UNIVERSITA' DEGLI STUDI DELL'INSUBRIA

Facoltà di Informatica

Corso di Laurea di Informatica

Sviluppo di un software di Packet-Filtering e Port-Filtering su reti TCP/IP

Relazione di Stage di

Stefano Nichele

Relatore

Elisabetta Binaghi

Correlatore

Riccardo Rovelli

Luglio 2007

INDICE

Software di Packet-Filtering e Port-Filtering su reti TCP/IP

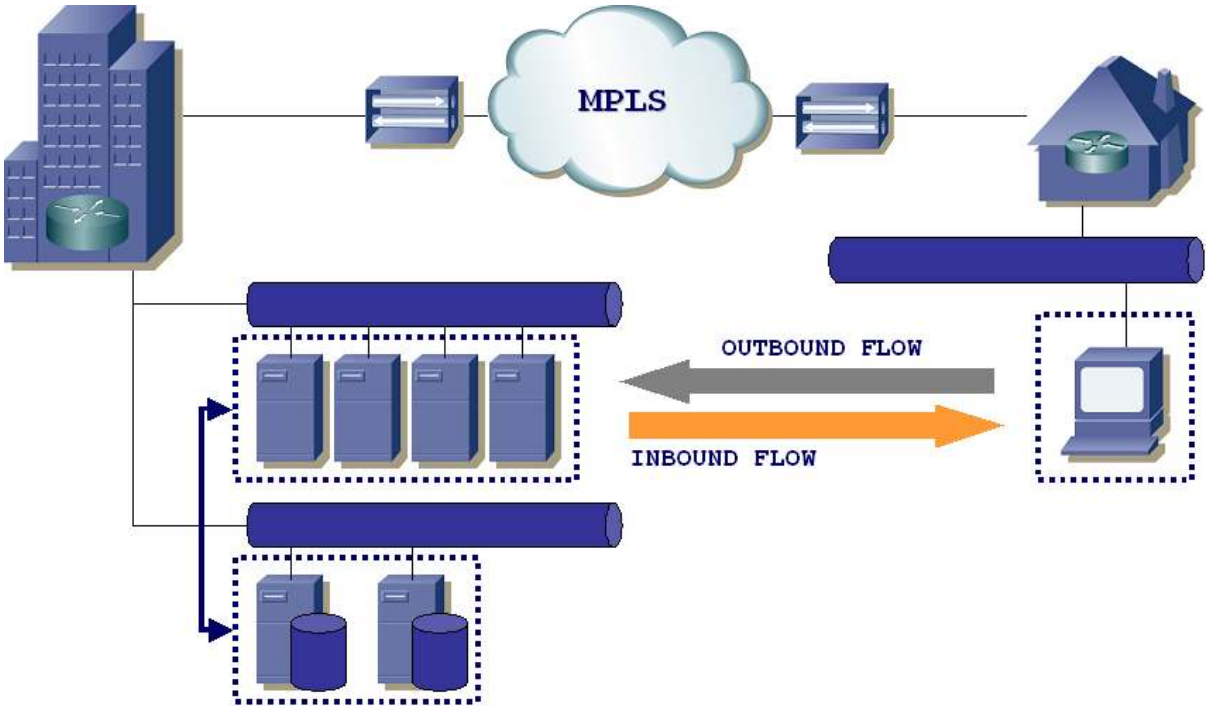
OBBIETTIVO DELLA TESI

L’obbiettivo di questa tesi di laurea è quello di analizzare, documentare e risolvere le problematiche di sicurezza relative alla rete geografica privata di proprietà dell’azienda XXXXX (cliente dell’azienda Gruppo Reti s.p.a.), alla quale sono interconnesse delle postazioni di gioco (all'incirca 20.000 unità) distribuite presso le sale gioco e le ricevitorie presenti in tutta Italia.
In particolare, dovrà essere sviluppata una componente software con funzionalità di controllo e filtro del traffico di rete.
Il progetto mi è stato commissionato dall’azienda Soluzione e Tecnologie s.r.l. (Gruppo Reti) di Busto Arsizio, presso la quale lavoro da oltre 2 anni ed ho svolto il periodo di tirocinio della durata complessiva di 3 mesi.

INQUADRAMENTO DELLA SITUAZIONE ATTUALE

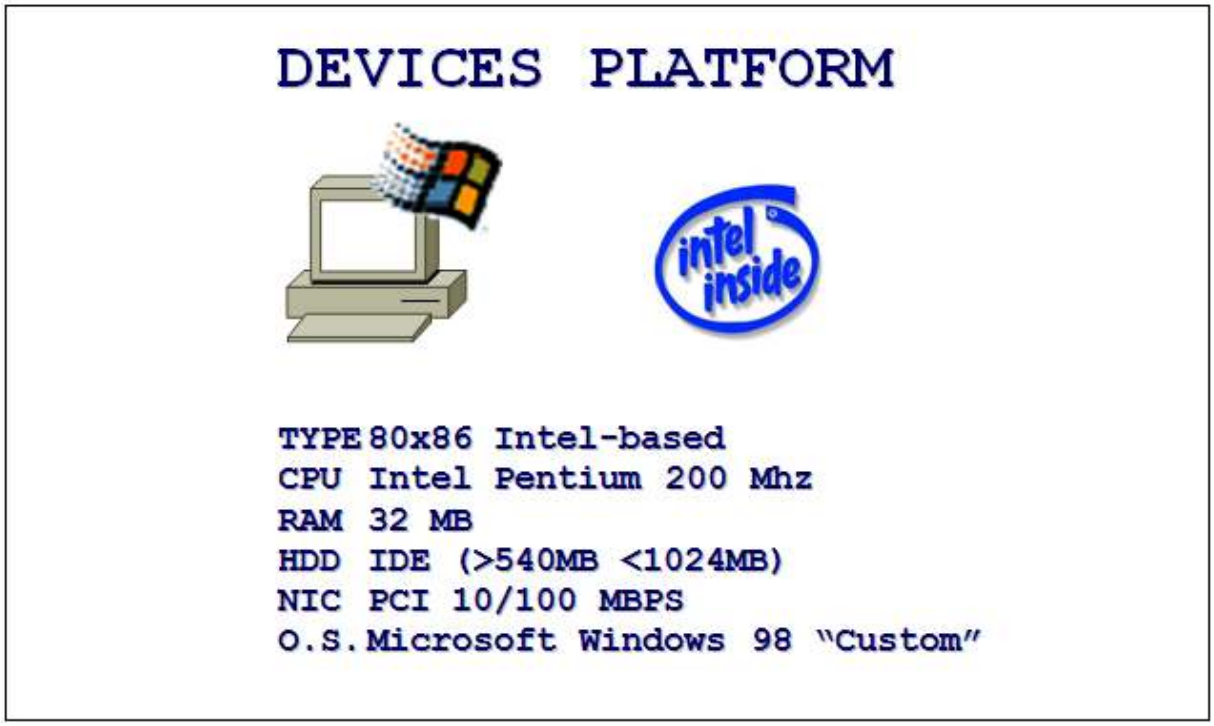
L'infrastruttura di rete privata del cliente sta attraversando in questo momento una fase transitoria nell'ambito della quale avverrà, in modo graduale e scaglionato, il passaggio del supporto di comunicazione della rete geografica (attualmente basato su protocolli X.25) verso un ambiente basato su protocolli di rete IP e su servizi di rete privata virtuale fornita dal provider di telecomunicazioni.

L'adozione di tale nuovo supporto di comunicazione trova le sue motivazioni tanto nell'ammodernamento tecnologico dell'infrastruttura di rete, quanto nell'ottimizzazione dei processi di gestione e mantenimento, oltre che ad una sensibile riduzione dei costi.



L'infrastruttura di rete privata, fornita da provider di telecomunicazioni esterni, è basata su canali VPN con tecnologia IP. Il centro-stella di tale infrastruttura è rappresentato dal Datacentre del cliente, nell'ambito del quale avviene la gestione dei servizi applicativi connessi alle funzionalità di gioco, nonché la gestione ed il mantenimento centralizzato dei dispositivi (all'incirca 20.000), distribuiti presso le periferie rappresentate da concessionarie esterne all'azienda e con le quali vengono mantenuti rapporti di tipo commerciale.

Le postazioni di gioco sono rappresentate da sistemi Intel-based identici tra loro tanto in termini hardware che software e sono equipaggiati con una versione "alleggerita" ad-hoc del sistema operativo Microsoft Windows 98.



NECESSITA' RISCONTRATE

Il passaggio da un'infrastruttura di rete basata su protocolli di rete X.25 all'attuale infrastruttura di rete basata su trasporto IP, se da una parte contribuisce a rendere più aggiornata, gestibile ed economica l'infrastruttura che garantisce l'interconnessione delle postazioni di gioco con il Datacentre del cliente, al contempo espone i sistemi ad una serie di possibili minacce informatiche, a fronte delle quali i sistemi operativi installati sui dispositivi end-point, discontinuati e privi dei "fix" rilasciati dal produttore (a causa della loro personalizzazione al di fuori dello standard) non sono in grado di opporre adeguate procedure di protezione.

Le problematiche di sicurezza riguardano sia tentativi di attacco informatico che propagazione di codice malizioso che possa essere portato nei confronti dei dispositivi end-point, anche a causa di una non ottimale segregazione del traffico nell'ambito dell'infrastruttura di rete geografica fornita dal Carrier di telecomunicazioni, ma soprattutto per possibili problemi di sicurezza che possano essere propagati dalle varie periferie presso cui i dispositivi end-point sono attestati e

sulle quali il cliente non e' in grado di detenere un controllo dal punto di vista giuridico.

Sulla base di tali minacce e di tali esposizioni dei sistemi, la necessità riscontrata riguarda la creazione di un framework software che, in grado di installarsi ed integrarsi con le peculiarità del sistema operativo installato, implementi specifiche funzionalità di protezione tali da controllare, limitare e regolamentare il traffico di rete a livello del singolo sistema.

A tali esigenze funzionali si accompagna l'esigenza tecnica di fare fronte alle limitate caratteristiche hardware dei dispositivi end-point e di minimizzare l'impatto di occupazione (memoria, disco) e di prestazioni sul sistema da parte delle componenti software che dovranno essere sviluppate.

SOLUZIONE PROPOSTA

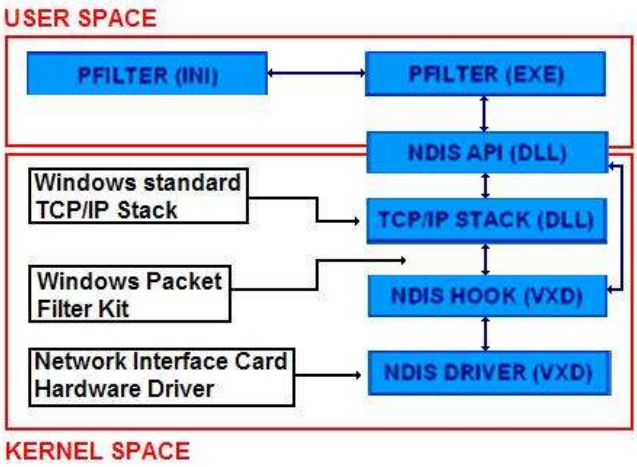
La soluzione proposta prevede la realizzazione di un pacchetto di protezione host-based (basata sull'installazione di una componente software all'interno dei dispositivi end-point) che, nell'ambito di una procedura di gestione e configurazione centralizzata, consenta di implementare funzionalità di sicurezza e controllo del traffico nell'ambito dei dispositivi end-point stessi.

La soluzione di protezione che verrà implementata riguarda funzionalità di Packet Filtering con particolare riferimento a funzionalità di controllo accessi a livello di indirizzamento di rete (ACLs – Access Control Lists) e di controllo delle porte logiche (Port Filtering).

L'efficacia della soluzione sarà basata sull'utilizzo di specifici "device driver" – in grado di integrarsi con il processo di comunicazione esistente, nell'ambito del sistema operativo, tra il "device driver" che assicura il funzionamento della scheda di rete ethernet e il "TCP/IP Stack", parte del sistema operativo stesso.

L’integrazione con tale processo di comunicazione permette di intercettare, modificare o bloccare in modo trasparente eventuale traffico di rete non aderente alle regolamentazioni che saranno definite nell’ambito della soluzione da implementarsi.

L’utilizzo di componenti software operanti a livello di Kernel, oltre che garantire un impatto minimale sulle prestazioni complessive del sistema, consente una totale ed efficace integrazione con il processo di funzionamento del sistema in termini di gestione del traffico IP, garantendo una protezione anche a fronte di potenziali vulnerabilità presenti nelle componenti del sistema operativo che assicurano il supporto di rete.



Nell’ambito della presente soluzione verranno integrate le componenti Runtime “Windows Packet Filter Kit” sviluppate da NTKernel (http://www.ntkernel.com), azienda leader nello sviluppo di Device Drivers per la piattaforma Microsoft Windows.

Tali Runtime, oltre a mettere a disposizione i Device Driver operanti a livello di Kernel che consentono di intercettare, modificare o bloccare il traffico prima che questo raggiunga le componenti di Stack TCP/IP del sistema, forniscono le adeguate interfacce di programmazione (API) che consentono la realizzazione delle componenti applicative di sicurezza senza la necessità di programmazione a livello di Kernel e fornendo i connettori per ambienti di sviluppo quali Visual Basic, Visual C++, C/C++.

Accanto alle componenti binarie, alle chiavi di registro, ai driver di sistema e alle librerie dinamiche da cui sarà composto il componente software che verrà sviluppato, saranno previsti appositi files destinati alla configurazione delle regole di sicurezza (“White Lists” e “Black Lists” associate agli indirizzi IP sorgenti, apertura e chiusura delle porte logiche a livello TCP ed UDP).

Il controllo degli indirizzi IP sorgenti e le limitazioni definite in termini di porte TCP/UDP permetteranno di limitare gli accessi alle funzionalità di gestione da parte delle sole sottoreti sulle quali siano attestati, presso il Datacentre del cliente, i sistemi adibiti alla gestione dei dispositivi end-point ed in particolare delle relative componenti di sicurezza.

COMPONENTE SOFTWARE WINPKFILTER

Windows Firewall Developer Kit (formalmente WINPKFILTER) rappresenta un framework con funzionalità di Packet Filtering ad alte prestazioni compatibile con i sistemi operativi Microsoft Windows 95/98/ME/NT/2K/XP che permette di filtrare in modo trasparente il traffico di rete in inbound ed outbound garantendo un impatto minimale sulle prestazioni del sistema e della rete e senza richiedere lo sviluppo di codice a basso livello.

Tale componente si contraddistingue per la flessibilità tanto in termini funzionali che da un punto di vista tecnico, permettendo di essere interfacciato da differenti ambienti e linguaggi di sviluppo tra cui Visual C++, Delphi, Visual Basic, C++ Builder.

WINPKFILTER e' sviluppato da NT Kernel Resources, un'azienda di sviluppo software e consulenza specializzata nello sviluppo a basso livello con particolare riferimento alla creazione di "device drivers" per i sistemi operativi Windows.

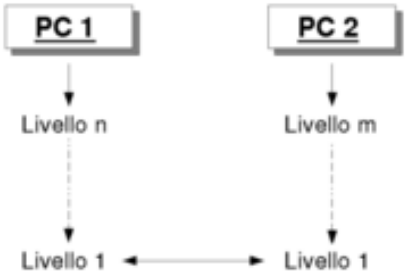
Le politiche di licenziamento di tali componenti prevedono l'acquisto di una "Runtime License" destinata ad applicazioni di sviluppo il cui costo e' di USD 3.495.

TEORIA E NOZIONI INTRODUTTIVE SULLE RETI

Nell'ambito delle telecomunicazioni, due o più macchine o host (computer, telefono, stampante, ecc...) possono comunicare tra loro rispettando norme che sono dette protocolli di rete. L'aderenza ai protocolli garantisce che due software in esecuzione su diverse macchine possano comunicare correttamente, anche se sono stati realizzati indipendentemente.

Ciascun protocollo regola normalmente solo una parte degli aspetti di una comunicazione. I diversi protocolli sono organizzati con un sistema detto "a livelli" : a ciascun livello viene usato uno specifico protocollo.

La divisione in livelli è fatta in modo che ciascun livello utilizzi i servizi offerti dal livello inferiore, e fornisca servizi più "ricchi" al livello superiore. I diversi livelli in un host comunicano tra loro tramite le interfacce. Ogni livello parla solo con quello immediatamente superiore e con quello immediatamente inferiore. I protocolli regolano invece la comunicazione tra due entità dello stesso livello, che serve a fornire servizi al livello superiore.



I vari livelli sono organizzati in pile di protocolli. Le pile di protocolli sono un modo flessibile per combinare componenti per realizzare un servizio.

Il livello più basso (1) è detto "livello fisico" e si occupa di gestire la trasmissione dei segnali attraverso il mezzo di trasporto (cavo, fibra ottica, infrarossi, ecc...). Il livello più elevato è chiamato "livello applicativo" ed è quello che permette all'utente di creare il messaggio da comunicare.

La divisione in livelli è piuttosto rigida a livello di specifica dei protocolli, mentre nell'implementazione spesso diversi livelli vengono implementati insieme in uno stesso modulo software.

In una rete a pacchetto ciascun livello della "pila protocollare" aggiunge ai pacchetti una intestazione, attraverso una operazione detta imbustamento.

LO STANDARD ISO/OSI

L'International Standard Organization (ISO) nel 1979 ha stabilito il protocollo Open Systems Interconnection (OSI), con l'intenzione di creare uno standard per le telecomunicazioni da usare nelle reti di tutto il mondo. All'atto pratico però, lo standard che viene comunemente usato nella maggior parte delle reti, è il TCP/IP, definito nella RFC 1155. Le differenze fondamentali dei due standard sono semplici: il primo è stato definito a tavolino da un'organizzazione super partes, mentre il secondo è opera di chi costruì materialmente le prime reti, sviluppandolo sul campo. Inoltre, lo standard ISO/OSI assegna un determinato compito ad ogni livello, mentre il TCP/IP è più "elastico" e permette di sviluppare protocolli che svolgono più di un compito-base.

Livello 1: fisico
Obiettivo: trasmettere un flusso di dati non strutturati attraverso un collegamento fisico, occupandosi della forma e del voltaggio del segnale. Ha a che fare con le procedure meccaniche e elettroniche necessarie a stabilire, mantenere e disattivare un collegamento fisico.

Livello 2: datalink
Obiettivo: permettere il trasferimento affidabile di dati attraverso il livello fisico. Invia trame di dati con la necessaria sincronizzazione ed effettua un controllo degli errori e delle perdite di segnale.
Questo livello si occupa di formare i dati da inviare attraverso il livello fisico, incapsulando i dati in un pacchetto provvisto di header (intestazione) e tail (coda), usati anche per sequenze di controllo.
Per ogni pacchetto ricevuto, il destinatario invia al mittente un pacchetto ACK (acknowledgement, conferma) contenente lo stato della trasmissione: il mittente deve ripetere l'invio dei pacchetti mal trasmessi e di quelli che non hanno ricevuto risposta.
Per ottimizzare l'invio degli ACK, si usa una tecnica detta Piggybacking, che consiste

nell'accodare ai messaggi in uscita gli ACK relativi ad una connessione in entrata, per ottimizzare l'uso del livello fisico. I pacchetti ACK possono anche essere raggruppati e mandati in blocchi.
Questo livello si occupa anche di controllare il flusso di dati: in caso di sbilanciamento di velocità di trasmissione, si occupa di rallentare l'opera della macchina più veloce, accordandola all'altra e minimizzando le perdite dovute a sovraccarico.
La sua unità dati fondamentale è la trama.

Livello 3: rete

Obiettivo: rende i livelli superiori indipendenti dai meccanismi e dalle tecnologie di trasmissione usate per la connessione. Si occupa di stabilire, mantenere e terminare una connessione.
È responsabile del routing (instradamento) dei pacchetti.
La sua unità dati fondamentale è il pacchetto.

Livello 4: trasporto

Obiettivo: permettere un trasferimento dati trasparente e affidabile (implementando anche un controllo degli errori e delle perdite) tra due host.
A differenza dei livelli precedenti, che si occupano di connessioni tra nodi contigui di una rete, il Trasporto (a livello logico) si occupa solo del punto di partenza e di quello di arrivo.
Si occupa anche di effettuare la frammentazione, di ottimizzare l'uso delle risorse di rete e di prevenire la congestione.
La sua unità dati fondamentale è il messaggio.

Livello 5: sessione

Obiettivo: controllare la comunicazione tra applicazioni. Stabilire, mantenere e terminare connessioni (sessioni) tra applicazioni cooperanti.
Esso consente di aggiungere, ai servizi forniti dal livello di trasporto, servizi più avanzati, quali la gestione del dialogo (mono o bidirezionale), la gestione del token (per effettuare mutua esclusione) o la sincronizzazione (inserendo dei checkpoint in modo da ridurre la quantità di dati da ritrasmettere in caso di gravi malfunzionamenti).

Si occupa anche di inserire dei punti di controllo nel flusso dati: in caso di errori nell'invio dei pacchetti, la comunicazione riprende dall'ultimo punto di controllo andato a buon fine.

Livello 6: presentazione
Obiettivo: trasformare i dati forniti dalle applicazioni in un formato standardizzato e offrire servizi di comunicazione comuni, come la crittografia, la compressione del testo e la riformattazione.
Esso consente di gestire la sintassi dell'informazione da trasferire. E sono previste tre diverse sintassi:

- astratta (definizione formale dei dati che gli applicativi si scambiano),
- concreta locale (come i dati sono rappresentati localmente)
- di trasferimento (come i dati sono codificati durante il trasferimento).

Livello 7: applicazione
Obiettivo: interfacciare utente e macchina.

LA SITUAZIONE PRECEDENTE NELLA RETE DEL CLIENTE

Come già detto in precedenza, l'infrastruttura di rete privata del cliente sta attraversando in questo momento una fase di migrazione da un ambiente basato su protocolli X.25 verso un ambiente basato su protocolli di rete IP.

Con X.25 si indica un protocollo di rete a commutazione di pacchetto. La commutazione di pacchetto è una tecnica di trasmissione nella quale è la rete stessa a ridirigere il singolo pacchetto di dati verso la corretta destinazione in base all'indirizzo contenuto nel pacchetto stesso. Una rete di tipo X.25 è composta da una serie di nodi interconnessi ai quali l'utente può collegarsi. Il lato utente della rete è denominato Data Terminal Equipment (DTE), gli apparati di rete interconnessi sono detti Data Circuit-terminating Equipment (DCE).
Il protocollo X.25 è stato sviluppato dall'International Telecommunications Union (ITU) e comprende i primi tre livelli dello standard OSI:
Livello 1 (fisico) comprende diversi standard come V.35, RS-232 e X.121

Livello 2 (collegamento) implementa lo standard ISO HDLC chiamato Link Access Procedure Balanced (LAPB) che fornisce un meccanismo di collegamento senza errori tra due nodi direttamente connessi. Gli errori sono rilevati e corretti ad ogni passaggio tra un nodo e l'altro. È questa caratteristica che fa dell'X.25 un protocollo così robusto ed adatto a linee disturbate. D'altro canto, poiché ogni frammento dev'essere ricevuto nella sua interezza e controllato prima di essere propagato al nodo successivo, si introduce una notevole latenza nella trasmissione, sono comuni latenze dell'ordine del mezzo secondo. Per questa ragione la dimensione tipica dei pacchetti dati utilizzati nelle connessioni X.25 è piuttosto piccola (128 o 256 byte). Protocolli moderni come Frame Relay o ATM, possono fare a meno di un forte controllo sugli errori perché sfruttano connessioni a bassa probabilità di errore.

Livello 3 (rete) comprende le specifiche necessarie per aprire o chiudere un collegamento tra due DTE, per il controllo del flusso e per combinare più collegamenti logici su una sola connessione fisica.

LE RETI TCP/IP – UN PO' DI STORIA

Nei primi anni settanta, la Defence Advanced Research Project Agency (DARPA) finanziò l'Università di Stanford e la BBN (Bolt, Beranek and Newman) per lo sviluppo di un insieme di protocolli di comunicazione da utilizzarsi per lo sviluppo di reti a commutazione di pacchetto, per l'interconnessione di calcolatori eterogenei. Fu così che nacque l'Internet Protocol Suite i cui due protocolli più noti sono il TCP (Trasmission Control Protocol) e l'IP (Internet Protocol).

Si fa riferimento a questa architettura di rete con la sigla TCP/IP. I creatori di tali protocolli di trasmissione, tuttora utilizzati nel web, sono nello specifico Robert Kahn e Vinton Cerf, a cui l'attuale Presidente degli Stati Uniti George W. Bush ha consegnato la Presidential Medal of Freedom, ovvero la più alta tra le onorificenze civili a stelle e strisce, il 9 novembre 2005. I due studiosi non sono nuovi a questo genere di premiazioni: all'inizio del 2005 è stato assegnato loro il prestigioso 2004 A.M. Turing Award, equivalente del Premio Nobel nel settore dell'Information Technology. Cerf e Kahn hanno sviluppato lo standard per la trasmissione di pacchetti via web nel lontano 1973, mentre lavoravano a un progetto di sviluppo dei sistemi di comunicazione voluto dalla DARPA (Defense Advanced Research Projects

Agency). Attualmente Vint Cerf, collabora con Google alla creazione degli standard per le future applicazioni e nel frattempo si dedica allo sviluppo di nuovi protocolli di comunicazione interplanetaria per il Jet Propulsion Lab della Nasa. Robert Kahn, invece, dopo 13 anni di servizio presso la DARPA è diventato presidente della Corporation for National Research Initiatives (CNRI).
Questi protocolli, utilizzabili gratuitamente da tutti perché di pubblico dominio fin dall'inizio, ottennero un elevato successo (utilizzati da un gruppo di ricercatori per ARPAnet).
TCP/IP è l'architettura adottata dalla rete internet. Negli anni novanta, nonostante la sua età, è stata (più o meno paradossalmente) l'unica architettura che ha interessato il mercato, al punto che gli enti di standardizzazione, di fronte al fatto compiuto della sua massiccia diffusione hanno dovuto darle la stessa dignità di ISO/OSI.

IL COMPONENTE SOFTWARE WINPKFILTER IN DETTAGLIO

WinpkFilter è un framework per la realizzazione di software per il packet filtering compatibile con Windows 9x/ME/NT/2000/XP/2003, che permette di filtrare (vedere e modificare) in maniera trasparente pacchetti Ethernet RAW, con impatto minimo sull'attività di rete e senza dover scrivere driver TDI o NDIS a basso livello.
L'installazione risulta particolarmente semplice; in particolare con S.O. Windows 98 è sufficiente copiare il file **ndisrd.vxd** nella cartella **\WINDOWS\SYSTEM** e creare la seguente chiave di registro necessaria per far partire il driver all'avvio del sistema:
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\NDISRD
All'interno di questa chiave deve essere creato il seguente valore **REG_SZ**:
"StaticVxD"="NDISRD.VxD"

PREPARAZIONE AL FILTERING – OGGETTI ISTANZIATI NELLA FASE INIZIALE – VC++

```
CNdisApi                api;
```

L'oggetto **api** di tipo **CNdisApi** serve per l'apertura e la gestione del driver Winpkfilter. In particolare, la funzione **api.IsDriverLoaded()** ritorna TRUE se il driver viene caricato con successo in memoria, in caso contrario ritorna FALSE.

```
TCP_AdapterList         AdList;
```

L'oggetto **AdList** di tipo **TCP_Adapterlist** è un oggetto che istanzia una struttura dati così composta:

```
struct TCP_AdapterList
{
  unsigned long     m_nAdapterCount;
  unsigned char     m_szAdapterNameList[ ADAPTER_LIST_SIZE ][
ADAPTER_NAME_SIZE ];
  HANDLE            m_nAdapterHandle [ ADAPTER_LIST_SIZE ];
  unsigned int      m_nAdapterMediumList[ ADAPTER_LIST_SIZE ];
  unsigned char     m_czCurrentAddress[ ADAPTER_LIST_SIZE ][
ETHER_ADDR_LENGTH ];
  unsigned short    m_usMTU [ ADAPTER_LIST_SIZE ];
}
```

La struct **TCP_AdapterList** viene utilizzata per contenere le informazioni relative agli Adapter di rete presenti sul pc sul quale viene eseguito il packet filtering. I campi principali hanno lo scopo di memorizzare le informazioni seguenti:

m_nAdapterCount: numero di interfacce MSTCP presenti.

m_szAdapterNameList: array che contiene i nomi degli adapters.

m_czCurrentAddress: array che contiene gli indirizzi Ethernet degli adapter attualmente configurati.

m_usMTU: array che contiene le informazioni dell'MTU di ogni interfaccia di rete.

Le informazioni sopra menzionate, vengono caricate richiamando la funzione **api.GetTcpipBoundAdaptersInfo(&AdList)**.

Successivamente, il controllo sul campo **m_nAdapterCount** ci dirà se sono presenti interfacce di rete da poter utilizzare:

```
if(AdList.m_nAdapterCount == 0)
```

```
ADAPTER_MODE        Mode;
```

La struct **ADAPTER_MODE** contiene le informazioni relative all'handle di ogni interfaccia di rete e un campo **dwFlags** che è una combinazione dei flags relativi alla modalità di funzionamento della interfaccia di rete:

MSTCP_FLAG_SENT_TUNNEL: i pacchetti provenienti dallo stack TCP/IP ed indirizzati alla scheda di rete vengono letti ed automaticamente inoltrati all'interfaccia di rete stessa
MSTCP_FLAG_RECV_TUNNEL: i pacchetti provenienti dalla scheda di rete ed indirizzati allo stack TCP/IP vengono letti ed automaticamente inoltrati allo stack TCP/IP stesso
MSTCP_FLAG_SENT_LISTEN: i pacchetti provenienti dallo stack TCP/IP ed indirizzati alla scheda di rete vengono letti ma non vengono inoltrati all'interfaccia di rete
MSTCP_FLAG_RECV_LISTEN: i pacchetti provenienti dalla scheda di rete ed indirizzati allo stack TCP/IP vengono letti ma non vengono automaticamente inoltrati allo stack TCP/IP
MSTCP_FLAG_FILTER_DIRECT: in modalità promiscua, lo stack TCP/IP riceve anche pacchetti non destinati all'interfaccia di rete corrente. Settando questo flag, solamente i pacchetti con MAC address uguale a quello corrente o uguali a FF:FF:FF:FF:FF:FF raggiungeranno il TCP/IP stack
MSTCP_FLAG_LOOPBACK_FILTER: permette il passaggio di pacchetti di loopback

MSTCP_FLAG_LOOPBACK_BLOCK: blocca i pacchetti di loopback. Questo flag spesso viene utilizzato in combinazione con la modalità promiscua, per prevenire che alcuni pacchetti vengano processati più di una volta.

FILE DI CONFIGURAZIONE

Per permettere l'esecuzione del programma, è necessaria la presenza di un file di configurazione che contenga le seguenti informazioni:

- ID dell'interfaccia di rete sulla quale attivare il filtro;
- Nome del file di log e dell'archivio dei log sui quali salvare l'elenco dei pacchetti bloccati;
- Indicazione della white-list in termini di indirizzi IP validi e porte TCP e UDP valide.

Il nome di default del file di configurazione è **config.cfg**. Se non specificato diversamente, il file di configurazione utilizzato è quello di default. E' altresì possibile utilizzare files di configurazione diversi, utilizzando la seguente sintassi:

```
packet_filtering.exe -c nomefile.cfg
```

Il file di configurazione deve essere un file di testo ANSI contenente le sezioni indicate in seguito. In particolare i campi songoli saranno nella forma:

```
nomecampo = "valore"
```

mentre i campi multipli saranno nella forma:

```
nomecampo[0] = "valore"
nomecampo[1] = "valore"
....
nomecampo[n] = "valore"
```

Se il primo carattere di una riga del file di log è # la riga viene considerata come commento e quindi non influenza il comportamento del programma. Il commento

finisce esclusivamente quando finisce la riga (fino alla pressione del tasto Enter il quale porta il cursore alla riga successiva)

Esempio:

 # commento commento @#[]qwerty!"£$%&/()=

Sezioni del file di configurazione:

LOG_NAME = "nomelog1.log"

Indica il nome del file .log all'interno del quale vengono inseriti i record relativi ai pacchetti bloccati (grandezza massima: circa 1Mb).
Esempio: LOG_NAME = "PackFilter.log"

LOGARCHIVE_NAME = "nomelog2.log"

Indica il nome del file .log all'interno del quale viene mantenuto l'archivio dei record relativi ai pacchetti bloccati (grandezza massima: circa 1Mb).
Esempio: LOGARCHIVE_NAME = "PackFilterArch.log"

ID = "0"

Indica l'ID dell'interfaccia di rete sulla quale applicare il filtro. Se è presente una sola scheda di rete deve essere utilizzato il valore di default (0). E' possibile utilizzare il tool **NetList** (che verrà spiegato in seguito) per risalire all'ID dell'interfaccia

IP[0] = "xxx.xxx.xxx.xxx"
IP[1] = "xxx.xxx.xxx.xxx"
....
IP[n] = "xxx.xxx.xxx.xxx"

Indicano l'elenco degli indirizzi IP sorgente consentiti. Saranno accettati i pacchetti IP ricevuti con indirizzo sorgente indicato.
esempio:
IP[0] = "10.1.2.61"
IP[1] = "10.1.2.68"

Vengono accettati anche valori nel formato "xxx.xxx.xxx.*" e "xxx.xxx.*.*" che indicano che tutta la classe di IP è consentita, accetta * solo nell'ultimo e nel penultimo byte.
esempio:
IP[2] = "10.1.2.*"
IP[3] = "10.2.*.*"
Indica che sono consentiti gli IP sorgenti nel range da 10.1.2.0 a 10.1.2.255 e nel range 10.2.1.1 a 10.2.255.255; quindi i pacchetti IP con indirizzo sorgente indicato dal range non saranno bloccati.

IPR[0] = "xxx.xxx.xxx.xxx-xxx.xxx.xxx.xxx"
IPR[1] = "xxx.xxx.xxx.xxx-xxx.xxx.xxx.xxx"
....
IPR[n] = "xxx.xxx.xxx.xxx-xxx.xxx.xxx.xxx"

Indica che sono accettati anche ranges nel formato "xxx.xxx.xxx.xxx-xxx.xxx.xxx.xxx". In questo modo non sarà presa in considerazione l'intera classe, ma solamente una parte.
esempio: IPR[0] = "10.1.2.1-10.1.2.10"
Consente la ricezione di pacchetti IP con indirizzo sorgente compreso tra 10.1.2.1 e 10.1.2.10, accetta l'indicazione del range solo nell'ultimo byte

PP[0] = "porta/protocollo"
PP[1] = "porta/protocollo"
....
PP[N] = "porta/protocollo"

Indica la regola di livello "trasporto" che definisce quali protocolli di livello 4 devono essere accettati e su quali porte.
esempio: PP[0] = "443/UDP"
Indica che per gli indirizzi elencati in precedenza devono essere accettati pacchetti UDP - porta 443.
Il valore del protocollo può essere TCP o UDP mentre il valore della porta deve essere compreso tra 0 e 65535 (così come definito dalla IANA – Internet Assigned Numbers Authority - www.iana.org)

Nota: per permettere il passaggio di traffico DHCP, è necessario inserire in white-list l'indirizzo del server DHCP dal quale accettare il traffico nonchè le porte logiche UDP utilizzate dal protocollo DHCP (UDP porte 67 e 68).

CARICARE ACL IN MEMORIA

Al fine di poter verificare l'aderenza alle regole definite nel file di configurazione e di renderne più veloce il controllo, è necessario caricare il contenuto del file di configurazione in memoria.
I dati relativi agli indirizzi IP ed alle porte consentite vengono memorizzati nelle seguenti strutture dati:

Lista di supporto elenco ip

```
struct IP_list
{
        struct IP_list *next;
        char address[16];
};
```

```
typedef struct IP_list IP_list;
```

Lista di supporto elenco porte

```
struct port_list
{
        struct port_list *next;
        int port;
        char proto[3];
};
```

```
typedef struct port_list port_list;
```

Lista di supporto elenco Ip-ranges

```
struct IP_range
```

```
{
        struct IP_range *next;
        char address1[16];
        char address2[16];
};
```

```
typedef struct IP_range IP_range;
```

Puntatore al primo elemento della lista degli IP

```
IP_list *IP_head;
```

Puntatore al primo elemento della lista delle porte

```
port_list *port_head;
```

Puntatore al primo elemento della lista dei ranges

```
IP_range *range_head;
```

Mediante tali liste, viene mantenuto il contenuto delle ACL in memoria.
Sono state codificate le seguenti funzioni di supporto:

```
void load_cfg()
```

Tale funzione carica in memoria il contenuto del file di configurazione (id dell'interfaccia di rete, indirizzi ip, porte tcp/udp, ranges di indirizzi ip). L'algoritmo utilizzato è il medesimo per le varie tipologie da leggere dal file di configurazione: viene eseguito un ciclo while e per ogni elemento nel file di configurazione che presenta l'etichetta da ricercare (ID, IP, PP, IPR) viene allocato un nuovo record delle struct sopraccitate, assegnando in maniera corretta i puntatori dell'elemento precedente a quello corrente e quello successivo a quello corrente a NULL.

```
void libera_port(port_list *p)
```

```
void libera_IP(IP_list *p)
```

```
void libera_range(IP_range *p)
```

Le precedenti funzioni deallocano le liste presenti in memoria prima della terminazione del programma. Le chiamate alle precedenti funzioni vengono eseguite a loro volta ricorsivamente dalla funzione **void free_mem()**.

ALTRE STRUTTURE DATI UTILIZZATE

Come detto in precedenza, l'ID dell'interfaccia di rete viene caricato dal file di configurazione. Tale valore viene conservato nella variabile **myAdapter**.

Dopo aver istanziato un handle sull'adapter prescelto, viene definita la modalità di filtro hardware direttamente sulla scheda di rete:

```
HANDLE hAdapter = AdList.m_nAdapterHandle[myAdapter];
```

```
api.SetHwPacketFilter ( hAdapter, NDIS_PACKET_TYPE_PROMISCUOUS )
```

In questo modo, verrà attivata la modalità di ricezione promiscua mediante la chiamata del metodo **SetHwPacketFilter** con flag **NDIS_PACKET_TYPE_PROMISCUOUS**.

Come già detto in precedenza, la struct **Mode** contiene anche le informazioni relative alle modalità di funzionamento del filtro software applicato sull'interfaccia di rete.

```
Mode.dwFlags = MSTCP_FLAG_RECV_TUNNEL;
```

Con la precedente istruzione, viene definita una regola che indica che il filtro software deve intercettare solo i pacchetti ricevuti dall'interfaccia di rete (quelli in ingresso, lasciando transitare indistintamente quelli in outgoing). In questo modo, i pacchetti in ingresso verranno "droppati" dall'interfaccia di rete e quindi, una volta analizzati, potranno essere scartati oppure inoltrati allo stack tcp/ip (solo se aderenti alle specifiche definite).

Saranno utilizzate anche le seguenti strutture dati e variabili, le quali verranno maggiormente dettagliate in seguito:

```
ETH_REQUEST     Request;
```

```
INTERMEDIATE_BUFFER PacketBuffer;
```

L'oggetto **Request** (di tipo **ETH_REQUEST**) è così composto:

```
struct ETH_REQUEST
{
        HANDLE                  hAdapterHandle;
        NDISRD_ETH_Packet       EthPacket;
}
```

hAdapterHandle: valore dell'handle della scheda di rete. Questa struttura viene utilizzata per inviare e ricevere pacchetti specifica l'interfaccia di rete utilizzata per tali operazioni.

EthPacket: questo campo mantiene un puntatore alla struttura **INTERMEDIATE_BUFFER**.

Essa è infatti così composta:

```
struct NDISRD_ETH_Packet
{
        PINTERMEDIATE_BUFFER            Buffer;
}
```

In questo caso, **Buffer** mantiene appunto il puntatore alla struttura **INTERMEDIATE_BUFFER**.

La struttura **INTERMEDIATE_BUFFER**, che conterrà il pacchetto ricevuto dall'interfaccia di rete, è così composta:

```
struct INTERMEDIATE_BUFFER
{
        LIST_ENTRY              m_qLink;
        ULONG                   m_dwDeviceFlags;
        ULONG                   m_Length;
        ULONG                   m_Flags; // NDIS_PACKET flags
        UCHAR                   m_IBuffer [MAX_ETHER_FRAME];
}
```

m_qLink: Campo usato internamente dal driver.
m_dwDeviceFlags: flag **PACKET_FLAG_ON_SEND** (se il pacchetto viene intercettato dallo stack TCP/IP) o **PACKET_FLAG_ON_RECEIVE** (se il pacchetto viene intercettato dall'interfaccia di rete)
m_Length: lunghezza del pacchetto contenuto nel campo **m_IBuffer**.
m_Flags: questo campo è una combinazione dei flag **NDIS_FLAGS_XXX** (definiti nel file ndis.h - Network Device Interface Specification definitions – parte del pacchetto delle w32api)
m_IBuffer: questo campo contiene il contenuto vero e proprio del pacchetto RAW Ethernet.

Pertanto, le variabili **Request** (di tipo **ETH_REQUEST**) e **PacketBuffer** (di tipo **INTERMEDIATE_BUFFER**) vengono utilizzate per la scansione dei singoli pacchetti.

Innanzitutto, tali variabili vengono inizializzate mediante l'api **ZeroMemory**, che "riempie" di 0 le variabili stesse:

```
ZeroMemory (&Request, sizeof(ETH_REQUEST) );
ZeroMemory (&PacketBuffer, sizeof(INTERMEDIATE_BUFFER) );
```

Ulteriori strutture dati utilizzate successivamente sono le seguenti:

```
ether_header* pEthHeader;       - puntatore header del pacchetto ethernet;
```

```
iphdr_ptr pIpHeader;            - puntatore header del pacchetto IP;
```

tcphdr_ptr pTcpHeader; *- puntatore header del pacchetto TCP;*

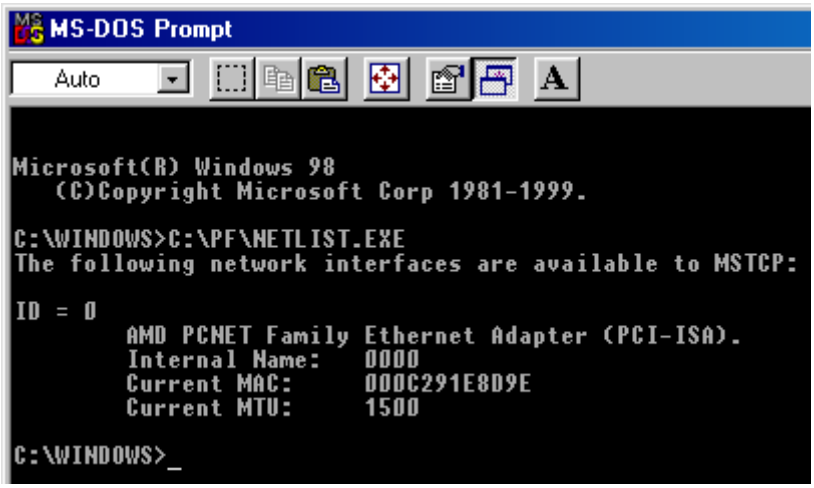
udphdr_ptr pUdpHeader; *- puntatore header del pacchetto UDP;*

TROVARE NETWORK ADAPTER (Programma NetList.exe di supporto)

Prima di poter eseguire il programma di packet filtering, è necessario inserire nel file di configurazione l'ID dell'interfaccia di rete sulla quale si vuole attivare il filtro (oltre alle ACLs).

A tale scopo, è stato sviluppato un tool di supporto per trovare tale identificativo.

Il tool **NetList.exe**, compatibili esclusivamente con Sistemi Operativi Windows 98 (in possesso del cliente) lista le schede di rete presenti sulla macchina.



In particolare, dopo avere istanziato gli oggetti necessari al funzionamento del driver, anche in questo caso viene utilizzato il campo **m_nAdapterCount** e viene eseguito un ciclo per ogni adapter trovato:

for (UINT i = 0; i < AdList.m_nAdapterCount; ++i)

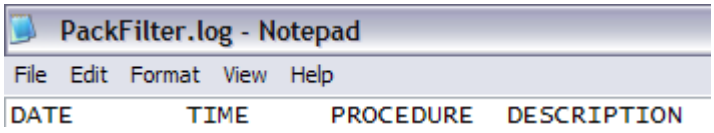
Dopodichè viene utilizzata la funzione **ConvertWindows9xAdapterName**, la quale converte il nome interno della scheda di rete ritornato dalla funzione **GetTcpipBoundAdaptersInfo** nel nome che è possibile vedere nella schermata di Windows in Network Connections properties.

In questo modo è possibile rintracciare l'interfaccia di rete desiderata.

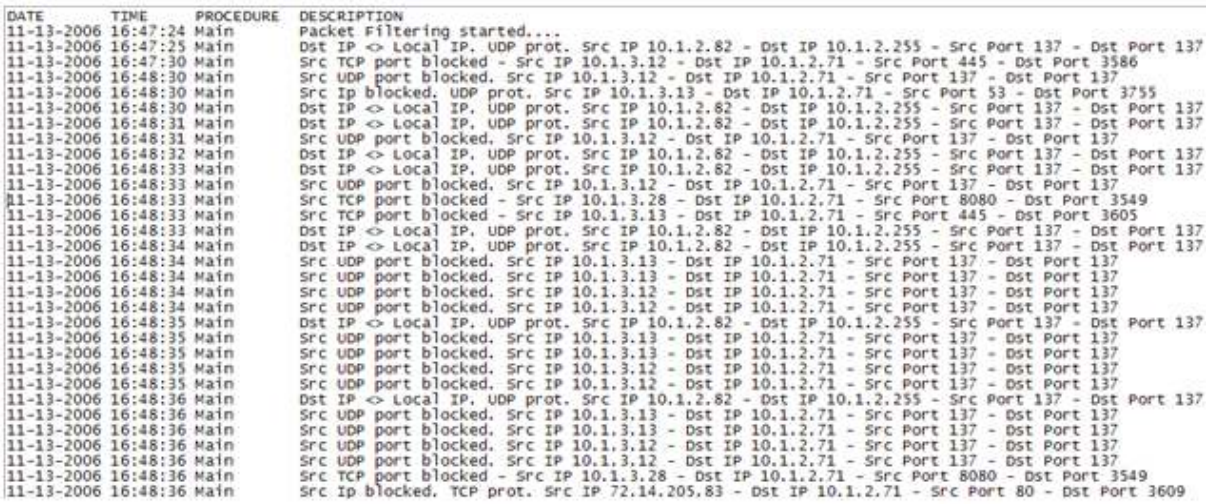
```
ConvertWindows9xAdapterName((const char*)AdList.m_szAdapterNameList[i],
szFriendlyName, MAX_PATH*4);
```

FILES DI LOG

Come indicato in precedenza, saranno presenti 2 files di log in formato testo. Il file di log vero e proprio, con l'indicazione degli ultimi pacchetti filtrati sarà della grandezza massima di 1Mb e conterrà un record per ogni pacchetto bloccato.
Il formato è il seguente:



Il secondo file, conterrà l'archivio del file di log e verrà riempito al raggiungimento della grandezza di 1Mb da parte del file di log stesso.
Quindi, il file di log sarà nuovamente vuoto, ma una copia sarà archiviata e disponibile per essere eventualmente spostata altrove per effettuare delle analisi sui pacchetti bloccati.
Un esempio di contenuto del file di log è il seguente:



```
DATE       TIME     PROCEDURE  DESCRIPTION
11-13-2006 16:47:24 Main       Packet Filtering started....
11-13-2006 16:47:25 Main       Dst IP <> Local IP, UDP prot. Src IP 10.1.2.82 - Dst IP 10.1.2.255 - Src Port 137 - Dst Port 137
11-13-2006 16:47:30 Main       Src TCP port blocked - Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 445 - Dst Port 3586
11-13-2006 16:48:30 Main       Src UDP port blocked. Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:30 Main       Src Ip blocked. UDP prot. Src IP 10.1.3.13 - Dst IP 10.1.2.71 - Src Port 53 - Dst Port 3755
11-13-2006 16:48:30 Main       Dst IP <> Local IP, UDP prot. Src IP 10.1.2.82 - Dst IP 10.1.2.255 - Src Port 137 - Dst Port 137
11-13-2006 16:48:31 Main       Dst IP <> Local IP, UDP prot. Src IP 10.1.2.82 - Dst IP 10.1.2.255 - Src Port 137 - Dst Port 137
11-13-2006 16:48:31 Main       Src UDP port blocked. Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:32 Main       Dst IP <> Local IP, UDP prot. Src IP 10.1.2.82 - Dst IP 10.1.2.255 - Src Port 137 - Dst Port 137
11-13-2006 16:48:33 Main       Dst IP <> Local IP, UDP prot. Src IP 10.1.2.82 - Dst IP 10.1.2.255 - Src Port 137 - Dst Port 137
11-13-2006 16:48:33 Main       Src UDP port blocked. Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:33 Main       Src TCP port blocked - Src IP 10.1.3.28 - Dst IP 10.1.2.71 - Src Port 8080 - Dst Port 3549
11-13-2006 16:48:33 Main       Src TCP port blocked - Src IP 10.1.3.13 - Dst IP 10.1.2.71 - Src Port 445 - Dst Port 3605
11-13-2006 16:48:33 Main       Dst IP <> Local IP, UDP prot. Src IP 10.1.2.82 - Dst IP 10.1.2.255 - Src Port 137 - Dst Port 137
11-13-2006 16:48:34 Main       Dst IP <> Local IP, UDP prot. Src IP 10.1.2.82 - Dst IP 10.1.2.255 - Src Port 137 - Dst Port 137
11-13-2006 16:48:34 Main       Src UDP port blocked. Src IP 10.1.3.13 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:34 Main       Src UDP port blocked. Src IP 10.1.3.13 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:34 Main       Src UDP port blocked. Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:34 Main       Src UDP port blocked. Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:35 Main       Dst IP <> Local IP, UDP prot. Src IP 10.1.2.82 - Dst IP 10.1.2.255 - Src Port 137 - Dst Port 137
11-13-2006 16:48:35 Main       Src UDP port blocked. Src IP 10.1.3.13 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:35 Main       Src UDP port blocked. Src IP 10.1.3.13 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:35 Main       Src UDP port blocked. Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:35 Main       Src UDP port blocked. Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:36 Main       Dst IP <> Local IP, UDP prot. Src IP 10.1.2.82 - Dst IP 10.1.2.255 - Src Port 137 - Dst Port 137
11-13-2006 16:48:36 Main       Src UDP port blocked. Src IP 10.1.3.13 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:36 Main       Src UDP port blocked. Src IP 10.1.3.13 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:36 Main       Src UDP port blocked. Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:36 Main       Src UDP port blocked. Src IP 10.1.3.12 - Dst IP 10.1.2.71 - Src Port 137 - Dst Port 137
11-13-2006 16:48:36 Main       Src TCP port blocked - Src IP 10.1.3.28 - Dst IP 10.1.2.71 - Src Port 8080 - Dst Port 3549
11-13-2006 16:48:36 Main       Src Ip blocked. TCP prot. Src IP 72.14.205.83 - Dst IP 10.1.2.71 - Src Port 80 - Dst Port 3609
```

LETTURA PACCHETTO ETHERNET

Una volta istanziati gli oggetti necessari e lette le ACL dal file di configurazione, si entra nel ciclo principale del programma:

```
while(TRUE)
{
        ESECUZIONE CONTROLLI
}
```

La prima istruzione eseguita all'interno del ciclo permettere di leggere il pacchetto mediante la funzione **ReadPacket**. Se non sono presenti pacchetti viene eseguita una **Sleep** della durata di 100 millisecondi e verrà nuovamente riletto il pacchetto successivo.

```
if(api.ReadPacket(&Request))
{
        pEthHeader = (ether_header*)PacketBuffer.m_lBuffer;
        ……
        ……
        …
}
else
{
        Sleep(100);
}
```

Se il pacchetto è disponibile e viene letto, il contenuto vero e proprio (presente nel campo **m_lBuffer**) viene fatto puntare al puntatore **pEthHeader** che è un pointer ad una struttura **ether_header** così composta:

```
typedef struct ether_header
{
        unsigned char       h_dest[ETH_ALEN];           /* destination eth addr */
        unsigned char       h_source[ETH_ALEN];     /* source ether addr*/
```

```
    unsigned short      h_proto;            /* packet type ID field  */
} ether_header, *ether_header_ptr;
```

La struct **ether_header** rappresenta l'header del pacchetto Ethernet ed in particolare i campi **h_dest** e **h_source** sono il mac address sorgente e destinatario mentre il campo **h_proto** rappresenta il protocollo di livello successivo.

I principali protocolli di livello successivo a quello Ethernet sono così codificati (per la lista completa vedere l'allegato 1):

DEFINIZIONE	**HEX**	**DEC**	**SPIEGAZIONE**
#define ETH_P_IP	0x0800	2048	Protocollo IPv4
#define ETH_P_RARP	0x8035	32821	Protocollo RARP
#define ETH_P_ARP	0x0806	2054	Protocollo ARP
#define ETH_P_IPV6	0x86DD	34525	Protocollo IPv6
#define ETH_P_LMIN	0x0000	0	Lunghezza minima pacchetto Ethernet
#define ETH_P_LMAX	0x05DC	1500	Se il valore del protocollo successivo è compreso tra 0x0000 e 0x05DC tale campo non assume più il significato di protocollo successivo ma assume il valore della lunghezza del pacchetto stesso che sarà codificato secondo IEEE 802.3

IL PROTOCOLLO ETHERNET

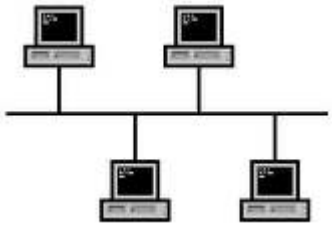
Prima di capire come poter filtrare correttamente i pacchetti Ethernet ed i protocolli successivi, è necessario comprenderne il funzionamento.

Ethernet è il nome di un protocollo per reti locali, sviluppato a livello sperimentale da Robert Metcalfe e David Boggs, suo assistente, alla Xerox PARC. La data ufficiale è il 1973 quando Metcalfe scrisse un promemoria ai suoi capi della Xerox sulle potenzialità di Ethernet. Nel 1976 Metcalfe e Boggs pubblicano un articolo dal titolo Ethernet: Distributed Packet-Switching For Local Computer Networks.

L'obiettivo originale dell'esperimento era ottenere una trasmissione affidabile a 3Mbps su cavo coassiale in condizioni di traffico contenuto, ma in grado di tollerare bene occasionali picchi di carico. Per regolamentare l'accesso al mezzo trasmissivo era stato adottato un protocollo di tipo CSMA/CD (Carrier Sense Multiple Access / Collision Detection).

Il successo dell'esperimento suscitò forte interesse e portò alla formazione di un gruppo di imprese, costituito da Xerox Corporation, Intel Corporation e Digital Equipment Corporation, che nel 1978 portarono alla standardizzazione 802.3 e il 30 settembre 1980 a pubblicare la versione 1.0 dello standard Ethernet.

Intanto Metcalfe lasciò Xerox nel 1979 per promuovere l'uso del PC e delle LAN per cui fondò 3Com. Metcalfe spesso attribuisce il successo di 3Com a Jerry Saltzer. Questi collaborò alla stesura di un articolo importantissimo dove suggeriva che l'architettura token ring fosse teoricamente superiore alla Ethernet. Con questo le grosse aziende decisero di non puntare su Ethernet mentre, al contrario, 3Com poté creare un business intorno al sistema riuscendo a guadagnarsi un ottimo vantaggio tecnico e a dominare sul mercato quando Ethernet prese piede.



Schema di una rete Ethernet

Successivamente, l'interesse delle imprese del settore aumentò al punto che l'IEEE costituì alcuni gruppi di studio finalizzati a perfezionare e consolidare Ethernet, nonché a creare numerosi altri standard correlati. Uno dei risultati raggiunti fu la pubblicazione, nel 1985, della prima versione dello standard IEEE 802.3, basato sull'originale specifica Ethernet.

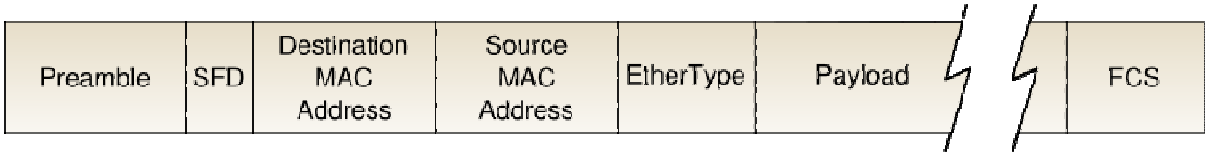
Connettori RJ 45 per reti Ethernet

Ethernet attualmente è il sistema LAN più diffuso per diverse ragioni:

- È nata molto presto e si è diffusa velocemente, per cui l'uscita di nuove tecnologie come FDDI e ATM hanno trovato il campo occupato;
- Rispetto ai sistemi concorrenti è più economica e facile da usare e la diffusione delle componenti hardware ne facilitano l'adozione;
- Funziona bene e genera pochi problemi (cosa rara nel campo informatico);
- È adeguata all'utilizzo con TCP/IP;
- Nonostante i suoi concorrenti fossero più veloci nella trasmissione dati, la Ethernet si è sempre ben difesa.

Il Pacchetto (Frame) Ethernet

Nonostante Ethernet abbia diverse tipologie, l'elemento comune è nella struttura del frame che viene definito DIX (DEC, Intel, Xerox) ed è rimasto fedele alla versione originale:



Questo è il frame ricevuto dallo strato di rete nella pila di protocolli. Gli elementi sono:

- **Preamble** - Preambolo (8 byte): I primi 7 byte hanno valore 10101010 e servono a svegliare gli adattatori del ricevente e a sincronizzare gli oscillatori con quelli del mittente. L'ultimo byte ha valore 10101011 e la serie dei due bit a 1 indica al destinatario che sta arrivando del contenuto importante. I bit del Preambolo vengono filtrati dalla scheda di rete e non arrivano a livelli successivi, quindi effettivamente la prima sequenza di bytes ricevuta a livello software sarà il MAC address di destinazione;
- **Destination MAC address** - Indirizzo di destinazione (6 byte): Questo campo contiene l'indirizzo LAN dell'adattatore di destinazione, se l'indirizzo non corrisponde il Livello fisico del protocollo lo scarta e non lo invia agli strati successivi.
- **Source MAC address** - Indirizzo sorgente (6 byte);
- **EtherType** - Campo tipo (2 byte): Questo campo indica il tipo di protocollo in uso durante la trasmissione e la lunghezza del campo dati;
- **Payload** - Campo dati (da 46 a 1500 byte): contiene i dati reali e possono essere di lunghezza variabile in base al MTU (Maximum Transmission Unit) della Ethernet. Se i dati superano la capacità massima, vengono suddivisi in più pacchetti;
- **FCS** - Controllo a ridondanza ciclica (CRC) (4 byte): permette di rilevare se sono presenti errori di trasmissione, in pratica il ricevente calcola il CRC mediante un algoritmo e lo confronta con quello ricevuto in questo campo.

Indirizzo Ethernet

Gli indirizzi sono tutti a 6 byte in quanto Ethernet definisce uno schema di indirizzamento a 48 bit: ogni nodo collegato, quindi, ha un indirizzo Ethernet univoco di questa lunghezza. Esso corrisponde all'indirizzo fisico della macchina ed è associato all'hardware.
Sono anche detti indirizzi hardware, indirizzi MAC (o MAC address) o indirizzi di livello 2.

ESEMPIO DI PACCHETTO ETHERNET

```
0000   01 00 5e 7f ff fa 00 02   d1 01 e7 93 08 00 45 00
0010   01 12 e9 9b 00 00 40 11   94 00 0a 01 02 44 ef ff
0020   ff fa 07 6c 07 6c 00 fe   c7 04 4e 4f 54 49 46 59
0030   20 2a 20 48 54 54 50 2f   31 2e 31 0d 0a 48 4f 53
0040   54 3a 20 32 33 39 2e 32   35 35 2e 32 35 35 2e 32
0050   35 30 3a 31 39 30 30 0d   0a 43 41 43 48 45 2d 43
0060   4f 4e 54 52 4f 4c 3a 20   6d 61 78 2d 61 67 65 3d
0070   31 38 30 30 0d 0a 4c 4f   43 41 54 49 4f 4e 3a 20
0080   68 74 74 70 3a 2f 2f 31   30 2e 31 2e 32 2e 36 38
0090   3a 34 38 33 36 30 2f 64   65 73 63 72 69 70 74 69
00a0   6f 6e 2e 78 6d 6c 0d 0a   4e 54 3a 20 75 70 6e 70
00b0   3a 72 6f 6f 74 64 65 76   69 63 65 0d 0a 4e 54 53
00c0   3a 20 73 73 64 70 3a 61   6c 69 76 65 0d 0a 53 45
00d0   52 56 45 52 3a 20 45 6d   62 65 64 64 65 64 20 55
00e0   50 6e 50 2f 31 2e 30 0d   0a 55 53 4e 3a 20 75 75
00f0   69 64 3a 75 70 6e 70 5f   56 53 33 31 30 30 2d 30
0100   30 30 32 44 31 30 31 45   37 39 33 3a 3a 75 70 6e
0110   70 3a 72 6f 6f 74 64 65   76 69 63 65 0d 0a 0d 0a
```

Non considerando il preambolo che viene omesso perché catturato a livello hw dalla scheda di rete, possiamo notare gli indirizzi MAC nei primi bytes:

```
0000   01 00 5e 7f ff fa 00 02   d1 01 e7 93 08 00 45 00
0010   01 12 e9 9b 00 00 40 11   94 00 0a 01 02 44 ef ff
0020   ff fa 07 6c 07 6c 00 fe   c7 04 4e 4f 54 49 46 59
0030   20 2a 20 48 54 54 50 2f   31 2e 31 0d 0a 48 4f 53
0040   54 3a 20 32 33 39 2e 32   35 35 2e 32 35 35 2e 32
0050   35 30 3a 31 39 30 30 0d   0a 43 41 43 48 45 2d 43
0060   4f 4e 54 52 4f 4c 3a 20   6d 61 78 2d 61 67 65 3d
0070   31 38 30 30 0d 0a 4c 4f   43 41 54 49 4f 4e 3a 20
0080   68 74 74 70 3a 2f 2f 31   30 2e 31 2e 32 2e 36 38
0090   3a 34 38 33 36 30 2f 64   65 73 63 72 69 70 74 69
00a0   6f 6e 2e 78 6d 6c 0d 0a   4e 54 3a 20 75 70 6e 70
00b0   3a 72 6f 6f 74 64 65 76   69 63 65 0d 0a 4e 54 53
00c0   3a 20 73 73 64 70 3a 61   6c 69 76 65 0d 0a 53 45
00d0   52 56 45 52 3a 20 45 6d   62 65 64 64 65 64 20 55
00e0   50 6e 50 2f 31 2e 30 0d   0a 55 53 4e 3a 20 75 75
00f0   69 64 3a 75 70 6e 70 5f   56 53 33 31 30 30 2d 30
0100   30 30 32 44 31 30 31 45   37 39 33 3a 3a 75 70 6e
0110   70 3a 72 6f 6f 74 64 65   76 69 63 65 0d 0a 0d 0a
```

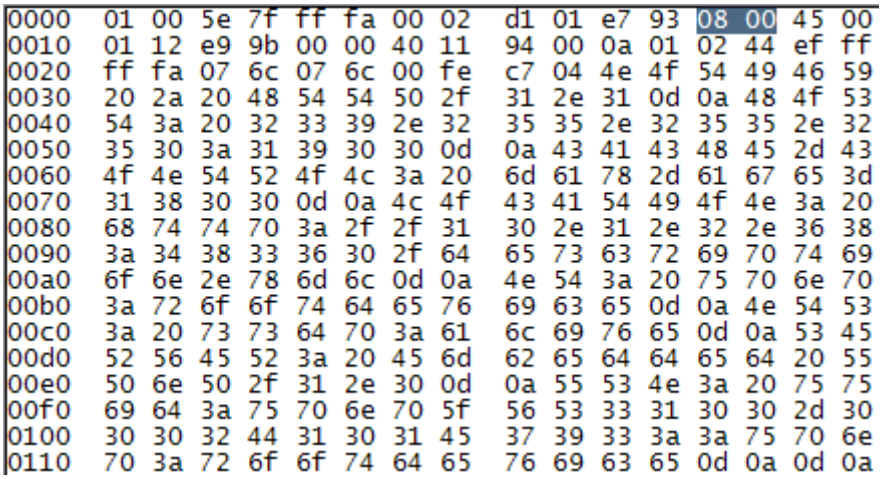
```
0000   01 00 5e 7f ff fa 00 02   d1 01 e7 93 08 00 45 00
0010   01 12 e9 9b 00 00 40 11   94 00 0a 01 02 44 ef ff
0020   ff fa 07 6c 07 6c 00 fe   c7 04 4e 4f 54 49 46 59
0030   20 2a 20 48 54 54 50 2f   31 2e 31 0d 0a 48 4f 53
0040   54 3a 20 32 33 39 2e 32   35 35 2e 32 35 35 2e 32
0050   35 30 3a 31 39 30 30 0d   0a 43 41 43 48 45 2d 43
0060   4f 4e 54 52 4f 4c 3a 20   6d 61 78 2d 61 67 65 3d
0070   31 38 30 30 0d 0a 4c 4f   43 41 54 49 4f 4e 3a 20
0080   68 74 74 70 3a 2f 2f 31   30 2e 31 2e 32 2e 36 38
0090   3a 34 38 33 36 30 2f 64   65 73 63 72 69 70 74 69
00a0   6f 6e 2e 78 6d 6c 0d 0a   4e 54 3a 20 75 70 6e 70
00b0   3a 72 6f 6f 74 64 65 76   69 63 65 0d 0a 4e 54 53
00c0   3a 20 73 73 64 70 3a 61   6c 69 76 65 0d 0a 53 45
00d0   52 56 45 52 3a 20 45 6d   62 65 64 64 65 64 20 55
00e0   50 6e 50 2f 31 2e 30 0d   0a 55 53 4e 3a 20 75 75
00f0   69 64 3a 75 70 6e 70 5f   56 53 33 31 30 30 2d 30
0100   30 30 32 44 31 30 31 45   37 39 33 3a 3a 75 70 6e
0110   70 3a 72 6f 6f 74 64 65   76 69 63 65 0d 0a 0d 0a
```

Quindi gli indirizzi MAC saranno:

MAC Address destinatario: 01:00:5E:7F:FF:FA

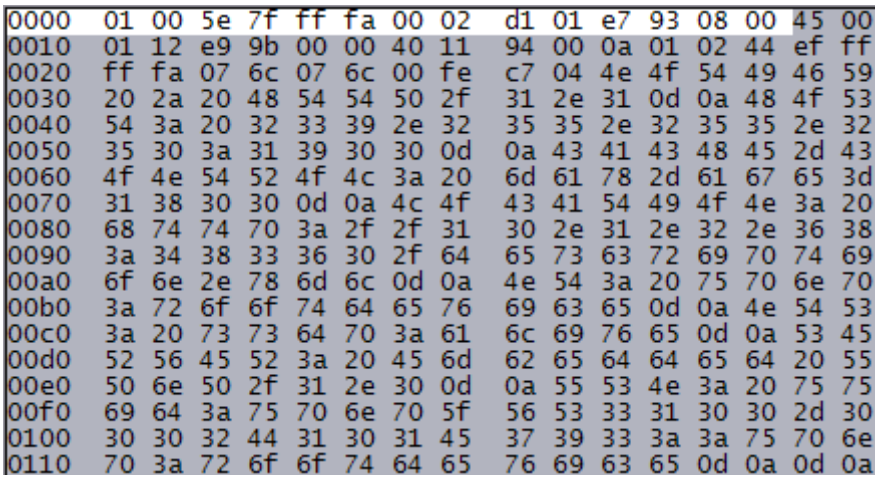
MAC Address sorgente: 00:02:D1:01:E7:93

Altre informazioni presenti nel header Ethernet sono le seguenti:



Protocollo successivo: IPv4

A partire dal byte successivo (campo Payload) il pacchetto dovrà essere interpretato con la codifica del protocollo successivo:



Il frame ricevuto può contenere errori, la maggior parte dei quali sono verificabili dal controllo CRC. Un frame che non supera il controllo CRC, viene scartato. Ethernet non prevede la ritrasmissione del frame scartato, né una notifica della sua perdita agli strati superiori. Ethernet è quindi inaffidabile, ma anche semplice ed economica.
Pertanto il campo relativo al Cyclic Redundancy Check non è visibile a livello software, in quanto viene verificato direttamente a livello hw e se vengono riscontrati errori nella trasmissione del pacchetto, esso viene scartato.

DUMP DEL PROTOCOLLO SUCCESSIVO

Di volta in volta, in funzione del protocollo identificato, è necessario eseguire un blocco diverso di istruzioni:

```
if(ntohs(pEthHeader->h_proto) == ETH_P_IP)
```

```
if(ntohs(pEthHeader->h_proto) == ETH_P_ARP)
```

```
if(ntohs(pEthHeader->h_proto) == ETH_P_RARP)
```

```
if(ntohs(pEthHeader->h_proto) == ETH_P_ARP)
```

```
if(ntohs(pEthHeader->h_proto) == ETH_P_IPV6)
```

```
if(ntohs(pEthHeader->h_proto) >= ETH_P_LMIN
&& ntohs(pEthHeader->h_proto) <= ETH_P_LMAX)
```

```
else // Altro tipo di protocollo
```

Gli unici pacchetti del presente livello che proseguiranno nell'analisi e verranno eventualmente inoltrati allo stack tcp/ip (e quindi non saranno scartati) saranno quelli IP, ARP e RARP.

La funzione **ntohs** converte il campo **h_proto** (che è un valore **u_short**) dal formato con ordine di tipo “**network byte**” al formato con ordine “**host byte**” (che solitamente è **little-endian**).

BIG ENDIAN E LITTLE ENDIAN

Gli aggettivi big-endian e little-endian si riferiscono ai bytes che sono più significativi in un tipo di dato multi-byte e descrivono l’ordine in cui una sequenza di bytes è memorizzata in memoria.
In un sistema big-endian, il valore più significativo della sequenza è memorizzato all’indirizzo di valore minore, mentre in un sistema little-endian il valore meno significativo della sequenza è memorizzato per primo. Per esempio, considerando il numero 1025 (2 elevato alla 20 + 1) memorizzato in un integer di 4 bytes:
00000000 00000000 00000100 00000001

Address	**Rappresentazione Big-Endian di 1025**	**Rappresentazione Little-Endian di 1025**
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000

I PACCHETTI ARP

Il protocollo ARP (Address Resolution Protocol), come specificato da RFC 826, è un protocollo che fornisce la "mappatura" tra l'indirizzo IP a 32bit (4byte) di un calcolatore e il suo MAC address, l'indirizzo fisico a 48bit (6 byte).
Quinti ARP è un protocollo di servizio, utilizzato in una rete di calcolatori che utilizzi il protocollo di rete IP sopra una rete di livello datalink che supporti il servizio di

broadcast. Se questo servizio non è disponibile, come ad esempio in ATM, devono essere utilizzati altri meccanismi.
Per inviare un pacchetto IP ad un calcolatore della stessa sottorete, è necessario incapsularlo in un pacchetto di livello datalink, che dovrà avere come indirizzo destinazione il mac address del calcolatore a cui lo si vuole inviare. ARP viene utilizzato per ottenere questo indirizzo.
Se il pacchetto deve essere inviato ad un calcolatore di un'altra sottorete, ARP viene utilizzato per scoprire il mac address del gateway.
In ogni calcolatore, il protocollo ARP tiene traccia delle risposte ottenute in una apposita cache, per evitare di utilizzare ARP prima di inviare ciascun pacchetto. Le voci della cache ARP vengono cancellate dopo un certo tempo di inutilizzo.

COME FUNZIONA ARP

L'host che vuole conoscere il mac address di un altro host, di cui conosce l'indirizzo IP, invia in broadcast una richiesta ARP (ARP-request), generata dal protocollo IP, contenente l'indirizzo IP dell'host di destinazione ed il proprio indirizzo MAC.
Tutti i calcolatori della sottorete ricevono la richiesta. In ciascuno di essi il protocollo ARP verifica se viene richiesto il proprio indirizzo IP. L'host di destinazione che riconoscerà il proprio IP nel pacchetto di ARP-request, provvederà ad inviare una risposta (ARP-reply) in unicast all'indirizzo MAC sorgente, contenente il proprio MAC.
In questo modo, ogni host può scoprire l'indirizzo fisico degli altri host sulla stessa sottorete.

IL PROTOCOLLO RARP

Reverse address resolution protocol (RARP) è un protocollo usato per risolvere un indirizzo fisico (indirizzo MAC) in un indirizzo IP. Questo sistema consente ad un host di conoscere un indirizzo IP chiedendolo in modalità broadcast agli altri host connessi alla rete, partendo dal MAC Address. RARP quindi è il protocollo inverso all'ARP ed è descritto nel RFC 903.

E’ pertanto necessario far “passare” tutti i pacchetti di tipo ARP e RARP al fine di garantire il corretto funzionamento della rete del cliente.

INOLTRO DEI PACCHETTI ARP/RARP ALLO STACK TCP/IP

Per inoltrare il pacchetto allo stack TCP/IP viene utilizzata la seguente istruzione:

```
api.SendPacketToMstcp(&Request);
```

Questa istruzione non fa altro che inviare il pacchetto RAW Ethernet al MSTCP, così come è stato ricevuto in origine dall’interfaccia di rete.

ESEMPIO: ARP Request:

```
0000  ff ff ff ff ff ff 00 0f  b0 59 d1 56 08 06 00 01
0010  08 00 06 04 00 01 00 0f  b0 59 d1 56 0a 01 02 4a
0020  00 00 00 00 00 00 0a 01  02 46 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 00
```

In questo caso si può notare che i valori dei campi del pacchetto Ethernet sono i seguenti:

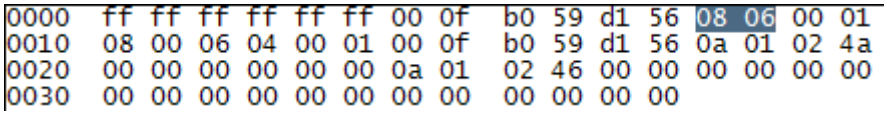
MAC Address destinatario: FF:FF:FF:FF:FF:FF – indica che il pacchetto è inoltrato in modalità broadcast (a tutti)

```
0000  ff ff ff ff ff ff 00 0f  b0 59 d1 56 08 06 00 01
0010  08 00 06 04 00 01 00 0f  b0 59 d1 56 0a 01 02 4a
0020  00 00 00 00 00 00 0a 01  02 46 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 00
```

MAC Address sorgente: 00:0F:B0:59:D1:56

```
0000  ff ff ff ff ff ff 00 0f  b0 59 d1 56 08 06 00 01
0010  08 00 06 04 00 01 00 0f  b0 59 d1 56 0a 01 02 4a
0020  00 00 00 00 00 00 0a 01  02 46 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 00
```

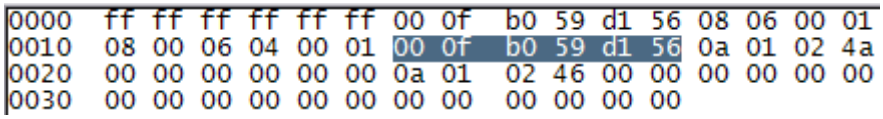
Protocollo successivo: ARP – 0x0806



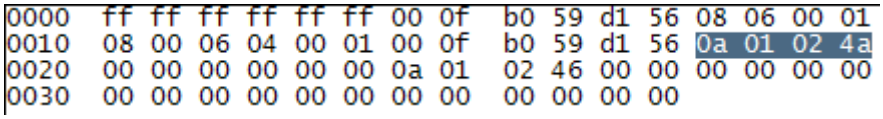
La specifica richiesta ARP Request è stata inoltrata dall'host con Mac Address 00:0F:B0:59:D1:56 (indirizzo IP 10.1.2.74) a tutti, per sapere qual è il Mac Address dell'host con indirizzo IP 10.1.2.70.

Questo si evince dai seguenti valori:

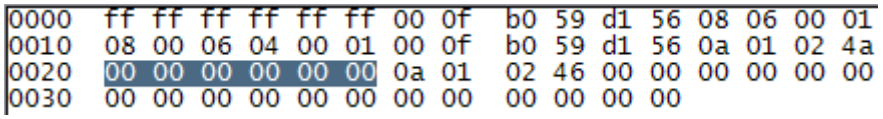
MAC Address sorgente: è uguale a quello indicato nell'header Ethernet



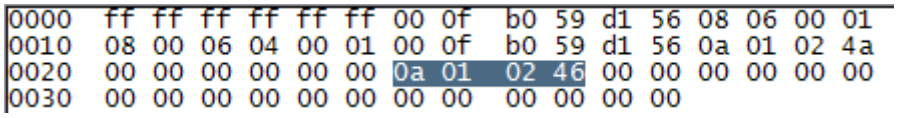
Indirizzo IP sorgente: 10.1.2.74



MAC Address destinatario: è il dato mancante



Indirizzo IP destinatario, del quale si vuole conoscere il MAC Address: 10.1.2.70



ESEMPIO: ARP Reply alla richiesta precedente

```
0000   00 0f b0 59 d1 56 00 02   3f d2 ea 66 08 06 00 01
0010   08 00 06 04 00 02 00 02   3f d2 ea 66 0a 01 02 46
0020   00 0f b0 59 d1 56 0a 01   02 4a
```

Possiamo notare che il pacchetto è indirizzato all'host con MAC Address 00:0F:B0:59:D1:56 e cioè proprio colui che aveva effettuato la richiesta.
Il valore che in precedenza era mancante ora viene riportato all'interno del pacchetto:

MAC Address dell'host con indirizzo IP 10.1.2.70: 00:0F:B0:59:D1:56

```
0000   00 0f b0 59 d1 56 00 02   3f d2 ea 66 08 06 00 01
0010   08 00 06 04 00 02 00 02   3f d2 ea 66 0a 01 02 46
0020   00 0f b0 59 d1 56 0a 01   02 4a
```

I PACCHETTI IP (IPv4)

IPv4 è la versione di rappresentazione di indirizzi IP attualmente in uso dell'Internet Protocol. Esso è descritto nell'IETF RFC 791, pubblicato per la prima volta nel settembre 1981.

Nel caso in cui il pacchetto contenuto in quello Ethernet sia di tipo IP, viene utilizzata la struttura dati pIpHeader indicata in precedenza:

```
iphdr_ptr pIpHeader;                    - puntatore header del pacchetto IP;
```

Quindi pIpHeader punterà al primo byte dell'header IP:

```
pPacket = &PacketBuffer;
pIpHeader = (iphdr_ptr)&pPacket->m_IBuffer[sizeof(ether_header)];
```

La struct iphdr è così composta:

```
typedef struct iphdr
{
```

```
        u_char      ip_hl:4,            /* header length */
                    ip_v:4;             /* version */
        u_char      ip_tos;             /* type of service */
        short   ip_len;                 /* total length */
        u_short     ip_id;              /* identification */
        short   ip_off;                 /* fragment offset field */
#define     IP_DF 0x4000                /* don't fragment flag */
#define     IP_MF 0x2000                /* more fragments flag */
        u_char      ip_ttl;             /* time to live */
        u_char      ip_p;               /* protocol */
        u_short     ip_sum;             /* checksum */
        struct  in_addr ip_src,ip_dst;  /* source and dest address */
} iphdr, *iphdr_ptr;
```

Il pacchetto IP viene definito datagramma. Lo schema seguente mostra come è fatto l'header del protocollo IPv4 (nella prima riga della tabella sono indicati i bit):

<table>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>0A</td><td>0B</td><td>0C</td><td>0D</td><td>0E</td><td>0F</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>1A</td><td>1B</td><td>1C</td><td>1D</td><td>1E</td><td>1F</td></tr>
<tr><td colspan="4">Versione</td><td colspan="4">IHL</td><td colspan="8">Servizio</td><td colspan="16">Lunghezza totale</td></tr>
<tr><td colspan="16">Identificazione</td><td colspan="4">Flags</td><td colspan="12">Offset frammentazione</td></tr>
<tr><td colspan="8">TTL</td><td colspan="8">Protocollo</td><td colspan="16">Checksum</td></tr>
<tr><td colspan="32">Indirizzo mittente</td></tr>
<tr><td colspan="32">Indirizzo destinatario</td></tr>
<tr><td colspan="24">Opzioni</td><td colspan="8">Padding</td></tr>
</table>

L'indirizzo IPv4 è formato da 32 bit, esso è univoco sulla rete di cui fa parte. Tale indirizzo, inoltre, non va assegnato all'host, ma alle connessioni fisiche alla rete che l'host possiede.

Concettualmente l'indirizzo IP si compone di due parti:

1. identificatore di rete e precisamente della sottorete
2. identificatore di host

Per semplificarne la lettura, ogni indirizzo IP viene descritto con 4 numeri in base decimale, in modo che ognuno rappresenti un byte (il valore di un byte varia da 0 a

255 quando lo consideriamo in base dieci), separati dal simbolo "punto"; un esempio di indirizzo IPv4 è 192.0.34.166.

Il numero di indirizzi univoci disponibili in IPv4 è $2^{32} = 256^4 = 4.294.967.296 \cong 4,3 \cdot 10^9$, ma bisogna tener presente che non vengono usati tutti, perché alcuni sono riservati a un particolare utilizzo (ad esempio gli indirizzi 0.0.0.0, 255.255.255.255, 192.0.34.166 e la classe 192.168.0.1/16) e perché certe classi non vengono sfruttate interamente per via della suddivisione interna in classi più piccole.
Gli indirizzi IP sono univoci a livello mondiale (o a livello di singola rete), e vengono assegnati in modo centralizzato da una gerarchia di enti appositi. Sono considerati una risorsa preziosa da gestire con cura. Per rafforzare questo concetto, si parla di "indirizzi IP pubblici".
Inizialmente l'autorità preposta era la **IANA** (Internet Assigned Number Authority), dopo il 1998 venne creato l'ICANN (Internet corporation for Assigned Names and Numbers) che opera tuttora. Essa è responsabile della gestione degli indirizzi IP in base alle direttive dell'RFC 2050.

CONTROLLI EFFETTUATI SUL PACCHETTO IP:

HEADER CHECKSUM

Il primo controllo che viene effettuato sul pacchetto IP è la verifica della correttezza del checksum dell'header.

L'header del pacchetto IP si presenta come in seguito (parte evidenziata, la sezione precedente rappresenta l'header del pacchetto ethenet, mentre la parte successiva il vero e proprio contenuto del pacchetto IP):

```
0000   01 00 5e 7f ff fa 00 02   d1 01 e7 93 08 00 45 00
0010   01 12 e9 9b 00 00 40 11   94 00 0a 01 02 44 ef ff
0020   ff fa 07 6c 07 6c 00 fe   c7 04 4e 4f 54 49 46 59
0030   20 2a 20 48 54 54 50 2f   31 2e 31 0d 0a 48 4f 53
0040   54 3a 20 32 33 39 2e 32   35 35 2e 32 35 35 2e 32
0050   35 30 3a 31 39 30 30 0d   0a 43 41 43 48 45 2d 43
0060   4f 4e 54 52 4f 4c 3a 20   6d 61 78 2d 61 67 65 3d
0070   31 38 30 30 0d 0a 4c 4f   43 41 54 49 4f 4e 3a 20
0080   68 74 74 70 3a 2f 2f 31   30 2e 31 2e 32 2e 36 38
0090   3a 34 38 33 36 30 2f 64   65 73 63 72 69 70 74 69
00a0   6f 6e 2e 78 6d 6c 0d 0a   4e 54 3a 20 75 70 6e 70
00b0   3a 72 6f 6f 74 64 65 76   69 63 65 0d 0a 4e 54 53
00c0   3a 20 73 73 64 70 3a 61   6c 69 76 65 0d 0a 53 45
00d0   52 56 45 52 3a 20 45 6d   62 65 64 64 65 64 20 55
00e0   50 6e 50 2f 31 2e 30 0d   0a 55 53 4e 3a 20 75 75
00f0   69 64 3a 75 70 6e 70 5f   56 53 33 31 30 30 2d 30
0100   30 30 32 44 31 30 31 45   37 39 33 3a 3a 75 70 6e
0110   70 3a 72 6f 6f 74 64 65   76 69 63 65 0d 0a 0d 0a
```

In particolare, i bytes che rappresentano il checksum dell'header sono:

```
0000   01 00 5e 7f ff fa 00 02   d1 01 e7 93 08 00 45 00
0010   01 12 e9 9b 00 00 40 11   94 00 0a 01 02 44 ef ff
0020   ff fa 07 6c 07 6c 00 fe   c7 04 4e 4f 54 49 46 59
0030   20 2a 20 48 54 54 50 2f   31 2e 31 0d 0a 48 4f 53
0040   54 3a 20 32 33 39 2e 32   35 35 2e 32 35 35 2e 32
0050   35 30 3a 31 39 30 30 0d   0a 43 41 43 48 45 2d 43
0060   4f 4e 54 52 4f 4c 3a 20   6d 61 78 2d 61 67 65 3d
0070   31 38 30 30 0d 0a 4c 4f   43 41 54 49 4f 4e 3a 20
0080   68 74 74 70 3a 2f 2f 31   30 2e 31 2e 32 2e 36 38
0090   3a 34 38 33 36 30 2f 64   65 73 63 72 69 70 74 69
00a0   6f 6e 2e 78 6d 6c 0d 0a   4e 54 3a 20 75 70 6e 70
00b0   3a 72 6f 6f 74 64 65 76   69 63 65 0d 0a 4e 54 53
00c0   3a 20 73 73 64 70 3a 61   6c 69 76 65 0d 0a 53 45
00d0   52 56 45 52 3a 20 45 6d   62 65 64 64 65 64 20 55
00e0   50 6e 50 2f 31 2e 30 0d   0a 55 53 4e 3a 20 75 75
00f0   69 64 3a 75 70 6e 70 5f   56 53 33 31 30 30 2d 30
0100   30 30 32 44 31 30 31 45   37 39 33 3a 3a 75 70 6e
0110   70 3a 72 6f 6f 74 64 65   76 69 63 65 0d 0a 0d 0a
```

Quindi il checksum dell'header del pacchetto sarà 0x9400.

Per capire se tale valore è corretto e quindi non ci sono stati errori nella trasmissione è necessario ricalcolarlo e confrontarlo con il valore all'interno del pacchetto.

Il calcolo del checksum avviene come segue:

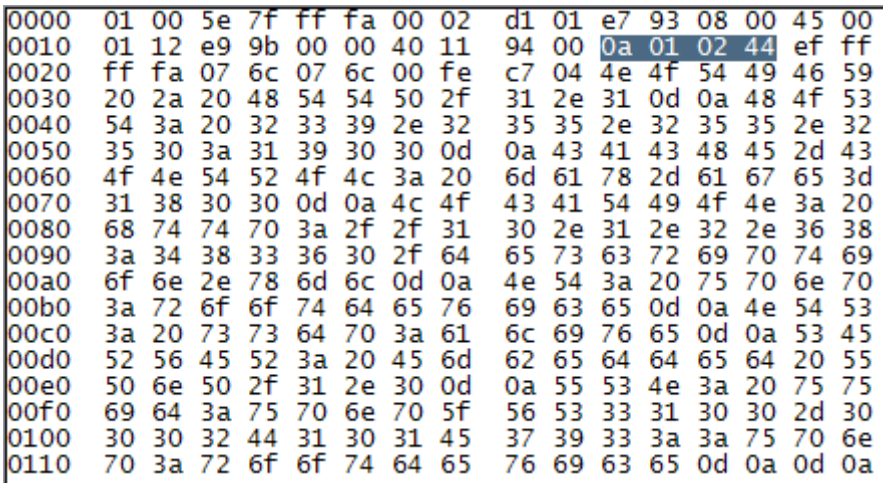
- suddivisione dell'header in blocchi da 16 bits (2 bytes)
- somma in complemento a 1 delle parti suddivise (per somma in complemento a 1 si intende che l'eventuale bit di resto deve essere sommato partendo dal bit meno significativo)
- calcolo (una volta effettuate le somme) del complemento a 1 del risultato (i bit 0 diventano 1 e viceversa)

Se il valore calcolato corrisponde a quello presente nel pacchetto (calcolato a sua volta dal mittente) allora il pacchetto procederà alla fase successiva di controllo altrimenti verrà scartato e si troverà traccia di tale situazione del file di log.

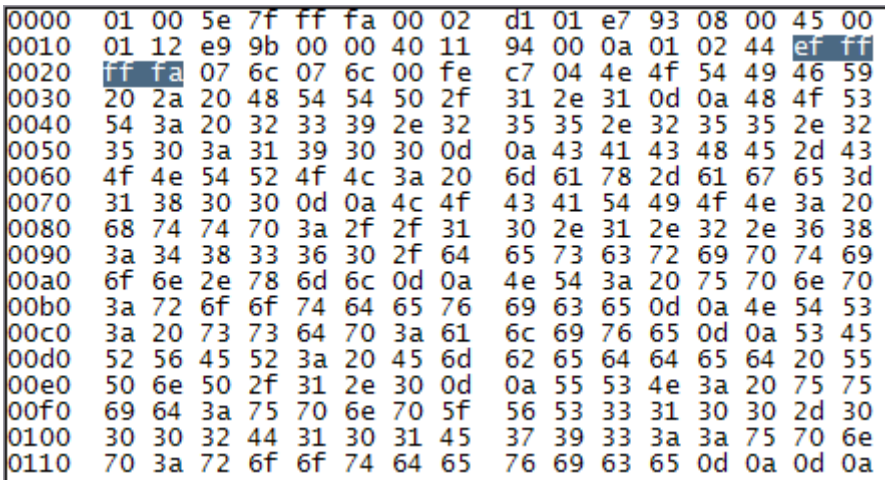
CONTROLLI SULL'INDIRIZZO IP DEL MITTENTE E DEL DESTINATARIO

Gli indirizzi IP sorgente e destinatario sono composti come segue:

Esempio:



Source: 10.1.2.68



Destination: 239.255.255.250

Ad ogni ciclo (ogni pacchetto esaminato) viene calcolato l'indirizzo IP locale dell'interfaccia di rete sulla quale è applicato il filtro.

Tale indirizzo viene confrontato con l'indirizzo del destinatario indicato nel pacchetto IP.

Successivamente, viene controllato l'indirizzo sorgente, verificando che sia compreso in quelli indicati in white-list.

Viene pertanto effettuato un ciclo sulle liste contenenti gli indirizzi IP consentiti, precedentemente memorizzati in memoria.

Per effettuare i controlli descritti in precedenza sono state codificate le seguenti funzioni, che restituiscono TRUE in caso di controlli con esito positivo e FALSE altrimenti:

```
If (ip_compare(ip_dest,localIP))
```

```
If (white_ip(ip_temp))
```

PROTOCOLLI SUCCESSIVI AL LIVELLO IP

Se il pacchetto IP ha superato tutti i controlli, dovrà essere verificata la natura del protocollo di livello successivo.

In particolare, i protocolli presi in considerazione ed utilizzati nella rete del cliente sono:

- TCP (saranno controllati i valori delle porte logiche)
- UDP (saranno controllati i valori delle porte logiche)
- IGMP (tutti i pacchetti IGMP sono consentiti)
- ICMP (tutti i pacchetti ICMP sono consentiti)

Inoltre, saranno presi in considerazione anche i seguenti casi particolari:

- Pacchetti DHCP (sono pacchetti UDP particolari)

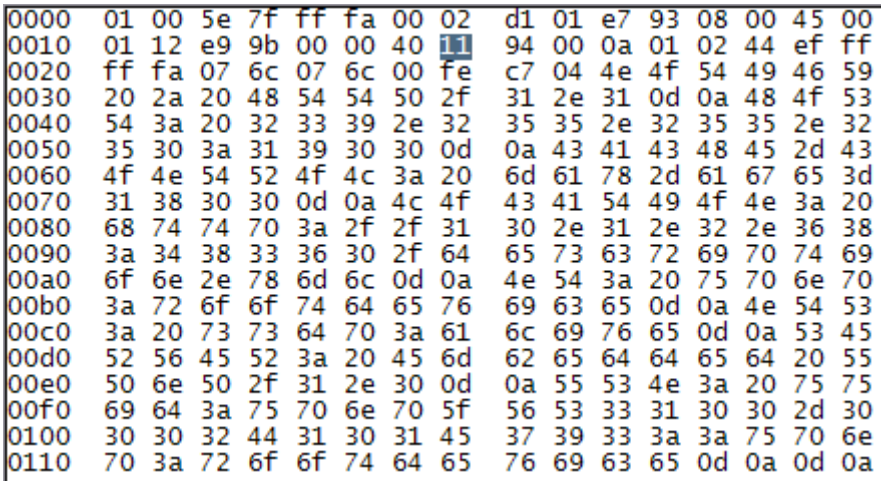
- Pacchetti frammentati
- Pacchetti Multicast (con indirizzi nel formato 224.*.*.*)

Per capire qual è il protocollo successivo ad IP, è necessario analizzare il campo **ip_p** dell'header:

pIpHeader->ip_p

che è rappresentato dai seguenti bytes:

Esempio:



Next Protocol: 0x11 UDP

Nella tabella seguente sono riassunti i principali codici di protocolli successivi ad IP:

DEFINIZIONE	HEX	DEC	SPIEGAZIONE
#define IPPROTO_ICMP	0x01	1	Pacchetto ICMP
#define IPPROTO_IGMP	0x02	2	Pacchetto IGMP
#define IPPROTO_TCP	0x06	6	Pacchetto TCP
#define IPPROTO_UDP	0x11	17	Pacchetto UDP

Protocollo ICMP

Il protocollo ICMP (Internet Control Message Protocol) è un protocollo di servizio che si preoccupa di trasmettere informazioni riguardanti malfunzionamenti, informazioni di controllo o messaggi tra i vari componenti di una rete di calcolatori.
ICMP è incapsulato direttamente in IP e viene utilizzato da molti programmi, tra cui ping e traceroute.
È definito nelle seguenti RFC:

- RFC 0792: Internet Control Message Protocol - settembre 1981
- RFC 1788: ICMP Domain Name Messages - aprile 1995
- RFC 1349: Type of Service in the Internet Protocol Suite - luglio 1992
- RFC 2463: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification - dicembre 1998
- RFC 2521: ICMP Security Failures Messages - marzo 1999

L'header del pacchetto ICMP è così composto:

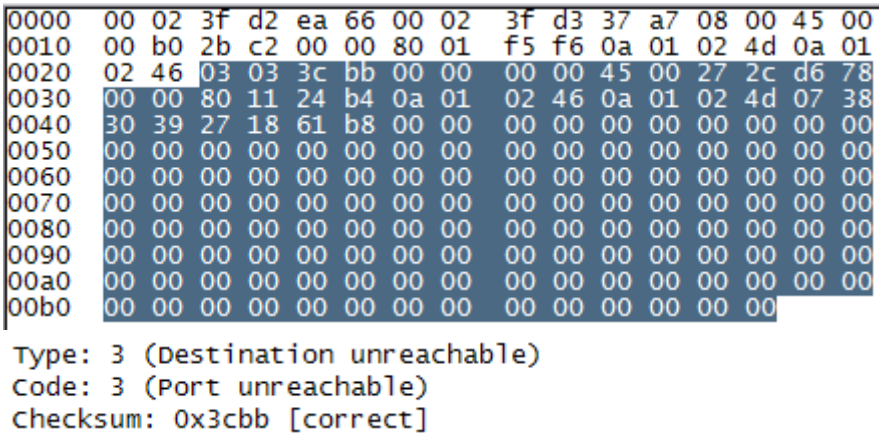
- Tipo: 1 Byte - specifica il formato del messaggio ICMP (La lista dei valori ammissibili per il campo 'tipo' è consultabile sulla RFC 1700: ASSIGNED NUMBERS;
- Codice: 1 Byte - ulteriore qualificazione del messaggio;
- Checksum dell'Header ICMP: 2 Bytes - controllo della correttezza del messaggio.

Successivamente è presente il contento vero e proprio del pacchetto ICMP, di lunghezza variabile in base al tipo di messaggio identificato dai campi "Tipo" e "Codice".

Come detto in precedenza, tutti i pacchetti ICMP vengono accettati ed inoltrati allo stack TCP/IP (devono avere chiaramente superato i controlli precedenti):

```
if(IpProto == IPPROTO_ICMP)
{
        api.SendPacketToMstcp(&Request);
}
```

Esempio pacchetto ICMP:



PROTOCOLLO IGMP

L'Internet Group Management Protocol è un protocollo per la gestione dei gruppi multicast. Costituisce, ad esempio, il mezzo per un host di informare il router ad esso collegato che un applicazione che funziona nell'host vuole unirsi a uno specifico gruppo multicast. IGMP opera fra un host e il router ad esso collegato direttamente, per coordinare i router multicast invece è richiesto un altro protocollo, così che i datagrammi multicast possano essere instradati alle loro destinazioni finali. Questa funzionalità è svolta da Algoritmi di Instradamento Multicast dello strato della rete: PIM, DVMRP, MOSFP.

I messaggi IGMP, la cui spiegazione è indicata a seguire, sono i seguenti:

- Membership query: da router e puo' essere Generale o Specifico;
- Membership report: da host;
- Leave group: da host.

Il primo messaggio consente di verificare se una sola macchina della rete è in ascolto. Se nessuna macchina è presente, il router termina la ricerca fino a nuova richiesta.

Il secondo messaggio viene inviato a tutta la rete con un IP broadcast e permette di richiedere il ricevimento del flusso dei dati. Esso corrisponde inoltre alla risposta del membership query del router che chiede se c'è qualcuno in ascolto.

Il terzo permette ad un host di finire la ricezione e lasciare il gruppo.

Con il termine Multicast, nelle reti di calcolatori, si indica la distribuzione simultanea di informazione verso un gruppo di destinazioni.
Il termine viene utilizzato anche per indicare un pacchetto inviato con tale modalità. Un indirizzo che si riferisce a un gruppo di destinazioni è detto a sua volta indirizzo Multicast.
In alternativa, un pacchetto destinato a tutti i calcolatori di una rete è detto Broadcast, uno destinato ad uno qualunque di un gruppo Anycast, uno destinato ad un solo calcolatore è Unicast.
Il modello di servizio multicast prevede che un calcolatore invii i pacchetti ad un indirizzo associato al gruppo multicast; il calcolatore sorgente invia una sola copia dell'informazione (indipendentemente dal numero di destinatari), saranno poi gli M-Router (Multicast Router) che moltiplicheranno l'informazione quando necessario. In questo modo se 50 computer (Gruppo) devono ricevere lo stesso file dalla stessa sorgente, quest'ultima invierà una sola copia del file, man mano che si naviga nella rete saranno gli M-Router che moltiplicheranno le informazioni fino al raggiungimento dei 50 computer. I computer che vogliono ricevere le "trasmissioni" del gruppo multicast si devono registrare per quel gruppo con qualche meccanismo, e la rete si occuperà di consegnare i pacchetti multicast a tutti quelli che si sono registrati.

La rete del cliente utilizza anche il protocollo IGMP; è pertanto necessario permettere il transito di pacchetti IGMP in ingresso sui client.

Per questo motivo viene effettuato un duplice controllo per la verifica del traffico multicast.

Innanzitutto viene verificato l'indirizzo ip di destinazione (quindi quello relativo al client che fa parte di un gruppo multicast) che deve necessariamente iniziare con **224.*.*.*** :

```
if ( (ip_dest[0]='2') && (ip_dest[1]='2') && (ip_dest[2]='4') )
```

Inoltre viene verificato che il protocollo successivo sia di tipo IGMP e quindi effettivamente l'interpretazione del pacchetto con indirizzo di destinazione multicast debba essere quella di un pacchetto IGMP:

```
if(IpProto == IPPROTO_IGMP)
```

Resta inteso che l'indirizzo sorgente deve essere presente in whit-list per poter effettivamente raggiungere lo stack tcp/ip.

PROTOCOLLO TCP

Transmission Control Protocol (TCP) è uno dei principali protocolli della Suite di protocolli Internet. TCP è il protocollo di trasporto, definito nel RFC 793, su cui si appoggiano gran parte delle applicazioni Internet.
Il TCP è un protocollo corrispondente al livello 4 (trasporto) del modello di riferimento OSI, e di solito è usato in combinazione con il protocollo di livello 3 (rete) IP. La corrispondenza con il modello OSI non è perfetta, in quanto il TCP e l'IP nascono prima. La loro combinazione è indicata come TCP/IP ed è, alle volte, erroneamente considerata un unico protocollo.
Il TCP nacque nel 1970 come frutto del lavoro di un gruppo di ricerca del dipartimento di difesa statunitense. I suoi punti di forza sono l'alta affidabilità e robustezza. La sua popolarità si deve anche grazie ad una sua implementazione diffusa dalla Università di Berkeley in California sotto forma di sorgenti.

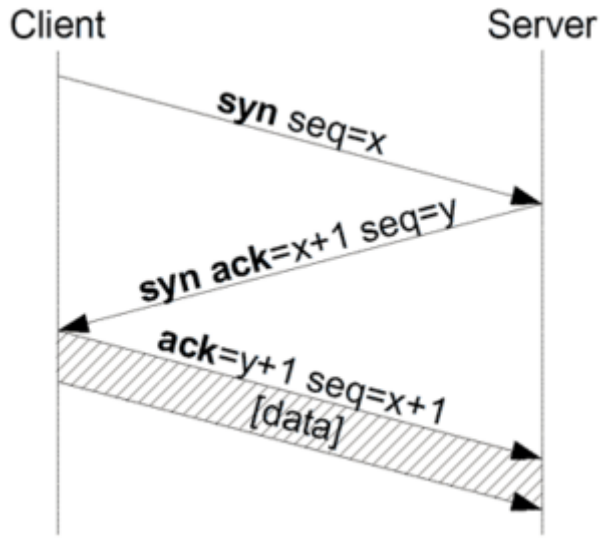
Le caratteristiche principali del TCP sono:

- La creazione di una connessione (protocollo orientato alla connessione)
- La gestione di connessioni punto-punto
- La garanzia che i dati trasmessi giungano a destinazione in ordine e senza perdita di informazione (tramite il meccanismo di acknowlegment e ritrasmissione).

Attraverso il meccanismo della finestra scorrevole, offre funzionalità di controllo di flusso e controllo della congestione, vitali per il buon utilizzo della rete IP, che non

offre alcuna garanzia in ordine alla consegna dei pacchetti, al ritardo, alla congestione.
Una funzione di multiplazione delle connessioni ottenuta attraverso il meccanismo delle porte.

INSTAURAZIONE DELLA CONNESSIONE TCP

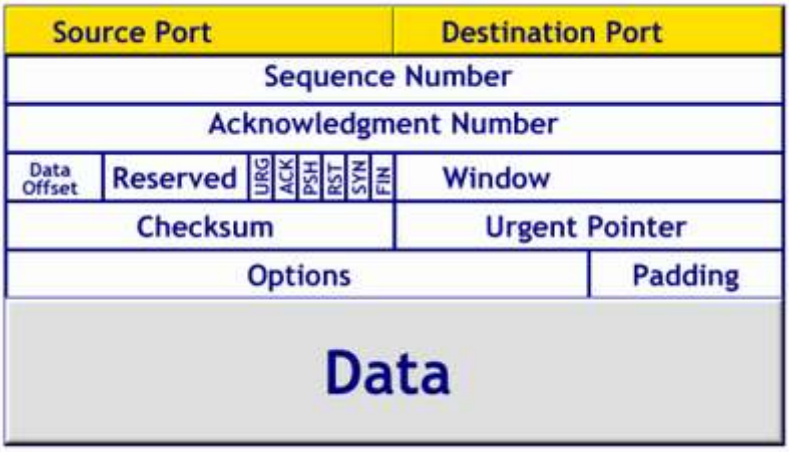


La procedura utilizzata per instaurare in modo affidabile una connessione TCP tra due host è chiamata three-way handshake (triplice stretta di mano), ad indicare la necessità di scambiare tre messaggi per garantire la corretta creazione della connessione. Supponiamo, per esemplificare, che l'host A (il client) intenda instaurare una comunicazione TCP con l'host B (il server); i passi indicati dalla tecnica three-way handshake sono:

1. A invia un segmento SYN a B, contenente il suo sequence number x;
2. B invia un segmento SYN/ACK ad A, contenente il suo sequence number y e l'acknowledgment del sequence number x di A;
3. A invia un segmento ACK a B con l'acknowledgement del sequence number y di B.

Avendo chiamate SYN poi insieme SYN + ACK e infine ACK se si cercano solo i segmenti di tipo ACK si ottengono tutte le nuove connessioni instaurate.

HEADER DEL PACCHETTO TCP



L'header di un segmento TCP è così strutturato:

- Porta sorgente (Source port) [16 bit]
- Porta di destinazione (Destination port) [16 bit]
- Numero di sequenza (Sequence number) [32 bit], indica la posizione del primo byte di dati del segmento TCP all'interno del flusso completo; se il flag SYN è impostato, il valore del sequence number corrisponde all'Initial Sequence Number (ISN);
- Numero di acknowledgment (Acknowledgment number) [32 bit], contiene il valore del prossimo sequence number che la sorgente del segmento TCP è in attesa di ricevere ed è utilizzato congiuntamente al flag ACK;
- Data offset [4 bit], indica la lunghezza (in word da 32 bit) dell'header del segmento TCP;
- 6 bit riservati (Reserved), non utilizzati e predisposti per sviluppi futuri del protocollo;
- Bit di controllo (Control bits) [6 bit], possono essere impostati ad 1 o 0 e indicano:
    - URG: il valore dell'urgent pointer è valido;
    - ACK: il valore dell'acknowledgment number è valido;
    - PSH: l'host che riceve il segmento TCP deve provvedere a trasferire i dati al Livello applicazioni il più velocemente possibile;
    - RST: reset della connessione;
    - SYN: se impostato, indica che si tratta del primo segmento della connessione;

    - FIN: se impostato, indica che si tratta dell'ultimo segmento della connessione;
- Finestra (Window) [16 bit], indica il numero di byte che il mittente è in grado di accettare a partire dal byte indicato dall'acknowledgment number;
- Checksum [16 bit], utilizzato per il controllo della validità del segmento;
- Urgent pointer [16 bit], puntatore al sequence number di dati con priorità di trasferimento;
- Opzioni (facoltative)
- Padding, utilizzato per completare i bit non utilizzati dalle opzioni

Quindi, nel caso in cui il pacchetto identificato sia di tipo TCP, la struttura dati utilizzata **pTcpHeader** sarà di tipo **tcphdr_ptr** :

```
tcphdr_ptr pTcpHeader;
```

che è così dichiarata:

```
// TCP header. Per RFC 793, September, 1981. In Little Endian
typedef struct tcphdr {
        u_short     th_sport;          /* source port */
        u_short     th_dport;          /* destination port */
        tcp_seq     th_seq;            /* sequence number */
        tcp_seq     th_ack;            /* acknowledgement number */
        u_char      th_x2:4,           /* (unused) */
                    th_off:4;          /* data offset */
        u_char      th_flags;
#define     TH_FIN      0x01
#define     TH_SYN      0x02
#define     TH_RST      0x04
#define     TH_PSH      0x08
#define     TH_ACK      0x10
#define     TH_URG      0x20
        u_short     th_win;            /* window */
        u_short     th_sum;            /* checksum */
```

```
        u_short         th_urp;                 /* urgent pointer */
} tcphdr, *tcphdr_ptr;
```

Pertanto, nel caso in cui venga riscontrata vera la condizione:

```
if(IpProto == IPPROTO_TCP && !frammento(pIpHeader->ip_off))
```

che indica che il protocollo successivo ad IP è TCP ed il pacchetto TCP non è frammentato (la frammentazione verrà trattata più avanti), pTcpHeader punterà all'header del pacchetto TCP:

```
pTcpHeader = (tcphdr_ptr)(((PUCHAR)pIpHeader) + sizeof(DWORD)*pIpHeader-
>ip_hl);
```

CONTROLLI EFFETTUATI SUL PACCHETTO TCP

Come già indicato in precedenza, le ACL (Access Control Lists) sono definite in termini di indirizzamento IP e porte logiche.
Pertanto sarà necessario controllare che la porta TCP di destinazione sul client locale sia presente nella lista delle porte consentite. Tale ultimo controllo consentirà al pacchetto TCP di raggiungere lo stack TCP/IP.

I valori delle porte logiche cosentite, inizialmente presenti nel file di configurazione e poi successivamente caricate in memoria, hanno un valore intero.

E' necessario trasformare il valore da ordine "network byte" ad "host byte" utilizzando la funzione ntohs:

```
dTcpPort = ntohs(pTcpHeader->th_dport);
```

Successivamente, mediante la funzione **port_white** viene verificato tale valore in white-list per tutti i record che si riferiscono al protocollo TCP:

```
if(port_white(dTcpPort,"TCP"))
```

In caso positivo il pacchetto termina positivamente il proprio percorso:

```
api.SendPacketToMstcp(&Request);
```

Per quanto concerne le porte logiche, non vengono riportate in allegato a causa del numero particolarmente alto. E' possibile reperire l'elenco ufficiale aggiornato delle Well-known ports, Registered Ports e Dynamic / Private Ports al seguente indirizzo:

http://www.iana.org/assignments/port-numbers

L'organismo preposto all'aggiornamento di tale elenco è IANA (Internet Assigned Numbers Authority).

ESEMPIO PACCHETTO TCP

IP Next Protocol: 0x06 (TCP)

```
0000   00 00 0c 07 ac 02 00 02   3f d2 ea 66 08 00 45 00
0010   00 28 cd cd 40 00 80 06   13 a4 0a 01 02 46 0a 01
0020   03 17 06 b6 01 bb 8f a7   e8 6c e0 f4 5e 7c 50 10
0030   ff 84 d6 fa 00 00
```

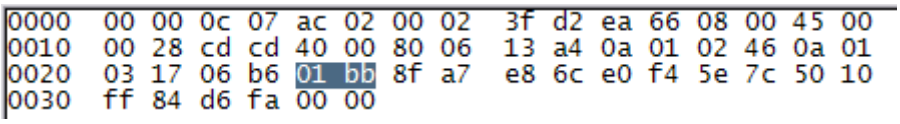
Header TCP

```
0000   00 00 0c 07 ac 02 00 02   3f d2 ea 66 08 00 45 00
0010   00 28 cd cd 40 00 80 06   13 a4 0a 01 02 46 0a 01
0020   03 17 06 b6 01 bb 8f a7   e8 6c e0 f4 5e 7c 50 10
0030   ff 84 d6 fa 00 00
```

Source Port: 0x06B6 = 1718

```
0000   00 00 0c 07 ac 02 00 02   3f d2 ea 66 08 00 45 00
0010   00 28 cd cd 40 00 80 06   13 a4 0a 01 02 46 0a 01
0020   03 17 06 b6 01 bb 8f a7   e8 6c e0 f4 5e 7c 50 10
0030   ff 84 d6 fa 00 00
```

Destination Port: 0x01BB = 443



PROTOCOLLO UDP

L'User Datagram Protocol (UDP) è uno dei principali protocolli della Suite di protocolli Internet. UDP è un protocollo di trasporto a pacchetto. È usato di solito in combinazione con il protocollo IP.

A differenza del TCP, non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi. L'UDP ha come caratteristica di essere un protocollo di rete inaffidabile, privo di connessione, ma in compenso molto rapido ed efficiente per le applicazioni "leggere" o time-sensitive. Infatti, è usato spesso per la trasmissione di informazioni audio o video. Dato che le applicazioni in tempo reale spesso richiedono un ritmo minimo di spedizione, non vogliono ritardare eccessivamente la trasmissione dei pacchetti e possono tollerare qualche perdita di dati, il modello di servizio TCP può non essere particolarmente adatto alle esigenze di queste applicazioni. L'UDP fornisce soltanto i servizi basilari del livello di trasporto, ovvero:

- multiplazione delle connessioni, ottenuta attraverso il meccanismo delle porte
- verifica degli errori mediante una checksum, inserita in un campo dell'intestazione del pacchetto.

mentre TCP garantisce anche il trasferimento affidabile dei dati, il controllo di flusso e il controllo della congestione.

UDP è un protocollo stateless ovvero privo di stato: non mantiene lo stato della connessione dunque rispetto a TCP ha informazioni in meno da memorizzare. Un server dedicato ad una particolare applicazione che sceglie UDP come protocollo di trasporto può supportare molti più client attivi.

Applicazioni che utilizzano UDP

Le applicazioni che hanno la necessità di un trasferimento affidabile dei loro dati si affidano ovviamente a TCP. Le applicazioni più elastiche riguardo alla perdita dei dati e dipendenti dal tempo si affidano invece a UDP. Inoltre si utilizza UDP per comunicazioni in broadcast (invio a tutti i terminali in una rete locale) e multicast (invio a tutti i terminali iscritti ad un servizio).
Un esempio delle applicazioni più diffuse e dei protocolli che adottano:

Applicazione	**Protocollo strato applicazione**	**Protocollo strato trasporto**
Posta elettronica	SMTP	TCP
Accesso a terminale remoto	telnet	TCP
Trasferimento file	FTP	TCP
Web	HTTP	TCP
Server di file remoto	NFS	tipicamente UDP
Telefonia su internet	proprietario	tipicamente UDP
Gestione della rete	SNMP	tipicamente UDP
Protocollo di routing	RIP	tipicamente UDP
Risoluzione dei nomi	DNS	tipicamente UDP
Configurazione dinamica indirizzi	DHCP	tipicamente UDP

Struttura di un datagramma UDP

Un datagramma UDP è composto da 5 campi di cui due sono opzionali. I campi sono:

Porta Sorgente	Porta Destinazione
Lunghezza	Checksum
Dati	

Porta Sorgente
contiene la porta sorgente da cui proviene il datagramma (Lunghezza : 16 bit)

Porta Destinazione
contiene la porta destinazione a cui deve essere spedito il datagramma (Lunghezza : 16 bit) .

Lunghezza
contiene la lunghezza totale del datagramma UDP (testata+dati) (Lunghezza : 16 bit)

Checksum
contiene il codice di controllo del datagramma (testata+dati). L'algoritmo di calcolo è definito nell' RFC del protocollo (Lunghezza: 16 bit)

Dati
contiene i dati del datagramma (Lunghezza: in teoria infinita)

Quindi, nel caso in cui il pacchetto identificato sia di tipo UDP, la struttura dati utilizzata **pUdpHeader** sarà di tipo **udphdr_ptr** :

```
udphdr_ptr pUdpHeader
```

che sarà così dichiarata:

```
/* UDP header  */
typedef struct          udphdr
```

```
{
        u_short     th_sport;          /* source port */
        u_short     th_dport;          /* destination port */
        u_short     length;            /* data length */
        u_short     th_sum;            /* checksum */
} udphdr, *udphdr_ptr;
```

Pertanto, nel caso in cui venga riscontrata vera la condizione:

```
if(IpProto == IPPROTO_UDP && !frammento(pIpHeader->ip_off))
```

che indica che il protocollo successivo ad IP è UDP ed il pacchetto UDP non è frammentato (la frammentazione verrà trattata più avanti), **pUdpHeader** punterà all'header del pacchetto UDP:

```
pUdpHeader = (udphdr_ptr)(((PUCHAR)pIpHeader) + sizeof(DWORD)*pIpHeader->ip_hl);
```

CONTROLLI EFFETTUATI SUL PACCHETTO UDP

I pacchetti UDP verranno trattati con una metodologia molto simile a quella utilizzata per i pacchetti TCP. In particolare, le ACL (Access Control Lists) sono definite in termini di indirizzamento IP e porte logiche.
Pertanto sarà necessario controllare che anche la porta UDP di destinazione sul client locale sia presente nella lista delle porte consentite. Tale ultimo controllo consentirà al pacchetto UDP di raggiungere lo stack TCP/IP ed essere poi interpretato correttamente a livello applicativo.

I valori delle porte logiche cosentite, inizialmente presenti nel file di configurazione e poi successivamente caricate in memoria, hanno un valore intero.

E' necessario trasformare il valore da ordine "network byte" ad "host byte" utilizzando la funzione ntohs:

```
dUdpPort = ntohs(pUdpHeader->th_dport);
```

Successivamente, mediante la funzione port_white viene verificato tale valore in white-list per tutti i record che si riferiscono al protocollo UDP:

```
if(port_white(dUdpPort,"UDP"))
```

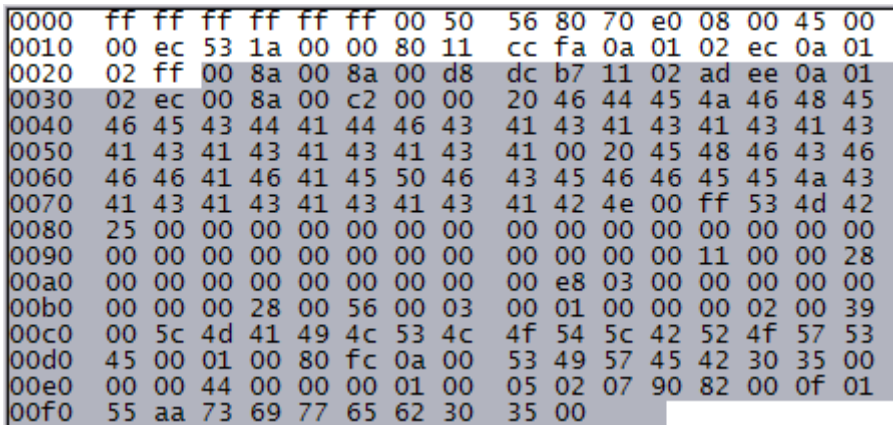
In caso positivo il pacchetto termina positivamente il proprio percorso:

```
api.SendPacketToMstcp(&Request);
```

Come indicato per le porte TCP, l'elenco delle porte UDP ufficiale ed aggiornato (Well-known ports, Registered Ports e Dynamic / Private Ports) è reperibile al seguente indirizzo:

http://www.iana.org/assignments/port-numbers

ESEMPIO PACCHETTO UDP



IP Next Protocol: 0x11 (UDP)

```
0000  ff ff ff ff ff ff 00 50  56 80 70 e0 08 00 45 00
0010  00 ec 53 1a 00 00 80 11  cc fa 0a 01 02 ec 0a 01
0020  02 ff 00 8a 00 8a 00 d8  dc b7 11 02 ad ee 0a 01
0030  02 ec 00 8a 00 c2 00 00  20 46 44 45 4a 46 48 45
0040  46 45 43 44 41 44 46 43  41 43 41 43 41 43 41 43
0050  41 43 41 43 41 43 41 43  41 00 20 45 48 46 43 46
0060  46 46 41 46 41 45 50 46  43 45 46 46 45 45 4a 43
0070  41 43 41 43 41 43 41 43  41 42 4e 00 ff 53 4d 42
0080  25 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090  00 00 00 00 00 00 00 00  00 00 00 00 11 00 00 28
00a0  00 00 00 00 00 00 00 00  00 e8 03 00 00 00 00 00
00b0  00 00 00 28 00 56 00 03  00 01 00 00 00 02 00 39
00c0  00 5c 4d 41 49 4c 53 4c  4f 54 5c 42 52 4f 57 53
00d0  45 00 01 00 80 fc 0a 00  53 49 57 45 42 30 35 00
00e0  00 00 44 00 00 00 01 00  05 02 07 90 82 00 0f 01
00f0  55 aa 73 69 77 65 62 30  35 00
```

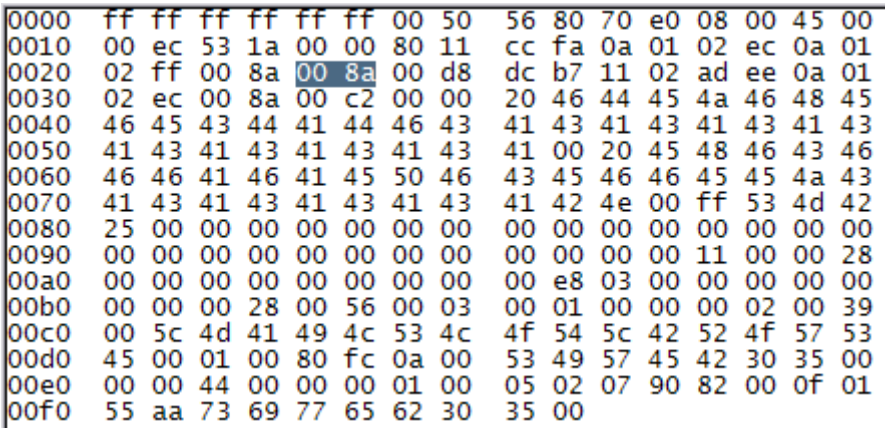
Header UDP

```
0000  ff ff ff ff ff ff 00 50  56 80 70 e0 08 00 45 00
0010  00 ec 53 1a 00 00 80 11  cc fa 0a 01 02 ec 0a 01
0020  02 ff 00 8a 00 8a 00 d8  dc b7 11 02 ad ee 0a 01
0030  02 ec 00 8a 00 c2 00 00  20 46 44 45 4a 46 48 45
0040  46 45 43 44 41 44 46 43  41 43 41 43 41 43 41 43
0050  41 43 41 43 41 43 41 43  41 00 20 45 48 46 43 46
0060  46 46 41 46 41 45 50 46  43 45 46 46 45 45 4a 43
0070  41 43 41 43 41 43 41 43  41 42 4e 00 ff 53 4d 42
0080  25 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090  00 00 00 00 00 00 00 00  00 00 00 00 11 00 00 28
00a0  00 00 00 00 00 00 00 00  00 e8 03 00 00 00 00 00
00b0  00 00 00 28 00 56 00 03  00 01 00 00 00 02 00 39
00c0  00 5c 4d 41 49 4c 53 4c  4f 54 5c 42 52 4f 57 53
00d0  45 00 01 00 80 fc 0a 00  53 49 57 45 42 30 35 00
00e0  00 00 44 00 00 00 01 00  05 02 07 90 82 00 0f 01
00f0  55 aa 73 69 77 65 62 30  35 00
```

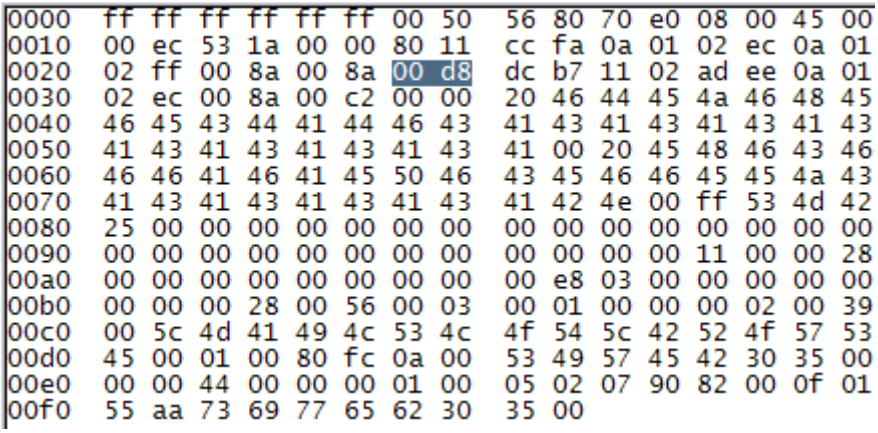
Source Port: 0x008A = 138 (netbios)

```
0000  ff ff ff ff ff ff 00 50  56 80 70 e0 08 00 45 00
0010  00 ec 53 1a 00 00 80 11  cc fa 0a 01 02 ec 0a 01
0020  02 ff 00 8a 00 8a 00 d8  dc b7 11 02 ad ee 0a 01
0030  02 ec 00 8a 00 c2 00 00  20 46 44 45 4a 46 48 45
0040  46 45 43 44 41 44 46 43  41 43 41 43 41 43 41 43
0050  41 43 41 43 41 43 41 43  41 00 20 45 48 46 43 46
0060  46 46 41 46 41 45 50 46  43 45 46 46 45 45 4a 43
0070  41 43 41 43 41 43 41 43  41 42 4e 00 ff 53 4d 42
0080  25 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0090  00 00 00 00 00 00 00 00  00 00 00 00 11 00 00 28
00a0  00 00 00 00 00 00 00 00  00 e8 03 00 00 00 00 00
00b0  00 00 00 28 00 56 00 03  00 01 00 00 00 02 00 39
00c0  00 5c 4d 41 49 4c 53 4c  4f 54 5c 42 52 4f 57 53
00d0  45 00 01 00 80 fc 0a 00  53 49 57 45 42 30 35 00
00e0  00 00 44 00 00 00 01 00  05 02 07 90 82 00 0f 01
00f0  55 aa 73 69 77 65 62 30  35 00
```

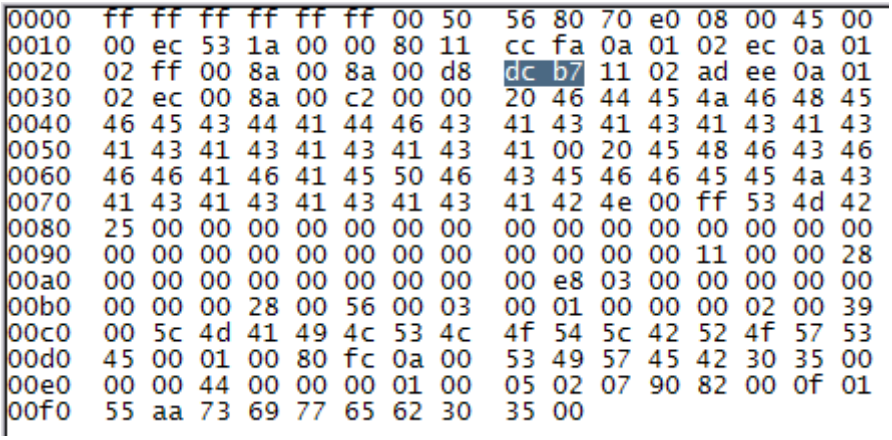
Destination Port: 0x008A = 138 (netbios)



Lunghezza: 0x00D8 = 216 bytes



Checksum: 0xDCB7



La parte successiva restante chiaramente rappresenta i dati veri e propri del datagramma UDP.

I PACCHETTI DHCP

Il DHCP, acronimo dall'inglese Dynamic Host Configuration Protocol (protocollo di configurazione dinamica degli indirizzi) è il protocollo usato per assegnare gli indirizzi IP ai calcolatori di una rete.
In una rete basata sul protocollo IP, ogni calcolatore ha bisogno di un indirizzo IP, scelto in modo tale che appartenga alla sottorete a cui è collegato e che sia univoco, ovvero che non ci siano altri calcolatori che stiano già usando quell'indirizzo.
Il compito di assegnare manualmente gli indirizzi IP ai calcolatori comporta un rilevante onere per gli amministratori di rete, soprattutto in reti di grandi dimensioni o in caso di numerosi computer che si connettono a rotazione solo ad ore o giorni determinati.
Il protocollo DHCP può essere usato anche per assegnare ai client diversi parametri necessari per il corretto funzionamento sulla rete a cui sono collegati. Tra i più comuni, oltre all'assegnazione dinamica dell'indirizzo IP, si possono citare: maschera di sottorete, default gateway, indirizzi dei server DNS, nome di dominio DNS di default, indirizzi dei server WINS, indirizzi dei server NTP ecc.
DHCP utilizza il protocollo UDP (e di conseguenza IP) e le porte registrate sono la 67 per il server e la 68 per il client.

FUNZIONAMENTO DEL PROTOCOLLO DHCP

Quando un calcolatore vuole ottenere un indirizzo tramite DHCP, attiva il processo DHCP client. In questo momento, il calcolatore non ha un indirizzo IP valido, quindi non può usare tutte le funzionalità della rete.
Esso invia quindi un pacchetto chiamato **DHCPDISCOVER** in broadcast sulla sottorete, con indirizzo IP sorgente messo convenzionalmente a 0.0.0.0, e destinazione 255.255.255.255. Tutti i server DHCP attivi sulla sottorete ricevono direttamente questo pacchetto, e possono rispondere (o meno) con un pacchetto di **DHCPOFFER**, in cui propongono un indirizzo IP al client. Questo pacchetto è indirizzato direttamente all'indirizzo di livello datalink del client (che non ha ancora un indirizzo IP), per cui può essere inviato solo da un server che si trovi sulla stessa sottorete.

Il client aspetta un certo tempo di ricevere una o più offerte, dopodiché ne seleziona una, ed invia un pacchetto di **DHCPREQUEST** al server che ha scelto. Questo gli conferma l'assegnazione dell'indirizzo con un pacchetto di **DHCPACK**.
A questo punto, il client è autorizzato ad usare l'indirizzo ricevuto per un tempo limitato, detto tempo di lease. Prima della scadenza, dovrà tentare di rinnovarlo inviando un nuovo pacchetto DHCPREQUEST al server, che gli risponderà con un DHCPACK se vuole prolungare l'assegnazione dell'indirizzo. Questi sono normali pacchetti IP scambiati tra due calcolatori che hanno indirizzi validi. Se il client non riesce a rinnovare l'indirizzo, tornerà allo stato iniziale (senza indirizzo assegnato – 0.0.0.0) cercando di farsene attribuire un altro.

CONTROLLI SUI PACCHETTI DHCP

In tale contesto, considerato il funzionamento del protocollo DHCP e le esigenze della rete del cliente (la quale sfrutta tali funzionalità DHCP), si sono resi necessari alcuni ulteriori controlli, al fine di consentire il transito di pacchetti DHCP atti ad assegnare l'indirizzo IP (ed altre configurazioni utili) ai client presenti in rete.

Premesso che non creeranno problemi i pacchetti di tipo DHCPREQUEST / DHCPACK, in quanto sarà sufficiente inserire l'indirizzo IP del server DHCP (o dei server se sono più di uno) tra gli indirizzi validi, nonché le porte UDP 67 e 68 tra quelle valide, i pacchetti di tipo DHCPDISCOVER / DHCPACK richiedono alcune considerazioni ulteriori.

Esisterà sicuramente un periodo di tempo (quando ancora il client non ha nessun indirizzo IP valido assegnato) in cui il client avrà indirizzo IP 0.0.0.0.
In tale intervallo di tempo, il client deve poter accettare offerte di indirizzi provenienti dai server DHCP presenti in rete.

Pertanto il controllo effettuato sarà il seguente:

```
if(ip_compare("0.0.0.0",localIP))
```

Come già specificato, verranno accettati pacchetti provenienti solo da indirizzi IP certificati:

```
if (white_ip(ip_temp))
```

Se, oltre a tali controlli (indirizzo IP attuale = 0.0.0.0 e IP sorgente certificato), il protocollo successivo ad IP sarà UDP:

```
IpProto = pIpHeader->ip_p;
```

```
if(IpProto == IPPROTO_UDP)
```

e la porta logica utilizzata sarà la 68:

```
pUdpHeader = (udphdr_ptr)(((PUCHAR)pIpHeader) + sizeof(DWORD)*pIpHeader-
>ip_hl);
dUdpPort = ntohs(pUdpHeader->th_dport);

if((dUdpPort == 68) && port_white(dUdpPort,"UDP"))
```

allora ci troveremo sicuramente in presenza di un pacchetto DHCPOFFER, il quale sarà inoltrato allo stack TCP/IP

```
api.SendPacketToMstcp(&Request);
```

ESEMPIO PRATICO DI FLUSSO DHCP

Time	Source	Destination	Protocol	Info
24.589989	0.0.0.0	255.255.255.255	DHCP	DHCP Discover
27.936767	10.1.2.249	10.1.2.70	DHCP	DHCP Offer
27.937106	0.0.0.0	255.255.255.255	DHCP	DHCP Request
27.940059	10.1.2.249	10.1.2.70	DHCP	DHCP ACK

1. inizialmente il client non avrà alcun indirizzo IP assegnato (0.0.0.0) ed invierà in broadcast (255.255.255.255) un pacchetto DHCP di tipo DHCP Discover;
2. il server DHCP (10.1.2.249) inoltrerà direttamente all'indirizzo di livello datalink (MAC ADDRESS) una offerta di indirizzo IP (10.1.2.70) – tipologia DHCP Offer;
3. il client (0.0.0.0) ancora senza un indirizzo IP valido, inoltrerà al server la richiesta di poter utilizzare l'indirizzo ricevuto;
4. il server confermerà l'indirizzo mediante un pacchetto DHCP Ack.

PACKET FRAGMENTATION

Ogni rete basata su trasmissione di pacchetti ha un grandezza MTU definita. L'MTU (Maximum Transmission Unit) è la dimensione del pacchetto di grandezza massima che può essere trasmesso in rete. Tale parametro è di solito associato alle interfacce di comunicazione quali schede di rete o porte seriali. Se un router deve trasmettere un pacchetto su una interfaccia che ha un MTU inferiore alla dimensione del pacchetto, il protocollo Internet effettua automaticamente la frammentazione, ovvero divide il pacchetto in due o più pacchetti più piccoli. I frammenti del pacchetto originale sono contrassegnati così il protocollo IP di destinazione è in grado di riassemblare i pacchetti nell'originale. Un qualsiasi router lungo il cammino potrebbe dover frammentare un pacchetto, e l'host di destinazione dovrà ricostruire il pacchetto originale dai frammenti.

Network	Standard MTU
Ethernet	1500 bytes
Token Ring	4096 bytes

Il concetto di MTU si riferisce alla dimensione del pacchetto IP. IP viene trasportato nella gran parte dei casi su Ethernet, che ha normalmente un carico pagante massimo di 1500 byte, a cui si aggiungono 14 byte di intestazione Ethernet. La MTU tipica su Internet è quindi di 1500 byte. Di questi, 20 sono normalmente occupati dall'header IP. Se viene utilizzato il protocollo TCP, questo occupa normalmente altri 20 byte per i propri header. Quindi la grandezza di un pacchetto TCP è tipicamente di 1460 byte.

I campo dell'header IP che sono coinvolti nel meccanismo della frammentazione sono Identificazione, Flags e Offset di Frammentazione.

Il campo Identificazione (16 bits) rapresenta un indice utilizzato durante il riassemlamento del datagramma.

Il campo Flags (3 bits) è composto da 3 flags di 1 bit ciascuno. Il significato è il seguente:
Bit 0: riservato, deve essere 0
Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.
Il campo Offset di Frammentazione (13 bits) indica dove all'interno del datagramma il frammento deve essere posizionato.
Tale offset è misurato in unità di 8 ottetti (64 bits); il primo frammento ha offset 0.

CONTROLLI SUI PACCHETTI FRAMMENTATI

La rete del cliente utilizza, in alcuni casi particolari, pacchetti che possono essere di grandezza superiore a 1500 bytes e quindi potenzialmente possono essere frammentati.

Tutti i controlli indicati in precedenza vengono eseguiti solo su pacchetti non frammentati (esempio **if(IpProto == IPPROTO_TCP && !frammento(pIpHeader->ip_off))**) ed anche sul primo pacchetto di una serie di frammenti (il primo pacchetto

rispecchia la disposizione dei campi del protocollo) perché non avrebbe significato effettuare controlli sui bytes che compongono un pacchetto frammentato (successivo al primo) in quando essi non rappresenterebbero effettivamente i campi teorici bensì il campo dati di un pacchetto più grande.

I pacchetti frammentati successivi al primo vengono inoltrati per convenzione allo stack TCP/IP. Se il primo pacchetto della serie non supererà i controlli e non sarà inoltrato allo stack allora tutti i pacchetti successivi saranno automaticamente scartati anche se raggiungeranno comunque il TCP/IP stack.

```
if (frammento(pIpHeader->ip_off))
{
api.SendPacketToMstcp(&Request);
}
```

La funzione che verifica la frammentazione dei pacchetti è così definita:

```
bool frammento(short offset)
{
        …
}
```

In particolare, riceverà in ingresso il valore offset (di tipo short cioè 2 bytes – 16 bits) che rappresenta 3 bits di Flags e 13 bits di offset vero e proprio.

Pertanto, una volta convertito il valore da network-order a host-order

```
offset = (offset>>8)&0x00FF | (offset<<8)&0xFF00;
```

mf_bit avrà un valore diverso da 0 se il bit more fragment del pacchetto sarà settato a 1.

```
short mf_bit, off_bit;
```

```
mf_bit = offset & IP_MF;
```

IP_MF è così definito:

```
#define        IP_MF 0x2000                /* more fragments flag */
```

Il valore vero e proprio dell'offset del pacchetto viene calcolato eseguendo una operazione logica **AND** con il valore **0x1FFF (0001 1111 1111 1111)** e cioè esattamente considerando solo i 13 bit relativi. Tale valore deve essere moltiplicato per 8 se si vuole ottenere il valore esatto.

```
off_bit = (offset & 0x1FFF) * 8;
```

La funzione ritornerà TRUE se il pacchetto è frammentato ed il frammento non è il primo di una serie di frammenti. Altrimenti ritornerà FALSE:

```
if (off_bit != 0x0000)

        return TRUE;

else

        return FALSE;
```

ESEMPIO DI FRAMMENTAZIONE

Primo pacchetto:

```
0000   00 02 3f d2 ea 66 00 0f   b0 59 d1 56 08 00 45 00
0010   05 dc 9d 64 20 00 80 11   5f 1b 0a 01 02 4a 0a 01
0020   02 46 06 cc 30 39 75 38   c5 e6 00 00 00 00 00 00
0030   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0040   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0050   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
```

```
Flags: 0x02 (More Fragments)
  0... = Reserved bit: Not set
  .0.. = Don't fragment: Not set
  ..1. = More fragments: Set
Fragment offset: 0
```

Rappresentazione binaria 0x2000 = 0010 0000 0000 0000

Pacchetto successivo

```
0000  00 02 3f d2 ea 66 00 0f  b0 59 d1 56 08 00 45 00
0010  05 dc 9d 64 20 b9 80 11  5e 62 0a 01 02 4a 0a 01
0020  02 46 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

```
Flags: 0x02 (More Fragments)
  0... = Reserved bit: Not set
  .0.. = Don't fragment: Not set
  ..1. = More fragments: Set
Fragment offset: 1480
```

Rappresentazione binaria 0x20B9 = 0010 0000 1011 1001

Offset (13 bits): 0000010111001 = 185 ottetti * 8 = 1480 bytes

PROCEDURA DI INSTALLAZIONE

Al fine di favorire la “messa in produzione” del software di packet e port filtering sui client Windows 98, è stata predisposta una distribuzione che comprende i seguenti files:

- readme.txt (file informativo)
- setup.bat (file di installazione)
- Packet_filtering.exe (l’eseguibile del programma)
- ndisrd.vxd (packet filtering driver)
- ndisapi.dll (libreria che fornisce le chiamate alle funzioni del driver)
- ndisrd.reg (chiavi di registro utilizzate dal setup)
- config.cfg (file di configurazione)
- Packet_filtering.lnk (shortcut utilizzata per far avviare il filtro allo startup)
- NetList.exe (tool che elenca le interfacce di rete presenti con relativi parametri)

Sarà sufficiente copiare i files in una qualsiasi directory del client ed eseguire il file setup.bat.
Tale file esegue i seguenti steps:

```
deltree /y C:\PF
md C:\PF
copy NDISAPI.DLL C:\PF
copy PACKET_FILTERING.EXE C:\PF
copy CONFIG.CFG C:\PF
copy NDISRD.VXD C:\WINDOWS\SYSTEM
regedit /s NDISRD.REG
copy NETLIST.EXE C:\PF
copy PACKET_FILTERING.LNK C:\WINDOWS\STARTM~1\Programs\StartUp
RUNDLL SHELL.DLL,RestartDialog
@cls
@exit
```

In particolare, il software viene installato nel percorso c:\PF e viene registrata la chiave di registro presente nel file ndisrd.reg. Viene anche inserita una shortcut nei programmi che partono automaticamente all'avvio del sistema. Il sistema viene anche riavviato per terminare e completare l'installazione.

Il file **NDISRD.REG** contiene il seguente codice:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\ndisrd]
"StaticVxD"="ndisrd.vxd"

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\ndisrd\Parameters]
```

CONCLUSIONI FINALI

L'obbiettivo della tesi è stato lo sviluppo di una framework software con funzionalità di packet e port filtering.

Tale progetto mi ha permesso di capire e sperimentare il funzionamento delle reti basate su protocolli di rete TCP/IP, entrando anche nel dettaglio della composizione dei pacchetti e del significato di ogni singolo bit in transito sulla rete.

Tale argomento solitamente viene trattato a livello didattico in maniera abbastanza teorica ed avere la possibilità di comprendere appieno i meccanismi di funzionamento di una rete di tale dimensione mi ha particolarmente gratificato.

Inoltre, nonostante la suite di protocolli utilizzati dalle reti più comuni (e chiaramente anche da internet) non sia di recente invenzione, il settore della sicurezza delle reti nel campo informatico è di assoluta attualità.

Il software prodotto, oltre ad essere attualmente utilizzato nella rete del cliente, sicuramente potrà subire modifiche ed ampliamenti in futuro in quanto è stato prodotto in maniera schematica, con possibilità di estendere i controlli ad ulteriori tipologie di protocolli dovessero essere utilizzati nella rete del cliente e protocolli di nuova implementazione (Es. Ipv6).

L'obbiettivo della tesi è stato quindi raggiunto ed inoltre la partecipazione a tale progetto mi ha fatto comprendere e riflettere sulla figura dell'informatico (inteso come programmatore, sistemista ecc.) e sulle possibilità per il mio futuro, avendo dovuto sviluppare del codice ma utilizzando nozioni sistemistiche e di networking.

RINGRAZIAMENTI

In merito al percorso universitario...

Ai miei genitori ed a mia nonna

Per avermi spronato in questi anni

Alla mia ragazza

per avermi sopportato e sostenuto nei momenti difficili….

In merito a questa tesi…

Alla professoressa Elisabetta Binaghi (Relatore)

Per la disponibilità mostratami

A Riccardo Rovelli (Gruppo Reti s.p.a.)

Per l'opportunità di sviluppare la tesi in azienda

BIBLIOGRAFIA

Nt Kernel Resources
http://www.ntkernel.com/

Wireshark network protocol analyzer
http://www.wireshark.org/

Internet Assigned Numbers Authority
http://www.iana.org/assignments/port-numbers
http://www.iana.org/assignments/ethernet-numbers

Wikipedia, the free encyclopedia
http://www.wikipedia.org

PC Tools Software
http://www.pctools.com/guides/registry/detail/280/

Expedient
http://help.expedient.com/broadband/mtu.shtml

Supporto tecnico Microsoft
http://support.microsoft.com/default.aspx

Tech FAQ
http://www.tech-faq.com/packet-fragmentation.shtml
http://www.tech-faq.com/mtu.shtml

Webopedia, computer and Internet technology definitions
http://www.webopedia.com/TERM/b/big_endian.html

Network Sourcery
http://www.networksorcery.com

Microsoft Development Network
http://msdn.microsoft.com

The code project
http://www.codeproject.com

Dejanews Google Groups
http://www.deja.com

ALLEGATO 1 - http://www.iana.org/assignments/ethernet-numbers

ETHER TYPES

(last updated 2007-03-29)

Many of the networks of all classes are Ethernets (10Mb) or
Experimental Ethernets (3Mb). These systems use a message "type"
field in much the same way the ARPANET uses the "link" field.

If you need an Ether Type, contact:

IEEE Registration Authority
IEEE Standards Department
445 Hoes Lane
Piscataway, NJ 08854
Phone +1 732 562 3813
Fax: +1 732 562 1571
Email: <ieee-registration-authority&ieee.org>
http://standards.ieee.org/regauth/index.html

The following list of EtherTypes is contributed unverified information
from various sources. Another list of EtherTypes is maintained by
Michael A. Patton and is accessible at:

<URL:http://www.cavebear.com/CaveBear/Ethernet/>
<URL:ftp://ftp.cavebear.com/pub/Ethernet-codes>

Assignments:

Ethernet		Exp. Ethernet		Description	References
decimal	Hex	decimal	octal		
0000	0000-05DC	-	-	IEEE802.3 Length Field	[XEROX]
0257	0101-01FF	-	-	Experimental	[XEROX]
0512	0200	512	1000	XEROX PUP (see 0A00)	[8,XEROX]
0513	0201	-	-	PUP Addr Trans (see 0A01)	[XEROX]
	0400			Nixdorf	[XEROX]
1536	0600	1536	3000	XEROX NS IDP	[133,XEROX]
	0660			DLOG	[XEROX]
	0661			DLOG	[XEROX]
2048	0800	513	1001	Internet IP (IPv4)	[IANA]
2049	0801	-	-	X.75 Internet	[XEROX]
2050	0802	-	-	NBS Internet	[XEROX]
2051	0803	-	-	ECMA Internet	[XEROX]
2052	0804	-	-	Chaosnet	[XEROX]
2053	0805	-	-	X.25 Level 3	[XEROX]
2054	0806	-	-	ARP	[IANA]
2055	0807	-	-	XNS Compatability	[XEROX]
2056	0808	-	-	Frame Relay ARP	[RFC1701]
2076	081C	-	-	Symbolics Private	[DCP1]
2184	0888-088A	-	-	Xyplex	[XEROX]
2304	0900	-	-	Ungermann-Bass net debugr	[XEROX]
2560	0A00	-	-	Xerox IEEE802.3 PUP	[XEROX]
2561	0A01	-	-	PUP Addr Trans	[XEROX]
2989	0BAD	-	-	Banyan VINES	[XEROX]
2990	0BAE	-	-	VINES Loopback	[RFC1701]
2991	0BAF	-	-	VINES Echo	[RFC1701]

4096	1000	-	-	Berkeley Trailer nego	[XEROX]
4097	1001-100F	-	-	Berkeley Trailer encap/IP	[XEROX]
5632	1600	-	-	Valid Systems	[XEROX]
16962	4242	-	-	PCS Basic Block Protocol	[XEROX]
21000	5208	-	-	BBN Simnet	[XEROX]
24576	6000	-	-	DEC Unassigned (Exp.)	[XEROX]
24577	6001	-	-	DEC MOP Dump/Load	[XEROX]
24578	6002	-	-	DEC MOP Remote Console	[XEROX]
24579	6003	-	-	DEC DECNET Phase IV Route	[XEROX]
24580	6004	-	-	DEC LAT	[XEROX]
24581	6005	-	-	DEC Diagnostic Protocol	[XEROX]
24582	6006	-	-	DEC Customer Protocol	[XEROX]
24583	6007	-	-	DEC LAVC, SCA	[XEROX]
24584	6008-6009	-	-	DEC Unassigned	[XEROX]
24586	6010-6014	-	-	3Com Corporation	[XEROX]
25944	6558	-	-	Trans Ether Bridging	[RFC1701]
25945	6559	-	-	Raw Frame Relay	[RFC1701]
28672	7000	-	-	Ungermann-Bass download	[XEROX]
28674	7002	-	-	Ungermann-Bass dia/loop	[XEROX]
28704	7020-7029	-	-	LRT	[XEROX]
28720	7030	-	-	Proteon	[XEROX]
28724	7034	-	-	Cabletron	[XEROX]
32771	8003	-	-	Cronus VLN	[131,DT15]
32772	8004	-	-	Cronus Direct	[131,DT15]
32773	8005	-	-	HP Probe	[XEROX]
32774	8006	-	-	Nestar	[XEROX]
32776	8008	-	-	AT&T	[XEROX]
32784	8010	-	-	Excelan	[XEROX]
32787	8013	-	-	SGI diagnostics	[AXC]
32788	8014	-	-	SGI network games	[AXC]
32789	8015	-	-	SGI reserved	[AXC]
32790	8016	-	-	SGI bounce server	[AXC]
32793	8019	-	-	Apollo Domain	[XEROX]
32815	802E	-	-	Tymshare	[XEROX]
32816	802F	-	-	Tigan, Inc.	[XEROX]
32821	8035	-	-	Reverse ARP	[48,JXM]
32822	8036	-	-	Aeonic Systems	[XEROX]
32824	8038	-	-	DEC LANBridge	[XEROX]
32825	8039-803C	-	-	DEC Unassigned	[XEROX]
32829	803D	-	-	DEC Ethernet Encryption	[XEROX]
32830	803E	-	-	DEC Unassigned	[XEROX]
32831	803F	-	-	DEC LAN Traffic Monitor	[XEROX]
32832	8040-8042	-	-	DEC Unassigned	[XEROX]
32836	8044	-	-	Planning Research Corp.	[XEROX]
32838	8046	-	-	AT&T	[XEROX]
32839	8047	-	-	AT&T	[XEROX]
32841	8049	-	-	ExperData	[XEROX]
32859	805B	-	-	Stanford V Kernel exp.	[XEROX]
32860	805C	-	-	Stanford V Kernel prod.	[XEROX]
32861	805D	-	-	Evans & Sutherland	[XEROX]
32864	8060	-	-	Little Machines	[XEROX]
32866	8062	-	-	Counterpoint Computers	[XEROX]
32869	8065	-	-	Univ. of Mass. @ Amherst	[XEROX]
32870	8066	-	-	Univ. of Mass. @ Amherst	[XEROX]
32871	8067	-	-	Veeco Integrated Auto.	[XEROX]
32872	8068	-	-	General Dynamics	[XEROX]
32873	8069	-	-	AT&T	[XEROX]
32874	806A	-	-	Autophon	[XEROX]
32876	806C	-	-	ComDesign	[XEROX]
32877	806D	-	-	Computgraphic Corp.	[XEROX]
32878	806E-8077	-	-	Landmark Graphics Corp.	[XEROX]
32890	807A	-	-	Matra	[XEROX]
32891	807B	-	-	Dansk Data Elektronik	[XEROX]

32892	807C	-	-	Merit Internodal	[HWB]
32893	807D-807F	-	-	Vitalink Communications	[XEROX]
32896	8080	-	-	Vitalink TransLAN III	[XEROX]
32897	8081-8083	-	-	Counterpoint Computers	[XEROX]
32923	809B	-	-	Appletalk	[XEROX]
32924	809C-809E	-	-	Datability	[XEROX]
32927	809F	-	-	Spider Systems Ltd.	[XEROX]
32931	80A3	-	-	Nixdorf Computers	[XEROX]
32932	80A4-80B3	-	-	Siemens Gammasonics Inc.	[XEROX]
32960	80C0-80C3	-	-	DCA Data Exchange Cluster	[XEROX]
32964	80C4	-	-	Banyan Systems	[XEROX]
32965	80C5	-	-	Banyan Systems	[XEROX]
32966	80C6	-	-	Pacer Software	[XEROX]
32967	80C7	-	-	Applitek Corporation	[XEROX]
32968	80C8-80CC	-	-	Intergraph Corporation	[XEROX]
32973	80CD-80CE	-	-	Harris Corporation	[XEROX]
32975	80CF-80D2	-	-	Taylor Instrument	[XEROX]
32979	80D3-80D4	-	-	Rosemount Corporation	[XEROX]
32981	80D5	-	-	IBM SNA Service on Ether	[XEROX]
32989	80DD	-	-	Varian Associates	[XEROX]
32990	80DE-80DF	-	-	Integrated Solutions TRFS	[XEROX]
32992	80E0-80E3	-	-	Allen-Bradley	[XEROX]
32996	80E4-80F0	-	-	Datability	[XEROX]
33010	80F2	-	-	Retix	[XEROX]
33011	80F3	-	-	AppleTalk AARP (Kinetics)	[XEROX]
33012	80F4-80F5	-	-	Kinetics	[XEROX]
33015	80F7	-	-	Apollo Computer	[XEROX]
33023	80FF-8103	-	-	Wellfleet Communications	[XEROX]
33031	8107-8109	-	-	Symbolics Private	[XEROX]
33072	8130	-	-	Hayes Microcomputers	[XEROX]
33073	8131	-	-	VG Laboratory Systems	[XEROX]
33074	8132-8136			Bridge Communications	[XEROX]
33079	8137-8138	-	-	Novell, Inc.	[XEROX]
33081	8139-813D	-	-	KTI	[XEROX]
	8148			Logicraft	[XEROX]
	8149			Network Computing Devices	[XEROX]
	814A			Alpha Micro	[XEROX]
33100	814C	-	-	SNMP	[JKR1]
	814D			BIIN	[XEROX]
	814E			BIIN	[XEROX]
	814F			Technically Elite Concept	[XEROX]
	8150			Rational Corp	[XEROX]
	8151-8153			Qualcomm	[XEROX]
	815C-815E			Computer Protocol Pty Ltd	[XEROX]
	8164-8166			Charles River Data System	[XEROX]
	817D			XTP	[XEROX]
	817E			SGI/Time Warner prop.	[XEROX]
	8180			HIPPI-FP encapsulation	[XEROX]
	8181			STP, HIPPI-ST	[XEROX]
	8182			Reserved for HIPPI-6400	[XEROX]
	8183			Reserved for HIPPI-6400	[XEROX]
	8184-818C			Silicon Graphics prop.	[XEROX]
	818D			Motorola Computer	[XEROX]
	819A-81A3			Qualcomm	[XEROX]
	81A4			ARAI Bunkichi	[XEROX]
	81A5-81AE			RAD Network Devices	[XEROX]
	81B7-81B9			Xyplex	[XEROX]
	81CC-81D5			Apricot Computers	[XEROX]
	81D6-81DD			Artisoft	[XEROX]
	81E6-81EF			Polygon	[XEROX]
	81F0-81F2			Comsat Labs	[XEROX]
	81F3-81F5			SAIC	[XEROX]
	81F6-81F8			VG Analytical	[XEROX]

	8203-8205			Quantum Software	[XEROX]
	8221-8222			Ascom Banking Systems	[XEROX]
	823E-8240			Advanced Encryption Syste	[XEROX]
	827F-8282			Athena Programming	[XEROX]
	8263-826A			Charles River Data System	[XEROX]
	829A-829B			Inst Ind Info Tech	[XEROX]
	829C-82AB			Taurus Controls	[XEROX]
	82AC-8693			Walker Richer & Quinn	[XEROX]
	8694-869D			Idea Courier	[XEROX]
	869E-86A1			Computer Network Tech	[XEROX]
	86A3-86AC			Gateway Communications	[XEROX]
	86DB			SECTRA	[XEROX]
	86DE			Delta Controls	[XEROX]
	86DD			IPv6	[IANA]
34543	86DF	-	-	ATOMIC	[Postel]
	86E0-86EF			Landis & Gyr Powers	[XEROX]
	8700-8710			Motorola	[XEROX]
34667	876B	-	-	TCP/IP Compression	[RFC1144]
34668	876C	-	-	IP Autonomous Systems	[RFC1701]
34669	876D	-	-	Secure Data	[RFC1701]
	880B			PPP	[IANA]
	8847			MPLS Unicast	[Rosen]
	8848			MPLS Multicast	[Rosen]
	8A96-8A97			Invisible Software	[XEROX]
34915	8863	-	-	PPPoE Discovery Stage	[RFC2516]
34915	8864	-	-	PPPoE Session Stage	[RFC2516]
36864	9000	-	-	Loopback	[XEROX]
36865	9001	-	-	3Com(Bridge) XNS Sys Mgmt	[XEROX]
36866	9002	-	-	3Com(Bridge) TCP-IP Sys	[XEROX]
36867	9003	-	-	3Com(Bridge) loop detect	[XEROX]
65280	FF00	-	-	BBN VITAL-LanBridge cache	[XEROX]
	FF00-FF0F			ISC Bunker Ramo	[XEROX]
65535	FFFF	-	-	Reserved	[RFC1701]

The standard for transmission of IP datagrams over Ethernets and Experimental Ethernets is specified in [RFC894] and [RFC895] respectively.

NOTE: Ethernet 48-bit address blocks are assigned by the IEEE.

IEEE Registration Authority
c/o Iris Ringel
IEEE Standards Department
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331
Phone +1 732 562 3813
Fax: +1 732 562 1571
Email: <i.ringel&ieee.org>