate the start of encrypted communication.

4. Application Data exchange – Packets 2462 – 2469 carry encrypted HTTPS payloads for the page request and response.

5(b) Protocols: TCP, TLSv1.2 Ports: 59634 (client ephemeral port), 443 (HTTPS server port)



2(c) My IP: 192.168.64.5, URL IP: 64.106.20.76

4(c) – www.unm.edu — Sequence of Events
Packets 7426 – 7450 show the HTTPS session setup for www.unm.edu (64.106.20.76).

1. TCP 3-way handshake – Packet 7426 is the SYN from my host (192.168.64.5 → 64.106.20.76, port 443).

Packet 7429 is the SYN-ACK from the server, and the following packet completes the handshake with an ACK.

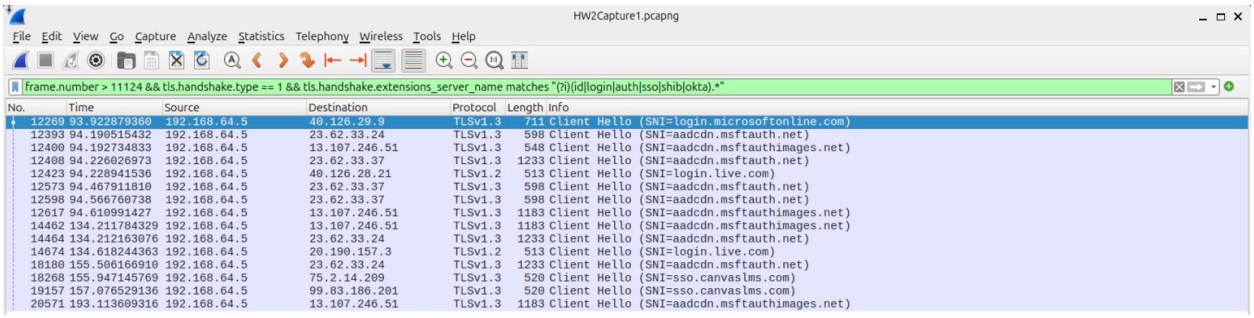
2. TLS 1.3 handshake – Packet 7432 contains the Client Hello, listing supported cipher suites and SNI www.cs.unm.edu.

Packets 7435 – 7438 include the Server Hello, certificate exchange, and cipher negotiation.

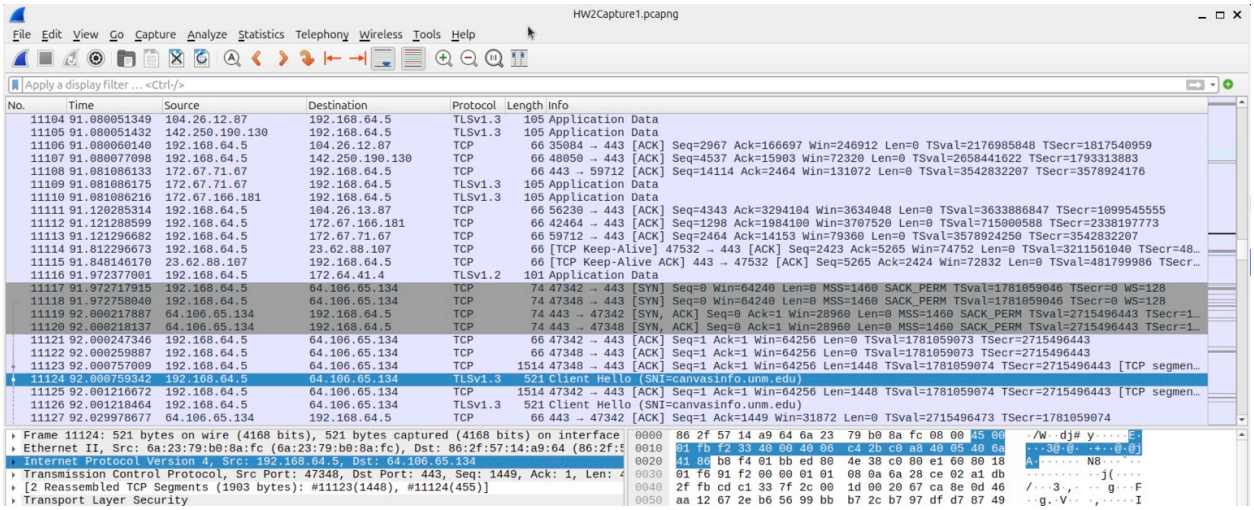
3. Change Cipher Spec / Finished – Packets 7439 – 7441 establish encryption for the TLS session.

4. Application Data exchange – Packets 7442 – 7450 carry the encrypted HTTPS content, corresponding to the web page request and server responses.

5(c) Protocols: TCP, TLSv1.3
Ports: 60398 (client ephemeral port), 443 (HTTPS server port)



Above, Microsoft login, Below, Canvas



2(d) My IP: My IP 192.168.64.5
Canvas (initial): canvasinfo.unm.edu → Dest IP: 64.106.65.134
Microsoft login: login.microsoftonline.com → Dest IP: 40.126.29.9 (row 12269)
Microsoft auth CDN: aadcdn.msauth.net → Dest IP: 23.60.163.24 (row 12393)
Microsoft auth images CDN: aadcdn.msauthimages.net → Dest IP: 23.60.163.24 (row 12400)
Live ID federation: login.live.com → Dest IPs: 13.107.246.51, 40.126.28.37 (rows 12423, 12425)
Canvas.lms → Dest IP:75.2.14.209

4(d) – Canvas (canvas.unm.edu) — Sequence of Events (Microsoft SSO Login)

Packets 11119 – 11148 and 12269 – 20571 together show the full HTTPS login process for Canvas at the University of New Mexico, which uses Microsoft 365 (Azure AD) single sign-on authentication.

1. TCP 3-way handshake – Packet 11119 is the SYN from my host (192.168.64.5 → 64.106.65.134, port 443). Packet 11120 is the SYN-ACK reply from the server, and 11122 is my ACK, completing the handshake.

2. TLS 1.3 handshake to Canvas – Packet 11124 contains the Client Hello, with SNI canvasinfo.unm.edu. Packets 11133 – 11134 show the Server Hello, certificate exchange, and cipher negotiation.

3. Change Cipher Spec / Finished – Packets 11135 – 11137 mark the transition to encrypted communication.

4. Initial Application Data (exchange with Canvas) – Packets 11138 – 11148 carry encrypted HTTPS payloads representing the Canvas login page request. Canvas then redirects the browser to Microsoft's authentication endpoint for single sign-on.

5. Redirect to Microsoft Login Service (Azure AD handshake) – Starting at packet 12269, the browser initiates new TLS 1.3 Client Hello sessions to:

• login.microsoftonline.com (12269) – initial authentication endpoint.

• aadcdn.msauth.net and aadcdn.msauthimages.net (12393 – 12433) – content delivery hosts for Microsoft login resources.

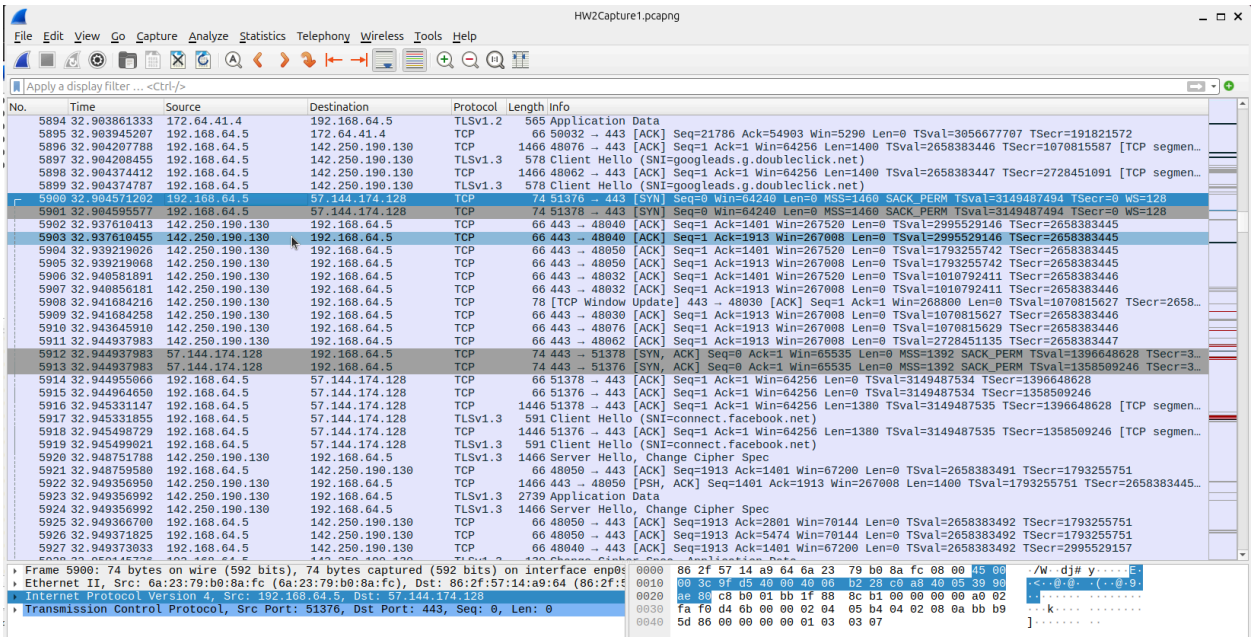
• login.live.com (12423, 12464) – Live ID federation service.

Each of these hosts responds with TLS 1.3 Server Hello and encrypted Application Data packets, completing Microsoft's secure sign-in sequence.

6. Credential and Token Exchange (Encrypted) – Within the following Application Data frames, the browser submits user credentials and receives OAuth tokens and redirect instructions from Microsoft 365 (Azure AD). These packets are fully encrypted inside TLS and therefore not readable in the capture.

7. Return to Canvas Session (SSO redirect) – Near the end of the chain (packets ≈ 20500 – 20571), new Client Hello messages target ssso.canvaslms.com and canvaslms.com, marking the final redirect back to Canvas with the authenticated session. Canvas then establishes a fresh TLS 1.3 connection to deliver the logged-in dashboard.

5(d) Protocols: TCP, TLSv1.3 , TLSv1.2
Ports: 47438 (canvasinfo) (44726 (microsoftonline), 46154, 53756, 46766, 46780, 46788, 53768, 36358, 33874, 60882, 43600 (msftauth), 55036, 35208(login.live), 60594, 35048(canvaslms) (client ephemeral ports), 443 (HTTPS server port)



2(e) My IP: 192.168.64.5 URL IP: 57.144.174.128
4(e) – facebook.com — Sequence of Events
Packets 5900 – 5926 show the HTTPS session setup for facebook.com

1. TCP 3-way handshake – Packet 5900 is the SYN from my host (192.168.64.5 → 57.144.174.128, port 443).

Packet 5912 is the SYN-ACK from the server, and the following packet is my ACK, completing the TCP handshake.

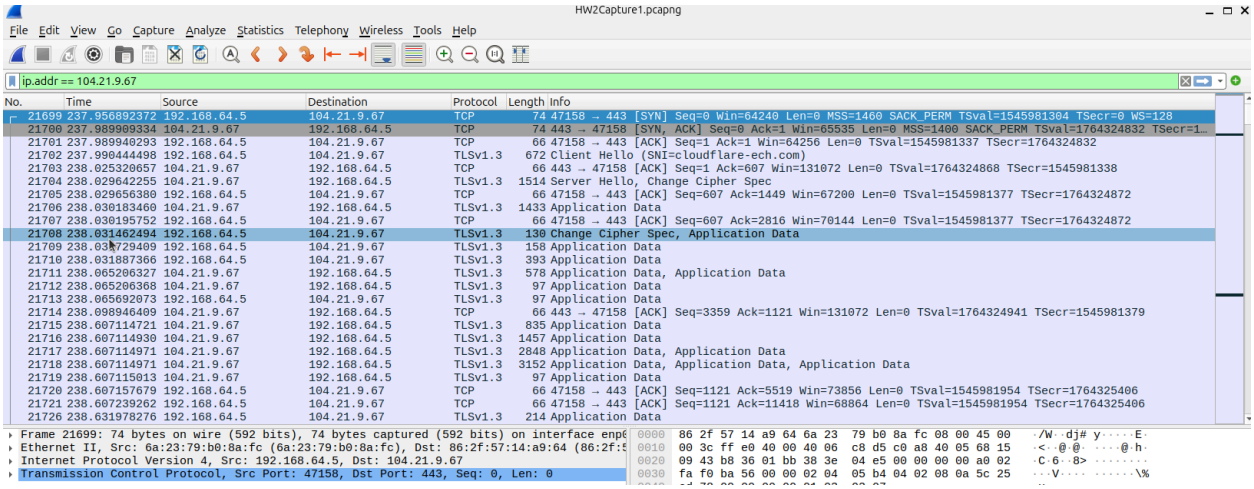
2. TLS 1.3 handshake – Packet 5914 contains the Client Hello, including the SNI connect.facebook.net.

Packets 5916 – 5917 show the Server Hello, cipher negotiation, and certificate exchange.

3. Change Cipher Spec / Finished – Packets 5918 – 5920 complete encryption setup.

4. Application Data exchange – Packets 5921 – 5926 carry the encrypted HTTPS payloads that correspond to Facebook's dynamic content and resources (scripts, media, etc.).

5(e) Protocols: TCP, TLSv1.3. Ports: 51376 (client ephemeral port), 443 (HTTPS server port)



2(f) My IP 192.168.64.6, URL IP: 104.21.9.67

4(f) – usenix.org — Sequence of Events

Packets 21699 – 21725 show the complete HTTPS session setup for usenix.org (served via Cloudflare at 104.21.9.67).

1. TCP 3-way handshake – Packet 21699 is the SYN from my host (192.168.64.5 → 104.21.9.67, port 443).

Packet 21700 is the SYN-ACK reply from the server, and 21701 is my ACK, completing the handshake.

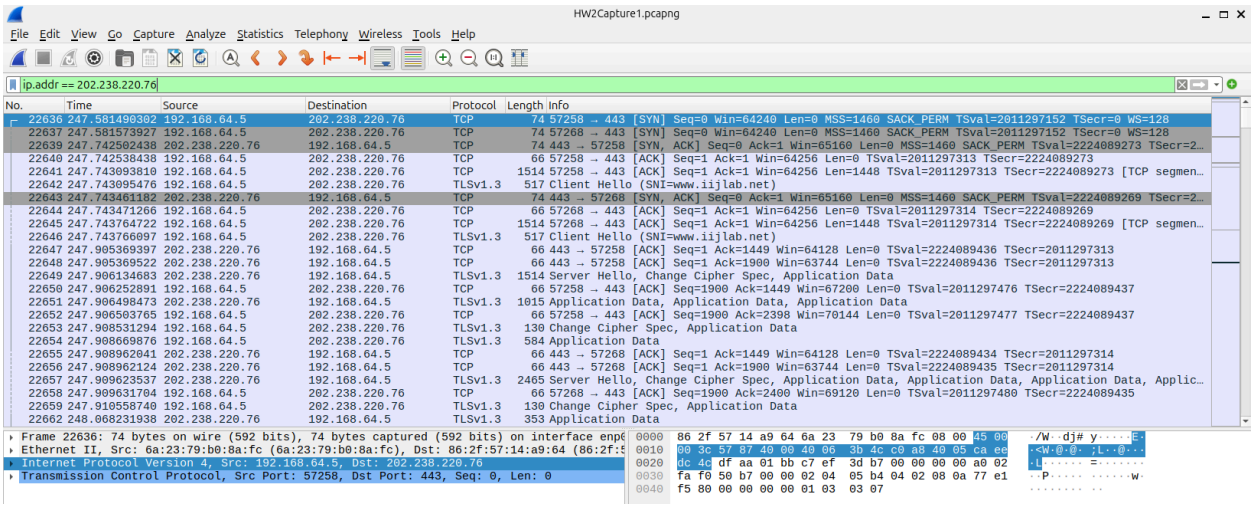
2. TLS 1.3 handshake – Packet 21703 contains the Client Hello, including SNI cloudflare-ech.com.

Packets 21704 and 21705 show the Server Hello and certificate exchange.

3. Change Cipher Spec / Finished – Packets 21706 – 21708 transition to encrypted communication.

4. Application Data exchange – Packets 21709 through 21725 carry encrypted HTTPS payloads representing my GET request and the server's responses.

5(f) Protocols: TCP, TLSv1.3, Ports: 47158 (client ephemeral port), 443 (HTTPS server port)



2(g) My IP 192.168.64.5, URL IP: 202.238.220.76

4(g) – iijlab.net (PDF) — Sequence of Events

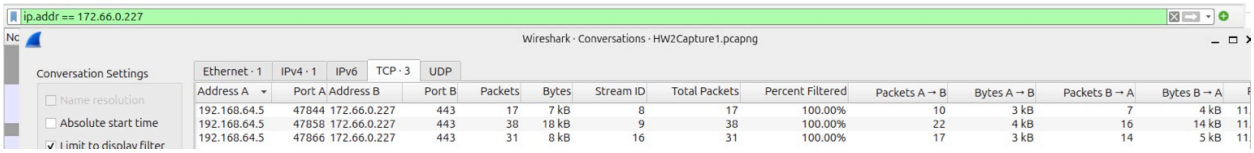
Packets 22636 – 22668 show the complete HTTPS session setup for iijlab.net (202.238.220.76).

1. TCP 3-way handshake – Packet 22636 is the SYN from my host (192.168.64.5 → 202.238.220.76, port 443).
   Packet 22637 is the SYN-ACK from the server, and 22638 is my ACK, completing the handshake.

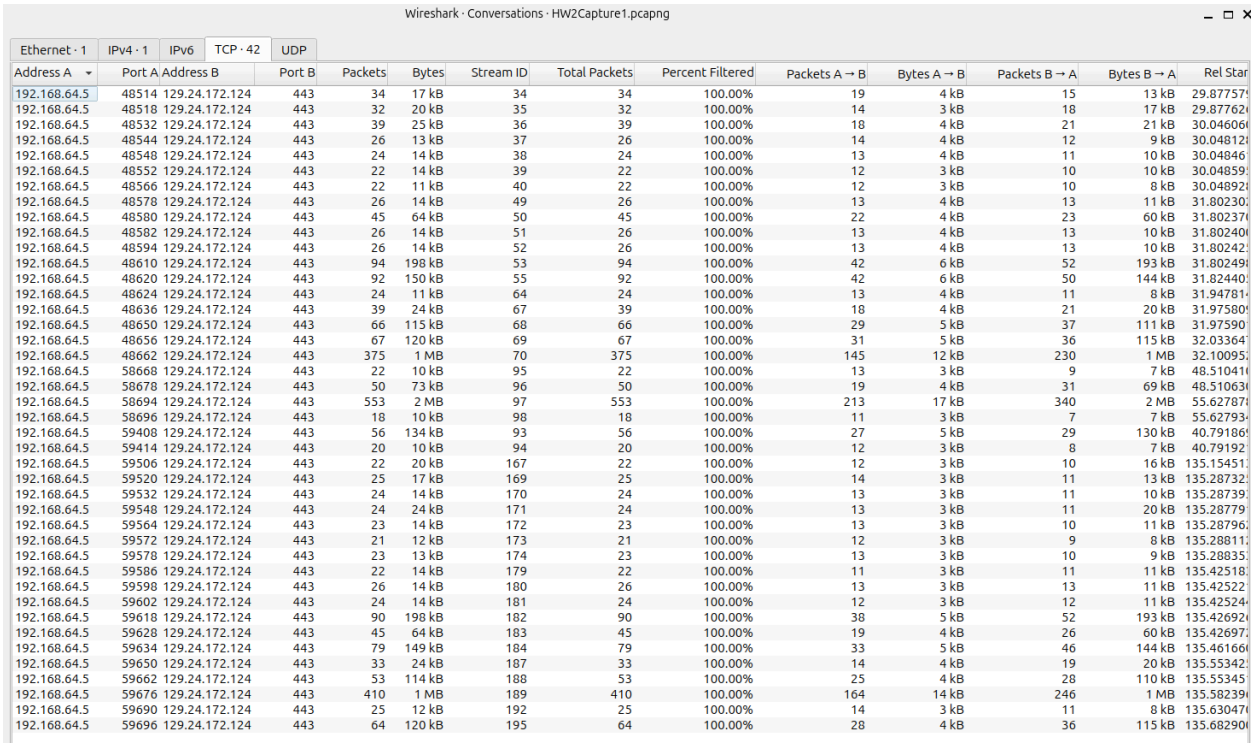
2. TLS 1.3 handshake – Packet 22641 contains the Client Hello, showing the SNI www.iijlab.net.
   Packets 22644 and 22645 show the Server Hello, certificate exchange, and cipher suite negotiation.

3. Change Cipher Spec / Finished – Packets 22646 – 22648 transition to encrypted communication.
4. Application Data exchange – Packets 22649 through 22668 carry encrypted HTTPS payloads representing the PDF download request and the server's response data.

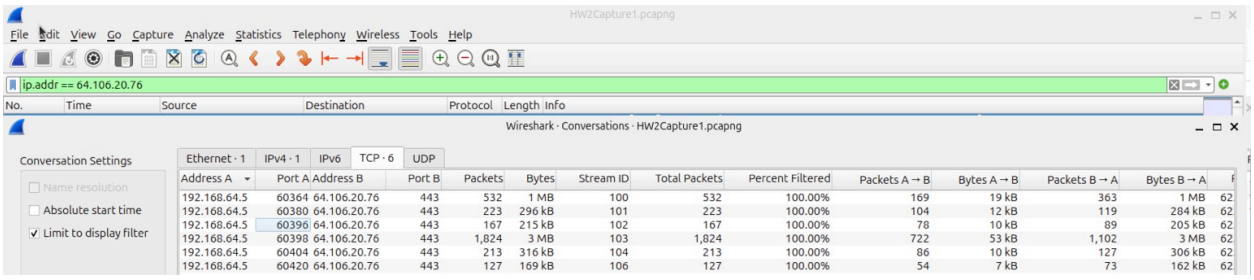
5(g) Protocols: TCP, TLSv1.3 Ports: 57258 (client ephemeral port), 443 (HTTPS server port)



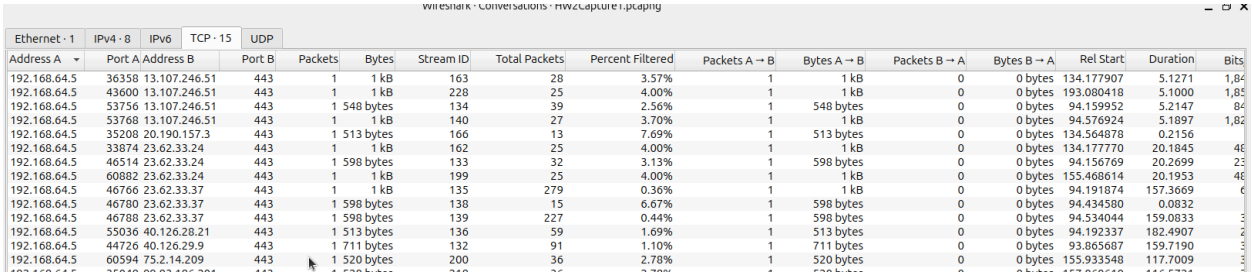
6(a) 17 + 38 + 31 = 86 packets



6(b)
34+32+39+26+24+22+22+26+45+26+26+94+92+24+39+66+67+375+22+50+553+18+56+20+22+25+24+24+23+21+23+22+26+24+90+45+79+33+53+410+25+64 = 2,831 packets



6(c) 532 + 223 + 167 + 1824 + 127 = 2,873 packets

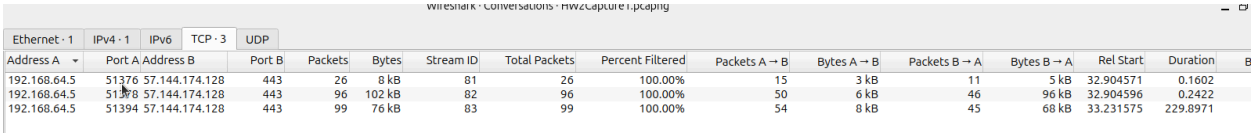


Wireshark · Conversations · HW2Capture1.pcapng

Ethernet · 1 | IPv4 · 8 | IPv6 | TCP · 15 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Total Packets	Percent Filtered	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits
192.168.64.5	36358	13.107.246.51	443	1	1 kB	163	28	3.57%	1	1 kB	0	0 bytes	134.177907	5.1271	1,84
192.168.64.5	43600	13.107.246.51	443	1	1 kB	228	25	4.00%	1	1 kB	0	0 bytes	193.080418	5.1000	1,85
192.168.64.5	53756	13.107.246.51	443	1	548 bytes	134	39	2.56%	1	548 bytes	0	0 bytes	94.159952	5.2147	84
192.168.64.5	53768	13.107.246.51	443	1	1 kB	140	27	3.70%	1	1 kB	0	0 bytes	94.576924	5.1897	1,82
192.168.64.5	35208	20.190.157.3	443	1	513 bytes	166	13	7.69%	1	513 bytes	0	0 bytes	134.564878	0.2156	
192.168.64.5	33874	23.62.33.24	443	1	1 kB	162	25	4.00%	1	1 kB	0	0 bytes	134.177770	20.1845	48
192.168.64.5	46514	23.62.33.24	443	1	598 bytes	133	32	3.13%	1	598 bytes	0	0 bytes	94.156769	20.2699	23
192.168.64.5	60882	23.62.33.24	443	1	1 kB	199	25	4.00%	1	1 kB	0	0 bytes	155.468614	20.1953	48
192.168.64.5	46766	23.62.33.37	443	1	1 kB	135	279	0.36%	1	1 kB	0	0 bytes	94.191874	157.3669	6
192.168.64.5	46780	23.62.33.37	443	1	598 bytes	138	15	6.67%	1	598 bytes	0	0 bytes	94.434580	0.0832	
192.168.64.5	46788	23.62.33.37	443	1	598 bytes	139	227	0.44%	1	598 bytes	0	0 bytes	94.534044	159.0833	3
192.168.64.5	55036	40.126.28.21	443	1	513 bytes	136	59	1.69%	1	513 bytes	0	0 bytes	94.192337	182.4907	2
192.168.64.5	44726	40.126.29.9	443	1	711 bytes	132	91	1.10%	1	711 bytes	0	0 bytes	93.865687	159.7190	3
192.168.64.5	60594	75.2.14.209	443	1	520 bytes	200	36	2.78%	1	520 bytes	0	0 bytes	155.933548	117.7009	3

6(d) Count until Canvas.lms → Dest IP:75.2.14.209
28+25+39+27+13+25+32+25+279+15+227+59+91+36 = 921 packets

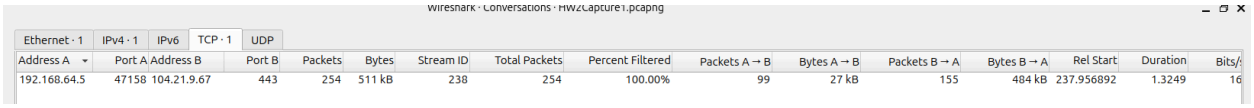


Wireshark · Conversations · HW2Capture1.pcapng

Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 3 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Total Packets	Percent Filtered	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	B
192.168.64.5	51376	57.144.174.128	443	26	8 kB	81	26	100.00%	15	3 kB	11	5 kB	32.904571	0.1602	
192.168.64.5	51378	57.144.174.128	443	96	102 kB	82	96	100.00%	50	6 kB	46	96 kB	32.904596	0.2422	
192.168.64.5	51394	57.144.174.128	443	99	76 kB	83	99	100.00%	54	8 kB	45	68 kB	33.231575	229.8971	

6(e) 26+96+99 = 221 packets

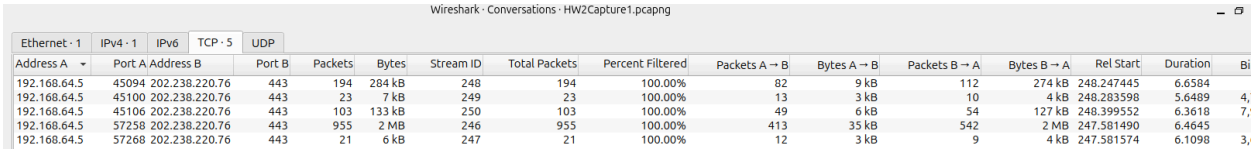


Wireshark · Conversations · HW2Capture1.pcapng

Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 1 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Total Packets	Percent Filtered	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/
192.168.64.5	47158	104.21.9.67	443	254	511 kB	238	254	100.00%	99	27 kB	155	484 kB	237.956892	1.3249	16

6(f) 254 packets

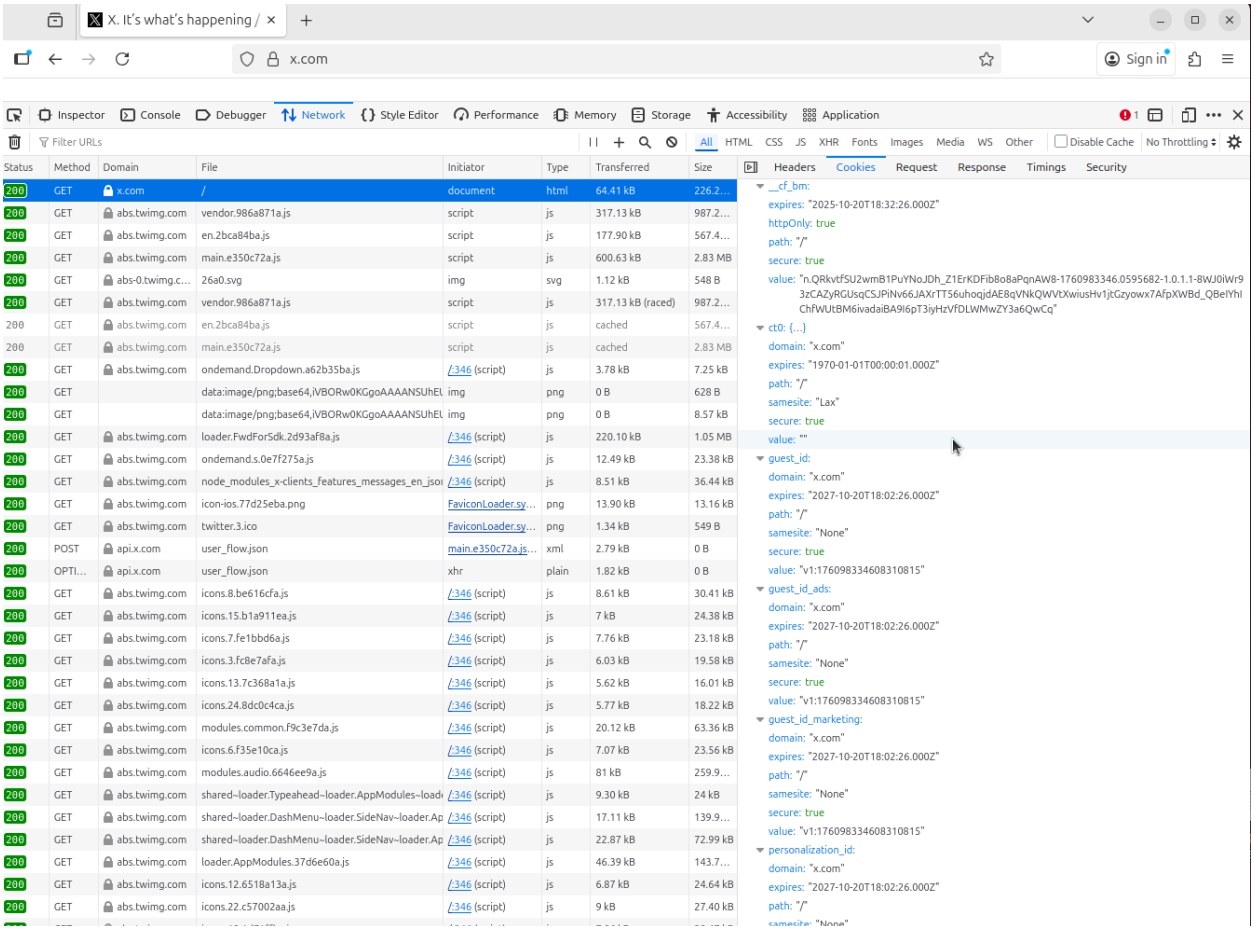


Wireshark · Conversations · HW2Capture1.pcapng

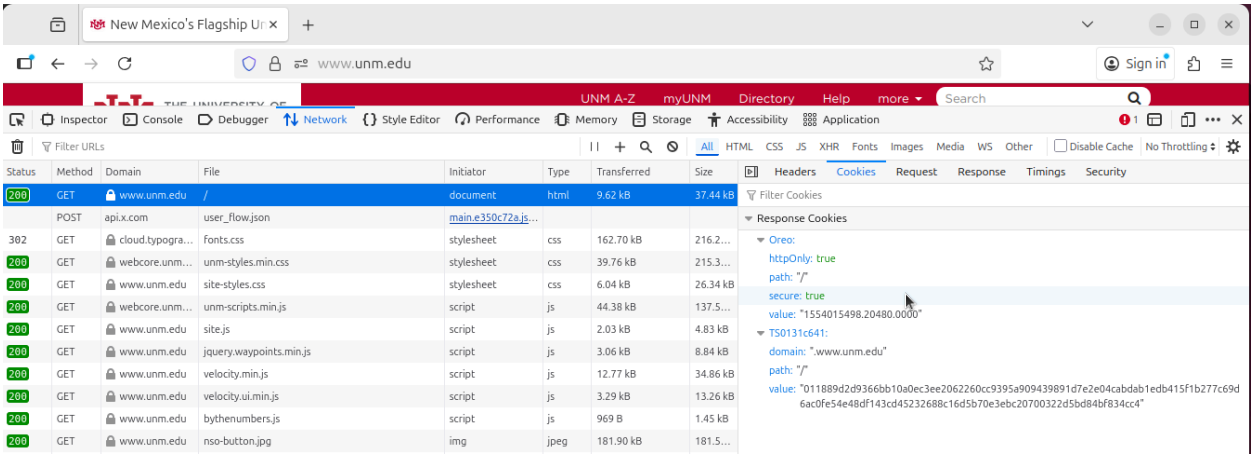
Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 5 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Total Packets	Percent Filtered	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bi
192.168.64.5	45094	202.238.220.76	443	194	284 kB	248	194	100.00%	82	9 kB	112	274 kB	248.247445	6.6584	
192.168.64.5	45100	202.238.220.76	443	23	7 kB	249	23	100.00%	13	3 kB	10	4 kB	248.283598	5.6489	4,
192.168.64.5	45106	202.238.220.76	443	103	133 kB	250	103	100.00%	49	6 kB	54	127 kB	248.399552	6.3618	7,
192.168.64.5	57258	202.238.220.76	443	955	2 MB	246	955	100.00%	413	35 kB	542	2 MB	247.581490	6.4645	
192.168.64.5	57268	202.238.220.76	443	21	6 kB	247	21	100.00%	12	3 kB	9	4 kB	247.581574	6.1098	3,

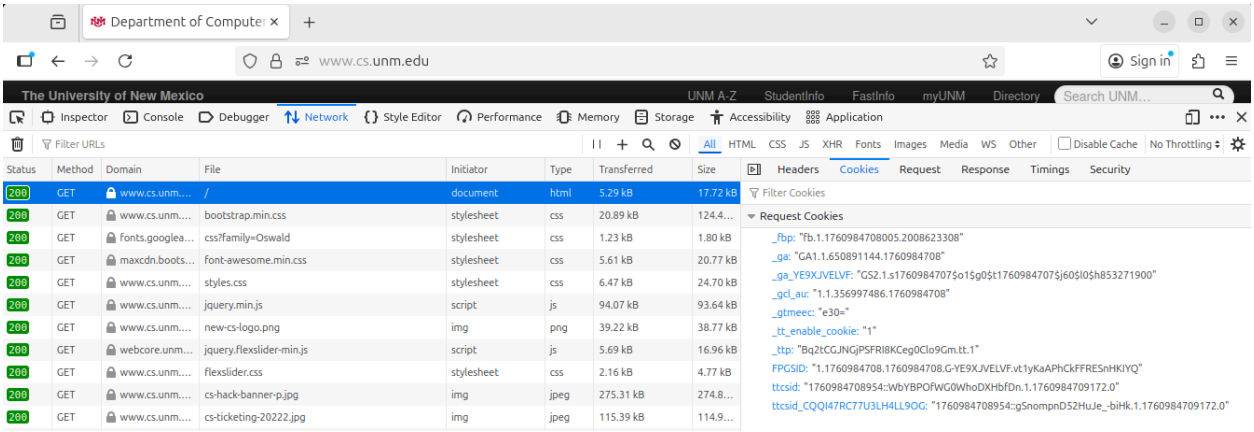
6(g) 194+23+103+955+21 = 1,296 packets



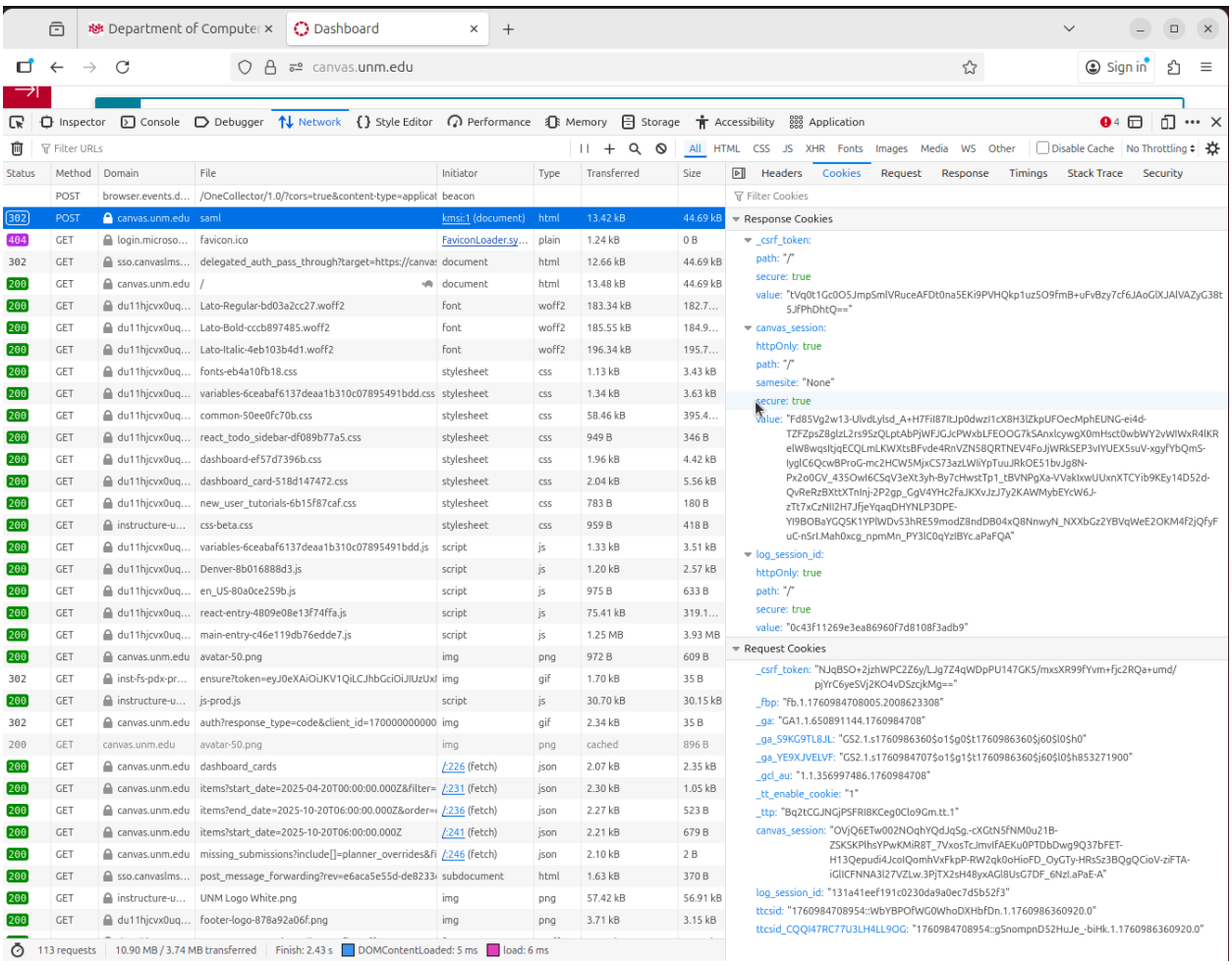
7(a) Observed response cookies:
cf_bm – Cloudflare bot-management cookie; helps detect automated traffic.
ct0 – Session token used by Twitter for login state verification.
guest_id – Unique anonymous visitor identifier.
guest_id_ads – Tracking identifier for ad services.
guest_id_marketing – Marketing tracking identifier.
personalization_id – Cookie used for content and ad personalization.



7(b) Observed response cookies: Oreo and TS0131c641.
These are standard session and load-balancing cookies used by the UNM web server to maintain user sessions and distribute requests across multiple backend servers.

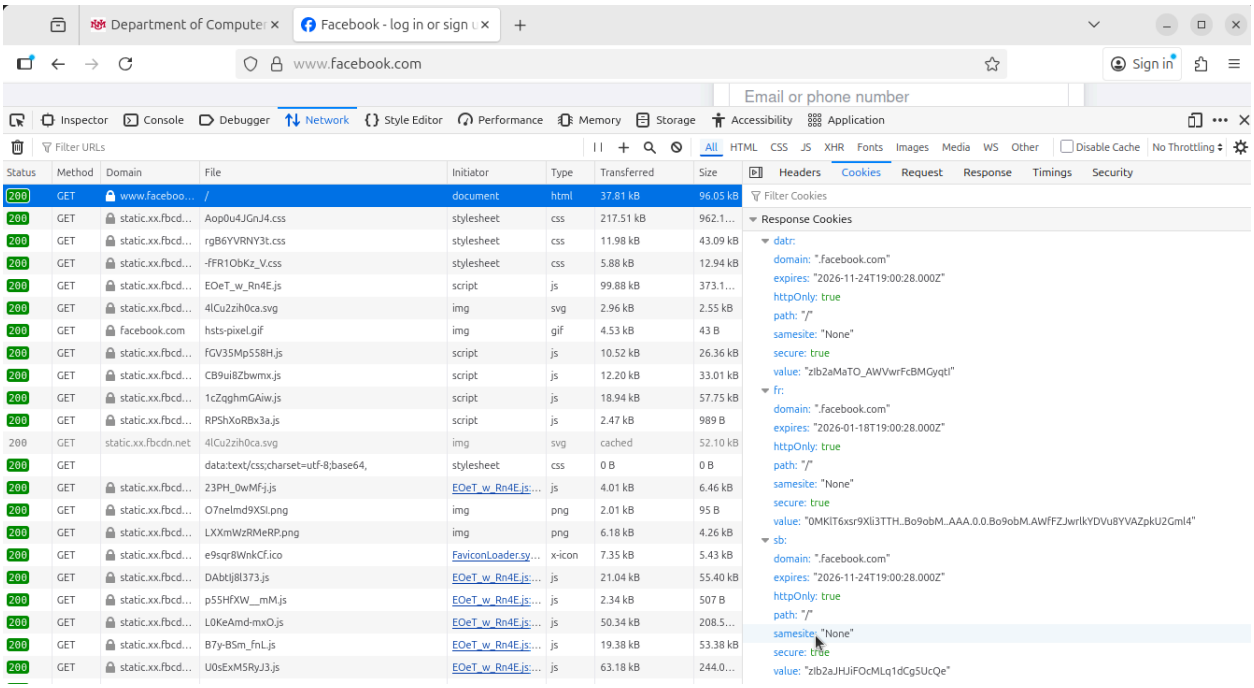


7(c) Observed request cookies: Google Analytics identifiers (_ga, _gid, _gcl_au) and Facebook or ad tracking cookies (_fbp, _tt_enable_cookie, ttcid).
These cookies are used to collect site usage data and integrate analytics and advertising features.

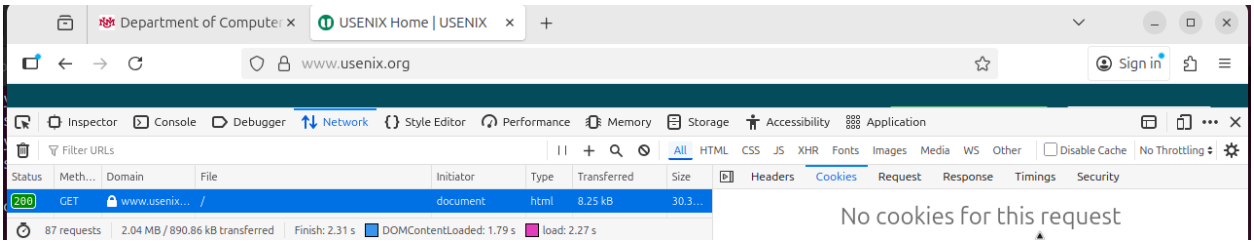


7(d) Response cookies were initially set by the Canvas application server to establish a secure session. These included _csrf_token, canvas_session, and log_session_id. The _csrf_token cookie guard against cross-site request forgery, ensuring that any subsequent POST requests made by the user are from legitimate, authenticated sessions. The canvas_session cookie creates and maintains the user's active login session; it contains an encrypted session identifier that links the user's browser to their authenticated Canvas account. The log_session_id cookie is used internally by Canvas to log session events, correlate user actions, and improve reliability and auditing within the LMS platform.

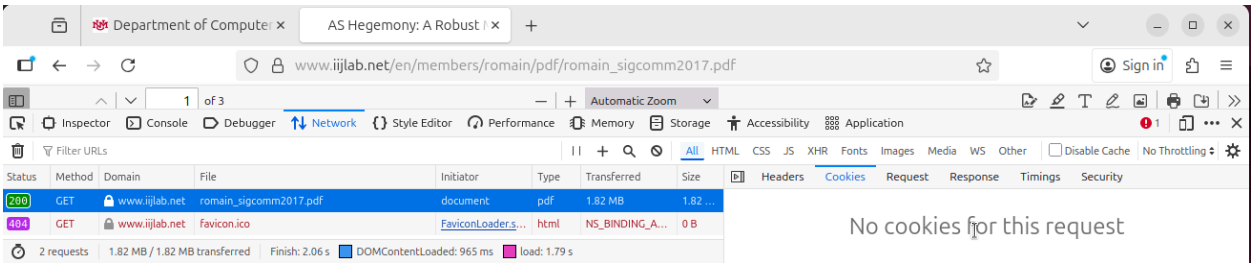
Immediately after these cookies were issued, the browser began including them in its request headers for all subsequent resource requests to Canvas, such as dashboard updates, images, and JavaScript components. The request cookies included _csrf_token, canvas_session, log_session_id, and third-party analytics identifiers like _ga, _gid, and ttcid. This is a typical example where the server sets authentication cookies and the browser returns them automatically with every new request to preserve continuity.



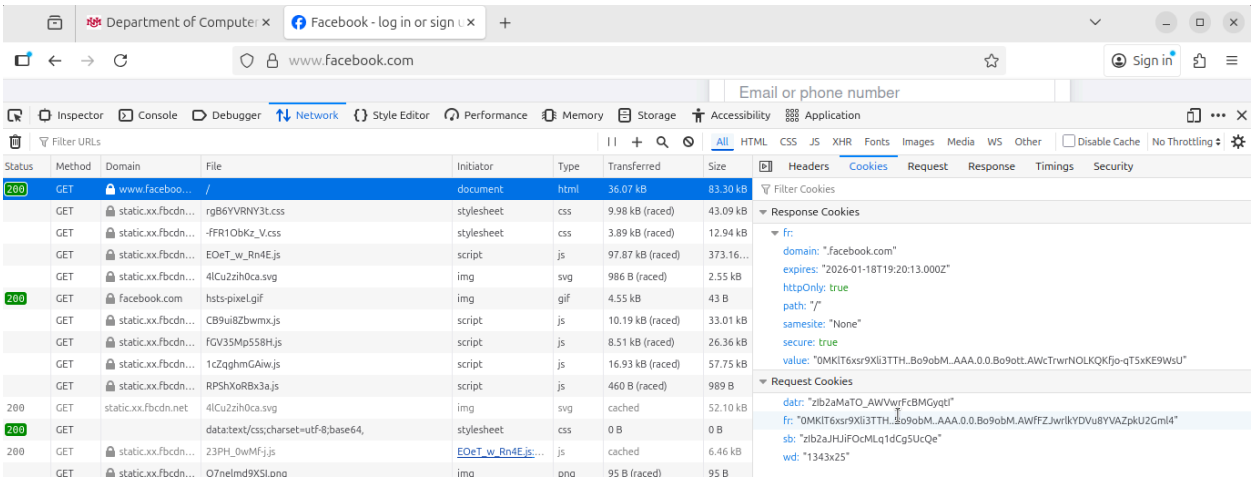
7(e) Response cookies included datr, fr, and sb. These are used to identify the browser, manage login sessions, and support security checks.



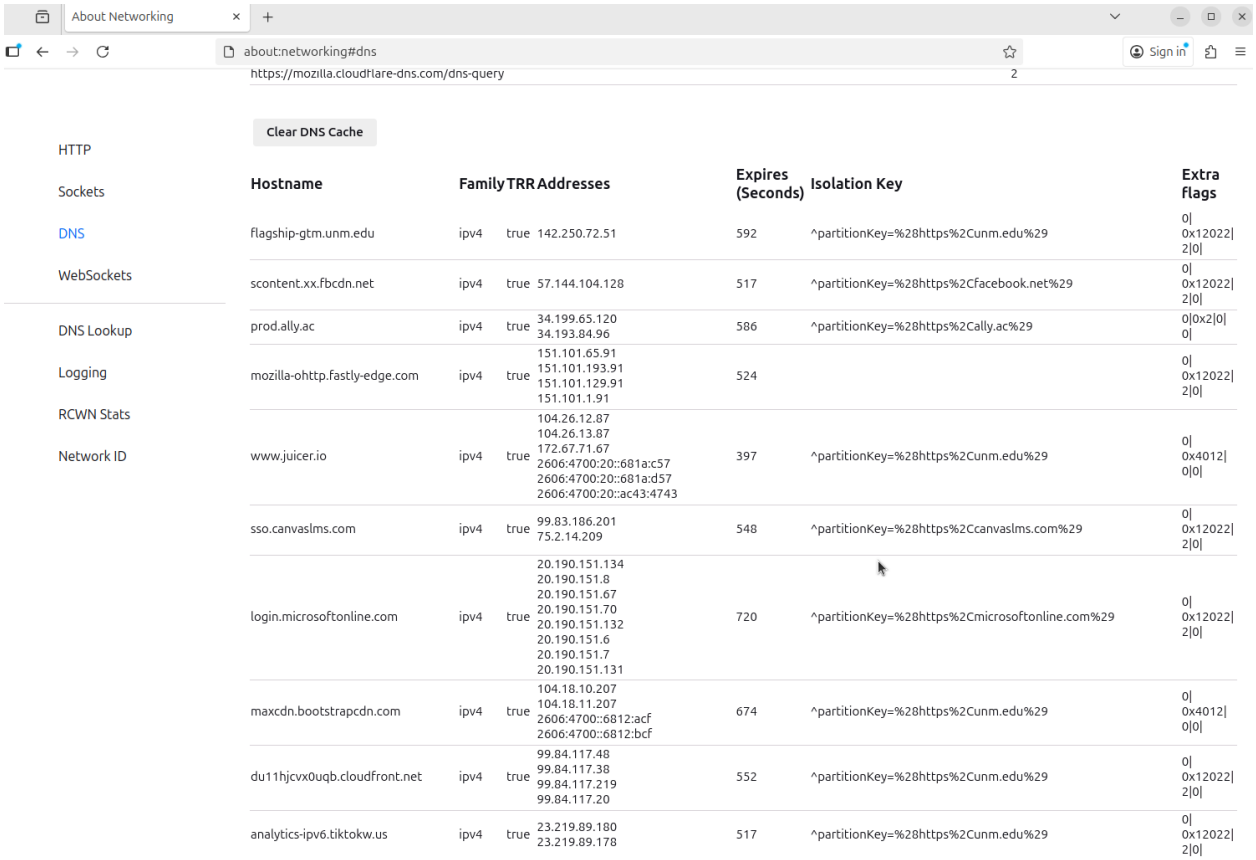
7(f) No cookies were set for this request



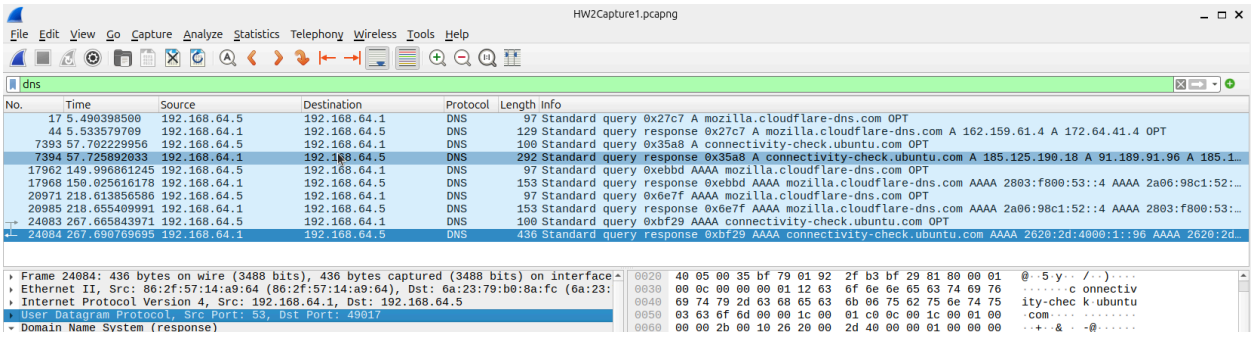
7(g) No cookies were set for this request



8(Choose e)
On this second visit to facebook.com, request cookies now appear alongside the response cookies. This happens because after the initial visit, Facebook's cookies (fr, datr, sb) were stored locally by the browser. When the site is revisited or new requests are made, the browser automatically sends these stored cookies back to the server in the HTTP request headers.

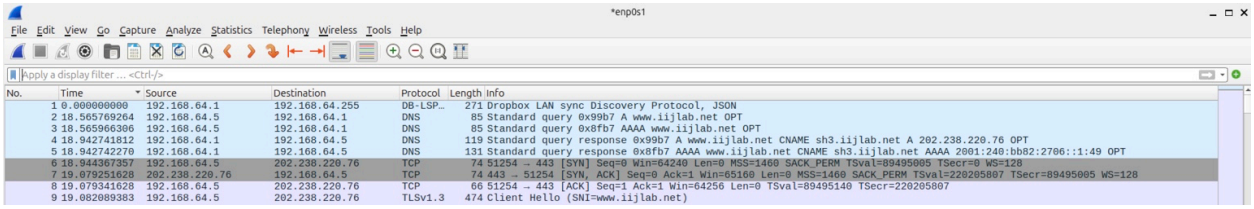


9 It's much larger than the 7-12 pages I visited.
I did not manually visit these URLs, but my browser requested them automatically while loading the main pages:
CDNs: Facebook (fbcdn.net), AWS CloudFront, BootstrapCDN.
Analytics/tracking — Juicer, TikTok analytics, UNM's Google Tag Manager variant.
Accessibility and browser services — Ally (ally.ac) and Mozilla Fastly Edge.

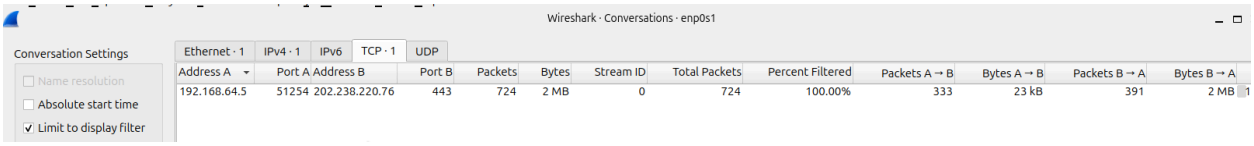


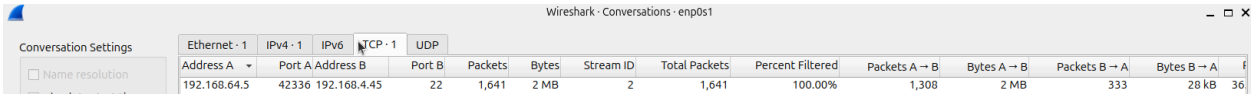
10 They all have a standard query response pair and all use port 53, but they have various lengths, and some are query type A, others AAAA

4

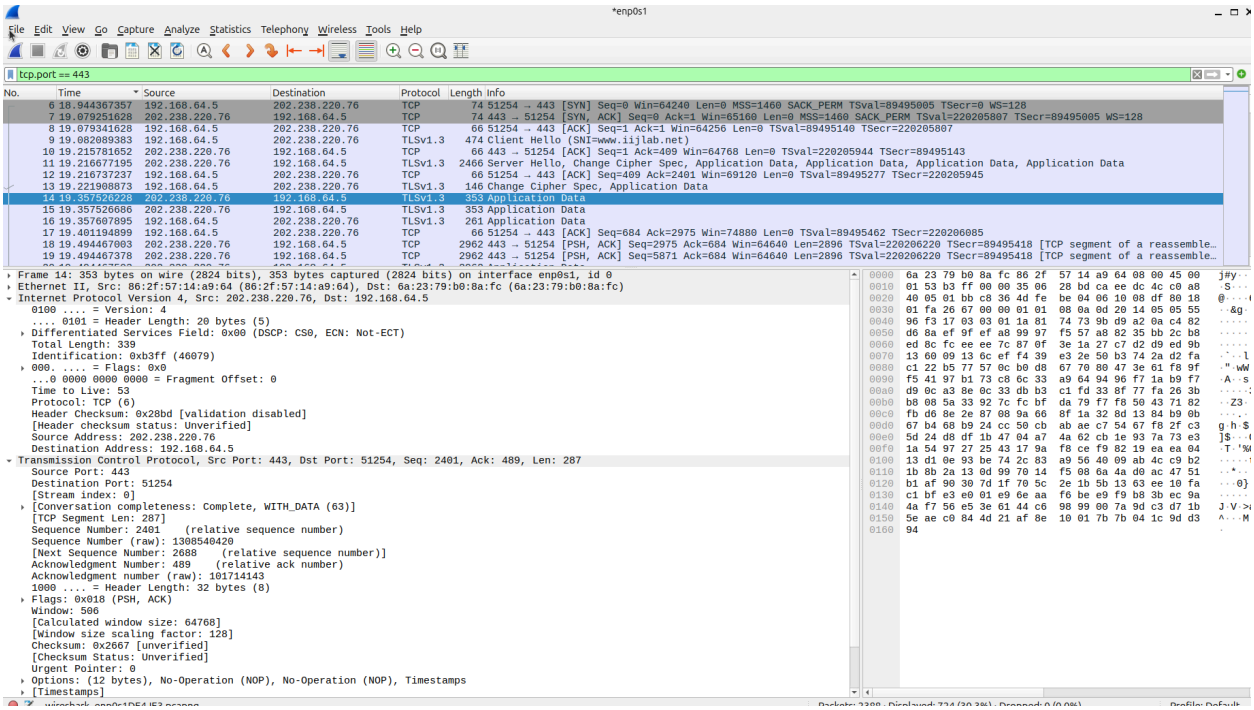


1)
In the capture, the client (192.168.64.5) first issues DNS queries to the local resolver (192.168.64.1) to resolve the hostname www.iijlab.net (frames 2–5). These are sent over UDP, using standard DNS query types: A record for IPv4 (frame 2), AAAA record for IPv6 (frame 3) The resolver responds (frames 4–5) with both IPv4 and IPv6 addresses, confirming that DNS resolution succeeded.
Immediately after DNS resolution, a TCP 3-way handshake occurs between: Source: 192.168.64.5,Destination: 202.238.220.76 and Ports: 51254 → 443
This establishes a reliable connection for HTTPS (HTTP over TLS). The packets: SYN → client initiate connection (frame 6), SYN-ACK → server acknowledgment (frame 7) ACK → client confirmation (frame 8). Following the handshake, we see the TLS Client Hello (frame 9), which begins the encrypted HTTPS session to request the PDF file (romain_sigcomm2017.pdf).





2) Download B->A = 391 packets
Upload A->B = 1,308 packets



3)
Source Port
443 (HTTPS server port)
Destination Port
51254 (ephemeral client port)
Sequence Number
2401 — identifies the first byte of data in this segment
Acknowledgment Number
489 — acknowledges receipt of all bytes up to 488 from the client
Header Length
32 bytes (indicates TCP header size)
Flags
PSH, ACK (0x18) — data being pushed to the receiver and acknowledged
Window Size Value
64,768 — receiver buffer size before scaling
Window Scaling Factor
128 — actual window = 64,768 × 128 = 8,388,608 bytes
Checksum
0x26C7 — used for error detection of header + payload
Urgent Pointer
0 (not used)
TCP Options
Timestamps (TSval = 220206220, TSecr = 89495418) used for RTT measurement

4) Checksum = 0x26c7 [unverified].

5) The checksum is calculated by performing a 1’s-complement sum over a the source IP, destination IP, protocol, segment length plus the entire TCP header and data, then taking the 1’s-complement of the result. When the receiver recomputes the checksum, any mismatch indicates that bits were altered in transit.

5

Redirection: The RFC talks about “3xx Redirection” codes and how they tell the client to go to a different URI. It includes things like 301 Moved Permanently and 302 Found. The idea is that when a page changes its address, the browser automatically finds the new one instead of showing an error.This is necessary because the web changes all the time, and redirection keeps links from breaking when sites move things around.

Client Error: The “4xx Client Error” section covers problems caused by the request itself, like 400 Bad Request, 403 Forbidden, and 404 Not Found. These tell you that something about your input or permissions is wrong. The RFC says the client shouldn’t repeat the request without fixing it.
This is necessary because it tells users when the problem is on their end instead of wasting the server’s time or leaving them guessing why it failed.

Server Error: The “5xx Server Error” codes mean the request was fine but the server couldn’t complete it. Examples are 500 Internal Server Error and 503 Service Unavailable. It basically tells the client, “we understand what you wanted, but something broke here.”
This is necessary because it separates server failures from client mistakes and helps developers figure out what’s actually wrong in a networked system.

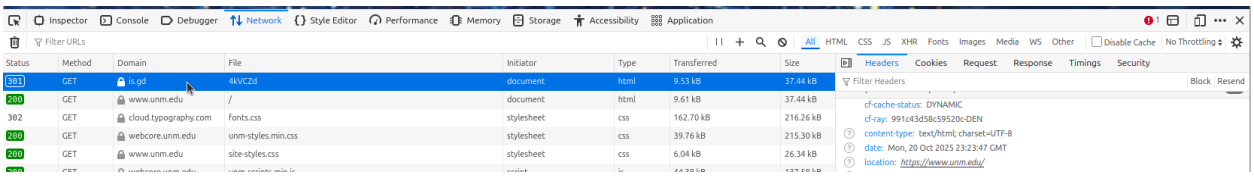
Caching: Caching is about saving responses so they don’t have to be downloaded again. The RFC talks about Cache-Control, ETag, and validation headers like Last-Modified. It’s all about saving time and bandwidth by letting browsers reuse what they already have.
This is necessary because it makes the web faster and reduces load on servers instead of re-downloading the same content constantly.

Expiration: This is how servers tell caches when stored data becomes “stale.” The RFC mentions headers like Expires and max-age to control that. It uses caching to manage freshness. This is necessary because it keeps information up to date without making clients check the server every second.

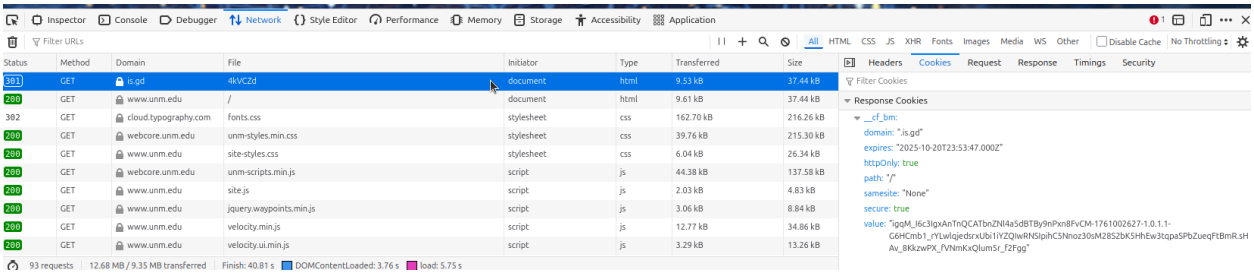
Security Considerations: This section lists HTTP risks, like caching private data, using unencrypted connections, or trusting headers too much. It suggests using TLS (HTTPS) and validating inputs. This is necessary because HTTP by itself isn’t secure, so these guidelines help prevent people from intercepting, changing, or stealing data during communication.

6

1,2,3,4,5
https://www.unm.edu/ : https://is.gd/4kVCZd, tinyurl.com/95rch3ce
https://www.usenix.org/ : https://is.gd/T9Pugg, tinyurl.com/4eh35n9d



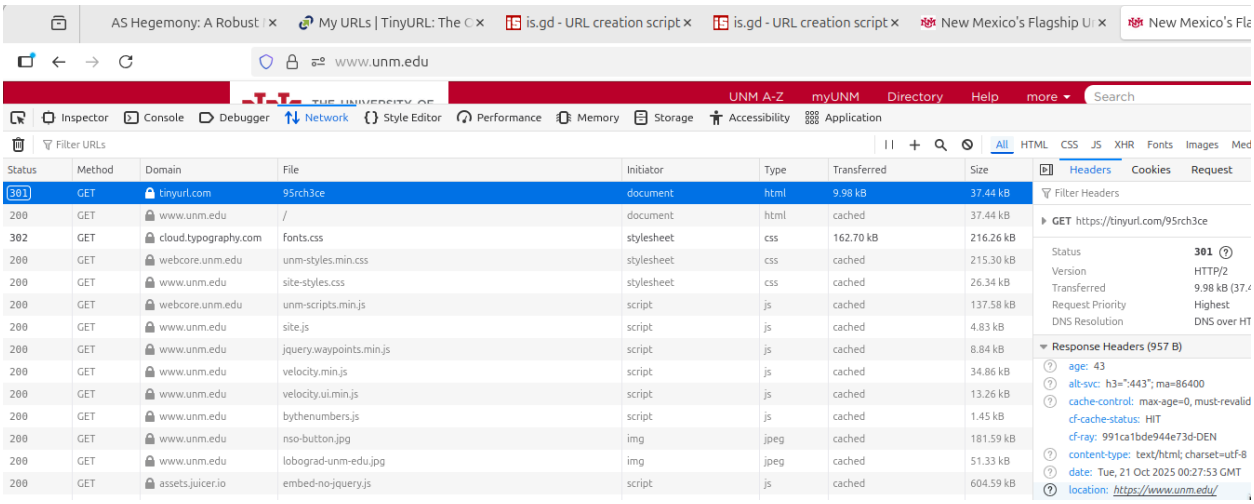
6(a) https://is.gd/4kVCZd, Status 301, Location https://www.unm.edu/
https://www.unm.edu/ Status 200, Location reached



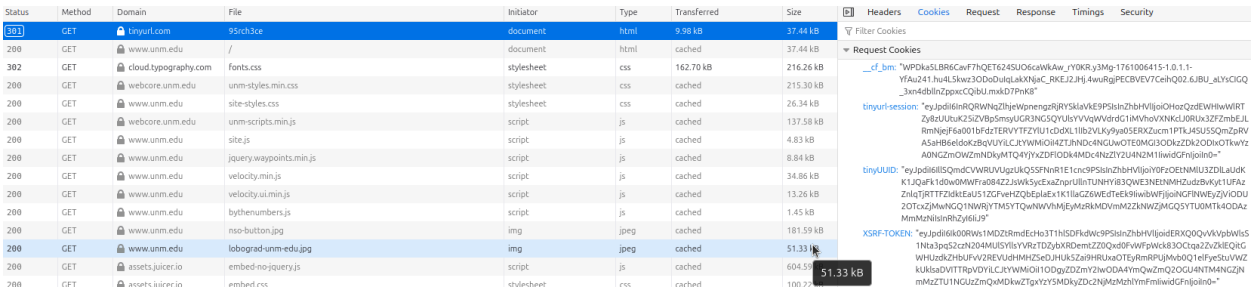
7(a) Just the shortener

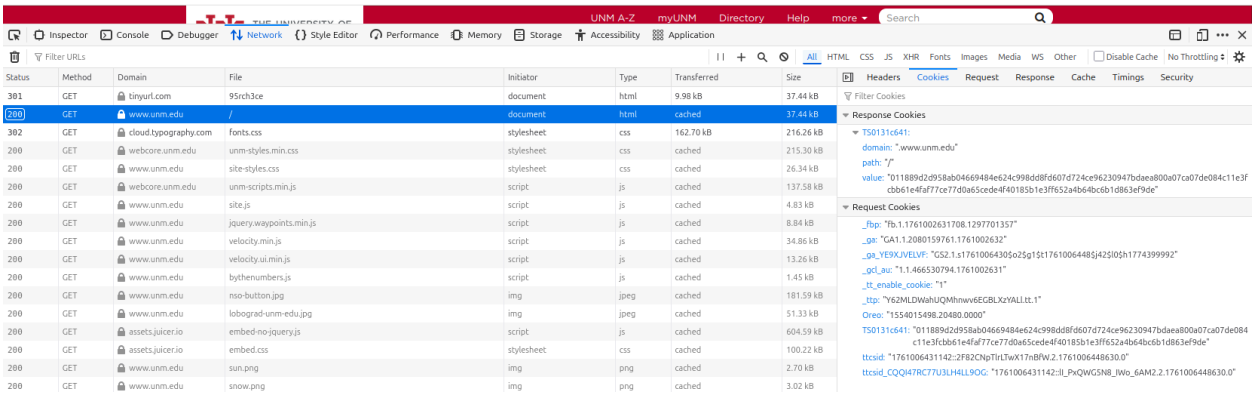
8(a) When I visited the short link https://is.gd/4kVCZd, the shortener domain (is.gd) set a cookie. The Set-Cookie header appeared in the response from the shortener before it redirected me to https://www.unm.edu/. The cookie from is.gd was not used by UNM's server, it just helped the shortener track that the redirect was successfully followed. It lets the shortener measure link usage and prevent abuse without affecting the target website.

9(a) The first response from is.gd had Status 301 (Moved Permanently) with a Location header pointing to https://www.unm.edu/. That tells the browser that the short link has a permanent destination, so it should automatically follow that new URL and possibly cache it for future use. The second response from www.unm.edu was Status 200 (OK), which means the request succeeded and the actual page content was delivered.The Location header specifies exactly where the browser should go next. Without it, the 301 would just tell the browser "this isn't here" with no destination.



6(b) tinyurl.com/95rch3ce Status 301, Location https://www.unm.edu/
https://www.unm.edu/ Status 200, Location reached

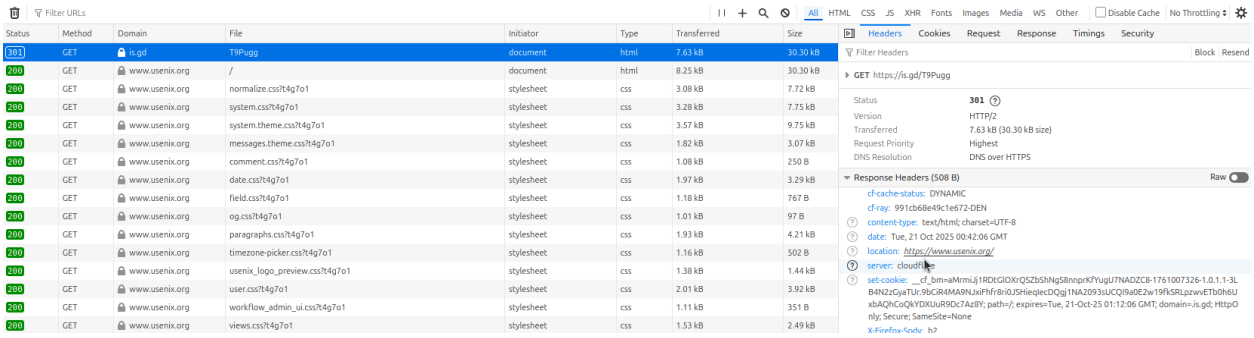




7(b)The Network tab shows that tinyurl.com sent a Set-Cookie header before redirecting to https://www.unm.edu/. UNM issued some new cookies but they weren't related to TinyURL, they were from accessing the site a second time.

8(b) When I accessed https://tinyurl.com/95rch3ce, the browser received a response from the TinyURL server with a Set-Cookie: header. The cookies here seem to contain encoded session or tracking information, probably used by TinyURL to count link clicks and monitor redirect performance.

9(b) The redirect chain had two main responses: 301 (Moved Permanently) from tinyurl.com with a Location header pointing to https://www.unm.edu/ and 200 (OK) from the final UNM site, meaning the content was successfully retrieved. The 301 code tells the browser to automatically go to the new URL and treat it as the permanent destination for that short link. The Location header provides the exact address of the target page.



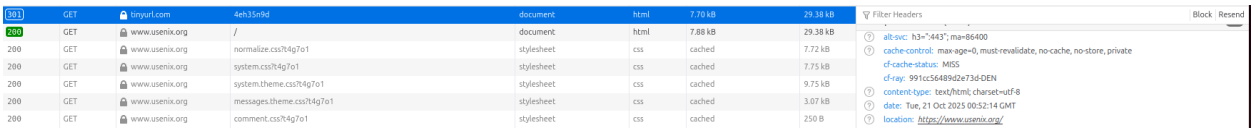
Note Status, Location and cookie are all present here. And we saw before that usenix itself doesn't have cookies.

6(c) https://is.gd/T9Pugg, Status 301, Location https://www.usenix.org/
https://www.usenix.org/ Status 200, Location reached

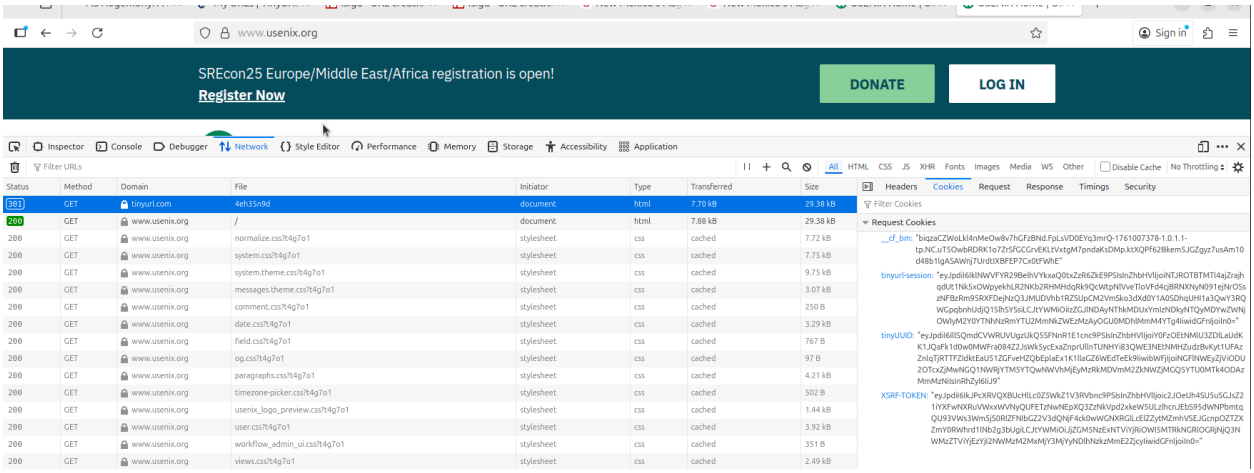
7(c) Just the shortener

8(c) When I visited the short link https://is.gd/T9Pugg, the shortener domain (is.gd) set a cookie. The Set-Cookie header appeared in the response from the shortener before it redirected me to https://www.usenix.org/. The cookie from is.gd was not used by usenix's server, it just helped the shortener track that the redirect was successfully followed. It lets the shortener measure link usage and prevent abuse without affecting the target website.

9(c) The first response from is.gd had Status 301 (Moved Permanently) with a Location header pointing to https://www.usenix.org/. That tells the browser that the short link has a permanent destination, so it should automatically follow that new URL and possibly cache it for future use. The second response from www.unm.edu was Status 200 (OK), which means the request succeeded and the actual page content was delivered.The Location header specifies exactly where the browser should go next. Without it, the 301 would just tell the browser “this isn’t here” with no destination.



6(d) tinyurl.com/4eh35n9d Status 301, Location https://www.usenix.org/
https://www.usenix.org/ Status 200, Location reached



And once again, we saw earlier that usenix itself doesnt have cookies.

7(d) The Network tab shows that tinyurl.com sent a Set-Cookie header before redirecting to https://www.usenix.org/. We saw earlier that usenix itself doesnt have cookies.

8(d) When I accessed tinyurl.com/4eh35n9d, the browser received a response from the TinyURL server with a Set-Cookie: header. The cookies here seem to contain encoded session or tracking information, probably used by TinyURL to count link clicks and monitor redirect performance.

9(b) The redirect chain had two main responses: 301 (Moved Permanently) from tinyurl.com with a Location header pointing to https://www.usenix.org/and 200 (OK) from the final usenix site, meaning the content was successfully retrieved. The 301 code tells the browser to automatically go to the new URL and treat it as the permanent destination for that short link. The Location header provides the exact address of the target page.