

Scopo di questa esercitazione è:

- 1) impostare ed ottenere informazioni sulla configurazione di rete di un elaboratore;
- 2) imparare ad utilizzare un analizzatore di rete;
- 3) confrontare il comportamento di flussi TCP e UDP;
- 4) imparare l'utilizzo di tool per il monitoraggio di una rete.

1 Analisi ed impostazione della configurazione di rete

Per visualizzare le impostazioni di rete del proprio PC con Linux occorre dare il comando

```
$ /sbin/ifconfig -a
```

che visualizza la lista delle interfacce di rete e, per ognuna, le impostazioni MAC e IP. L'interfaccia ethernet viene indicata con `eth0` e si può osservare l'indirizzo MAC (6 byte esadecimali), l'indirizzo IPv4 (nel formato *dotted*) e la netmask (nel formato *dotted*).

Per visualizzare l'indirizzo del default gateway occorre usare il comando

```
$ /sbin/route
```

che stampa a video una routing table che, nel caso di un PC con una sola interfaccia IP, è ridotta a due sole righe:

- quella delle destinazioni raggiungibili direttamente (stessa rete)
- tutte le altre, raggiungibili tramite appunto il default gateway.

ESERCIZIO

Si possono usare gli stessi programmi per configurare l'indirizzo IP e la netmask in relazione ad una interfaccia di rete, e il default gateway. Ad esempio queste operazioni sono necessarie per dare operatività di rete alla macchine virtuali VMWare.

```
$ /sbin/ifconfig eth0 X.Y.Z.W netmask 255.255.255.0 up  
$ /sbin/route add default gw X.Y.Z.1
```

NOTA: chiedere al docente i valori di X.Y.Z.W

1.1 Internet Control Message Protocol (ICMP)

Il protocollo Internet Control Message Protocol (ICMP) è stato progettato per riportare anomalie che accadono nel routing di pacchetti IP e verificare lo stato della rete. ICMP è specificato nel RFC 792. La tabella seguente riporta i tipi di pacchetti ICMP. I messaggi ICMP sono imbustati nel pacchetto IP il cui campo Protocol riporta il codice 0x01. I messaggi ICMP hanno un campo *Type* che ne indica il tipo.

I messaggi che riportano anomalie sono, ad esempio, *destination unreachable*, *time exceeded for a datagram* e *parameter problem on a datagram*. I messaggi di verifica della raggiungibilità di un nodo sono *echo request* e *echo reply*. Il messaggio *redirect* indica una condizione di stimolo ad un routing

migliore, in quanto un router è stato attraversato inutilmente (ha dovuto ritrasmettere il messaggio sulla stessa rete da cui lo ha ricevuto). Quando un host riceve un pacchetto di *routing redirect* tratta l'informazione in esso contenuta in modo simile a quella specificata da un comando di `route add` ed associa quindi un router diverso da quello di default a quella destinazione.

Gli ultimi messaggi ad essere stati introdotti nel protocollo ICMP sono *address mask request* e *address mask reply*, per permettere ad una interfaccia di scoprire automaticamente la netmask usata in quella rete.

Il tool *ping* utilizza ICMP per verificare la raggiungibilità di un'interfaccia IP (di un host o di un router). Il programma viene lanciato da shell con il comando:

```
$ ping indirizzo_IP
```

Il programma invia un messaggio ICMP *Echo request* all'indirizzo IP specificato che dovrebbe rispondere con un messaggio ICMP *Echo reply*.

Esempio:

```
$ ping 157.27.10.10
```

Il programma invia più messaggi e calcola anche il round trip time a seguito della risposta.

NOTA: purtroppo presso gli amministratori di rete si sta diffondendo la consuetudine di bloccare i messaggi ICMP sul primo router attraversato nella propria rete.

1.2 Address Resolution Protocol (ARP)

Il protocollo ARP permette di conoscere l'indirizzo MAC associato ad un indirizzo IP sulla stessa rete di livello 2. Le associazioni sono mantenute in una cache temporanea da cui vengono cancellate dopo un certo tempo. Per visualizzare il contenuto della cache ARP del proprio PC occorre usare il comando

```
$ arp -a
```

1.3 Domain Name Service (DNS)

Domain Name Service (DNS) è un servizio utilizzato per la risoluzione di nomi di host in indirizzi IP. Il servizio permette così di utilizzare i nomi e le parole di uso comune per fare riferimento ad host su cui risiedono delle applicazioni, ad esempio un sito Internet. In pratica un nome di un host viene associato al suo indirizzo IP. Ad esempio il sito di Novell `www.novell.com` in realtà è solo un modo facile per identificare l'host residente all'indirizzo `130.57.5.25`. La possibilità di attribuire nomi simbolici agli indirizzi IP degli host è essenziale per l'usabilità di Internet, perché gli esseri umani trovano più facile ricordare nomi testuali, mentre gli host ed i router sono raggiungibili solo utilizzando gli indirizzi IP numerici.

Il servizio di associazione nomi/IP è realizzato tramite un database distribuito, costituito dai server DNS.

Un nome è costituito da una serie di stringhe separate da punti, ad esempio `it.wikipedia.org`. La stringa più a destra è detta *dominio di primo livello* (o TLD, Top Level Domain), per esempio `.org` o `.it`. Un dominio di secondo livello consiste in due parti, per esempio `wikipedia.org`, e così via. Ogni ulteriore stringa a sinistra specifica un'ulteriore suddivisione. Quando un dominio di secondo livello viene registrato all'assegnatario, questo è autorizzato a usare i nomi di dominio relativi ai successivi livelli come `it.wikipedia.org` (dominio di terzo livello) e altri come `some.other.stuff.wikipedia.org` (dominio di quinto livello) e così via; questo meccanismo garantisce l'unicità dei nomi.

Ogni volta che si fa accesso ad un host su Internet specificandone il nome (ad es. un sito web o un server di posta) il nostro PC chiede ad un server DNS l'indirizzo IP corrispondente che sarà poi usato per spedirgli i pacchetti.

La lista con il o i server DNS da contattare deve essere impostata dall'amministratore di rete per ogni host. Su Linux tale lista è contenuta nel file `/etc/resolv.conf` che può essere visualizzato nel modo seguente:

```
$ cat /etc/resolv.conf
```

In laboratorio tale file dovrebbe essere impostato come segue:

```
$ vi /etc/resolv.conf
search sci.univr.it
nameserver 157.27.10.10
```

La direttiva `search` indica quale dominio usare quando nelle richieste viene indicato solo il nome dell'host (dominio sottointeso). La direttiva `nameserver` invece indica l'indirizzo IP del server DNS. Un programma per shell che risolve i nomi in indirizzi IP è `dig` che invoca un server DNS e stampa a video la risposta.

Esempio:

```
$ dig www.univr.it
```

mostra l'IP del server web di Ateneo cioè 157.27.6.235

Esercizi

1: lanciare ping indicando rispettivamente l'indirizzo di loopback, il proprio IP e quello del vicino e verificare il round trip time nei tre casi. Cambia nel tempo anche per lo stesso IP ? Qual è il minore ?

2: visualizzare il contenuto della cache ARP prima e dopo aver lanciato il comando ping su un host della stessa rete e verificare cosa cambia.

2 Introduzione agli analizzatori di rete

Esistono strumenti SW che consentono di analizzare tutti i pacchetti che arrivano alla propria interfaccia di rete:

- **tcpdump** (http://www.tcpdump.org/tcpdump_man.html) storico tool in linea di comando per Linux
- **wireshark** (<http://www.wireshark.org/docs/>) nuovo tool con interfaccia grafica disponibile per Linux, Windows e Mac
- **tethereal** (<http://www.ethereal.com/docs/man-pages/tethereal.1.html>) versione in linea di comando di Wireshark
- **analyzer** (<http://analyzer.polito.it/>) sviluppato al Politecnico di Torino per Windows
- **windump** (<http://www.winpcap.org/windump>) per Windows.

Questi tool di analisi si basano tutti sulla libreria C chiamata *libpcap*, reperibile al sito <http://www.tcpdump.org> (tcpdump è lo sniffer per eccellenza che sfrutta la libpcap).

Le principali funzioni di questa libreria sono la possibilità di cercare e trovare interfacce di rete, gestire potenti filtri di cattura, analizzare ciascun pacchetto, gestire errori e statistiche di cattura.

2.1 Scaricamento e installazione

I tool si possono scaricare liberamente dalle rispettive pagine web o probabilmente sono presenti già nell'installazione della propria distribuzione Linux.

ATTENZIONE: per poter utilizzare le funzionalità di cattura di questi tool in ambiente Linux bisogna essere autenticati come utente *root* o aver installato il tool con *setuid* a root. In ogni caso questi tool possono essere utilizzati per analizzare catture precedentemente effettuate da utente root e salvate su file.

2.2 Alcuni concetti sullo *sniffing*

Sniffing in reti ethernet non-switched: In questo tipo di reti ethernet il mezzo trasmissivo è condiviso tramite un *hub* centrale, quindi tutte le schede di rete dei computer nella rete locale ricevono tutti i pacchetti, anche quelli destinati ad altri, selezionando i propri a seconda dell'indirizzo *MAC* (indirizzo hardware specifico della scheda di rete). Lo sniffing in questo caso consiste nell'impostare sull'interfaccia di rete la cosiddetta **modalità promiscua**, che disattiva il “filtro hardware” basato sul MAC permettendo al sistema l'ascolto di tutto il traffico passante sul cavo.

Sniffing in reti ethernet switched: In questo caso l'apparato centrale della rete, definito switch, si preoccupa, dopo un breve transitorio, di inoltrare su ciascuna porta solo il traffico destinato al dispositivo collegato a quella porta; ciascuna interfaccia di rete riceve, quindi solo i pacchetti destinati al proprio indirizzo, i pacchetti multicast e quelli broadcast. L'impostazione della modalità promiscua è quindi insufficiente per poter intercettare il traffico in una rete gestita da switch. Un metodo per poter ricevere tutto il traffico dallo switch da una porta qualunque è il **MAC flooding**. Tale tecnica consiste nell'inviare ad uno switch pacchetti appositamente costruiti per riempire la *CAM table* dello switch di indirizzi MAC fittizi. Questo attacco costringe lo switch ad entrare in una condizione detta di *fail open* che lo fa comportare come un hub, inviando così gli stessi dati a tutti gli apparati ad esso collegati.

2.3 Utilizzo di *tcpdump*

Questo applicativo è un tool di cattura, attraverso il quale è possibile monitorare il traffico in una rete. Il tool permette, tra l'altro, di limitare la cattura dei pacchetti impostando dei filtri basati, ad esempio, sull'interfaccia di ascolto, sul protocollo o sulla porta utilizzata. Sono inoltre disponibili una serie di opzioni, in particolare vi è la possibilità di limitare il numero di pacchetti catturati o quanti byte acquisire per ciascun pacchetto. Inoltre è possibile salvare su file i pacchetti catturati per leggerli con lo stesso programma in un secondo tempo.

Si riportano alcuni esempi di utilizzo; si noti che per catturare pacchetti occorre lanciare il programma dall'utente root mentre per analizzare file di catture precedenti si può essere utenti non privilegiati.

Utilizzo con le impostazioni di default e senza salvare i pacchetti catturati (vengono solo visualizzati):

```
$ /usr/sbin/tcpdump
```

Analizziamo il comando con le varie opzioni utilizzate:

```
$ /usr/sbin/tcpdump -i eth0 -w eth0.log -s 0 tcp port 80
```

-i eth0 : rappresenta l'interfaccia dalla quale si intende catturare

-w eth0.log : rappresenta il nome del file in cui verranno messi i pacchetti catturati

-s 0 : la flag -s permette di specificare quanti byte acquisire da ogni singolo pacchetto (default 68); con -s0 si richiede di acquisire l'intero pacchetto

tcp : le catture da effettuarsi saranno relative a questo protocollo

port 80 : questo campo limita le catture ai soli pacchetti che utilizzano come source o destination port la porta 80, che, nel caso TCP, è la porta utilizzata dal protocollo *HTTP (Hyper Text Transport Protocol)* che realizza il servizio Web.

Per interrompere il monitoraggio occorre utilizzare la combinazione *CTRL + C*.

Altri esempi di utilizzo di tcpdump:

1) viene selezionata l'interfaccia *eth0* e catturato il flusso da e verso *edalab-srv01.sci.univr.it*, scrivendo sul file *eth0.log*

```
$ /usr/sbin/tcpdump -i eth0 -w eth0.log host edalab-srv01.sci.univr.it
```

2) come il precedente esempio ma in questo caso vengono letti solo i pacchetti IP contenenti TCP.

```
$ /usr/sbin/tcpdump -i eth0 -w eth0.log \  
ip proto tcp host edalab-srv01.sci.univr.it
```

3) come il precedente ma in questo caso vengono letti solo i pacchetti da e verso la porta 80 (http).

```
$ /usr/sbin/tcpdump -i eth0 -w eth0.log \  
ip proto tcp host edalab-srv01.sci.univr.it \  
and port 23
```

4) come il precedente esempio ma in questo caso vengono letti solo i pacchetti verso l'host *edalab-srv01.sci.univr.it* (*dst* sta per *destination* mentre *src* è *source*)

```
$ /usr/sbin/tcpdump -i eth0 -w eth0.log \  
ip proto tcp \  
dst host edalab-srv01.sci.univr.it \  
and port 80
```

5) come il precedente esempio ma in questo caso vengono visualizzati i pacchetti precedentemente salvati nel file *eth0.log*; anche in questo caso si può applicare un filtro che permette di visualizzare solo i pacchetti di interesse

```
$ /usr/sbin/tcpdump -r eth0.log host edalab-srv01.sci.univr.it
```

Si ricorda che non serve essere root quando TCPDUMP legge un file precedentemente catturato.

Le capacità di analisi e filtraggio vanno ben oltre questi esempi introduttivi, si consiglia di far riferimento alle pagine del manuale per verificarne le potenzialità.

2.4 Utilizzo di *wireshark*

Alcune caratteristiche:

- i dati possono essere acquisiti “from the wire” (direttamente dal cavo) oppure possono essere letti su un file di cattura precedente
- i dati possono essere catturati dal vivo da reti Ethernet, IEEE 802.11, Token Ring, ecc.
- i dati di rete catturati possono essere esplorati tramite un'interfaccia grafica

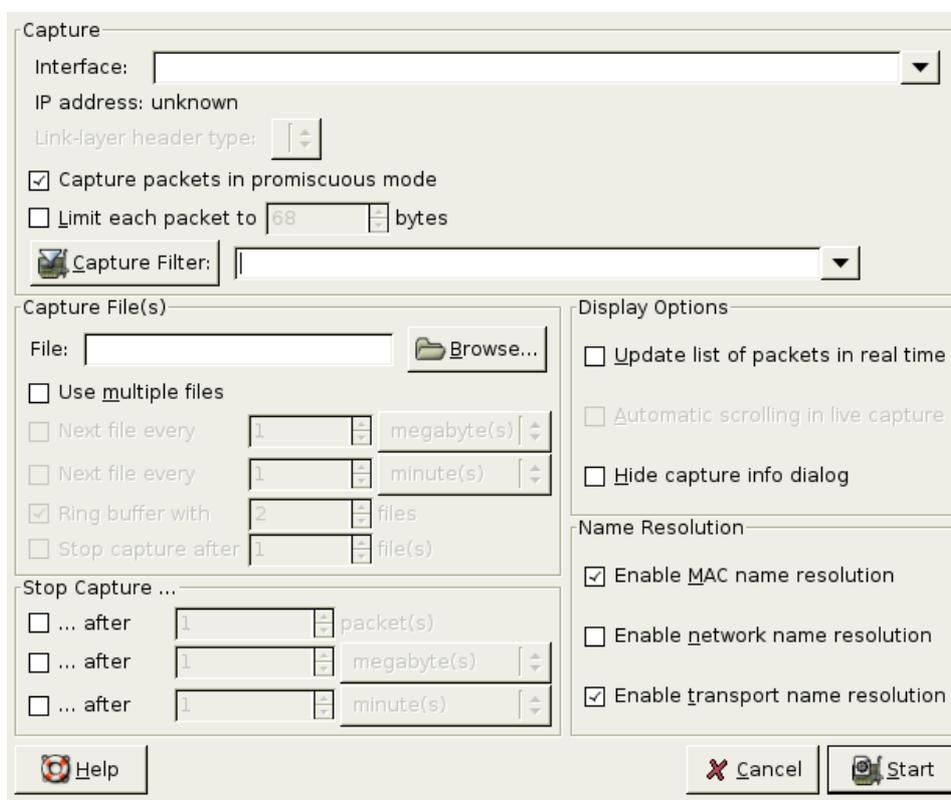
- i filtri di visualizzazione possono essere usati per colorare o visualizzare selettivamente le informazioni sommarie sui pacchetti
- i protocolli di comunicazione possono essere scomposti, in quanto wireshark riesce a “comprendere” la struttura dei diversi protocolli di rete, quindi è in grado di visualizzare incapsulamenti e campi singoli, ed di interpretare il loro significato
- è possibile studiare le statistiche di una connessione TCP e di estrarne il contenuto.

Per lanciare l'applicazione:

```
$ wireshark
```

si ricordi che è necessario essere l'utente *root* se si intende fare una cattura diretta dalle interfacce di rete.

Per avviare la cattura di pacchetti occorre prima di tutto aprire il menu *Capture/Start*.



Premendo *Start* si avvia la cattura e viene visualizzata una finestra che mostra le statistiche di cattura. Per terminare la cattura (se non si è impostato un limite in pacchetti, secondi o byte) si preme il *Stop* in basso a tale finestra.

Dopo il termine della cattura la finestra principale di wireshark mostra i dati catturati. Tale finestra è divisa in tre parti:

- tabella dei pacchetti catturati
- vista sull'imbustamento dei protocolli all'interno del pacchetto selezionato nella tabella
- vista sui dati grezzi del pacchetto selezionato nella tabella

The screenshot shows the Wireshark interface with a list of 20 captured packets. The details pane for the selected packet (No. 1) shows the following structure:

- Ethernet II, Src: Dell_a7:19:fa (00:11:43:a7:19:fa), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
- Internet Protocol, Src: 157.27.242.154 (157.27.242.154), Dst: 157.27.242.255 (157.27.242.255)
- User Datagram Protocol, Src Port: netbios-dgm (138), Dst Port: netbios-dgm (138)
- NetBIOS Datagram Service
- SMB (Server Message Block Protocol)
- SMB MailSlot Protocol
- Microsoft Windows Browser Protocol

The packet bytes pane shows the following hex dump:

```

0000 ff ff ff ff ff ff 00 11 43 a7 19 fa 08 00 45 00 ..... C.....E.
0010 00 df ed ff 00 00 40 11 6c 3d 9d 1b f2 9a 9d 1b .....@.l=.....
0020 f2 ff 00 8a 00 8a 00 cb 1f aa 11 02 f2 cb 9d 1b .....
0030 f2 9a 00 8a 00 b5 00 00 20 45 42 45 44 45 44 45 ..... EBEDEDE
0040 46 45 4d 45 46 46 43 45 42 46 45 45 50 46 43 45 FEMEFFFF BFEFFPCE

```

E' possibile vedere un sommario sulle statistiche scegliendo il menu `Statistics/Summary`.

The screenshot shows the `Statistics/Summary` dialog box with the following information:

- Name:** /mnt/findus/Documents and Settings/davide/Documenti/didattica/reti/laboratorio/aa0607/analisi-rete/cattura.cap
- Length:** 47723 bytes
- Format:** libpcap (tcpdump, Ethereal, etc.)
- Packet size limit:** 65535 bytes
- Time:**
 - First packet: 2007-06-08 09:22:23
 - Last packet: 2007-06-08 09:23:22
 - Elapsed: 00:00:59
- Capture:**
 - Interface: unknown
 - Dropped packets: unknown
 - Capture filter: unknown
- Display:**
 - Display filter: none
 - Marked packets: 0

Traffic	Captured	Displayed
Between first and last packet	59.040 sec	
Packets	285	
Avg. packets/sec	4.827	
Avg. packet size	151.000 bytes	
Bytes	43139	
Avg. bytes/sec	730.674	
Avg. MBit/sec	0.006	

Close

E' possibile anche visualizzare un sommario sui pacchetti catturati con le informazioni ordinate per protocolli e visualizzate nell'ordine in cui i protocolli sono incapsulati. Le colonne la cui etichetta inizia con "End" indicano le informazioni quando i corrispondenti protocolli sono nell'imbustamento più interno del pacchetto.

Protocol	% Packets	Packets	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100.00%	285	43139	0.006	0	0	0.000
Ethernet	100.00%	285	43139	0.006	0	0	0.000
Internet Protocol	67.02%	191	36636	0.005	0	0	0.000
User Datagram Protocol	52.98%	151	30721	0.004	0	0	0.000
NetBIOS Datagram Service	6.67%	19	4480	0.001	0	0	0.000
SMB (Server Message Block Protocol)	6.67%	19	4480	0.001	0	0	0.000
SMB MailSlot Protocol	6.67%	19	4480	0.001	0	0	0.000
Microsoft Windows Browser Protocol	6.67%	19	4480	0.001	19	4480	0.001
NetBIOS Name Service	9.47%	27	2484	0.000	27	2484	0.000
Data	10.18%	29	4921	0.001	29	4921	0.001
Routing Information Protocol	4.21%	12	6472	0.001	12	6472	0.001
Bootstrap Protocol	1.75%	5	1738	0.000	5	1738	0.000
Common Unix Printing System (CUPS) Browsing Protocol	0.70%	2	474	0.000	2	474	0.000
Domain Name Service	6.67%	19	3413	0.000	19	3413	0.000
Hypertext Transfer Protocol	5.96%	17	4721	0.001	17	4721	0.001
Service Location Protocol	6.67%	19	1630	0.000	19	1630	0.000
EtherNet/IP (Industrial Protocol)	0.70%	2	388	0.000	0	0	0.000
Malformed Packet	0.70%	2	388	0.000	2	388	0.000
Transmission Control Protocol	13.33%	38	5795	0.001	20	1336	0.000
Post Office Protocol	3.51%	10	3009	0.000	10	3009	0.000
Data	2.81%	8	1450	0.000	8	1450	0.000
Internet Group Management Protocol	0.70%	2	120	0.000	2	120	0.000
Address Resolution Protocol	25.96%	74	4440	0.001	74	4440	0.001
Internet Protocol Version 6	7.02%	20	2063	0.000	0	0	0.000
	2.11%	6	516	0.000	0	0	0.000
Internet Control Message Protocol v6	2.11%	6	516	0.000	6	516	0.000
Internet Control Message Protocol v6	2.81%	8	576	0.000	8	576	0.000
User Datagram Protocol	2.11%	6	971	0.000	0	0	0.000

E' possibile limitare la cattura ai soli pacchetti che rispettano certi requisiti impostando un filtro di cattura nel menu Capture/Options. Nella sezione apposita è possibile scrivere l'espressione del filtro usando il linguaggio riportato in Appendice oppure selezionare un filtro pre-esistente cliccando sull'apposito bottone. I filtri di cattura si usano quando la quantità di pacchetti che passano sul tratto di rete è tale da mettere in crisi le prestazioni della macchina.

E' possibile migliorare la visualizzazione dei vari pacchetti nella tabella principale colorando le righe in base al tipo di protocolli o indirizzi coinvolti. I filtri di coloramento si possono impostare nel menu View/Colouring rules.

Edit

New

Edit...

Delete

Manage

Export...

Import...

Clear

Filter

[List is processed in order until match is found]

Name	String
Bad TCP	tcp.analysis.flags
HSRP State Change	hsrp.state != 8 && hsrp.state != 16
Spanning Tree Topology Change	stp.type == 0x80
OSPF State Change	ospf.msg != 1
ICMP errors	icmp.type eq 3 icmp.type eq 4 icmp.type e
ARP	arp
ICMP	icmp
TCP RST	tcp.flags.reset eq 1
Low TTL	ip.ttl < 5
Checksum Errors	edp.checksum_bad==1 ip.checksum_bad==

Order

Up

Move selected filter up or down

Down

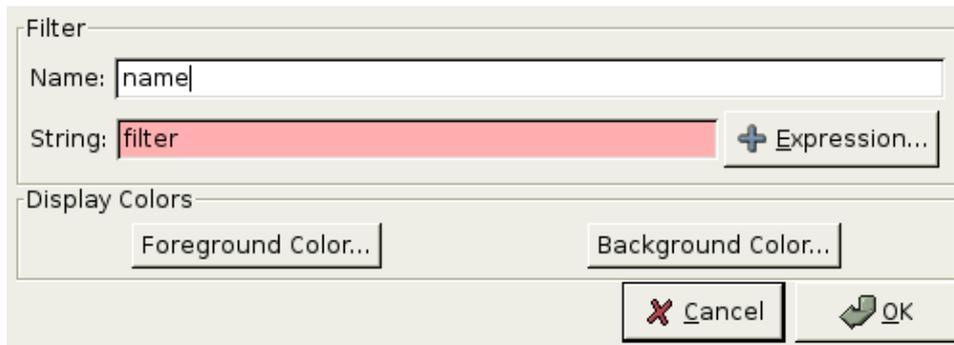
Save

Apply

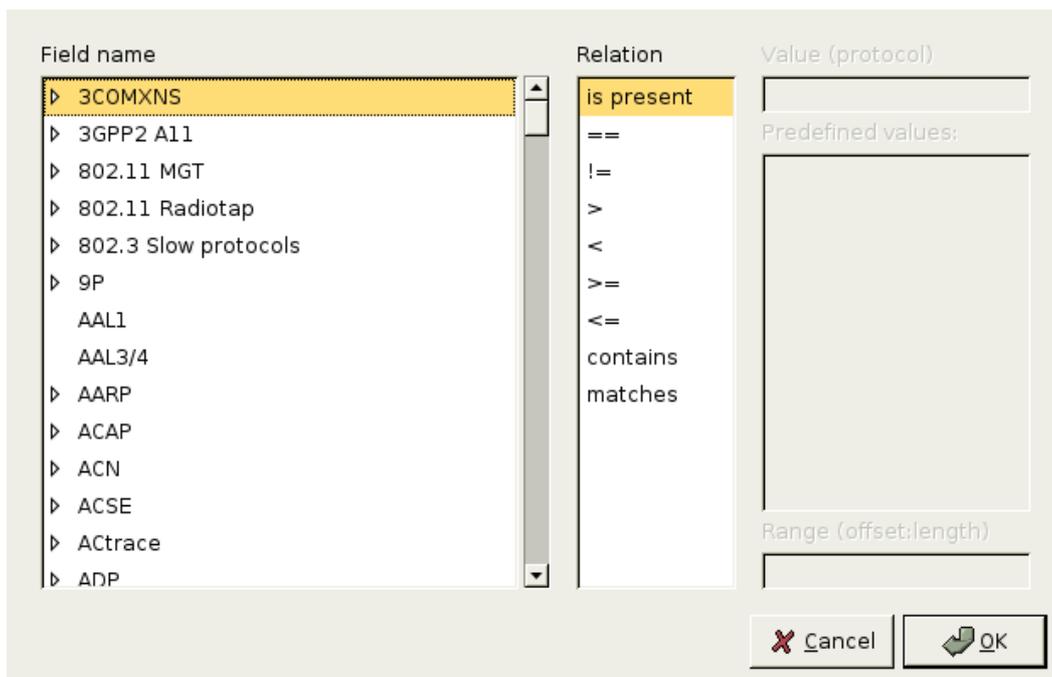
Close

OK

E' possibile creare nuovi filtri con il bottone New che apre una finestra in cui si può impostare nome, regole e colori. Le regole si impostano con un linguaggio diverso da quello dei filtri di cattura (si noti il pulsante Expression che semplifica la scrittura delle regole combinando campi dei pacchetti, valori e operatori logici).



E' possibile anche visualizzare solo le righe della tabella principale che rispettano certe regole inserite in una apposita riga Filter. Anche in questo caso le regole seguono una sintassi diversa da quella dei filtri di cattura ma il pulsante Expression semplifica la loro scrittura perché apre una finestra dove è possibile legare campi dei pacchetti, valori e operatori logici. Cliccando sul bottone Filter: è possibile invece selezionare un filtro predefinito.



Per quanto riguarda TCP, selezionando un pacchetto TCP nella finestra principale, è possibile seguire l'intero flusso di dati di quella "conversazione" mediante la voce Analyze/Follow TCP Stream e studiare l'andamento di alcuni parametri del protocollo mediante la voce Statistics/TCP stream graph.

Esercizio

Provare a catturare i pacchetti nelle seguenti condizioni:

1. nessuna attività volontaria sul proprio PC; confrontare cattura in modalità promiscua e non.
2. ping a 127.0.0.1 e al proprio IP: cosa cambia ?

3. ping alla macchina del vicino
4. accesso a <http://www.google.it>. Visualizzare tutti i pacchetti cercando di dedurre la dinamica del protocollo e ricavare il contenuto di dati applicativi che passano nella connessione.
5. applicazione di posta. “sniffare” la password dell'utente *reti* che si connette all'host del docente tramite un client di posta.

Si raccomanda di usare dei filtri di visualizzazione per isolare il traffico a cui si è interessati.

3 Confronto tra TCP e UDP

Scopo di questo capitolo è imparare a generare e analizzare flussi TCP e UDP. Per questo scopo si utilizza un tool chiamato `nttcp` che viene eseguito tra due macchine con il ruolo di trasmettitore e ricevitore, rispettivamente. Il ricevitore si mette in attesa di dati dal trasmettitore che, una volta avviato, comincia a trasmettere dati usando il livello trasporto deciso e i parametri impostati.

Esercizio A

Mettendosi a coppie, un membro lancia il programma come ricevitore:

```
$ nttcp -r -l 8192
```

il ricevitore si mette ad ascoltare sulla porta di default l'instaurazione di una connessione TCP (per l'opzione `-l` si veda dopo). A questo punto entrambi i membri lancino la cattura di pacchetti e successivamente l'altro membro lanci il programma come trasmettitore impostando come destinatario l'indirizzo IP del ricevitore e dicendo di spedire un *buffer* (opzione `-n`) di lunghezza 8192 byte (opzione `-l`). Dopo che entrambi i programmi sono usciti si fermano le catture.

```
$ nttcp -t -l 8192 -n 1 X.Y.Z.W
```

Domande:

1. Esistono pacchetti speciali che precedono/seguono la trasmissione dei dati ?
2. Quanti byte sono stati spediti dall'applicazione ? Come sono distribuiti nei vari pacchetti IP ?
3. A fronte della quantità di dati dichiarati dall'applicazione, quanti byte sono trasmessi sul cavo (suggerimento: usare Protocol Hierarchy Statistics) ?
4. Qual è il tempo utilizzato per il trasferimento: dal punto di vista del trasmettitore ? del ricevitore ?
5. Selezionare un pacchetto del flusso TCP e visualizzare il Time-Sequence Graph. Come cambia il grafico a seconda che si selezioni un pacchetto TX->RX e un pacchetto RX->TX ?
6. Come al punto precedente ma visualizzando il numero di byte/s con Statistics/IO Graphs

Esercizio B

Tutto come l'Esercizio A ma aggiungendo nella linea di comando di entrambe le istanze l'opzione `-u` per indicare di usare UDP invece di TCP.

Confrontare le risposte alle 6 domande nei due casi.

Esercizio C

Mettendosi a coppie, un membro lancia il programma come ricevitore:

```
$ nttcp -r -l 1000
```

il ricevitore si mette ad ascoltare sulla porta di default l'instaurazione di una connessione TCP (per l'opzione -l si veda dopo). A questo punto entrambi i membri lancino la cattura di pacchetti e successivamente l'altro membro lanci il programma come trasmettitore impostando come destinatario l'indirizzo IP del ricevitore e dicendo di spedire 50 *buffer* (opzione -n) di lunghezza 1000 byte (opzione -l) . Durante la trasmissione staccare il cavo di rete del ricevitore per 1 secondo. Dopo che entrambi i programmi sono usciti si fermano le catture.

```
$ nttcp -t -l 1000 -n 50 X.Y.Z.W
```

Domande:

1. Si notano delle differenze tra la lista dei pacchetti catturati al trasmettitore e al ricevitore ?
2. Si utilizzi sul PC trasmettitore il filtro di visualizzazione per le ritrasmissioni `tcp.analysis.retransmission` se presente nella distribuzione. Che cosa si nota nel numero di sequenza dei pacchetti visualizzati ?
3. Tornare a visualizzare tutti i pacchetti TCP. Sul PC trasmettitore selezionare un pacchetto di dati e visualizzare il Time-Sequence Graphs (Steven's) e il throughput (con `Statistics/IO Graphs`). Cosa si nota ?

Provare a riprodurre altre situazioni anomale verificando cosa rispondono le istanze di `nttcp` (non serve catturare pacchetti ma i curiosi possono farlo):

1. lanciare il trasmettitore senza aver lanciato il ricevitore (attendere poco/molto prima di lanciare il ricevitore)
2. killare il ricevitore mentre è in corso un trasferimento.

Esercizio D

Tutto come l'Esercizio C ma aggiungendo nella linea di comando di entrambe le istanze l'opzione -u per indicare di usare UDP invece di TCP.

Confrontare le risposte alle 5 domande nei due casi (quindi anche prove sulle situazioni anomale).

4 Strumenti per il monitoraggio della rete

4.1 Snort

coming soon...

4.2 Look@Lan

coming soon...

APPENDICE A. Sintassi dei filtri di cattura

La descrizione della sintassi è presa dalla man page di tcpdump.

An expression selects which packets will be dumped. If no expression is given, all packets on the net will be dumped. Otherwise, only packets for which expression is 'true' will be dumped.

The expression consists of one or more primitives. Primitives usually consist of an id (name or number) preceded by one or more qualifiers. There are three different kinds of qualifier:

type qualifiers say what kind of thing the id name or number refers to. Possible types are **host**, **net** and **port**. E.g., 'host foo', 'net 128.3', 'port 20'. If there is no type qualifier, **host** is assumed.

dir qualifiers specify a particular transfer direction to and/or from *id*. Possible directions are **src**, **dst**, **src or dst** and **src and dst**. E.g., 'src foo', 'dst net 128.3', 'src or dst port ftp-data'. If there is no dir qualifier, **src or dst** is assumed. For 'null' link layers (i.e. point to point protocols such as slip) the **inbound** and **outbound** qualifiers can be used to specify a desired direction.

proto qualifiers restrict the match to a particular protocol. Possible protos are: **ether**, **fddi**, **tr**, **ip**, **ip6**, **arp**, **rarp**, **decnet**, **tcp** and **udp**. E.g., 'ether src foo', 'arp net 128.3', 'tcp port 21'. If there is no proto qualifier, all protocols consistent with the type are assumed. E.g., 'src foo' means '(ip or arp or rarp) src foo' (except the latter is not legal syntax), 'net bar' means '(ip or arp or rarp) net bar' and 'port 53' means '(tcp or udp) port 53'.

['fddi' is actually an alias for 'ether'; the parser treats them identically as meaning 'the data link level used on the specified network interface.' FDDI headers contain Ethernet-like source and destination addresses, and often contain Ethernet-like packet types, so you can filter on these FDDI fields just as with the analogous Ethernet fields. FDDI headers also contain other fields, but you cannot name them explicitly in a filter expression.

Similarly, 'tr' is an alias for 'ether'; the previous paragraph's statements about FDDI headers also apply to Token Ring headers.]

In addition to the above, there are some special 'primitive' keywords that don't follow the pattern: **gateway**, **broadcast**, **less**, **greater** and arithmetic expressions. All of these are described below.

tives. E.g., 'host foo and not port ftp and not port ftp-data'. To save typing, identical qualifier lists can be omitted. E.g., 'tcp dst port ftp or ftp-data or domain' is exactly the same as 'tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain'.

Allowable primitives are:

dst host *host*

True if the IPv4/v6 destination field of the packet is *host*, which may be either an address or a name.

src host *host*
 True if the IPv4/v6 source field of the packet is *host*.

host *host*
 True if either the IPv4/v6 source or destination of the packet is *host*. Any of the above host expressions can be prepended with the keywords, **ip**, **arp**, **rarp**, or **ip6** as in:
ip host *host*
 which is equivalent to:
ether proto \ip and host *host*
 If *host* is a name with multiple IP addresses, each address will be checked for a match.

ether dst *ehost*
 True if the ethernet destination address is *ehost*. *Ehost* may be either a name from /etc/ethers or a number (see **ethers(3N)** for numeric format).

ether src *ehost*
 True if the ethernet source address is *ehost*.

ether host *ehost*
 True if either the ethernet source or destination address is *ehost*.

gateway *host*
 True if the packet used *host* as a gateway. I.e., the ethernet source or destination address was *host* but neither the IP source nor the IP destination was *host*. *Host* must be a name and must be found both by the machine's host-name-to-IP-address resolution mechanisms (host name file, DNS, NIS, etc.) etc.). (An equivalent expression is **ether host ehost and not host** *host* which can be used with either names or numbers for *host* / *ehost*.) This syntax does not work in IPv6-enabled configuration at this moment.

dst net *net*
 True if the IPv4/v6 destination address of the packet has a network number of *net*. *Net* may be either a name from /etc/networks or a network number (see *networks(4)* for details).

src net *net*
 True if the IPv4/v6 source address of the packet has a network number of *net*.

net *net*
 True if either the IPv4/v6 source or destination address of the packet has a network number of *net*.

net net mask *netmask*
 True if the IP address matches *net* with the specific *netmask*. May be qualified with **src** or **dst**. Note that this syntax is not valid for IPv6 *net*.

net net/len
 True if the IPv4/v6 address matches *net* with a netmask *len* bits wide. May be qualified with **src** or **dst**.

dst port *port*
 True if the packet is ip/tcp, ip/udp, ip6/tcp or ip6/udp and has a destination port value of *port*. The *port* can be a number or a name used in /etc/services (see

tcp(4P) and **udp(4P)**). If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port number is checked (e.g., **dst port 513** will print both tcp/login traffic and udp/who traffic, and **port domain** will print both tcp/domain and udp/domain traffic).

src port *port*

True if the packet has a source port value of *port*.

True if either the source or destination port of the packet is *port*. Any of the above port expressions can be prepended with the keywords, **tcp** or **udp**, as in:

tcp src port *port*

which matches only tcp packets whose source port is *port*.

less *length*

True if the packet has a length less than or equal to *length*. This is equivalent to:

len <= *length*.

greater *length*

True if the packet has a length greater than or equal to *length*. This is equivalent to:

len >= *length*.

ip proto *protocol*

True if the packet is an IP packet (see **ip(4P)**) of protocol type *protocol*. *Protocol* can be a number or one of the names *icmp*, *icmp6*, *igmp*, *igrp*, *pim*, *ah*, *esp*, *vrrp*, *udp*, or *tcp*. Note that the identifiers *tcp*, *udp*, and *icmp* are also keywords and must be escaped via backslash (\), which is \\ in the C-shell. Note that this primitive does not chase the protocol header chain.

ip6 proto *protocol*

True if the packet is an IPv6 packet of protocol type *protocol*. Note that this primitive does not chase the protocol header chain.

ip6 protochain *protocol*

True if the packet is IPv6 packet, and contains protocol header with type *protocol* in its protocol header chain. For example,

ip6 protochain 6

matches any IPv6 packet with TCP protocol header in the protocol header chain. The packet may contain, for example, authentication header, routing header, or hop-by-hop option header, between IPv6 header and TCP header. The BPF code emitted by this primitive is complex and cannot be optimized by BPF optimizer code in *tcpdump*, so this can be somewhat slow.

ip protochain *protocol*

Equivalent to **ip6 protochain** *protocol*, but True if the packet is an ethernet broadcast packet. The *ether* keyword is optional.

ip broadcast

True if the packet is an IP broadcast packet. It checks for both the all-zeroes and all-ones broadcast conventions, and looks up the local subnet mask.

ether multicast

True if the packet is an ethernet multicast packet. The *ether* keyword is optional. This is shorthand for **`ether[0] & 1 != 0'**.

ip multicast
True if the packet is an IP multicast packet.

ip6 multicast
True if the packet is an IPv6 multicast packet.

ether proto protocol
True if the packet is of ether type protocol. Protocol can be a number or one of the names *ip*, *ip6*, *arp*, *rarp*, *atalk*, *aarp*, *decnet*, *sca*, *lat*, *mopdl*, *moprc*, *iso*, *stp*, *ipx*, or *netbeui*. Note these identifiers are also keywords and must be escaped via backslash (\).

[In the case of FDDI (e.g., **fddi protocol arp**) and Token Ring (e.g., **tr protocol arp**), for most of those protocols, the protocol identification comes from the 802.2 Logical Link Control (LLC) header, which is usually layered on top of the FDDI or Token Ring header.

When filtering for most protocol identifiers on FDDI or Token Ring, *tcpdump* checks only the protocol ID field of an LLC header in so-called SNAP format with an Organizational Unit Identifier (OUI) of 0x000000, for encapsulated Ethernet; it doesn't check whether the packet is in SNAP format with an OUI of 0x000000.

The exceptions are *iso*, for which it checks the DSAP (Destination Service Access Point) and SSAP (Source Service Access Point) fields of the LLC header, *stp* and *netbeui*, packet with an OUI of 0x080007 and the Appletalk etype.

In the case of Ethernet, *tcpdump* checks the Ethernet type field for most of those protocols; the exceptions are *iso*, *sap*, and *netbeui*, for which it checks for an 802.3 frame and then checks the LLC header as it does for FDDI and Token Ring, *atalk*, where it checks both for the Appletalk etype in an Ethernet frame and for a SNAP-format packet as it does for FDDI and Token Ring, *aarp*, where it checks for the Appletalk ARP etype in either an Ethernet frame or an 802.2 SNAP frame with an OUI of 0x000000, and *ipx*, where it checks for the IPX etype in an Ethernet frame, the IPX DSAP in the LLC header, the 802.3 with no LLC header encapsulation of IPX, and the IPX etype in a SNAP frame.]

decnet src host
True if the DECNET source address is *host*, which may be an address of the form `10.123`, or a DECNET host name. [DECNET host name support is only available on Ultrix systems that are configured to run DECNET.]

decnet dst host
True if the DECNET destination address is *host*.

decnet host host
True if either the DECNET source or destination address is *host*.

ip, ip6, arp, rarp, atalk, aarp, decnet, iso, stp, ipx, netbeui
Abbreviations for:

ether proto p
where *p* is one of the above protocols.

lat, moprc, mopdl

Abbreviations for:

ether proto p

where *p* is one of the above protocols. Note that *tcpdump* does not currently know how to parse these protocols.

vlan [vlan_id]

True if the packet is an IEEE 802.1Q VLAN packet. If *[vlan_id]* is specified, only encountered in *expression* changes the decoding offsets for the remainder of *expression* on the assumption that the packet is a VLAN packet.

tcp, udp, icmp

Abbreviations for:

ip proto p or ip6 proto p

where *p* is one of the above protocols.

iso proto protocol

True if the packet is an OSI packet of protocol type *protocol*. *Protocol* can be a number or one of the names *clnp*, *esis*, or *isis*.

clnp, esis, isis

Abbreviations for:

iso proto p

where *p* is one of the above protocols. Note that *tcpdump* does an incomplete job of parsing these protocols.

expr relop expr

True if the relation holds, where *relop* is one of *>*, *<*, *>=*, *<=*, *=*, *!=*, and *expr* is an arithmetic expression composed of integer constants (expressed in standard C syntax), the normal binary operators *+*, *-*, ***, */*, *&*, *|*, a length operator, and special packet data accessors. To access data inside the packet, use the following syntax:

proto [expr : size]

Proto is one of **ether**, **fddi**, **tr**, **ip**, **arp**, **rarp**, **tcp**, **udp**, **icmp** or **ip6**, and indicates the protocol layer for the index operation. Note that *tcp*, *udp* and other upper-layer protocol types only apply to IPv4, not IPv6 (this will be fixed in the future). The byte offset, relative to the indicated protocol layer, is given by *expr*. *Size* is optional and indicates the number of bytes in the field of interest; it can be either one, two, or four, and defaults to one. The length operator, indicated by the keyword **len**, gives the length of the packet.

For example, **`ether[0] & 1 != 0`** catches all multicast traffic. The expression **`ip[0] & 0xf != 5`** catches all IP packets with options. The expression **`ip[6:2] & 0x1fff = 0`** catches only unfragmented datagrams and frag zero of fragmented datagrams. This always means the first byte of the TCP header, and never means the first byte of an intervening fragment.

Some offsets and field values may be expressed as names rather than as numeric values. The following protocol header field offsets are available: **icmptype** (ICMP type field), **icmpcode** (ICMP code field), and **tcpflags** (TCP flags field).

The following ICMP type field values are available: **icmp-echoreply**, **icmp-unreach**,

icmp-sourcequench, icmp-redirect, icmp-echo, icmp-routeradvert, icmp-routersolicit, icmp-timxceed, icmp-paramprob, icmp-tstamp, icmp-tstampreply, icmp-ireq, icmp-ireqreply, icmp-maskreq, icmp-maskreply.

The following TCP flags field values are available: **tcp-fin, tcp-syn, tcp-rst, tcp-push, tcp-push, tcp-ack, tcp-urg.**

Primitives may be combined using:

A parenthesized group of primitives and operators (parentheses are special to the Shell and must be escaped).

Negation (**'!' or 'not'**).

Concatenation (**'&&' or 'and'**).

Alternation (**'||' or 'or'**).

Negation has highest precedence. Alternation and concatenation have equal precedence and associate left to right. Note that explicit **and** tokens, not juxtaposition, are now required for concatenation.

If an identifier is given without a keyword, the most recent keyword is assumed. For example,

not host vs and ace
is short for
not host vs and host ace
which should not be confused with
not (host vs or ace)

Expression arguments can be passed to *tcpdump* as either a single argument or as multiple arguments, whichever is more convenient. Generally, if the expression contains Shell metacharacters, it is before being parsed.

EXAMPLES

To print all packets arriving at or departing from *sundown*:

```
tcpdump host sundown
```

To print traffic between *helios* and either *hot* or *ace*:

```
tcpdump host helios and \( hot or ace \)
```

To print all IP packets between *ace* and any host except *helios*:

```
tcpdump ip host ace and not helios
```

To print all traffic between local hosts and hosts at Berkeley:

```
tcpdump net ucb-ether
```

To print all ftp traffic through internet gateway *snup*: (note that the expression is quoted to prevent the shell from (mis-)interpreting the parentheses):

```
tcpdump 'gateway snup and (port ftp or ftp-data)'
```

To print traffic neither sourced from nor destined for local hosts (if you gateway to one other net, this stuff should never make it onto your local net).

```
tcpdump ip and not net localnet
```

To print the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a non-local host.

```
tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net localnet'
```

To print IP packets longer than 576 bytes sent through gateway *snup*:

```
tcpdump 'gateway snup and ip[2:2] > 576'
```

To print IP broadcast or multicast packets that were not

sent via ethernet broadcast or multicast:

```
tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'
```

To print all ICMP packets that are not echo requests/replies (i.e., not ping packets):

```
tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'
```