Gestione di Reti

Probe Netflow in Java

Anno Accademico 2009 / 2010

Francesco Baldini

Indice generale

Indice generale

Logica della divisione su più file

Classi Che Implementano dei Thread

Probe.java:

è una classe che realizza il thread main, in cui vi è l'entry point del programma,
la sua funzione è quella di soddisfare le richieste che passa l'utente e attivare il thread che ha il compito di catturare i pacchetti e il thread che analizza i flussi e che gestisce gli invii dei flussi.

Sniffer.java:

è una classe che gestisce la logica della cattura dei pacchetti dalla scheda di rete, una volta catturati, li parsa in Record Netflow e quest'ultimi vengono inseriti nella cache, che è una Mappa da Stringhe a Record Netflow.

FlowAnalyzer.java:

è una classe che realizza l'analisi della cache del probe, controlla la terminazione dei record di flusso, crea i pacchetti UDP contenenti i flussi (per un massimo di 30 record a pacchetto) e un header di Flusso. I pacchetti sono inviati al Collector. L'invio dei dati è effettuato sfruttando come protocollo di livello trasporto: UDP.

Classi Di Libreria

Ipv4Parser:

è una classe di libreria che mappa pacchetti catturati dalla scheda di rete tramite la libreria jpcap (che è un wrapper della libreria pcap o winpcap) in strutture dati del probe quali NetFlow5 Record, mappando solo i campi pertinenti del livello rete Ipv4.

TCPParser:

è una classe di libreria che mappa pacchetti catturati dalla scheda di rete tramite la libreria jpcap (che è un wrapper della libreria pcap o winpcap) in strutture dati del probe quali NetFlow5 Record, mappando solo i campi pertinenti del livello trasporto TCP.

UDPParser:

è una classe di libreria che mappa pacchetti catturati dalla scheda di rete tramite la libreria jpcap (che è un wrapper della libreria pcap o winpcap) in strutture dati del probe quali NetFlow5 Record, mappando solo i campi pertinenti del livello trasporto UDP.

ICMPParser:

è una classe di libreria che mappa pacchetti catturati dalla scheda di rete tramite la libreria jpcap

(che è un wrapper della libreria pcap o winpcap) in strutture dati del probe quali NetFlow5 Record, mappando solo i campi pertinenti del livello trasporto ICMP.

NetFlow5Header:

è una classe di libreria che rappresenta la struttura dati di header Netflow versione 5 secondo la RFC di NetFlow versione v5

FORMATO FLOW HEADER

Bytes	Contents	Description
0-1	version	NetFlow export format version number
2-3	count	Number of flows exported in this packet (1-30)
4-7	sys_uptime	Current time in milliseconds since the export device booted
8-11	unix_secs	Current count of seconds since 0000 UTC 1970
12-15	unix_nsecs	Residual nanoseconds since 0000 UTC 1970
16-19	flow_sequence	Sequence counter of total flows seen
20	engine_type	Type of flow-switching engine
21	engine_id	Slot number of the flow-switching engine
22-23	sampling_interval	First two bits hold the sampling mode; remaining 14 bits hold value of sampling interval

NetFlow5Record:

è una classe di libreria che rappresenta la struttura dati di record Netflow versione 5 secondo la RFC di NetFlow versione v5

FORMATO FLOW RECORD

Bytes	Contents	Description
0-3	srcaddr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of next hop router
12-13	input	SNMP index of input interface
14-15	output	SNMP index of output interface
16-19	dPkts	Packets in the flow
20-23	dOctets	Total number of Layer 3 bytes in the packets of the flow
24-27	first	SysUptime at start of flow
28-31	last	SysUptime at the time the last packet of the flow was received
32-33	srcport	TCP/UDP source port number or equivalent
34-35	dstport	TCP/UDP destination port number or equivalent
36	pad1	Unused (zero) bytes
37	tcp_flags	Cumulative OR of TCP flags

38	prot	IP protocol type (for example, TCP = 6; UDP = 17)
39	tos	IP type of service (ToS)
40-41	src_as	Autonomous system number of the source, either origin or peer
42-43	dst_as	Autonomous system number of the destination, either origin or peer
44	src_mask	Source address prefix mask bits
45	dst_mask	Destination address prefix mask bits
46-47	pad2	Unused (zero) bytes

NetFlow5Packet:

è una classe di libreria che rappresenta la struttura dati di pacchetto NetFlow5, formato da un header NetFlow5 e un array di record NetFlow5 di al più 30 posizioni.

```
struct netflow5_packet
{
struct flow_ver5_hdr flowHeader;
struct flow_ver5_rec flowRecord[30];
} NetFlow5Record;
```

La classe ha l'utilità di incapsulare tutti i dati sui flussi (header + records) che saranno inviati al collctor su UDP.

Maggiori dettagli per ogni classe sviluppata riguardo metodi e variabili sono contenuti nei commenti del codice sorgente, o nella documentazione javadoc.

Le principali scelte di progetto e la struttura dei programmi

Il processo Probe è strutturato in tre Thread: Main, Sniffer, FlowAnalzyer.

Il thread Main effettua il controllo degli argomenti passati al processo, li esamina li parsa, inoltre alloca memoria per le strutture dati condivise: cache, questa struttura dati è una Map, che mappa Integer in NetFlow5Record; infine il thread main crea i thread Sniffer, FlowAnalzyer.

Il Thread Sniffer ha il compito di catturare i pacchetti dalla scheda di rete che l'utente ha specificato fra i vari argomenti del processo, la cattura dei pacchetti è effettuata utilizzando la libreria jpcap che incapsula la libreria pcap, la cattura avviene in modalità promiscua in modo tale da analizzare tutto il traffico del proprio dominio di broadcast.
I pacchetti una volta catturati vengono parsati e quindi mappati in strutture dati NetFlow5Record, e quest'ultimi vengono aggiunti nella cache, oppure se il flusso esisteva già nella cache questo flusso viene aggiornato con i nuovi dati provenienti dall'ultimo pacchetto catturato.

Il Thread FlowAnalzyer carica da un file di properties il tempo di frequenza T espresso in millisecondi di analisi della cache, il tempo per ritenere i flussi inattivi e il tempo per ritenere i flussi troppo lunghi. Una volta effettuato il caricamento, il thread ogni T millisecondi esamina la cache controllando se vi sono dei i flussi terminati, se vi sono crea pacchetti NetFlow5Packet

questi sono formati da un header e al più 30 record, quindi i pacchetti inviati sono in funzione dei record che devono essere inviati (flussi terminati). Riepilogando ogni T millisecondi viene analizzata la cache e il Numero pacchetti inviati sarà uguale NumeroFlussiTerminati / 30 approssimato all'intero superiore. Ogni pacchetto verrà inserito nella parte dati in un segmento UDP ed inviato su rete al collector.
La parte dati del pacchetto IP sarà al più occupata da :

- Dati a livello applicazione: 24 + 48*30 (record) byte = 1464
- L'header UDP è di 8 byte

L'header IP è di 20byte + opzioni.
Considerando i dati procedenti, sulle MTU tipiche quali Ethernet di 1500 byte, i dati degli oggetti NetFlow5Packet stanno sicuramente su un unico pacchetto IP senza dover frammentare il pacchetto a livello rete.

Le classi Ipv4Parser, TCPParser, UDPParser, ICMPParser parsano i pacchetti jpcap.packet.Packet in strutture dati NetFlow5Record, ogni classe parsa rispettivamente gli header del proprio protocollo, settando i relativi valori nella struttura dati NetFlow5Record.

Le classi NetFlow5Packet, NetFlow5Record, NetFlow5Header sono delle classi che hanno la funzione di wrapper delle strutture definite nelle RFC per Netflow versione 5, le classi wrappano rispettivamente le strutture riguardanti i pacchetti a livello applicazione che vengono inviati su rete, la struttura dell'header contenuta nel pacchetto ed infine i record che sono contenuti all'interno del pacchetto.
Inoltre queste classi forniscono una serie di metodi setter e getter per ottenere i dati dei campi.
Riguardo alcuni metodi delle classi NetFlow5Record, ad esempio i metodi che restituiscono la porta sorgente o porta destinazione, restituiscono un short poiché le porte sono codificate su 16bit, ma dato che in java gli short sono su 16bit e con segno qualora uno volesse stampare il numero di porta è necessario convertire lo short su un int a 32bit non con il cast, ma utilizzando un operatore bit a bit ad esempio: porta = 0xFFFF & record.getDstPort().
Lo stesso ragionamento vale per IP sorgente e destinazione stavolta ragionando su 32 bit anziché su 16 bit mettendolo su un long.

L'invio dei dati su rete è effettuato utilizzando il protocollo UDP e i dati sono inviati rispettando il formato stabilito nella la RFC di Netflow versione 5.

Le politiche di sincronizzazione tra thread utilizzate

I thread Sniffer, FlowAnalyzer lavorando in un modello ad ambiente globale, accedono allo stato condiviso che è allocato in un oggetto della classe HahMap.

La competizione avviene nel seguente modo:
Per garantire la mutua esclusione nella sezione critica ho deciso di utilizzare i blocchi synchronized. Quindi il thread che vuole eseguire il blocco sincronizzato deve acquisire il lock sull'oggetto, il quale viene rilasciato nel momento in cui il thread termina l'esecuzione del blocco (es: return, throw, esecuzione dell'ultima istruzione del blocco).

La cooperazione avviene nel seguente modo:
Il Thread Sniffer inserisce nella cache i NetFlow5Record, il Thread FlowAnalyzer analizza periodicamente la cache in base al tempo settato dall'utente, controllando i flussi terminati, se vi sono li elimina dalla cache e invia su UDP i creati opportunamente NetFlow5Packet.

Manuale d'uso

Un breve manuale d'uso che descrive i comandi necessari per avviare il client Probe :

Il processo cliente Probe può essere eseguito passando i seguenti parametri da riga di comando:

java -jar GR-Probe.jar -i networkInterfaceName -o host:port [-f bfpFilter]

Gli argomenti *-i networkInterfaceName* ed "*-o host:port* sono obbligatori, mentre "-f bfpFilter" è opzionale. L'ordine degli argomenti deve essere rispettato.

Descrizione degli argomenti

- -i networkInterfaceName rappresenta l'interfaccia di rete da cui catturare i pacchetti. Può essere passato il nome della scheda (ad esempio *-i Marvell*) , oppure il tipo della scheda di rete (ad esempio *-i Ethernet*).
- *-o host:port* rappresenta rispettivamente prima l'indirizzo IP su cui gira il server e poi la porta sull'host, su cui il server è in ascolto.
- *-f bfpFilter* è opzionale e rappresenta l'espressione di filtro da applicare alla cattura dei pacchetti dalla scheda di rete, *bfpFilter* non deve necessariamente una sola stringa ma puo' essere immesso come un insieme di stringhe (ad esempio: *-f port 80*)

Il Processo Probe se rileva un errore che causa la terminazione del processo,stampa a video il messaggio con la relativa spiegazione di causa di terminazione.

File di Properties

il file di properties si trova all'interno dell'archio GR-Probe.jar, il file si chiama Var.properties e le variabili sono definite per default a:

FLOW_TOO_LONG_SECONDS = 1800
FLOW_INACTIVE_SECONDS = 15
ANALYSIS_FREQUENCY = 5000

Testing

il test è stato effettuato implementando un semplice collector che stampa a video i risultati visti.
Il processo server Collector può essere eseguito passando i seguenti parametri da riga di comando:

java -jar GR-Collector.jar [numeroPorta]

Opzionalmente è possibile quindi passare il numero porta di ascolto, altrimenti il processo per default è in ascolto sulla porta 2121.

Come Installare il supporto per la libreria Jpcap

Microsoft Windows

1.Download and install Java Runtime Environment 6 (if you want to simply run Jpcap-based applications) or JDK 6 (if you want to develop Jpcap-based applications).

2.Download and install the latest WinPcap.

3.Download and run Jpcap Self Installer for windows.

If you have the previous version of Jpcap, please uninstall it first before installing a new version.

If you forgot to uninstall the previous version, search 'Jpcap.dll' and 'jpcap.jar' from your system, delete them, and then install a new version.

It is recommended to use JRE/JDK 6 or higher because Jpcap installer assumes JRE/JDK 6.

If you are using JRE/JDK 5, set your 'CLASSPATH' to include '%SystemRoot%\Sun\Java\lib\ext'

Linux

Use RPM package (Fedora, RedHat)

Download and install Jpcap RPM package.

For some distributions (e.g., Mandriva), JDK6 is also installed automatically.

However, for some distributions (e.g., Fedora Core), you have to manually install Sun's JDK6 for Linux as a RPM package before installing Jpcap RPM package.

Use Debian package (Ubuntu, GNU/Debian)

Download and install Jpcap Debian package.

For some distributions (e.g., Ubuntu), JDK6 is also installed automatically.

For some distributions (e.g., Debian GNU/Lunux), you may have to edit apt-line (e.g., include "non-free") before installing Jpcap Debian package. Also, you may have to update your GLIBC to version 2.4 or higher.

If you cannot install using RPM/Debian package, you can try to build from the source.

It is recommended to use Sun's JRE/JDK 6 or higher because Jpcap package assumes JRE/JDK 6.

If you are using JRE/JDK 5, set your 'CLASSPATH' to include '/usr/java/packages/lib/ext'

Also, Jpcap currently does not work with gcj / gij, so please use Sun's JRE/JDK.

Mac OS X

1.Both Java and libpcap are preinstalled on Mac OS X.

If any of them is missing you should be able to install them from the Mac OS X install DVD.

2.Download and install Xcode.

The default installation of Xcode should provide you with the toolchain required for compiling Jpcap.

3.Download and extract Jpcap source build.

4.Go to '[Jpcap extracted directory]/src/c' directory.

5.Run 'make'.

6.Copy 'libjpcap.jnilib' to '/Library/Java/Extensions/' directory.

7.Copy '[Jpcap extracted directory]/lib/jpcap.jar' to '/Library/Java/Extensions/'

Or, place 'jpcap.jar' to any directory and include it to your CLASSPATH.

Source build (Linux/FreeBSD/Solaris)

1.Make sure you have 'gcc' and 'make' installed to compile Jpcap.

Other software/packages may be necessary (for example, you need 'build-essential' package to install on Ubuntu).

2.Download and install libpcap (ver.0.9.4 or later) if not installed.

3.Download and extract Jpcap source build.

4.Go to '[Jpcap extracted directory]/src/c' directory.

5.Run 'make'.

6.Copy 'libjpcap.so' to '[Java installed directory]/jre/lib/<arch>'. <arch> is either 'i386' or 'sparc'.

Or, if you are using Java 6, copy 'libjpcap.so' to '/usr/java/packages/lib/ext'.

Or, place 'libjpcap.so' in the directory where your application is located.

7.Copy '[Jpcap extracted directory]/lib/jpcap.jar' to '/usr/lib' or '[Java installed directory]/jre/lib/ext'.

Or, place 'jpcap.jar' to any directory and include it in your CLASSPATH.

Maggiori informazioni reperibili all'URL:

http://netresearch.ics.uci.edu/kfujii/jpcap/doc/install.html