UNIVERSITÀ DI PISA

Progetto di Complementi di Gestione delle Reti

Daniele Sartiano
425290

Il progetto sviluppato implementa uno sniffer in grado di catturare pacchetti, di visualizzare all'utente finale i flussi esportati, la geolocalizzazione degli indirizzi ip di questi, e un grafico con il throughput dei pacchetti con procollo TCP, UDP e ICMP.

Architettura

Il programma è stato realizzato con multithreading, in modo tale da sfruttare le architetture multicore e quindi di poter scalare in base al numero di CPU della macchina utilizzata. Esso consta di un thread che cattura pacchetti dal device selezionato dall'utente, questi vengono smistati su di N code circolari, le quali sono gestite da N thread diversi; questo inoltre, in base al protocollo del pacchetto, incrementa il contatore del protocollo di esso (attualmente vengono supportati i protocolli TCP, UDP e ICMP).
Il generico thread "*i*" che gestisce la generica coda circolare "*i*", preleva il pacchetto catturato e:

- nel caso in cui, nella struttura dati contenente i flussi, non vi è traccia di un flusso avente medesima chiave del pacchetto, allora viene inserito in questa;
- altrimenti viene aggiornato il flusso, sommando i byte del pacchetto, incrementando il numero di pacchetti totali e aggiornando il timestamp che indica l'ultima volta in cui è stato aggiornato.

Nel caso in cui l'utente abbia specificato tramite argomenti specifici, di attivare il round robin database, allora viene creato un thread, il quale si occupa di creare un file, con nome passato come argomento al programma, e di salvare i contatori dei pacchetti tramite RRD Tool (http://oss.oetiker.ch/rrdtool/); tramite quest'ultimo verrà creato il grafico riassuntivo del throughtput del totale dei pacchetti catturati e dei protocolli TCP, UDP e ICMP.

I flussi verranno estratti dalla struttura dati che li contiene, tramite un thread, il quale scansiona la struttura e nel caso in cui un flusso sia in questa da più di 60 secondi., oppure che non abbia ricevuto aggiornamenti da più di 60 secondi, allora questo viene eliminato dalla struttura dati e viene posto in due buffer, i quali verranno poi utilizzati per visualizzarli sull'interfaccia web (visualizzazione dei flussi, e visualizzazione tramite mappa della geolocalizzazione degli indirizzi ip di questi).

Un altro thread, si occupa della gestione dell'HTTP server mongoose (http://code.google.com/p/mongoose/).

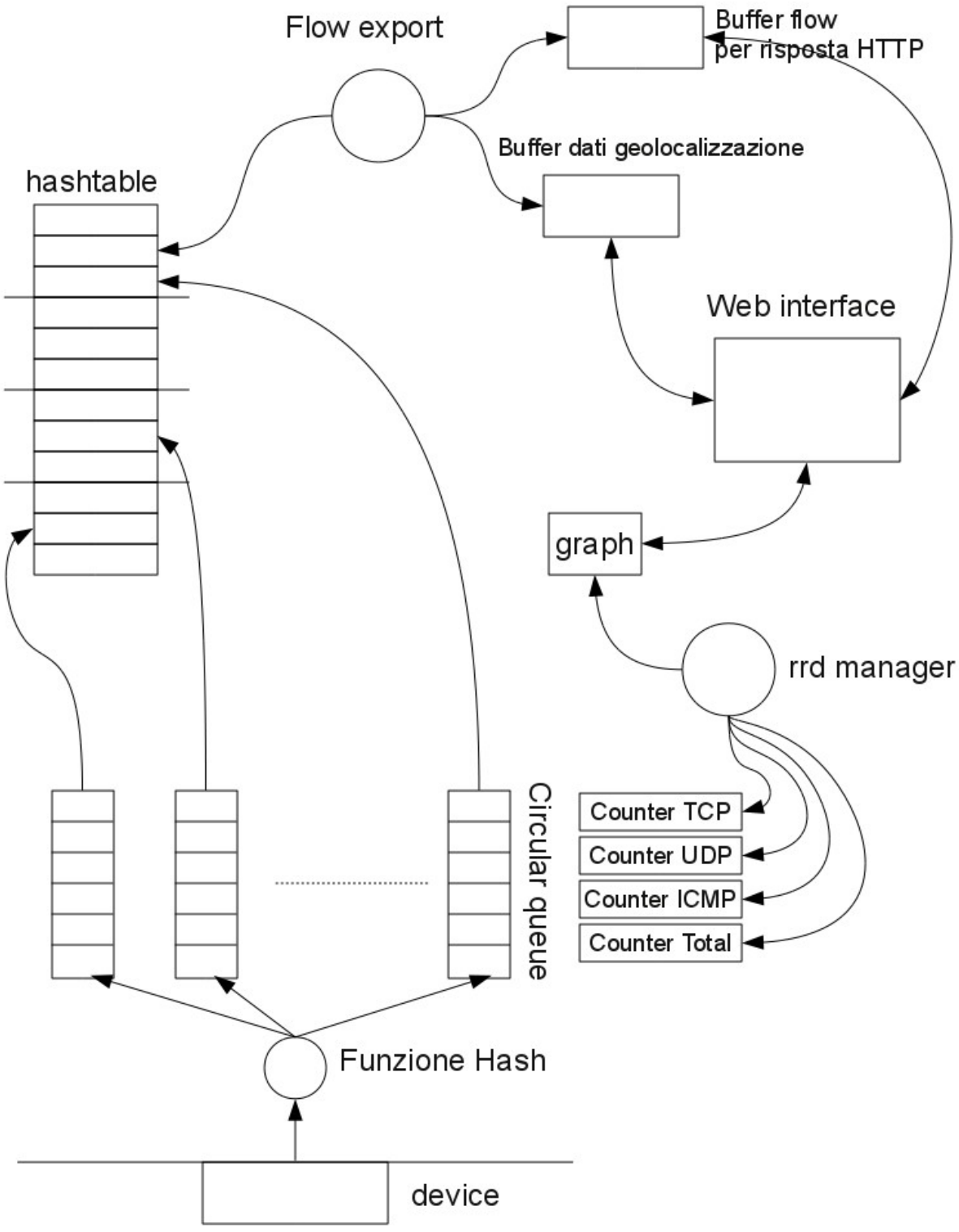
L'utente per poter visualizzare i dati dell'applicazione, potrà sfruttare un client web (attualmente è stato testato firefox). Sono state previste 3 pagine in html:

- nella prima vengono visualizzati i flussi catturati.
  - Lato client:
    il client effettua delle chiamate ajax al server http, il quale risponde con una stringa contenente gli ultimi flussi da visualizzare.
    Ricevuto i dati il client aggiorna la textarea.
  - Lato server:
    per la gestione dei dati da inviare al client, i dati del generico flusso esportato, vengono concatenati in una stringa, la quale funge da buffer. Questa viene "svuotata" dopo di che è stata inviata al client.
- una pagina visualizza la mappa con la geolocalizzazione degli indirizzi ip dei flussi:
  - Lato client:
    anche in questo caso, il client effettua delle chiamate ajax per reperire i dati per aggiornare la mappa.
    La mappa utilizzata è quella offerta da google (http://maps.google.com/), ad ogni dato ricevuto, aggiunge un marker (nel caso in cui non sia già presente) sulla mappa.
  - Lato server:

per prelevare i dati relativi alla geolocalizzazione degli indirizzi i ip, è stata utilizzata la versione “light” (free) di maxmind (http://www.maxmind.com/).
Quando un flusso viene esportato, viene chiamata la funzione per reperire le coordinate geografiche di questo, dopo di che vengono inserite le informazioni nel buffer citato in precedenza.
Quando il client effettua una richiesta, vengono inviati i dati ad esso e svuotato il buffer.

- L'ultima pagina visualizza il grafico prodotto da RRD Tool
  - In questo caso, la pagina in html è statica, quando viene inviato dal server codice HTML della pagina, essa ha già il nome esatto dell'imagine creata.
    Quindi con una funzione in javascritpt, viene fatto il refresh di quest'immagine, creata da RRD Tool.

Nella pagina seguente uno schema di massima dell'architettura dellìapplicazione sviluppata.


Flow export
Buffer flow
per risposta HTTP
Buffer dati geolocalizzazione
hashtable
Web interface
graph
rrd manager
Counter TCP
Counter UDP
Counter ICMP
Counter Total
Circular queue
Funzione Hash
device

Files

- **daniele_sartiano.h – daniele_sartiano.c**

  Questo il file contenente il main, qui vengono fatti partire i threads che gestiranno l'applicazione:
  - il thread che effettua la pcap_loop per catturare i pacchetti;
  - il thread per la gestione di RRD Tool;
  - il thread per che esporta i flussi;
  - N thread che gestiscono le N code contenenti i pacchetti catturati e che li inseriscono o aggiornano nella hashtable;
  - il thread per la gestione dell'HTTP server.

  L'applicazione quando viene eseguita, in base agli argomenti passati, può leggere un file .pcap, catturare da un'interfaccia specificata dall'utente, salvare i dati su un round robin database.

  Nel caso in cui si debba leggere un file .pcap, viene invocata la funzione *pcap_open_offline*, viene così letto il file contenente i dati dei pacchetti catturati e vengono mostrate le informazioni relative ai pacchetti contenuti in questo file.
  Nel caso in cui si debba catturare i dati "live", l'utente deve selezionare un device su cui catturare i pacchetti, dopo di che viene invocata la funzione *pcap_open_live*.
  Il thread cattura i pacchetti che passano sul device selezionato, nel caso essi abbiano protocollo UDP, TCP o ICMP, incrementa i contatori appropriati, calcola la funzione di hash sulla chiave (ip sorgente, porta sorgente, ip destinatario, porta destinataria) ed inserisce il pacchetto nella coda avente indice :

  $$i = hash\_key \% N\_THREAD$$

  In questo modo, il thread associato a quella coda, potrà gestire i flussi relativi a quel pacchetto, senza avere interferenze dagli altri thread, i quali gestiscono le altre code, e quindi aggiornare i dati del flusso nella tabella hash senza meccanismi di locking.
  Il thread associato alla coda con indice "*i*", si occupa di estrarre l'elemento dalla coda, di creare un flusso con i dati di esso e di inserirlo nella tabella hash. Nel caso in cui il flusso fosse già presente nella tabella, allora esso verrà aggiornato, sommando la dimensione del pacchetto catturato ad esso, incrementando il numero di pacchetti e aggiornando il timestamp relativo all'ultimo aggiornamento del flusso, altrimenti verrà inserito in una nuova entry della tabella hash con chiave :

  $$k = hash\_key \% SIZE\_HASHTABLE$$

  Il thread che si occupa di esportare i flussi, controlla ad intervalli regolari la hashtable, nel caso in cui trovi un flusso "scaduto", elimina il flusso dalla tabella, recupera le informazioni per la geolocalizzazione e le inserisce nel buffer apposito, inserisce le informazioni da visualizzare all'utente in un altro buffer.

- **flow.h – flow.c**

  In questi file vengono definite le strutture dati per rappresentare un flusso e le operazioni associate ad esso:

```
typedef struct struct_flow_key {
      in_addr_t ip_src, ip_dst;
      u_int16_t port_src, port_dst;
} flow_key;

typedef struct struct_flow {
      flow_key *key;
      u_int length;
      u_int pkts;
      struct timeval first;
      struct timeval last;
} flow;
```

La chiave di un flusso è composta da ip sorgente, porta sorgente, ip destinario e porta destinataria del pacchetto catturato.
Il flusso inoltre ha le informazioni relative al numero di byte di esso, il numero di pacchetti da cui è composto, il timestamp di quando è stato catturato la prima volta e dell'ultima volta in cui è stato aggiornato.

- **manage_rrd.h – manage_rrd.c**

  In questi files sono definite le operazioni per la gestione di RRD Tool:
  - la creazione del file rrd;
  - la creazione del grafico riassuntivo;
  - la scrittura sul file rrd.

- **flow_queue.h – flow_queue.c**

  La coda circolare definita in questi files, è utilizzata per l'inserimento e l'estrazione dei pacchetti catturati dal processo che effettua la “pcap_loop”.
  La coda è definita come segue:

```
typedef struct struct_element {
      unsigned int hash_key;
      flow* f;
} element;

typedef struct struct_queue {
      element* content[N];
      int front, rear;
      int size;
} queue;
```

  Un elemento della coda è costituito da un flusso e dal risultato della funzione hash calcolata sulla chiave di esso.
  La coda è stata implementata come coda circolare.

- **hashtable.h – hashtable.c**

  Qui viene definita la struttura dati che contiene i flussi catturati.
  Essa è una hashtable lockless, viene utilizzata in lock nel caso in cui una entry di essa viene inserita oppure rimossa, non nel caso in venga aggiornata.

Per la gestione delle collisioni, è stata utilizzata una lista associata ad ogni entry, nel caso di collisione quindi, verrà aggiunto un elemento della hashtable in testa alla lista della entry.

```
struct entry {
	flow *f;
	struct entry *next;
};

struct hashtable {
	int number_entry;
	int size;
	struct entry **table;
};
```

Una entry della tabella è composta da un flusso e dal puntatore alla prossimo elemento della lista.

- **flow_export.h – flow_export.c**

  Qui vengono definite le operazioni che effettua il thread che dovrà esportare i flussi dalla tabella hash.
  Esso scansiona la tabella hash ad intervalli regolari, cercando i flussi "scaduti", quando ne incontra uno, lo elimina dalla tabella hash, inserisce le informazioni per la visualizzazione delle informazioni sul flusso tramite http server in una stringa, recupera le informazioni per la geolocalizzazione e li inserisce nel buffer opportuno.

- **mongoose.h – mongoose.c**

  Questi file implementano l'HTTP server embedded moongose, sono stati prelevati ed inseriti semplicemente nel progetto.

Pagine web

Per la visualizzazione dei dati, è stato scelto di utilizzare un browser; la porta su cui è aperto il server è la 8000.
Le pagine attuali sono:

- **http://localhost:8000**

  In questa pagine vengono visualizzati i flussi esportati.
  Vi è una textarea, in cui vengono appesi i dati reperiti tramite richieste ajax all'HTTP server (in questo caso aperto sulla porta 8000).
  Di seguito lo script in javascript per effettuare le richieste al server:

```
setTimeout("makeRequest()", 3000);

var http_request = false;

function makeRequest() {
	http_request = false;
	if (window.XMLHttpRequest) {
		http_request = new XMLHttpRequest();
		if (http_request.overrideMimeType)
```

```
                {http_request.overrideMimeType('text/html');
            }
        } else if (window.ActiveXObject) {
            try {
                http_request = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (e) {
                try { http_request = new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e) {
                }
            }
        }
        if (!http_request) {
            alert('Cannot create XMLHTTP instance');
            return false;
        }
        http_request.onreadystatechange = alertContents;
        http_request.open('GET', 'get_data', true);
        http_request.send(null);
        setTimeout("makeRequest()", 3000);
}

function alertContents() {
        if (http_request.readyState == 4) {
            if (http_request.status == 200) {
                result = http_request.responseText;
                document.getElementById('textarea_ajax').value += result;
                document.getElementById('textarea_ajax').scrollTop =
                        document.getElementById('textarea_ajax').scrollHeight;
            } else {
                alert('There was a problem with the request.');
            }
        }
}
```

In questo caso, il client effettua delle richieste ogni 3 secondi.

- **http://localhost:8000/map**

  In questa pagina, viene visualizzata una mappa, nella quale vengono inseriti i vari marker, corrispondenti alle coordinate degli indirizzi ip, recuperate tramite la libreria MaxMind. Nel caso in cui la localizzazione di un flusso fosse già visualizzata, il marker associato non viene aggiunto alla mappa.
  Di seguito il codice javascript:

```
var map = null;
var markers = new Array();
function initialize() {
        if (GBrowserIsCompatible()) {
            map = new Gmap2(document.getElementById("map_canvas"));
            map.setCenter(new GLatLng(37.4419, -122.1419), 2);
             var customUI = map.getDefaultUI();map.setUI(customUI);
        }
}

function showAddress(address) {
        var lan_lon = address.split(',');
        var end = 0;var i = 0;
        while(end != 1) {
            if(lan_lon[i+2] != undefined && lan_lon[i] != '-' ) {
                var marker = createMarker(new GLatLng(parseFloat(lan_lon[i]),
                            parseFloat(lan_lon[i+1])), lan_lon[i+2]);
```

```
                        if(!contains(marker)) {
                                markers.push(marker);
                                map.addOverlay(marker);
                        }
                        document.getElementById('info_marker').innerHTML =
marker.getLatLng().toString() + ' ' + marker.getTitle();
                        i+=3;
                }
                else  {
                        end = 1;
                }
        }
}

function contains(element){
        for(var index=0;index < markers.length;index++) {
                if(markers[index].getLatLng().equals(element.getLatLng()) ) {
                        return true;
                }
        }
        return false;
}

function createMarker(point, ip_address) {
        var marker = new GMarker(point, {title: ip_address});
        marker.value = ip_address;
        GEvent.addListener(marker, "click", function() {
                var myHtml = "<b>" + ip_address + "</b>" ;
                map.openInfoWindowHtml(point, myHtml);
        });
        return marker;
}

setTimeout('makeGeoRequest()', 5000);

var http_geo_request = false;

function makeGeoRequest() {
        http_geo_request = false;
        if (window.XMLHttpRequest) {
                http_geo_request = new XMLHttpRequest();
                if (http_geo_request.overrideMimeType {
                        http_geo_request.overrideMimeType('text/html');
                }
        } else if (window.ActiveXObject) {
                try {
                        http_geo_request = new ActiveXObject("Msxml2.XMLHTTP");
                } catch (e) {
                        try {
                                http_geo_request = new ActiveXObject("Microsoft.XMLHTTP");
                        } catch (e) {}
                }
        }
        if (!http_geo_request) {
                alert('Cannot create XMLHTTP instance');
                return false;
        }
        http_geo_request.onreadystatechange = reloadMap;
        http_geo_request.open('GET', 'get_geo_data', true);
        http_geo_request.send(null);
        setTimeout('makeGeoRequest()', 5000);
}

function reloadMap() {
```

```
        if (http_geo_request.readyState == 4) {
                if (http_geo_request.status == 200) {
                        result = http_geo_request.responseText;
                        if(result != "") {
                                document.getElementById('address_span').innerHTML = result;
                                showAddress(result);
                        }
                } else {
                        alert('There was a problem with the request.');
                }
        }
}
```

Il client in questo caso effettua le richieste al server ogni 5 secondi.
Nel caso in cui non sia installato il DB GeoIPCity.dat, (scaricabile all'indirizzo http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz) allora verrà visualizzato un messaggio con il link con cui poterlo scaricare.

- **http://localhost:8000/graph**

  In questa pagina viene visualizzato il grafico prodotto da RRD Tool, il quale visualizza i throughput dei pacchetti con protocollo TCP, UDP, TCP e il totale di questi.
  Il codice javascript, si occupa di fare il refresh dell'immagine ad intervalli regolari.
  Di seguito il codice:

```
var image;
var c = 0;
var first = 0;
function count(nameFile){
        if(first == 0){
                first = 1;
                document.getElementById('refresh').style.display = '';
        }
        image.src=nameFile+(++c);
}
function init(nameFile) {
        image = document.getElementById('refresh');
        if( image ) {
                var fun = "count('" + nameFile + "')";
                setInterval(fun,5000);
        }
}
```

  L'immagine viene ricaricata ogni 5 secondi.
  Nel caso in cui, l'utente non abbia specificato l'uso di RRD Tool, allora verrà visualizzato un semplice messaggio con scritto che RRD Tool non è abilitato.

Utilizzo:

L'applicazione accetta i seguenti parametri:

- -i
  tramite questo parametro è possibile catturare i pacchetti “live”. Verrà poi richiesto di selezionare il device su cui catturare i pacchetti.
- -f <nome_file>.pcap

vengono letti i pacchetti salvati sul file passato come parametro.

- -r <nome_file>.rrd
  viene prodotto un round robin database con il nome specificato e attivate le funzionalità descritte nel file manage.c.
- -h
  viene mostrato l'help con i parametri appena descritti.

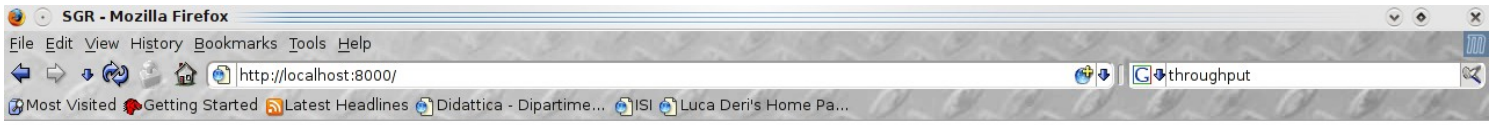
Esempio

```
# ./daniele_sartiano -i -r nome.rrd
1. eth0 (No description available)
Address: 2.0.0.0
2. wmaster0 (No description available)
Address: 3.0.0.0
3. wlan0 (No description available)
Address: 4.0.0.0  192.168.1.100  0.0.0.0
4. usb1 (USB bus number 1)
5. usb2 (USB bus number 2)
6. usb3 (USB bus number 3)
7. usb4 (USB bus number 4)
8. usb5 (USB bus number 5)
9. usb6 (USB bus number 6)
10. usb7 (USB bus number 7)
11. lo (No description available)
Address: 1.0.0.0  127.0.0.1  0.0.0.0
Select interface:
3

rrdtool create nome.rrd --step 3 DS:total_pkt:COUNTER:5:0:1000000
RRA:AVERAGE:0.5:1:3600 DS:tcp_pkt:COUNTER:5:0:1000000 RRA:AVERAGE:0.5:1:3600
DS:udp_pkt:COUNTER:5:0:1000000 RRA:AVERAGE:0.5:1:3600
DS:icmp_pkt:COUNTER:5:0:1000000 RRA:AVERAGE:0.5:1:3600
Immagine file: nome.png

1 -- 1248183188:321416/tcp/64.233.161.83:80/192.168.1.100:38666
```

In questo caso viene attivata la cattura live dall'interfaccia *wlan0*, e viene prodotto un round robin database di con nome "*nome.rrd*", dopo di che vengono visualizzate delle informazioni sui pacchetti catturati sull'output.
Collegandosi tramite browser sui seguenti indirizzi:

- http://localhost:8000
- http://localhost:8000/map
- http://localhost:8000/graph

è possibile visualizzare le informazioni, di seguito degli snapshot che illustrano la visualizzazione dei dati.

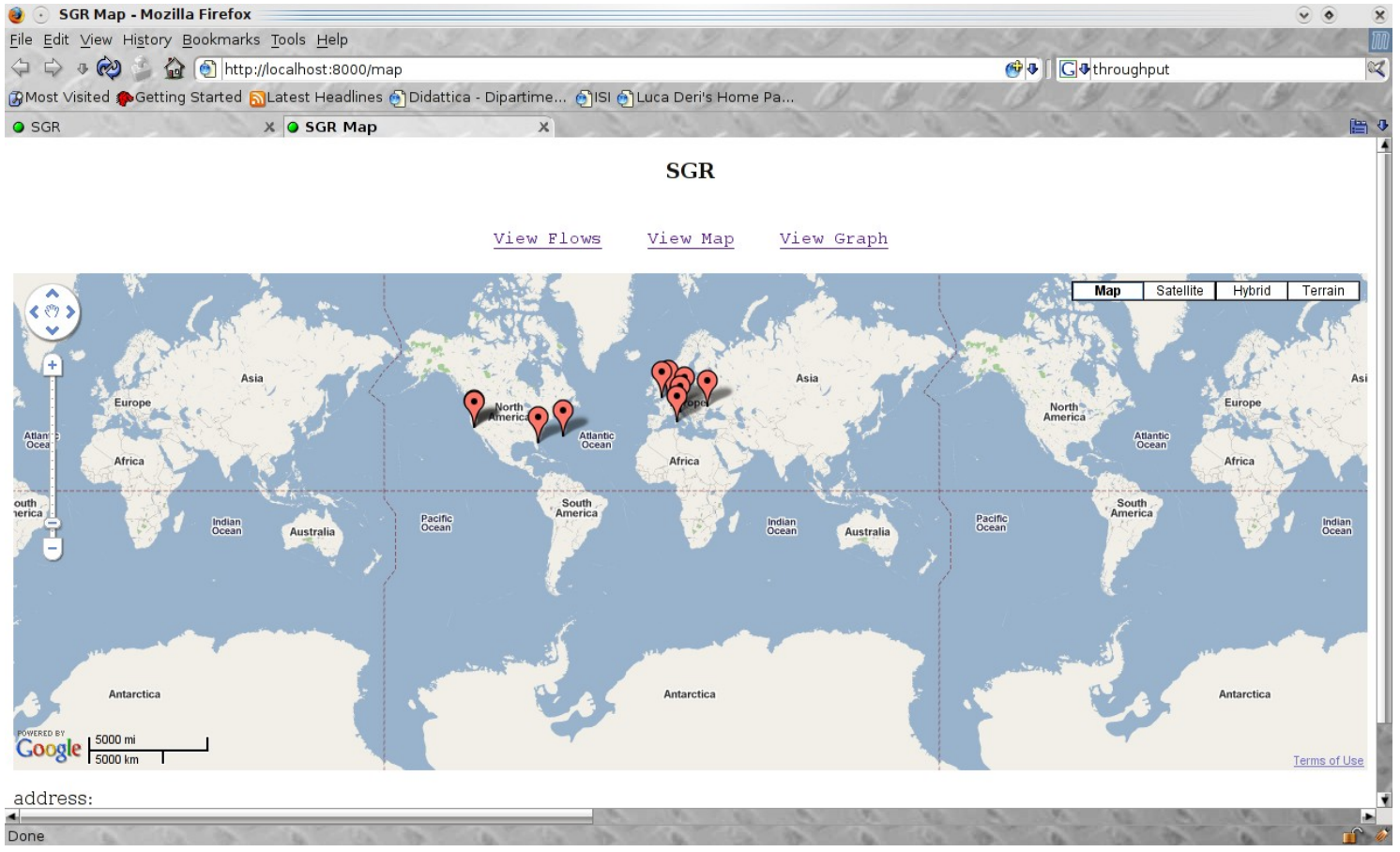


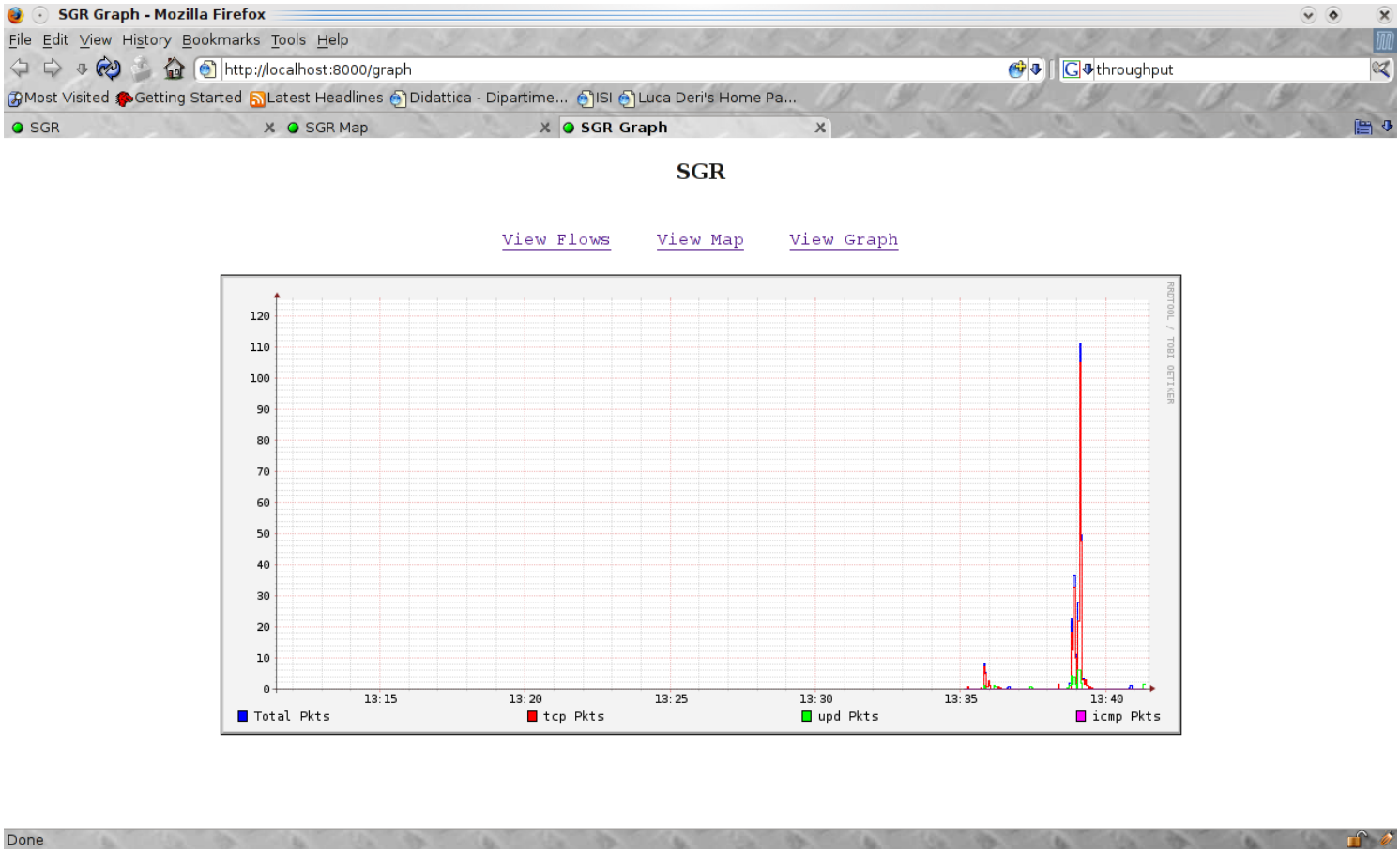
SGR

View Flows View Map View Graph

ip_src:ip_port	ip_dst:port_dst	total_bytes	packets_number
192.168.1.100:56932	64.233.161.83:80	66	1
64.233.161.83:80	192.168.1.100:56932	94	1
208.67.222.222:53	192.168.1.100:39089	102	1
192.168.1.100:39089	208.67.222.222:53	75	1
208.67.222.222:53	192.168.1.100:51555	134	1
192.168.1.100:51555	208.67.222.222:53	75	1
192.168.1.100:56932	64.233.161.83:80	240	4
64.233.161.83:80	192.168.1.100:56932	297	4
64.233.161.83:80	192.168.1.100:47039	16269	21
192.168.1.100:47039	64.233.161.83:80	8533	23
151.51.12.248:5766	192.168.1.100:42774	84	1
192.168.1.100:42774	151.51.12.248:5766	158	1
78.22.105.67:57509	192.168.1.100:42774	62	1
192.168.1.100:42774	78.22.105.67:57509	161	1
87.6.92.165:24482	192.168.1.100:42774	62	1
192.168.1.100:42774	87.6.92.165:24482	161	1
90.146.245.179:4830	192.168.1.100:34332	70	1
192.168.1.100:34332	90.146.245.179:4830	395	2
72.188.226.147:4736	192.168.1.100:42774	62	1
192.168.1.100:42774	72.188.226.147:4736	160	1
199.172.208.160:1920	192.168.1.100:42774	62	1
192.168.1.100:42774	199.172.208.160:1920	160	1

Done





Sviluppi futuri

Sarebbe possibile fare un file di configurazione dove poter specificare alcune opzioni da caricare a runtime, come ad esempio:

- numero di thread (e di conseguenza, numero di code circolari) che gestiscono le code circolari contenenti i pacchetti catturati;
- il tempo necessario per far scadere un flusso;
- il tempo in cui aggiornare il file rrd;
- la porta su cui è aperto il server HTTP;
- ecc...

Si potrebbero supportare altri protocolli oltre a TCP, UDP e ICMP.
Si potrebbero creare delle statistiche più dettagliate sui flussi, inoltre sarebbe possibile visualizzare più informazioni di essi.