TCPDUMP basic and advanced options

A guide to beginner level and expert options for the command tcpdump

Tcpdump is a packet analyzer that prints out a description of packets being transmitted or received over a network. This document contains real capture commands and what they display.The filters displayed below are meant for copy & paste into a device. The interface and options can simply be changed to suit your needs. Filters can be put in quotes to ensure the filter string is interpreted correctly.

Basic Command Options and Usage

tcpdump [options] [filter]

This table is demonstrating the basic option usage.

Options	Description
Interface Selection	
-D	Show the list of available interfaces on which tcpdump can listen.
-i eth0	Listen on interface eth0.
-i any	Listen on any available interface (cannot be done in promiscuous mode. Requires Linux kernel 2.2 or greater)
Packet Header Output Verbosity (affecting the "dump line")	
-e	Print the link-level header on each dump line.
-q	Be less verbose (than the default) while capturing packets
-v	Be verbose while capturing packets: For example, the time to live, identification, total length and options in an IP packet are printed. Also enables additional packet integrity checks such as verifying the IP and ICMP header checksum.

-vv	Be more verbose while capturing packets: For example, additional fields are printed from NFS reply packets, and SMB packets are fully decoded.
-vvv	Be very verbose while capturing packets: For example, telnet SB ... SE options are printed in full.
-n	Don't do reverse DNS lookup for domain name; display IP addresses. Still resolves service name for ports.
-nn	Don’t resolve hostnames or port names.
-S	Print absolute sequence numbers.
Packet Details Output (Hex and ASCII)	
-x	Print the basic headers, plus print the data of each packet in hex, excluding the link level header
-xx	Print the basic headers, plus print the data of each packet in hex, including its link level header, in hex
-X	Print the data of each packet in both hex and ASCII, excluding the link level header
-XX	Print the data of each packet in both hex and ASCII, also including the link level header
Obscure Options	
-A	Print each packet (minus its link level header) in ASCII.
-b	Print the AS number in BGP packets in ASDOT notation rather than ASPLAIN notation.

Capture Files

Capture Redirected To A File	
tcpdump -w capture.cap	Record the packet capture to a file called capture.cap

tcpdump -v -w capture.cap	Record the packet capture to a file called capture.cap but display on-screen how many packets have been captured in real-time
tcpdump -r capture.cap	Display the packets of a file called capture.cap
tcpdump -vvv -r capture.cap	Display the packets using maximum detail of a file called capture.cap
Capture Buffer Limits	
tcpdump -s 500	Capture 500 bytes of data for each packet rather than the default of 68 bytes.
tcpdump -s 0	Capture all bytes of data within the packet.
tcpdump -c 100	Limit the capture to 100 packets

Advanced Command Usage

tcpdump [options] [filter]

Common Filters

These are real capture commands and what they display. These are just showing the filtering strings, and are probably the most valuable reference for the seasoned engineer.

The filters are displayed below are meant for copy & paste into a device. The interface and options can simply be changed to suit your needs. Filters can be put in quotes to ensure the filter string is interpreted correctly.

When you are analyzing a captured tcpdump, it is often useful to find packets with specific properties. To assist with this process, the tcpdump utility allows the creation of filter expressions based on the following protocol types: ether, fddi, ip, arp, rarp, tcp, udp, icmp

Note: Jump to the Basic Options section if you don't know what the options below are doing.

Layer2	Displays
tcpdump -i swp1 -en ether proto 0x0800	Match IPv4 traffic using EtherType
tcpdump -i swp1 -en ether[12:2] == 0x0800 tcpdump -i swp1 -en ether[12:2] == 2048	Match IPv4 frames using frame offset to match EtherType hex value ...or can use decimal value.
tcpdump -i swp1 -en ether proto 0x0806	Match ARP traffic using EtherType
tcpdump -i swp1 -en ether proto 0x8100	Match VLAN (IEEE 802.1q) traffic using EtherType
tcpdump -i swp1 -en ether proto 0x88a8	Match Provider Bridging (IEEE 802.1ad) traffic using EtherType
tcpdump -i swp1 -en ether proto 0x88cc	Match LLDP frames
tcpdump -i swp1 -en ether proto 0x8809 tcpdump -i swp1 -en ether dst 01:80:c2:00:00:02	Match LACP and other Slow Protocols (IEEE 802.3) using EtherType Match LACP and other Slow Protocols (IEEE 802.3) using destination multicast MAC
tcpdump -i swp1 -en ether dst host 00:12:34:56:78:90	Match all frames with the specified destination MAC address.
tcpdump -i swp1 -en ether src host 00:12:34:56:78:90	Match all frames with the specified source MAC address.
tcpdump -i swp1 -en ether[0] & 1 != 0	Match all multicast traffic
tcpdump -i swp1 -en ether dst host 01:80:c2:00:00:00	Match IEEE STP BPDU frames sent

tcpdump -i swp1 -en ether[20:2]==0x010b	Match Cisco PVST+ BPDU frames sent on native VLAN (untagged)
tcpdump -i swp1 -en ether[24:2]==0x010b	Match Cisco PVST+ BPDU frames sent VLAN tagged
tcpdump -i swp1 -en ether[20:2]==0x010b or ether[24:2]==0x010b	Match Cisco PVST+ BPDU frames sent VLAN tagged or Untagged
tcpdump -i swp1 -en ether host 0:2:b3:7:10:73 and not host 192.168.103.1	Gather all traffic passing through a specific gateway (firewall, router), where 0:2:b3:7:10:73 is the gateway's MAC address and 192.168.103.1 is the gateway's IP address, excluding traffic to and from the gateway itself
Layer2 filtering requiring PCAP file	**Ethernet fields, other than SMAC, DMAC, and Ethertype, can't be used as filters on-the-fly.**
tcpdump -i swp1 -w capture.pcap	Capture the frames to a pcap file
tcpdump -nnr capture.pcap not vlan	Display untagged frames
tcpdump -nnr capture.pcap vlan	Display VLAN tagged (802.1Q) frames
tcpdump -nnr capture.pcap vlan 3999	Display specific VLAN tagged (802.1Q) frames
tcpdump -nnr capture.pcap len == 64	Display frames equal to 64 bytes
Layer3	**Displays**
tcpdump -i swp1 -n port 67 and port 68	Match any BOOTP/DHCP discovery and DHCP offer traffic
tcpdump -i swp1 -n ip	Match any IP packet
tcpdump -i swp1 -n icmp	Match any ICMP packet

tcpdump -i swp1 -n igmp	Match any IGMP packet
tcpdump -i swp1 -n ip proto ospf tcpdump -i swp1 -n ip proto 89	Match any OSPF packet (v2 and v3) Or same filter using the IP protocol number
tcpdump -i swp1 -n	
tcpdump -i swp1 -n dst host 192.168.1.1	Match any packets where the destination IP is 192.168.1.1/32
tcpdump -i swp1 -n src host 192.168.1.1	Match any packets where the source IP is 192.168.1.1/32
tcpdump -i swp1 -n host 192.168.1.1	Match any packets where the source or destination IP is 192.168.1.1/32
tcpdump -i swp1 -n dst net 192.168.1.0/24	Match any packets where the destination network is 192.168.1.0/24
tcpdump -i swp1 -n src net 192.168.1.0/24	Match any packets where the source network is 192.168.1.0/24
tcpdump -i swp1 -n net 192.168.1.0/24	Match any packets where the source or destination network is 192.168.1.0/24
Raw header matching (proto [expr : size])	
tcpdump -i swp1 -n 'ip[6] = 64'	Match the DF bit (don't fragment, to avoid IP fragmentation)
tcpdump -i swp1 -n 'ip[6] = 32'	Matching MF bit (more fragment) set, will match the fragmented datagrams but not the last fragment (has the 2nd bit set to 0).
tcpdump -i swp1 -n '((ip[6:2] > 0) and (not ip[6] = 64))'	Matching the fragments and the last fragments. The (not ip[6] = 64) excludes those with DF set.

tcpdump -i swp1 -n 'ip[0] > 69' tcpdump -i swp1 -n 'ip[0] & 15 > 5' tcpdump -i swp1 -n 'ip[0] & 0xf > 5'	IP Options exist in the packet header. <- Same thing using decimal masking. <- Same thing using hex masking.
tcpdump -i swp1 -n 'ip[8] < 5'	Matching datagrams with low TTL (< 5). Checks for traceroute or looping packets.
tcpdump -i swp1 -n 'ip[2:2] > 600'	Matching packets longer than X bytes. Here X = 600.
tcpdump -i swp1 -n 'ip[9] == 89'	Match OSPF (v2 and v3)
Layer4	
tcpdump -i swp1 -nv tcp	Match all tcp
tcpdump -i swp1 -nv udp	Match all udp
tcpdump -i swp1 -nv dst port 23	Match any packets where the destination port is 23 (UDP or TCP)
tcpdump -i swp1 -nv dst portrange 1-1023	Match any packets where the destination port is is between 1 and 1023 inclusive
tcpdump -i swp1 -nv tcp dst portrange 1-1023	Match only TCP packets where the destination port is is between 1 and 1023 inclusive
tcpdump -i swp1 -nv udp dst portrange 1-1023	Match only UDP packets where the destination port is is between 1 and 1023 inclusive
tcpdump -i swp1 -nv domain tcpdump -i swp1 -nv port 53 tcpdump -i swp1 -nv udp port domain	Match DNS using keyword (resolved by looking in /etc/services) Match DNS using port number Match DNS, but avoids ambiguity between transport-layer protocols

Raw header matching (proto [expr : size])	
tcpdump -i swp1 -nv 'tcp[13] == 1'	Matching only the FIN flag set
tcpdump -i swp1 -nv 'tcp[13] == 2'	Matching only the SYN flag set
tcpdump -i swp1 -nv 'tcp[13] == 8'	Matching only the PSH flag set
tcpdump -i swp1 -nv 'tcp[13] & 32 != 0'	Matching only the URG flag set
tcpdump -i swp1 -nv 'tcp[13] & 2 == 2'	Matching either SYN only or SYN-ACK datagrams
tcpdump -i swp1 -nv 'tcp[13] == 18'	Matching SYN-ACK
tcpdump -i swp1 -nv 'tcp[13] = 24'	Matching PSH-ACK packets
tcpdump -i swp1 -nv 'tcp[13] & 18 == 18'	Matching SYN and ACK, ignore all others
tcpdump -i swp1 -nv 'tcp[13] & 4 == 4'	Matching RST flag set, ignore the others
tcpdump -i swp1 -nv 'tcp[13] & 3 == 3'	Matching SYN and FIN, ignore all others
tcpdump -i swp1 -nv 'tcp[13] & 3 != 0'	Matching either SYN or FIN set
tcpdump -i swp1 -nv 'tcp[13] == 2 or tcp[13] == 1'	Matching 'only SYN' or 'only FIN' set
tcpdump -i swp1 -nv 'tcp[13] & 1 = 1'	Matching any combination containing FIN

tcpdump -i swp1 -nv 'tcp[tcpflags] == tcp-ack'	Matching all packets with TCP-ACK (An easier way to filter flags)
tcpdump -i swp1 -nv 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0	Matching all packets with TCP-SYN or TCP-FIN set
Layer3+4	
tcpdump -i swp1 -nv "dst host 192.168.1.1 and dst port 23"	Match any packets with destination IP 192.168.1.1 and destination port 23
tcpdump -i swp1 -nv "dst host 192.168.1.1 and (dst port 80 or dst port 443)"	Match any packets with destination IP 192.168.1.1 and destination port 80 or 443
tcpdump -i swp1 -nv '((port 67 or port 68) and (udp[38:4] = 0x3e0ccf08))'	Match dhcp packets including a specific client MAC Address. This field is composed of octets 29-35 of the DHCP header. Therefore, 8 octets for UDP header + 28 octets until Client MAC Address + 2 octets offset (drop the first 2 octets of MAC address to allow a 4 octet comparison) = 38 (our total offset). *** Tcpdump only allows matching on a maximum of 4 bytes (octets), not the 6 bytes of a MAC address.
Protocol Filtering	
tcpdump -i swp1 -nv icmp	Match any ICMP packets:
tcpdump -i swp1 '(icmp[0] = 0) and (icmp[0] = 8)'	Match ICMP Echo and Echo Replies

tcpdump -i swp1 'icmp[0] = 3'	Match ICMP Destination Unreachable
tcpdump -i swp1 'icmp[0] = 5'	Match ICMP Redirect
tcpdump -i swp1 'icmp[0] = 11'	Match ICMP Time Exceeded
tcpdump -i swp1 -nv arp	Match any ARP packets:
tcpdump -i swp1 -nv "icmp or arp"	Match either ICMP or ARP packets:
tcpdump -i swp1 -nv "broadcast or multicast"	Match any packets that are broadcast or multicast:
tcpdump -i swp1 -nv ip	Match IPv4 packets
tcpdump -i swp1 -nv ip6	Match IPv6 packets
tcpdump -i swp1 -nv vlan 2007 and igmp	Match all IGMP frames on vlan 2007

Logic and Masking

In some of the examples above, we use logic and masking to create complex filters. Let's take a closer look at how these work.

Combine Expressions	
Negation	! or "not" (without the quotes)
Concatanate	&& or "and"
Alternate	|| or "or"

- Match any TCP traffic on port 80 (web) with 192.168.1.254 or 192.168.1.200 as destination host
- tcpdump -i eth1 '((tcp) and (port 80) and ((dst host 192.168.1.254) or (dst host 192.168.1.200)))'
- Match any ICMP traffic involving the destination with physical/MAC address 00:01:02:03:04:05
- tcpdump -i eth1 '((icmp) and ((ether dst host 00:01:02:03:04:05)))'
- Match any traffic for the destination network 192.168 except destination host 192.168.1.200
- tcpdump -i eth1 '((tcp) and ((dst net 192.168) and (not dst host 192.168.1.200)))

Protocol Masking	
proto[x] == z	Match byte x.
proto[x:y]	Start filtering from byte x for y bytes. ip[2:2] would filter bytes 3 and 4 (first byte begins by 0)
proto[x:y] & z = 0	Match bits set to 0 when applying mask z to proto[x:y]
proto[x:y] & z !=0	Some bits are set when applying mask z to proto[x:y]
proto[x:y] & z = z	All bits are set to z when applying mask z to proto[x:y]
proto[x:y] = z	proto[x:y] has exactly the bits set to z
Operators	>, <, >=, <=, =, !=

The correct way of "masking" the first half of the byte:

0100 0101 : 1st byte originally

0000 1111 : mask (0xf in hex or 15 in decimal). 0 will mask the values, while 1 will keep the values intact.

=========

0000 0101 : final result

You should see the mask as a power switch. 1 means on/enabled, 0 means off/disabled.

Protocol Headers

Of course, it is important to know what the protocol headers look like before diving into more advanced filters.

```
IPv4 header

-----------

Byte  |------ 0 ------|------ 1 ------|------ 2 ------|------ 3 ------|

       0                   1                   2                   3

Bit    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      |Version|  IHL  |Type of Service|          Total Length         |

      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      |         Identification        |Flags|      Fragment Offset    |

      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      |  Time to Live |    Protocol   |         Header Checksum       |

      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                          Source Address                          |

      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      |                       Destination Address                        |

      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      |                       Options                   |    Padding     | <--
optional

      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      |                              DATA ...                            |

      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+



TCP header

----------

Byte  |------ 0 ------|------ 1 ------|------ 2 ------|------ 3 ------|

       0                   1                   2                   3

Bits   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|          Source Port          |       Destination Port        |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|                        Sequence Number                        |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|                    Acknowledgment Number                      |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|  Data |       |C|E|U|A|P|R|S|F|                               |

| Offset|  Res. |W|C|R|C|S|S|Y|I|            Window             |

|       |       |R|E|G|K|H|T|N|N|                               |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|           Checksum            |         Urgent Pointer        |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|                    Options                    |    Padding    |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              data                             |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```