09 Server Configuration

09 Server Configuration

This is our reference server configuration, which we use for most of our testing:

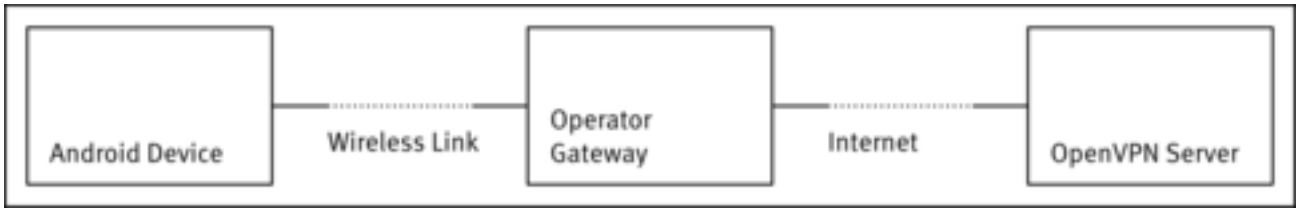
```
server 172.21.0.0 255.255.0.0
dev tun
local x.x.x.x
port 1194

topology net30

push "dhcp-option DNS 8.8.8.8"

# pushing this only works, if compression is enabled on the client

#comp-lzo no
#push "comp-lzo no"

user nobody
group nogroup
persist-key
persist-tun

ping 120
ping-restart 0

tran-window 28800

tls-auth easy-rsa-2.0/keys/tls-auth.key
ca easy-rsa-2.0/keys/ca.crt
cert easy-rsa-2.0/keys/demo.featvpn.com.crt
key easy-rsa-2.0/keys/demo.featvpn.com.key
dh easy-rsa-2.0/keys/dh2048.pem

script-security 3
auth-user-pass-verify verify.sh via-env

verb 4
```

If you are unable to connect FEAT VPN to your OpenVPN server, please try this configuration and see whether it makes things work. If it does, then please try to determine which difference between the reference configuration and your configuration it is that prevents FEAT VPN from connecting to your OpenVPN server when using your configuration. Then please report your findings in our support forum. Please also let us know in our support forum, if you cannot connect to your server when using the reference configuration.

As you can see in the reference configuration, you can push a DNS server to FEAT VPN.

The `ping 120` directive is particularly interesting and you probably need to experiment with it a little

to provide robust VPN services to your mobile users. Let's first take a look at the underlying problem. Here's how a mobile phone connects to your VPN server when connected to the Internet via a mobile data connection.



Mobile phones access the Internet via an operator gateway. The operator gateway is essentially a firewall with network address translation (NAT). Tunnel packets from the mobile phone reach the operator gateway via the mobile data connection. The operator gateway replaces the source IP address and the source TCP or UDP port of the tunnel packet with the IP address of the operator gateway and a TCP or UDP port of the operator gateway. The resulting modified tunnel packet is then forwarded via the Internet to the VPN server.

Suppose that the operator gateway sticks a given port **x** into the tunnel packet before forwarding it to the VPN server. Subsequent tunnel packets then receive the same port, **x**, before being forwarded. So, from the perspective of the VPN server, all tunnel packets come from port **x**. The operator gateway, just like any NAT-enabled firewall, remembers the port assigned to the first packet of a TCP connection or UDP conversation and assigns the same port to all subsequent packets of the same TCP connection or UDP conversation. However, if no packets are transmitted via a TCP connection or in a UDP conversation for a given amount of time, then the operator gateway drops all knowledge about the TCP connection or UDP conversation, including the port, **x**. The operator gateway assumes that the TCP connection or UDP conversation is not in use anymore and reuses the port, **x**, for a new, different TCP connection or UDP conversation.

Suppose that our VPN tunnel is idle for too long. The operator gateway then forgets everything about our VPN tunnel, in particular that it used port **x** for it. Suppose that we then send a tunnel packet to the VPN server. The operator gateway now does not recognize our packet as belonging to an existing TCP connection or UDP conversation. Instead, it believes that it is seeing a new TCP connection or UDP conversation. So, a new, different port, **y**, is assigned to our tunnel packet. The port is then inserted into our tunnel packet and sent to the VPN server.

This confuses the OpenVPN server. It originally received our tunnel packets from port **x**, now it receives them from port **y**. As the new port, **y**, is different from the original port, **x**, OpenVPN drops all packets with the new port - our tunnel hangs. As an aside, note that the `float` option, which is supposed to take care of changing IP addresses and ports, only seems to work for point-to-point OpenVPN configurations and not for certificate-based point-to-multi-point configurations.

So, what can we do? We can prevent our VPN tunnel from being idle for too long. This is where the `ping` option comes into play: We would like to make the VPN server ping the mobile client regularly in order to prevent a timeout on the operator gateway. We ping in this direction, i.e., from the server to the client and not the other way around, because tunnel packets from the server are less likely to get lost. Tunnel packets from the client travel over a wireless link and are thus prone to packet loss, in particular in case of a weak or no signal. Pinging from the server to the client is thus more reliable.

Now we face a dilemma. On the one hand, we want to ping as often as possible in order to keep the operator gateway from considering our VPN tunnel to be idle. On the other hand, we want to ping the least possible number of times, as each received ping packet wakes up our mobile device. If the mobile data connection is idle, our device turns off the radio, which results in a substantial improvement of battery life. However, if the server pings every few seconds, then the mobile data connection is never turned off and the battery drains.

We have seen mobile networks, where `ping 120` on the server, i.e., one ping packet every two minutes, perfectly keeps a VPN tunnel functional. But we have also seen mobile networks, where even `ping 30`, i.e., one ping packet every 30 seconds, is not enough. On one of these networks we

had to use `ping 10`, i.e., one ping packet every 10 seconds. On most phones this kept the radio constantly active and resulted in a battery life of only 5 hours.

Ultimately, configuring this option is a compromise. Suppose a scenario where most of your users are on a network in which `ping 120` works. But there are also other users for whom one ping packet every two minutes is not enough. In this case it may be wiser to ask the latter users to accept that their VPN tunnel may break down when they do not use it. They can always push the **Reconnect** button on the status screen to reconnect and thus fix the hanging VPN tunnel without having to enter their credentials again.

Note that the same problem applies to devices that are connected to Wi-Fi networks. Wi-Fi routers are NAT-enabled firewalls, just like operator gateways. However, for Wi-Fi networks, `ping 120` typically seems to work. Also, a Wi-Fi connection seems to be more benign in terms of battery use than a mobile 3G or LTE data connection.

So, if `ping 120` does not work for some of your users when they are connected to the Internet via a mobile data connection, asking them to establish the VPN tunnel only via Wi-Fi may be another option to address this issue.

Source URL: http://www.featvpn.com/09-server-configuration