2/4

Bien

2- Llamemos:

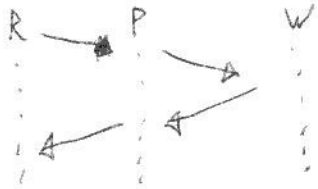
R: Browser de Ryan

J: Browser de Juliano

P: Proxy

W: Servidor de onlyonepicture

a) La siguiente conversación se da entre los dispositivos:



R pide index.html a P

P pide index.html a W

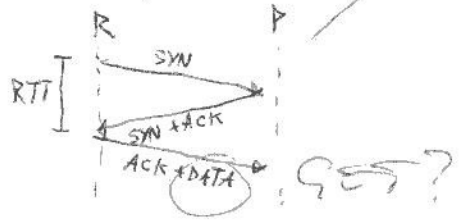
W responde a P con index

P responde a R con index

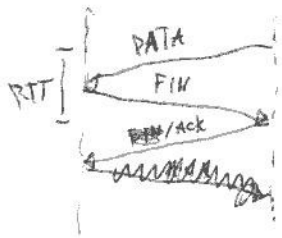
R lee el contenido de la página y apenas encuentra la imagen realiza la misma conversación de recién, pero pidiendo esta vez la imagen.

Veamos cuanto tiempo toma esto:

Cuando R pide a P o P pide a W, debe abrir una conección TCP y mandar el request. Esto toma 1.5 RTTs si tomamos T_{tx} como despreciable.



Cuando W responde a P o P responde a R, se envían los datos (que entran en un segmento TCP) y se cierra la conección. Esto toma 1/2 RTT hasta obtener los datos y luego 1 RTT para cerrar la conexión.



Finalmente, vemos que tardan 1.5 + 1.5 + 0.5 + 0.5 = 4 RTTs hasta que W recibe el HTML y luego 1.5 + 1.5 + 0.5 + 0.5 + 1 = 5 RTTs hasta que recibe la imagen y se cierran todas las conexiones.

Tiempo total: 9 RTTs ✓

b) ✓ pide el html a P:

GET /index.html HTTP/1.1 Host: <hostname>

- P chequea si el recurso se modificó en W ✓

GET /index.html HTTP/1.1

Host: <hostname>

IF-modified-after: <date> ✓

- W responde que no se modificó

GET index.html HTTP/1.1 ~~200 OK~~ NO ESTARIA

3?? Not Modified

- P responde al J con la ~~el~~ HTML cacheado

GET /index.html HTTP/1.1

200 OK

Mime-type: Text/Html

<newline> ✓

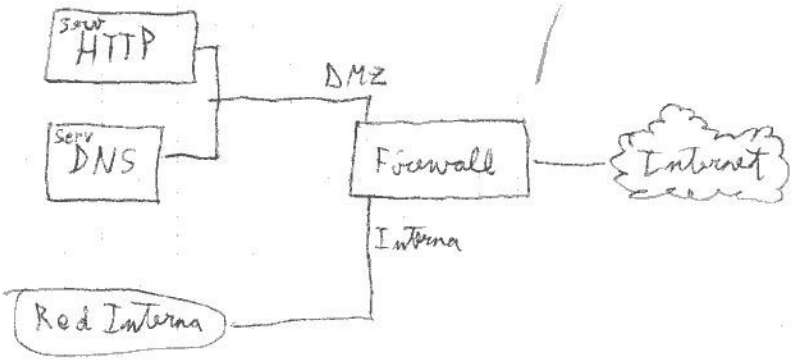
<contenido html>

Luego se vuelve a realizar la misma conversación, pero pidiendo el recurso de la imágen en vez de /index.html; por lo que se debuelve el contenido correspondiente con un Mime-type apropiado. ✓

3/4

B-

3) a. La red se organiza de la siguiente manera:



El firewall cuenta con las siguientes reglas, y una política por defecto DROP.

<interna, *, Serv HTTP, HTTP, ALLOW>
<interna, *, Serv DNS, DNS, ALLOW>
<interna, *, internet, HTTP, ALLOW>
<interna, *, ~~internet~~ Server DNS, HTTP, ALLOW>
<Serv DNS, *, internet, DNS, ALLOW>
<internet, *, Serv HTTP, HTTP, ALLOW>
<internet, *, Serv DNS, DNS, ALLOW>

no filtra.

¿puerto?

filtra indica TCP/UDP

b. La nueva api recibe los siguientes parámetros get:

method: getPriceSecure

Fruit: <Fruit>

user: <user>

digest: <digest>

donde <digest> es el resultado de correr una función de hash predefinida sobre la querystring ~~con~~ sin el digest y el secreto compartido concatenado al final.

El servidor debe computar el hash de la misma forma usando su secreto compartido y comparalo con el valor del digest para verificar la integridad.

B-

4/4

4) 2)

RTT	SSTHRESH	RWND	LBS	Flight Size	CWND	
1	64 kB	64 kB	4 kB	4 kB	4 kB ✓	CWND = IW
2	"	60 KB	12 kB	8 kB	8 kB ✓	
3	"	52 kB	28 kB	16 kB	16 kB ✓	
4	"	36 kB	60 kB	32 kB	32 kB ✓	
5	"	4 kB	64 kB	4 kB	64 kB ✓	
6	"	0 kB	64 kB	0 kB	65 kB ✓	CWND > SSTHRESH
7	"	0 kB	64 kB	0 kB	65 kB ✓	
8	"	0 kB	64 kB	0 kB	4 kB ✓	Reinicio por tiempo Idle
9	64 kB	64 kB	68 kB ✓	4 kB ✓	4 kB ✓	
10	64 kB	64 kB	76 kB ✓	8 kB ✓	8 kB ✓	
11	64 kB	64 kB	80 kB ✓	4 kB ✓	16 kB ✓	Se envía lo que queda

b) Como pasó más de un RTO se reinicia CWND. RWND se mantiene constante en 64 kB! La siguiente tabla muestra el comportamiento de las variables:

RTT	CWND	SS THRESH	LBS	Flight Size	
1	4kB	64kB	4kB	4kB	✓
2	8kB	64kB	12kB	8kB	✓
3	16kB	64kB	28kB	16kB	✓ Se caen todos los paquetes
Hay otro RTT esperando los acks → 4	2kB	8kB	14kB	2kB	✓ ~~Se caen todos~~
5	4kB	8kB	18kB	4kB	✓
6	8kB	8kB	26kB	8kB	✓
7	~~9kB~~ 10K	8kB	35kB	9kB	CWND ≥ SSTHRESH ✓
8	10kB 12K	8kB	45kB	10kB	
9	11kB	8kB	56kB	11kB	
10	12kB	8kB	60kB	60kB	Se envía lo que resta

CWND += 1 SMSS

En total tardó 10 RTTs = 2s