Esame di Gestione di Rete

Relazione progetto

“Monitoraggio di traffico in rete locale.”

Simone Citi matr. 304618
Luca Silva matr. 241715
Daniele Gavazzi matr. 252201

Introduzione

L'idea di questo progetto nasce dall'esigenza di mettere in pratica alcune nozioni apprese durante il corso. Dopo una rapida analisi su che cosa avremmo voluto sviluppare abbiamo deciso di creare un applicativo per il monitoraggio di rete su porte locali utilizzate da applicazioni molto diffuse quali il client P2P eMule, il client Torrent ed il client IRC; ma anche sulle porte dei protocolli che supportano la maggior parte del traffico domestico quali la porta 80 per il web, la porta 22 per il protocollo di sicurezza SSH e la porta 25 usata dal protocollo di posta SMTP. Questo in virtù del fatto che sarebbero state interessanti per la quantità di dati generata.
Dopo aver verificato di cosa avessimo bisogno per lo sviluppo abbiamo pensato di adattare l'hardware in nostro possesso, di utilizzare una piccola rete locale domestica dotata di una connessione ADSL, con un' elevata attività e diversità di accessi al web e quindi una mole di dati interessante, e infine sviluppare il software.
Il progetto mostra inoltre le potenzialità di applicazioni quali nProbe per il monitoraggio del traffico di rete locali attraversate da grandi o piccole quantità di dati.
Nel particolare nProbe, assieme ai grafici di RRD Tool, ci mostra come sia diverso il comportamento del traffico quando si tratta di semplici applicazioni o scambio di dati in rete (come rete si intende comunque la rete globale internet) rispetto al download di file con dimensioni elevate.
Infine nel grafico relativo ai servizi peer to peer come Torrent o eMule vediamo parecchi picchi con elevato dislivello dovuti alla forte irregolarità del traffico, mentre se vediamo il grafico del servizio IRC (che è praticamente il grafico prodotto dall'analisi dela chat dell'applicazione mIRC) notiamo che il grafico è molto più regolare e ci sono solo pochi picchi con elevato dislivello.

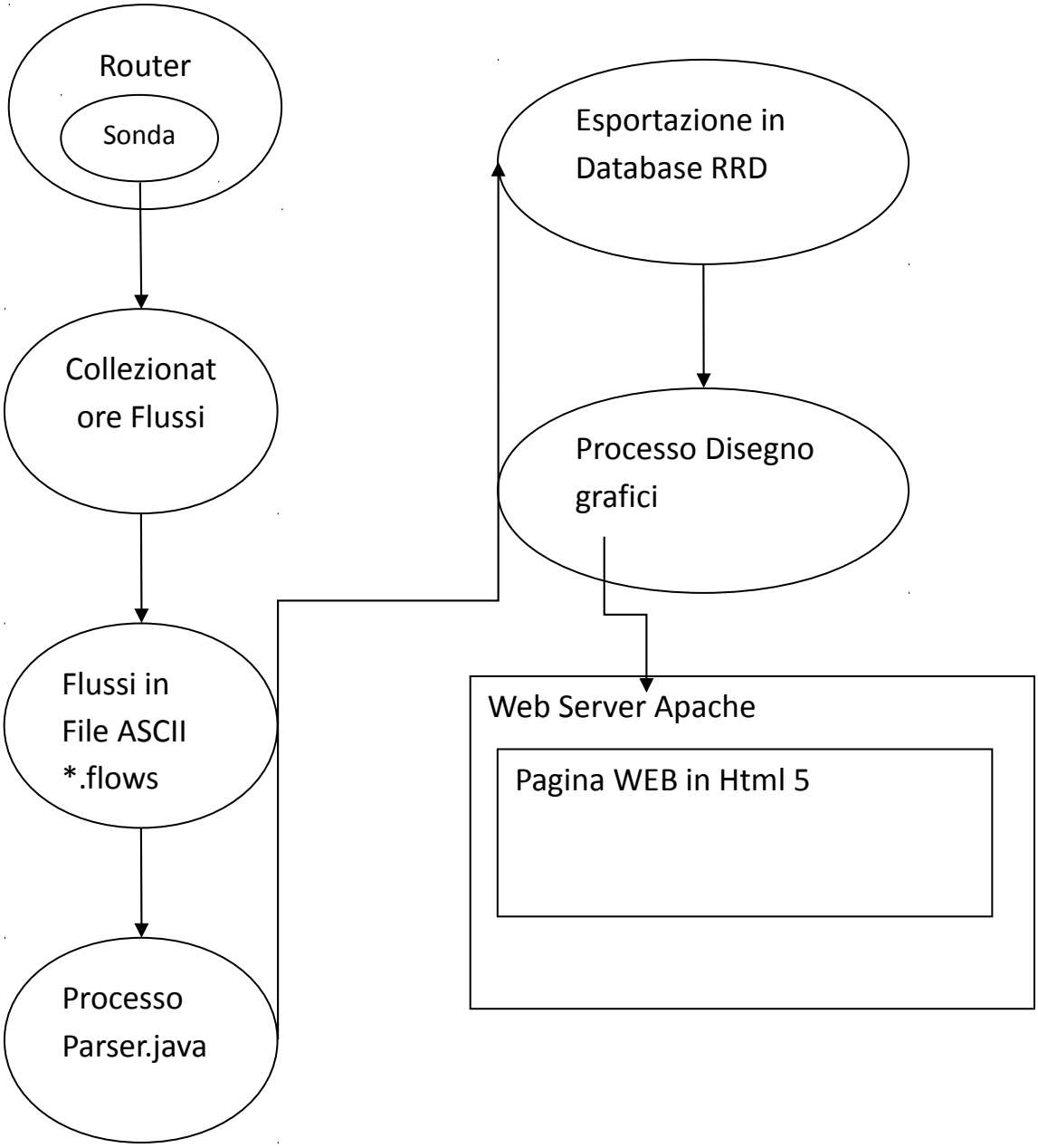
Scelte Progettuali

Per catturare il traffico della rete locale è stato utilizzato un modem-router Mikrotik RB533R5. Questo dispositivo, una volta configurato opera da router per l'instradamento su linea ADSL ed anche da router per la gestione del traffico all'interno della rete stessa. Questo dispositivo è stato scelto in quanto esempio di prodotto di basso costo (molto inferiore rispetto ai professionali CISCO) che garantisce una gamma completa di funzionalità ad elevate prestazioni; a dimostrazione di ciò è uno dei pochi modelli di router sul mercato (sempre per una fascia di prezzo medio bassa) dotato di un modulo per la raccolta di flussi NetFlow.
Per l'analisi dei flussi raccolti dal Mikrotik è stato utilizzato il software nProbe visto durante il corso e messo a disposizione da parte del professor Luca Deri; il software è stato utilizzato nella versione 6 e configurato in modo da funzionare in "modalità Probe". nProbe ci ha permesso di estrarre dai flussi NetFlow i dati a noi necessari e trasferirli in file di testo (con estensione ".flow") organizzati secondo il template di flusso da noi configurato. Tutti i flussi utilizzati rispecchiano lo standard NetFlow versione 9 che, a differenza delle versioni precedenti che hanno un loro specifico formato non modificabile, è dinamico e aperto alle estensioni.
Di seguito riportiamo il template utilizzato:

%L4_DST_PORT %IPV4_SRC_ADDR %IPV4_DST_ADDR %IN_PKTS %IN_BYTES %L4_SRC_PORT %PROTOCOL %FIRST_SWITCHED %LAST_SWITCHED %HTTP_URL

Per la lettura dei dati è stato scelto di implementare un parser in linguaggio Java che si occupa di leggere con periodicità oraria i file di testo contenenti i dati dei flussi ordinati secondo la linea temporale; i dati così raccolti dal parser vengono formattati per essere poi inseriti negli archivi del database RRD.
RRD Tool è particolarmente indicato per questo tipo di uso perché ci permette di archiviare grandi quantità di dati in pochi MegaByte, ed a organizzare i dati secondo un ordine cronologico (in un dato istante ho solo un dato che nel nostro caso indica la quantità di dati trasferiti).
Altra comoda caratteristica del database da noi sfruttata è la possibilità di generare vari tipi di grafici dei dati archiviati. Per la gestione e la sincronizzazione delle applicazioni utilizzate (nProbe, scrittura grafici, Parser) sono stati utilizzati dei semplici script Bash eseguiti sul server Altanis.

In figura il diagramma di flusso del progetto:



Configurazione Mikrotik per raccolta dati

Per raccogliere i dati di traffico della rete Altanis è stato impiegato una Routerboard Mikrotik serie RB533R5 costituito da 3 porte ethernet e 2 interfacce wireless (disabilitate); il dispositivo ci ha permesso di collezionare i dati e raccoglierli in flussi NetFlow.

Il Mikrotik ha la possibilità di essere configurato attraverso 2 interfacce:

- il configuration Tool proprietario Winbox
- l'interfaccia SSH

il dispositivo è progettato in moduli indipendenti gestiti da package che è possibile attivare o disattivare in base alle esigenza tecniche dell'utilizzo finale.

In particolare i package da noi installati:

```
[admin@MikroTik] > system package print
Flags: X - disabled
 #   NAME                                                                                    VERSION
SCHEDULED
 0   system                                                                                  3.11
 1 X mpls                                                                                    3.11
 2   routerboard                                                                             3.11
 3   routeros-mipsle                                                                         3.11
 4   hotspot                                                                                 3.11
 5   wireless                                                                                3.11
 6 X ipv6                                                                                    3.11
 7   routing                                                                                 3.11
 8   security                                                                                3.11
 9   dhcp                                                                                    3.11
10   ppp                                                                                     3.11
11   advanced-tools                                                                          3.11
```

L'abilitazione del package 'routing' ci permette di accedere alle funzioni per la raccolta flussi NetFlow.

Il dispositivo utilizzato come gateway nella rete ha le seguenti configurazioni:

1) Interfaccia PPPOE su Router ADSL configurato in bridging (connessione ADSL Libero):

```
[admin@MikroTik] > interface pppoe-client print
Flags: X - disabled, R - running
0  R name="pppoe-out1" max-mtu=1480 max-mru=1480 mrru=disabled interface=ether1 user="benvenuto"
password="ospite" profile=default service-name="WIND" ac-name="" add-default-route=yes dial-on-demand=no
use-peer-dns=no allow=pap,chap,mschap1,mschap2
```

2) Indirizzi ip:

```
[admin@MikroTik] > ip address print
Flags: X - disabled, I - invalid, D - dynamic
#   ADDRESS            NETWORK         BROADCAST       INTERFACE
 0   192.168.51.1/24    192.168.51.0    192.168.51.255  ether2
 1   192.168.1.2/24     192.168.1.0     192.168.1.255   ether1
 2 D 151.40.243.10/32   151.6.144.72    0.0.0.0         pppoe-out1
[admin@MikroTik] >
```

L'indirizzo con id 2 è quello dinamico assegnato dall'operatore

3) Le rotte:

```
[admin@MikroTik] > ip route print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
 #      DST-ADDRESS        PREF-SRC        G GATEWAY                  DISTANCE INTERFACE
 0 ADS  0.0.0.0/0                          r 151.6.144.72             1        pppoe-out1
 1 ADC  151.6.144.72/32    151.40.243.10                              0        pppoe-out1
 2 ADC  192.168.1.0/24     192.168.1.2                                0        ether1
 3 ADC  192.168.51.0/24    192.168.51.1                               0        bridge1
```

nella rotta con id 3 l'interfaccia virtuale bridge1 è costituita da un L2 virtuale tra ether1 e ether2 (bridging effettuato con software proprietario Mikrotik).

Configurazione dei Flussi NetFlow.

Come valori di default abbiamo impostato il collezionatore in ascolto su tutte le interfacce (ether1-3 e pppoe-out1).

I parametri configurabili sono 3:

- Cache entries: la quantità di memoria utilizzata dai flussi attivi, impostata a 4kB\...)
- Active flow timeout: la durata massima che può avere un flusso, impostata a 2 min
- Inactive flow timeout: la durata massima del tempo di inattività di un flusso impostata a 15 sec

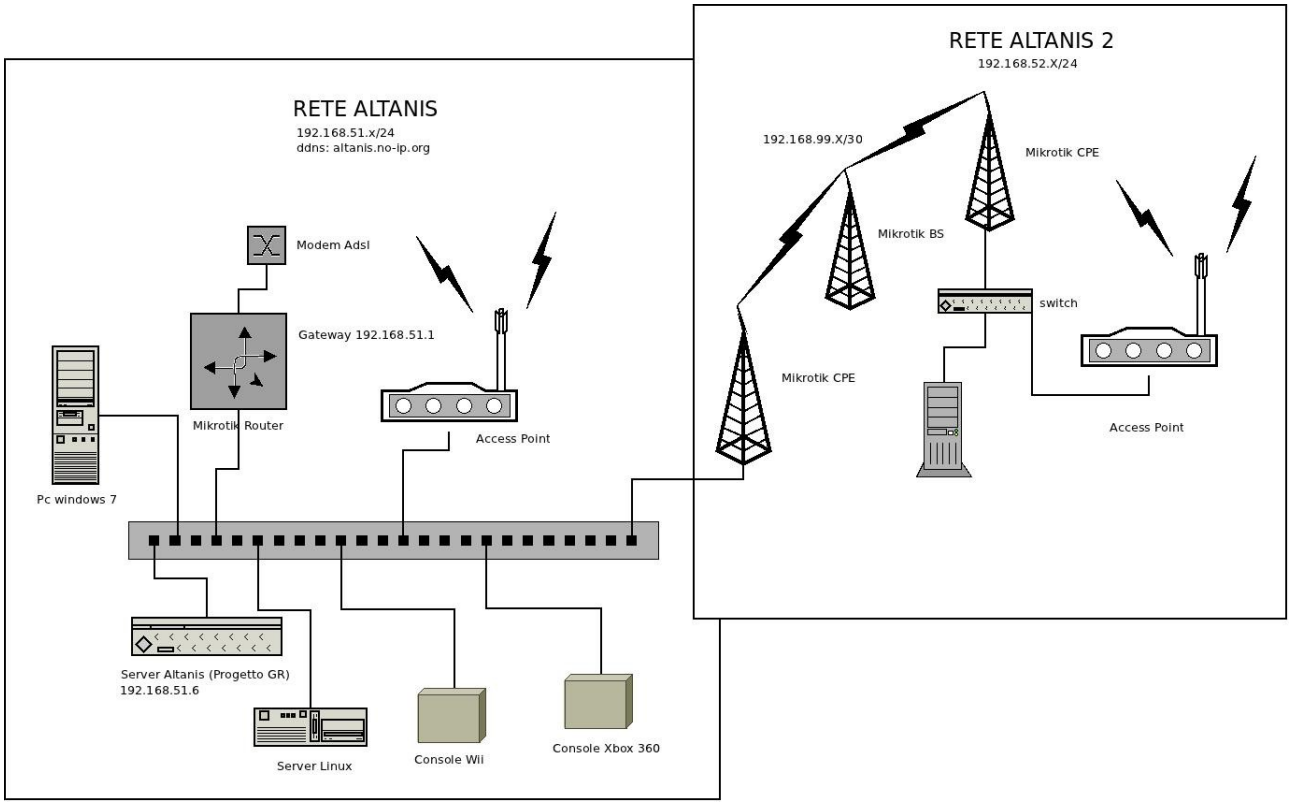
Comando:

```
[admin@MikroTik] > ip traffic-flow set enabled=yes cache-entries=4k active-flow-timeout=00:02:00 inactive-flow-
timeout=00:00:15
```

Inoltre abbiamo configurato il Mikrotik per inviare i flussi collezionati al collezionatore all'indirizzo interno 192.168.51.6 dove è presente il demone nProbe in ascolto sulla porta 2055

```
[admin@MikroTik] > ip traffic-flow target add address=192.168.51.6:2055 version=9
[admin@MikroTik] > /ip traffic-flow target print
Flags: X - disabled
 #    ADDRESS               VERSION
 0    192.168.51.6:2055     9
```

Di seguito la struttura della rete locale impiegata nel progetto:



Protocollo netFlow in Breve

NetFlow è un protocollo di rete Sviluppato da Cisco per il monitoraggio del traffico IP (RFC 3954); I dati vengono raccolti in flussi di rete dove ognuno dei quali è definito come un insieme di pacchetti legati da un insieme comune di proprietà (ad esempio hanno stesso indirizzo IP e la stessa porta).

La versione v9 (adottata in questo progetto) a differenza delle precedenti rimpiazza i flussi unidirezionali con flussi bidirezionali definiti da un template inviato periodicamente per decodificare il flusso, applica inoltre una numerazione di sequenza per i flussi. Esempio di Tipi di dato di un Template:

1. Sorgente IP address
2. Destinazione IP address
3. Porta Sorgente
4. Porta Destinazione
5. Protocollo ip

Questa specifica implementazione di flussi basati su Template non era presente nelle versioni precedenti.

L’applicazione nProbe

nProbe è un’applicazione sviluppata da “ntop project” che viene utilizzata per sondare il traffico di rete o per collezionare flussi in standard netFlow provenienti da un’altra sonda. Dei tre possibili utilizzi dell’applicazione (Probe, Collector e Proxy) sono state sfruttate le caratteristiche di Probe che ci ha permesso di caricare i flussi standard NetFlow collezionati dal router Mikrotik e estrarne i dati a noi necessari secondo il template richiesto a nProbe stesso. Il comando usato per eseguire nProbe viene eseguito da script Bash assieme ad altri comandi che servono a far partire tutti i servizi necessari.
Di seguito il comando utilizzato per lanciare in esecuzione l’applicazione da shell linux:

```
Bash~:/root/nprobe_6.5.1_062711/.libs/lt-nprobe -b 2 -i none -3 2055 -n none -T "%L4_DST_PORT
%IPV4_SRC_ADDR %IPV4_DST_ADDR %IN_PKTS %IN_BYTES %L4_SRC_PORT  %PROTOCOL %FIRST_SWITCHED
%LAST_SWITCHED %HTTP_URL " -P /testnProbe/ >/dev/null & echo $! > /var/www/pids/nprobe.pid
```

Con questo comando nProbe è impostato per creare ogni 60 secondi su disco dei file formato *.flows contenenti i dati collezionati in base al template impostato; nProbe organizza i file in modo gerarchico in cartelle annidate secondo lo schema /testnprobe/anno/mese/giorno/minuto.flows che semplifica l’accesso e l’interrogazione sequenziale dei file del Parser.

Database RRD

RRDtool (Round Robin Database Tool) è un sistema opensource per memorizzare e visualizzare dati di serie temporali (ad esempio la larghezza di banda di rete,temperatura, carico medio dei server). Memorizza i dati in modo molto compatto, aggregando i dati in base al timestamp in modo da mantenere dimensioni gestibili l’archivio. RRDtool è dotato di un comando per la creazione di grafici utili per l’analisi dei dati.
Come già anticipato abbiamo sfruttato la capacità di memorizzare i dati aggregandoli in base al tempo di raccolta per rappresentare la quantità di byte trasmessi al minuto.
I database creati dallo script sonda.sh (al primo lancio) vengono successivamente popolati dal java Parser tramite una serie di chiamate del comando rrdupdate.

Parser

Il Parser è l’applicazione java che si occupa di leggere i flussi NetFlow standard, salvati dal collezionatore nProbe in file ASCII *.flows, e elaborarne i dati contenuti in modo da raggrupparli in base al numero della porta di rete.
L’esecuzione del Parser avviene ogni ora sui file scritti su hard disk da nProbe (un file ogni minuto di traffico raccolto) nell’ultima ora, ed ogni volta che laprocedura ha inizio viene eseguito per ogni porta da controllare (nel nostro caso 7 porte, quindi 7 esecuzioni di Parser). Ad ogni esecuzione vengono letti i file, da ognuno di questi viene estratto il TimeStamp e la quantità di byte riscontrati sulla porta richiesta al momento del lancio dell'applicazione (parametro del comando); la coppia TimeStamp-bytes viene poi utilizzata per creare una stringa che verrà usata per lanciare dal programma stesso un comando bash su console di sistema; il comando provocherà la scrittura dei dati nel database RRD come coppia che indica la quantità di dati trasmessi sulla porta (bytes) nel determinato tempo (Timestamp).

```
Comando generato sulla porta 80 alle ore 17.25 del 10/11/2011:

Bash~:java Parser 80
Lancio il comando ->rrdtool update /var/www/80.rrd 1320938700:381 1320938760:0 1320938820:0
1320938880:0 1320938940:19720 1320939000:21437 1320939060:364 1320939120:3670 1320939180:52
1320939240:0 1320939300:0 1320939360:0 1320939420:0 1320939480:0 1320939540:0 1320939600:5503
1320939660:104 1320939720:104 1320939780:0 1320939840:19609 1320939900:187372 1320939960:90953
1320940020:112876 1320940080:2703 1320940140:49492 1320940200:69527 1320940260:59565
1320940320:34076 1320940380:4868 1320940440:5576 1320940500:13561 1320940560:9047 1320940620:3689
1320940680:7375 1320940740:9614 1320940800:804 1320940860:11361 1320940920:104 1320940980:10693
1320941040:7877 1320941100:156 1320941160:10064 1320941220:52 1320941280:13506 1320941340:11747
1320941400:104 1320941460:15146 1320941520:65108 1320941580:11614 1320941640:63012 1320941700:116806
1320941760:0 1320941820:0 1320941880:866 1320941940:108387 1320942000:9704 1320942060:2172
1320942120:624 1320942180:6903
```

Al termine dell'esecuzione di Parser troviamo quindi i database RRDTool delle varie porte popolato dai dati raccolti e pronti ad essere utilizzati per la presentazione all'utente.

Di seguito il codice in linguaggio Java del del Parser (Parser.java):

```
import java.io.*;
import java.sql.Timestamp;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;


public class Parser{

//OVERVIEW: si definisce I due formati che dovranno avere i dati di tipo Data utilizzati in seguito:
//            AAAA/MM/GG e MM/GG.

        private static final SimpleDateFormat monthDayYearformatter = new
                                                SimpleDateFormat("yyyy/MM/dd/HH/mm");
        private static final SimpleDateFormat monthDayformatter = new SimpleDateFormat("MMMMM dd");

        public static String timestampToMonthDayYear(Timestamp timestamp) {

//EFFECTS: se il dato in ingresso è un oggetto di tipo Timestamp non nullo ritorna una
//            stringa con il formato data AA/MM/GG separato dal carattere "/" e relativo
//            al timestamp in ingresso.


                if (timestamp == null) {
                        return null;
                }else{
                        return monthDayYearformatter.format((java.util.Date) timestamp);
                }
        }

        public static String timestampToMonthDay(Timestamp timestamp) {

//EFFECTS: se il dato in ingresso è un oggetto di tipo Timestamp non nullo ritorna una
//            stringa con il formato data MM/GG separato dal carattere "/" e relativo
//            al timestamp in ingresso; l'anno viene scartato.

                if (timestamp == null) {
                        return null;
                }else {
                        return monthDayformatter.format((java.util.Date) timestamp);
                }
        }

        public static java.sql.Timestamp getTimestamp() {

//EFFECTS: ritorna il Timestamp relativo alla data-ora attuale (al momento della
//            chiamata) ottenuta dal sistema.


                java.util.Date today = new java.util.Date();
                return new java.sql.Timestamp(today.getTime());
        }


        public static Timestamp dateToTimestamp (String data){

//REQUIRES: "data" deve contenere il formato stringa valido di una data.
//EFFECTS: dateToTimestamp ritorna il Timestamp relativo alla stringa passata come
//            parametro.

                Timestamp timeStampDate=null;
                try {
                        DateFormat formatter;
                        Date date;
                        formatter = new SimpleDateFormat("yyyy/MM/dd/HH/mm");
                        date = (Date)formatter.parse(data);
                        timeStampDate = new Timestamp(date.getTime());
                }catch (ParseException e) {
                        e.printStackTrace();
                }
                return timeStampDate;
        }
```

```
	public static String FlowsToRRD (String file){

//REQUIRES: file deve corrispondere al nome di un file esistente sul File System della
//            macchina ospitante l'applicazione.
//OVERVIEW: Legge il file passato come parametro una riga per volta tramite un ciclo.
//            La linea letta da readLine() viene separata in stringhe che vengono
//            memorizzate ordinatamente nel vettore token[] (vengono esclusi i caratteri
//            '|'); alla posizione token[4] possiamo leggere la quantità di dati
//            trasmessi, in token[0] la porta di transito.
//            bytes contiene la somma dei dati riscontrati su tutte le porte.
//EFFECTS: la funzione ritorna una stringa composta dal timestamp del dato (preso da
//            file) e il valore della variabile bytes.

		File doc=new File("/testnProbe/"+file);
		int bytes = 0;
		try {
			FileInputStream fis = new FileInputStream(doc);
			InputStreamReader isr=new InputStreamReader(fis);
			BufferedReader br=new BufferedReader(isr);

			String line;
			String[] token;
			boolean EOF=false;

			while (!EOF){
				line=br.readLine();
				if (line != null){
					token = line.split("[|]");
					if(!token[4].equals("IN_BYTES"))
						bytes = bytes + Integer.parseInt(token[4]);
				}else EOF=true;
			}
				isr.close();
			}catch (Exception e1) {
				e1.printStackTrace();
			}

			return " "+(Parser.dateToTimestamp(file).getTime()/1000)+":"+bytes;
		}
	}

	public static String FlowsToRRD (String file, String port){

//REQUIRES: file deve corrispondere al nome di un file esistente sul File System della
//            macchina ospitante l'applicazione.
//            port deve essere una stringa corrispondente ad un numero.
//OVERVIEW: Legge il file passato come parametro una riga per volta tramite un ciclo.
//            La linea letta da readLine() viene separata in stringhe che vengono
//            memorizzate ordinatamente nel vettore token[] (vengono esclusi i caratteri
//            '|'); alla posizione token[4] possiamo leggere la quantità di dati
//            trasmessi, in token[0] la porta di transito.
//            bytes contiene la somma dei dati riscontrati sulla porta token[0].
//EFFECTS: la funzione ritorna una stringa composta dal timestamp del dato (preso da
//            file) e il valore della variabile bytes.

		File doc=new File("/testnProbe/"+file);
		int bytes = 0;

		try {
			FileInputStream fis = new FileInputStream(doc);
			InputStreamReader isr=new InputStreamReader(fis);
			BufferedReader br=new BufferedReader(isr);
			String line;
			String[] token;
			boolean EOF=false;

			while (!EOF){
				line=br.readLine();
				if (line != null){
					token = line.split("[|]");
					if (token[0].equals(port)) {
						bytes = bytes + Integer.parseInt(token[4]);
					}
				}else EOF=true;
			}
			isr.close();
		}catch (Exception e1) {
			e1.printStackTrace();
		}
		return " "+(Parser.dateToTimestamp(file).getTime()/1000)+":"+bytes;
	}
```

```
	public static void main(String[] args){

//REQUIRES: la lista dei parametri di ingresso deve avere un elemento che corrisponderà
//            alla porta da ricercare nei flussi; args[0] > 1 && args[0] < 65536.
//            Nel caso di args[0] < 1 || args[0] > 65536 verranno analizzati i flussi
//            relativi a tutto il traffico.
//            Gli archivi relativi alle basi di dati da popolare devono essere create
//            precedentemente all'esecuzione dell'applicazione java Parser.
//OVERVIEW: c rappresenta il comando per aggiornare il database rrdTool.
//EFFECTS: ritorna la stringa s che rappresenta il comando completo che viene eseguito
//            per fare l'update, sul database RRD relativo alla porta interessata, dei
//            dati raccolti.

		String c = "rrdtool update /var/www/"+args[0]+".rrd";
		String s;

		if(Integer.parseInt(args[0]) > 1 && Integer.parseInt(args[0]) < 65536) {
			for (int i=1; i<60; i++){
				s = timestampToMonthDayYear(new Timestamp(System.currentTimeMillis() -
				3600000 + (i*60000)))+".flows";
				c=c+FlowsToRRD(s,args[0]) ;
			}
		}else{
			for (int i=1; i<60; i++){
				s = timestampToMonthDayYear(new Timestamp(System.currentTimeMillis() -
				3600000 + (i*60000)))+".flows";
				c = c + FlowsToRRD(s) ;
			}
		}
		ProvaCommand.toBash(c);
	}
}
```

Script Bash

Le operazioni su terminale necessarie alla configurazione e di avvio del sistema di analisi del traffico sono eseguite su terminale tramite quattro piccoli script bash:

carico.sh

Script bash che ciclicamente (ogni 10 secondi) legge il carico della CPU monitorizzato dal tool di sistema loadavg, inserisce i dati letti nel database RRD e disegna il grafico dei dati raccolti nei 20 minuti precedenti al comando rrdtool graph; il grafico viene poi visualizzato sulla pagina in linguaggio PHP "monitoraggio.php" dell'interfaccia, dove resta costantemente aggiornato in modo autonomo alla pagina web.

```
while [ 1 ] ; do
        echo "updating load.."
        echo ""
        #estraiamo il carico dell'ultimo minuto
        CURLOAD=`cat /proc/loadavg | cut -f 1 -d \ `
        #memorizziamo il valore ottenuto
        rrdtool update /var/www/loadav.rrd N:$CURLOAD
        #diamo qualche informazione a video
        CURTIMEIS=`date`
        echo "updated at "$CURTIMEIS" with "$CURLOAD
        echo ""
        #attendiamo 10 secondi prima di ripetere il tutto
        sleep 10s
                rrdtool  graph  /var/www/graph/loadav.png  DEF:load=/var/www/loadav.rrd:load:AVERAGE
AREA:load#0000ff:Load --start -15m
done
```

sonda.sh

Script bash che si occupa inizialmente della creazione dei database RRD, successivamente entra in un ciclo infinito dove viene lanciata l'applicazione java Parser; viene eseguita una istanza della applicazione per ogni porta da analizzare (compresa la porta 0 che rappresenta il traffico di tutte le porte). Prima di ripetere il ciclo viene loggato in un file la data dell'ultimo lancio e le operazione vengono ripetute dopo un'ora.

Codice script:

```
#!/bin/bash
#rrdtool    create    /var/www/80.rrd    --start    1308143559    --step    60    DS:octet:GAUGE:70:U:U
RRA:AVERAGE:0.5:1:1296000
#echo Archivio Giornaliero Creato port 80

#rrdtool    create    /var/www/22.rrd    --start    1308143559    --step    60    DS:octet:GAUGE:70:U:U
RRA:AVERAGE:0.5:1:1296000
#echo Archivio Giornaliero Creato port 22

#rrdtool    create    /var/www/25.rrd    --start    1308143559    --step    60    DS:octet:GAUGE:70:U:U
RRA:AVERAGE:0.5:1:1296000
#echo Archivio Giornaliero Creato port 25

#rrdtool    create    /var/www/14536.rrd    --start    1308143559    --step    60    DS:octet:GAUGE:70:U:U
RRA:AVERAGE:0.5:1:1296000
#echo Archivio Giornaliero Creato port 14536

#rrdtool    create    /var/www/0.rrd    --start    1308143559    --step    60    DS:octet:GAUGE:70:U:U
RRA:AVERAGE:0.5:1:1296000
#echo Archivio Giornaliero Creato Port TUTTE

#rrdtool    create    /var/www/4662.rrd    --start    1308143559    --step    60    DS:octet:GAUGE:70:U:U
RRA:AVERAGE:0.5:1:1296000
#echo Archivio Giornaliero Creato Port 4662

#rrdtool    create    /var/www/6667.rrd    --start    1308143559    --step    60    DS:octet:GAUGE:70:U:U
RRA:AVERAGE:0.5:1:1296000
#echo Archivio Giornaliero Creato Port 6667
sleep 3600

while true; do
java Parser 80
java Parser 22
java Parser 25
java Parser 14536
java Parser 0
java Parser 4662
java Parser 6667
echo Parser Lanciato in data: >> /var/www/logs/sonda.log
date >> /var/www/logs/sonda.log
sleep 3600
done
```

grafica.sh

Script che si occupa del disegno dei grafici tramite il comando rrdtool graph. Il comando viene eseguito 4 volte per ogni porta disegnando i grafici relativi agli intervalli di tempo della durata di 2 ore, 12 ore, 1 giorno, 1 settimana. La stampa viene ripetuta ogni ora.

```
#/bin/bash

while true;do
rrdtool graph /var/www/graph/80-2h.png DEF:x=/var/www/80.rrd:octet:AVERAGE AREA:x#0000FF:x --start -2h
rrdtool graph /var/www/graph/22-2h.png DEF:x=/var/www/22.rrd:octet:AVERAGE AREA:x#0000FF:x --start -2h
rrdtool graph /var/www/graph/25-2h.png DEF:x=/var/www/25.rrd:octet:AVERAGE AREA:x#0000FF:x --start -2h
rrdtool graph /var/www/graph/14536-2h.png DEF:x=/var/www/14536.rrd:octet:AVERAGE AREA:x#0000FF:x --start -2h
rrdtool graph /var/www/graph/0-2h.png DEF:x=/var/www/0.rrd:octet:AVERAGE AREA:x#0000FF:x --start -2h
rrdtool graph /var/www/graph/4662-2h.png DEF:x=/var/www/4662.rrd:octet:AVERAGE AREA:x#0000FF:x --start -2h
rrdtool graph /var/www/graph/6667-2h.png DEF:x=/var/www/6667.rrd:octet:AVERAGE AREA:x#0000FF:x --start -2h


rrdtool graph /var/www/graph/80-1d.png DEF:x=/var/www/80.rrd:octet:AVERAGE AREA:x#0000FF:x --start -1d
rrdtool graph /var/www/graph/22-1d.png DEF:x=/var/www/22.rrd:octet:AVERAGE AREA:x#0000FF:x --start -1d
rrdtool graph /var/www/graph/25-1d.png DEF:x=/var/www/25.rrd:octet:AVERAGE AREA:x#0000FF:x --start -1d
rrdtool graph /var/www/graph/14536-1d.png DEF:x=/var/www/14536.rrd:octet:AVERAGE AREA:x#0000FF:x --start -1d
rrdtool graph /var/www/graph/0-1d.png DEF:x=/var/www/0.rrd:octet:AVERAGE AREA:x#0000FF:x --start -1d
rrdtool graph /var/www/graph/4662-1d.png DEF:x=/var/www/4662.rrd:octet:AVERAGE AREA:x#0000FF:x --start -1d
rrdtool graph /var/www/graph/6667-1d.png DEF:x=/var/www/6667.rrd:octet:AVERAGE AREA:x#0000FF:x --start -1d


rrdtool graph /var/www/graph/80-12h.png DEF:x=/var/www/80.rrd:octet:AVERAGE AREA:x#0000FF:x --start -12h
rrdtool graph /var/www/graph/22-12h.png DEF:x=/var/www/22.rrd:octet:AVERAGE AREA:x#0000FF:x --start -12h
rrdtool graph /var/www/graph/25-12h.png DEF:x=/var/www/25.rrd:octet:AVERAGE AREA:x#0000FF:x --start -12h
rrdtool graph /var/www/graph/14536-12h.png DEF:x=/var/www/14536.rrd:octet:AVERAGE AREA:x#0000FF:x --start -12h
rrdtool graph /var/www/graph/0-12h.png DEF:x=/var/www/0.rrd:octet:AVERAGE AREA:x#0000FF:x --start -12h
rrdtool graph /var/www/graph/4662-12h.png DEF:x=/var/www/4662.rrd:octet:AVERAGE AREA:x#0000FF:x --start -12h
rrdtool graph /var/www/graph/6667-12h.png DEF:x=/var/www/6667.rrd:octet:AVERAGE AREA:x#0000FF:x --start -12h


rrdtool graph /var/www/graph/80-7d.png DEF:x=/var/www/80.rrd:octet:AVERAGE AREA:x#0000FF:x --start -7d
rrdtool graph /var/www/graph/22-7d.png DEF:x=/var/www/22.rrd:octet:AVERAGE AREA:x#0000FF:x --start -7d
rrdtool graph /var/www/graph/25-7d.png DEF:x=/var/www/25.rrd:octet:AVERAGE AREA:x#0000FF:x --start -7d
rrdtool graph /var/www/graph/14536-7d.png DEF:x=/var/www/14536.rrd:octet:AVERAGE AREA:x#0000FF:x --start -7d
rrdtool graph /var/www/graph/0-7d.png DEF:x=/var/www/0.rrd:octet:AVERAGE AREA:x#0000FF:x --start -7d
rrdtool graph /var/www/graph/4662-7d.png DEF:x=/var/www/4662.rrd:octet:AVERAGE AREA:x#0000FF:x --start -7d
rrdtool graph /var/www/graph/6667-7d.png DEF:x=/var/www/6667.rrd:octet:AVERAGE AREA:x#0000FF:x --start -7d


sleep 3600
done
```

startsonde.sh

Questo script lancia in esecuzione, uno dopo l'altro, nProbe (con il comando già visto in precedenza) e gli script sonda.sh, carico.sh e grafica.sh. Con il comando utilizzato reindirizziamo lo standard output su uscita null.

```
#! /bin/bash
echo "Inizio Attivazione Processi:"
echo "Lancio di nprobe"
/root/nprobe_6.5.1_062711/.libs/lt-nprobe  -b  2  -i  none  -3  2055  -n  none  -T  "%L4_DST_PORT
%IPV4_SRC_ADDR  %IPV4_DST_ADDR  %IN_PKTS  %IN_BYTES  %L4_SRC_PORT  %PROTOCOL  %FIRST_SWITCHED
%LAST_SWITCHED %HTTP_URL " -P /testnProbe/ >/dev/null & echo $! > /var/www/pids/nprobe.pid
sleep 10
echo "lanciato"
echo "Lancio la sonda RRD"
/progettoGR/GR/bin/sonda.sh > /dev/null & echo $! > /var/www/pids/sonda.pid
echo " Lancio processo Carico CPU"
/var/www/carico.sh  >/dev/null & echo $! > /var/www/pids/carico.pid
sleep 5
echo "Lancio Processo Disegno"
/var/www/grafica.sh > /dev/null & echo $! > /var/www/pids/grafica.pid
```

HTML5

La presentazione del progetto che abbiamo realizzato è stata effettuata tramite lo standard HTML versione 5; è stato utilizzato un template standard scelto in modo tale da rendere più chiare le informazioni da presentare. La semplicità della parte grafica è stata combinata con fogli di stile CSS e la capacità del PHP di eseguire script Bash sul server remoto.
Le parti principali del progetto sono raccolte nelle sezioni del sito internet che si trovano alle pagine "monitoraggio" e "Porte".
Nella sezione "Monitoraggio" vengono tenute sotto controllo le performance del server sul quale si trovano il collezionatore ed il parser dei dati. Un grafico aggiornato in modo continuo da uno script Bash presenta il carico della CPU del server stesso (mette in evidenza la mole di lavoro che viene fatta per preparare i dati). Sono presenti anche tre indicatori rettangolari colorati che rappresentano i tre processi (relativi al progetto) che sono in esecuzione sul server: nProbe, il Parser java e lo script per la scrittura dei grafici dai database RRD. Questi indicatori sono di colore verde quando il processo è attivo oppure di colore rosso quando il processo non è avviato o è stato interrotto; è stato scelto di indicare questi tre processi in quanto la loro esecuzione regolare è necessaria per verificare che le operazioni periodiche siano eseguite tutte correttamente.
Nella sezione "porte" sono presenti i link alle pagine relative alle porte analizzate: 22 (SSH), 25 (SMTP), 80 (HTML), 14536 (torrent download), 4662 (eMule), 6667 (IRC) ed una pagina per il traffico complessivo (rappresentato nel codice java dalla porta 0). Per ogni porta sono presenti quattro grafici che presentano l'andamento della quantità di traffico in intervalli diversi di tempo (2h, 12h, 24h, 7g). I grafici delle porte vengono ridisegnati ogni ora da un processo dedicato.

Problemi riscontrati

Durante l'elaborazione del progetto abbiamo riscontrato talvolta alcune problematiche:

- In principio creammo una macchina virtuale su cui sviluppare il progetto, durante le varie configurazioni e settaggi ci accorgemmo di qualche discrepanza nella configurazione di nProbe: infatti il collezionamento dei dati avveniva in modo errato. Praticamente venivano collezionati i dati della scheda di rete virtualizzata sulla macchina virtuale (quindi un loop) invece dei dati del Mikrotik.
- Problemi con il timestamp: nella notazione/identificazione per il timestamp abbiamo dovuto creare un metodo ad hoc nel parser che permettesse di ovviare a questo problema di "sintassi temporale", poiché grazie a questo è possibile leggere i percorsi di dove sono i flussi divisi per cartelle (in anno / mese / giorno)
- Problemi di sincronizzazione: abbiamo riscontrato, durante la lunga esecuzione del progetto, dei problemi tra nProbe e il parser, perchè con l'avvio in contemporanea dello script bash sonda.sh con nProbe generava un errore a causa del fatto che il flusso più recente (es. quello dell'ultimo minuto) non era ancora stato scritto sul disco. Abbiamo risolto anticipando l'avvio di nProbe sugli altri processi di 60 secondi.
- Guasti fisici: nell'ultima parte abbiamo avuto un grosso guasto fisico alla macchina su cui era presente il progetto con perdita di dati (si era bruciato l'hard disk, non siamo riusciti a determinare il motivo) e della macchina stessa; fortunatamente tutto si è risolto grazie alle copie di backup distribuite ad ognuno dei partecipanti al progetto e all'utilizzo di una seconda macchina server.

Manuale d'uso per l'installazione

Per il corretto funzionamento dell'applicazione e di tutti i suoi componenti i file che la compongono devono essere disposti su un PC operante con O/S Linux Ubuntu (come indicato di seguito è certamente valido per le versioni a partire dalla 10.10 Natty) nelle cartelle opportune secondo il seguente schema (cartella > file o cartella contenuta; file o cartella contenuta; ...):

(document root di Apache) /var/www/ > 0.rrd ; 14536.rrd ; 22.rrd ; 25.rrd ; 4662.rrd ; 6667.rrd ; 80.rrd ; carico.sh ; documentazione.html ; grafica.sh ; graph / ; mages / ; index.html ; js / ; loadav.png ; loadav.rrd ; logs / ; monitoraggio.php pids / ; porta-14536.html ; porta-22.html ; porta-25.html ; porta-4662.html ; porta-6667.html ; porta-80.html ; porta-all.html ; porte.html ; progetto.html ; startsonde.sh ; styles / ; utils /

/progettoGR/GR/bin > Parser.class ; ProvaCommand.class ; sonda.sh ; startsonde.sh

(archivio file *.flows) /testnProbe/ > /2011/ ; /2012/ (cartella avente il nome dell'anno in cui è eseguita la raccolta dei dati e la scrittura dei file *.flows ivi contenuti in)

Il corretto funzionamento dell'applicazione è verificato su macchine avente la seguente configurazione Hardware / Software minima:

CPU : AMD Sempron(tm) Processor LE-1100 1900 Mhz
RAM: 2 GB DDR3
O/S: Ubuntu 10.10 Natty
connessione di rete locale 100 Mbps
connessione internet base (56 kbps)
pacchetto applicativo apache2 installato
pacchetto applicativo rrdtool installato
pacchetto applicativo php5 installato
browser web con supporto HTML5: firefox da v8.x, Iexplorer da v9.x, Safari da v5.1 , Chrome da v15 , Opera da v11.1 (fonte web: findmebyIP.com)