

UNIVERSITÀ DEGLI STUDI DI BARI

CORSO DI LAUREA MAGISTRALE IN
INFORMATICA

GitHub Chat

Documentazione di Progetto

a cura di

Quercia Luciano (prima versione)

Raimondi Vincenzo

Sistemi per la collaborazione in rete

a.a. 2013-2014

Indice

Introduzione	3
1 Tecnologie utilizzate	3
1.1 WebRTC	3
1.1.1 PeerConnection	4
1.1.2 Signaling	4
1.1.3 Eventuali problemi di connettività	6
1.1.4 DataChannel	7
1.2 Google Chrome Extensions	8
1.2.1 Extension UIs	8
1.2.2 Background Pages	9
1.2.3 Content Script	9
1.3 GitHub	9
2 Architettura e implementazione	10
2.1 manifest.json	11
2.2 background.html	12
2.3 background.js	12
2.4 popup.html	13
2.5 popup.js	13
2.6 contentscript.js	13
2.7 DataChannel.js	13
2.8 _Locales	14
2.9 GitHubChat.crx	14
3 Uso di GitHub Chat	15
3.1 Esempio di utilizzo	16
4 Problemi noti e sviluppi futuri	18
Riferimenti bibliografici	19

Introduzione

Il progetto GitHubChat si pone l'obiettivo di permettere la comunicazione istantanea tra utenti di GitHub.com che visitano un particolare progetto software ospitato dal sito stesso.

La comunicazione instaurata consiste in una chat

- multi-a-molti
- diretta
- non mediata da server (salvo alcune eccezioni)

1 Tecnologie utilizzate

Per la realizzazione del progetto, è stata creata un'estensione del browser Google Chrome che utilizzi la nuova tecnologia WebRTC per mettere in comunicazione tra loro, in maniera diretta, i browser degli utenti.

1.1 WebRTC

WebRTC è una tecnologia open source nata il 1° giugno 2011 che consente ai browser di comunicare direttamente in tempo reale. È basata su HTML5 e JavaScript. [6]

Il progetto, è stato reso open source da Google nel Maggio 2011. Lo scopo era creare un nuovo standard che permettesse la comunicazione peer to peer fra browser anche diversi. Il lavoro di standardizzazione è iniziato all'IETF per quanto concerne il protocollo e alla W3C per le API del browser.

La bozza di WebRTC è ancora in fase di sviluppo ma delle implementazioni funzionanti sono già disponibili nelle ultime versioni stabili sia di Google Chrome che di Mozilla Firefox. Le componenti principali di WebRTC includono:

- **MediaStream** (conosciuta come `getUserMedia`), che permette ad un web browser di accedere a webcam e microfono
- **PeerConnection**, che gestisce le chiamate audio/video
- **DataChannels**, che permettono ai browser di condividere dati di qualsiasi tipo (chat, files, etc.)

1.1.1 PeerConnection

PeerConnection è la componente di WebRTC che permette di instaurare una connessione tra peers in maniera stabile ed efficiente. Tale API rende trasparente agli sviluppatori tutta la procedura di connessione anche su reti non affidabili, grazie a codecs e protocolli che eseguono in automatico operazioni di cancellazione dell'eco, adattamento alla banda disponibile, riduzione del rumore, miglioramento della qualità dell'immagine, ecc.

Quando i due pc sono direttamente collegati (ad esempio quando appartengono alla stessa rete locale) la connessione avviene direttamente, senza bisogno di alcun server intermedio. La maggior parte dei dispositivi, però, soprattutto ultimamente, agisce dietro NAT o protetti da Firewall.

Per poter instaurare una connessione tra questi peers è necessario utilizzare un server intermedio che gestisca la funzionalità di presenza online, lo scambio di segnali tra i peers e l'attraversamento di NAT e Firewall.

1.1.2 Signaling

WebRTC utilizza PeerConnection per trasmettere flussi di dati tra i peers, ma ha anche bisogno di un meccanismo per coordinare la comunicazione e inviare messaggi di controllo, un processo conosciuto come signaling.

Il signaling è utilizzato per scambiare tre tipi di informazioni:

- messaggi di controllo della sessione, che inizializzano o chiudono la comunicazione e segnalano gli errori;
- messaggi relativi alla configurazione di rete, che permettono per esempio di capire gli indirizzi IP dei computer coinvolti;
- messaggi relativi alle capacità multimediali, come ad esempio quali codecs e quale risoluzione video utilizzare per la comunicazione.

L'acquisizione e lo scambio di informazioni di rete e di peers può essere fatto simultaneamente, ma entrambi i processi devono essere completati prima che inizi lo streaming dell'audio-video o dei dati.

L'architettura offerta/risposta utilizzata è chiamata JSEP (JavaScript Session Establishment Protocol).

Il diagramma seguente ne illustra l'architettura:

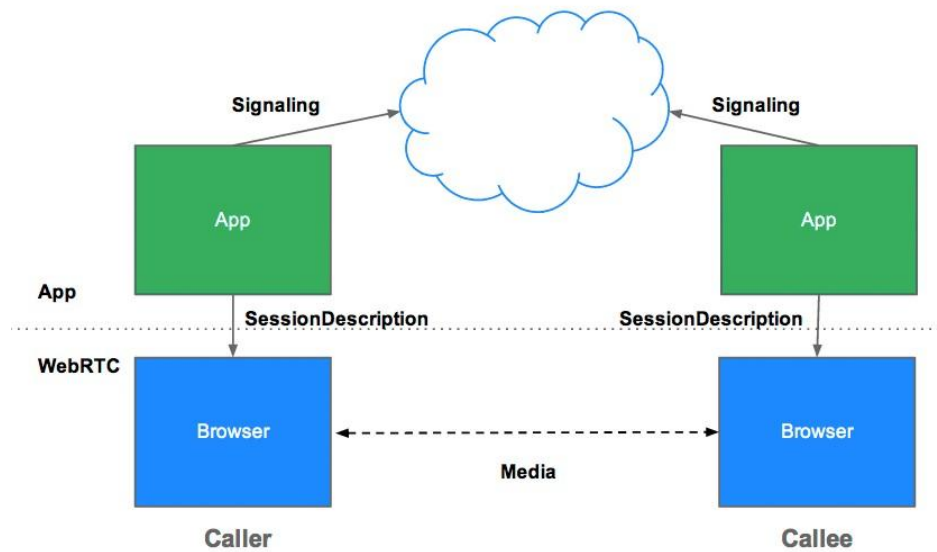


Figura 1: JavaScript Session Establishment Protocol [1]

Una volta che il processo di signaling è stato completato con successo, i dati possono essere scambiati tra i peers direttamente o, se questo fosse impossibile, attraverso un server di relay intermedio (cfr. 1.1.3).

Lo streaming vero e proprio di dati sarà affidato all'oggetto `PeerConnection`.

1.1.3 Eventuali problemi di connettività

Per mettere in comunicazione due peers non collegati direttamente, viene molto spesso utilizzato il framework ICE (Interactive Connectivity Establishment [8]). Tale framework cerca di instaurare la connessione sfruttando diversi metodi, a vari livelli, utilizzando eventualmente server STUN e TURN.

Un server STUN (Session Traversal Utilities for NAT) [5] permette alle applicazioni in esecuzione su un computer di scoprire la presenza ed i tipi di NAT e firewall che si interpongono tra il computer e la rete pubblica. STUN permette a queste applicazioni di conoscere gli indirizzi IP e le porte con cui il dispositivo NAT li sta rendendo visibili sulla rete pubblica.

TURN (Traversal Using Relays around NAT) [9], invece, è un protocollo che permette a

un client dietro NAT o firewall di ricevere dati attraverso connessioni TCP o UDP.

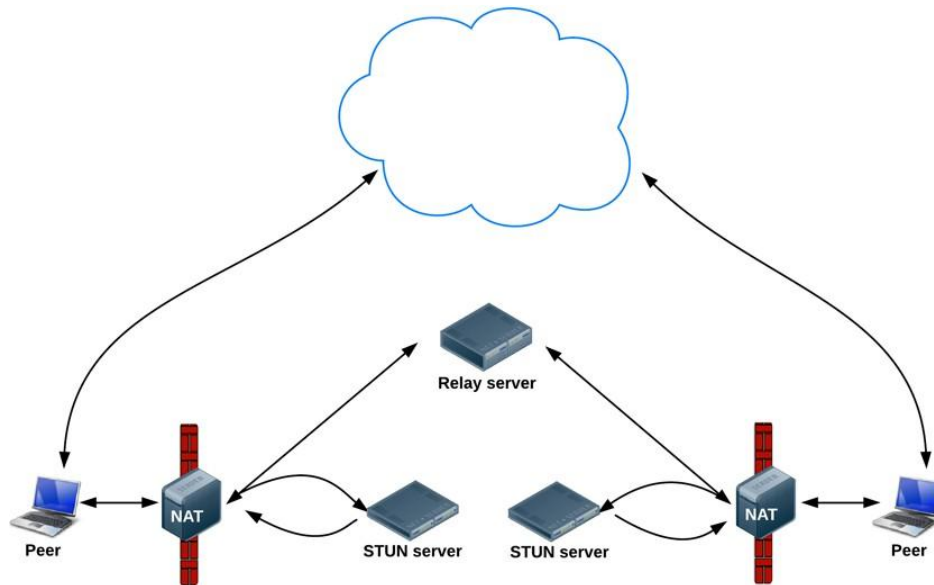


Figura 2: Diagramma di funzionamento di ICE

La sequenza di tentativi svolta dal framework ICE è elencata di seguito:

1. Connessione diretta UDP
2. Connessione diretta UDP, utilizzando server STUN per la scoperta dell'indirizzo pubblico e la porta
3. Connessione diretta TCP con HTTP
4. Connessione diretta TCP con HTTPS
5. Connessione mediata da relay TURN

Riassumendo, ICE tenta inizialmente la connessione diretta UDP, utilizzando eventualmente STUN per i peers dietro NAT. In caso di fallimento della connessione diretta, utilizza un server relay TURN per mediare la conversazione.

Il processo di ricerca del peer con cui connettersi prende il nome di ricerca di candidati.

1.1.4 DataChannel

DataChannel permette lo scambio peer2peer di qualsiasi tipo di dati, assicurando bassa latenza e alto throughput. Tale API può essere utilizzata in moltissimi casi, come per esempio applicazioni di desktop remoto, chat testuale in tempo reale, trasferimento di file e videogames.

1.2 Google Chrome Extensions

Come la maggior parte dei browser attualmente in circolazione, Google Chrome è estendibile mediante applicazioni chiamate Google Chrome Extensions.

Google ospita numerose estensioni sullo store online del suo browser, chiamato Chrome Web Store.

Le estensioni sono scritte principalmente in Javascript e, se prevedono interfacce con l'utente, in HTML e CSS.

In base alle feature da sviluppare, è possibile ricorrere a differenti componenti (per applicazioni complesse anche in modo combinato) ciascuna con dei permessi e degli obiettivi ben definiti.

1.2.1 Extension UIs

Molte applicazioni necessitano di una UI che può essere nella forma di *browser action* o *page action*.

Browser Action Viene utilizzata quando l'estensione è rilevante alla maggior parte delle pagine. Compare un'icona cliccabile nella toolbar del browser per l'avvio.



Figura 3: Browser Action

Page Action Viene utilizzata quando l'estensione è utile solo su alcune pagine. Compare un'icona nella barra degli indirizzi solo per le pagine interessate (molto spesso un filtro sull'URL).



Figura 4: Page Action

Popup In entrambi i casi precedenti, la UI dell'applicazione, se presente, viene caricata in un popup.



Figura 5: Popup

1.2.2 Background Pages

Molte estensioni hanno una background page, una pagina invisibile che contiene la logica principale dell'estensione. browser-action-with-popup

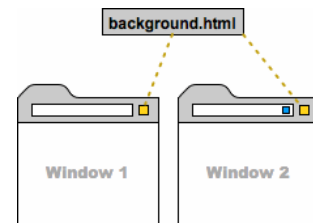


Figura 6: Background Page

1.2.3 Content Script

Se l'estensione necessita di interagire con le pagine web, allora utilizzerà un content script. Un content script è del codice Javascript eseguito nel contesto della pagina all'interno della quale è caricato.

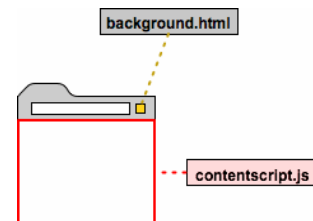


Figura 7: Content Script

La comunicazione fra le varie componenti dell'applicazione avviene tramite passaggio di messaggi JSON.

1.3 GitHub

GitHub è un servizio web di hosting per lo sviluppo di progetti software che usa il sistema di controllo di versione Git. [7]

Gli utenti registrati possono creare dei repository software pubblici gratuitamente. Con un account premium (a partire da 7\$ al mese) è possibile creare repository privati.

Tutti repository sono raggiungibili tramite lo stesso schema URL, del tipo:

`https://github.com/<username> /<project name>` dove

- **`<username>`** è il nome utente dell'autore del progetto
- **`<project name>`** è il nome del progetto, univoco tra i repository di **`<username>`** ma non all'interno di GitHub.com

2 Architettura e implementazione

L'applicazione realizzata è open-source. Il codice sorgente è liberamente scaricabile da GitHub all'indirizzo:

<http://www.github.com/lucianoq/githubchat> (versione1)

<http://www.github.com/vivimaster/githubchat> (versione2)

L'architettura dell'applicazione è descritta dal seguente schema:

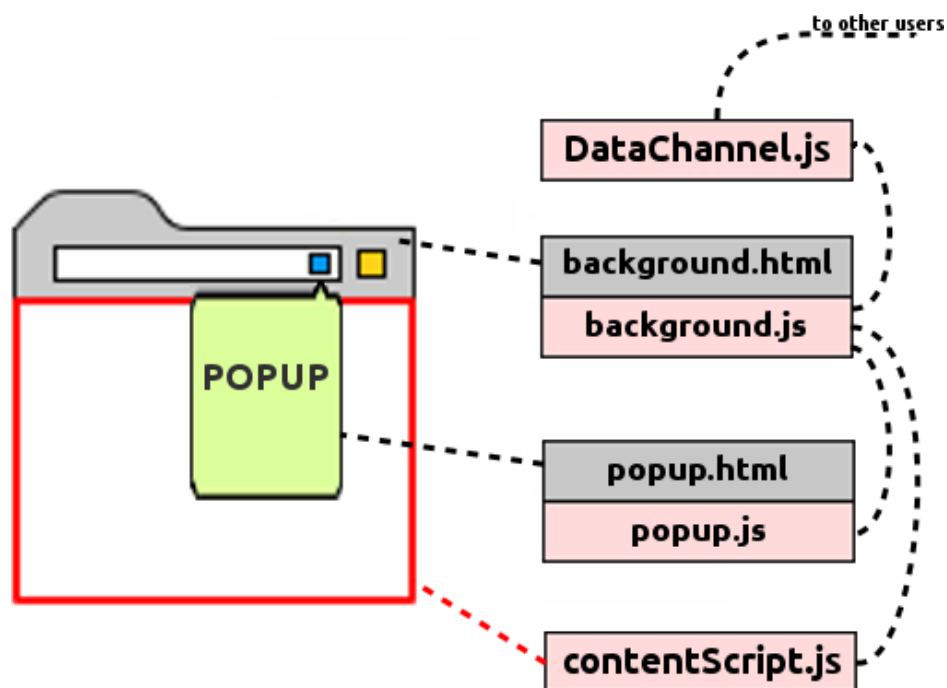


Figura 8: Architettura dell'applicazione

Qui di seguito verranno descritte le varie componenti che costituiscono l'applicazione e in alcuni casi verranno presentate porzioni di codice sorgente.

2.1 manifest.json

Il file manifesto descrive ed espone tutte le caratteristiche dell'estensione: nome, versione, icone, permessi, etc.

La versione del manifesto utilizzata è la 2.0, che, rispetto alla 1.0, introduce delle novità soprattutto dal lato della sicurezza con la nuova CSP (Content Security Policy).

```
{
  "manifest_version": 2,
  "name": "GitHub Chat",
  "version": "0.0.2.1",
  "default_locale": "en",
  "description": "MSG extDescription ",
  "icons": {
    "256": "images/logo256.png",
    "128": "images/logo128.png",
    "64": "images/logo64.png",
    "32": "images/logo32.png",
    "16": "images/logo16.png"
  },
  "page_action": {
    "default_icon": "images/logo16.png",
    "default_popup": "popup.html",
    "default_title": "GitHub Chat"
  },
  "author": "Luciano Quercia, Raimondi Vincenzo",
  "background": {
    "page": "background.html"
  },
  "content_scripts": [
    {
      "matches": ["*://*/*"],
      "js": ["contentScript.js"],
      "run_at": "document_end"
    }
  ],
  "content_security_policy": "script-src 'self' https://; object-src 'self'",
  "permissions": [
    "tabs",
    "http://www.github.com/*",
    "https://github.com/*"
  ]
}
```

2.2 background.html

Pagina HTML che non verrà mai mostrata. È l'entry point dell'estensione. Viene caricata all'avvio del browser e si occupa di chiamare i due file javascript DataChannel.js e background.js che, per questioni di sicurezza (CSP) devono risiedere in file separati.

2.3 background.js

Rappresenta il processo silente e sempre attivo nel browser. Comunica con contentScript.js per ottenere informazioni sulla pagina di GitHub, con DataChannel.js per la comunicazione con gli altri peer e con popup.js per la visualizzazione dei risultati.

Ad esempio:

```
chrome.tabs.onUpdated.addListener(checkForValidUrl);

function checkForValidUrl(tabId, changeInfo, tab) {
  var matched = tab.url.match(/github.com\/([^\s\t\v\n\r\0]+)
  \\/([^\s\t\v\n\r\0]+)/i);
  ...
}
```

si occupa di attivare la Page Action (1.2.1) solo nelle pagine dei progetti di GitHub.

```
if (data==CODESCRITTURA) {
  if (typingnotification)
    typingnotification.innerHTML=userid[0]+chrome.i18n.getMessage("iswriting");
    window.clearTimeout( reset_typing_notification);
    reset_typing_notification =setTimeout(function() {
      if (views.length > 0) {
        var popup = views[0];
        var typingnotification =
        popup.document.getElementById('typing-notification');
      }
      if (typingnotification)
        typingnotification.innerHTML="";
    }, 10000);
}
```

si occupa di gestire le typing_notification.

```
inotification=0;
window.setInterval(function() {
  if (fired){
    inotification++;
    inotification=inotification%2;
    if (newmessage) {
      if (inotification==0)
        chrome.pageAction.setIcon({path:
"images/logo16N.png",tabId:firedTabId});
      else
```

```

        chrome.pageAction.setIcon({path:
"images/logo16.png", tabId:firedTabId});
    }
    else{
        chrome.pageAction.setIcon({path:
"images/logo16.png",tabId:firedTabId});
    }
}
}, 500);

```

Si occupa di far lampeggiare l'icona della chat al ricevimento di nuovi messaggi

2.4 popup.html

Pagina HTML5 che contiene i controlli per l'inserimento e la lettura dei messaggi. Il <div> chat-output è inizialmente vuoto. Verrà riempito dinamicamente dal javascript. Il <div> chat-input contiene, invece, un <input type="text"> per la scrittura di nuovi messaggi. Lo stile della pagina è stato esternalizzato nel file style.css mentre il codice javascript nel file popup.js.

2.5 popup.js

Il file contiene la definizione delle funzioni javascript collegate alla UI. Comunica direttamente con background.js per la raccolta di tutti i messaggi arrivati fino a quel momento e la relativa visualizzazione nel popup, oltre che per il reperimento degli utenti online da visualizzare su richiesta dell'utente.

Gestisce, inoltre, gli eventi collegati alla typing notification e alla pressione del tasto Invio nell'input text.

2.6 contentscript.js

Il file content script è un sorgente javascript che viene eseguito nel contesto della pagina web visualizzata. È l'unico programma, per ragioni di sicurezza, in grado di accedere direttamente al DOM della pagina.

In questa applicazione, lo script viene utilizzato per recuperare il nome e l'avatar dell'utente loggato su GitHub in quel momento.

2.7 DataChannel.js

DataChannel.js [3] è una libreria javascript scritta da Muaz Khan [4] per la scrittura di applicazioni che necessitano di canali multi-a-molti per lo scambio di messaggi o file tramite

WebRTC.

2.8 _Locales

Contengono i file necessari per la traduzione dell'applicazione nelle diverse lingue secondo lo standard di internazionalizzazione di google.

L'applicazione supporta l'inglese come lingua di default e l'italiano.

```
{  
  
    /*manifest*/  
    "extDescription":{"message":"For instant messaging in GitHub  
projects"},  
  
    /*popup*/  
    "me":{"message":"Me"},  
    "genericError":{"message":"Impossible to continue"},  
    "connectedUsers":{"message":"Users Online:"},  
    "noUsers":{"message":"There aren't users in this chatroom"},  
  
    /*background*/  
    "onLeave":{"message":"Leaved room"},  
    "iswriting":{"message":" is writing.."},  
    "onopen":{"message":"Connected"},  
    "doLogin":{"message":"Sign-in to use chat"}  
  
}
```

2.9 GitHubChat.crx

Come tutte le estensioni di Chrome, l'applicazione è contenuta interamente in un file .crx, cioè un file .zip con un header particolare.

Questo header contiene, nell'ordine:

- magic number identificativo dell'estensione
- versione
- lunghezza della chiave (vedi sotto)
- lunghezza della firma (vedi sotto)
- chiave RSA pubblica dell'autore
- firma del pacchetto ottenuta con la funzione hash SHA-1 e codificata con la chiave privata dell'autore.

3 Uso di GitHub Chat

Terminata l'installazione, l'estensione resta sempre attiva e silente nel browser. Non necessita di avvio.

Quando raggiungiamo una pagina web contenente un progetto sul sito GitHub.com, l'estensione si attiva facendo comparire una Page Action 1.2.1.

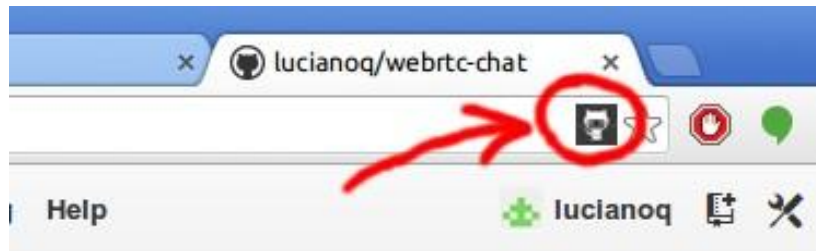


Figura 9: GitHubChat Page Action

Il click sulla Page Action apre il popup dell'estensione che contiene la UI.

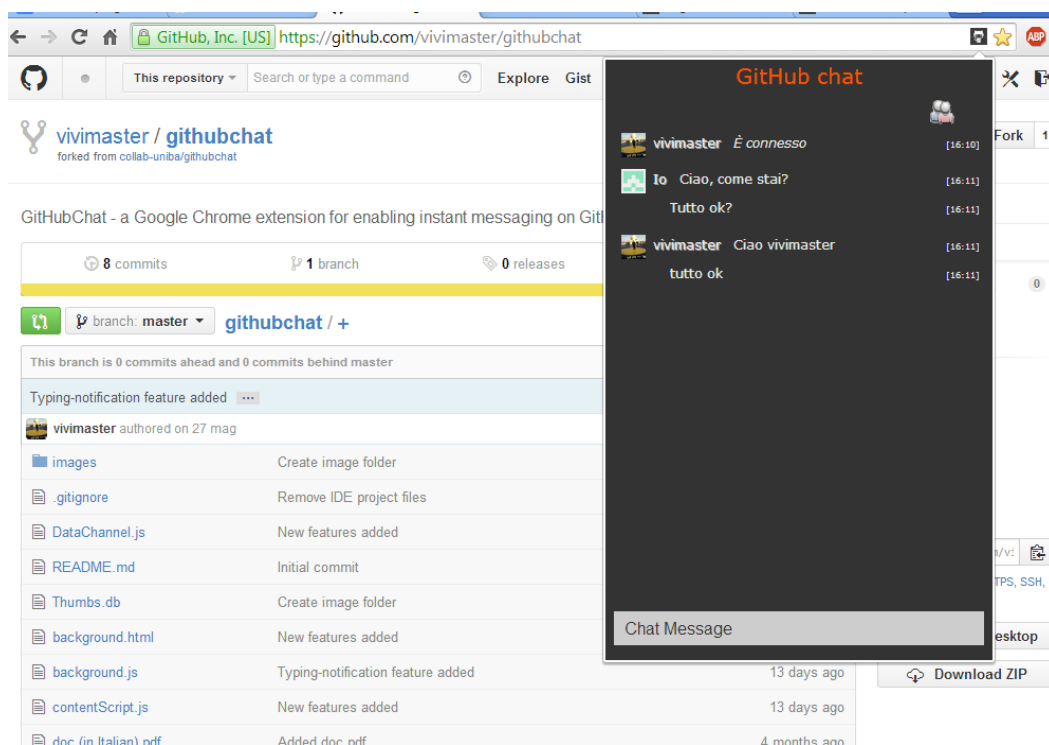


Figura 10: Esempio di Popup

È sufficiente visitare la pagina di un progetto per instaurare una connessione fra tutti gli utenti presenti. La connessione è del tutto trasparente.

3.1 Esempio di utilizzo


L'utente vuole chiedere delle informazioni riguardo il progetto Tornado di Facebook.

Per prima cosa visita la pagina del progetto:



Come descritto in figura 4, nella barra degli indirizzi compare un'icona. Se nessun utente era connesso in quel momento su quel progetto, il sistema si occupa di creare un nuovo canale di comunicazione in attesa di nuovi arrivati.

Se, invece, era già presente qualcuno in ascolto, i due utenti stabiliscono una connessione diretta. Il popup che si apre rappresenta solo la GUI dell'applicazione. I messaggi continuano ad arrivare anche con il popup chiuso.

Una volta caricata la pagina utilizzando l'icona nella barra di navigazione potrà visionare gli utenti della chat e scoprire se ce ne sono online cliccando sul pulsante: 

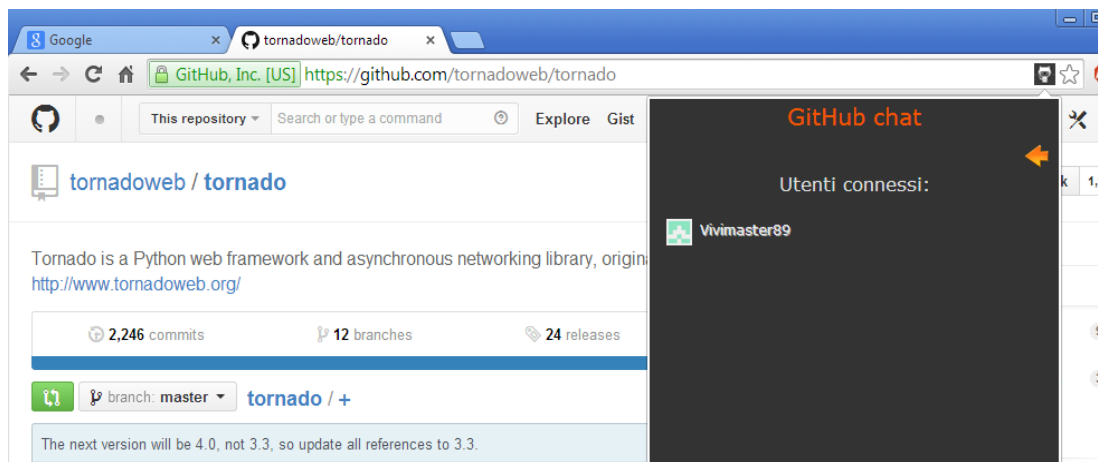




Figura 11: Esempio di chat

Quando un nuovo utente si conatterà, così come alla ricezione di un messaggio l'icona della chat inizierà a lampeggiare alternando  con 

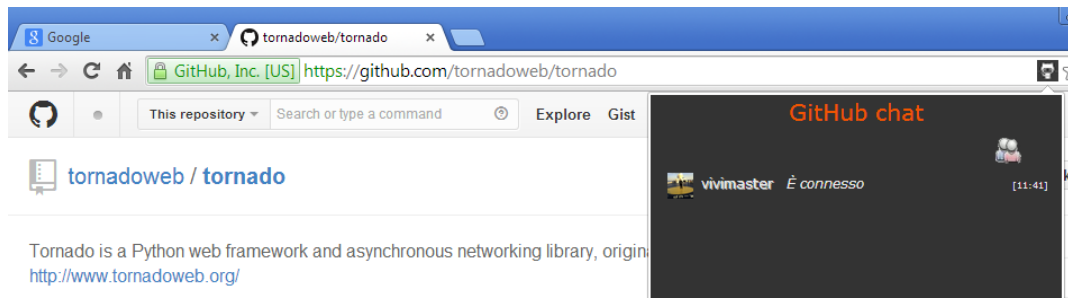


Figura 12: Esempio di chat

I due utenti ora possono iniziare a comunicare tra loro ricevendo inoltre il feedback della typing notification.

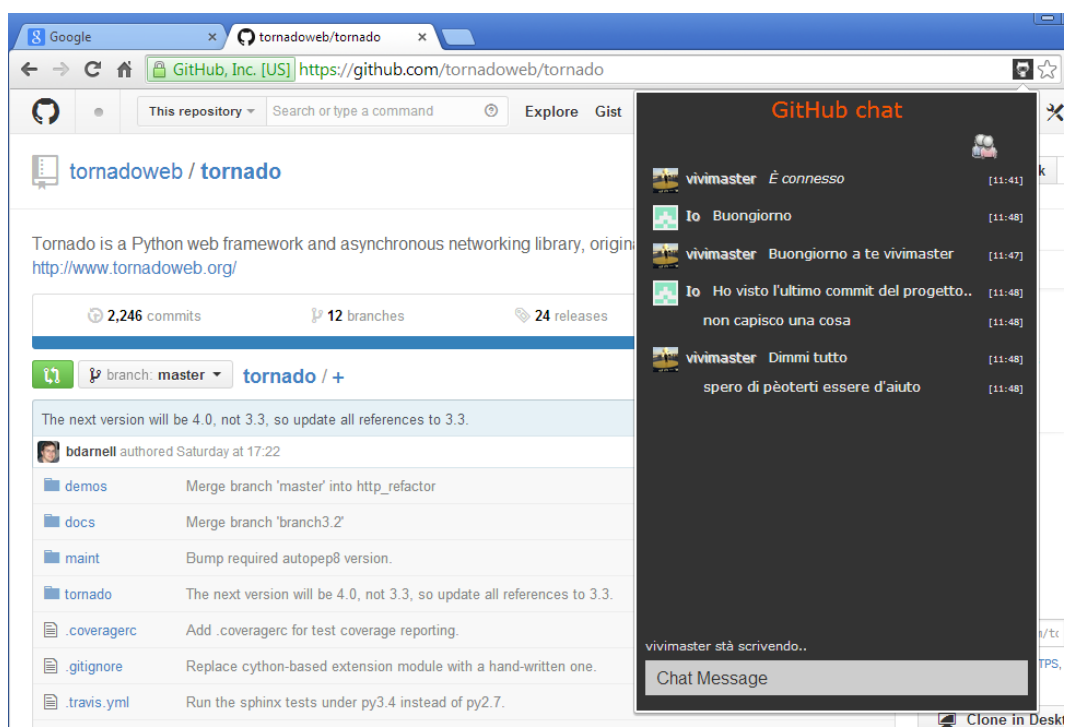


Figura 13: Esempio di chat

4 Problemi noti e sviluppi futuri

Il problema più importante di un sistema di comunicazione molti-a- molti che utilizza un protocollo di comunicazione diretta è la difficile scalabilità.

Questo perché il numero di connessioni instaurate cresce in maniera quadratica al numero degli utenti presenti. Se tra 2 utenti è necessaria una sola connessione, fra 5 utenti il numero di connessioni sale a 10, mentre fra 10 utenti arriviamo a 45.

Certo, l'assenza di un server centrale conserva i suoi vantaggi in termini di sicurezza e privacy, ma il problema della scalabilità è notevole.

Possibili miglioramenti futuri potrebbero intervenire in questa direzione, ad esempio introducendo dei nodi affidabili con ruolo di relayer. Inoltre è possibile integrare l'applicazione con le Desktop Notifications di Chrome per rimanere aggiornati sui messaggi in arrivo durante la navigazione.

Riferimenti bibliografici

[1] Dutton, S. WebRTC in the real world: STUN, TURN and signaling. <http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/>.

[2] Google Inc. WebRTC—Official site. <http://www.webrtc.org/>.

[3] Khan, M. Data Channel - WebRTC library. <https://github.com/muaz-khan/WebRTC-Experiment/tree/master/DataChannel>, 2013.

[4] Khan, M. Muaz Khan Profile — GitHub. <https://github.com/muaz-khan>, 2013.

[5] Wikipedia, L'enciclopedia libera. STUN. <http://it.wikipedia.org/wiki/STUN>, 2013.

[6] Wikipedia, L'enciclopedia libera. WebRTC. <http://it.wikipedia.org/wiki/WebRTC>, 2013.

[7] Wikipedia, the free encyclopedia. Github. <http://it.wikipedia.org/wiki/GitHub>, 2013.

[8] Wikipedia, the free encyclopedia. Interactive connectivity establishment. http://en.wikipedia.org/wiki/Interactive_Connectivity-Establishment, 2013.

[9] Wikipedia, the free encyclopedia. Traversal Using Relays around NAT (TURN). <http://en.wikipedia.org/wiki/Traversal-Using-Relay-NAT>, 2013.