POLITECNICO DI TORINO
Repository ISTITUZIONALE

WinPcap: una libreria Open Source per l'analisi di rete

Original
WinPcap: una libreria Open Source per l'analisi di rete / Degioanni, L.; Baldi, Mario; Risso, FULVIO GIOVANNI OTTAVIO; Varenni, G.. - (2003), pp. 237-246. (Intervento presentato al convegno AICA 2003 tenutosi a Trento nel September 15-17, 2003).

Availability:
This version is available at: 11583/1417045 since:

Publisher:
AICA

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 October 2024

WINPCAP: UNA LIBRERIA OPEN SOURCE PER L'ANALISI DI RETE

Loris Degioanni, Mario Baldi, Fulvio Risso, Gianluca Varenni
Dipartimento di Automatica e Informatica, Politecnico di Torino, Italia
{loris.degioanni,mario.baldi,fulvio.risso,gianluca.varenni}@polito.it

Sommario: in questo articolo si presenta WinPcap, una libreria Open Source per l'analisi di del traffico di rete sviluppata dal Computer Networks Group (NetGroup) presso il Politecnico di Torino. Tale libreria si è affermata come standard de-facto nel settore degli strumenti di analisi di rete per ambienti Windows.

1. INTRODUZIONE

La rapida evoluzione del settore delle reti di calcolatori, unita al vasto utilizzo delle tecnologie di networking ed alla crescente diffusione di Internet, fa sì che la potenza e la complessità delle strutture di comunicazione cresca costantemente. Tuttavia, a differenza dell'infrastruttura di tipo telefonico la cui complessità di gestione è di quasi esclusiva competenza degli operatori, il problema dell'amministrazione di una rete di calcolatori interessa anche strutture quali piccole aziende e, addirittura, utenti domestici.

Una conseguenza fondamentale è che, mentre nel caso di un operatore telefonico, il costo (pur alto) degli strumenti necessari all'amministrazione della rete può essere trascurabile, per il piccolo utente questi costi possono essere proibitivi. Il software Open Source è pertanto una possibilità interessante per questa classe di utenti; esso inoltre offre, anche maggiori doti di versatilità e di personalizzazione.

La versatilità è una parola chiave fondamentale per gli strumenti di analisi di rete; infatti anche quelli a carattere commerciale sono spesso realizzati sotto forma di oggetti software per via della maggiore versatilità rispetto a soluzioni hardware (l'hardware non si può duplicare, difficilmente si può adattare a nuove tipologie di rete ed è difficilmente personalizzabile). Esempi sono gli analizzatori di rete, le applicazioni per l'archiviazione del traffico, i programmi per il monitoraggio (sia in tempo reale che off line) della rete, per la ricerca di vulnerabilità e i sistemi per la rilevazione di attacchi (Network Intrusion Detection System). Queste applicazioni sono accomunate dalla necessità di accedere a basso livello ai dati che scorrono sulla rete, ossia necessitano di un meccanismo che estragga i pacchetti che transitano sul canale prima che questi vengano manipolati dagli stack di rete: tale operazione viene definita *cattura* o *sniffing*. Essendo tali applicazioni prevalentemente oggetti software, il mondo Open Source può giocare una parte importante nello sviluppo di nuovi prodotti, basti pensare all'analizzatore di rete Ethereal [9], che è considerato di qualità superiore a molti prodotti commerciali sviluppati da aziende blasonate.

Questo articolo descrive WinPcap, una libreria appartenente al mondo dell'Open Source che aggiunge funzionalità di cattura e di analisi di rete ai sistemi operativi Windows. Grazie alle sue prestazioni e alla sua versatilità, WinPcap è uno dei prodotti più conosciuti e apprezzati dell'Open Source italiano; è considerato uno standard per l'analisi di rete ed ha aperto la strada all'uso dei sistemi operativi Microsoft in un campo che in precedenza era appannaggio esclusivo del mondo Unix.

Gli obiettivi che hanno portato alla realizzazione di WinPcap comprendono numerosi punti. Innanzitutto, la realizzazione di un sistema di pubblico dominio per l'accesso di basso livello alla rete su sistemi Windows, ma compatibile con altre piattaforme. Infatti, la compatibilità a livello di API con altri sistemi come Linux o BSD favorisce lo sviluppo di applicazioni multipiattaforma e supporta il porting dei tool dall'ambiente Unix, in cui sono stati sviluppati molti strumenti largamente conosciuti e utilizzati, all'ambiente Windows. WinPcap si prefigge di affrontare problematiche di efficienza nel campo dell'analisi di rete e di conseguenza sviluppa un'architettura particolarmente ottimizzata in grado di supportare le applicazioni più impegnative. Un sistema con tali caratteristiche ha colmato una grave lacuna e ha stimolato lo sviluppo di moltissime nuove applicazioni, sia Open Source, sia proprietarie [6].

Il presente articolo è organizzato come segue: la sezione 2 presenta il sistema di cattura Berkeley Packet Filter (BPF), considerato lo stato dell'arte in questo settore. La sezione 3 descrive la struttura di WinPcap mostrando i vari moduli che lo compongono e la loro interazione. La quarta sezione scende nei dettagli dell'architettura, spiega il funzionamento di WinPcap e presenta alcune caratteristiche avanzate. La sezione 5 mostra i risultati di alcuni test prestazionali eseguiti sul sistema, dimostrandone l'effettiva efficienza. La sezione 6 sottolinea il contributo che WinPcap ha dato al mondo dell'Open Source. La sezione 7 presenta infine alcune conclusioni finali.

2. STATO DELL'ARTE: IL BPF

La struttura di WinPcap deriva da quella del *Berkeley Packet Filter* (BPF) [2] sviluppato presso l'Università della California. Il BPF è il sistema di cattura che viene solitamente utilizzato con i sistemi operativi della famiglia BSD. Esso è considerato la migliore e più efficiente soluzione di sniffing fra quelle offerte in ambiente Unix: l'articolo che ne presenta l'implementazione, scritto nel 1993 da McCanne e Jacobson [2], è considerato una pietra miliare in questo campo.

L'architettura descritta è in realtà costituita da due moduli ben distinti: il BPF che è un'estensione del sistema operativo, opera all'interno del kernel e si interfaccia con la scheda di rete, e una libreria funzionante a livello utente, `libpcap` [1], che esporta i servizi del BPF alle applicazioni tramite una API di alto livello. `libpcap` fornisce un buon livello di astrazione ed è pertanto utilizzabile oltre che su BSD anche su un ampio ventaglio di sistemi operativi, non necessariamente dotati del BPF (es. Linux, Solaris, AIX). Praticamente tutti i più conosciuti tool di rete per Unix si appoggiano a `libpcap` per accedere ai pacchetti, fra di essi si possono per esempio annoverare tcpdump[3], Ethereal[9], Snort[8], ntop[7].

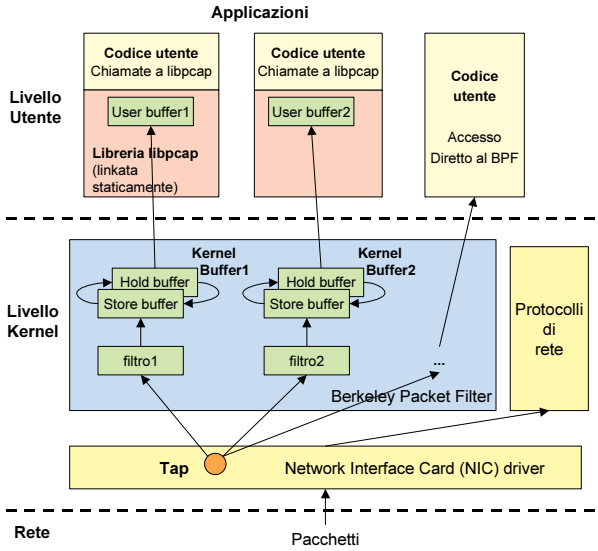

Figura 1. Struttura del BPF.

Il BPF (la cui struttura è mostrata in Figura 1) è essenzialmente implementato come un device driver che si appoggia al driver della scheda di rete (detta Network Interface Card o NIC). Il NIC driver, appena trasferito un pacchetto in memoria e prima di passarlo agli stack protocollari, invoca esplicitamente una funzione del BPF – chiamata `tap()` – che si occupa di intercettare e copiare il pacchetto.

La `tap()` effettua innanzitutto un'operazione di filtraggio, cioè determina se il pacchetto è di interesse per l'applicazione e in caso negativo lo scarta immediatamente. Se invece il pacchetto è accettato dal filtro, la `tap()` lo inserisce in un buffer allocato dal driver, detto *kernel buffer* (si veda la Figura 1). Il BPF usa un sistema a doppio buffer (per un totale di 64Kbytes) in cui una porzione di memoria contiene i pacchetti in arrivo dalla rete mentre l'altra conserva quelli che verranno passati all'applicazione. I due buffer vengono scambiati quando il primo è pieno ed il secondo è vuoto. L'applicazione preleva i pacchetti tramite una lettura (effettuata di norma con la chiamata di sistema

`read()`) sul "file" (risultante dalla virtualizzazione, fatta a livello di sistema operativo, del device driver BPF) relativo al BPF. Questa lettura attiva una funzione del BPF che sposta i pacchetti in una zona di memoria a livello utente allocata e gestita da `libpcap`.

3. ARCHITETTURA DI WINPCAP

WinPcap, come mostrato in Figura 2, ricalca nelle linee generali la struttura del BPF, aggiungendo tuttavia diversi elementi e raffinando alcuni di quelli esistenti per migliorare le prestazioni; inoltre WinPcap garantisce la compatibilità con la grande maggioranza dei sistemi operativi della famiglia Windows[1]. I prossimi paragrafi descriveranno i vari componenti dell'architettura, spiegando come interagiscono fra di loro per fornire i servizi all'utente.

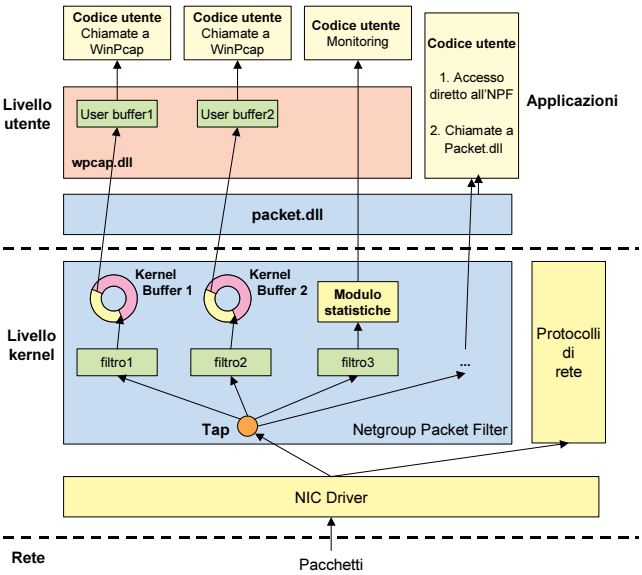

Figura 2. Architettura di WinPcap.

3.1. I COMPONENTI DI WINPCAP

WinPcap è composto da tre moduli. Il primo, ***NetGroup Packet Filter*** (***NPF***), come nel caso del BPF è un device driver che funziona a livello kernel. Il NPF è implementato come un virtual device driver (*VXD*) nel file npf.vxd in Windows 95/98/ME e come un kernel-mode driver nel file `npf.sys` in Windows NT/2000/XP. Il NPF si occupa delle operazioni di basso livello fra le quali la più importante è l'interfacciamento con la scheda di rete per la cattura dei pacchetti. Il secondo modulo è una libreria dinamica, `packet.dll`, il cui compito è quello di creare un substrato identico in tutti i sistemi operativi Windows per garantire la massima portabilità delle applicazioni. Il terzo ed ultimo modulo, `wpcap.dll`, anch'esso sotto forma di una libreria dinamica, esporta un set di funzioni di alto livello per la cattura e l'analisi di rete. `wpcap` è compatibile con quello di `libpcap` per Unix.

3.1.1. IL DRIVER NPF

Il grosso del lavoro svolto da un sistema come WinPcap è relativo all'interfacciamento con la scheda di rete tramite il driver che la governa, l'estrazione del traffico che circola sulla rete, la scrematura dei dati ottenuti in base alle specifiche dell'utente (*filtraggio*) — tutte operazioni non permesse ad una

[1] La versione di WinPcap attualmente distribuita supporta Windows 95, 98, ME, NT 4, 2000, XP e Windows Server 2003; esiste una versione sperimentale per Windows CE.

normale applicazione. Questo è il motivo per cui è necessario appoggiarsi ad un device driver, il quale rappresenta sostanzialmente un'estensione al kernel del sistema operativo.

La collocazione dell'NPF all'interno del kernel dei sistemi Windows non è semplice come nel caso del BPF. In BSD è previsto un aggancio esplicito per il driver di cattura: ai driver delle interfacce è richiesto come specifica di richiamare la `tap()` prima di eseguire ogni altra operazione. In Windows non è previsto nulla di questo tipo, inoltre non essendo disponibili né i sorgenti del kernel né quelli dei driver di rete, è impossibile creare un "aggancio" per la cattura.

La soluzione adottata consiste nell'implementazione dell'NPF come driver di protocollo. Infatti, pur non rendendo disponibili i sorgenti della pila protocollare (protocol stack) di rete, Microsoft definisce e documenta pubblicamente l'interfaccia NDIS (Network Driver Interface Specification) [4], che viene utilizzata dal kernel di Windows per far comunicare i moduli che lo compongono. Da un lato, NDIS fornisce una libreria di funzioni standard che i driver di rete possono come interfaccia con il kernel e con l'hardware in maniera ragionevolmente indipendente dal sistema, dall'altra specifica un preciso meccanismo di interazione fra i driver di rete. Sfruttando questo meccanismo è possibile scrivere e installare nuovi driver di rete senza conoscere i dettagli del resto dello pila protocollare.

La Figura 3 mostra il tipico modello a strati di NDIS e la posizione dell'NPF al suo interno. Si nota subito come esso sia in parallelo e non in cascata di con i protocolli di alto livello come il TCP/IP. Questo consente l'interfacciamento con un'ampia gamma di tecnologie di livello fisico, in quanto NDIS, sui cui NPF poggia, cerca di essere ove possibile indipendente dal formato e dalla dimensione delle trame.

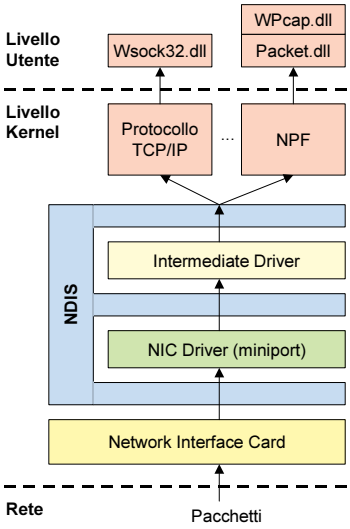

Figura 3. WinPcap e NDIS.

La collocazione dell'NPF pone anche alcuni problemi. Il primo di essi è quello delle *prestazioni*: l'interazione con NDIS è notevolmente più complessa del semplice meccanismo a callback utilizzato dal BPF. Questo ha richiesto una maggiore attenzione per ottenere prestazioni elevate; il risultato ottenuto (documentato nella sezione 5) è eccellente e le prestazioni risultanti sono addirittura migliori di quelli dei sistemi Unix da cui deriva NPF deriva. Il secondo problema è che con l'architettura di WinPcap, il NPF cattura i pacchetti solo dopo una serie di possibili manipolazioni da parte di driver di livello inferiore (quelli della scheda di rete in primis). Dunque, i pacchetti possono essere stati modificati rispetto al loro formato originale. Il caso tipico è quello del protocollo PPP oppure delle trame IEEE 802.11, le quali sono trasformate in una Ethernet "virtuale" per le sopracitate ragioni di indipendenza dal livello fisico che semplificano il lavoro dei protocolli soprastanti.

3.1.2. LE LIBRERIE

I servizi forniti dal driver sono esportati alle applicazioni tramite due librerie. La prima, `packet.dll` funziona a basso livello e gestisce l'interazione con l'NPF nei vari sistemi operativi Windows. Bisogna infatti ricordare che la struttura di un driver, le chiamate che esso usa per interagire

con il kernel e l'interfaccia per scambiare dati con le applicazioni variano radicalmente da un sistema operativo all'altro; in particolare, l'architettura di un driver per Windows 95/98/ME è del tutto differente da quella di un driver per Windows NT e successivi. `packet.dll` si occupa di gestire le differenze ed esporta una API che è consistente nei diversi ambienti. Questo risultato è stato ottenuto sviluppando una libreria diversa per ogni famiglia di sistemi operativi, e facendo sì che tutte queste librerie esportino chiamate identiche. In questo modo un programma basato su WinPcap può funzionare su tutti i sistemi Win32 senza alcuna modifica nè ricompilazione. `packet.dll` si occupa inoltre di una serie di operazioni dipendenti dal sistema ma non direttamente correlate alla cattura dei pacchetti quali il caricamento/scaricamento del driver, l'ottenimento della lista delle interfacce di rete con relativi indirizzi hardware e di livello network (ogni sistema operativo conserva questi dati in maniera differente).

`wpcap.dll` fornisce invece una API con un livello di astrazione superiore e include funzioni per creare filtri, salvare i pacchetti su disco, gestire i buffer e così via. Questa libreria non interagisce direttamente con il driver ma si appoggia a `packet.dll`, che quindi di norma non viene usata direttamente dal programmatore. Una delle caratteristiche principali di `wpcap.dll` è la compatibilità con `libpcap`. In effetti la prima può essere considerata un superset della seconda, in quanto ne fornisce tutte le chiamate aggiungendo in più una serie di funzionalità (ad esempio monitoring di rete e generazione di traffico) che sfruttano le caratteristiche avanzate del driver NPF (si veda il paragrafo 4.2).

4. PROBLEMATICHE PRESTAZIONALI

Questo capitolo descrive in maggior dettaglio l'architettura di WinPcap e in particolare del driver NPF, cuore del sistema approfondendo alcuni aspetti tecnici importanti nella determinazione delle prestazioni. Ci si soffermerà dapprima sulla funzionalità principale, la cattura dei pacchetti, evidenziando le differenze e le migliorie rispetto ai sistemi esistenti. In seguito verranno presentate alcune estensioni per l'analisi di rete ad alte prestazioni caratteristiche di WinPcap.

4.1. IL MECCANISMO DI CATTURA

La cattura di pacchetti e la loro consegna diretta alle applicazioni, operazione di per sè concettualmente semplice, è in realtà piuttosto complessa se si vogliono ottenere buone prestazioni.

La prima operazione eseguita dall'NPF in seguito alla ricezione di un pacchetto è il *filtraggio*. Filtrare significa scremare il flusso di dati diretto verso l'applicativo di cattura in base alle esigenze dell'applicazione stessa. È importante che questa operazione venga eseguita il più possibile all'inizio della catena di manipolazioni del pacchetto in modo da evitare copie superflue e spreco di spazio nei buffer. Di conseguenza il filtraggio è delegato all'NPF quando il pacchetto è ancora nella memoria del driver della scheda di rete, sfruttando la possibilità di NDIS di accedere a questa memoria.

Il sistema di filtraggio dell'NPF è compatibile con quello del BPF e consiste in un processore virtuale in grado di interpretare comandi impartiti attraverso istruzioni assembler, secondo un set di istruzioni semplice ma sufficientemente completo. Un filtro è creato dall'utente tramite un compilatore fornito in WinPcap: esso riceve in ingresso una stringa con una condizione booleana (per esempio, "*ip and not udp*") e la traduce nel corrispondente bytecode (per esempio, "*controlla che il campo* `ethertype` *della trama Ethernet sia uguale a 0x800 e che il* `protocol type` *della trama IP sia diverso da 17; in caso affermativo accetta il pacchetto*"). A differenza di tutti gli altri sistemi di cattura forniti di processore virtuale, l'NPF non interpreta i filtri, ma sfrutta tecniche di compilazione dinamica per eseguirli direttamente attraverso l'impiego di codice nativo della CPU locale. Infatti, quando un filtro viene installato dall'utente, il driver sfrutta un compilatore *just in time(JIT)* per tradurre il bytecode in una procedura in assembler x86. Questa procedura viene invocata per ogni pacchetto ricevuto e decide se questo verrà conservato oppure scartato. La compilazione JIT dei filtri porta benefici significativi in termini di efficienza, permettendo di avere un incremento della velocità di esecuzione del filtro fino a 5 volte rispetto alla sua interpretazione.

Un'altra caratteristica peculiare dell'NPF è la scelta architetturale alla base del kernel buffer. Esso è di fondamentale importanza per evitare la perdita di pacchetti nel caso di picchi di traffico di rete (*burst*) o di applicazioni con basse caratteristiche prestazionali nella fase relativa alla loro elaborazione. In WinPcap la tipica architettura a doppio buffer è stata scartata a favore di un'architettura a buffer circolare

la cui dimensione, normalmente 1 Mbyte, è configurabile dall'utente per permettere di bilanciare una maggiore tolleranza ai burst con un più efficiente utilizzo di memoria.

Il sistema di buffering dell'NPF è più complicato da gestire (soprattutto nel caso di pacchetti di dimensione variabile), ma apporta benefici non indifferenti in termini di prestazioni e utilizzo delle risorse, permettendo di sfruttare al meglio la memoria a disposizione anche in caso di burst di traffico. In tale situazione infatti è probabile che il processo di cattura e filtraggio, scatenato da un interrupt all'arrivo di un pacchetto della scheda e pertanto a priorità elevata, prenda il sopravvento sull'applicazione a livello utente impedendole di essere schedulata. Questo fa sì che essa non riesca a svuotare completamente la sua porzione di memoria. Nel sistema BPF originale questo fenomeno blocca di fatto lo scambio (swap) fra i due buffer. Il verificarsi di una situazione analoga con un buffer circolare, come realizzato nell'NPF, è altamente improbabile. Inoltre, per migliorare ulteriormente l'uso della memoria, l'NPF trasferisce i dati all'applicazione tramite una funzione di copia che aggiorna lo stato del buffer durante lo spostamento, permettendo di utilizzare immediatamente la quantità di memoria liberata anche se la fase di copia non è ancora terminata.

Nell'interazione fra driver e applicazione un parametro molto importante è il numero di context switch, cioè di transizioni da livello applicazione a livello kernel[2]. Il context switch è un processo notoriamente costoso in termini di colpi di clock. La soluzione per diminuire il numero di context switch è spostare i pacchetti fra driver e applicazione a gruppi invece che uno per volta, tecnica che permette anche di rendere meno influente il sovraccarico di lavoro relativo alla funzione di copia. L'NPF permette di specificare la quantità *minima* di dati che verrà trasferita all'applicazione con una sola chiamata: è pertanto possibile che l'applicazione non venga immediatamente informata dell'arrivo di nuovi dati da elaborare, anche se questi sono fisicamente presenti nel kernel buffer. Tramite questo meccanismo si regolarizza la frequenza dei context switch, come effetto collaterale, si ottiene che l'applicazione che usa i dati viene risvegliata solo in presenza di una certa quantità minima di dati da elaborare. Per evitare la perdita di reattività alla ricezione dei dati — caratteristica indipensabile ad alcune applicazioni — è possibile impostare un tempo massimo scaduto il quale l'applicazione verrà risvegliata qualunque sia lo stato di riempimento del buffer.

4.2. SOLUZIONI AVANZATE PER L'ANALISI DI RETE

Nonostante le ottimizzazioni presentate nei paragrafi precedenti, la cattura dei pacchetti rimane un processo dal costo non indifferente le cui prestazioni rappresentano una criticità soprattutto su reti ad alta velocità. Per supportare al meglio le applicazioni, l'NPF fornisce una serie di funzionalità avanzate che permettono un ulteriore incremento di prestazioni nel caso di operazioni ricorrenti nell'analisi di rete. L'idea di base è di spostare parte dell'elaborazione dei dati nel driver: il codice che elabora i pacchetti è posizionato nel kernel del sistema operativo e l'applicazione riceve il risultato dell'elaborazione. Ciò permette di minimizzare copie (i pacchetti non vengono più copiati all'applicativo) e context switch, con grande vantaggio per l'efficienza.

4.2.1. MONITORING DEL TRAFFICO

Spesso le applicazioni che eseguono il monitoring della rete non necessitano di tutto il pacchetto, ma solamente di un riepilogo delle caratteristiche del traffico in transito.

L'NPF è in grado di calcolare statistiche sui dati ricevuti in maniera autonoma e programmabile e di restituire all'applicazione i risultati ottenuti. I vantaggi di tale approccio sono schematizzati in Figura 4: nessun buffer viene allocato, né a livello kernel né a livello utente, di conseguenza non viene effettuato alcun tipo di copia dei pacchetti. Le statistiche sono consegnate all'utente a intervalli regolari (di norma ogni secondo) definibili tramite un'apposita chiamata. Questo minimizza il numero di context swich.

---

[2] In realtà il termine context switch è usato impropriamente non solo in questo articolo ma anche nel resto della letteratura, in quanto quello di cui si parla è un cambiamento (switch) del privilegio di esecuzione (da ring 3 a ring 0 e viceversa in ambiente Windows) e non necessariamente un cambio di contesto che è un'azione più complicata, quindi più lunga.

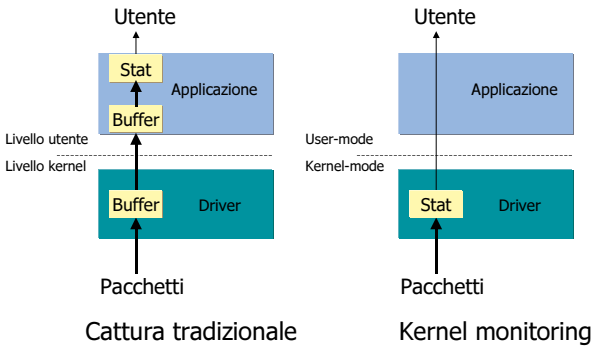

Figura 4. Confronto fra cattura e kernel monitoring.

La Figura 5 mostra come il sistema di monitoring dell'NPF sfrutti il filtro come *classificatore* di pacchetti. E possibile per esempio definire filtri che selezionino solo i pacchetti web (porta 80), oppure il traffico generato da uno specifico host. I pacchetti selezionati sono passati ad un modulo che ne esegue il conteggio tramite contatori oppure tramite una tabella di hash. I risultati del conteggio sono trasmessi all'applicazione allo scadere di un timeout.

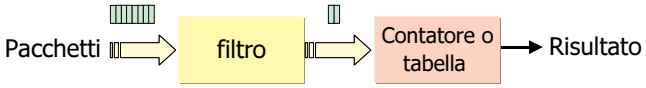

Figura 5. Funzionamento del sistema di monitoring.

4.2.2. Salvataggio del Traffico su Disco

Spesso le applicazioni basate su WinPcap salvano immediatamente il traffico su disco (*disk dump*), rimandando la vera elaborazione ad una fase successiva. Questo è un comportamento tipico degli analizzatori di rete, di molti tool per la sicurezza e in generale di tutti quei programmi che, non essendo in grado di reggere un'analisi in tempo reale del traffico, effettuano elaborazioni off-line dei pacchetti.

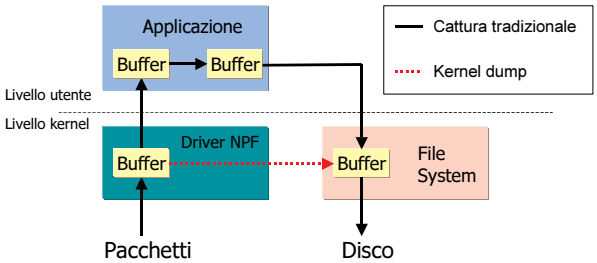

Figura 6. Dump del traffico su disco, sistemi tradizionale e ottimizzato.

In Figura 6, la freccia a linea continua evidenzia il percorso dei pacchetti quando vengono salvati su disco dopo la cattura: entrano in gioco quattro buffer (due di WinPcap, e due relativi al buffering dei dati che viene fatto dall'applicativo e dal sistema operativo durante una scrittura su disco); di conseguenza vengono eseguite quattro copie di ogni pacchetto. Inoltre sono richieste diverse transizioni da livello kernel a livello utente. La freccia a linea tratteggiata rossa mostra come è stato ottimizzato il processo: il driver NPF è in grado di comunicare direttamente con il file system trasmettendogli i pacchetti non appena vengono ricevuti dalla scheda di rete. Questa scorciatoia riduce radicalmente la necessità di memoria e il carico sulla CPU, permettendo di ottenere prestazioni nettamente superiori.

4.2.3. Generazione di Traffico

Una funzionalità molto richiesta per il collaudo di reti e apparati è la possibilità di generare traffico "grezzo", cioè trame di livello 2 che verranno spedite senza alcun tipo di elaborazione da parte della

macchina mittente. La maggior parte degli scenari di collaudo richiede un'alta efficienza nell'invio dei pacchetti (per poter stimolare la rete fino ai suoi limiti) e in ogni caso un controllo stretto sulle tempistiche con cui essi vengono spediti. Tutto questo non si può di norma realizzare con programmi a livello utente in quanto la lentezza dei context switch e la latenza imposta dall'interazione fra l'applicazione e i moduli kernel rendono le prestazioni molto basse e poco controllabili.

WinPcap offre tre modalità per la generazione di pacchetti. Nella modalità più semplice, la *scrittura singola*, è possibile inviare all'NPF un buffer contenente un pacchetto che il driver si occuperà di spedire sulla rete senza nessuna modifica. Questa modalità non brilla per la sua efficienza, ma è molto semplice da utilizzare ed è quindi ideale per i casi in cui le prestazioni non rappresentino una criticità. Questo metodo di invio è fornito da sistemi analoghi a WinPcap, come per esempio il BPF, oppure dai *raw sockets* di Windows 2000/XP.

In caso si voglia generare un traffico elevato, WinPcap fornisce la cosiddetta *scrittura ripetuta*: l'invio di un singolo pacchetto viene ripetuto dall'NPF per un numero di volte configurabile. Il pacchetto è scambiato una sola volta fra applicazione e driver, per cui viene annullato l'overhead dei context switch ed il processo diventa estremamente efficiente. Un'ulteriore e più complessa modalità di scrittura è quella *buffered* (con memorizzazione). In questo caso l'applicazione consegna al driver un buffer che contiene più pacchetti insieme a informazioni sulle tempistiche di spedizione di ognuno di essi. Il driver trasferisce l'intero buffer nella memoria del kernel ed è in grado di interpretarlo inviando i dati all'istante richiesto dall'applicazione, con precisione dell'ordine del microsecondo. Aver memorizzato i pacchetti nell'NPF permette di annullare i context switch, rendendo possibili prestazioni più elevate.

I meccanismi per la scrittura dei pacchetti (comprese l'allocazione e la gestione dei buffer) sono esportati all'utente attraverso un insieme di funzioni di `wpcap.dll`.

5. VALUTAZIONE DELLE PRESTAZIONI

E' stata effettuata un'accurata analisi delle prestazioni di WinPcap per determinare l'effettiva efficienza dell'architettura. Questo passo è indispensabile per stabilire la qualità del sistema e per capire se sia in grado di apportare reali vantaggi rispetto ad analoghi sistemi già esistenti. Inoltre, questo consente anche di determinare con precisione il peso di ogni operazione svolta durante i processi di cattura e analisi, in modo da poter individuare e rimuovere (o almeno limitare) i colli di bottiglia presenti.

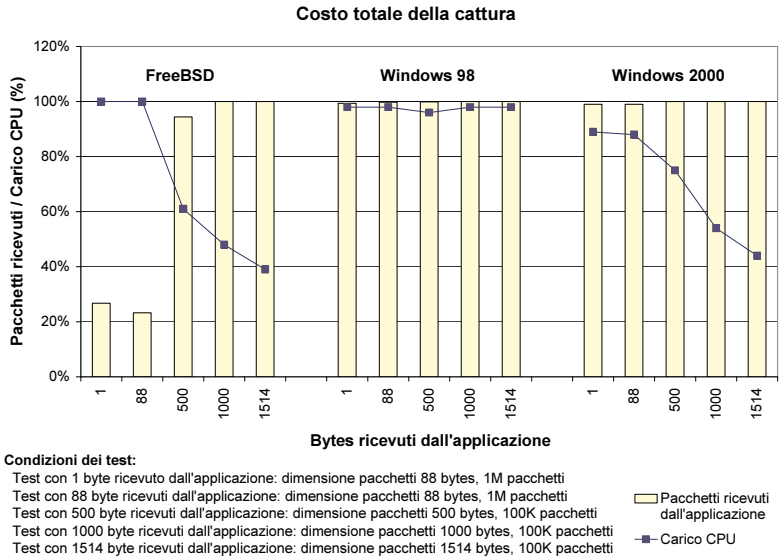

Figura 7. Costo della cattura di un pacchetto con il BPF e con WinPcap.

Questa sezione riporta i risultati più significativi. I test sono stati effettuati connettendo fra di loro due macchine tramite un collegamento Ethernet diretto. La prima macchina agisce da generatore di pacchetti, utilizzando un programma che sfrutta le funzionalità di "scrittura" di WinPcap. Sulla seconda macchina vengono effettuate le misure prestazionali. Quest'ultima possiede diverse partizioni, ognuna delle quali

ospita un diverso sistema operativo e versioni dei driver modificate per permetterne le misure, realizzate grazie all'uso di contatori interni ai processori della famiglia Pentium.

Il primo test che viene presentato mette a confronto le prestazioni — sotto forma di carico della CPU e di pacchetti effettivamente catturati — del processo di cattura nella sua totalità in diversi sistemi operativi. Il test è svolto eseguendo su tutti i sistemi una semplice applicazione che cattura tutti i pacchetti in transito scartandoli immediatamente dopo averli ricevuti. La macchina sotto test è un Pentium-II 400MHz, su cui sono installati sia sistemi operativi Windows, sia FreeBSD per consentire un confronto tra le due soluzioni; il driver BPF del FreeBSD è stato modificato in modo da destinare 1 Mbyte di memoria per il kernel buffer (la stessa quantità messa a disposizione dall'NPF di WinPcap), invece dei tradizionali 64 Kbyte. I risultati della Figura 7 mostrano chiaramente la superiorità di WinPcap. Si noti tra l'altro la maggior efficienza di Windows 2000 rispetto a Windows 98 per quel che riguarda il carico di CPU: questa differenza è dovuta alla maggior cura posta nell'ottimizzazione del driver.

La Figura 8 mostra invece il costo, in colpi di clock, per l'elaborazione di un pacchetto nel caso di cattura e di monitoring a livello kernel. Si nota subito il guadagno che un approccio come quello descritto nel paragrafo 4.2.1 può apportare: escludendo il costo fisso delle elaborazioni effettuate dal NIC driver, il miglioramento è di 8 volte.

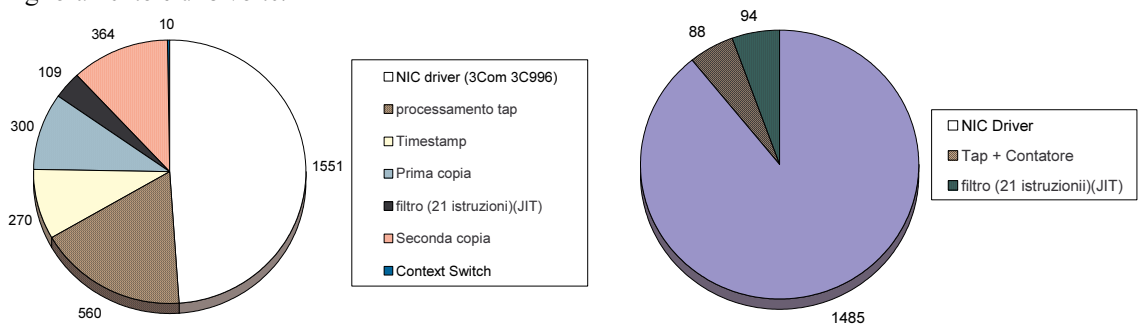

Figura 8. Costo per elaborare un pacchetto da 64 bytes nel caso di cattura e monitoring a livello kernel.

Un'ulteriore conferma a quanto appena detto arriva dalle misurazioni prestazionali sul salvataggio del traffico su disco effettuati su una workstation Pentium III a 500 MHz con Windows 2000, i cui risultati sono mostrati in Figura 9. Si può notare come le funzionalità di disk dump dell'NPF permettano di sfruttare molto meglio la CPU e di conseguenza di archiviare una quantità di dati molto maggiore a parità di prestazioni del calcolatore usato per la cattura. Il salvataggio a livello utente infatti inizia a saturare la capacità di calcolo della CPU già a 60.000 pacchetti al secondo, facendola diventare il collo di bottiglia; mediamente la versione con salvataggio diretto dei pacchetti su disco si riesce a savare un numero di pacchetti anche doppio rispetto alla soluzione a livello utente.

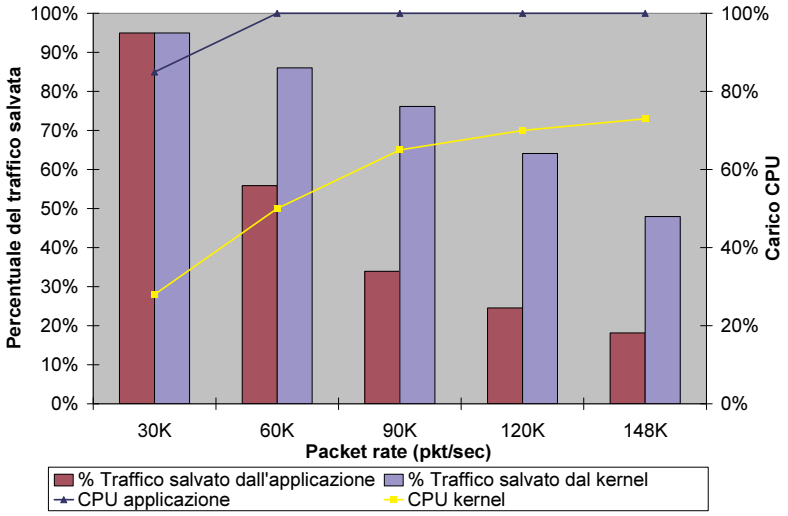

Figura 9. Costo del salvataggio del traffico su disco.

6. CONTRIBUTO AL MONDO DELL'OPEN SOURCE

WinPcap può essere considerato un fenomeno anomalo e interessante nel panorama del software Open Source per diverse ragioni. La prima è che il prodotto è focalizzato fin dalla sua nascita sul mondo Windows; non sono molti i prodotti Open Source concepiti esplicitamente per questo sistema operativo in particolar modo per quel che riguarda soluzioni di basso livello. WinPcap in quest'ottica è il primo tentativo di creare un servizio di base completamente aperto per Windows. La correttezza di tale approccio e la richiesta da parte degli utenti di soluzioni di questo tipo anche in sistemi operativi proprietari è probabilmente una delle ragioni del successo di WinPcap, che si è ormai imposto come standard di fatto contando centinaia di migliaia di utilizzatori in tutto il mondo.

Siccome la compatibilità multipiattaforma continua ad essere un punto fondamentale per molti sviluppatori, lo sviluppo di WinPcap e quello di `libpcap` si sono progressivamente avvicinati per realizzare e mantenere una libreria per la cattura pacchetti indipendente dal sistema operativo, che funzioni su tutti i sistemi Windows e Unix. Tale sforzo è, probabilmente, unico per quel che riguarda un'attività di basso livello così strettamente dipendente dal sistema, e si è rivelato vincente vista la grande quantità di applicazioni multipiattaforma sviluppati a partire dalle suddette librerie [6]. Il mondo dell'analisi di rete può oggi contare su un numero non indifferente di strumenti di altissima qualità utilizzabili su qualsiasi sistema operativo.

Il codice sorgente di WinPcap [5] e i suoi binari sono distribuiti con licenza BSD. Questa licenza è stata preferita alla più classica licenza GNU in quanto garantisce massima libertà agli utenti e non impone alcun vincolo alla modifica ed al riuso, neppure in campo commerciale.

7. CONCLUSIONI

In questo articolo è stata presentato WinPcap, una completa soluzione Open Source per l'accesso a basso livello ai servizi di rete per sistemi operativi Windows. L'articolo ha descritto l'architettura di WinPcap, evidenziando le scelte eseguite nella sua progettazione, dando risalto alle innovazioni rispetto alle soluzioni concorrenti, e presentando alcune ottimizzazioni volte al miglioramento delle prestazioni. Sono quindi stati presentati i risultati di alcuni test che quantificano la validità delle soluzioni realizzative adottate, dimostrando l'effettiva efficienza del sistema. Infine è stato sottolineato il contributo che questo software ha dato al mondo del software libero.

BIBLIOGRAFIA

[1] V. Jacobson, C. Leres and S. McCanne, libpcap, Lawrence Berkeley Laboratory, Berkeley, CA. Giugno 1994. Disponibile presso http://www.tcpdump.org/.

[2] S. McCanne and V. Jacobson, The BSD Packet Filter: A New Architecture for User-level Packet Capture. *Winter USENIX Technical Conference*, San Diego, CA, gennaio 1993.

[3] V. Jacobson, C. Leres and S. McCanne, tcpdump, Lawrence Berkeley Laboratory, Berkeley, CA, Gennaio 1991. Disponibile presso http://www.tcpdump.org/.

[4] Microsoft Corp., 3Com Corp., NDIS, Network Driver Interface Specification, Maggio 1988.

[5] Sito web di WinPcap, http://winpcap.polito.it.

[6] Tool basati su WinPcap, http://winpcap.polito.it/misc/links.htm.

[7] L. Deri e S. Suin. Effective traffic Measurement Using ntop. *IEEE Communications Magazine*, 38(5):138-145, Maggio 2000.

[8] M. Roesch. Snort, Lightweight Intrusion Detection for Networks. *LISA Conference*, Novembre 1999.

[9] Sito web di Etheral, http://www.ethereal.com.