

# Relazione 4

## Analisi del traffico di Telegram web

Mattia Chiarle 269969, Michele Ferrero 268542, Gabriele Ferro 268510

Gruppo 22 Anno 2021/2022



## Sezione 1 - Descrizione

L'applicazione analizzata è Telegram web, un servizio di interfaccia browser per l'applicazione di messaggistica istantanea Telegram. Il servizio permette di inviare messaggi ai propri contatti, tramite l'utilizzo della rete internet, salvandoli sul cloud proprietario in modo da avere una maggiore portabilità delle chat. In aggiunta al suo scopo base è possibile inviare messaggi vocali, effettuare videochiamate, chiamate vocali e condividere file multimediali di qualsiasi formato. È inoltre possibile creare gruppi di conversazione, permettendo l'interazione tra più utenti contemporaneamente, la creazione di canali di comunicazione, utilizzati per l'invio unidirezionale di messaggi al gruppo di persone che vi partecipa, e di creare e utilizzare bot per lo svolgimento di funzioni automatiche di ogni tipo.

Il principale vantaggio del servizio, che ne ha aumentato anche la diffusione nel periodo del lancio, è l'essere completamente open source. Questo comporta una maggiore fiducia da parte degli utenti, i quali possono visualizzare liberamente il codice sorgente dell'applicazione, oltre che una maggiore prospettiva di sviluppo di nuove funzionalità e una più rapida risoluzione dei problemi grazie all'ampia interazione della comunità di sviluppo.

Durante l'analisi verrà analizzato il comportamento dell'applicazione durante le operazioni di invio di messaggi di testo, immagini, messaggi vocali, file di grandi dimensioni, durante le chiamate e videochiamate e l'eventuale gestione dell'interruzione della connessione di rete in queste situazioni.

## Sezione 2 - Descrizione del testbed

Il test è stato effettuato con questo setup:

- Host A:
  - OS: Windows 11
  - Browser: Google Chrome
  - IP: 192.168.17.61
- Host B:
  - OS: MacOS 12.4
  - Browser: Google Chrome
  - IP: 192.168.17.114, 192.168.68.127

Telegram attualmente rende disponibili tre versioni della propria versione web: la versione K, la versione Z e quella precedente. L'esistenza delle due versioni è motivata da una questione di competizione interna: si pensa infatti che avere un team parallelo che lavora su un progetto molto simile possa spronare gli sviluppatori a fare meglio. Tutte le considerazioni presenti nella relazione saranno riferite alla versione K (ovvero quella che è stata caricata di default).

I software o servizi utilizzati sono stati:

- Wireshark per l'analisi dei protocolli utilizzati dalla applicazione
- Inspector di Google Chrome per l'analisi dei pacchetti
- <https://whois.domaintools.com/> per individuare informazioni aggiuntive sugli IP
- <https://www.iplocation.net/> per localizzare gli IP con maggiore precisione

## Sezione 3 - Analisi del traffico dati

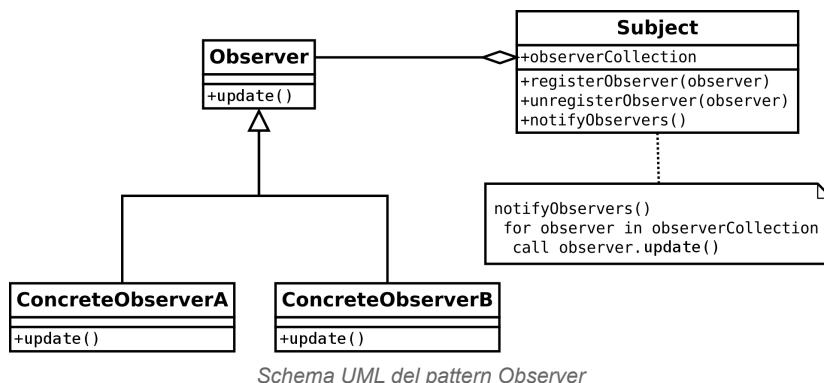
### 3.1 Analisi generale

La pila protocollare adottata da Telegram web risulta leggermente modificata rispetto a quella studiata durante le lezioni. Fino allo strato 4 la struttura risulta invariata; cambia tuttavia lo strato 7. Infatti, essendo un'applicazione web ci si aspetterebbe un traffico https (ovvero uno scambio di pacchetti TLS), mentre Telegram preferisce sfruttare il protocollo

- Layer 7 (Application): **High-level RPC API**
- Layer 6 (Presentation): **Type Language**
- Layer 5 (Session): **MTProto session**
- Layer 4 (Transport):
  - 4.3: **MTProto transport protocol**
  - 4.2: **MTProto obfuscation (optional)**
  - 4.1: **Transport protocol**
- Layer 3 (Network): IP
- Layer 2 (Data link): MAC/LLC
- Layer 1 (Physical): IEEE 802.3, IEEE 802.11, etc...

automatizzata la procedura di ricezione dei messaggi: al caricamento del sito verrà stabilita una connessione permanente con un server API che svolgerà la funzione di notificatore. Questa connessione viene inoltre sfruttata per tutte le funzionalità che non richiedono l'invio o la ricezione di molti dati: abilita la ricezione dei messaggi in tempo reale, lo scambio di messaggi per capire se la connessione è ancora attiva o ci sono problemi di rete e infine il download dei file multimediali di piccole dimensioni ricevuti.

Nel corretto funzionamento il client e il server concordano su un timeout  $T$ : se non ci sono aggiornamenti durante  $T$  (nuovi messaggi,...) allo scadere del timer interno il client contatta il server per capire se la connessione è ancora attiva o se ci sono problemi di rete. Nel caso in cui invece ci sia un aggiornamento il server pubblica un aggiornamento che verrà scaricato da tutti i client interessati. Una volta terminato lo scambio di dati il timer viene ripristinato al valore  $T$ . Questo risolve molti problemi: al posto di avere server molto potenti e costosi che, ogni volta che viene inviato un messaggio, cercano tutti i partecipanti a quella conversazione (e tutti i loro dispositivi) per comunicargli personalmente la presenza di un nuovo messaggio si tende ad alleggerire il carico di lavoro grazie alle richieste da parte dei client ai server API, i quali forniranno i vari aggiornamenti. Si sfrutta quindi il design pattern dell'observer: i client si iscrivono come observer per gli aggiornamenti di tutte le loro conversazioni (identificate da codici numerici per semplificare la creazione di un database) e quando viene mandato un messaggio i server (che funzionano come subject) creano una notifica che, grazie ai server API, viene automaticamente scaricata da tutti gli observer. I vari modi di usare i server API verranno analizzati in seguito nei vari test.



Schema UML del pattern Observer

Tutti i messaggi sono criptati grazie a un apposito layer di MTProto; benché non si tratti di una crittografia end-to-end (in quanto renderebbe molto più complicato garantire la possibilità di poter usare Telegram su più dispositivi mantenendoli sincronizzati e di poter avere un backup in tempo reale di tutte le conversazioni sul cloud) dovrebbe essere un protocollo molto sicuro. La crittografia end-to-end è abilitata solo per chiamate, videochiamate e chat segrete (che sono tutte funzionalità eseguite solo da un dispositivo per volta indirizzate verso un solo altro dispositivo).

### 3.2 Caricamento dell'applicazione web

Quando il sito viene caricato inizialmente viene mostrato un QR code. Questo viene generato automaticamente sfruttando dei metodi forniti direttamente da Telegram, e valgono solo per un certo tempo (il QR code e la validità sono restituiti dai metodi). È quindi necessario autenticarsi inquadrando il QR code con un dispositivo già connesso. Questo processo ha una duplice funzionalità: oltre a permettere un'autenticazione lato client, riducendo la possibilità di rubare un account, passando da un dispositivo già connesso la nuova applicazione verrà automaticamente aggiunta alla sessione dell'utente. Telegram infatti non ragiona a livello di applicazione ma di sessione: quest'ultima include tutte le applicazioni su cui mi sono autenticato. Questa scelta semplifica ovviamente molto la gestione del multidispositivo.

Si può poi identificare una determinata applicazione sfruttando la sessione e il key identifier. Questo risulta molto utile ad esempio per abilitare le chat segrete: poiché sono criptate end-to-end queste non saranno disponibili contemporaneamente su tutti i dispositivi ma soltanto su quello da cui è stata avviata e/o accettata, ed è quindi necessario identificare univocamente uno specifico dispositivo all'interno della sessione.

Una volta effettuato il login viene mostrata la schermata classica, contenente tutte le conversazioni. Analizzando i file scaricati si trova il file html, dei file CSS (per gestire l'estetica del sito), degli script (per gestire la parte dinamica del sito), dei font personalizzati di Telegram e delle immagini. In particolare, queste ultime includono sia delle immagini che servono a Telegram per funzionare (come la favicon o il pattern dello sfondo) sia delle immagini legate alle nostre conversazioni (le immagini del profilo e le anteprime di eventuali immagini che abbiamo ricevuto). Inoltre, analizzando più nel dettaglio l'inspector di Chrome si nota come Telegram salvi in locale dei database che gli servono per il suo corretto funzionamento. Sono in particolare memorizzate le chat attive (utili per comunicare al server API di quali conversazioni siamo listener), i set degli sticker usati e gli utenti memorizzati tra i contatti o con cui abbiamo una chat o un gruppo in comune. Inoltre, è presente la sezione "session" in cui sono salvati dei dati di configurazione utili a Telegram per funzionare correttamente. Come si può vedere nell'immagine, sono memorizzati ad esempio il numero di build, la lista dei contatti, le eventuali bozze dei messaggi, le ricerche recenti, le emoji recentemente inviate e tutte le impostazioni attualmente configurate.

Applicazione	C	Parti dalla chiave	X
Manifest	#	Chiave	
Service Workers	0	"allDialogsLoaded"	↳ {0: true, 1: true, undefined: true}
Spazio di archiviazione	1	"authState"	↳ {_: 'authStateSignedIn'}
	2	"build"	↳ 177
Spazio di archiviazione	3	"chatContextMenuHintWasShown"	↳ false
Archiviazione locale	4	"contactsList"	↳ [397] [420000, 21792412, 28170698, 32941423, 46517131, 70482796, 70633768, 7460]
Archiviazione sessione	5	"drafts"	↳ [259723151: {}]
IndexedDB			↳ 259723151: {date: 1654791380, entities: undefined, message: "----->https://www.youtube.com/watch?v=dQw4w9WgXc0-----", pFlags: {no_webpage: true}, rMessage: "-----&t;https://www.youtube.com/watch?v=dQw4w9WgXc0&t;-----", reply_to_msg_id: undefined, _: "draftMessage"}
			↳ {keywords: {}, version: 10879, langCode: 'en'}
keyvalue - https://web.telegram	6	"emojiKeywords_en"	↳ {2: {}, 4: {}, 5: {}}
tweb - https://web.telegram	7	"filters"	↳ {}
	8	"hiddenPinnedMessages"	↳ true
SQL web	9	"keepSigned"	↳ {_: 'langPackDifference', lang_code: 'en', from_version: 0, version: 584915, st: 19379191807}
Cookie	10	"langPack"	↳ {notifyUsers: {}, notifyBroadcasts: {}, notifyChats: {}}
Considera attendibili i token	11	"maxSeenMsgId"	↳ {0: Array(0), 1: Array(0)}
Gruppi di interesse	12	"notifySettings"	↳ {volume: 1, muted: true, playbackRate: 1, playbackRates: {}, loop: false, _}
Cache	13	"pinnedOrders"	↳ {push_action_mutel: 'Mute background alerts for 1 day', push_action_settings: 1654790797088}
Spazio di archiviazione cache	14	"playbackParams"	↳ {nosound: true, nodesktop: undefined, volume: 0.5, novibrate: undefined, noprev: []}
Cache back-forward	15	"push_lang"	↳ []
Servizi in background	16	"push_last_alive"	↳ {messagesTextSize: 16, distanceUnit: 'kilometers', sendShortcut: 'enter', anima: 1654779330900}
Recupero dello sfondo	17	"push_settings"	↳ 472925813
Sincronizzazione in backrou	18	"recentEmoji"	↳ undefined
Notifiche	19	"recentSearch"	↳ {correspondents: {}}
Gestore dei pagamenti	20	"settings"	↳ {seq: 328, pts: 408742, date: 1654791144}
Sincronizzazione periodica in	21	"stateCreatedTime"	↳ "1.4.3"
Messaggi push	22	"stateId"	
API di reporting	23	"topPeers"	
Frame	24	"topPeersCache"	
top	25	"updates"	
	26	"version"	

### 3.3 IP contattati

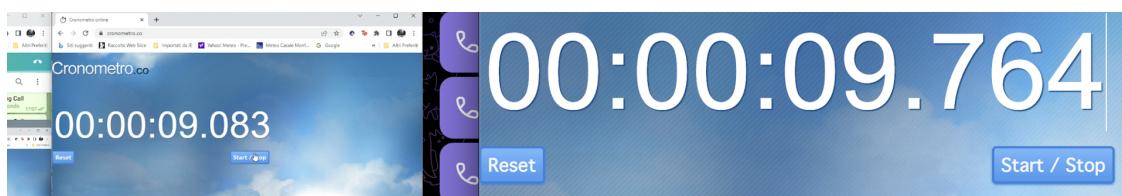
Visto che Telegram è un sito senza pubblicità o tracker il numero di IP contattati è estremamente limitato. Analizzando il file HAR ottenuto dopo il caricamento della pagina in condizioni classiche tramite lo script bash in appendice si osserva che vengono contattati soltanto 6 IP, relativi a kws2-1.web.telegram.org, kws2.web.telegram.org, kws4-1.web.telegram.org, kws4.web.telegram.org, venus.web.telegram.org e web.telegram.org. I primi 4 sono contattati per aprire le connessioni coi vari server API; web.telegram.org è invece il nome del sito di telegram web, contattato all'apertura della pagina. Infine, venus.web.telegram.org è probabilmente un server di smistamento contattato durante l'apertura della connessione verso i server API: se infatti si scollega la connessione Wi-Fi Telegram, sfruttando i meccanismi descritti precedentemente, rileva la presenza di problemi di rete. Per averne la certezza manda una richiesta di connessione verso un server API, che ovviamente fallisce. Analizzando quest'ultima (presente nella foto in basso) si nota come effettivamente l'URL contattato per provare ad aprire la connessione sia proprio venus.web.telegram.org.

Nome	Intestazioni	Anteprima	Risposta	Iniziatore	Tempistiche
apiw1	Richiedi URL: https://venus.web.telegram.org/apiw1 Norme sul referer: strict-origin-when-cross-origin				
favicon.ico	▼ Intestazioni delle richieste ⚠ Vengono mostrate intestazioni provvisorie Scopri di più Referer: https://web.telegram.org/ User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0 Safari/537.36				
favicon.ico					

Per quanto riguarda gli IP contattati, questi ovviamente variano leggermente in base al dispositivo utilizzato. È stato notato come Telegram possieda dei server proprietari, collocati in varie parti del mondo. In particolare sono state osservate due diverse famiglie di IP: una viene usata per tutta la parte di messaggistica, mentre l'altra viene contattata durante le chiamate o le videochiamate.

Uno degli IP osservati durante lo scambio di messaggi è 149.154.167.99. Attraverso dei siti appositi si riesce a capire che è localizzato in Inghilterra. Il risultato è coerente con quelli ottenuti da ping: si ottiene un RTT di circa 40 ms. Effettuando un ping verso altri indirizzi localizzati a Londra si ottengono risultati simili, quindi si può assumere che quanto presente sui siti sia corretto. Questo è inoltre coerente anche con la logica di Telegram: visto che le conversazioni sono in tempo reale è necessario avere una latenza relativamente bassa, e di conseguenza servono molti server dislocati nel mondo per garantire un buon servizio ovunque.

Per le chiamate (sia normali sia videochiamate) viene invece contattato l'IP 91.108.13.39. Questo viene localizzato in America (più precisamente in Florida). Nonostante in un primo momento possa sembrare assurdo il ping conferma questo risultato: i RTT ottenuti oscillano tra i 150 ms e i 300 ms. Questo collide in parte con la logica delle chiamate: ci si aspetterebbe infatti di avere un RTT molto basso per permettere l'interattività, mentre questo nella realtà non si verifica. Addirittura, facendo un test con un cronometro durante una videochiamata è stato rilevato un ritardo di 700 ms, rendendo molto difficile la sua buona riuscita.



Nell'immagine sono presenti un cronometro eseguito in locale sul computer (a destra) e uno visibile nella condivisione schermo (a sinistra) fatti partire contemporaneamente. Il ritardo è di circa 700 ms.

In realtà nei vari test la posizione dell'IP è variata: sono stati contattati anche server a Singapore o in Olanda. Questo fa pensare che la scelta del server dipenda dal loro carico, si sceglie il server libero più vicino, ma questo può essere distante complicando molto l'interazione. Lo stesso problema non si verifica anche con i messaggi (o comunque è molto più raro) perché il carico di lavoro richiesto a un server per gestire una chiamata è ovviamente molto maggiore rispetto a quello per inviare dei messaggi. La scelta diversa dei server può inoltre dipendere dalla necessità di avere una criptazione end-to-end: più che server veri e propri questi IP corrispondono probabilmente a dei relay, e questo spiegherebbe la variazione di IP.

### 3.4 Invio messaggio

All'arrivo di un messaggio testuale non vengono aperti ulteriori websocket. Presumibilmente il testo viene letto dal websocket predisposto per gli update (la webAPI sempre attiva) che, notando un aggiornamento sul server, lo scarica. Analizzando il suddetto websocket con l'inspector, all'interno della sezione "messaggi", si nota che all'arrivo del messaggio testuale la dimensione dei messaggi scambiati aumenta (rispetto alla dimensione dei messaggi keep-alive) confermando la teoria formulata.

### 3.5 Chat segrete

Per le chat segrete il meccanismo di comunicazione è simile a quello delle conversazioni standard, ma cambia il protocollo di crittografia adottato: viene infatti sfruttata la end-to-end encryption. In questo modo i dati saranno criptati, ma in aggiunta non transiteranno neanche da un server intermedio: il destinatario dei pacchetti sarà direttamente la persona con cui si stanno scambiando messaggi (a meno di un relay sfruttato per ovviare al problema del NAT traversal). Di conseguenza, come accennato precedentemente, queste conversazioni sono disponibili soltanto sul dispositivo da cui vengono avviate. Nel caso in cui un altro utente creasse una chat segreta, questa sarebbe avviata con l'ultimo dispositivo abilitato su cui è stato effettuato l'accesso per il destinatario.

È inoltre impossibile avviare una chat segreta dal browser web, nonostante sia disponibile la UI per farlo. Questo è probabilmente un bug di Telegram e non è riconducibile all'utilizzo del protocollo end-to-end: infatti le chiamate e le videochiamate funzionano correttamente da browser.

### 3.6 Invio messaggio vocale

Durante l'invio di un messaggio vocale l'utente apre una connessione con un server API per caricare l'audio. Per ridurre il tempo di download il ricevitore non scarica interamente il vocale ma solo una parte (il server alla richiesta iniziale risponde inserendo sul response header un content range ridotto).

Se il messaggio vocale non risulta troppo lungo non viene aperta una connessione dedicata con un server API in download (in upload serve sempre).

### 3.7 Invio immagine (compressa e non)

Durante la ricezione di immagini il comportamento risulta leggermente diverso. Il websocket primario viene utilizzato per il download dell'anteprima dell'immagine visualizzata nell'elenco delle chat.



Questo è la visualizzazione classica di una chat su Telegram web.  
L'immagine piccola a sinistra della scritta "Photo" è l'anteprima scaricata dal websocket primario.

Anche qui viene supposto l'invio diretto dell'anteprima all'interno dell'update di notifica come per i messaggi di testo, in quanto quest'ultima ha una risoluzione molto bassa e non vengono aperte nuove connessioni con altri server API. Una volta aperta la chat contenente l'immagine ricevuta si presentano due scenari in base alla tipologia di compressione scelta in fase di invio: è possibile scegliere se inviare l'immagine in formato originale o previa compressione.

Nel caso si scelga di non comprimere l'immagine essa viene mostrata come se fosse un file privo di anteprima di dimensione maggiore. Il download automatico in questo caso risulta disattivato ed è quindi necessario premere sull'apposito pulsante per avviare il download dell'immagine alla risoluzione originale.

Nel caso in cui l'immagine venga invece compressa in fase di invio, essa verrà mostrata direttamente all'interno della chat. Il download avviene in maniera automatica all'apertura della chat e viene effettuato ad una risoluzione inferiore per permettere almeno di mostrarne l'anteprima. Una volta aperta l'immagine essa viene scaricata nuovamente ma questa volta alla risoluzione massima possibile a seguito della compressione.

In tutte le situazioni in cui l'immagine viene scaricata, per ognuna delle sue risoluzioni e dimensioni, viene aperto un nuovo websocket, il quale viene sfruttato soltanto per effettuare il download (e viene quindi chiuso subito dopo). L'immagine scaricata risulta inoltre visibile all'interno dell'inspector.

Se un'immagine venisse inviata molteplici volte all'interno della stessa chat o se venisse inviata la stessa immagine da un'altra chat, essa non viene scaricata nuovamente poiché l'applicazione riconosce che la stessa immagine è già presente all'interno della cache. Questo comportamento è presumibilmente sfruttato dal lato del server, o meglio del cloud, per permettere di non duplicare lo spazio occupato da immagini uguali e quindi il riferimento alla stessa risulta identico per i diversi update ricevuti. Il meccanismo è equivalente a quello studiato con HTTP ed è probabilmente equivalente alla ricezione di un messaggio 304 (not modified): quando il websocket principale riceve un update si accorge che l'immagine possiede lo stesso identifier di un file già presente in cache e quindi non la scarica nuovamente.

### 3.8 Invio video

Per l'invio dei video Telegram offre due soluzioni in base alle proprie necessità:

- Invio come file di dimensioni originali
- Invio compresso

Il primo metodo tratta il video come un file generico: la visualizzazione risulta possibile solo dopo che il file è stato scaricato.

Il secondo metodo, invece, sfrutta un algoritmo di compressione di Telegram per inviare il file. Alla ricezione del messaggio l'utente riceve solo la preview del video (immagine ottenuta tramite l'apertura di un websocket). Quando viene aperto il video parte la riproduzione automatica. Il meccanismo è molto simile alla riproduzione di video in streaming: si aprono più websockets per scaricare in contemporanea frammenti del filmato, cercando di creare un buffer in modo da ovviare al problema del jittering. Nel caso di invii multipli dello stesso video anche in questo caso, così come per le immagini, i server sono in grado di riconoscere che il file è sempre lo stesso ed eliminano il duplicato. Tuttavia, poiché i video sono di solito molto pesanti non possono essere salvati nella cache del dispositivo e vengono quindi eliminati il prima possibile. Di conseguenza, anche se il file inviato è lo stesso il client non riesce a rendersene conto (in quanto la cache è stata svuotata) e lo scarica come se fosse un nuovo video.

Per quanto riguarda l'utente che riceve il filmato, dato che la cache del browser non permette di salvare grossi quantitativi di dati, questo porta ad avere download multipli degli stessi frame nel caso in cui ci si spostasse lungo la linea temporale del filmato. Questo conferma ulteriormente l'ipotesi precedente.

Nel caso in cui la connessione dovesse venire a mancare, il download del video rimarrebbe incompleto rendendo riproducibile solo la parte già scaricata.

Diverso discorso è nel caso in cui l'utente abbia ricevuto un video sotto forma di file generico: in quel caso non è possibile andare a visionare parte del video poiché Telegram lo considera come un unico file e quindi attende di completare il download per renderlo fruibile all'utente solo all'esterno del browser e non tramite il player integrato.

Infine per i messaggi inoltrati, essendo già presenti sul cloud, viene semplicemente allegato al messaggio il link di riferimento necessario al suo invio. Non vengono dunque effettuati ulteriori caricamenti di file durante l'inoltro.

### 3.9 Invio file generico

Per quanto riguarda l'invio di file l'applicazione apre 3 websocket per permettere un invio in parallelo: in questo modo non si soffre in maniera drastica la perdita di un pacchetto durante l'invio a causa del controllo di congestione di TCP e si mantengono quasi sempre buone velocità di upload.

Durante il download il comportamento è pressoché lo stesso: vengono aperte connessioni sempre con 3 websockets. In generale però queste non vengono chiuse immediatamente al termine dello scambio di dati ma dopo circa 11 secondi di inattività del socket. Questo è un meccanismo simile a HTTP persistente: le connessioni rimangono temporaneamente attive per permettere di velocizzare ulteriori operazioni di download (se presenti).

□ ⚡ apius	101	websocket	<u>websocket.ts:49</u>	0 B	11.65 s
□ ⚡ apius	101	websocket	<u>websocket.ts:49</u>	0 B	11.30 s
□ ⚡ apius	101	websocket	<u>websocket.ts:49</u>	0 B	10.90 s

*Test effettuato per verificare per quanto tempo rimangono attivi i websocket al termine del download.*

*Per ottenere il valore numerico preciso è stato annullato un download non appena questo è partito.*

In questo caso non è presente l'opzione per inviare il file compresso. Una volta terminato il download sarà possibile salvare il file nella memoria del dispositivo fino a quando questo sarà presente all'interno della cache del browser (altrimenti il download verrà ripetuto).

### 3.10 Chiamata vocale, videochiamate e screen sharing

Le chiamate vocali, insieme alle videochiamate, sono le uniche funzionalità che utilizzano un diverso protocollo per il trasferimento dei pacchetti. Per tutte le altre funzioni, infatti, viene usato TCP con TLS 1.3 che ne permette la criptazione, mentre, per ridurre la latenza sulle chiamate utilizziamo SRTP che si basa su RTP e fornisce una criptazione dei dati. RTP si basa su UDP; partendo da esso aggiunge al livello 4 informazioni come un identifier univoco per la sorgente, un sequence number per ricostruire il flusso multimediale ed un payload type per specificare la codifica del payload. Il motivo per cui si utilizza UDP e non TCP principalmente è la latenza media più bassa: UDP invia alla stessa velocità a prescindere dallo stato della rete (in quanto non include un meccanismo di controllo di congestione) mentre TCP in caso di perdite rallenta riducendo la finestra di invio. Inoltre, UDP non attende ACK dal ricevitore al contrario di TCP; questo aumenta ovviamente la probabilità di perdita dei pacchetti, ma nell'ambito delle chiamate è preferibile perdere qualche frame ma mantenere una buona latenza.

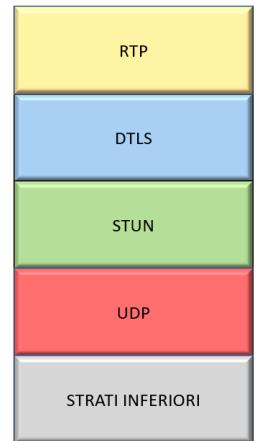
Durante le chiamate si presenta il problema di dover superare il NAT per poter raggiungere l'utente da contattare. Per risolverlo viene adoperato il protocollo STUN (che incapsula SRTP): inizialmente si prova a raggiungere direttamente l'utente da contattare per recuperarne l'indirizzo IP e permettere dunque la creazione di una connessione diretta tra i due host (e alleggerire l'eventuale carico sui server). Nel caso in cui questo tentativo non vada a buon fine si procede con la comunicazione tramite un relay server utilizzando il protocollo TURN (Traversal Using Relays around NAT), il quale non è comunque in grado di decifrare il flusso RTP poiché la criptazione avviene end to end.

192.168.1.114	91.108.9.53	STUN	158 Refresh Request user: 1654808672:385b12571c17213ca6 realm: telegram.org with nonce
192.168.1.114	91.108.9.53	STUN	146 ChannelData TURN Message
91.108.9.53	192.168.1.114	STUN	110 Refresh Success Response lifetime: 600
192.168.1.114	91.108.9.53	STUN	158 Refresh Request user: 1654808672:385b12571c17213ca6 realm: telegram.org with nonce
192.168.1.114	91.108.9.53	STUN	556 ChannelData TURN Message
91.108.9.53	192.168.1.114	STUN	110 Refresh Success Response lifetime: 600

*Pacchetti STUN e relativo lifetime visualizzati su Wireshark.*

Per mantenere attiva una connessione TURN al termine del lifetime precedente i due host si mandano pacchetti per verificare che siano ancora in grado di comunicare: in caso positivo incrementano il lifetime, mentre in caso negativo (se un host dovesse scollegarsi dalla rete) il lifetime rimane nullo e, dopo alcune richieste, la connessione viene terminata.

Nel caso in cui sia stata attivata la chiamata su STUN (connessione diretta tra i due host), quando un utente si disconnette l'altro, in seguito all'invio dei suoi pacchetti, riceverà degli ICMP host unreachable che chiuderanno la connessione. Se invece un host chiude la chiamata l'altro riceverà degli ICMP port unreachable e la connessione verrà terminata.



## Sezione 4 - Conclusioni

Riassumendo, Telegram web è un'applicazione web di messaggistica e chiamate basata principalmente sul protocollo TCP con TLS 1.3, con alcune funzionalità che per necessità sfruttano protocolli diversi come STUN e RTP criptati. La maggior parte delle operazioni avviene tramite websockets (HTTP 101 Switching protocols) che contattano i server API del servizio. Ogni file multimediale viene scaricato sempre attraverso server API ma utilizzando websockets diversi dal principale, il quale è incaricato del download di messaggi e di file di piccole dimensioni. Infatti, è stato osservato che qualunque contenuto multimediale sotto a una certa dimensione (a parte i file) possa essere inviato attraverso il server API sempre aperto all'interno dei messaggi di update, mentre per quelli più pesanti, per non appesantire il canale usato per gli aggiornamenti, vengono aperte connessioni apposite.

Attualmente la velocità di download è limitata lato server da Telegram, ma in futuro, con l'arrivo di Telegram premium, questo limite potrà essere rimosso attraverso un pagamento.

Il multi dispositivo è sviluppato in maniera ottimale tramite lo sfruttamento del cloud, il quale permette di avere backup sincronizzati tra i vari dispositivi ad esclusione dei servizi che richiedono crittografia end-to-end.

- ben fatto e completo
- si poteva arricchire con qualche grafico nel tempo

10/10

## Appendice

Connessione al server durante la fase di login. Si può notare in particolare l'autenticazione con Client Hello e Server Hello di TLS.

19 0.235979	192.168.68.127	92.123.180.25	TCP	66 60056 → 443 [ACK] Seq=1 Ack=1 Win=132480 Len=0 TSval=32060893...
20 0.235981	192.168.68.127	92.123.180.25	TLSv1.3	583 Client Hello
21 0.250029	92.122.248.69	192.168.68.127	TCP	74 443 → 60057 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1452 S...
22 0.250194	192.168.68.127	92.122.248.69	TCP	66 60057 → 443 [ACK] Seq=1 Ack=1 Win=132480 Len=0 TSval=10042725...
23 0.250195	192.168.68.127	92.122.248.69	TLSv1.2	583 Client Hello
24 0.253547	92.123.180.25	192.168.68.127	TCP	66 443 → 60056 [ACK] Seq=1 Ack=518 Win=30080 Len=0 TSval=1754314...
25 0.269234	92.123.180.25	192.168.68.127	TLSv1.3	1506 Server Hello, Change Cipher Spec, Application Data
26 0.269235	92.123.180.25	192.168.68.127	TCP	1506 443 → 60056 [ACK] Seq=1441 Ack=518 Win=30080 Len=1440 TSval=1...
27 0.269236	92.123.180.25	192.168.68.127	TLSv1.3	1061 Application Data, Application Data, Application Data
28 0.269237	92.122.248.69	192.168.68.127	TCP	66 443 → 60057 [ACK] Seq=1 Ack=518 Win=64768 Len=0 TSval=3304767...
29 0.269237	92.122.248.69	192.168.68.127	TLSv1.2	1506 Server Hello
30 0.269238	92.122.248.69	192.168.68.127	TCP	1506 443 → 60057 [PSH, ACK] Seq=1441 Ack=518 Win=64768 Len=1440 TS...
31 0.269239	92.122.248.69	192.168.68.127	TLSv1.2	1282 Certificate [TCP segment of a reassembled PDU]

Websocket sempre attivo visualizzato tramite il browser inspector.

□ <a href="#">apiws</a>	101	websocket	<a href="#">websocket.ts:49</a>	0 B	Pending
-------------------------	-----	-----------	---------------------------------	-----	---------

Effetti applicati dai file scaricati all'avvio della registrazione di un audio.



Ricezione immagini (icona elenco chat + download formato esteso).

□ <a href="#">data:image/jpeg;bas...</a>	200	jpeg	<a href="#">blur.ts:82</a>	(memory cache)	0 ms
□ <a href="#">apiws</a>	101	websocket	<a href="#">websocket.ts:49</a>	0 B	10.30 s
□ <a href="blob:https://web.telegram.org/9c9ec3d4-3a42-41ac-ba...">blob:https://web.telegram.org/9c9ec3d4-3a42-41ac-ba...</a>	200	jpeg	<a href="#">renderImageFromUrl.ts:40</a>	0 B	2 ms

Script bash per ricavare i server contattati.

```

1 file=""
2 outFile=""
3
4 differentServers=$(cat $file | egrep "\"url\": " | cut -d "/" -f 3 | sort | uniq)
5 numServers=$(cat $file | egrep "\"url\": " | cut -d "/" -f 3 | sort | uniq | wc -l)
6 echo -e "ci sono $numServers server diversi contattati:\n${differentServers}" > $outFile
7
  
```

Output script.

```
1 ci sono 6 server diversi contattati:
2 kws2-1.web.telegram.org
3 kws2.web.telegram.org
4 kws4-1.web.telegram.org
5 kws4.web.telegram.org
6 venus.web.telegram.org
7 web.telegram.org
8
```

IP contattato per le videochiamate.

Geolocation data from [IP2Location](#) (Product: DB6, updated on 2022-6-1)

IP Address	Country	Region	City
91.108.13.39	United States of America 	Florida	Virginia Gardens
ISP	Organization	Latitude	Longitude
Telegram Messenger Network	Not Available	25.8104	-80.3023

IP contattato per i messaggi.

Geolocation data from [IP2Location](#) (Product: DB6, updated on 2022-6-1)

IP Address	Country	Region	City
149.154.167.99	United Kingdom of Great Britain and Northern Ireland 	England	Warrington
ISP	Organization	Latitude	Longitude
Telegram Messenger Network	Not Available	52.1845	-0.6876