openvpn

Section: Maintenance Commands (8)
Index

NAME

openvpn - secure IP tunnel daemon.

SYNOPSIS

openvpn [options ...]

INTRODUCTION

OpenVPN is an open source VPN daemon by James Yonan. Because OpenVPN tries to be a universal VPN tool offering a great deal of flexibility, there are a lot of options on this manual page. If you're new to OpenVPN, you might want to skip ahead to the examples section where you will see how to construct simple VPNs on the command line without even needing a configuration file.

Also note that there's more documentation and examples on the OpenVPN web site: *http://openvpn.net/*

And if you would like to see a shorter version of this manual, see the openvpn usage message which can be obtained by running **openvpn** without any parameters.

DESCRIPTION

OpenVPN is a robust and highly flexible VPN daemon. OpenVPN supports SSL/TLS security, ethernet bridging, TCP or UDP tunnel transport through proxies or NAT, support for dynamic IP addresses and DHCP, scalability to hundreds or thousands of users, and portability to most major OS platforms.

OpenVPN is tightly bound to the OpenSSL library, and derives much of its crypto capabilities from it.

OpenVPN supports conventional encryption using a pre-shared secret key **(Static Key mode)** or public key security **(SSL/TLS mode)** using client & server certificates. OpenVPN also supports non-encrypted TCP/UDP tunnels.

OpenVPN is designed to work with the **TUN/TAP** virtual networking interface that exists on most platforms.

Overall, OpenVPN aims to offer many of the key features of IPSec but with a relatively lightweight footprint.

OPTIONS

OpenVPN allows any option to be placed either on the command line or in a configuration file. Though all command line options are preceded by a double-leading-dash ("--"), this prefix can be removed when an option is placed in a configuration file.

--help
Show options.

--config file
Load additional config options from **file** where each line corresponds to one command line option, but with the leading '--' removed.

If **--config file** is the only option to the openvpn command, the **--config** can be removed, and the command can be given as **openvpn file**

Note that configuration files can be nested to a reasonable depth.

Double quotation or single quotation characters ("", '') can be used to enclose single parameters containing whitespace, and "#" or ";" characters in the first column can be used to denote comments.

Note that OpenVPN 2.0 and higher performs backslash-based shell escaping for characters not in single quotations, so the following mappings should be observed:

```
\\       Maps to a single backslash character (\).
\"       Pass a literal doublequote character ("), don't
         interpret it as enclosing a parameter.
\[SPACE] Pass a literal space or tab character, don't
         interpret it as a parameter delimiter.
```

For example on Windows, use double backslashes to represent pathnames:

```
secret "c:\\OpenVPN\\secret.key"
```

For examples of configuration files, see *http://openvpn.net/examples.html*

Here is an example configuration file:

```
#
# Sample OpenVPN configuration file for
# using a pre-shared static key.
#
# '#' or ';' may be used to delimit comments.

# Use a dynamic tun device.
dev tun

# Our remote peer
remote mypeer.mydomain

# 10.1.0.1 is our local VPN endpoint
# 10.1.0.2 is our remote VPN endpoint
ifconfig 10.1.0.1 10.1.0.2

# Our pre-shared static key
secret static.key
```

Tunnel Options:

--mode m
Set OpenVPN major mode. By default, OpenVPN runs in point-to-point mode ("p2p"). OpenVPN 2.0 introduces a new mode ("server") which implements a multi-client server capability.

--local host
Local host name or IP address for bind. If specified, OpenVPN will bind to this address only. If unspecified, OpenVPN will bind to all interfaces.

--remote host [port] [proto]
Remote host name or IP address. On the client, multiple **--remote** options may be specified for redundancy, each referring to a different OpenVPN server. Specifying multiple **--remote** options for this purpose is a special case of the more general connection-profile feature. See the **<connection>** documentation below.

The OpenVPN client will try to connect to a server at **host:port** in the order specified by the list of **--remote** options.

proto indicates the protocol to use when connecting with the remote, and may be "tcp" or "udp".

The client will move on to the next host in the list, in the event of connection failure. Note that at any given time, the OpenVPN client will at most be connected to one server.

Note that since UDP is connectionless, connection failure is defined by the **--ping** and **--ping-restart** options.

Note the following corner case: If you use multiple **--remote** options, AND you are dropping root privileges on the client with **--user** and/or **--group,** AND the client is running a non-Windows OS, if the client needs to switch to a different server, and that server pushes back different TUN/TAP or route settings, the client may lack the necessary privileges to close and reopen the TUN/TAP interface. This could cause the client to exit with a fatal error.

If **--remote** is unspecified, OpenVPN will listen for packets from any IP address, but will not act on those packets unless they pass all authentication tests. This requirement for authentication is binding on all potential peers, even those from known and supposedly trusted IP addresses (it is very easy to forge a source IP address on a UDP packet).

When used in TCP mode, **--remote** will act as a filter, rejecting connections from any host which does not match **host.**

If **host** is a DNS name which resolves to multiple IP addresses, the first address returned by the system getaddrinfo() function will be used (no DNS randomization inside OpenVPN 2.3.x, and it will not try multiple addresses).

--remote-random-hostname
Prepend a random string (6 bytes, 12 hex characters) to hostname to prevent DNS caching. For example, "foo.bar.gov" would be modified to "<random-chars>.foo.bar.gov".

<connection>

Define a client connection profile. Client connection profiles are groups of OpenVPN options that describe how to connect to a given OpenVPN server. Client connection profiles are specified within an OpenVPN configuration file, and each profile is bracketed by **<connection>** and **</connection>.**

An OpenVPN client will try each connection profile sequentially until it achieves a successful connection.

--remote-random can be used to initially "scramble" the connection list.

Here is an example of connection profile usage:

```
client
dev tun

<connection>
remote 198.19.34.56 1194 udp
</connection>

<connection>
remote 198.19.34.56 443 tcp
</connection>

<connection>
remote 198.19.34.56 443 tcp
http-proxy 192.168.0.8 8080
http-proxy-retry
</connection>

<connection>
remote 198.19.36.99 443 tcp
http-proxy 192.168.0.8 8080
http-proxy-retry
</connection>

persist-key
persist-tun
pkcs12 client.p12
ns-cert-type server
verb 3
```

First we try to connect to a server at 198.19.34.56:1194 using UDP. If that fails, we then try to connect to 198.19.34.56:443 using TCP. If that also fails, then try connecting through an HTTP proxy at 192.168.0.8:8080 to 198.19.34.56:443 using TCP. Finally, try to connect through the same proxy to a server at 198.19.36.99:443 using TCP.

The following OpenVPN options may be used inside of a **<connection>** block:

bind, connect-retry, connect-retry-max, connect-timeout, explicit-exit-notify, float, fragment, http-proxy, http-proxy-option, http-proxy-retry, http-proxy-timeout, link-mtu, local, lport, mssfix, mtu-disc, nobind, port, proto, remote, rport, socks-proxy, socks-proxy-retry, tun-mtu and tun-mtu-extra.

A defaulting mechanism exists for specifying options to apply to all **<connection>** profiles. If any of the above options (with the exception of **remote**) appear outside of a **<connection>** block, but in a configuration file which has one or more **<connection>** blocks, the option setting will be used as a default for **<connection>** blocks which follow it in the configuration file.

For example, suppose the **nobind** option were placed in the sample configuration file above, near the top of the file, before the first **<connection>** block. The effect would be as if **nobind** were declared in all **<connection>** blocks below it.

--proto-force p
When iterating through connection profiles, only consider profiles using protocol **p** ('tcp'|'udp').

--remote-random
When multiple **--remote** address/ports are specified, or if connection profiles are being used, initially randomize the order of the list as a kind of basic load-balancing measure.

--proto p
Use protocol **p** for communicating with remote host. **p** can be **udp, tcp-client,** or **tcp-server.**

The default protocol is **udp** when **--proto** is not specified.

For UDP operation, **--proto udp** should be specified on both peers.

For TCP operation, one peer must use **--proto tcp-server** and the other must use **--proto tcp-client.** A peer started with **tcp-server** will wait indefinitely for an incoming connection. A

peer started with **tcp-client** will attempt to connect, and if that fails, will sleep for 5 seconds (adjustable via the **--connect-retry** option) and try again infinite or up to N retries (adjustable via the **--connect-retry-max** option). Both TCP client and server will simulate a SIGUSR1 restart signal if either side resets the connection.

OpenVPN is designed to operate optimally over UDP, but TCP capability is provided for situations where UDP cannot be used. In comparison with UDP, TCP will usually be somewhat less efficient and less robust when used over unreliable or congested networks.

This article outlines some of problems with tunneling IP over TCP:

http://sites.inka.de/sites/bigred/devel/tcp-tcp.html

There are certain cases, however, where using TCP may be advantageous from a security and robustness perspective, such as tunneling non-IP or application-level UDP protocols, or tunneling protocols which don't possess a built-in reliability layer.

--connect-retry n
For **--proto tcp-client,** take **n** as the number of seconds to wait between connection retries (default=5).

--connect-timeout n
For **--proto tcp-client,** set connection timeout to **n** seconds (default=10).

--connect-retry-max n
For **--proto tcp-client,** take **n** as the number of retries of connection attempt (default=infinite).

--show-proxy-settings
Show sensed HTTP or SOCKS proxy settings. Currently, only Windows clients support this option.

--http-proxy server port [authfile|'auto'|'auto-nct'] [auth-method]
Connect to remote host through an HTTP proxy at address **server** and port **port.** If HTTP Proxy-Authenticate is required, **authfile** is a file containing a username and password on 2 lines, or "stdin" to prompt from console.

auth-method should be one of "none", "basic", or "ntlm".

HTTP Digest authentication is supported as well, but only via the **auto** or **auto-nct** flags (below).

The **auto** flag causes OpenVPN to automatically determine the **auth-method** and query stdin or the management interface for username/password credentials, if required. This flag exists on OpenVPN 2.1 or higher.

The **auto-nct** flag (no clear-text auth) instructs OpenVPN to automatically determine the authentication method, but to reject weak authentication protocols such as HTTP Basic Authentication.

--http-proxy-retry
Retry indefinitely on HTTP proxy errors. If an HTTP proxy error occurs, simulate a SIGUSR1 reset.

--http-proxy-timeout n
Set proxy timeout to **n** seconds, default=5.

--http-proxy-option type [parm]
Set extended HTTP proxy options. Repeat to set multiple options.

VERSION version -- Set HTTP version number to **version** (default=1.0).

AGENT user-agent -- Set HTTP "User-Agent" string to **user-agent.**

--socks-proxy server [port] [authfile]
Connect to remote host through a Socks5 proxy at address **server** and port **port** (default=1080). **authfile** (optional) is a file containing a username and password on 2 lines, or "stdin" to prompt from console.

--socks-proxy-retry
Retry indefinitely on Socks proxy errors. If a Socks proxy error occurs, simulate a SIGUSR1 reset.

--resolv-retry n
If hostname resolve fails for **--remote,** retry resolve for **n** seconds before failing.

Set **n** to "infinite" to retry indefinitely.

By default, **--resolv-retry infinite** is enabled. You can disable by setting n=0.

--float
Allow remote peer to change its IP address and/or port number, such as due to DHCP (this is the default if **--remote** is not used). **--float** when specified with **--remote** allows an OpenVPN session to initially connect to a peer at a known address, however if packets arrive from a new address and pass all authentication tests, the new address will take control of the session. This is useful when you are connecting to a peer which holds a dynamic address such as a dial-in user or DHCP client.

Essentially, **--float** tells OpenVPN to accept authenticated packets from any address, not only the

address which was specified in the **--remote** option.

--ipchange cmd
Run command **cmd** when our remote ip-address is initially authenticated or changes.

cmd consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single- or double-quoted and/or escaped using a backslash, and should be separated by one or more spaces.

When **cmd** is executed two arguments are appended after any arguments specified in **cmd** , as follows:

cmd ip_address port_number

Don't use **--ipchange** in **--mode server** mode. Use a **--client-connect** script instead.

See the "Environmental Variables" section below for additional parameters passed as environmental variables.

If you are running in a dynamic IP address environment where the IP addresses of either peer could change without notice, you can use this script, for example, to edit the */etc/hosts* file with the current address of the peer. The script will be run every time the remote peer changes its IP address.

Similarly if *our* IP address changes due to DHCP, we should configure our IP address change script (see man page for **dhcpcd**(8)) to deliver a **SIGHUP** or **SIGUSR1** signal to OpenVPN. OpenVPN will then reestablish a connection with its most recently authenticated peer on its new IP address.

--port port
TCP/UDP port number or port name for both local and remote (sets both **--lport** and **--rport** options to given port). The current default of 1194 represents the official IANA port number assignment for OpenVPN and has been used since version 2.0-beta17. Previous versions used port 5000 as the default.
--lport port
Set local TCP/UDP port number or name. Cannot be used together with **--nobind** option.
--rport port
Set TCP/UDP port number or name used by the **--remote** option. The port can also be set directly using the **--remote** option.
--bind
Bind to local address and port. This is the default unless any of **--proto tcp-client** , **--http-proxy** or **--socks-proxy** are used.
--nobind
Do not bind to local address and port. The IP stack will allocate a dynamic port for returning packets. Since the value of the dynamic port could not be known in advance by a peer, this option is only suitable for peers which will be initiating connections by using the **--remote** option.
--dev tunX | tapX | null
TUN/TAP virtual network device (**X** can be omitted for a dynamic device.)

See examples section below for an example on setting up a TUN device.

You must use either tun devices on both ends of the connection or tap devices on both ends. You cannot mix them, as they represent different underlying network layers.

tun devices encapsulate IPv4 or IPv6 (OSI Layer 3) while **tap** devices encapsulate Ethernet 802.3 (OSI Layer 2).

--dev-type device-type
Which device type are we using? **device-type** should be **tun** (OSI Layer 3) or **tap** (OSI Layer 2). Use this option only if the TUN/TAP device used with **--dev** does not begin with **tun** or **tap.**
--topology mode
Configure virtual addressing topology when running in **--dev tun** mode. This directive has no meaning in **--dev tap** mode, which always uses a **subnet** topology.

If you set this directive on the server, the **--server** and **--server-bridge** directives will automatically push your chosen topology setting to clients as well. This directive can also be manually pushed to clients. Like the **--dev** directive, this directive must always be compatible between client and server.

mode can be one of:

net30 -- Use a point-to-point topology, by allocating one /30 subnet per client. This is designed to allow point-to-point semantics when some or all of the connecting clients might be Windows systems. This is the default on OpenVPN 2.0.

p2p -- Use a point-to-point topology where the remote endpoint of the client's tun interface always points to the local endpoint of the server's tun interface. This mode allocates a single IP

address per connecting client. Only use when none of the connecting clients are Windows systems. This mode is functionally equivalent to the **--ifconfig-pool-linear** directive which is available in OpenVPN 2.0 and is now deprecated.

subnet -- Use a subnet rather than a point-to-point topology by configuring the tun interface with a local IP address and subnet mask, similar to the topology used in **--dev tap** and ethernet bridging mode. This mode allocates a single IP address per connecting client and works on Windows as well. Only available when server and clients are OpenVPN 2.1 or higher, or OpenVPN 2.0.x which has been manually patched with the **--topology** directive code. When used on Windows, requires version 8.2 or higher of the TAP-Win32 driver. When used on *nix, requires that the tun driver supports an **ifconfig**(8) command which sets a subnet instead of a remote endpoint IP address.

This option exists in OpenVPN 2.1 or higher.

Note: Using **--topology subnet** changes the interpretation of the arguments of **--ifconfig** to mean "address netmask", no longer "local remote".

--tun-ipv6
Build a tun link capable of forwarding IPv6 traffic. Should be used in conjunction with **--dev tun** or **--dev tunX.** A warning will be displayed if no specific IPv6 TUN support for your OS has been compiled into OpenVPN.

See below for further IPv6-related configuration options.

--dev-node node
Explicitly set the device node rather than using /dev/net/tun, /dev/tun, /dev/tap, etc. If OpenVPN cannot figure out whether **node** is a TUN or TAP device based on the name, you should also specify **--dev-type tun** or **--dev-type tap.**

Under Mac OS X this option can be used to specify the default tun implementation. Using **--dev-node utun** forces usage of the native Darwin tun kernel support. Use **--dev-node utunN** to select a specific utun instance. To force using the tun.kext (/dev/tunX) use **--dev-node tun**. When not specifying a **--dev-node** option openvpn will first try to open utun, and fall back to tun.kext.

On Windows systems, select the TAP-Win32 adapter which is named **node** in the Network Connections Control Panel or the raw GUID of the adapter enclosed by braces. The **--show-adapters** option under Windows can also be used to enumerate all available TAP-Win32 adapters and will show both the network connections control panel name and the GUID for each TAP-Win32 adapter.

--lladdr address
Specify the link layer address, more commonly known as the MAC address. Only applied to TAP devices.
--iproute cmd
Set alternate command to execute instead of default iproute2 command. May be used in order to execute OpenVPN in unprivileged environment.
--ifconfig l rn
Set TUN/TAP adapter parameters. **l** is the IP address of the local VPN endpoint. For TUN devices in point-to-point mode, **rn** is the IP address of the remote VPN endpoint. For TAP devices, or TUN devices used with **--topology subnet, rn** is the subnet mask of the virtual network segment which is being created or connected to.

For TUN devices, which facilitate virtual point-to-point IP connections (when used in **--topology net30** or **p2p** mode), the proper usage of **--ifconfig** is to use two private IP addresses which are not a member of any existing subnet which is in use. The IP addresses may be consecutive and should have their order reversed on the remote peer. After the VPN is established, by pinging **rn,** you will be pinging across the VPN.

For TAP devices, which provide the ability to create virtual ethernet segments, or TUN devices in **--topology subnet** mode (which create virtual "multipoint networks"), **--ifconfig** is used to set an IP address and subnet mask just as a physical ethernet adapter would be similarly configured. If you are attempting to connect to a remote ethernet bridge, the IP address and subnet should be set to values which would be valid on the the bridged ethernet segment (note also that DHCP can be used for the same purpose).

This option, while primarily a proxy for the **ifconfig**(8) command, is designed to simplify TUN/TAP tunnel configuration by providing a standard interface to the different ifconfig implementations on different platforms.

--ifconfig parameters which are IP addresses can also be specified as a DNS or /etc/hosts file resolvable name.

For TAP devices, **--ifconfig** should not be used if the TAP interface will be getting an IP address lease from a DHCP server.

--ifconfig-noexec
Don't actually execute ifconfig/netsh commands, instead pass **--ifconfig** parameters to scripts using environmental variables.

--ifconfig-nowarn
Don't output an options consistency check warning if the **--ifconfig** option on this side of the connection doesn't match the remote side. This is useful when you want to retain the overall benefits of the options consistency check (also see **--disable-occ** option) while only disabling the ifconfig component of the check.

For example, if you have a configuration where the local host uses **--ifconfig** but the remote host does not, use **--ifconfig-nowarn** on the local host.

This option will also silence warnings about potential address conflicts which occasionally annoy more experienced users by triggering "false positive" warnings.

--route network/IP [netmask] [gateway] [metric]
Add route to routing table after connection is established. Multiple routes can be specified. Routes will be automatically torn down in reverse order prior to TUN/TAP device close.

This option is intended as a convenience proxy for the **route**(8) shell command, while at the same time providing portable semantics across OpenVPN's platform space.

netmask default -- 255.255.255.255

gateway default -- taken from **--route-gateway** or the second parameter to **--ifconfig** when **--dev tun** is specified.

metric default -- taken from **--route-metric** otherwise 0.

The default can be specified by leaving an option blank or setting it to "default".

The **network** and **gateway** parameters can also be specified as a DNS or /etc/hosts file resolvable name, or as one of three special keywords:

vpn_gateway -- The remote VPN endpoint address (derived either from **--route-gateway** or the second parameter to **--ifconfig** when **--dev tun** is specified).

net_gateway -- The pre-existing IP default gateway, read from the routing table (not supported on all OSes).

remote_host -- The **--remote** address if OpenVPN is being run in client mode, and is undefined in server mode.

--max-routes n
Allow a maximum number of n **--route** options to be specified, either in the local configuration file, or pulled from an OpenVPN server. By default, n=100.

--route-gateway gw|'dhcp'
Specify a default gateway **gw** for use with **--route.**

If **dhcp** is specified as the parameter, the gateway address will be extracted from a DHCP negotiation with the OpenVPN server-side LAN.

--route-metric m
Specify a default metric **m** for use with **--route.**

--route-delay [n] [w]
Delay **n** seconds (default=0) after connection establishment, before adding routes. If **n** is 0, routes will be added immediately upon connection establishment. If **--route-delay** is omitted, routes will be added immediately after TUN/TAP device open and **--up** script execution, before any **--user** or **--group** privilege downgrade (or **--chroot** execution.)

This option is designed to be useful in scenarios where DHCP is used to set tap adapter addresses. The delay will give the DHCP handshake time to complete before routes are added.

On Windows, **--route-delay** tries to be more intelligent by waiting **w** seconds (w=30 by default) for the TAP-Win32 adapter to come up before adding routes.

--route-up cmd
Run command **cmd** after routes are added, subject to **--route-delay.**

cmd consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single- or double-quoted and/or escaped using a backslash, and should be separated by one or more spaces.

See the "Environmental Variables" section below for additional parameters passed as environmental variables.

--route-pre-down cmd

Run command **cmd** before routes are removed upon disconnection.

cmd consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single- or double-quoted and/or escaped using a backslash, and should be separated by one or more spaces.

See the "Environmental Variables" section below for additional parameters passed as environmental variables.

--route-noexec
Don't add or remove routes automatically. Instead pass routes to **--route-up** script using environmental variables.

--route-nopull
When used with **--client** or **--pull,** accept options pushed by server EXCEPT for routes and dhcp options like DNS servers.

When used on the client, this option effectively bars the server from adding routes to the client's routing table, however note that this option still allows the server to set the TCP/IP properties of the client's TUN/TAP interface.

--allow-pull-fqdn
Allow client to pull DNS names from server (rather than being limited to IP address) for **--ifconfig, --route,** and **--route-gateway.**

--client-nat snat|dnat network netmask alias
This pushable client option sets up a stateless one-to-one NAT rule on packet addresses (not ports), and is useful in cases where routes or ifconfig settings pushed to the client would create an IP numbering conflict.

network/netmask (for example 192.168.0.0/255.255.0.0) defines the local view of a resource from the client perspective, while **alias/netmask** (for example 10.64.0.0/255.255.0.0) defines the remote view from the server perspective.

Use **snat** (source NAT) for resources owned by the client and **dnat** (destination NAT) for remote resources.

Set **--verb 6** for debugging info showing the transformation of src/dest addresses in packets.

--redirect-gateway flags...
Automatically execute routing commands to cause all outgoing IP traffic to be redirected over the VPN. This is a client-side option.

This option performs three steps:

(1) Create a static route for the **--remote** address which forwards to the pre-existing default gateway. This is done so that **(3)** will not create a routing loop.

(2) Delete the default gateway route.

(3) Set the new default gateway to be the VPN endpoint address (derived either from **--route-gateway** or the second parameter to **--ifconfig** when **--dev tun** is specified).

When the tunnel is torn down, all of the above steps are reversed so that the original default route is restored.

Option flags:

local -- Add the **local** flag if both OpenVPN servers are directly connected via a common subnet, such as with wireless. The **local** flag will cause step **1** above to be omitted.

autolocal -- Try to automatically determine whether to enable **local** flag above.

def1 -- Use this flag to override the default gateway by using 0.0.0.0/1 and 128.0.0.0/1 rather than 0.0.0.0/0. This has the benefit of overriding but not wiping out the original default gateway.

bypass-dhcp -- Add a direct route to the DHCP server (if it is non-local) which bypasses the tunnel (Available on Windows clients, may not be available on non-Windows clients).

bypass-dns -- Add a direct route to the DNS server(s) (if they are non-local) which bypasses the tunnel (Available on Windows clients, may not be available on non-Windows clients).

block-local -- Block access to local LAN when the tunnel is active, except for the LAN gateway itself. This is accomplished by routing the local LAN (except for the LAN gateway address) into the tunnel.

--link-mtu n
Sets an upper bound on the size of UDP packets which are sent between OpenVPN peers. It's best not to set this parameter unless you know what you're doing.

--redirect-private [flags]
Like --redirect-gateway, but omit actually changing the default gateway. Useful when pushing private subnets.

--tun-mtu n
Take the TUN device MTU to be **n** and derive the link MTU from it (default=1500). In most cases, you will probably want to leave this parameter set to its default value.

The MTU (Maximum Transmission Units) is the maximum datagram size in bytes that can be sent unfragmented over a particular network path. OpenVPN requires that packets on the control or data channels be sent unfragmented.

MTU problems often manifest themselves as connections which hang during periods of active usage.

It's best to use the **--fragment** and/or **--mssfix** options to deal with MTU sizing issues.

--tun-mtu-extra n
Assume that the TUN/TAP device might return as many as **n** bytes more than the **--tun-mtu** size on read. This parameter defaults to 0, which is sufficient for most TUN devices. TAP devices may introduce additional overhead in excess of the MTU size, and a setting of 32 is the default when TAP devices are used. This parameter only controls internal OpenVPN buffer sizing, so there is no transmission overhead associated with using a larger value.

--mtu-disc type
Should we do Path MTU discovery on TCP/UDP channel? Only supported on OSes such as Linux that supports the necessary system call to set.

'**no**' -- Never send DF (Don't Fragment) frames
'**maybe**' -- Use per-route hints
'**yes**' -- Always DF (Don't Fragment)

--mtu-test
To empirically measure MTU on connection startup, add the **--mtu-test** option to your configuration. OpenVPN will send ping packets of various sizes to the remote peer and measure the largest packets which were successfully received. The **--mtu-test** process normally takes about 3 minutes to complete.

--fragment max
Enable internal datagram fragmentation so that no UDP datagrams are sent which are larger than **max** bytes.

The **max** parameter is interpreted in the same way as the **--link-mtu** parameter, i.e. the UDP packet size after encapsulation overhead has been added in, but not including the UDP header itself.

The **--fragment** option only makes sense when you are using the UDP protocol (**--proto udp**).

--fragment adds 4 bytes of overhead per datagram.

See the **--mssfix** option below for an important related option to **--fragment.**

It should also be noted that this option is not meant to replace UDP fragmentation at the IP stack level. It is only meant as a last resort when path MTU discovery is broken. Using this option is less efficient than fixing path MTU discovery for your IP link and using native IP fragmentation instead.

Having said that, there are circumstances where using OpenVPN's internal fragmentation capability may be your only option, such as tunneling a UDP multicast stream which requires fragmentation.

--mssfix max
Announce to TCP sessions running over the tunnel that they should limit their send packet sizes such that after OpenVPN has encapsulated them, the resulting UDP packet size that OpenVPN sends to its peer will not exceed **max** bytes. The default value is **1450.**

The **max** parameter is interpreted in the same way as the **--link-mtu** parameter, i.e. the UDP packet size after encapsulation overhead has been added in, but not including the UDP header itself.

The **--mssfix** option only makes sense when you are using the UDP protocol for OpenVPN peer-to-peer communication, i.e. **--proto udp.**

--mssfix and **--fragment** can be ideally used together, where **--mssfix** will try to keep TCP from needing packet fragmentation in the first place, and if big packets come through anyhow (from protocols other than TCP), **--fragment** will internally fragment them.

Both **--fragment** and **--mssfix** are designed to work around cases where Path MTU discovery is broken on the network path between OpenVPN peers.

The usual symptom of such a breakdown is an OpenVPN connection which successfully starts, but

then stalls during active usage.

If **--fragment** and **--mssfix** are used together, **--mssfix** will take its default **max** parameter from the **--fragment max** option.

Therefore, one could lower the maximum UDP packet size to 1300 (a good first try for solving MTU-related connection problems) with the following options:

--tun-mtu 1500 --fragment 1300 --mssfix

--sndbuf size
Set the TCP/UDP socket send buffer size. Currently defaults to 65536 bytes.

--rcvbuf size
Set the TCP/UDP socket receive buffer size. Currently defaults to 65536 bytes.

--mark value
Mark encrypted packets being sent with value. The mark value can be matched in policy routing and packetfilter rules. This option is only supported in Linux and does nothing on other operating systems.

--socket-flags flags...
Apply the given flags to the OpenVPN transport socket. Currently, only **TCP_NODELAY** is supported.

The **TCP_NODELAY** socket flag is useful in TCP mode, and causes the kernel to send tunnel packets immediately over the TCP connection without trying to group several smaller packets into a larger packet. This can result in a considerably improvement in latency.

This option is pushable from server to client, and should be used on both client and server for maximum effect.

--txqueuelen n
(Linux only) Set the TX queue length on the TUN/TAP interface. Currently defaults to 100.

--shaper n
Limit bandwidth of outgoing tunnel data to **n** bytes per second on the TCP/UDP port. Note that this will only work if mode is set to p2p. If you want to limit the bandwidth in both directions, use this option on both peers.

OpenVPN uses the following algorithm to implement traffic shaping: Given a shaper rate of *n* bytes per second, after a datagram write of *b* bytes is queued on the TCP/UDP port, wait a minimum of *(b / n)* seconds before queuing the next write.

It should be noted that OpenVPN supports multiple tunnels between the same two peers, allowing you to construct full-speed and reduced bandwidth tunnels at the same time, routing low-priority data such as off-site backups over the reduced bandwidth tunnel, and other data over the full-speed tunnel.

Also note that for low bandwidth tunnels (under 1000 bytes per second), you should probably use lower MTU values as well (see above), otherwise the packet latency will grow so large as to trigger timeouts in the TLS layer and TCP connections running over the tunnel.

OpenVPN allows **n** to be between 100 bytes/sec and 100 Mbytes/sec.

--inactive n [bytes]
Causes OpenVPN to exit after **n** seconds of inactivity on the TUN/TAP device. The time length of inactivity is measured since the last incoming or outgoing tunnel packet. The default value is 0 seconds, which disables this feature.

If the optional **bytes** parameter is included, exit if less than **bytes** of combined in/out traffic are produced on the tun/tap device in **n** seconds.

In any case, OpenVPN's internal ping packets (which are just keepalives) and TLS control packets are not considered "activity", nor are they counted as traffic, as they are used internally by OpenVPN and are not an indication of actual user activity.

--ping n
Ping remote over the TCP/UDP control channel if no packets have been sent for at least **n** seconds (specify **--ping** on both peers to cause ping packets to be sent in both directions since OpenVPN ping packets are not echoed like IP ping packets). When used in one of OpenVPN's secure modes (where **--secret, --tls-server,** or **--tls-client** is specified), the ping packet will be cryptographically secure.

This option has two intended uses:

(1) Compatibility with stateful firewalls. The periodic ping will ensure that a stateful firewall rule which allows OpenVPN UDP packets to pass will not time out.

(2) To provide a basis for the remote to test the existence of its peer using the **--ping-exit** option.

--ping-exit n
Causes OpenVPN to exit after **n** seconds pass without reception of a ping or other packet from remote. This option can be combined with **--inactive, --ping,** and **--ping-exit** to create a two-tiered inactivity disconnect.

For example,

openvpn [options...] --inactive 3600 --ping 10 --ping-exit 60

when used on both peers will cause OpenVPN to exit within 60 seconds if its peer disconnects, but will exit after one hour if no actual tunnel data is exchanged.

--ping-restart n
Similar to **--ping-exit,** but trigger a **SIGUSR1** restart after **n** seconds pass without reception of a ping or other packet from remote.

This option is useful in cases where the remote peer has a dynamic IP address and a low-TTL DNS name is used to track the IP address using a service such as *http://dyndns.org/* + a dynamic DNS client such as **ddclient.**

If the peer cannot be reached, a restart will be triggered, causing the hostname used with **--remote** to be re-resolved (if **--resolv-retry** is also specified).

In server mode, **--ping-restart, --inactive,** or any other type of internally generated signal will always be applied to individual client instance objects, never to whole server itself. Note also in server mode that any internally generated signal which would normally cause a restart, will cause the deletion of the client instance object instead.

In client mode, the **--ping-restart** parameter is set to 120 seconds by default. This default will hold until the client pulls a replacement value from the server, based on the **--keepalive** setting in the server configuration. To disable the 120 second default, set **--ping-restart 0** on the client.

See the signals section below for more information on **SIGUSR1.**

Note that the behavior of **SIGUSR1** can be modified by the **--persist-tun, --persist-key, --persist-local-ip,** and **--persist-remote-ip** options.

Also note that **--ping-exit** and **--ping-restart** are mutually exclusive and cannot be used together.

--keepalive n m
A helper directive designed to simplify the expression of **--ping** and **--ping-restart** in server mode configurations.

The server timeout is set twice the value of the second argument. This ensures that a timeout is detected on client side before the server side drops the connection.

For example, **--keepalive 10 60** expands as follows:

```
 if mode server:
   ping 10
   ping-restart 120
   push "ping 10"
   push "ping-restart 60"
 else
   ping 10
   ping-restart 60
```

--ping-timer-rem
Run the **--ping-exit** / **--ping-restart** timer only if we have a remote address. Use this option if you are starting the daemon in listen mode (i.e. without an explicit **--remote** peer), and you don't want to start clocking timeouts until a remote peer connects.

--persist-tun
Don't close and reopen TUN/TAP device or run up/down scripts across **SIGUSR1** or **--ping-restart** restarts.

SIGUSR1 is a restart signal similar to **SIGHUP,** but which offers finer-grained control over reset options.

--persist-key
Don't re-read key files across **SIGUSR1** or **--ping-restart.**

This option can be combined with **--user nobody** to allow restarts triggered by the **SIGUSR1** signal. Normally if you drop root privileges in OpenVPN, the daemon cannot be restarted since it will now be unable to re-read protected key files.

This option solves the problem by persisting keys across **SIGUSR1** resets, so they don't need to

be re-read.

--persist-local-ip
Preserve initially resolved local IP address and port number across **SIGUSR1** or **--ping-restart** restarts.

--persist-remote-ip
Preserve most recently authenticated remote IP address and port number across **SIGUSR1** or **--ping-restart** restarts.

--mlock
Disable paging by calling the POSIX mlockall function. Requires that OpenVPN be initially run as root (though OpenVPN can subsequently downgrade its UID using the **--user** option).

Using this option ensures that key material and tunnel data are never written to disk due to virtual memory paging operations which occur under most modern operating systems. It ensures that even if an attacker was able to crack the box running OpenVPN, he would not be able to scan the system swap file to recover previously used ephemeral keys, which are used for a period of time governed by the **--reneg** options (see below), then are discarded.

The downside of using **--mlock** is that it will reduce the amount of physical memory available to other applications.

--up cmd
Run command **cmd** after successful TUN/TAP device open (pre **--user** UID change).

cmd consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single- or double-quoted and/or escaped using a backslash, and should be separated by one or more spaces.

The up command is useful for specifying route commands which route IP traffic destined for private subnets which exist at the other end of the VPN connection into the tunnel.

For **--dev tun** execute as:

cmd tun_dev tun_mtu link_mtu ifconfig_local_ip ifconfig_remote_ip [init | restart]

For **--dev tap** execute as:

cmd tap_dev tap_mtu link_mtu ifconfig_local_ip ifconfig_netmask [init | restart]

See the "Environmental Variables" section below for additional parameters passed as environmental variables.

Note that if **cmd** includes arguments, all OpenVPN-generated arguments will be appended to them to build an argument list with which the executable will be called.

Typically, **cmd** will run a script to add routes to the tunnel.

Normally the up script is called after the TUN/TAP device is opened. In this context, the last command line parameter passed to the script will be *init*. If the **--up-restart** option is also used, the up script will be called for restarts as well. A restart is considered to be a partial reinitialization of OpenVPN where the TUN/TAP instance is preserved (the **--persist-tun** option will enable such preservation). A restart can be generated by a SIGUSR1 signal, a **--ping-restart** timeout, or a connection reset when the TCP protocol is enabled with the **--proto** option. If a restart occurs, and **--up-restart** has been specified, the up script will be called with *restart* as the last parameter.

NOTE: on restart, OpenVPN will not pass the full set of environment variables to the script. Namely, everything related to routing and gateways will not be passed, as nothing needs to be done anyway - all the routing setup is already in place. Additionally, the up-restart script will run with the downgraded UID/GID settings (if configured).

The following standalone example shows how the **--up** script can be called in both an initialization and restart context. (NOTE: for security reasons, don't run the following example unless UDP port 9999 is blocked by your firewall. Also, the example will run indefinitely, so you should abort with control-c).

openvpn --dev tun --port 9999 --verb 4 --ping-restart 10 --up 'echo up' --down 'echo down' --persist-tun --up-restart

Note that OpenVPN also provides the **--ifconfig** option to automatically ifconfig the TUN device, eliminating the need to define an **--up** script, unless you also want to configure routes in the **--up** script.

If **--ifconfig** is also specified, OpenVPN will pass the ifconfig local and remote endpoints on the command line to the **--up** script so that they can be used to configure routes such as:

route add -net 10.0.0.0 netmask 255.255.255.0 gw $5

--up-delay
Delay TUN/TAP open and possible **--up** script execution until after TCP/UDP connection establishment with peer.

In **--proto udp** mode, this option normally requires the use of **--ping** to allow connection initiation to be sensed in the absence of tunnel data, since UDP is a "connectionless" protocol.

On Windows, this option will delay the TAP-Win32 media state transitioning to "connected" until connection establishment, i.e. the receipt of the first authenticated packet from the peer.

--down cmd
Run command **cmd** after TUN/TAP device close (post **--user** UID change and/or **--chroot**). **cmd** consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single- or double-quoted and/or escaped using a backslash, and should be separated by one or more spaces.

Called with the same parameters and environmental variables as the **--up** option above.

Note that if you reduce privileges by using **--user** and/or **--group,** your **--down** script will also run at reduced privilege.

--down-pre
Call **--down** cmd/script before, rather than after, TUN/TAP close.
--up-restart
Enable the **--up** and **--down** scripts to be called for restarts as well as initial program start. This option is described more fully above in the **--up** option documentation.
--setenv name value
Set a custom environmental variable **name=value** to pass to script.
--setenv FORWARD_COMPATIBLE 1
Relax config file syntax checking so that unknown directives will trigger a warning but not a fatal error, on the assumption that a given unknown directive might be valid in future OpenVPN versions.

This option should be used with caution, as there are good security reasons for having OpenVPN fail if it detects problems in a config file. Having said that, there are valid reasons for wanting new software features to gracefully degrade when encountered by older software versions.

It is also possible to tag a single directive so as not to trigger a fatal error if the directive isn't recognized. To do this, prepend the following before the directive: **setenv opt**

Versions prior to OpenVPN 2.3.3 will always ignore options set with the **setenv opt** directive.

See also **--ignore-unknown-option**

--setenv-safe name value
Set a custom environmental variable **OPENVPN_name=value** to pass to script.

This directive is designed to be pushed by the server to clients, and the prepending of "OPENVPN_" to the environmental variable is a safety precaution to prevent a LD_PRELOAD style attack from a malicious or compromised server.

--ignore-unknown-option opt1 opt2 opt3 ... optN
When one of options **opt1 ... optN** is encountered in the configuration file the configuration file parsing does not fail if this OpenVPN version does not support the option. Multiple **--ignore-unknown-option** options can be given to support a larger number of options to ignore.

This option should be used with caution, as there are good security reasons for having OpenVPN fail if it detects problems in a config file. Having said that, there are valid reasons for wanting new software features to gracefully degrade when encountered by older software versions.

--ignore-unknown-option is available since OpenVPN 2.3.3.

--script-security level
This directive offers policy-level control over OpenVPN's usage of external programs and scripts. Lower **level** values are more restrictive, higher values are more permissive. Settings for **level:**

0 -- Strictly no calling of external programs.
1 -- (Default) Only call built-in executables such as ifconfig, ip, route, or netsh.
2 -- Allow calling of built-in executables and user-defined scripts.
3 -- Allow passwords to be passed to scripts via environmental variables (potentially unsafe).

OpenVPN releases before v2.3 also supported a **method** flag which indicated how OpenVPN should call external commands and scripts. This could be either **execve** or **system.** As of OpenVPN v2.3, this flag is no longer accepted. In most *nix environments the execve() approach has been used without any issues.

Some directives such as --up allow options to be passed to the external script. In these cases

make sure the script name does not contain any spaces or the configuration parser will choke because it can't determine where the script name ends and script options start.

To run scripts in Windows in earlier OpenVPN versions you needed to either add a full path to the script interpreter which can parse the script or use the **system** flag to run these scripts. As of OpenVPN v2.3 it is now a strict requirement to have full path to the script interpreter when running non-executables files. This is not needed for executable files, such as .exe, .com, .bat or .cmd files. For example, if you have a Visual Basic script, you must use this syntax now:

```
--up 'C:\\Windows\\System32\\wscript.exe C:\\Program\ Files\\OpenVPN\\config\\my-up-script.vbs
```

Please note the single quote marks and the escaping of the backslashes (\) and the space character.

The reason the support for the **system** flag was removed is due to the security implications with shell expansions when executing scripts via the system() call.

--disable-occ
Don't output a warning message if option inconsistencies are detected between peers. An example of an option inconsistency would be where one peer uses **--dev tun** while the other peer uses **--dev tap.**

Use of this option is discouraged, but is provided as a temporary fix in situations where a recent version of OpenVPN must connect to an old version.

--user user
Change the user ID of the OpenVPN process to **user** after initialization, dropping privileges in the process. This option is useful to protect the system in the event that some hostile party was able to gain control of an OpenVPN session. Though OpenVPN's security features make this unlikely, it is provided as a second line of defense.

By setting **user** to *nobody* or somebody similarly unprivileged, the hostile party would be limited in what damage they could cause. Of course once you take away privileges, you cannot return them to an OpenVPN session. This means, for example, that if you want to reset an OpenVPN daemon with a **SIGUSR1** signal (for example in response to a DHCP reset), you should make use of one or more of the **--persist** options to ensure that OpenVPN doesn't need to execute any privileged operations in order to restart (such as re-reading key files or running **ifconfig** on the TUN device).

--group group
Similar to the **--user** option, this option changes the group ID of the OpenVPN process to **group** after initialization.
--cd dir
Change directory to **dir** prior to reading any files such as configuration files, key files, scripts, etc. **dir** should be an absolute path, with a leading "/", and without any references to the current directory such as "." or "..".

This option is useful when you are running OpenVPN in **--daemon** mode, and you want to consolidate all of your OpenVPN control files in one location.

--chroot dir
Chroot to **dir** after initialization. **--chroot** essentially redefines **dir** as being the top level directory tree (/). OpenVPN will therefore be unable to access any files outside this tree. This can be desirable from a security standpoint.

Since the chroot operation is delayed until after initialization, most OpenVPN options that reference files will operate in a pre-chroot context.

In many cases, the **dir** parameter can point to an empty directory, however complications can result when scripts or restarts are executed after the chroot operation.

Note: if OpenVPN is built using the PolarSSL SSL library, **--chroot** will only work if a /dev/urandom device node is available inside the chroot directory **dir.** This is due to the way PolarSSL works (it wants to open /dev/urandom every time randomness is needed, not just once at startup) and nothing OpenVPN can influence.

--setcon context
Apply SELinux **context** after initialization. This essentially provides the ability to restrict OpenVPN's rights to only network I/O operations, thanks to SELinux. This goes further than **--user** and **--chroot** in that those two, while being great security features, unfortunately do not protect against privilege escalation by exploitation of a vulnerable system call. You can of course combine all three, but please note that since setcon requires access to /proc you will have to provide it inside the chroot directory (e.g. with mount --bind).

Since the setcon operation is delayed until after initialization, OpenVPN can be restricted to just network-related system calls, whereas by applying the context before startup (such as the

OpenVPN one provided in the SELinux Reference Policies) you will have to allow many things required only during initialization.

Like with chroot, complications can result when scripts or restarts are executed after the setcon operation, which is why you should really consider using the **--persist-key** and **--persist-tun** options.

--daemon [progname]
Become a daemon after all initialization functions are completed. This option will cause all message and error output to be sent to the syslog file (such as /var/log/messages), except for the output of scripts and ifconfig commands, which will go to /dev/null unless otherwise redirected. The syslog redirection occurs immediately at the point that **--daemon** is parsed on the command line even though the daemonization point occurs later. If one of the **--log** options is present, it will supercede syslog redirection.

The optional **progname** parameter will cause OpenVPN to report its program name to the system logger as **progname.** This can be useful in linking OpenVPN messages in the syslog file with specific tunnels. When unspecified, **progname** defaults to "openvpn".

When OpenVPN is run with the **--daemon** option, it will try to delay daemonization until the majority of initialization functions which are capable of generating fatal errors are complete. This means that initialization scripts can test the return status of the openvpn command for a fairly reliable indication of whether the command has correctly initialized and entered the packet forwarding event loop.

In OpenVPN, the vast majority of errors which occur after initialization are non-fatal.

Note: as soon as OpenVPN has daemonized, it can not ask for usernames, passwords, or key pass phrases anymore. This has certain consequences, namely that using a password-protected private key will fail unless the **--askpass** option is used to tell OpenVPN to ask for the pass phrase (this requirement is new in 2.3.7, and is a consequence of calling daemon() before initializing the crypto layer).

Further, using **--daemon** together with **--auth-user-pass** (entered on console) and **--auth-nocache** will fail as soon as key renegotiation (and reauthentication) occurs.

--syslog [progname]
Direct log output to system logger, but do not become a daemon. See **--daemon** directive above for description of **progname** parameter.
--errors-to-stderr
Output errors to stderr instead of stdout unless log output is redirected by one of the **--log** options.
--passtos
Set the TOS field of the tunnel packet to what the payload's TOS is.
--inetd [wait|nowait] [progname]
Use this option when OpenVPN is being run from the inetd or **xinetd(8)** server.

The **wait/nowait** option must match what is specified in the inetd/xinetd config file. The **nowait** mode can only be used with **--proto tcp-server.** The default is **wait.** The **nowait** mode can be used to instantiate the OpenVPN daemon as a classic TCP server, where client connection requests are serviced on a single port number. For additional information on this kind of configuration, see the OpenVPN FAQ: *http://openvpn.net/faq.html#oneport*

This option precludes the use of **--daemon, --local,** or **--remote.** Note that this option causes message and error output to be handled in the same way as the **--daemon** option. The optional **progname** parameter is also handled exactly as in **--daemon.**

Also note that in **wait** mode, each OpenVPN tunnel requires a separate TCP/UDP port and a separate inetd or xinetd entry. See the OpenVPN 1.x HOWTO for an example on using OpenVPN with xinetd: *http://openvpn.net/1xhowto.html*

--log file
Output logging messages to **file,** including output to stdout/stderr which is generated by called scripts. If **file** already exists it will be truncated. This option takes effect immediately when it is parsed in the command line and will supercede syslog output if **--daemon** or **--inetd** is also specified. This option is persistent over the entire course of an OpenVPN instantiation and will not be reset by SIGHUP, SIGUSR1, or **--ping-restart.**

Note that on Windows, when OpenVPN is started as a service, logging occurs by default without the need to specify this option.

--log-append file
Append logging messages to **file.** If **file** does not exist, it will be created. This option behaves exactly like **--log** except that it appends to rather than truncating the log file.
--suppress-timestamps
Avoid writing timestamps to log messages, even when they otherwise would be prepended. In

particular, this applies to log messages sent to stdout.

--writepid file
Write OpenVPN's main process ID to **file.**

--nice n
Change process priority after initialization (**n** greater than 0 is lower priority, **n** less than zero is higher priority).

--fast-io
(Experimental) Optimize TUN/TAP/UDP I/O writes by avoiding a call to poll/epoll/select prior to the write operation. The purpose of such a call would normally be to block until the device or socket is ready to accept the write. Such blocking is unnecessary on some platforms which don't support write blocking on UDP sockets or TUN/TAP devices. In such cases, one can optimize the event loop by avoiding the poll/epoll/select call, improving CPU efficiency by 5% to 10%.

This option can only be used on non-Windows systems, when **--proto udp** is specified, and when **--shaper** is NOT specified.

--multihome
Configure a multi-homed UDP server. This option needs to be used when a server has more than one IP address (e.g. multiple interfaces, or secondary IP addresses), and is not using **--local** to force binding to one specific address only. This option will add some extra lookups to the packet path to ensure that the UDP reply packets are always sent from the address that the client is talking to. This is not supported on all platforms, and it adds more processing, so it's not enabled by default.

Note: this option is only relevant for UDP servers.

Note 2: if you do an IPv6+IPv4 dual-stack bind on a Linux machine with multiple IPv4 address, connections to IPv4 addresses will not work right on kernels before 3.15, due to missing kernel support for the IPv4-mapped case (some distributions have ported this to earlier kernel versions, though).

--echo [parms...]
Echo **parms** to log output.

Designed to be used to send messages to a controlling application which is receiving the OpenVPN log output.

--remap-usr1 signal
Control whether internally or externally generated SIGUSR1 signals are remapped to SIGHUP (restart without persisting state) or SIGTERM (exit).

signal can be set to "SIGHUP" or "SIGTERM". By default, no remapping occurs.

--verb n
Set output verbosity to **n** (default=1). Each level shows all info from the previous levels. Level 3 is recommended if you want a good summary of what's happening without being swamped by output.

0 -- No output except fatal errors.
1 to 4 -- Normal usage range.
5 -- Output **R** and **W** characters to the console for each packet read and write, uppercase is used for TCP/UDP packets and lowercase is used for TUN/TAP packets.
6 to 11 -- Debug info range (see errlevel.h for additional information on debug levels).

--status file [n]
Write operational status to **file** every **n** seconds.

Status can also be written to the syslog by sending a **SIGUSR2** signal.

--status-version [n]
Choose the status file format version number. Currently **n** can be 1, 2, or 3 and defaults to 1.

--mute n
Log at most **n** consecutive messages in the same category. This is useful to limit repetitive logging of similar message types.

--comp-lzo [mode]
Use fast LZO compression -- may add up to 1 byte per packet for incompressible data. **mode** may be "yes", "no", or "adaptive" (default).

In a server mode setup, it is possible to selectively turn compression on or off for individual clients.

First, make sure the client-side config file enables selective compression by having at least one **--comp-lzo** directive, such as **--comp-lzo no.** This will turn off compression by default, but allow a future directive push from the server to dynamically change the on/off/adaptive setting.

Next in a **--client-config-dir** file, specify the compression setting for the client, for example:

```
comp-lzo yes
push "comp-lzo yes"
```

The first line sets the **comp-lzo** setting for the server side of the link, the second sets the client side.

--comp-noadapt
When used in conjunction with **--comp-lzo,** this option will disable OpenVPN's adaptive compression algorithm. Normally, adaptive compression is enabled with **--comp-lzo.**

Adaptive compression tries to optimize the case where you have compression enabled, but you are sending predominantly incompressible (or pre-compressed) packets over the tunnel, such as an FTP or rsync transfer of a large, compressed file. With adaptive compression, OpenVPN will periodically sample the compression process to measure its efficiency. If the data being sent over the tunnel is already compressed, the compression efficiency will be very low, triggering openvpn to disable compression for a period of time until the next re-sample test.

--management IP port [pw-file]
Enable a TCP server on **IP:port** to handle daemon management functions. **pw-file,** if specified, is a password file (password on first line) or "stdin" to prompt from standard input. The password provided will set the password which TCP clients will need to provide in order to access management functions.

The management interface can also listen on a unix domain socket, for those platforms that support it. To use a unix domain socket, specify the unix socket pathname in place of **IP** and set **port** to 'unix'. While the default behavior is to create a unix domain socket that may be connected to by any process, the **--management-client-user** and **--management-client-group** directives can be used to restrict access.

The management interface provides a special mode where the TCP management link can operate over the tunnel itself. To enable this mode, set **IP** = "tunnel". Tunnel mode will cause the management interface to listen for a TCP connection on the local VPN address of the TUN/TAP interface.

While the management port is designed for programmatic control of OpenVPN by other applications, it is possible to telnet to the port, using a telnet client in "raw" mode. Once connected, type "help" for a list of commands.

For detailed documentation on the management interface, see the management-notes.txt file in the **management** folder of the OpenVPN source distribution.

It is strongly recommended that **IP** be set to 127.0.0.1 (localhost) to restrict accessibility of the management server to local clients.

--management-client
Management interface will connect as a TCP/unix domain client to **IP:port** specified by **--management** rather than listen as a TCP server or on a unix domain socket.

If the client connection fails to connect or is disconnected, a SIGTERM signal will be generated causing OpenVPN to quit.

--management-query-passwords
Query management channel for private key password and **--auth-user-pass** username/password. Only query the management channel for inputs which ordinarily would have been queried from the console.
--management-query-proxy
Query management channel for proxy server information for a specific **--remote** (client-only).
--management-query-remote
Allow management interface to override **--remote** directives (client-only). **--management-external-key** Allows usage for external private key file instead of **--key** option (client-only).
--management-forget-disconnect
Make OpenVPN forget passwords when management session disconnects.

This directive does not affect the **--http-proxy** username/password. It is always cached.

--management-hold
Start OpenVPN in a hibernating state, until a client of the management interface explicitly starts it with the **hold release** command.
--management-signal
Send SIGUSR1 signal to OpenVPN if management session disconnects. This is useful when you wish to disconnect an OpenVPN session on user logoff. For --management-client this option is not needed since a disconnect will always generate a SIGTERM.
--management-log-cache n
Cache the most recent **n** lines of log file history for usage by the management channel.
--management-up-down
Report tunnel up/down events to management interface.

--management-client-auth
Gives management interface client the responsibility to authenticate clients after their client certificate has been verified. See management-notes.txt in OpenVPN distribution for detailed notes.

--management-client-pf
Management interface clients must specify a packet filter file for each connecting client. See management-notes.txt in OpenVPN distribution for detailed notes.

--management-client-user u
When the management interface is listening on a unix domain socket, only allow connections from user **u.**

--management-client-group g
When the management interface is listening on a unix domain socket, only allow connections from group **g.**

--plugin module-pathname [init-string]
Load plug-in module from the file **module-pathname,** passing **init-string** as an argument to the module initialization function. Multiple plugin modules may be loaded into one OpenVPN process.

For more information and examples on how to build OpenVPN plug-in modules, see the README file in the **plugin** folder of the OpenVPN source distribution.

If you are using an RPM install of OpenVPN, see /usr/share/openvpn/plugin. The documentation is in **doc** and the actual plugin modules are in **lib.**

Multiple plugin modules can be cascaded, and modules can be used in tandem with scripts. The modules will be called by OpenVPN in the order that they are declared in the config file. If both a plugin and script are configured for the same callback, the script will be called last. If the return code of the module/script controls an authentication function (such as tls-verify, auth-user-pass-verify, or client-connect), then every module and script must return success (0) in order for the connection to be authenticated.

Server Mode

Starting with OpenVPN 2.0, a multi-client TCP/UDP server mode is supported, and can be enabled with the **--mode server** option. In server mode, OpenVPN will listen on a single port for incoming client connections. All client connections will be routed through a single tun or tap interface. This mode is designed for scalability and should be able to support hundreds or even thousands of clients on sufficiently fast hardware. SSL/TLS authentication must be used in this mode.

--server network netmask ['nopool']
A helper directive designed to simplify the configuration of OpenVPN's server mode. This directive will set up an OpenVPN server which will allocate addresses to clients out of the given network/netmask. The server itself will take the ".1" address of the given network for use as the server-side endpoint of the local TUN/TAP interface.

For example, **--server 10.8.0.0 255.255.255.0** expands as follows:

```
mode server
tls-server
push "topology [topology]"

if dev tun AND (topology == net30 OR topology == p2p):
  ifconfig 10.8.0.1 10.8.0.2
  if !nopool:
    ifconfig-pool 10.8.0.4 10.8.0.251
  route 10.8.0.0 255.255.255.0
  if client-to-client:
    push "route 10.8.0.0 255.255.255.0"
  else if topology == net30:
    push "route 10.8.0.1"

if dev tap OR (dev tun AND topology == subnet):
  ifconfig 10.8.0.1 255.255.255.0
  if !nopool:
    ifconfig-pool 10.8.0.2 10.8.0.254 255.255.255.0
  push "route-gateway 10.8.0.1"
  if route-gateway unset:
    route-gateway 10.8.0.2
```

Don't use **--server** if you are ethernet bridging. Use **--server-bridge** instead.

--server-bridge gateway netmask pool-start-IP pool-end-IP
--server-bridge ['nogw']

A helper directive similar to **--server** which is designed to simplify the configuration of OpenVPN's server mode in ethernet bridging configurations.

If **--server-bridge** is used without any parameters, it will enable a DHCP-proxy mode, where connecting OpenVPN clients will receive an IP address for their TAP adapter from the DHCP server running on the OpenVPN server-side LAN. Note that only clients that support the binding of a DHCP client with the TAP adapter (such as Windows) can support this mode. The optional **nogw** flag (advanced) indicates that gateway information should not be pushed to the client.

To configure ethernet bridging, you must first use your OS's bridging capability to bridge the TAP interface with the ethernet NIC interface. For example, on Linux this is done with the **brctl** tool, and with Windows XP it is done in the Network Connections Panel by selecting the ethernet and TAP adapters and right-clicking on "Bridge Connections".

Next you you must manually set the IP/netmask on the bridge interface. The **gateway** and **netmask** parameters to **--server-bridge** can be set to either the IP/netmask of the bridge interface, or the IP/netmask of the default gateway/router on the bridged subnet.

Finally, set aside a IP range in the bridged subnet, denoted by **pool-start-IP** and **pool-end-IP,** for OpenVPN to allocate to connecting clients.

For example, **server-bridge 10.8.0.4 255.255.255.0 10.8.0.128 10.8.0.254** expands as follows:

```
mode server
tls-server

ifconfig-pool 10.8.0.128 10.8.0.254 255.255.255.0
push "route-gateway 10.8.0.4"
```

In another example, **--server-bridge** (without parameters) expands as follows:

```
mode server
tls-server

push "route-gateway dhcp"
```

Or **--server-bridge nogw** expands as follows:

```
mode server
tls-server
```

--push option
Push a config file option back to the client for remote execution. Note that **option** must be enclosed in double quotes (""). The client must specify **--pull** in its config file. The set of options which can be pushed is limited by both feasibility and security. Some options such as those which would execute scripts are banned, since they would effectively allow a compromised server to execute arbitrary code on the client. Other options such as TLS or MTU parameters cannot be pushed because the client needs to know them before the connection to the server can be initiated.

This is a partial list of options which can currently be pushed: **--route, --route-gateway, --route-delay, --redirect-gateway, --ip-win32, --dhcp-option, --inactive, --ping, --ping-exit, --ping-restart, --setenv, --persist-key, --persist-tun, --echo, --comp-lzo, --socket-flags, --sndbuf, --rcvbuf**

--push-reset
Don't inherit the global push list for a specific client instance. Specify this option in a client-specific context such as with a **--client-config-dir** configuration file. This option will ignore **--push** options at the global config file level.

--push-peer-info
Push additional information about the client to server. The additional information consists of the following data:

IV_VER=<version> -- the client OpenVPN version

IV_PLAT=[linux|solaris|openbsd|mac|netbsd|freebsd|win] -- the client OS platform

IV_HWADDR=<mac address> -- the MAC address of clients default gateway

IV_LZO_STUB=1 -- if client was built with LZO stub capability

UV_<name>=<value> -- client environment variables whose names start with "UV_"

--disable
Disable a particular client (based on the common name) from connecting. Don't use this option to disable a client due to key or password compromise. Use a CRL (certificate revocation list) instead

(see the **--crl-verify** option).

This option must be associated with a specific client instance, which means that it must be specified either in a client instance config file using **--client-config-dir** or dynamically generated using a **--client-connect** script.

--ifconfig-pool start-IP end-IP [netmask]
Set aside a pool of subnets to be dynamically allocated to connecting clients, similar to a DHCP server. For tun-style tunnels, each client will be given a /30 subnet (for interoperability with Windows clients). For tap-style tunnels, individual addresses will be allocated, and the optional **netmask** parameter will also be pushed to clients.

--ifconfig-pool-persist file [seconds]
Persist/unpersist ifconfig-pool data to **file,** at **seconds** intervals (default=600), as well as on program startup and shutdown.

The goal of this option is to provide a long-term association between clients (denoted by their common name) and the virtual IP address assigned to them from the ifconfig-pool. Maintaining a long-term association is good for clients because it allows them to effectively use the **--persist-tun** option.

file is a comma-delimited ASCII file, formatted as <Common-Name>,<IP-address>.

If **seconds** = 0, **file** will be treated as read-only. This is useful if you would like to treat **file** as a configuration file.

Note that the entries in this file are treated by OpenVPN as suggestions only, based on past associations between a common name and IP address. They do not guarantee that the given common name will always receive the given IP address. If you want guaranteed assignment, use **--ifconfig-push**

--ifconfig-pool-linear
Modifies the **--ifconfig-pool** directive to allocate individual TUN interface addresses for clients rather than /30 subnets. NOTE: This option is incompatible with Windows clients.

This option is deprecated, and should be replaced with **--topology p2p** which is functionally equivalent.

--ifconfig-push local remote-netmask [alias]
Push virtual IP endpoints for client tunnel, overriding the --ifconfig-pool dynamic allocation.

The parameters **local** and **remote-netmask** are set according to the **--ifconfig** directive which you want to execute on the client machine to configure the remote end of the tunnel. Note that the parameters **local** and **remote-netmask** are from the perspective of the client, not the server. They may be DNS names rather than IP addresses, in which case they will be resolved on the server at the time of client connection.

The optional **alias** parameter may be used in cases where NAT causes the client view of its local endpoint to differ from the server view. In this case **local/remote-netmask** will refer to the server view while **alias/remote-netmask** will refer to the client view.

This option must be associated with a specific client instance, which means that it must be specified either in a client instance config file using **--client-config-dir** or dynamically generated using a **--client-connect** script.

Remember also to include a **--route** directive in the main OpenVPN config file which encloses **local,** so that the kernel will know to route it to the server's TUN/TAP interface.

OpenVPN's internal client IP address selection algorithm works as follows:

1 -- Use **--client-connect script** generated file for static IP (first choice).
2 -- Use **--client-config-dir** file for static IP (next choice).
3 -- Use **--ifconfig-pool** allocation for dynamic IP (last choice).

--iroute network [netmask]
Generate an internal route to a specific client. The **netmask** parameter, if omitted, defaults to 255.255.255.255.

This directive can be used to route a fixed subnet from the server to a particular client, regardless of where the client is connecting from. Remember that you must also add the route to the system routing table as well (such as by using the **--route** directive). The reason why two routes are needed is that the **--route** directive routes the packet from the kernel to OpenVPN. Once in OpenVPN, the **--iroute** directive routes to the specific client.

This option must be specified either in a client instance config file using **--client-config-dir** or dynamically generated using a **--client-connect** script.

The **--iroute** directive also has an important interaction with **--push** "route ...". **--iroute** essentially defines a subnet which is owned by a particular client (we will call this client A). If you would like other clients to be able to reach A's subnet, you can use **--push** "route ..." together with **--client-to-client** to effect this. In order for all clients to see A's subnet, OpenVPN must push this route to all clients EXCEPT for A, since the subnet is already owned by A. OpenVPN accomplishes this by not not pushing a route to a client if it matches one of the client's iroutes.

--client-to-client
Because the OpenVPN server mode handles multiple clients through a single tun or tap interface, it is effectively a router. The **--client-to-client** flag tells OpenVPN to internally route client-to-client traffic rather than pushing all client-originating traffic to the TUN/TAP interface.

When this option is used, each client will "see" the other clients which are currently connected. Otherwise, each client will only see the server. Don't use this option if you want to firewall tunnel traffic using custom, per-client rules.

--duplicate-cn
Allow multiple clients with the same common name to concurrently connect. In the absence of this option, OpenVPN will disconnect a client instance upon connection of a new client having the same common name.

--client-connect cmd
Run **command cmd** on client connection.

cmd consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single- or double-quoted and/or escaped using a backslash, and should be separated by one or more spaces.

The command is passed the common name and IP address of the just-authenticated client as environmental variables (see environmental variable section below). The command is also passed the pathname of a freshly created temporary file as the last argument (after any arguments specified in **cmd**), to be used by the command to pass dynamically generated config file directives back to OpenVPN.

If the script wants to generate a dynamic config file to be applied on the server when the client connects, it should write it to the file named by the last argument.

See the **--client-config-dir** option below for options which can be legally used in a dynamically generated config file.

Note that the return value of **script** is significant. If **script** returns a non-zero error status, it will cause the client to be disconnected.

--client-disconnect cmd
Like **--client-connect** but called on client instance shutdown. Will not be called unless the **--client-connect** script and plugins (if defined) were previously called on this instance with successful (0) status returns.

The exception to this rule is if the **--client-disconnect** command or plugins are cascaded, and at least one client-connect function succeeded, then ALL of the client-disconnect functions for scripts and plugins will be called on client instance object deletion, even in cases where some of the related client-connect functions returned an error status.

The **--client-disconnect** command is passed the same pathname as the corresponding **--client-connect** command as its last argument. (after any arguments specified in **cmd**).

--client-config-dir dir
Specify a directory **dir** for custom client config files. After a connecting client has been authenticated, OpenVPN will look in this directory for a file having the same name as the client's X509 common name. If a matching file exists, it will be opened and parsed for client-specific configuration options. If no matching file is found, OpenVPN will instead try to open and parse a default file called "DEFAULT", which may be provided but is not required. Note that the configuration files must be readable by the OpenVPN process after it has dropped it's root privileges.

This file can specify a fixed IP address for a given client using **--ifconfig-push,** as well as fixed subnets owned by the client using **--iroute.**

One of the useful properties of this option is that it allows client configuration files to be conveniently created, edited, or removed while the server is live, without needing to restart the server.

The following options are legal in a client-specific context: **--push, --push-reset, --iroute, --ifconfig-push,** and **--config.**

--ccd-exclusive
Require, as a condition of authentication, that a connecting client has a **--client-config-dir** file.

--tmp-dir dir
Specify a directory **dir** for temporary files. This directory will be used by openvpn processes and script to communicate temporary data with openvpn main process. Note that the directory must be writable by the OpenVPN process after it has dropped it's root privileges.

This directory will be used by in the following cases:

* **--client-connect** scripts to dynamically generate client-specific configuration files.

* **OPENVPN_PLUGIN_AUTH_USER_PASS_VERIFY** plugin hook to return success/failure via auth_control_file when using deferred auth method

* **OPENVPN_PLUGIN_ENABLE_PF** plugin hook to pass filtering rules via pf_file

--hash-size r v
Set the size of the real address hash table to **r** and the virtual address table to **v.** By default, both tables are sized at 256 buckets.

--bcast-buffers n
Allocate **n** buffers for broadcast datagrams (default=256).

--tcp-queue-limit n
Maximum number of output packets queued before TCP (default=64).

When OpenVPN is tunneling data from a TUN/TAP device to a remote client over a TCP connection, it is possible that the TUN/TAP device might produce data at a faster rate than the TCP connection can support. When the number of output packets queued before sending to the TCP socket reaches this limit for a given client connection, OpenVPN will start to drop outgoing packets directed at this client.

--tcp-nodelay
This macro sets the TCP_NODELAY socket flag on the server as well as pushes it to connecting clients. The TCP_NODELAY flag disables the Nagle algorithm on TCP sockets causing packets to be transmitted immediately with low latency, rather than waiting a short period of time in order to aggregate several packets into a larger containing packet. In VPN applications over TCP, TCP_NODELAY is generally a good latency optimization.

The macro expands as follows:

```
if mode server:
  socket-flags TCP_NODELAY
  push "socket-flags TCP_NODELAY"
```

--max-clients n
Limit server to a maximum of **n** concurrent clients.

--max-routes-per-client n
Allow a maximum of **n** internal routes per client (default=256). This is designed to help contain DoS attacks where an authenticated client floods the server with packets appearing to come from many unique MAC addresses, forcing the server to deplete virtual memory as its internal routing table expands. This directive can be used in a **--client-config-dir** file or auto-generated by a **--client-connect** script to override the global value for a particular client.

Note that this directive affects OpenVPN's internal routing table, not the kernel routing table.

--stale-routes-check n [t]
Remove routes haven't had activity for **n** seconds (i.e. the ageing time).

This check is ran every **t** seconds (i.e. check interval).

If **t** is not present it defaults to **n**

This option helps to keep the dynamic routing table small. See also **--max-routes-per-client**

--connect-freq n sec
Allow a maximum of **n** new connections per **sec** seconds from clients. This is designed to contain DoS attacks which flood the server with connection requests using certificates which will ultimately fail to authenticate.

This is an imperfect solution however, because in a real DoS scenario, legitimate connections might also be refused.

For the best protection against DoS attacks in server mode, use **--proto udp** and **--tls-auth.**

--learn-address cmd
Run command **cmd** to validate client virtual addresses or routes.

cmd consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single- or double-quoted and/or escaped using a backslash, and should be separated by one or more spaces.

Three arguments will be appended to any arguments in **cmd** as follows:

[1] operation -- "add", "update", or "delete" based on whether or not the address is being added to, modified, or deleted from OpenVPN's internal routing table.
[2] address -- The address being learned or unlearned. This can be an IPv4 address such as "198.162.10.14", an IPv4 subnet such as "198.162.10.0/24", or an ethernet MAC address (when **--dev tap** is being used) such as "00:FF:01:02:03:04".
[3] common name -- The common name on the certificate associated with the client linked to this address. Only present for "add" or "update" operations, not "delete".

On "add" or "update" methods, if the script returns a failure code (non-zero), OpenVPN will reject the address and will not modify its internal routing table.

Normally, the **cmd** script will use the information provided above to set appropriate firewall entries on the VPN TUN/TAP interface. Since OpenVPN provides the association between virtual IP or MAC address and the client's authenticated common name, it allows a user-defined script to configure firewall access policies with regard to the client's high-level common name, rather than the low level client virtual addresses.

--auth-user-pass-verify cmd method
Require the client to provide a username/password (possibly in addition to a client certificate) for authentication.

OpenVPN will run **command cmd** to validate the username/password provided by the client.

cmd consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single- or double-quoted and/or escaped using a backslash, and should be separated by one or more spaces.

If **method** is set to "via-env", OpenVPN will call **script** with the environmental variables **username** and **password** set to the username/password strings provided by the client. Be aware that this method is insecure on some platforms which make the environment of a process publicly visible to other unprivileged processes.

If **method** is set to "via-file", OpenVPN will write the username and password to the first two lines of a temporary file. The filename will be passed as an argument to **script,** and the file will be automatically deleted by OpenVPN after the script returns. The location of the temporary file is controlled by the **--tmp-dir** option, and will default to the current directory if unspecified. For security, consider setting **--tmp-dir** to a volatile storage medium such as **/dev/shm** (if available) to prevent the username/password file from touching the hard drive.

The script should examine the username and password, returning a success exit code (0) if the client's authentication request is to be accepted, or a failure code (1) to reject the client.

This directive is designed to enable a plugin-style interface for extending OpenVPN's authentication capabilities.

To protect against a client passing a maliciously formed username or password string, the username string must consist only of these characters: alphanumeric, underbar ('_'), dash ('-'), dot ('.'), or at ('@'). The password string can consist of any printable characters except for CR or LF. Any illegal characters in either the username or password string will be converted to underbar ('_').

Care must be taken by any user-defined scripts to avoid creating a security vulnerability in the way that these strings are handled. Never use these strings in such a way that they might be escaped or evaluated by a shell interpreter.

For a sample script that performs PAM authentication, see **sample-scripts/auth-pam.pl** in the OpenVPN source distribution.

--opt-verify
Clients that connect with options that are incompatible with those of the server will be disconnected.

Options that will be compared for compatibility include dev-type, link-mtu, tun-mtu, proto, tun-ipv6, ifconfig, comp-lzo, fragment, keydir, cipher, auth, keysize, secret, no-replay, no-iv, tls-auth, key-method, tls-server, and tls-client.

This option requires that **--disable-occ** NOT be used.

--auth-user-pass-optional
Allow connections by clients that do not specify a username/password. Normally, when **--auth-user-pass-verify** or **--management-client-auth** is specified (or an authentication plugin module), the OpenVPN server daemon will require connecting clients to specify a username and password. This option makes the submission of a username/password by clients optional, passing the responsibility to the user-defined authentication module/script to accept or deny the client

based on other factors (such as the setting of X509 certificate fields). When this option is used, and a connecting client does not submit a username/password, the user-defined authentication module/script will see the username and password as being set to empty strings (""). The authentication module/script MUST have logic to detect this condition and respond accordingly.

--client-cert-not-required
Don't require client certificate, client will authenticate using username/password only. Be aware that using this directive is less secure than requiring certificates from all clients.

If you use this directive, the entire responsibility of authentication will rest on your **--auth-user-pass-verify** script, so keep in mind that bugs in your script could potentially compromise the security of your VPN.

If you don't use this directive, but you also specify an **--auth-user-pass-verify** script, then OpenVPN will perform double authentication. The client certificate verification AND the **--auth-user-pass-verify** script will need to succeed in order for a client to be authenticated and accepted onto the VPN.

--username-as-common-name
For **--auth-user-pass-verify** authentication, use the authenticated username as the common name, rather than the common name from the client cert.

--compat-names [no-remapping] (DEPRECATED)
Until OpenVPN v2.3 the format of the X.509 Subject fields was formatted like this:
/C=US/L=Somewhere/CN=John Doe/emailAddress=john@example.com
In addition the old behaviour was to remap any character other than alphanumeric, underscore ('_'), dash ('-'), dot ('.'), and slash ('/') to underscore ('_'). The X.509 Subject string as returned by the **tls_id** environmental variable, could additionally contain colon (':') or equal ('=').
When using the **--compat-names** option, this old formatting and remapping will be re-enabled again. This is purely implemented for compatibility reasons when using older plug-ins or scripts which does not handle the new formatting or UTF-8 characters.
In OpenVPN v2.3 the formatting of these fields changed into a more standardised format. It now looks like:
C=US, L=Somewhere, CN=John Doe, emailAddress=john@example.com
The new default format in OpenVPN v2.3 also does not do the character remapping which happened earlier. This new format enables proper support for UTF-8 characters in the usernames, X.509 Subject fields and Common Name variables and it complies to the RFC 2253, UTF-8 String Representation of Distinguished Names.

The **no-remapping** mode flag can be used with the **--compat-names** option to be compatible with the now deprecated --no-name-remapping option. It is only available at the server. When this mode flag is used, the Common Name, Subject, and username strings are allowed to include any printable character including space, but excluding control characters such as tab, newline, and carriage-return. no-remapping is only available on the server side.

Please note: This option is immediately deprecated. It is only implemented to make the transition to the new formatting less intrusive. It will be removed either in OpenVPN v2.4 or v2.5. So please make sure you use the **--verify-x509-name** option instead of **--tls-remote** as soon as possible and update your scripts where necessary.

--no-name-remapping (DEPRECATED)
The **--no-name-remapping** option is an alias for **--compat-names no-remapping.** It ensures compatibility with server configurations using the **--no-name-remapping** option.

Please note: This option is now deprecated. It will be removed either in OpenVPN v2.4 or v2.5. So please make sure you support the new X.509 name formatting described with the **--compat-names** option as soon as possible.

--port-share host port [dir]
When run in TCP server mode, share the OpenVPN port with another application, such as an HTTPS server. If OpenVPN senses a connection to its port which is using a non-OpenVPN protocol, it will proxy the connection to the server at **host:port.** Currently only designed to work with HTTP/HTTPS, though it would be theoretically possible to extend to other protocols such as ssh.

dir specifies an optional directory where a temporary file with name N containing content C will be dynamically generated for each proxy connection, where N is the source IP:port of the client connection and C is the source IP:port of the connection to the proxy receiver. This directory can be used as a dictionary by the proxy receiver to determine the origin of the connection. Each generated file will be automatically deleted when the proxied connection is torn down.

Not implemented on Windows.

Client Mode

Use client mode when connecting to an OpenVPN server which has **--server, --server-bridge,** or

--mode server in it's configuration.

--client
A helper directive designed to simplify the configuration of OpenVPN's client mode. This directive is equivalent to:

```
pull
tls-client
```

--pull
This option must be used on a client which is connecting to a multi-client server. It indicates to OpenVPN that it should accept options pushed by the server, provided they are part of the legal set of pushable options (note that the **--pull** option is implied by **--client**).

In particular, **--pull** allows the server to push routes to the client, so you should not use **--pull** or **--client** in situations where you don't trust the server to have control over the client's routing table.

--auth-user-pass [up]
Authenticate with server using username/password. **up** is a file containing username/password on 2 lines (Note: OpenVPN will only read passwords from a file if it has been built with the --enable-password-save configure option, or on Windows by defining ENABLE_PASSWORD_SAVE in win/settings.in).

If **up** is omitted, username/password will be prompted from the console.

The server configuration must specify an **--auth-user-pass-verify** script to verify the username/password provided by the client.

--auth-retry type
Controls how OpenVPN responds to username/password verification errors such as the client-side response to an AUTH_FAILED message from the server or verification failure of the private key password.

Normally used to prevent auth errors from being fatal on the client side, and to permit username/password requeries in case of error.

An AUTH_FAILED message is generated by the server if the client fails **--auth-user-pass** authentication, or if the server-side **--client-connect** script returns an error status when the client tries to connect.

type can be one of:

none -- Client will exit with a fatal error (this is the default).
nointeract -- Client will retry the connection without requerying for an **--auth-user-pass** username/password. Use this option for unattended clients.
interact -- Client will requery for an **--auth-user-pass** username/password and/or private key password before attempting a reconnection.

Note that while this option cannot be pushed, it can be controlled from the management interface.

--static-challenge t e
Enable static challenge/response protocol using challenge text **t,** with echo flag given by **e** (0|1).

The echo flag indicates whether or not the user's response to the challenge should be echoed.

See management-notes.txt in the OpenVPN distribution for a description of the OpenVPN challenge/response protocol.

--server-poll-timeout n
when polling possible remote servers to connect to in a round-robin fashion, spend no more than **n** seconds waiting for a response before trying the next server. As this only makes sense in client-to-server setups, it cannot be used in point-to-point setups using **--secret** symmetrical key mode.

--explicit-exit-notify [n]
In UDP client mode or point-to-point mode, send server/peer an exit notification if tunnel is restarted or OpenVPN process is exited. In client mode, on exit/restart, this option will tell the server to immediately close its client instance object rather than waiting for a timeout. The **n** parameter (default=1) controls the maximum number of attempts that the client will try to resend the exit notification message. OpenVPN will not send any exit notifications unless this option is enabled.

Data Channel Encryption Options:

These options are meaningful for both Static & TLS-negotiated key modes (must be compatible between peers).

--secret file [direction]
Enable Static Key encryption mode (non-TLS). Use pre-shared secret **file** which was generated with **--genkey.**

The optional **direction** parameter enables the use of 4 distinct keys (HMAC-send, cipher-encrypt, HMAC-receive, cipher-decrypt), so that each data flow direction has a different set of HMAC and cipher keys. This has a number of desirable security properties including eliminating certain kinds of DoS and message replay attacks.

When the **direction** parameter is omitted, 2 keys are used bidirectionally, one for HMAC and the other for encryption/decryption.

The **direction** parameter should always be complementary on either side of the connection, i.e. one side should use "0" and the other should use "1", or both sides should omit it altogether.

The **direction** parameter requires that **file** contains a 2048 bit key. While pre-1.5 versions of OpenVPN generate 1024 bit key files, any version of OpenVPN which supports the **direction** parameter, will also support 2048 bit key file generation using the **--genkey** option.

Static key encryption mode has certain advantages, the primary being ease of configuration.

There are no certificates or certificate authorities or complicated negotiation handshakes and protocols. The only requirement is that you have a pre-existing secure channel with your peer (such as **ssh**) to initially copy the key. This requirement, along with the fact that your key never changes unless you manually generate a new one, makes it somewhat less secure than TLS mode (see below). If an attacker manages to steal your key, everything that was ever encrypted with it is compromised. Contrast that to the perfect forward secrecy features of TLS mode (using Diffie Hellman key exchange), where even if an attacker was able to steal your private key, he would gain no information to help him decrypt past sessions.

Another advantageous aspect of Static Key encryption mode is that it is a handshake-free protocol without any distinguishing signature or feature (such as a header or protocol handshake sequence) that would mark the ciphertext packets as being generated by OpenVPN. Anyone eavesdropping on the wire would see nothing but random-looking data.

--key-direction
Alternative way of specifying the optional direction parameter for the **--tls-auth** and **--secret** options. Useful when using inline files (See section on inline files).

--auth alg
Authenticate packets with HMAC using message digest algorithm **alg.** (The default is **SHA1**). HMAC is a commonly used message authentication algorithm (MAC) that uses a data string, a secure hash algorithm, and a key, to produce a digital signature.

OpenVPN's usage of HMAC is to first encrypt a packet, then HMAC the resulting ciphertext.

In static-key encryption mode, the HMAC key is included in the key file generated by **--genkey.** In TLS mode, the HMAC key is dynamically generated and shared between peers via the TLS control channel. If OpenVPN receives a packet with a bad HMAC it will drop the packet. HMAC usually adds 16 or 20 bytes per packet. Set **alg=none** to disable authentication.

For more information on HMAC see *http://www.cs.ucsd.edu/users/mihir/papers/hmac.html*

--cipher alg
Encrypt data channel packets with cipher algorithm **alg.** The default is **BF-CBC,** an abbreviation for Blowfish in Cipher Block Chaining mode. Blowfish has the advantages of being fast, very secure, and allowing key sizes of up to 448 bits. Blowfish is designed to be used in situations where keys are changed infrequently.

For more information on blowfish, see *http://www.counterpane.com/blowfish.html*

To see other ciphers that are available with OpenVPN, use the **--show-ciphers** option.

OpenVPN supports the CBC, CFB, and OFB cipher modes, however CBC is recommended and CFB and OFB should be considered advanced modes.

Set **alg=none** to disable encryption.

--keysize n
Size of cipher key in bits (optional). If unspecified, defaults to cipher-specific default. The **--show-ciphers** option (see below) shows all available OpenSSL ciphers, their default key sizes, and whether the key size can be changed. Use care in changing a cipher's default key size. Many ciphers have not been extensively cryptanalyzed with non-standard key lengths, and a larger key may offer no real guarantee of greater security, or may even reduce security.

--prng alg [nsl]

(Advanced) For PRNG (Pseudo-random number generator), use digest algorithm **alg** (default=sha1), and set **nsl** (default=16) to the size in bytes of the nonce secret length (between 16 and 64).

Set **alg=none** to disable the PRNG and use the OpenSSL RAND_bytes function instead for all of OpenVPN's pseudo-random number needs.

--engine [engine-name]
Enable OpenSSL hardware-based crypto engine functionality.

If **engine-name** is specified, use a specific crypto engine. Use the **--show-engines** standalone option to list the crypto engines which are supported by OpenSSL.

--no-replay
(Advanced) Disable OpenVPN's protection against replay attacks. Don't use this option unless you are prepared to make a tradeoff of greater efficiency in exchange for less security.

OpenVPN provides datagram replay protection by default.

Replay protection is accomplished by tagging each outgoing datagram with an identifier that is guaranteed to be unique for the key being used. The peer that receives the datagram will check for the uniqueness of the identifier. If the identifier was already received in a previous datagram, OpenVPN will drop the packet. Replay protection is important to defeat attacks such as a SYN flood attack, where the attacker listens in the wire, intercepts a TCP SYN packet (identifying it by the context in which it occurs in relation to other packets), then floods the receiving peer with copies of this packet.

OpenVPN's replay protection is implemented in slightly different ways, depending on the key management mode you have selected.

In Static Key mode or when using an CFB or OFB mode cipher, OpenVPN uses a 64 bit unique identifier that combines a time stamp with an incrementing sequence number.

When using TLS mode for key exchange and a CBC cipher mode, OpenVPN uses only a 32 bit sequence number without a time stamp, since OpenVPN can guarantee the uniqueness of this value for each key. As in IPSec, if the sequence number is close to wrapping back to zero, OpenVPN will trigger a new key exchange.

To check for replays, OpenVPN uses the *sliding window* algorithm used by IPSec.

--replay-window n [t]
Use a replay protection sliding-window of size **n** and a time window of **t** seconds.

By default **n** is 64 (the IPSec default) and **t** is 15 seconds.

This option is only relevant in UDP mode, i.e. when either **--proto udp** is specifed, or no **--proto** option is specified.

When OpenVPN tunnels IP packets over UDP, there is the possibility that packets might be dropped or delivered out of order. Because OpenVPN, like IPSec, is emulating the physical network layer, it will accept an out-of-order packet sequence, and will deliver such packets in the same order they were received to the TCP/IP protocol stack, provided they satisfy several constraints.

(a) The packet cannot be a replay (unless **--no-replay** is specified, which disables replay protection altogether).

(b) If a packet arrives out of order, it will only be accepted if the difference between its sequence number and the highest sequence number received so far is less than **n.**

(c) If a packet arrives out of order, it will only be accepted if it arrives no later than **t** seconds after any packet containing a higher sequence number.

If you are using a network link with a large pipeline (meaning that the product of bandwidth and latency is high), you may want to use a larger value for **n.** Satellite links in particular often require this.

If you run OpenVPN at **--verb 4,** you will see the message "Replay-window backtrack occurred [x]" every time the maximum sequence number backtrack seen thus far increases. This can be used to calibrate **n.**

There is some controversy on the appropriate method of handling packet reordering at the security layer.

Namely, to what extent should the security layer protect the encapsulated protocol from attacks which masquerade as the kinds of normal packet loss and reordering that occur over IP networks?

The IPSec and OpenVPN approach is to allow packet reordering within a certain fixed sequence

number window.

OpenVPN adds to the IPSec model by limiting the window size in time as well as sequence space.

OpenVPN also adds TCP transport as an option (not offered by IPSec) in which case OpenVPN can adopt a very strict attitude towards message deletion and reordering: Don't allow it. Since TCP guarantees reliability, any packet loss or reordering event can be assumed to be an attack.

In this sense, it could be argued that TCP tunnel transport is preferred when tunneling non-IP or UDP application protocols which might be vulnerable to a message deletion or reordering attack which falls within the normal operational parameters of IP networks.

So I would make the statement that one should never tunnel a non-IP protocol or UDP application protocol over UDP, if the protocol might be vulnerable to a message deletion or reordering attack that falls within the normal operating parameters of what is to be expected from the physical IP layer. The problem is easily fixed by simply using TCP as the VPN transport layer.

--mute-replay-warnings
Silence the output of replay warnings, which are a common false alarm on WiFi networks. This option preserves the security of the replay protection code without the verbosity associated with warnings about duplicate packets.

--replay-persist file
Persist replay-protection state across sessions using **file** to save and reload the state.

This option will strengthen protection against replay attacks, especially when you are using OpenVPN in a dynamic context (such as with **--inetd)** when OpenVPN sessions are frequently started and stopped.

This option will keep a disk copy of the current replay protection state (i.e. the most recent packet timestamp and sequence number received from the remote peer), so that if an OpenVPN session is stopped and restarted, it will reject any replays of packets which were already received by the prior session.

This option only makes sense when replay protection is enabled (the default) and you are using either **--secret** (shared-secret key mode) or TLS mode with **--tls-auth.**

--no-iv
(Advanced) Disable OpenVPN's use of IV (cipher initialization vector). Don't use this option unless you are prepared to make a tradeoff of greater efficiency in exchange for less security.

OpenVPN uses an IV by default, and requires it for CFB and OFB cipher modes (which are totally insecure without it). Using an IV is important for security when multiple messages are being encrypted/decrypted with the same key.

IV is implemented differently depending on the cipher mode used.

In CBC mode, OpenVPN uses a pseudo-random IV for each packet.

In CFB/OFB mode, OpenVPN uses a unique sequence number and time stamp as the IV. In fact, in CFB/OFB mode, OpenVPN uses a datagram space-saving optimization that uses the unique identifier for datagram replay protection as the IV.

--use-prediction-resistance
Enable prediction resistance on PolarSSL's RNG.

Enabling prediction resistance causes the RNG to reseed in each call for random. Reseeding this often can quickly deplete the kernel entropy pool.

If you need this option, please consider running a daemon that adds entropy to the kernel pool.

Note that this option only works with PolarSSL versions greater than 1.1.

--test-crypto
Do a self-test of OpenVPN's crypto options by encrypting and decrypting test packets using the data channel encryption options specified above. This option does not require a peer to function, and therefore can be specified without **--dev** or **--remote.**

The typical usage of **--test-crypto** would be something like this:

openvpn --test-crypto --secret key

or

openvpn --test-crypto --secret key --verb 9

This option is very useful to test OpenVPN after it has been ported to a new platform, or to isolate problems in the compiler, OpenSSL crypto library, or OpenVPN's crypto code. Since it is a self-test mode, problems with encryption and authentication can be debugged independently of network

and tunnel issues.

TLS Mode Options:

TLS mode is the most powerful crypto mode of OpenVPN in both security and flexibility. TLS mode works by establishing control and data channels which are multiplexed over a single TCP/UDP port. OpenVPN initiates a TLS session over the control channel and uses it to exchange cipher and HMAC keys to protect the data channel. TLS mode uses a robust reliability layer over the UDP connection for all control channel communication, while the data channel, over which encrypted tunnel data passes, is forwarded without any mediation. The result is the best of both worlds: a fast data channel that forwards over UDP with only the overhead of encrypt, decrypt, and HMAC functions, and a control channel that provides all of the security features of TLS, including certificate-based authentication and Diffie Hellman forward secrecy.

To use TLS mode, each peer that runs OpenVPN should have its own local certificate/key pair (**--cert** and **--key**), signed by the root certificate which is specified in **--ca.**

When two OpenVPN peers connect, each presents its local certificate to the other. Each peer will then check that its partner peer presented a certificate which was signed by the master root certificate as specified in **--ca.**

If that check on both peers succeeds, then the TLS negotiation will succeed, both OpenVPN peers will exchange temporary session keys, and the tunnel will begin passing data.

The OpenVPN distribution contains a set of scripts for managing RSA certificates & keys, located in the *easy-rsa* subdirectory.

The easy-rsa package is also rendered in web form here: *http://openvpn.net/easyrsa.html*

--tls-server
Enable TLS and assume server role during TLS handshake. Note that OpenVPN is designed as a peer-to-peer application. The designation of client or server is only for the purpose of negotiating the TLS control channel.

--tls-client
Enable TLS and assume client role during TLS handshake.

--ca file
Certificate authority (CA) file in .pem format, also referred to as the *root* certificate. This file can have multiple certificates in .pem format, concatenated together. You can construct your own certificate authority certificate and private key by using a command such as:

openssl req -nodes -new -x509 -keyout ca.key -out ca.crt

Then edit your openssl.cnf file and edit the **certificate** variable to point to your new root certificate **ca.crt.**

For testing purposes only, the OpenVPN distribution includes a sample CA certificate (ca.crt). Of course you should never use the test certificates and test keys distributed with OpenVPN in a production environment, since by virtue of the fact that they are distributed with OpenVPN, they are totally insecure.

--capath dir
Directory containing trusted certificates (CAs and CRLs). Available with OpenSSL version >= 0.9.7 dev. Not available with PolarSSL.

When using the **--capath** option, you are required to supply valid CRLs for the CAs too. CAs in the capath directory are expected to be named <hash>.<n>. CRLs are expected to be named <hash>.r<n>. See the **-CApath** option of **openssl verify** , and the **-hash** option of **openssl x509** and **openssl crl** for more information.

--dh file
File containing Diffie Hellman parameters in .pem format (required for **--tls-server** only). Use

openssl dhparam -out dh1024.pem 1024

to generate your own, or use the existing dh1024.pem file included with the OpenVPN distribution. Diffie Hellman parameters may be considered public.

--cert file
Local peer's signed certificate in .pem format -- must be signed by a certificate authority whose certificate is in **--ca file.** Each peer in an OpenVPN link running in TLS mode should have its own certificate and private key file. In addition, each certificate should have been signed by the key of a certificate authority whose public key resides in the **--ca** certificate authority file. You can easily make your own certificate authority (see above) or pay money to use a commercial service such as thawte.com (in which case you will be helping to finance the world's second space tourist :). To

generate a certificate, you can use a command such as:

openssl req -nodes -new -keyout mycert.key -out mycert.csr

If your certificate authority private key lives on another machine, copy the certificate signing request (mycert.csr) to this other machine (this can be done over an insecure channel such as email). Now sign the certificate with a command such as:

openssl ca -out mycert.crt -in mycert.csr

Now copy the certificate (mycert.crt) back to the peer which initially generated the .csr file (this can be over a public medium). Note that the **openssl ca** command reads the location of the certificate authority key from its configuration file such as **/usr/share/ssl/openssl.cnf** -- note also that for certificate authority functions, you must set up the files **index.txt** (may be empty) and **serial** (initialize to **01**).

--extra-certs file
Specify a **file** containing one or more PEM certs (concatenated together) that complete the local certificate chain.

This option is useful for "split" CAs, where the CA for server certs is different than the CA for client certs. Putting certs in this file allows them to be used to complete the local certificate chain without trusting them to verify the peer-submitted certificate, as would be the case if the certs were placed in the **ca** file.

--key file
Local peer's private key in .pem format. Use the private key which was generated when you built your peer's certificate (see **--cert file** above).

--tls-version-min version ['or-highest']
Enable TLS version negotiation, and set the minimum TLS version we will accept from the peer (default is "1.0"). Examples for version include "1.0", "1.1", or "1.2". If 'or-highest' is specified and version is not recognized, we will only accept the highest TLS version supported by the local SSL implementation.

Also see **--tls-version-max** below, for information on compatibility.

--tls-version-max version
Set the maximum TLS version we will use (default is the highest version supported). Examples for version include "1.0", "1.1", or "1.2".

If and only if this is set to 1.0, and OpenSSL is used (not PolarSSL), then OpenVPN will set up OpenSSL to use a fixed TLSv1 handshake. All other configurations will autonegotiate in the given limits, and the choice of handshake versions is left to the SSL implementation.

--pkcs12 file
Specify a PKCS #12 file containing local private key, local certificate, and root CA certificate. This option can be used instead of **--ca, --cert,** and **--key.** Not available with PolarSSL.

--verify-hash hash
Specify SHA1 fingerprint for level-1 cert. The level-1 cert is the CA (or intermediate cert) that signs the leaf certificate, and is one removed from the leaf certificate in the direction of the root. When accepting a connection from a peer, the level-1 cert fingerprint must match **hash** or certificate verification will fail. Hash is specified as XX:XX:... For example: AD:B0:95:D8:09:C8:36:45:12:A9:89:C8:90:09:CB:13:72:A6:AD:16

--pkcs11-cert-private [0|1]...
Set if access to certificate object should be performed after login. Every provider has its own setting.

--pkcs11-id name
Specify the serialized certificate id to be used. The id can be gotten by the standalone **--show-pkcs11-ids** option.

--pkcs11-id-management
Acquire PKCS#11 id from management interface. In this case a NEED-STR 'pkcs11-id-request' real-time message will be triggered, application may use pkcs11-id-count command to retrieve available number of certificates, and pkcs11-id-get command to retrieve certificate id and certificate body.

--pkcs11-pin-cache seconds
Specify how many seconds the PIN can be cached, the default is until the token is removed.

--pkcs11-protected-authentication [0|1]...
Use PKCS#11 protected authentication path, useful for biometric and external keypad devices. Every provider has its own setting.

--pkcs11-providers provider...
Specify a RSA Security Inc. PKCS #11 Cryptographic Token Interface (Cryptoki) providers to load. This option can be used instead of **--cert, --key,** and **--pkcs12.**

If p11-kit is present on the system, its **p11-kit-proxy.so** module will be loaded by default if either the **--pkcs11-id** or **--pkcs11-id-management** options are specified without **--pkcs11-provider** being given.

--pkcs11-private-mode mode...
Specify which method to use in order to perform private key operations. A different mode can be specified for each provider. Mode is encoded as hex number, and can be a mask one of the following:

0 (default) -- Try to determine automatically.
1 -- Use sign.
2 -- Use sign recover.
4 -- Use decrypt.
8 -- Use unwrap.

--cryptoapicert select-string
Load the certificate and private key from the Windows Certificate System Store (Windows/OpenSSL Only).

Use this option instead of **--cert** and **--key.**

This makes it possible to use any smart card, supported by Windows, but also any kind of certificate, residing in the Cert Store, where you have access to the private key. This option has been tested with a couple of different smart cards (GemSAFE, Cryptoflex, and Swedish Post Office eID) on the client side, and also an imported PKCS12 software certificate on the server side.

To select a certificate, based on a substring search in the certificate's subject:

cryptoapicert "SUBJ:Peter Runestig"

To select a certificate, based on certificate's thumbprint:

cryptoapicert "THUMB:f6 49 24 41 01 b4 ..."

The thumbprint hex string can easily be copy-and-pasted from the Windows Certificate Store GUI.

--key-method m
Use data channel key negotiation method **m.** The key method must match on both sides of the connection.

After OpenVPN negotiates a TLS session, a new set of keys for protecting the tunnel data channel is generated and exchanged over the TLS session.

In method 1 (the default for OpenVPN 1.x), both sides generate random encrypt and HMAC-send keys which are forwarded to the other host over the TLS channel.

In method 2, (the default for OpenVPN 2.0) the client generates a random key. Both client and server also generate some random seed material. All key source material is exchanged over the TLS channel. The actual keys are generated using the TLS PRF function, taking source entropy from both client and server. Method 2 is designed to closely parallel the key generation process used by TLS 1.0.

Note that in TLS mode, two separate levels of keying occur:

(1) The TLS connection is initially negotiated, with both sides of the connection producing certificates and verifying the certificate (or other authentication info provided) of the other side. The **--key-method** parameter has no effect on this process.

(2) After the TLS connection is established, the tunnel session keys are separately negotiated over the existing secure TLS channel. Here, **--key-method** determines the derivation of the tunnel session keys.

--tls-cipher l
A list **l** of allowable TLS ciphers delimited by a colon (":").

This setting can be used to ensure that certain cipher suites are used (or not used) for the TLS connection. OpenVPN uses TLS to secure the control channel, over which the keys that are used to protect the actual VPN traffic are exchanged.

The supplied list of ciphers is (after potential OpenSSL/IANA name translation) simply supplied to the crypto library. Please see the OpenSSL and/or PolarSSL documentation for details on the cipher list interpretation.

Use **--show-tls** to see a list of TLS ciphers supported by your crypto library.

Warning! **--tls-cipher** is an expert feature, which - if used correcly - can improve the security of your VPN connection. But it is also easy to unwittingly use it to carefully align a gun with your foot, or just break your connection. Use with care!

The default for --tls-cipher is to use PolarSSL's default cipher list when using PolarSSL or "DEFAULT:!EXP:!PSK:!SRP:!kRSA" when using OpenSSL.

--tls-timeout n
Packet retransmit timeout on TLS control channel if no acknowledgment from remote within **n** seconds (default=2). When OpenVPN sends a control packet to its peer, it will expect to receive an acknowledgement within **n** seconds or it will retransmit the packet, subject to a TCP-like exponential backoff algorithm. This parameter only applies to control channel packets. Data channel packets (which carry encrypted tunnel data) are never acknowledged, sequenced, or retransmitted by OpenVPN because the higher level network protocols running on top of the tunnel such as TCP expect this role to be left to them.

--reneg-bytes n
Renegotiate data channel key after **n** bytes sent or received (disabled by default). OpenVPN allows the lifetime of a key to be expressed as a number of bytes encrypted/decrypted, a number of packets, or a number of seconds. A key renegotiation will be forced if any of these three criteria are met by either peer.

--reneg-pkts n
Renegotiate data channel key after **n** packets sent and received (disabled by default).

--reneg-sec n
Renegotiate data channel key after **n** seconds (default=3600).

When using dual-factor authentication, note that this default value may cause the end user to be challenged to reauthorize once per hour.

Also, keep in mind that this option can be used on both the client and server, and whichever uses the lower value will be the one to trigger the renegotiation. A common mistake is to set **--reneg-sec** to a higher value on either the client or server, while the other side of the connection is still using the default value of 3600 seconds, meaning that the renegotiation will still occur once per 3600 seconds. The solution is to increase --reneg-sec on both the client and server, or set it to 0 on one side of the connection (to disable), and to your chosen value on the other side.

--hand-window n
Handshake Window -- the TLS-based key exchange must finalize within **n** seconds of handshake initiation by any peer (default = 60 seconds). If the handshake fails we will attempt to reset our connection with our peer and try again. Even in the event of handshake failure we will still use our expiring key for up to **--tran-window** seconds to maintain continuity of transmission of tunnel data.

--tran-window n
Transition window -- our old key can live this many seconds after a new a key renegotiation begins (default = 3600 seconds). This feature allows for a graceful transition from old to new key, and removes the key renegotiation sequence from the critical path of tunnel data forwarding.

--single-session
After initially connecting to a remote peer, disallow any new connections. Using this option means that a remote peer cannot connect, disconnect, and then reconnect.

If the daemon is reset by a signal or **--ping-restart,** it will allow one new connection.

--single-session can be used with **--ping-exit** or **--inactive** to create a single dynamic session that will exit when finished.

--tls-exit
Exit on TLS negotiation failure.

--tls-auth file [direction]
Add an additional layer of HMAC authentication on top of the TLS control channel to protect against DoS attacks.

In a nutshell, **--tls-auth** enables a kind of "HMAC firewall" on OpenVPN's TCP/UDP port, where TLS control channel packets bearing an incorrect HMAC signature can be dropped immediately without response.

file (required) is a key file which can be in one of two formats:

(1) An OpenVPN static key file generated by **--genkey** (required if **direction** parameter is used).

(2) DEPRECATED A freeform passphrase file. In this case the HMAC key will be derived by taking a secure hash of this file, similar to the **md5sum**(1) or **sha1sum**(1) commands. This option is deprecated and will stop working in OpenVPN 2.4 and newer releases.

OpenVPN will first try format (1), and if the file fails to parse as a static key file, format (2) will be used.

See the **--secret** option for more information on the optional **direction** parameter.

--tls-auth is recommended when you are running OpenVPN in a mode where it is listening for packets from any IP address, such as when **--remote** is not specified, or **--remote** is specified with **--float.**

The rationale for this feature is as follows. TLS requires a multi-packet exchange before it is able to authenticate a peer. During this time before authentication, OpenVPN is allocating resources

(memory and CPU) to this potential peer. The potential peer is also exposing many parts of OpenVPN and the OpenSSL library to the packets it is sending. Most successful network attacks today seek to either exploit bugs in programs (such as buffer overflow attacks) or force a program to consume so many resources that it becomes unusable. Of course the first line of defense is always to produce clean, well-audited code. OpenVPN has been written with buffer overflow attack prevention as a top priority. But as history has shown, many of the most widely used network applications have, from time to time, fallen to buffer overflow attacks.

So as a second line of defense, OpenVPN offers this special layer of authentication on top of the TLS control channel so that every packet on the control channel is authenticated by an HMAC signature and a unique ID for replay protection. This signature will also help protect against DoS (Denial of Service) attacks. An important rule of thumb in reducing vulnerability to DoS attacks is to minimize the amount of resources a potential, but as yet unauthenticated, client is able to consume.

--tls-auth does this by signing every TLS control channel packet with an HMAC signature, including packets which are sent before the TLS level has had a chance to authenticate the peer. The result is that packets without the correct signature can be dropped immediately upon reception, before they have a chance to consume additional system resources such as by initiating a TLS handshake. **--tls-auth** can be strengthened by adding the **--replay-persist** option which will keep OpenVPN's replay protection state in a file so that it is not lost across restarts.

It should be emphasized that this feature is optional and that the passphrase/key file used with **--tls-auth** gives a peer nothing more than the power to initiate a TLS handshake. It is not used to encrypt or authenticate any tunnel data.

--askpass [file]
Get certificate password from console or **file** before we daemonize.

For the extremely security conscious, it is possible to protect your private key with a password. Of course this means that every time the OpenVPN daemon is started you must be there to type the password. The **--askpass** option allows you to start OpenVPN from the command line. It will query you for a password before it daemonizes. To protect a private key with a password you should omit the **-nodes** option when you use the **openssl** command line tool to manage certificates and private keys.

If **file** is specified, read the password from the first line of **file.** Keep in mind that storing your password in a file to a certain extent invalidates the extra security provided by using an encrypted key (Note: OpenVPN will only read passwords from a file if it has been built with the --enable-password-save configure option, or on Windows by defining ENABLE_PASSWORD_SAVE in win/settings.in).

--auth-nocache
Don't cache **--askpass** or **--auth-user-pass** username/passwords in virtual memory.

If specified, this directive will cause OpenVPN to immediately forget username/password inputs after they are used. As a result, when OpenVPN needs a username/password, it will prompt for input from stdin, which may be multiple times during the duration of an OpenVPN session.

When using --auth-nocache in combination with a user/password file and --chroot or --daemon, make sure to use an absolute path.

This directive does not affect the **--http-proxy** username/password. It is always cached.

--tls-verify cmd
Run command **cmd** to verify the X509 name of a pending TLS connection that has otherwise passed all other tests of certification (except for revocation via **--crl-verify** directive; the revocation test occurs after the **--tls-verify** test).

cmd should return 0 to allow the TLS handshake to proceed, or 1 to fail.

cmd consists of a path to script (or executable program), optionally followed by arguments. The path and arguments may be single- or double-quoted and/or escaped using a backslash, and should be separated by one or more spaces.

When **cmd** is executed two arguments are appended after any arguments specified in **cmd** , as follows:

cmd certificate_depth subject

These arguments are, respectively, the current certificate depth and the X509 common name (cn) of the peer.

This feature is useful if the peer you want to trust has a certificate which was signed by a certificate authority who also signed many other certificates, where you don't necessarily want to trust all of them, but rather be selective about which peer certificate you will accept. This feature

allows you to write a script which will test the X509 name on a certificate and decide whether or not it should be accepted. For a simple perl script which will test the common name field on the certificate, see the file **verify-cn** in the OpenVPN distribution.

See the "Environmental Variables" section below for additional parameters passed as environmental variables.

--tls-export-cert directory
Store the certificates the clients uses upon connection to this directory. This will be done before --tls-verify is called. The certificates will use a temporary name and will be deleted when the tls-verify script returns. The file name used for the certificate is available via the peer_cert environment variable.

--x509-username-field [ext:]fieldname
Field in the X.509 certificate subject to be used as the username (default=CN). Typically, this option is specified with **fieldname** as either of the following:

--x509-username-field emailAddress
--x509-username-field ext:subjectAltName

The first example uses the value of the "emailAddress" attribute in the certificate's Subject field as the username. The second example uses the **ext:** prefix to signify that the X.509 extension **fieldname** "subjectAltName" be searched for an rfc822Name (email) field to be used as the username. In cases where there are multiple email addresses in **ext:fieldname**, the last occurrence is chosen.

When this option is used, the **--verify-x509-name** option will match against the chosen **fieldname** instead of the Common Name.

Please note: This option has a feature which will convert an all-lowercase **fieldname** to uppercase characters, e.g., ou -> OU. A mixed-case **fieldname** or one having the **ext:** prefix will be left as-is. This automatic upcasing feature is deprecated and will be removed in a future release.

--tls-remote name (DEPRECATED)
Accept connections only from a host with X509 name or common name equal to **name.** The remote host must also pass all other tests of verification.

NOTE: Because tls-remote may test against a common name prefix, only use this option when you are using OpenVPN with a custom CA certificate that is under your control. Never use this option when your client certificates are signed by a third party, such as a commercial web CA.

Name can also be a common name prefix, for example if you want a client to only accept connections to "Server-1", "Server-2", etc., you can simply use **--tls-remote Server**

Using a common name prefix is a useful alternative to managing a CRL (Certificate Revocation List) on the client, since it allows the client to refuse all certificates except for those associated with designated servers.

--tls-remote is a useful replacement for the **--tls-verify** option to verify the remote host, because **--tls-remote** works in a **--chroot** environment too.

Please also note: This option is now deprecated. It will be removed either in OpenVPN v2.4 or v2.5. So please make sure you support the new X.509 name formatting described with the **--compat-names** option as soon as possible by updating your configurations to use **--verify-x509-name** instead.

--verify-x509-name name type
Accept connections only if a host's X.509 name is equal to **name.** The remote host must also pass all other tests of verification.

Which X.509 name is compared to **name** depends on the setting of type. **type** can be "subject" to match the complete subject DN (default), "name" to match a subject RDN or "name-prefix" to match a subject RDN prefix. Which RDN is verified as name depends on the **--x509-username-field** option. But it defaults to the common name (CN), e.g. a certificate with a subject DN "C=KG, ST=NA, L=Bishkek, CN=Server-1" would be matched by:

--verify-x509-name 'C=KG, ST=NA, L=Bishkek, CN=Server-1' and **--verify-x509-name Server-1 name** or you could use **--verify-x509-name Server- name-prefix** if you want a client to only accept connections to "Server-1", "Server-2", etc.

--verify-x509-name is a useful replacement for the **--tls-verify** option to verify the remote host, because **--verify-x509-name** works in a **--chroot** environment without any dependencies.

Using a name prefix is a useful alternative to managing a CRL (Certificate Revocation List) on the client, since it allows the client to refuse all certificates except for those associated with designated servers.

NOTE: Test against a name prefix only when you are using OpenVPN with a custom CA certificate that is under your control. Never use this option with type "name-prefix" when your client certificates are signed by a third party, such as a commercial web CA.

--x509-track attribute
Save peer X509 **attribute** value in environment for use by plugins and management interface. Prepend a '+' to **attribute** to save values from full cert chain. Values will be encoded as X509_<depth>_<attribute>=<value>. Multiple **--x509-track** options can be defined to track multiple attributes. Not available with PolarSSL.

--ns-cert-type client|server
Require that peer certificate was signed with an explicit **nsCertType** designation of "client" or "server".

This is a useful security option for clients, to ensure that the host they connect with is a designated server.

See the easy-rsa/build-key-server script for an example of how to generate a certificate with the **nsCertType** field set to "server".

If the server certificate's nsCertType field is set to "server", then the clients can verify this with **--ns-cert-type server.**

This is an important security precaution to protect against a man-in-the-middle attack where an authorized client attempts to connect to another client by impersonating the server. The attack is easily prevented by having clients verify the server certificate using any one of **--ns-cert-type, --verify-x509-name,** or **--tls-verify.**

--remote-cert-ku v...
Require that peer certificate was signed with an explicit **key usage.**

This is a useful security option for clients, to ensure that the host they connect to is a designated server.

The key usage should be encoded in hex, more than one key usage can be specified.

--remote-cert-eku oid
Require that peer certificate was signed with an explicit **extended key usage.**

This is a useful security option for clients, to ensure that the host they connect to is a designated server.

The extended key usage should be encoded in oid notation, or OpenSSL symbolic representation.

--remote-cert-tls client|server
Require that peer certificate was signed with an explicit **key usage** and **extended key usage** based on RFC3280 TLS rules.

This is a useful security option for clients, to ensure that the host they connect to is a designated server.

The **--remote-cert-tls client** option is equivalent to **--remote-cert-ku 80 08 88 --remote-cert-eku "TLS Web Client Authentication"**

The key usage is digitalSignature and/or keyAgreement.

The **--remote-cert-tls server** option is equivalent to **--remote-cert-ku a0 88 --remote-cert-eku "TLS Web Server Authentication"**

The key usage is digitalSignature and (keyEncipherment or keyAgreement).

This is an important security precaution to protect against a man-in-the-middle attack where an authorized client attempts to connect to another client by impersonating the server. The attack is easily prevented by having clients verify the server certificate using any one of **--remote-cert-tls, --verify-x509-name,** or **--tls-verify.**

--crl-verify crl ['dir']
Check peer certificate against the file **crl** in PEM format.

A CRL (certificate revocation list) is used when a particular key is compromised but when the overall PKI is still intact.

Suppose you had a PKI consisting of a CA, root certificate, and a number of client certificates. Suppose a laptop computer containing a client key and certificate was stolen. By adding the stolen certificate to the CRL file, you could reject any connection which attempts to use it, while preserving the overall integrity of the PKI.

The only time when it would be necessary to rebuild the entire PKI from scratch would be if the

root certificate key itself was compromised.

If the optional **dir** flag is specified, enable a different mode where **crl** is a directory containing files named as revoked serial numbers (the files may be empty, the contents are never read). If a client requests a connection, where the client certificate serial number (decimal string) is the name of a file present in the directory, it will be rejected.

Note: As the crl file (or directory) is read every time a peer connects, if you are dropping root privileges with **--user,** make sure that this user has sufficient privileges to read the file.

SSL Library information:

--show-ciphers
(Standalone) Show all cipher algorithms to use with the **--cipher** option.
--show-digests
(Standalone) Show all message digest algorithms to use with the **--auth** option.
--show-tls
(Standalone) Show all TLS ciphers supported by the crypto library. OpenVPN uses TLS to secure the control channel, over which the keys that are used to protect the actual VPN traffic are exchanged. The TLS ciphers will be sorted from highest preference (most secure) to lowest.

Be aware that whether a cipher suite in this list can actually work depends on the specific setup of both peers (e.g. both peers must support the cipher, and an ECDSA cipher suite will not work if you are using an RSA certificate, etc.).

--show-engines
(Standalone) Show currently available hardware-based crypto acceleration engines supported by the OpenSSL library.

Generate a random key:

Used only for non-TLS static key encryption mode.

--genkey
(Standalone) Generate a random key to be used as a shared secret, for use with the **--secret** option. This file must be shared with the peer over a pre-existing secure channel such as **scp**(1)
--secret file
Write key to **file.**

TUN/TAP persistent tunnel config mode:

Available with linux 2.4.7+. These options comprise a standalone mode of OpenVPN which can be used to create and delete persistent tunnels.

--mktun
(Standalone) Create a persistent tunnel on platforms which support them such as Linux. Normally TUN/TAP tunnels exist only for the period of time that an application has them open. This option takes advantage of the TUN/TAP driver's ability to build persistent tunnels that live through multiple instantiations of OpenVPN and die only when they are deleted or the machine is rebooted.

One of the advantages of persistent tunnels is that they eliminate the need for separate **--up** and **--down** scripts to run the appropriate **ifconfig**(8) and **route**(8) commands. These commands can be placed in the the same shell script which starts or terminates an OpenVPN session.

Another advantage is that open connections through the TUN/TAP-based tunnel will not be reset if the OpenVPN peer restarts. This can be useful to provide uninterrupted connectivity through the tunnel in the event of a DHCP reset of the peer's public IP address (see the **--ipchange** option above).

One disadvantage of persistent tunnels is that it is harder to automatically configure their MTU value (see **--link-mtu** and **--tun-mtu** above).

On some platforms such as Windows, TAP-Win32 tunnels are persistent by default.

--rmtun
(Standalone) Remove a persistent tunnel.
--dev tunX | tapX
TUN/TAP device
--user user

Optional user to be owner of this tunnel.

--group group
Optional group to be owner of this tunnel.

Windows-Specific Options:

--win-sys path
Set the Windows system directory pathname to use when looking for system executables such as **route.exe** and **netsh.exe.** By default, if this directive is not specified, OpenVPN will use the SystemRoot environment variable.

This option have changed behaviour in OpenVPN 2.3. Earlier you had to define **--win-sys env** to use the SystemRoot environment variable, otherwise it defaulted to C:\WINDOWS. It is not needed to use the **env** keyword any more, and it will just be ignored. A warning is logged when this is found in the configuration file.

--ip-win32 method
When using **--ifconfig** on Windows, set the TAP-Win32 adapter IP address and netmask using **method.** Don't use this option unless you are also using **--ifconfig.**

manual -- Don't set the IP address or netmask automatically. Instead output a message to the console telling the user to configure the adapter manually and indicating the IP/netmask which OpenVPN expects the adapter to be set to.

dynamic [offset] [lease-time] -- Automatically set the IP address and netmask by replying to DHCP query messages generated by the kernel. This mode is probably the "cleanest" solution for setting the TCP/IP properties since it uses the well-known DHCP protocol. There are, however, two prerequisites for using this mode: (1) The TCP/IP properties for the TAP-Win32 adapter must be set to "Obtain an IP address automatically," and (2) OpenVPN needs to claim an IP address in the subnet for use as the virtual DHCP server address. By default in **--dev tap** mode, OpenVPN will take the normally unused first address in the subnet. For example, if your subnet is 192.168.4.0 netmask 255.255.255.0, then OpenVPN will take the IP address 192.168.4.0 to use as the virtual DHCP server address. In **--dev tun** mode, OpenVPN will cause the DHCP server to masquerade as if it were coming from the remote endpoint. The optional offset parameter is an integer which is > -256 and < 256 and which defaults to 0. If offset is positive, the DHCP server will masquerade as the IP address at network address + offset. If offset is negative, the DHCP server will masquerade as the IP address at broadcast address + offset. The Windows **ipconfig /all** command can be used to show what Windows thinks the DHCP server address is. OpenVPN will "claim" this address, so make sure to use a free address. Having said that, different OpenVPN instantiations, including different ends of the same connection, can share the same virtual DHCP server address. The **lease-time** parameter controls the lease time of the DHCP assignment given to the TAP-Win32 adapter, and is denoted in seconds. Normally a very long lease time is preferred because it prevents routes involving the TAP-Win32 adapter from being lost when the system goes to sleep. The default lease time is one year.

netsh -- Automatically set the IP address and netmask using the Windows command-line "netsh" command. This method appears to work correctly on Windows XP but not Windows 2000.

ipapi -- Automatically set the IP address and netmask using the Windows IP Helper API. This approach does not have ideal semantics, though testing has indicated that it works okay in practice. If you use this option, it is best to leave the TCP/IP properties for the TAP-Win32 adapter in their default state, i.e. "Obtain an IP address automatically."

adaptive -- (Default) Try **dynamic** method initially and fail over to **netsh** if the DHCP negotiation with the TAP-Win32 adapter does not succeed in 20 seconds. Such failures have been known to occur when certain third-party firewall packages installed on the client machine block the DHCP negotiation used by the TAP-Win32 adapter. Note that if the **netsh** failover occurs, the TAP-Win32 adapter TCP/IP properties will be reset from DHCP to static, and this will cause future OpenVPN startups using the **adaptive** mode to use **netsh** immediately, rather than trying **dynamic** first. To "unstick" the **adaptive** mode from using **netsh,** run OpenVPN at least once using the **dynamic** mode to restore the TAP-Win32 adapter TCP/IP properties to a DHCP configuration.

--route-method m
Which method **m** to use for adding routes on Windows?

adaptive (default) -- Try IP helper API first. If that fails, fall back to the route.exe shell command.
ipapi -- Use IP helper API.
exe -- Call the route.exe shell command.

--dhcp-option type [parm]
Set extended TAP-Win32 TCP/IP properties, must be used with **--ip-win32 dynamic** or **--ip-win32 adaptive.** This option can be used to set additional TCP/IP properties on the TAP-Win32 adapter, and is particularly useful for configuring an OpenVPN client to access a Samba server across the

VPN.

DOMAIN name -- Set Connection-specific DNS Suffix.

DNS addr -- Set primary domain name server address. Repeat this option to set secondary DNS server addresses.

WINS addr -- Set primary WINS server address (NetBIOS over TCP/IP Name Server). Repeat this option to set secondary WINS server addresses.

NBDD addr -- Set primary NBDD server address (NetBIOS over TCP/IP Datagram Distribution Server) Repeat this option to set secondary NBDD server addresses.

NTP addr -- Set primary NTP server address (Network Time Protocol). Repeat this option to set secondary NTP server addresses.

NBT type -- Set NetBIOS over TCP/IP Node type. Possible options: **1** = b-node (broadcasts), **2** = p-node (point-to-point name queries to a WINS server), **4** = m-node (broadcast then query name server), and **8** = h-node (query name server, then broadcast).

NBS scope-id -- Set NetBIOS over TCP/IP Scope. A NetBIOS Scope ID provides an extended naming service for the NetBIOS over TCP/IP (Known as NBT) module. The primary purpose of a NetBIOS scope ID is to isolate NetBIOS traffic on a single network to only those nodes with the same NetBIOS scope ID. The NetBIOS scope ID is a character string that is appended to the NetBIOS name. The NetBIOS scope ID on two hosts must match, or the two hosts will not be able to communicate. The NetBIOS Scope ID also allows computers to use the same computer name, as they have different scope IDs. The Scope ID becomes a part of the NetBIOS name, making the name unique. (This description of NetBIOS scopes courtesy of NeonSurge@abyss.com)

DISABLE-NBT -- Disable Netbios-over-TCP/IP.

Note that if **--dhcp-option** is pushed via **--push** to a non-windows client, the option will be saved in the client's environment before the up script is called, under the name "foreign_option_{n}".

--tap-sleep n
Cause OpenVPN to sleep for **n** seconds immediately after the TAP-Win32 adapter state is set to "connected".

This option is intended to be used to troubleshoot problems with the **--ifconfig** and **--ip-win32** options, and is used to give the TAP-Win32 adapter time to come up before Windows IP Helper API operations are applied to it.

--show-net-up
Output OpenVPN's view of the system routing table and network adapter list to the syslog or log file after the TUN/TAP adapter has been brought up and any routes have been added.

--dhcp-renew
Ask Windows to renew the TAP adapter lease on startup. This option is normally unnecessary, as Windows automatically triggers a DHCP renegotiation on the TAP adapter when it comes up, however if you set the TAP-Win32 adapter Media Status property to "Always Connected", you may need this flag.

--dhcp-release
Ask Windows to release the TAP adapter lease on shutdown. This option has the same caveats as **--dhcp-renew** above.

--register-dns
Run net stop dnscache, net start dnscache, ipconfig /flushdns and ipconfig /registerdns on connection initiation. This is known to kick Windows into recognizing pushed DNS servers.

--pause-exit
Put up a "press any key to continue" message on the console prior to OpenVPN program exit. This option is automatically used by the Windows explorer when OpenVPN is run on a configuration file using the right-click explorer menu.

--service exit-event [0|1]
Should be used when OpenVPN is being automatically executed by another program in such a context that no interaction with the user via display or keyboard is possible. In general, end-users should never need to explicitly use this option, as it is automatically added by the OpenVPN service wrapper when a given OpenVPN configuration is being run as a service.

exit-event is the name of a Windows global event object, and OpenVPN will continuously monitor the state of this event object and exit when it becomes signaled.

The second parameter indicates the initial state of **exit-event** and normally defaults to 0.

Multiple OpenVPN processes can be simultaneously executed with the same **exit-event** parameter. In any case, the controlling process can signal **exit-event,** causing all such OpenVPN processes to exit.

When executing an OpenVPN process using the **--service** directive, OpenVPN will probably not

have a console window to output status/error messages, therefore it is useful to use **--log** or **--log-append** to write these messages to a file.

--show-adapters
(Standalone) Show available TAP-Win32 adapters which can be selected using the **--dev-node** option. On non-Windows systems, the **ifconfig**(8) command provides similar functionality.

--allow-nonadmin [TAP-adapter]
(Standalone) Set **TAP-adapter** to allow access from non-administrative accounts. If **TAP-adapter** is omitted, all TAP adapters on the system will be configured to allow non-admin access. The non-admin access setting will only persist for the length of time that the TAP-Win32 device object and driver remain loaded, and will need to be re-enabled after a reboot, or if the driver is unloaded and reloaded. This directive can only be used by an administrator.

--show-valid-subnets
(Standalone) Show valid subnets for **--dev tun** emulation. Since the TAP-Win32 driver exports an ethernet interface to Windows, and since TUN devices are point-to-point in nature, it is necessary for the TAP-Win32 driver to impose certain constraints on TUN endpoint address selection.

Namely, the point-to-point endpoints used in TUN device emulation must be the middle two addresses of a /30 subnet (netmask 255.255.255.252).

--show-net
(Standalone) Show OpenVPN's view of the system routing table and network adapter list.

PKCS#11 Standalone Options:

--show-pkcs11-ids [provider] [cert_private]
(Standalone) Show PKCS#11 token object list. Specify cert_private as 1 if certificates are stored as private objects.

If p11-kit is present on the system, the **provider** argument is optional; if omitted the default **p11-kit-proxy.so** module will be queried.

--verb option can be used BEFORE this option to produce debugging information.

IPv6 Related Options

The following options exist to support IPv6 tunneling in peer-to-peer and client-server mode. All options are modeled after their IPv4 counterparts, so more detailed explanations given there apply here as well (except for **--topology** , which has no effect on IPv6).

--ifconfig-ipv6 ipv6addr/bits ipv6remote
configure IPv6 address **ipv6addr/bits** on the ``tun'' device. The second parameter is used as route target for **--route-ipv6** if no gateway is specified.

--route-ipv6 ipv6addr/bits [gateway] [metric]
setup IPv6 routing in the system to send the specified IPv6 network into OpenVPN's ``tun'' device

--server-ipv6 ipv6addr/bits
convenience-function to enable a number of IPv6 related options at once, namely **--ifconfig-ipv6, --ifconfig-ipv6-pool, --tun-ipv6** and **--push tun-ipv6** Is only accepted if ``--mode server'' or ``--server'' is set.

--ifconfig-ipv6-pool ipv6addr/bits
Specify an IPv6 address pool for dynamic assignment to clients. The pool starts at **ipv6addr** and increments by +1 for every new client (linear mode). The **/bits** setting controls the size of the pool. Due to implementation details, the pool size must be between /64 and /112.

--ifconfig-ipv6-push ipv6addr/bits ipv6remote
for ccd/ per-client static IPv6 interface configuration, see **--client-config-dir** and **--ifconfig-push** for more details.

--iroute-ipv6 ipv6addr/bits
for ccd/ per-client static IPv6 route configuration, see **--iroute** for more details how to setup and use this, and how **--iroute** and **--route** interact.

SCRIPTING AND ENVIRONMENTAL VARIABLES

OpenVPN exports a series of environmental variables for use by user-defined scripts.

Script Order of Execution

--up
Executed after TCP/UDP socket bind and TUN/TAP open.

--tls-verify

Executed when we have a still untrusted remote peer.

--ipchange
Executed after connection authentication, or remote IP address change.

--client-connect
Executed in **--mode server** mode immediately after client authentication.

--route-up
Executed after connection authentication, either immediately after, or some number of seconds after as defined by the **--route-delay** option.

--route-pre-down
Executed right before the routes are removed.

--client-disconnect
Executed in **--mode server** mode on client instance shutdown.

--down
Executed after TCP/UDP and TUN/TAP close.

--learn-address
Executed in **--mode server** mode whenever an IPv4 address/route or MAC address is added to OpenVPN's internal routing table.

--auth-user-pass-verify
Executed in **--mode server** mode on new client connections, when the client is still untrusted.

String Types and Remapping

In certain cases, OpenVPN will perform remapping of characters in strings. Essentially, any characters outside the set of permitted characters for each string type will be converted to underbar ('_').

Q: Why is string remapping necessary?

A: It's an important security feature to prevent the malicious coding of strings from untrusted sources to be passed as parameters to scripts, saved in the environment, used as a common name, translated to a filename, etc.

Q: Can string remapping be disabled?

A: Yes, by using the **--no-name-remapping** option, however this should be considered an advanced option.

Here is a brief rundown of OpenVPN's current string types and the permitted character class for each string:

X509 Names: Alphanumeric, underbar ('_'), dash ('-'), dot ('.'), at ('@'), colon (':'), slash ('/'), and equal ('='). Alphanumeric is defined as a character which will cause the C library isalnum() function to return true.

Common Names: Alphanumeric, underbar ('_'), dash ('-'), dot ('.'), and at ('@').

--auth-user-pass username: Same as Common Name, with one exception: starting with OpenVPN 2.0.1, the username is passed to the OPENVPN_PLUGIN_AUTH_USER_PASS_VERIFY plugin in its raw form, without string remapping.

--auth-user-pass password: Any "printable" character except CR or LF. Printable is defined to be a character which will cause the C library isprint() function to return true.

--client-config-dir filename as derived from common name or username: Alphanumeric, underbar ('_'), dash ('-'), and dot ('.') except for "." or ".." as standalone strings. As of 2.0.1-rc6, the at ('@') character has been added as well for compatibility with the common name character class.

Environmental variable names: Alphanumeric or underbar ('_').

Environmental variable values: Any printable character.

For all cases, characters in a string which are not members of the legal character class for that string type will be remapped to underbar ('_').

Environmental Variables

Once set, a variable is persisted indefinitely until it is reset by a new value or a restart,

As of OpenVPN 2.0-beta12, in server mode, environmental variables set by OpenVPN are scoped according to the client objects they are associated with, so there should not be any issues with scripts having access to stale, previously set variables which refer to different client instances.

bytes_received
Total number of bytes received from client during VPN session. Set prior to execution of the **--client-disconnect** script.

bytes_sent
Total number of bytes sent to client during VPN session. Set prior to execution of the **--client-disconnect** script.

common_name
The X509 common name of an authenticated client. Set prior to execution of **--client-connect, --client-disconnect,** and **--auth-user-pass-verify** scripts.

config
Name of first **--config** file. Set on program initiation and reset on SIGHUP.

daemon
Set to "1" if the **--daemon** directive is specified, or "0" otherwise. Set on program initiation and reset on SIGHUP.

daemon_log_redirect
Set to "1" if the **--log** or **--log-append** directives are specified, or "0" otherwise. Set on program initiation and reset on SIGHUP.

dev
The actual name of the TUN/TAP device, including a unit number if it exists. Set prior to **--up** or **--down** script execution.

foreign_option_{n}
An option pushed via **--push** to a client which does not natively support it, such as **--dhcp-option** on a non-Windows system, will be recorded to this environmental variable sequence prior to **--up** script execution.

ifconfig_broadcast
The broadcast address for the virtual ethernet segment which is derived from the **--ifconfig** option when **--dev tap** is used. Set prior to OpenVPN calling the *ifconfig* or *netsh* (windows version of ifconfig) commands which normally occurs prior to **--up** script execution.

ifconfig_ipv6_local
The local VPN endpoint IPv6 address specified in the **--ifconfig-ipv6** option (first parameter). Set prior to OpenVPN calling the *ifconfig* or *netsh* (windows version of ifconfig) commands which normally occurs prior to **--up** script execution.

ifconfig_ipv6_netbits
The prefix length of the IPv6 network on the VPN interface. Derived from the /nnn parameter of the IPv6 address in the **--ifconfig-ipv6** option (first parameter). Set prior to OpenVPN calling the *ifconfig* or *netsh* (windows version of ifconfig) commands which normally occurs prior to **--up** script execution.

ifconfig_ipv6_remote
The remote VPN endpoint IPv6 address specified in the **--ifconfig-ipv6** option (second parameter). Set prior to OpenVPN calling the *ifconfig* or *netsh* (windows version of ifconfig) commands which normally occurs prior to **--up** script execution.

ifconfig_local
The local VPN endpoint IP address specified in the **--ifconfig** option (first parameter). Set prior to OpenVPN calling the *ifconfig* or *netsh* (windows version of ifconfig) commands which normally occurs prior to **--up** script execution.

ifconfig_remote
The remote VPN endpoint IP address specified in the **--ifconfig** option (second parameter) when **--dev tun** is used. Set prior to OpenVPN calling the *ifconfig* or *netsh* (windows version of ifconfig) commands which normally occurs prior to **--up** script execution.

ifconfig_netmask
The subnet mask of the virtual ethernet segment that is specified as the second parameter to **--ifconfig** when **--dev tap** is being used. Set prior to OpenVPN calling the *ifconfig* or *netsh* (windows version of ifconfig) commands which normally occurs prior to **--up** script execution.

ifconfig_pool_local_ip
The local virtual IP address for the TUN/TAP tunnel taken from an **--ifconfig-push** directive if specified, or otherwise from the ifconfig pool (controlled by the **--ifconfig-pool** config file directive). Only set for **--dev tun** tunnels. This option is set on the server prior to execution of the **--client-connect** and **--client-disconnect** scripts.

ifconfig_pool_netmask
The virtual IP netmask for the TUN/TAP tunnel taken from an **--ifconfig-push** directive if specified, or otherwise from the ifconfig pool (controlled by the **--ifconfig-pool** config file directive). Only set for **--dev tap** tunnels. This option is set on the server prior to execution of the **--client-connect** and **--client-disconnect** scripts.

ifconfig_pool_remote_ip
The remote virtual IP address for the TUN/TAP tunnel taken from an **--ifconfig-push** directive if specified, or otherwise from the ifconfig pool (controlled by the **--ifconfig-pool** config file directive). This option is set on the server prior to execution of the **--client-connect** and **--client-disconnect** scripts.

link_mtu
The maximum packet size (not including the IP header) of tunnel data in UDP tunnel transport mode. Set prior to **--up** or **--down** script execution.

local
The **--local** parameter. Set on program initiation and reset on SIGHUP.

local_port
The local port number, specified by **--port** or **--lport.** Set on program initiation and reset on SIGHUP.

password
The password provided by a connecting client. Set prior to **--auth-user-pass-verify** script execution only when the **via-env** modifier is specified, and deleted from the environment after the script returns.

proto
The **--proto** parameter. Set on program initiation and reset on SIGHUP.

remote_{n}
The **--remote** parameter. Set on program initiation and reset on SIGHUP.

remote_port_{n}
The remote port number, specified by **--port** or **--rport.** Set on program initiation and reset on SIGHUP.

route_net_gateway
The pre-existing default IP gateway in the system routing table. Set prior to **--up** script execution.

route_vpn_gateway
The default gateway used by **--route** options, as specified in either the **--route-gateway** option or the second parameter to **--ifconfig** when **--dev tun** is specified. Set prior to **--up** script execution.

route_{parm}_{n}
A set of variables which define each route to be added, and are set prior to **--up** script execution.

parm will be one of "network", "netmask", "gateway", or "metric".

n is the OpenVPN route number, starting from 1.

If the network or gateway are resolvable DNS names, their IP address translations will be recorded rather than their names as denoted on the command line or configuration file.

route_ipv6_{parm}_{n}
A set of variables which define each IPv6 route to be added, and are set prior to **--up** script execution.

parm will be one of "network" or "gateway" ("netmask" is contained as "/nnn" in the route_ipv6_network_{n}, unlike IPv4 where it is passed in a separate environment variable).

n is the OpenVPN route number, starting from 1.

If the network or gateway are resolvable DNS names, their IP address translations will be recorded rather than their names as denoted on the command line or configuration file.

peer_cert
Temporary file name containing the client certificate upon connection. Useful in conjunction with --tls-verify

script_context
Set to "init" or "restart" prior to up/down script execution. For more information, see documentation for **--up.**

script_type
Prior to execution of any script, this variable is set to the type of script being run. It can be one of the following: **up, down, ipchange, route-up, tls-verify, auth-user-pass-verify, client-connect, client-disconnect,** or **learn-address.** Set prior to execution of any script.

signal
The reason for exit or restart. Can be one of **sigusr1, sighup, sigterm, sigint, inactive** (controlled by **--inactive** option), **ping-exit** (controlled by **--ping-exit** option), **ping-restart** (controlled by **--ping-restart** option), **connection-reset** (triggered on TCP connection reset), **error,** or **unknown** (unknown signal). This variable is set just prior to down script execution.

time_ascii
Client connection timestamp, formatted as a human-readable time string. Set prior to execution of the **--client-connect** script.

time_duration
The duration (in seconds) of the client session which is now disconnecting. Set prior to execution of the **--client-disconnect** script.

time_unix
Client connection timestamp, formatted as a unix integer date/time value. Set prior to execution of the **--client-connect** script.

tls_digest_{n}
Contains the certificate SHA1 fingerprint/digest hash value, where **n** is the verification level. Only set for TLS connections. Set prior to execution of **--tls-verify** script.

tls_id_{n}
A series of certificate fields from the remote peer, where **n** is the verification level. Only set for TLS connections. Set prior to execution of **--tls-verify** script.

tls_serial_{n}
The serial number of the certificate from the remote peer, where **n** is the verification level. Only set for TLS connections. Set prior to execution of **--tls-verify** script. This is in the form of a decimal string like "933971680", which is suitable for doing serial-based OCSP queries (with OpenSSL, do not prepend "0x" to the string) If something goes wrong while reading the value from the certificate it will be an empty string, so your code should check that. See the

contrib/OCSP_check/OCSP_check.sh script for an example.

tls_serial_hex_{n}
Like **tls_serial_{n}**, but in hex form (e.g. "12:34:56:78:9A").

tun_mtu
The MTU of the TUN/TAP device. Set prior to **--up** or **--down** script execution.

trusted_ip (or trusted_ip6)
Actual IP address of connecting client or peer which has been authenticated. Set prior to execution of **--ipchange, --client-connect,** and **--client-disconnect** scripts. If using ipv6 endpoints (udp6, tcp6), **trusted_ip6** will be set instead.

trusted_port
Actual port number of connecting client or peer which has been authenticated. Set prior to execution of **--ipchange, --client-connect,** and **--client-disconnect** scripts.

untrusted_ip (or untrusted_ip6)
Actual IP address of connecting client or peer which has not been authenticated yet. Sometimes used to **nmap** the connecting host in a **--tls-verify** script to ensure it is firewalled properly. Set prior to execution of **--tls-verify** and **--auth-user-pass-verify** scripts. If using ipv6 endpoints (udp6, tcp6), **untrusted_ip6** will be set instead.

untrusted_port
Actual port number of connecting client or peer which has not been authenticated yet. Set prior to execution of **--tls-verify** and **--auth-user-pass-verify** scripts.

username
The username provided by a connecting client. Set prior to **--auth-user-pass-verify** script execution only when the **via-env** modifier is specified.

X509_{n}_{subject_field}
An X509 subject field from the remote peer certificate, where **n** is the verification level. Only set for TLS connections. Set prior to execution of **--tls-verify** script. This variable is similar to **tls_id_{n}** except the component X509 subject fields are broken out, and no string remapping occurs on these field values (except for remapping of control characters to "_"). For example, the following variables would be set on the OpenVPN server using the sample client certificate in sample-keys (client.crt). Note that the verification level is 0 for the client certificate and 1 for the CA certificate.

```
X509_0_emailAddress=me@myhost.mydomain
X509_0_CN=Test-Client
X509_0_O=OpenVPN-TEST
X509_0_ST=NA
X509_0_C=KG
X509_1_emailAddress=me@myhost.mydomain
X509_1_O=OpenVPN-TEST
X509_1_L=BISHKEK
X509_1_ST=NA
X509_1_C=KG
```

INLINE FILE SUPPORT

OpenVPN allows including files in the main configuration for the **--ca, --cert, --dh, --extra-certs, --key, --pkcs12, --secret** and **--tls-auth** options.

Each inline file started by the line **<option>** and ended by the line **</option>**

Here is an example of an inline file usage

```
<cert>
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
</cert>
```

When using the inline file feature with **--pkcs12** the inline file has to be base64 encoded. Encoding of a .p12 file into base64 can be done for example with OpenSSL by running **openssl base64 -in input.p12**

SIGNALS

SIGHUP
Cause OpenVPN to close all TUN/TAP and network connections, restart, re-read the configuration file (if any), and reopen TUN/TAP and network connections.

SIGUSR1
Like **SIGHUP,** except don't re-read configuration file, and possibly don't close and reopen TUN/TAP device, re-read key files, preserve local IP address/port, or preserve most recently authenticated

remote IP address/port based on **--persist-tun, --persist-key, --persist-local-ip,** and **--persist-remote-ip** options respectively (see above).

This signal may also be internally generated by a timeout condition, governed by the **--ping-restart** option.

This signal, when combined with **--persist-remote-ip,** may be sent when the underlying parameters of the host's network interface change such as when the host is a DHCP client and is assigned a new IP address. See **--ipchange** above for more information.

SIGUSR2
Causes OpenVPN to display its current statistics (to the syslog file if **--daemon** is used, or stdout otherwise).

SIGINT, SIGTERM
Causes OpenVPN to exit gracefully.

TUN/TAP DRIVER SETUP

If you are running Linux 2.4.7 or higher, you probably have the TUN/TAP driver already installed. If so, there are still a few things you need to do:

Make device: **mknod /dev/net/tun c 10 200**

Load driver: **modprobe tun**

EXAMPLES

Prior to running these examples, you should have OpenVPN installed on two machines with network connectivity between them. If you have not yet installed OpenVPN, consult the INSTALL file included in the OpenVPN distribution.

TUN/TAP Setup:

If you are using Linux 2.4 or higher, make the tun device node and load the tun module:

mknod /dev/net/tun c 10 200

modprobe tun

If you installed from RPM, the **mknod** step may be omitted, because the RPM install does that for you.

Only Linux 2.4 and newer are supported.

For other platforms, consult the INSTALL file at *http://openvpn.net/install.html* for more information.

Firewall Setup:

If firewalls exist between the two machines, they should be set to forward UDP port 1194 in both directions. If you do not have control over the firewalls between the two machines, you may still be able to use OpenVPN by adding **--ping 15** to each of the **openvpn** commands used below in the examples (this will cause each peer to send out a UDP ping to its remote peer once every 15 seconds which will cause many stateful firewalls to forward packets in both directions without an explicit firewall rule).

If you are using a Linux iptables-based firewall, you may need to enter the following command to allow incoming packets on the TUN device:

iptables -A INPUT -i tun+ -j ACCEPT

See the firewalls section below for more information on configuring firewalls for use with OpenVPN.

VPN Address Setup:

For purposes of our example, our two machines will be called **may.kg** and **june.kg.** If you are constructing a VPN over the internet, then replace **may.kg** and **june.kg** with the internet hostname or IP address that each machine will use to contact the other over the internet.

Now we will choose the tunnel endpoints. Tunnel endpoints are private IP addresses that only have meaning in the context of the VPN. Each machine will use the tunnel endpoint of the other machine to access it over the VPN. In our example, the tunnel endpoint for may.kg will be 10.4.0.1 and for june.kg, 10.4.0.2.

Once the VPN is established, you have essentially created a secure alternate path between the two hosts which is addressed by using the tunnel endpoints. You can control which network traffic passes between the hosts (a) over the VPN or (b) independently of the VPN, by choosing whether to use (a) the

VPN endpoint address or (b) the public internet address, to access the remote host. For example if you are on may.kg and you wish to connect to june.kg via **ssh** without using the VPN (since **ssh** has its own built-in security) you would use the command **ssh june.kg.** However in the same scenario, you could also use the command **telnet 10.4.0.2** to create a telnet session with june.kg over the VPN, that would use the VPN to secure the session rather than **ssh.**

You can use any address you wish for the tunnel endpoints but make sure that they are private addresses (such as those that begin with 10 or 192.168) and that they are not part of any existing subnet on the networks of either peer, unless you are bridging. If you use an address that is part of your local subnet for either of the tunnel endpoints, you will get a weird feedback loop.

Example 1: A simple tunnel without security

On may:

openvpn --remote june.kg --dev tun1 --ifconfig 10.4.0.1 10.4.0.2 --verb 9

On june:

openvpn --remote may.kg --dev tun1 --ifconfig 10.4.0.2 10.4.0.1 --verb 9

Now verify the tunnel is working by pinging across the tunnel.

On may:

ping 10.4.0.2

On june:

ping 10.4.0.1

The **--verb 9** option will produce verbose output, similar to the **tcpdump**(8) program. Omit the **--verb 9** option to have OpenVPN run quietly.

Example 2: A tunnel with static-key security (i.e. using a pre-shared secret)

First build a static key on may.

openvpn --genkey --secret key

This command will build a random key file called **key** (in ascii format). Now copy **key** to june over a secure medium such as by using the **scp**(1) program.

On may:

openvpn --remote june.kg --dev tun1 --ifconfig 10.4.0.1 10.4.0.2 --verb 5 --secret key

On june:

openvpn --remote may.kg --dev tun1 --ifconfig 10.4.0.2 10.4.0.1 --verb 5 --secret key

Now verify the tunnel is working by pinging across the tunnel.

On may:

ping 10.4.0.2

On june:

ping 10.4.0.1

Example 3: A tunnel with full TLS-based security

For this test, we will designate **may** as the TLS client and **june** as the TLS server. *Note that client or server designation only has meaning for the TLS subsystem. It has no bearing on OpenVPN's peer-to-peer, UDP-based communication model.*

First, build a separate certificate/key pair for both may and june (see above where **--cert** is discussed for more info). Then construct Diffie Hellman parameters (see above where **--dh** is discussed for more info). You can also use the included test files client.crt, client.key, server.crt, server.key and ca.crt. The .crt files are certificates/public-keys, the .key files are private keys, and ca.crt is a certification authority who has signed both client.crt and server.crt. For Diffie Hellman parameters you can use the included file dh1024.pem. *Note that all client, server, and certificate authority certificates and keys included in the OpenVPN distribution are totally insecure and should be used for testing only.*

On may:

openvpn --remote june.kg --dev tun1 --ifconfig 10.4.0.1 10.4.0.2 --tls-client --ca ca.crt --cert client.crt --key client.key --reneg-sec 60 --verb 5

On june:

openvpn --remote may.kg --dev tun1 --ifconfig 10.4.0.2 10.4.0.1 --tls-server --dh dh1024.pem --ca ca.crt --cert server.crt --key server.key --reneg-sec 60 --verb 5

Now verify the tunnel is working by pinging across the tunnel.

On may:

ping 10.4.0.2

On june:

ping 10.4.0.1

Notice the **--reneg-sec 60** option we used above. That tells OpenVPN to renegotiate the data channel keys every minute. Since we used **--verb 5** above, you will see status information on each new key negotiation.

For production operations, a key renegotiation interval of 60 seconds is probably too frequent. Omit the **--reneg-sec 60** option to use OpenVPN's default key renegotiation interval of one hour.

Routing:

Assuming you can ping across the tunnel, the next step is to route a real subnet over the secure tunnel. Suppose that may and june have two network interfaces each, one connected to the internet, and the other to a private network. Our goal is to securely connect both private networks. We will assume that may's private subnet is 10.0.0.0/24 and june's is 10.0.1.0/24.

First, ensure that IP forwarding is enabled on both peers. On Linux, enable routing:

echo 1 > /proc/sys/net/ipv4/ip_forward

and enable TUN packet forwarding through the firewall:

iptables -A FORWARD -i tun+ -j ACCEPT

On may:

route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.4.0.2

On june:

route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.4.0.1

Now any machine on the 10.0.0.0/24 subnet can access any machine on the 10.0.1.0/24 subnet over the secure tunnel (or vice versa).

In a production environment, you could put the route command(s) in a script and execute with the **--up** option.

FIREWALLS

OpenVPN's usage of a single UDP port makes it fairly firewall-friendly. You should add an entry to your firewall rules to allow incoming OpenVPN packets. On Linux 2.4+:

iptables -A INPUT -p udp -s 1.2.3.4 --dport 1194 -j ACCEPT

This will allow incoming packets on UDP port 1194 (OpenVPN's default UDP port) from an OpenVPN peer at 1.2.3.4.

If you are using HMAC-based packet authentication (the default in any of OpenVPN's secure modes), having the firewall filter on source address can be considered optional, since HMAC packet authentication is a much more secure method of verifying the authenticity of a packet source. In that case:

iptables -A INPUT -p udp --dport 1194 -j ACCEPT

would be adequate and would not render the host inflexible with respect to its peer having a dynamic IP address.

OpenVPN also works well on stateful firewalls. In some cases, you may not need to add any static rules to the firewall list if you are using a stateful firewall that knows how to track UDP connections. If you specify **--ping n,** OpenVPN will be guaranteed to send a packet to its peer at least once every **n** seconds. If **n** is less than the stateful firewall connection timeout, you can maintain an OpenVPN

connection indefinitely without explicit firewall rules.

You should also add firewall rules to allow incoming IP traffic on TUN or TAP devices such as:

iptables -A INPUT -i tun+ -j ACCEPT

to allow input packets from tun devices,

iptables -A FORWARD -i tun+ -j ACCEPT

to allow input packets from tun devices to be forwarded to other hosts on the local network,

iptables -A INPUT -i tap+ -j ACCEPT

to allow input packets from tap devices, and

iptables -A FORWARD -i tap+ -j ACCEPT

to allow input packets from tap devices to be forwarded to other hosts on the local network.

These rules are secure if you use packet authentication, since no incoming packets will arrive on a TUN or TAP virtual device unless they first pass an HMAC authentication test.

FAQ

http://openvpn.net/faq.html

HOWTO

For a more comprehensive guide to setting up OpenVPN in a production setting, see the OpenVPN HOWTO at *http://openvpn.net/howto.html*

PROTOCOL

For a description of OpenVPN's underlying protocol, see *http://openvpn.net/security.html*

WEB

OpenVPN's web site is at *http://openvpn.net/*

Go here to download the latest version of OpenVPN, subscribe to the mailing lists, read the mailing list archives, or browse the SVN repository.

BUGS

Report all bugs to the OpenVPN team <info@openvpn.net>.

SEE ALSO

dhcpcd(8), **ifconfig**(8), **openssl**(1), **route**(8), **scp**(1) **ssh**(1)

NOTES

This product includes software developed by the OpenSSL Project (*http://www.openssl.org/*)

For more information on the TLS protocol, see *http://www.ietf.org/rfc/rfc2246.txt*

For more information on the LZO real-time compression library see *http://www.oberhumer.com /opensource/lzo/*

COPYRIGHT

AUTHORS

James Yonan <jim@yonan.net>

Index

This document was created by man2html, using the manual pages.
Time: 06:53:40 GMT, June 08, 2015

Last modified on 08/04/15 07:36:46