**TTL Overview**

*Disclaimer:  
The following list is a best effort overview of some widely used TCP/IP stacks. The information was provided by vendors and many helpful system administrators. We would like to thank all these contributors for their precious help ! SWITCH cannot, however, take any responsibility that the provided information is correct. Furthermore, SWITCH cannot be made liable for any damage that may arise by the use of this information.*

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

| OS Version |"safe" | tcp\_ttl | udp\_ttl |

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

AIX n 60 30

DEC Pathworks V5 n 30 30

FreeBSD 2.1R y 64 64

HP/UX 9.0x n 30 30

HP/UX 10.01 y 64 64

Irix 5.3 y 60 60

Irix 6.x y 60 60

Linux y 64 64

MacOS/MacTCP 2.0.x y 60 60

OS/2 TCP/IP 3.0 y 64 64

OSF/1 V3.2A n 60 30

Solaris 2.x y 255 255

SunOS 4.1.3/4.1.4 y 60 60

Ultrix V4.1/V4.2A n 60 30

VMS/Multinet y 64 64

VMS/TCPware y 60 64

VMS/Wollongong 1.1.1.1 n 128 30

VMS/UCX (latest rel.) y 128 128

MS WfW n 32 32

MS Windows 95 n 32 32

MS Windows NT 3.51 n 32 32

MS Windows NT 4.0 y 128 128

<http://www.map.meteoswiss.ch/map-doc/ftp-probleme.htm>

# Default Time To Live (TTL) values

TTL is a timer value included in packets sent over TCP/IP-based networks that tells the recipients how long to hold or use the packet or any of its included data before expiring and discarding the packet or data.

Folloing are the list of Devices / Operating system with Default TTL values

|  |  |  |  |
| --- | --- | --- | --- |
| **OS/Device** | **Version** | **Protocol** | **TTL** |
| AIX |  | TCP | 60 |
| AIX |  | UDP | 30 |
| AIX | 3.2, 4.1 | ICMP | 255 |
| BSDI | BSD/OS 3.1 and 4.0 | ICMP | 255 |
| Compa | Tru64 v5.0 | ICMP | 64 |
| Cisco |  | ICMP | 254 |
| DEC Pathworks | V5 | TCP and UDP | 30 |
| Foundry |  | ICMP | 64 |
| FreeBSD | 2.1R | TCP and UDP | 64 |
| FreeBSD | 3.4, 4.0 | ICMP | 255 |
| FreeBSD | 5 | ICMP | 64 |
| HP-UX | 9.0x | TCP and UDP | 30 |
| HP-UX | 10.01 | TCP and UDP | 64 |
| HP-UX | 10.2 | ICMP | 255 |
| HP-UX | 11 | ICMP | 255 |
| HP-UX | 11 | TCP | 64 |
| Irix | 5.3 | TCP and UDP | 60 |
| Irix | 6.x | TCP and UDP | 60 |
| Irix | 6.5.3, 6.5.8 | ICMP | 255 |
| juniper |  | ICMP | 64 |
| MPE/IX (HP) |  | ICMP | 200 |
| Linux | 2.0.x kernel | ICMP | 64 |
| Linux | 2.2.14 kernel | ICMP | 255 |
| Linux | 2.4 kernel | ICMP | 255 |
| Linux | Red Hat 9 | ICMP and TCP | 64 |
| MacOS/MacTCP | 2.0.x | TCP and UDP | 60 |
| MacOS/MacTCP | X (10.5.6) | ICMP/TCP/UDP | 64 |
| NetBSD |  | ICMP | 255 |
| Netgear FVG318 |  | ICMP and UDP | 64 |
| OpenBSD | 2.6 & 2.7 | ICMP | 255 |
| OpenVMS | 07.01.2002 | ICMP | 255 |
| OS/2 | TCP/IP 3.0 |  | 64 |
| OSF/1 | V3.2A | TCP | 60 |
| OSF/1 | V3.2A | UDP | 30 |
| Solaris | 2.5.1, 2.6, 2.7, 2.8 | ICMP | 255 |
| Solaris | 2.8 | TCP | 64 |
| Stratus | TCP\_OS | ICMP | 255 |
| Stratus | TCP\_OS (14.2-) | TCP and UDP | 30 |
| Stratus | TCP\_OS (14.3+) | TCP and UDP | 64 |
| Stratus | STCP | ICMP/TCP/UDP | 60 |
| SunOS | 4.1.3/4.1.4 | TCP and UDP | 60 |
| SunOS | 5.7 | ICMP and TCP | 255 |
| Ultrix | V4.1/V4.2A | TCP | 60 |
| Ultrix | V4.1/V4.2A | UDP | 30 |
| Ultrix | V4.2 – 4.5 | ICMP | 255 |
| VMS/Multinet |  | TCP and UDP | 64 |
| VMS/TCPware |  | TCP | 60 |
| VMS/TCPware |  | UDP | 64 |
| VMS/Wollongong | 1.1.1.1 | TCP | 128 |
| VMS/Wollongong | 1.1.1.1 | UDP | 30 |
| VMS/UCX |  | TCP and UDP | 128 |
| Windows | for Workgroups | TCP and UDP | 32 |
| Windows | 95 | TCP and UDP | 32 |
| Windows | 98 | ICMP | 32 |
| Windows | 98, 98 SE | ICMP | 128 |
| Windows | 98 | TCP | 128 |
| Windows | NT 3.51 | TCP and UDP | 32 |
| Windows | NT 4.0 | TCP and UDP | 128 |
| Windows | NT 4.0 SP5- |  | 32 |
| Windows | NT 4.0 SP6+ |  | 128 |
| Windows | NT 4 WRKS SP 3, SP 6a | ICMP | 128 |
| Windows | NT 4 Server SP4 | ICMP | 128 |
| Windows | ME | ICMP | 128 |
| Windows | 2000 pro | ICMP/TCP/UDP | 128 |
| Windows | 2000 family | ICMP | 128 |
| Windows | Server 2003 |  | 128 |
| Windows | XP | ICMP/TCP/UDP | 128 |

## [Passive OS Fingerprinting](http://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting)



*Network traffic from a computer can be analyzed to detect what operating system it is running. This is to a large extent due to differences in how the TCP/IP stack is implemented in various operating systems. We will in this blog post explain the different methods that can be used to identify what operating a computer is running by analyzing the packets it generates on the network.*

**Active approaches**

The popular port scanner [Nmap](http://nmap.org/) can identify the operating system (OS) of a remote computer by sending six packets with [specially crafted option combinations](http://nmap.org/book/osdetect-methods.html) in the TCP layer (for example window scale, NOP and EOL options). Nmap then watches how the scanned host responds to these odd packets. [Fyodor](http://insecure.org/fyodor/) (author of Nmap) gives a good overview of these techniques in [issue 54 of phrack magazine](http://www.phrack.org/issues.html?issue=54&id=9#article) from way back in 1998.

**Passive OS identification**

Active measures, like those employed by Nmap, are unfortunately not available when doing passive analysis of live traffic or when analyzing previously captured network traffic. Passive analysis requires much more subtle variations in the network traffic to be observed, in order to identify a computer's OS. A simple but effective passive method is to inspect the initial Time To Live (TTL) in the IP header and the TCP window size (the size of the receive window) of the first packet in a TCP session, i.e. the SYN or SYN+ACK packet.

Below are some typical initial TTL values and window sizes of common operating systems:

|  |  |  |
| --- | --- | --- |
| **Operating System (OS)** | **IP Initial TTL** | **TCP window size** |
| **Linux (kernel 2.4 and 2.6)** | 64 | 5840 |
| **Google's customized Linux** | 64 | 5720 |
| **FreeBSD** | 64 | 65535 |
| **Windows XP** | 128 | 65535 |
| **Windows 7, Vista and Server 2008** | 128 | 8192 |
| **Cisco Router (IOS 12.4)** | 255 | 4128 |

One reason for why the TTL and window size values varies between different OS's is because the RFC's for [TCP](http://www.ietf.org/rfc/rfc793.txt) and [IP](http://www.ietf.org/rfc/rfc791.txt) do not require implementations to use any particular default value for these fields. There is, however, a *recommendation* in [RFC 1700](http://tools.ietf.org/html/rfc1700) saying:

The current recommended default time to live (TTL) for the Internet Protocol (IP) is 64

This recommendation is obviously not followed in many IP implementations.

The initial TTL value is often a bit tricky to analyze since the TTL value of a sniffed packet will vary depending on where you sniff it from. The sending host will set the TTL value to the OS's default TTL value, but this value will then be decremented by one for every router the packet passes on its way to the destination IP address. An observed IP packet with a TTL value of 57 can therefore be expected to be a packet with an initial TTL of 64 that has done 7 router hops before it was sniffed.

The TTL and window size table above can be used in order to do manual OS fingerprinting of network traffic. Here is an example showing how to display relevant fields of the first few packets from the [publicly available pcap file](http://www.netresec.com/?page=PcapFiles) for the [2009-M57-Patents scenario](http://digitalcorpora.org/corpora/scenarios/m57-patents-scenario) with tshark:

$ tshark -r day12-1.dmp -R "tcp.flags.syn eq 1" -T fields -e ip.src -e ip.ttl -e tcp.window\_size -c 16 | sort -u  
192.168.1.105  128  8192  
192.168.1.106  128  65535  
74.125.19.139  54   5720  
87.106.12.47   45   5840  
87.106.12.77   45   5840  
87.106.13.61   45   5840  
87.106.13.62   45   5840  
87.106.1.47    45   5840  
87.106.1.89    45   5840  
87.106.66.233  45   5840

The first column here is the IP address (ip.src), the second is the TTL (ip.ttl) and the third the TCP window size (tcp.window\_size). Note that the TTL value is only at the initial value for the hosts on the local network (192.168.1.0/24), while the packets from the other hosts seem to have performed 10 or 19 router hops. We can, just by matching the TTL and window sizes of these hosts with the table above, easily determine that 192.168.1.106 is running Windows XP (TTL=128, window\_size=65535) and 192.168.1.105 is running some more modern flavor of Windows (TTL=128, window\_size=8192). The google machine (with IP 74.125.19.139) can also easily be singled out due to its characteristic window size of 5720. The other machines (87.106.x.x) all seem to be running Linux.

Do you feel manual OS classification would take too much time? There are, luckily, multiple tools like [ettercap](http://ettercap.sourceforge.net/), [p0f](http://lcamtuf.coredump.cx/p0f.shtml), [Satori](http://myweb.cableone.net/xnih/) and [NetworkMiner](http://www.netresec.com/?page=NetworkMiner) which all automate the OS identification task. Just feed these tools with some live network traffic or a pcap file and they'll fingerprint the OS's for you.

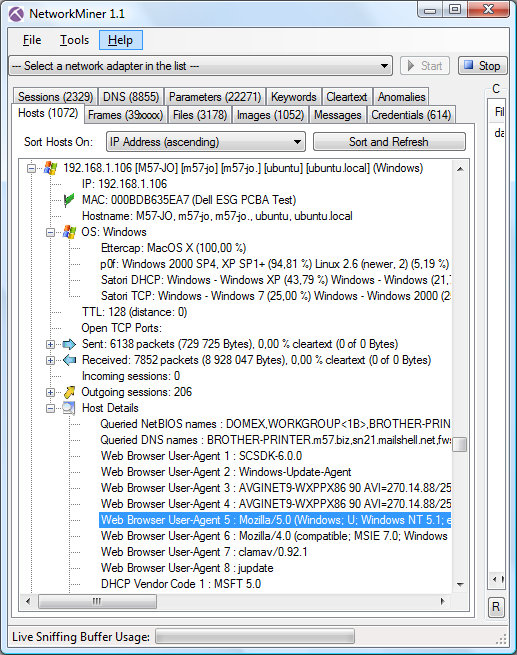
**DHCP Fingerprinting**

An alternative to fingerprinting the TCP/IP stack implementation of an OS is to look at its DHCP implementation. Eric Kollmann (the creator of Satori) has written a great paper on DHCP fingerprinting titled [Chatter on the Wire: A look at DHCP traffic](http://myweb.cableone.net/xnih/download/Chatter-DHCP.pdf). Eric's DHCP fingerprinting database is used in his tool [Satori](http://myweb.cableone.net/xnih/) as well as in [NetworkMiner](http://www.netresec.com/?page=NetworkMiner).

There is also a project titled [Fingerbank](http://www.fingerbank.org/english/about.html), which maintains another DHCP fingerprinting database.

**Application Layer**

Even more info about the operating system of a host can be extracted by inspecting the application layer data in traffic, such as server banners in HTTP, SSH and FTP as well as HTTP client [User-Agent](http://en.wikipedia.org/wiki/User_agent) strings. All these layer 7 banner types are displayed in NetworkMiner's Hosts tab under the “Host Details” node.



A User-Agent string showing “Windows NT 5.1” (like in the screen shot above) means that the client is running Windows XP. Microsoft provides an article titled [Understanding User-Agent Strings](http://msdn.microsoft.com/en-us/library/ms537503(v=vs.85).aspx), which provides this mapping between User-Agent strings and operating system:

|  |  |
| --- | --- |
| **Platform token** | **Description** |
| Windows NT 6.1 | Windows 7 |
| Windows NT 6.0 | Windows Vista |
| Windows NT 5.2 | Windows Server 2003; Windows XP x64 Edition |
| Windows NT 5.1 | Windows XP |

Happy fingerprinting!

The Signatures

There are four areas that we will look at to determine the operating system (however there are other signatures that can be used). These signatures are:

* TTL - What the operating system sets the Time To Live on the outbound packet.
* Window Size - What the operating system sets the Window Size at.
* DF - Does the operating system set the Don't Fragment bit.
* TOS - Does the operating system set the Type of Service, and if so, at what.

By analyzing these factors of a packet, you may be able to determine the remote operating system. This system is not 100% accurate, and works better for some operating systems then others. No single signature can reliably determine the remote operating system. However, by looking at several signatures and combining the information, you increase the accuracy of identifying the remote host. An example would be the easiest way to explain. Below is the sniffer trace of a system sending a packet. This system launched a mountd exploit against me, so I want to learn more about it. I do not want to finger or nmap the box, that could give me away. Rather, I want to study the information passively. This sig

<http://www.symantec.com/connect/articles/passive-fingerprinting>

**Why do I care about the network latency (time between hops) in my Traceroutes.**

Formal Definition of Network Latency: The overall performance of distributed applications depends on the hosts and the network connectivity between them. *Network latency* is the delay that is introduced by the network; it excludes the hosts’ software processing at the source and destination end-points. Network latency can be measured either *one-way* (the time from the source sending a packet to the destination receiving it), or *round-trip* (which is the sum of the one-way latency from source to destination plus the one-way latency from the destination back to the source).

Some literature and tools define network latency as the delay from the time of the start of packet transmission at the sender to the start of packet reception at the receiver. The different definitions can lead to substantial discrepancies in certain situations. To avoid confusion, be sure to interpret latency metrics correctly according to specific and relevant definitions. It is also important to distinguish between network latency and propagation delay: propagation delay is the travel time of one bit from one side to the other inside of the cable, while network latency is the time to deliver an entire data unit (packet) from one host to the other.

The factors that influence network latency are:

1) Wire transport speed.

Not to be confused with the amount of data a wire carry in a second, I am referring to the raw speed at which data travels on a wire. Once on the wire, the traversal time from end to end. For the most part, we can assume data travels at the speed of light: 186,000 miles per second.

2) Distance.

How far is the data traveling. Even though data travels at the speed of light, a hop across the United States will cost you about 4 milliseconds, and a hop up to a stationary satellite  ( round trip about 44,000 miles) adds a minimum of 300 milliseconds. I have worked through an example of how you can  trace latency across a satellite link below.

3) Number of hops.

How many switching points are there between source and destination? Each hop requires the data to move from one wire to another, and this requires a small amount of waiting to get on the next wire. Each hop can be an additional 2 or  3 milliseconds.

4) Overhead processing on a hop.

This can also add up, sometimes at the end points points, people like to look at the data, usually for security reasons, on their firewall. Depending on the number of features and processing power of the firewall this can also add a wide range of latency. Normal is from 1 or 2 milliseconds, but that can blow up to 50 milliseconds or in some cases even more when you turn on too many features on your firewall.

Here are some common latency measurements in milliseconds (ms):

1 ms – within your LAN  
25 ms – my home cable service in London to servers located in mainland UK  
90 ms – typical home DSL in the US to google.com  
100-150 ms – the transatlantic cable between the UK and New York state  
600-2000 ms – typical VSAT remote to hub link