MODULARITÀ APPLICATA ALL'ELABORAZIONE DI PACCHETTI DI RETE: IL LINGUAGGIO NETPDL

M. Baldi, F. Risso
Dipartimento di Automatica e Informatica, Politecnico di Torino
{mario.baldi, fulvio.risso}@polito.it

Questo articolo presenta il linguaggio NetPDL, un linguaggio basato su XML che permette la creazione di applicazioni che interagiscono direttamente con la rete (ad es. firewall, IDS, sniffer) con una struttura maggiormente modulare, mantenendone però la necessaria efficienza dal punto di vista elaborativo. Inoltre, vengono presentate alcune tecnologie aggiuntive (PDML, PSML) che mirano a facilitare lo scambio di informazioni relativamente all'elaborazione di pacchetti di rete, che è un fattore abilitante per la creazione di applicazioni modulari. Queste tecnologie sono state implementate nella libreria open-source NetBee che presenta un esempio delle funzionalità e delle prestazioni ottenibili con queste tecnologie.

1. Introduzione

La diffusione delle reti di comunicazione ha provocato un notevole cambiamento architetturale in molti applicativi, i quali tendono oggi a focalizzarsi su uno specifico problema delegando altre funzionalità secondarie (che vengono viste come funzioni "non mission-critical") ad altre applicazioni. Questa riscoperta della modularità porta ad una notevole efficienza (ogni applicativo fa, al meglio, quello che è il suo compito primario) ed è possibile solamente grazie alla velocità e all'efficienza delle moderne reti di calcolatori.
A questo approccio modulare fanno eccezione gli applicativi di rete. Da questa prospettiva, è tuttavia necessario distinguere due tipi di applicativi di rete. La prima categoria comprende applicazioni (ad esempio un web server o un web client) che usano la rete come un semplice "trasporto" di dati. Queste applicazioni inviano e ricevono i dati attraverso una interfaccia di alto livello (ad es. i *sockets*) e non sono interessate in alcun modo ai dettagli interni alla rete. La seconda categoria (che è l'argomento di riferimento di questo articolo) comprende le applicazioni che interagiscono direttamente con i pacchetti di rete e pertanto devono avere una conoscenza approfondita dei meccanismi interni alla rete stessa. A questa seconda categoria afferiscono firewall, NAT, IDS, analizzatori di pacchetti (sniffer), strumenti di monitoraggio del traffico, etc.
Una maggiore modularizzazione può portare un notevole beneficio a questa seconda categoria di applicazioni in quanto permette di evitare lo spreco di risorse necessarie per gestire aspetti secondari al problema per cui l'applicativo è progettato. Ad esempio, un firewall potrebbe concentrare le sue attenzioni nel controllare che un pacchetto che trasporta una pagina web non contenga del codice malevolo, delegando piuttosto una entità esterna (ottimizzata a questo scopo) a localizzare l'inizio effettivo dei dati TCP. Una azienda che produce un firewall potrebbe pertanto concentrare le proprie attenzioni sulle regole di

sicurezza, su nuove tipologie di attacco, delegando altri moduli a localizzare le informazioni (all'interno del pacchetto) di cui ha la necessità, e così via.
Un esempio dei benefici di una elaborazione maggiormente modulare può essere vista in Figura 1, che presenta un frammento di codice che controlla se una trama Ethernet contiene un segmento TCP. In questo esempio viene controllato prima se la trama Ethernet trasporta un pacchetto IPv4 e, in caso affermativo, se il protocollo di trasporto è TCP.

```
if ((packet[12]==0x800) && (packet[23]==6))
  /* TCP packet */
else
  /* Non TCP packet */
```

Figura 1. Filtraggio di pacchetti TCP imbustati in Ethernet/IP.

Tuttavia, nel caso in cui l'applicazione abbia la necessità di supportare anche IPv6, il codice deve essere modificato come mostrato in Figura 2 per tenere in conto anche della nuova possibilità di imbustamento.

```
if (((packet[12]==0x800) && (packet[23]==6) ||
    ((packet[12]==0x86dd) && (packet[20]==6)))
{
  /* TCP packet */
}
else
{
  /* Non TCP packet */
}
```

Figura 2. Filtraggio di pacchetti TCP imbustati in Ethernet/IPv4-IPv6.

Come risulta evidente, il codice necessario per filtrare un pacchetto TCP può diventare estremamente complesso in caso di supporto di link-layer multipli (Ethernet, WiFi, etc.), oppure in presenza di opzioni negli header IPv4-IPv6.
Questo esempio dimostra non solo come l'elaborazione di un pacchetto di rete possa essere estremamente complessa, ma come questa operazione sia di scarso interesse concettuale per una serie di applicativi (es. firewall) che, pur necessitando di queste operazioni per il loro funzionamento, che non sono parte delle loro funzioni principali. Da questo punto di vista, per un programmatore di un firewall sarebbe preferibile scrivere semplicemente del codice sul modello di quello mostrato in Figura 3, nel quale il processamento di basso livello è delegato in toto ad un'entità esterna.

```
if (Packet.Contains("tcp"))
  /* TCP packet */
else
  /* Non TCP packet */
```

Figura 3. Frammento di codice che delega ad una entità esterna il compito di verificare che un pacchetto contenga dati TCP.

La Figura 4 mostra un possibile scenario futuro nel quale numerosi applicativi che richiedono l'elaborazione diretta di pacchetti di rete adottano un approccio modulare: ad esempio, un elevato numero di applicazioni può trovare

vantaggioso appoggiarsi a componenti esterni quali ad esempio *packet decoders* oppure *packet filters*.

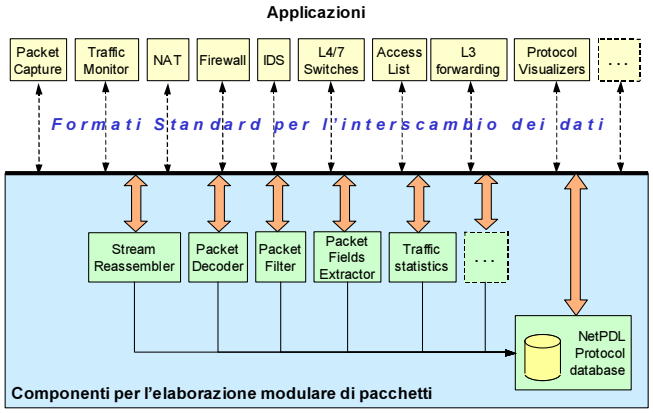

Figura 4. Esempio di elaborazione modulare di pacchetti di rete.

Questo articolo propone una prima realizzazione di questa visione e presenta alcuni linguaggi basati su XML pensati per facilitare la realizzazione di una maggiore modularità degli applicativi di rete. Il primo (e più importante) linguaggio, *Network Protocol Description Language* (NetPDL), affronta il problema di identificare univocamente ogni campo di ogni protocollo di rete. In altre parole, definisce il formato di ogni singolo protocollo e si incarica di dare un nome univoco ad ogni campo (ad es. `ip.src` per quanto riguarda il campo "indirizzo IP sorgente"). Altri due linguaggi che vengono presentati brevemente sono il *Packet Description Markup Language* (PDML) e il *Packet Summary Markup Language* (PSML), che mirano a definire il formato di interscambio dei dati tra un modulo di elaborazione dedicato e l'applicativo ricevente, in quanto queste entità necessitano di scambiarsi i dati non solo attraverso interfacce standard, ma anche con una codifica standard (e condivisa) dei dati stessi.
I linguaggi NetPDL, PDML e PSML sono i primi esempi di linguaggi basati su XML orientati alla realizzazione di applicativi modulari per il processamento di pacchetti di rete, come mostrato in Figura 4. Lo standard XML può essere una buona scelta sia per la sua flessibilità (e la sua attuale diffusione), sia per la presenza di numerosi strumenti software (molti open-source) in grado di gestire files XML. Grazie a questi strumenti, un programmatore può concentrarsi maggiormente sulla semantica dei dati in quanto il controllo sintattico che viene fatto automaticamente da queste librerie XML. Inoltre, un linguaggio basato su XML può essere facilmente esteso (nuovi elementi o nuovi attributi di elementi esistenti) pur mantenendo la compatibilità con le precedenti versioni.
Questo articolo evidenzierà le caratteristiche principali del linguaggio NetPDL nella Sezione 2, quindi presenterà una sua valutazione prestazionale nella Sezione 3. La Sezione 4 è dedicata a proporre in maniera più sistematica i concetti della modularità e presenterà un esempio di realizzazione di questo scenario attraverso la libreria NetBee. Infine, la Sezione 5 evidenzierà le principali conclusioni di questo lavoro.

2. NetPDL: un linguaggio per la descrizione del formato dei pacchetti di rete

Il *Network Protocol Description Language* (*NetPDL*) [4] è un linguaggio semplice e generico che mira a descrivere il formato di un pacchetto di rete dal punto di vista del formato dei dati (*packet header*) e dell'incapsulamento del protocollo stesso. La semplicità è dovuta soprattuto al fatto che il NetPDL non comprende primitive di descrizione temporale del protocollo (macchina a stati). Inoltre, è facilmente estendibile grazie al fatto di essere basato su XML.

2.1. Panoramica del linguaggio

NetPDL si compone di un certo numero di primitive composte da un elemento XML (es. `<proto>`) e caratterizzate da un insieme di attributi (es. `name="Ethernet"`). Ad esempio, un elemento indica l'inizio di un nuovo campo, mentre la sua dimensione è espressa come attributo dell'elemento stesso.

La Figura 5 mostra un estratto di una descrizione relativa ad un header Ethernet. Questo protocollo è composto da tre campi di lunghezza fissa (rispettivamente di sei, sei e due bytes), ed è caratterizzato da contenere altri protocolli a seguire. Il formato della trama è contenuta nella sezione `<fields>`, mentre l'imbustamento è contenuto negli elementi `<protoref>` inclusi nella sezione `<nextproto>`. La scelta del protocollo successivo avviene attraverso il valore assunto dal campo `type-length`, che è utilizzato dall'espressione di valutazione contenuta nella sezione `<expr>`. Alcuni protocolli predefiniti (`_startproto` e `_defaultproto`) vengono utilizzati in casi particolari: il primo per identificare la tecnologia di livello data-link sulla quale la trama è stata generata, mentre il secondo offre un protocollo di "default" da utilizzarsi qualora non esistano descrizioni per il protocollo contenuto nel pacchetto stesso.

```
<proto name="Ethernet">
  <fields>
    <fixed name="dst" size="6"/>
    <fixed name="src" size="6"/>
    <fixed name="type-length" size="2"/>
  </fields>

  <nextproto>
    <switch>
      <expr type="int">
        <fieldref name="type-length">
      </expr>

      <case value="2048"><protoref name="IP"/></case>
      <case value="2054"><protoref name="ARP"/></case>
    </switch>
  </nextproto>
</proto>
```

Figura 5. Estratto della descrizione NetPDL relativa al protocollo Ethernet.

La maggioranza degli header dei protocolli utilizzano un insieme di campi con caratteristiche abbastastanza standard e che ricadono in cinque diverse categorie. La maggioranza dei campi sono a lunghezza fissa con lunghezza multipla di un byte (da cui l'elemento `<fixed>`), mentre in alcuni casi un campo è composto da bit non allineati al byte (da cui l'elemento `<masked>`). Altri campi sono caratterizzati dal fatto che la loro lunghezza può essere variabile (e

ricavabile solamente al momento dell'elaborazione del protocollo stesso) in base ad altre informazioni contenute nel pacchetto, da cui i campi di tipo `<variable>`. Infine, l'elemento `<line>` definisce un campo variabile terminato da un carattere "a capo" e `<padding>` è utilizzato per riallineare un header a multipli di 16 o 32 bit.
Un campo è completamente caratterizzato dal valori della sua *lunghezza*, del suo *numero di ripetizioni* e dalla sua *posizione* all'interno dell'header, anche se le ultime due informazioni sono spesso superflue (normalmente un campo è presente una sola volta nell'header e la sua posizione è "a seguire" il campo precedente) e pertanto vengono indicate solamente nel caso in cui si differenzino dai valori di default. Viceversa, la lunghezza di ogni campo è espressa da un un attributo `size` che deve essere sempre presente.

2.2. Caratteristiche avanzate del linguaggio NetPDL

Purtroppo, alcuni protocolli di rete hanno funzionalità particolari che non possono essere espresse attraverso i soli elementi precedenti. Ad esempio, perfino un protocollo comune come IP è contraddistinto da una parte obbligatoria (i primi 20 bytes) ed una parte opzionale che è presente solamente a fronte del verificarsi di determinate condizioni nell'header iniziale. Il NetPDL gestisce situazioni di questo tipo attraverso la definizione di primitive per l'elaborazione condizionale: il formato dei pacchetti può essere differenziato a seconda della presenza o meno di alcuni valori in certi campi. Il NetPDL prevede l'elaborazione condizionale, il supporto a campi ripetuti e il supporto *stateful* (in alcuni protocolli l'elaborazione dipende da pacchetti precedentemente scambiati). Infine, nel caso anche queste funzionalità avanzate non siano sufficienti, il NetPDL prevede il meccanismo del *plug-in* per gestire uno specifico protocollo con codice nativo. Questo meccanismo aggiuntivo permette di mantenere semplice il NetPDL (non è necessario aggiungere primitive ad-hoc per protocolli specifici), pur garantendone il suo utilizzo anche in caso di protocolli "patologici". Ad esempio, il meccanismo dei *plug-in* è utilizzato per la definizione del formato del protocollo DNS, il quale utilizza un meccanismo di compattamento delle informazioni estremamente peculiare e che non si trova in altri protocolli.

2.3. Estensibilità del linguaggio NetPDL

Una delle caratteristiche fondamentali del linguaggio NetPDL, derivate dalla sua natura *XML-based*, è la sua *estensibilità*, ossia la possibilità di aggiungere nuove primitive (sotto forma di elementi o attributi XML) al linguaggio stesso. Questo permette alle applicazioni di essere conformi con un set base del linguaggio (gli elementi sconosciuti vengono semplicemente ignorati), cosicchè è possibile definire estensioni al linguaggio base mantenendo la compatibilità con gli altri applicativi. Un esempio di estensione riguarda la validità di ciascun campo: per alcune classi di applicativi potrebbe essere importante conoscere che i valori ammessi per un certo campo sono compresi in un certo intervallo, da cui l'importanza di poter estendere il linguaggio secondo le proprie esigenze.
Attualmente è stata definita una sola estensione al linguaggio (la *NetPDL Visualization Extension*) che definisce le modalità con le quali il valore di un campo deve essere visualizzato da un applicativo. Ad esempio, il valore di un

campo a 32 bit dovrà essere visualizzato come un numero esadecimale qualora ci si riferisca ad un CRC, mentre dovrà essere visualizzato in notazione decimale puntata qualora sia un indirizzo IP. Questa estensione definisce due diverse viste di un pacchetto: una vista di *sommario* relativamente ad ogni protocollo, nella quale vengono indicati i campi principali del protocollo stesso e i relativi valori assunti nel pacchetto in esame, e una vista di *dettaglio* che elenca tutti i campi (e i relativi valori) presenti all'interno del protocollo stesso. Queste estensioni definiscono nuovi elementi e attributi che vengono utilizzati all'interno di un *template di visualizzazione*, a cui si accede attraverso gli attributi `showsumtemplate` e `showtemplate`, come mostrato in Figura 6.
Tra le informazioni più importanti contenute in un template di visualizzazione vi sono gli attributi `showtype`, `showgrp`, e `showsep`, che indicano rispettivamente il formato (esadecimale, decimale, ascii o binario) di ogni byte, come i bytes devono essere raggruppati insieme, e il carattere di separazione tra i vari gruppi. Ad esempio, i campi *MAC Source* e *MAC Destination* in Figura 6 sono associati al template di visualizzazione `EthMAC`, il quale visualizza i valori dividendo l'indirizzo MAC in due parti da tre byte ciascuno (come specificato dall'attributo `showgrp`), visualizzando le due porzioni in esadecimale (attributo `showtype`) separate dal carattere "`-`" (attributo `showsep`), con un risultato finale ad esempio di `000800-AB34F9`.

```
<proto name="Ethernet" longname="Ethernet 802.3“
         showsumtemplate="eth">
  <fields>
    <fixed name="dst" longname="MAC Destination" size="6"
         showtemplate="EthMAC"/>
    <fixed name="src" longname="MAC Source" size="6"
         showtemplate="EthMAC"/>
    <fixed name="type-length" longname="Ethertype - Length" size="2“
         showtemplate="FieldHex"/>
  </fields>
  ...
</proto>
...
<netpdlshow>
  <showtemplate name="FieldHex" showtype="hex"/>
  <showtemplate name="EthMAC" showtype="hex" showgrp="3" showsep="-"/>

  <showsumtemplate name="ethernet">
    <section name="next"/>
    <text value="Eth: "/>
    <pdmlfield name="src" attrib="show"/>
    <text value=" => "/>
    <pdmlfield name="dst" attrib="show"/>
  </showsumtemplate>
</netpdlshow>
```

Figura 6. Esempio di NetPDL Visualization Extension relativo al protocollo Ethernet.

In Figura 6 è mostrato anche il template di visualizzazione relativo alla vista di sommario applicata ad un intero protocollo. Nell'esempio, ogni trama Ethernet verrà associata ad una vista repilogativa iniziante con la stringa "`Eth:`" seguita dall'indirizzo MAC sorgente, dalla stringa "=>" e dall'indirizzo MAC destinazione, con un risultato simile al seguente:

```
Eth: 0001C7-B75007 => 000629-393D7E
```

3. Analisi prestazionale del linguaggio NetPDL

Una delle maggiori critiche al linguaggio NetPDL è relativa alle presunte minori prestazioni di uno strumento che utilizza questa tecnologia (completamente generica e svincolata da ogni protocollo) anzichè una tecnologia magari meno

generale, ma più immediata e realizzabile con codice nativo. In altre parole, la differenza è, ad esempio, tra riconoscere i campi presenti in una trama Ethernet attraverso il parsing della descrizione NetPDL applicata al pacchetto di rete piuttosto che con uno spezzone di codice nativo (ad esempio in linguaggio C) che assegna immediatamente i primi 6 bytes all'indirizzo MAC destinazione, e così via. Intuitivamente, la seconda soluzione, ancorchè di validità limitata (non funzionerebbe in caso di una rete WiFi), sembra nettamente più efficiente.
Per smentire questa critica sono stati effettuati alcuni test prestazionali miranti a confrontare il modulo di decodifica dei pacchetti implementato nella libreria NetBee [2] (che rappresenta la prima implementazione di decodificatore di pacchetti completamente basato sulla tecnologia NetPDL) con Tethereal [3], uno sniffer a riga di comando che condivide lo stesso motore di decodifica con il fratello maggiore Ethereal [3]. I test, eseguiti su un personal computer Pentium IV – 2.4GHz, mirano a decodificare un insieme di pacchetti contenuti in alcuni file. I risultati, mostrati in TABELLA I, indicano che le prestazioni ottenute da NetBee e Tethereal sono estremamente simili, con un tempo di processamento medio per pacchetto rispettivamente di 75 e 66 µs. Nel caso di decodifica parziale (per ogni campo viene riportato il valore in esadecimale, la posizione nel pacchetto e la sua dimensione ma non viene applicato nessun template di visualizzazione), NetBee riduce i tempi di processamento da 75 a 39 µs per ogni pacchetto; questa funzionalità non è disponibile in Tethereal e pertanto non può essere confrontata.
Questi risultati forniscono una buona indicazione delle prestazioni ottenibili attraverso l'impiego della tecnologia NetPDL, dimostrando chiaramente come software basati su NetPDL possono essere estremamente competitivi anche quando vengono confrontati con software basati su codice nativo: in altre parole, la tecnologia NetPDL non inserisce, in sè, particolari penalizzazioni prestazionali in quanto la velocità di elaborazione dipende soprattutto dalla qualità del motore di elaborazione basato su questo linguaggio.

TABELLA I
CONFRONTO PRESTAZIONALE TRA DECODIFICA DI PACCHETTI CON CODICE NATIVO E CON CODICE BASATO SU NETPDL.

	Software	**Risultati**
Decodifica pacchetti (completa; comprende i valori dei campi e la loro stampa in formato leggibile)	Tethereal (codice nativo)	66 µs/pacchetto
	NetBee	75 µs/pacchetto
Decodifica pacchetti (parziale; comprende solamente i valori dei campi e la stampa in esadecimale)	NetBee	39 µs/pacchetto

4. Modularità applicata alle applicazioni di rete: la libreria NetBee

La libreria NetBee fornisce una prima implementazione di moduli software basati su NetPDL ed è attualmente utilizzata dallo sniffer Analyzer [1] (a partire dalla versione 3.0). Questa libreria implementa un modulo di decodifica di pacchetti, uno di formattazione di campi (ad esempio per trasformare un *dump* esadecimale di un campo in un indirizzo IP e viceversa) e uno (sperimentale) di filtraggio di pacchetti in tempo reale basati sulla tecnologia NetPDL. Questa libreria esporta un'interfaccia estremamente pulita che permette ad un programmatore di delegare queste funzionalità di più basso livello alla libreria, concentrandosi sulle operazioni (per lui) a maggiore valore aggiunto. D'altro canto, i moduli di questa libreria sono particolarmente ottimizzati in previsione di

un uso da parte di vari applicativi; è superfluo notare come una maggiore ottimizzazione della libreria NetBee porta inevitabilmente vantaggi a tutte le applicazioni che si appoggiano su di essa. La Figura 7 mostra un estratto di codice che utilizza questa libreria ed è evidente come la completa decodifica di un pacchetto possa essere fatta con un numero di linee di codice estremamente limitato. Analogamente, anche altre funzionalità frequenti quali la ricerca di campi, o di campi con particolari valori, o altro ancora sono accessibili in maniera estremamente semplificata.

```
while (1)
{
struct _nbPDMLPacket *PDMLPacket;
struct _nbPDMLProto *ProtocolItem;

  // Read packet from file or network
  Res= PacketSource->Read(&PacketHeader, &PacketData);

  if (Res == nbFAILURE)
    break;

  // Decode packet
  Decoder->DecodePacket(DataLinkCode, PacketCounter,
                        PacketHeader, PacketData);

  // Get the current decoded packet
  PDMLReader->GetCurrentPacket(&PDMLPacket);

  // Print some global information about the packet
  printf("Packet number %d\n", PDMLPacket->Number);
  printf("Total lenght= %d\n", PDMLPacket->Length);

  // Retrieve the 1st protocol contained in the packet
  ProtocolItem= PDMLPacket->FirstProto;

  // Scan the current packet and print on screen the most
  // relevant data related to each proto contained in it
  while(ProtocolItem)
  {
    printf ("Protocol %s: size %d, offset %d\n",
             ProtocolItem->LongName, ProtocolItem->Size,
             ProtocolItem->Position);

    ProtocolItem= ProtocolItem->NextProto;
  }
}
```

Figura 7. Estratto di codice che utilizza la libreria NetBee: decodifica e stampa della vista di dettaglio di ogni pacchetto.

Nonostante questa libreria sia ancora ad uno stadio sperimentale, include un database di 64 protocolli principalmente relativi alla suite TCP/IP, tra i quali Ethernet, Token Ring, VLAN, IP, IPv6, TCP, UDP, DHCP, DNS, RIP, OSPF, BGP, PIM.

La libreria NetBee definisce inoltre alcuni altri linguaggi necessari per l'interscambio di dati con gli applicativi. Infatti, anche se la tecnologia NetPDL riveste un'importanza fondamentale per poter creare applicazioni che operano su qualsivoglia protocollo, è necessario definire adeguatamente i formati di interscambio dei dati tra gli applicativi e la libreria stessa, ad esempio quando un pacchetto completamente decodificato da NetBee deve poter essere visualizzato da un analizzatore di rete. NetBee include altri due linguaggi basati su XML, il *Packet Description Markup Language* (PDML) e il *Packet Summary Markup Language* (PSML), i quali definiscono rispettivamente il formato dei dati relativi ad un pacchetto di rete completamente decodificato (un file PDML riporta tutti i protocolli e i campi presenti in un certo pacchetto associati ai relativi valori), e un sommario del pacchetto stesso. Questi linguaggi possono essere utilizzati come formato di interscambio verso un applicativo che

necessita di conoscere come è fatto un pacchetto di rete: nel caso di PDML, quando necessita di conoscere quali sono i protocolli e i campi che lo compongono e i valori associati ad ogni campo (ad esempio per poter applicare determinate politiche di sicurezza basate su di essi), nel caso di PSML per conoscere le informazioni più importanti contenute nel pacchetto (ad esempio per visualizzarle in una finestra di riepilogo dei dati catturati).
Dal punto di vista della modularità delle applicazioni che necessitano di eleborazione di pacchetti di rete, NetBee fornisce un ottimo esempio di come questa possa essere realizzata in quanto implementa (efficientemente) un insieme di moduli software in grado di gestire qualsiasi protocollo (grazie al NetPDL), e definisce dei formati di interscambio tra i moduli stessi e le applicazioni.
La libreria NetBee è ragionevolmente compatta (500KBytes) ed è implementata in ambiente Windows sotto forma di libreria dinamica (DLL). Questa libreria dimostra la fattibilità, l'efficienza e la semplicità (con particolare riferimento agli sviluppatori di applicativi di alto livello) di un nuovo approccio modulare relativamente alle applicazioni che necessitano di funzionalità di processamento di pacchetti di rete. Da questo punto di vista, Analyzer beneficia in maniera sostanziale della modularità fornita attraverso NetBee: ad esempio, l'implementazione nativa di queste funzionalità in Analyzer richiederebbe una aggiunta di circa il 25% di codice (da circa 1.5Mbytes a circa 2Mbytes), relativa a funzionalità sostanzialmente secondarie per l'applicativo.

5. Conclusioni

Questo articolo contiene tre importanti contributi: propone l'utilizzo di nuove forme di modularità applicate ad applicativi che richiedono l'elaborazione di pacchetti di rete, definisce nuovi linguaggi che possono essere utilizzati per realizzare la visione precedente, e infine dimostra (attraverso la libreria NetBee) la fattibilità, l'efficienza e l'efficacia di questo approccio. In particolare, il linguaggio NetPDL può essere utilizzato per creare applicativi slegati da ogni protocollo di rete e pertanto potenzialmente capaci di supportare nuovi protocolli semplicemente aggiornando i relativi files NetPDL. Siccome questi possono essere anche interpretati a run-time, può essere addirittura possibile aggiornare il database di protocolli supportati senza far ripartire l'applicazione: ad esempio, questa funzionalità è implementata nella libreria NetBee (e in Analyzer, che è basato su di essa). Relativamente all'aggiornamento del database dei protocolli, è possibile addirittura memorizzare il database dei protocolli su Internet in modo che ogni utente possa accedere ai dati aggiornati e possa immediatamente supportare nuovi protocolli.
I linguaggi PDML e PSML possono essere visti come esempi di formati standard per l'interscambio di dati relativamente a pacchetti di rete. Grazie alla loro semplicità sono stati utilizzati anche da altri gruppi di ricerca e sono stati implementati anche in altri strumenti software quale l'analizzatore di rete Ethereal.
Dal punto di vista della modularità, probabilmente il campo di applicazione più importante è quello degli applicativi legati alla sicurezza (firewall, IDS, monitor). Infatti, in essi è necessario essere estremamente rapidi a supportare nuovi protocolli di rete (anche a livello applicativo), ed è fondamentale supportare link-

layer di diverso tipo (Ethernet, WiFi, ecc.) e pertanto sia la possibilità di creare applicativi svincolati da specifici protocolli, sia la possibilità di modularizzare gli applicativi stessi può essere di grande aiuto per gli sviluppatori.
La libreria NetBee è una implementazione efficiente che dimostra la fattibilità e le possibilità di creare applicazioni modulari per quanto riguarda l'elaborazione di pacchetti di rete. Questa libreria, che supporta principalmente la decodifica e il filtraggio dei pacchetti, è liberamente disponibile su Internet [2].

6. Bibliografia

[1] NetGroup, Politecnico di Torino, Analyzer 3.0. Disponibile all'indirizzo http://analyzer.polito.it/30alpha/, Marzo 2003.

[2] NetGroup, Politecnico di Torino, NetBee. Disponibile all'indirizzo http://www.nbee.org/, Agosto 2004.

[3] Ethereal, analizzatore di rete di pubblico dominio. Disponibile all'indirizzo http://www.ethereal.com.

[4] NetGroup, Politecnico di Torino, Il linguaggio NetPDL, Maggio 2003. Disponibile all'indirizzo http://www.nbee.org/NetPDL/.