MMS Ether-Real Network Analyzer

The Skunk Works from

Herbert Falk

The concept of an Open Source based MMS analyzer has always been intriguing, but until recently the idea lacked the open source technology. During several meetings, people asked if something open source could be created in order to facilitate adoption of MMS based protocols (e.g. ICCP/TASE.2, UCA, and IEC 61850). Initially when I started thinking about such an analyzer, 4 years ago there was one low cost solution available, however it proved difficult to extend (or maybe I just wasn't appropriately motivated). I stumbled upon Ethereal (www.ethereal.com) a couple of years ago and have used it to capture network packets since then.

The resurrection of the MMS Ethernet analyzer came during an August 2003 Interop test of some security extensions for ICCP. We used Ethereal to analyze SSL/TLS transactions and accidentally discovered that Ethereal could analyze RFC-1006/TP0 and CLNP/TP4 traffic. At that time, it became evident that Ethereal represented a platform that could be extended, it was open source, an observer asked if it was extensible. Thus the SkunkWorks project began.

Why SkunkWorks? Because the Ethereal analyzer is technology/code contributed by multiple people. My work, performed after business hours, was to add the packet decoding capabilities required for ICCP/TASE.2, UCA, and IEC 61850. The result is what you currently have.

Although not fully tested, it is fully functional. As with most open source projects, there are querks that must be contended with and these are documented in the following sections. But to summarize the state of affairs:

> MMS support for OSI and TCP profiles is implemented. Certain services have not been implemented or have been implemented and have not been tested. The decoding is appropriate for UCA, ICCP/TASE.2, and IEC 61850 profiles that utilize MMS.
>
> IEC 61850 GSSE/ UCA GOOSE Support. This is implemented and has been fully tested.
>
> IEC 61850 GOOSE: This is implemented and has been fully tested.
>
> IEC 61850-9-2 SMV: This is implemented and has been partially tested.
>
> IEC 61850 GSE Management: This is implemented, but has not been tested (no traces available).

<u>Problem Reporting</u>

In order to further refine and test the analyzer, I hope that you use the analyzer and report to me (herb@sisconet.com). Please use [Analyzer]: in the subject field so that I can differentiate you email from SPAM.

If there are problems, please supply a capture file, description of the problem, and the "packet number"(s) that are exhibiting the problem. Additionally, return email address and contact information are requested.

Since this is a home project, responses will not be immediate, but hopefully timely.

MMS Decoder in Ethereal

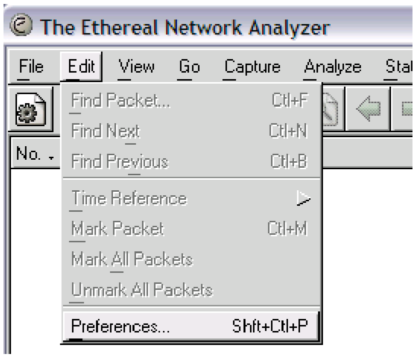
The MMS decoder supports: OSI Transport (tested) and TCP/IP (Tested).

The TCP/IP profile currently supports the display of SSL/TLS, but not the ACSE Security parameters.

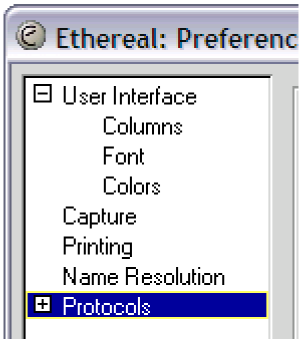
An oddity in the decoder is that the Initiate Request/Response will appear as the PRES (presentation) protocol.

In order to have the decoder work properly, you must select that the COTP (connection oriented Transport protocol) perform reassembly.

1). Select Edit/Preferences

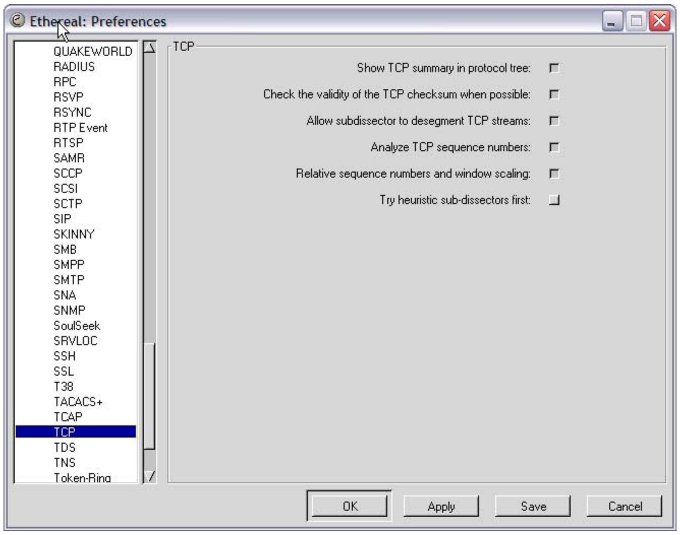


2). Expand Protocols


Ethereal: Preferenc
User Interface
Columns
Font
Colors
Capture
Printing
Name Resolution
Protocols

3). Select Reassembly

For TCP:



TCP

Show TCP summary in protocol tree: □

Check the validity of the TCP checksum when possible: □

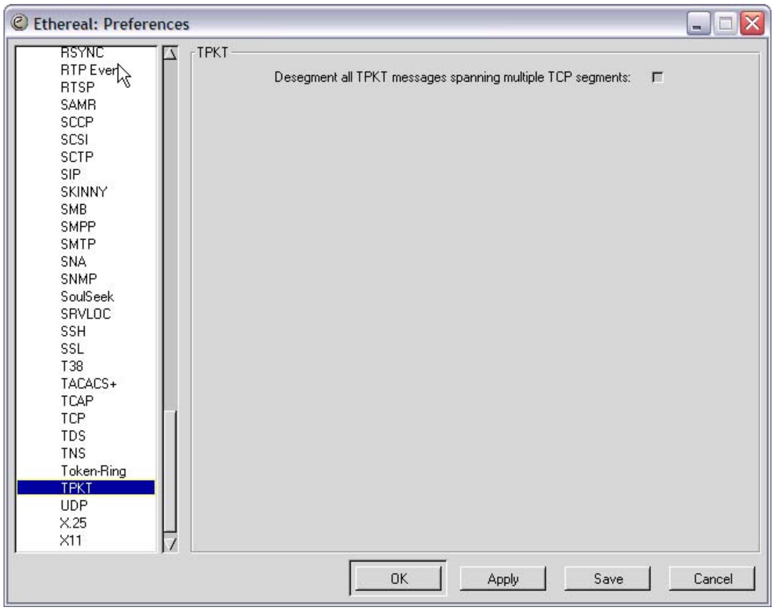
Allow subdissector to desegment TCP streams: □

Analyze TCP sequence numbers: □

Relative sequence numbers and window scaling: □

Try heuristic sub-dissectors first: □

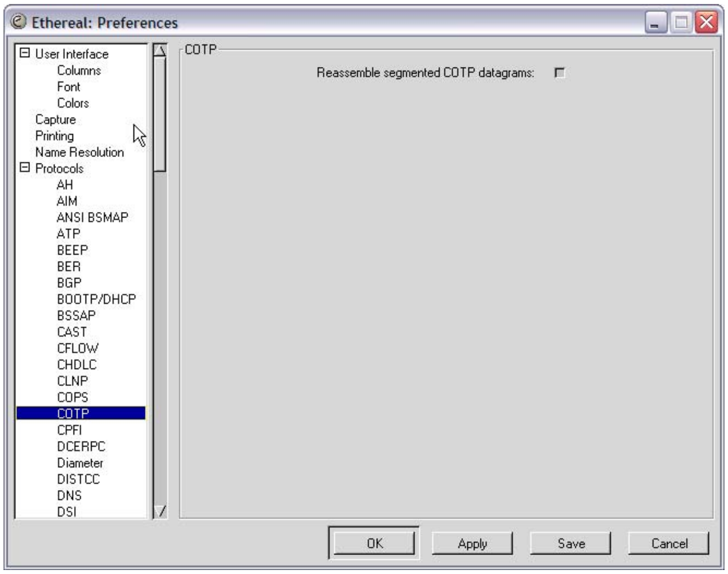
For TPKT:



TPKT

Desegment all TPKT messages spanning multiple TCP segments: □

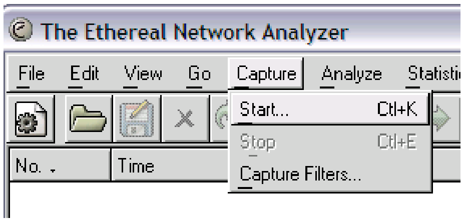
For COTP:



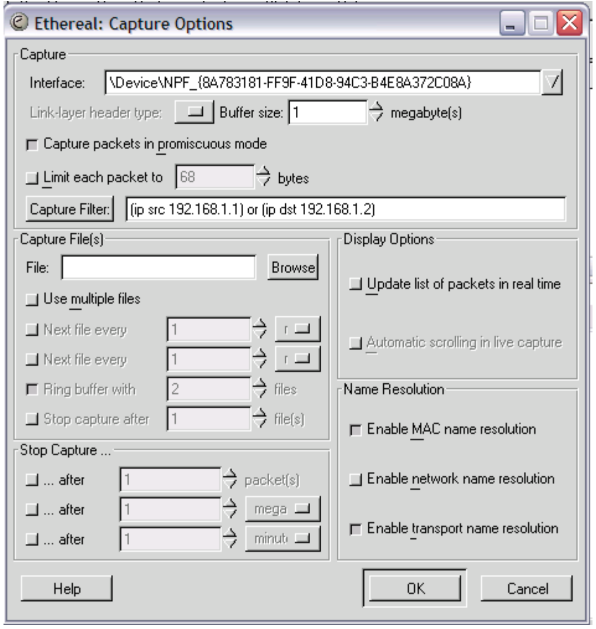
Reassemble segmented COTP datagrams: ☐

Setting up Capture Filters

Step 1:



Step 2: Create a capture string (this can be done on the capture filter line or by creating a permanent filter).


Ethereal: Capture Options

Capture

Interface: \Device\NPF_{8A783181-FF9F-41D8-94C3-B4E8A372C08A}

Link-layer header type: Buffer size: 1 megabyte(s)

☑ Capture packets in promiscuous mode

□ Limit each packet to 68 bytes

Capture Filter: (ip src 192.168.1.1) or (ip dst 192.168.1.2)

Capture File(s)

File:

□ Use multiple files

□ Next file every 1

□ Next file every 1

☑ Ring buffer with 2 files

□ Stop capture after 1 file(s)

Stop Capture ...

□ ... after 1 packet(s)

□ ... after 1 mega

□ ... after 1 minut

Display Options

□ Update list of packets in real time

□ Automatic scrolling in live capture

Name Resolution

☑ Enable MAC name resolution

□ Enable network name resolution

☑ Enable transport name resolution

Some suggested filters:

To capture all traffic to/from a address:

Host <ipAddress>

To capture traffic between two nodes:
“
((Ip src <ipAddress>) and (ip dst <ipAddress1>))
or
((Ip src <ipAddress1>) and (ip dst <ipAddress>))
“

Another suggestion is to select “Update list of packets in real time”. This will show the packets as they arrive.

You can also use a filter that is host address based (e.g. host <ip address>) that will capture all packets inbound and outbound from the specified address.

MMS Service Decoding and Testing

Service Implementation and Tested Matrix					
	Request		Response		
Service	Decoded	Tested	Decoded	Tested	
Initiate	X	X	X	X	
Conclude	X	X	X	X	
Abort	X	X	X	X	
Status	X	X	X	X	
GetNameList	X	X	X	X	
Identify	X	X	X	X	
Rename					
Read	X	X	X	X	
Write	X	X	X	X	
GetVariableAccessAttributes	X	X	X	X	
DefineNamedVariable	X		X		
DefineScatteredAccess					
ScatteredAccessAttributes					

Service Implementation and Tested Matrix					
	Request		Response		
Service	Decoded	Tested	Decoded	Tested	
DeleteVariableAccess	X		X		
DefineNamedVariableList	X	X	X	X	
GetNamedVariableListAttributes	X		X		
DeleteNamedVariableList	X		X		
DefineNamedType			X		
GetNamedTypeAttributes					
DeleteNamedType			X		
Input	X		X		
Output	X		X		
TakeControl					
RelinquishControl			X		
DefineSemaphore					
DeleteSemaphore					
ReportSemaphoreStatus					
ReportPoolSemaphoreStatus					
ReportSemaphoreEntryStatus					
DownloadSegment	X				
TerminateDownloadSequence					
InitiateUploadSequence					
UploadSegment	X				
TerminateUploadSegment	X				
RequestDomainDownload					
RequestDomainUpload					
LoadDomainContent					
StoreDomainContent					
DeleteDomain					
GetDomainAttributes	X	X	X	X	
CreateProgramInvocation	X		X		
DeleteProgramInvocation	X		X		
Start	X		X		
Stop	X		X		
Resume	X		X		
Reset	X		X		
Kill	X		X		
GetProgramInvocationAttributes	X		X		
ObtainFile	X		X		
DefinEventCondition			X		
DeleteEventCondition			X		
GetEventConditionAttributes					
ReportEventConditionStatus					
AlterEventConditionMonitoring			X		
TriggerEvent			X		
DefineEventAction			X		
DeleteEventAction			X		
GetEventActionAttributes					
ReportEventActionStatus			X		
DefineEventEnrollment			X		
DeleteEventEnrollment			X		
ReportEventEnrollmentStatus					
GetEventEnrollmentAttributes					

Service Implementation and Tested Matrix					
	Request		Response		
Service	Decoded	Tested	Decoded	Tested	
AcknowledgeEventNotification			X		
GetAlarmSummary					
GetAlarmEnrollmentSummary					
ReadJournal	X	Partial	X	Partial	
WriteJournal	X		X		
InitializeJournal	X		X		
ReportJournalStatus	X	Partial	X	Partial	
CreateJournal			X		
DeleteJournal			X		
GetCapabilityList	X	X	X	X	
FileOpen	X	X	X	X	
FileRead	X	X	X	X	
FileClose	X	X	X	X	
FileRename	X		X		
FileDirectory	X	X	X	X	

IEC GOOSE Decoder

There is an IEC GOOSE decoder in the package. It is suggested that the Ethernet protocol capture filter be used:

ether proto *protocol where protocol should be 0x88b8*

Additionally, the multicast filter could be used.

IEC GSSE Decoder

Use the multicast capture for this protocol. And then use the Filter capability in Ethereal.

IEC SMV Decoder

ether proto *protocol where protocol should be 0x88bA*

Use the multicast capture for this protocol. And then use the Filter capability in Ethereal.

IEC GSE Decoder

Use the multicast capture for this protocol. And then use the Filter capability in Ethereal. It is also possible to use the **ether proto** *protocol where protocol should be 0x88b9.*

Capture Instructions (from www.tcpdump.org)

expression

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

ber) preceded by one or more qualifiers. There are three different kinds of qualifier:

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

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

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

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

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

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

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

Allowable primitives are:

dst host *host*

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

src host *host*

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

host *host*

True if either the IPv4/v6 source or destination of the packet is *host*. Any of the above host expressions can be prepended with the keywords, **ip**, **arp**, **rarp**, or **ip6** as in:

ip host *host*

which is equivalent to:

ether proto *\ip* **and host** *host*

If *host* is a name with multiple IP addresses, each address will be checked for a match.

ether dst *ehost*

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

ether src *ehost*

True if the ethernet source address is *ehost*.

ether host *ehost*

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

gateway *host*

True if the packet used *host* as a gateway. I.e., the ethernet source or destination address was *host* but neither the IP source nor the IP destination was *host*. *Host* must be a name and must be found both by the machine's host-name-to-IP-address resolution mechanisms (host name file, DNS, NIS, etc.) etc.). (An equivalent expression is

ether host *ehost* **and not host** *host*

which can be used with either names or numbers for *host / ehost*.) This syntax does not work in IPv6-enabled configuration at this moment.

dst net *net*

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

src net *net*

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

net *net*

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

net *net* **mask** *netmask*

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

net *net*/*len*

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

dst port *port*

True if the packet is ip/tcp, ip/udp, ip6/tcp or ip6/udp and has a destination port value of *port*. The *port* can be a number or a name used in /etc/services (see **tcp(4P)** and **udp(4P)**). If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port number is checked (e.g., **dst port 513** will print both tcp/login traffic and udp/who traffic, and **port domain** will print both tcp/domain and udp/domain traffic).

src port *port*

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

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

tcp src port *port*

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

less *length*

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

len <= *length*.

greater *length*

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

len >= *length*.

ip proto *protocol*

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

ip6 proto *protocol*

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

ip6 protochain *protocol*

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

```
ip6 protochain 6
```

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

ip protochain *protocol*

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

ip broadcast

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

ether multicast

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

ip multicast

True if the packet is an IP multicast packet.

ip6 multicast

True if the packet is an IPv6 multicast packet.

ether proto *protocol*

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

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

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

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

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

decnet src *host*

True if the DECNET source address is *host*, which may be an address of the form ``10.123'', or a DECNET host name. [DECNET host name support is only available on

Ultrix systems that are configured to run DECNET.]

decnet dst *host*

True if the DECNET destination address is *host*.

decnet host *host*

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

ip, **ip6**, **arp**, **rarp**, **atalk**, **aarp**, **decnet**, **iso**, **stp**, **ipx**, *netbeui*

Abbreviations for:

ether proto *p*

where *p* is one of the above protocols.

lat, **moprc**, **mopdl**

Abbreviations for:

ether proto *p*

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

vlan *[vlan_id]*

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

tcp, **udp**, **icmp**

Abbreviations for:

ip proto *p* **or ip6 proto** *p*

where *p* is one of the above protocols.

iso proto *protocol*

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

clnp, esis, isis

Abbreviations for:

iso proto *p*

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

expr relop expr

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

proto [*expr* : *size*]

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

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

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

The following ICMP type field values are available: **icmp-echoreply**, **icmp-unreach**, **icmp-sourcequench**, **icmp-redirect**, **icmp-echo**, **icmp-routeradvert**, **icmp-routersolicit**, **icmp-timxceed**, **icmp-paramprob**, **icmp-tstamp**, **icmp-tstampreply**, **icmp-ireq**, **icmp-ireqreply**, **icmp-maskreq**, **icmp-maskreply**.

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

Primitives may be combined using:

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

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

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

Alternation (`||' or `or').

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

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

```
not host vs and ace
```

is short for

```
not host vs and host ace
```

which should not be confused with

```
not ( host vs or ace )
```

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

EXAMPLES

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

```
tcpdump host sundown
```

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

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

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

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

To print all traffic between local hosts and hosts at

Berkeley:

```
tcpdump net ucb-ether
```

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

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

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

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

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

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

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

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

To print IP broadcast or multicast packets that were *not* sent via ethernet broadcast or multicast:

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

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

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