Jami

Oggi ci scostiamo dalla messaggistica centralizzata o federata e ci avviciniamo a quella *peer to peer*, ovvero quella per cui i messaggi non si appoggiano ad un server ma partono da un client e vanno diretti all'altro client.

Jami

Jami è un'app di messaggistica *peer to peer* in cui il riconoscimento degli utenti (che chiameremo host, anche in realtà ogni host può avere più utenze) è affidato ad una DHT (vedi sotto che cosa sia).

Tipicamente i client p2p hanno vantaggi e svantaggi, tra cui

1. PRO: non sono censurabili chiudendo un server centralizzato (e non c'è quindi un server centralse che se via giù blocca tutte le comunicazioni)
2. PRO: sono quindi indipendenti da un'entità "di controllo"
3. CONTRO: mostrano l'indirizzo IP di connessione
4. CONTRO: necessitano di stare in ascolto fisso: non essendoci un server centralizzato, se due utenti vogliono contattarsi devono essere online contemporaneamente (e su un telefono questo è un problema perché significa non mandarlo in deep sleep e consumare TONNELLATE di batteria)

Il punto 3, per quanto mi riguarda, è un compromesso accettabile. Sotto rete mobile si è comunque dietro doppio nat (per cui riduce la raggiungibilità diretta), mentre in caso di client desktop è possibile usare una VPN.

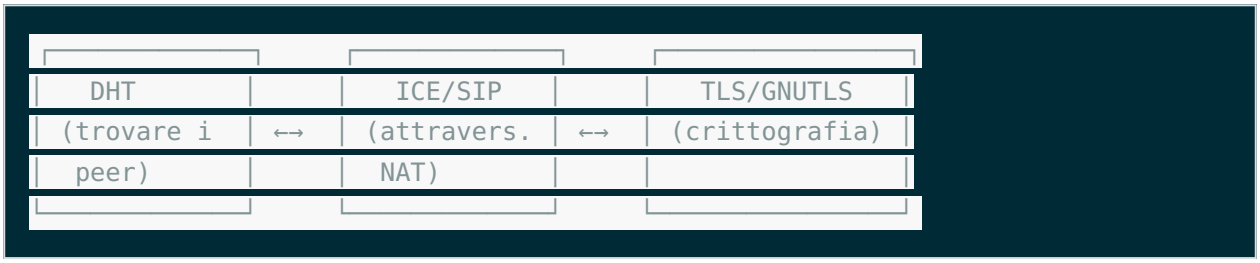
Nel caso di Jami, il punto 4. è stato risolto prevedendo la possibilità di (non obbligando a) appoggiarsi ad un proxy DHT + notifiche FCM: il client si appoggia a dei server (ufficiali o on premise) che stanno in ascolto per voi: quando vi arriva un messaggio, vi mandano la notifica push e permettono al vostro processo Jami di prendere in carico il messaggio.

Se il destinatario non è raggiungibile al momento, il messaggio non viene garantito come consegnato e può andare perso e la sua consegna può essere ritentata solo in base al comportamento del client e della

rete P2P, non tramite uno storage centralizzato persistente. In questo caso il "consiglio" è quello di installare un Jami server (https://docs.jami.net/en_US/user/jami-for-servers.html), ovvero un processo clone del vostro client, con le stesse chiavi su un qualcosa che sta fisso acceso (per esempio un Raspberry): cioè di fatto non è un server in senso classico: è solo un companion legato alla vostra utenza che ascolta quando voi non potete e vi dice le novità quando tornate online.

Jami inoltre permette il multidevice (cosa che altri competitor non offrono.)

Funzionamento a volo d'uccello



Come in ogni storia di crittografia che si rispetti, dobbiamo tirare in ballo *Alice* e *Bob*.

Flusso *molto* semplificato:

- Alice crea un account → genera una coppia di chiavi (pubblica/privata): la chiave pubblica è l'identità di Alice (hash dell'chiave)
- Alice pubblica la sua presenza sulla DHT
- Bob cerca l'ID di Alice sulla DHT → ottiene indirizzo IP/porta corrente
- Connessione diretta (o tramite TURN se NAT problematico)
- Comunicazione cifrata end-to-end

Cos'è una DHT? (Distributed Hash Table)

Intanto partiamo col rispolverare la definizione di *Hash Table*, ossia una tabella che associa ad una chiave un valore hash

```
chiave "mario"  →  valore "abcd123456"
chiave "lucia"  →  valore "cde0123212"
chiave "marco"  →  valore "9091019abf"
```

Una hash table tradizionale risiede su un solo host -> se l'host va già, la HT non è più consultabile.

Il problema

Voglio una hash table che:

- non sia su un singolo host (no single point of failure)
- si possano aggiungere/rimuovere host senza rompere tutto
- la si possa interrogare da qualsiasi host e ottenere la risposta

La soluzione

Prendiamo 1000 host. Ad ognuno diamo un ID numerico casuale tra 0 e 2^160 (un numero enorme).

Prendiamo una chiave, per esempio "mario". Ne facciamo l'hash, otteniamo un numero nello stesso intervallo.

Regola fondamentale: la chiave "mario" viene memorizzata sull'host il cui ID è il più vicino possibile all'hash di "mario".

"Cosa significa più vicino?" — si usa la distanza XOR, ma per capirci: immaginate i numeri su una linea circolare. L'host che possiede la chiave è quello con l'ID numericamente più vicino all'hash della chiave.

Esempio di ricerca di un valore

Vogliamo trovare il valore associato a "mario".

- calcoliamo hash("mario") = 42 (tirato a caso eh)
- contattiamo un host qualsiasi della DHT (uno qualunque)
- quello risponde: "Io non ho la chiave 42, ma so chi è più vicino di me a 42: contatta l'host X, lui è più vicino"
- andiamo da X, e si ripete la manfrina, fino a trovare quello che ha la chiave

Jami (come BitTorrent) usa la DHT Kademlia, che portano a trovare la chiave in pochi passi (logaritmici) anche con miliardi di host.

Esempio di pubblicazione di un valore

È l'operazione duale rispetto alla precedente:

- Alice vuole pubblicare hash(chiave_pubblica_di_Alice) → indirizzo_IP_di_Alice.
- calcola l'hash della sua chiave pubblica
- trova (con lo stesso meccanismo di ricerca) gli host più vicini a quell'hash
- invia loro la coppia (chiave, valore)

Quegli host la memorizzano per un certo tempo (TTL).

Cosa succede se un host se ne va o mente

La DHT è ridondante: una chiave non è memorizzata su un solo host ma sui k host più vicini (tipicamente k=8 o k=20).

Se un host scompare, la chiave è ancora sugli altri. Periodicamente gli host si "riorganizzano" e replicano i dati.

Un host interrogato ha pochissimo margine per mentire sulla ricerca perché:

- vengono interrogati più host in parallelo per evitare nodi instabili, e la struttura geometrica della rete permette di verificare se qualcuno sta instradando la richiesta nella direzione sbagliata.
- la validazione delle chiavi firmate riduce la possibilità di *injection* (anche se in linea prettamente teorica non elimina del tutto il rischio di *poisoning* della DHT).

Quindi cosa è una DHT

Una DHT è un sistema dove un insieme di host si coordina per fare da "rubrica distribuita": data una chiave, ti dicono quale host ha il valore associato, senza che nessun host sappia tutto.

Diagramma OpenDHT

Da tenere presente è che il server di bootstrap è necessario, ma:

- esiste il caching
- è possibile girare il proprio nodo opendht (quindi non c'è dipendenza da quello ufficiale di Jami)

[Che il talpone sia con voi]

[EOF]

Fluxer Icon

Canale di discussione e supporto

Canale di discussione e supporto. Per confrontarti su configurazioni o altro entra su fluxer.dreadful.work (https://fluxer.dreadful.work/invite/dreadful).