Optimizing performance on gigabit networks

It is easily possible to saturate a 100 Mbps network using an OpenVPN tunnel. The throughput of the tunnel will be very close to the throughput of regular network interface. On gigabit networks and faster this is not so easy to achieve. This page explains how to increase the throughput of a VPN tunnel to near-linespeed for a 1 Gbps network. Some initial investigations using a 10 Gbps network are also explained.

Network setup

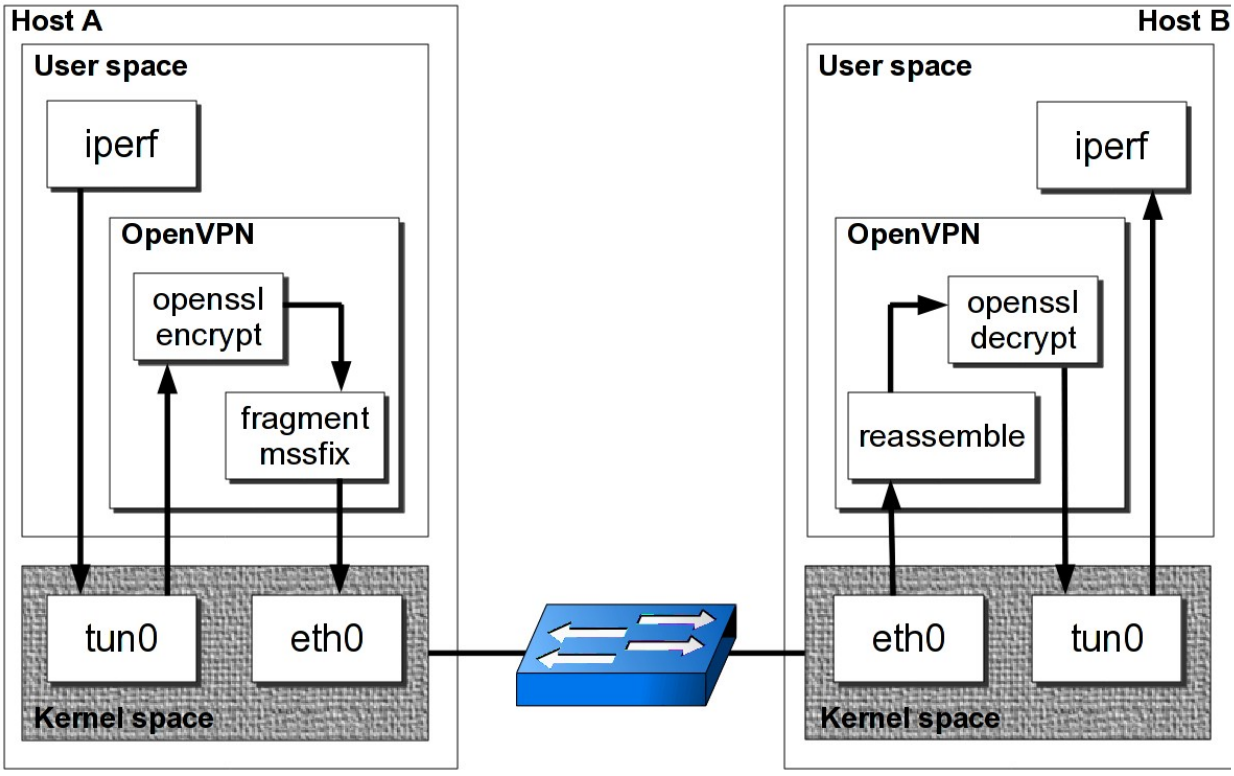
For this setup several machines were used, all connected to gigabit switches:

- two servers running CentOS 5.5 64bit, with an Intel Xeon E5440 CPU running @ 2.83GHz; the L2 cache size is 6 MB.
- a server running CentOS 5.5 64bit, with an Intel Xeon X5660 CPU running @ 2.80GHz; the L2 cache size is 12 MB. This CPU has support for the AES-NI instructions.
- a laptop running Fedora 14 64bit, with an Intel i5-560M CPU running @ 2.66GHz; the L2 cache size is 3 MB. This CPU also has support for the AES-NI instructions.

Before starting, the "raw" network speed was measured using 'iperf'. As expected, **iperf** reported consistent numbers around **940 Mbps**, which is (almost) optimal for a gigabit LAN. The MTU size on all switches in the gigabit LAN was set to 1500.

Understanding the flow of packets

It is important to understand how packets flow from the 'iperf' client via the OpenVPN tunnel to the 'iperf' server. The following diagram helps to clarify the flow:



when an 'iperf' packet is sent to the VPN server IP address, the packet enters the kernel's 'tun0' device. The packet is then forwarded to the userspace OpenVPN process, where the headers are stripped. The packet is encrypted and signed using OpenSSL calls. The encryption and signing algorithms can be configured using the '--cipher' and '--auth' options.

The resulting packet is then fragmented into pieces according to the '--fragment' and --mssfix' options. Afterwards, the encrypted packet is sent out over the regular network to the OpenVPN server. On the server, the process is reversed. First, the packet is reassembled, then decrypted and finally sent out the 'tun0' interface.

Standard setup

The default OpenVPN for CentOS 5 currently is 2.1.4; the system OpenSSL version is 0.9.8e-fips.

Using a very plain shared secret key setup for both server (listener)

```
openvpn --dev tun --proto udp --port 11000 --secret secret.key --ifconfig 192.1
```

and client

```
openvpn --dev tun --proto udp --port 11000 --secret secret.key --ifconfig 192.1
  --remote server
```

an **iperf** result of **156 Mbps** is obtained.

By switching to the cipher **aes-256-cbc** the performance drops even further to **126 Mbps**. These results were obtained on the two E5440 based servers.

Tweaked setup

The first tweak made was:

- increase the MTU size of the tun adapter ('--tun-mtu') to 6000 bytes. This resembles Jumbo frames on a regular Ethernet LAN. Note that the MTU size on the underlying network switches was **not** altered.
- disable OpenVPN's internal fragmentation algorithm using '--fragment 0'.
- disable OpenVPN's 'TCP Maximum Segment Size' limitor using '--mssfix 0'.

```
openvpn --dev tun --proto udp --port 11000 --secret secret.key --ifconfig 192.1
  --tun-mtu 6000 --fragment 0 --mssfix 0
```

and client

```
openvpn --dev tun --proto udp --port 11000 --secret secret.key --ifconfig 192.1
  --tun-mtu 6000 --fragment 0 --mssfix 0 --remote server
```

Now an **iperf** result of **307 Mbps** is obtained.

By playing with the '--tun-mtu' size we obtain (all speeds in Mbps)

MTU	Blowfish	AES256
1500	158	126
6000	307	220
9000	370	249
12000	416	252
24000	466	259
36000	470	244
48000	**510**	247
60000	488	221

(Please note that for all measurement a standard deviation of ~5% applies)

For the default Blowfish cipher the optimal value for the 'tun-mtu' parameters for a link between these two servers seems to be **48000** bytes. For this 'tun'mtu' setting the VPN throughput increases from 160 Mbps to **510** Mbps.

Similarly, for the AES-256 cipher the optimal value is **24000** bytes.

Explanation

By increasing the MTU size of the tun adapter **and** by disabling OpenVPN's internal fragmentation routines the throughput can be increased quite dramatically. The reason behind this is that by feeding larger packets to the OpenSSL encryption and decryption routines the performance will go up. The second advantage of not internally fragmenting packets is that this is left to the operating system and to the kernel network device drivers. For a LAN-based setup this can work, but when handling various types of remote users (road warriors, cable modem users, etc) this is not always a possibility.

txqueuelen

The default value for **tx_queue_len** in linux is 1000, however, openvpn overrides this default and sets it to 100. You can re-set it to the original default of 1000 by specifying **--txqueuelen 1000**. This can greatly improve throughput in scenarios where using jumbo frames (**--tun-mtu**) is not possible, such as over the internet. The expected performance gain is around 5x, being from 60mbps to almost 300mbps on some tests I carried on a real coast-to-coast scenario (with different ISPs).

txqueuelen	bandwidth
100	63
1000	292

Please note that, as the name indicates, this only affects the transmission, not the reception. So in case an asymmetric connection is used, this parameter will only have an observable effect on a peer whose transmission speed is greater than ~100mbps.

Using OpenSSL 1.0.0 with AES-NI patch

The second tweak made was to relink OpenVPN 2.1.4 using the OpenSSL 1.0.0a libraries with the Intel AES-NI patch applied. This patch is included by default in Fedora 12 and higher.

Previously it was reported that the Intel AES-NI patch caused the performance on non-AES-NI capable hardware to improve by a factor of 2. Closer investigation showed that the system OpenSSL library

0.9.8e-fips is actually at fault: after recompiling OpenSSL from source, with or without the Intel AES-NI patch, the performance also doubled. The Fedora 12 version of OpenSSL, 1.0.0-fips, and higher do not show this performance penalty.

Testing was done similar to the previous tweak

```
./openvpn --dev tun --proto udp --port 11000 --secret secret.key --ifconfig 192
  --tun-mtu 6000 --fragment 0 --mssfix 0 --cipher aes-256-cbc
```

and client

```
./openvpn --dev tun --proto udp --port 11000 --secret secret.key --ifconfig 192
  --tun-mtu 6000 --fragment 0 --mssfix 0  --cipher aes-256-cbc --remote server
```

Now an **iperf** result of **407 Mbps** is obtained.

By playing with the '--tun-mtu' size we obtain (all speeds in Mbps)

MTU	Blowfish	AES256
6000	310	407
9000	385	410
12000	417	478
24000	470	540
36000	510	561
48000	500	**585**
60000	500	582

(Please note that for all measurement a standard deviation of ~5% applies)

For the default Blowfish cipher the optimal value for the 'tun-mtu' parameters for a link between these two servers now seems to be **36000** bytes, although the difference for higher MTU sizes is minimal. Also note that the performance numbers are nearly identical to those generated using the system OpenSSL 0.9.8e-fips library.

When using the AES-256 cipher there is huge performance gain. The optimal MTU value now is **48000** bytes, but overall performance increased by a factor of 2 for nearly all MTU sizes.

Explanation

Compiling OpenSSL from scratch doubles the OpenSSL speed by a factor of 2. This can also be seen when running

```
openssl speed -evp aes-256-cbc
```

This is caused mainly by the fact that the CentOS-supplied OpenSSL 0.9.8e-fips library seems broken. The Fedora 12+ supplied OpenSSL 1.0.0-fips library does not induce the same penalty.

Using OpenSSL 1.0.0 with AES-NI capable hardware

The real performance gain is expected from the AES-NI capable hardware, such as the Intel Xeon X5660 and i5-560M CPUs. Using the exact same setup as above, but now using

```
engine aesni
```

and by using

```
./openvpn --dev tun --proto udp --port 11000 --secret secret.key --ifconfig 192
  --tun-mtu 9000 --fragment 0 --mssfix 0 --cipher aes-256-cbc --engine aesni
```

for the server and

```
./openvpn --dev tun --proto udp --port 11000 --secret secret.key --ifconfig 192
  --tun-mtu 9000 --fragment 0 --mssfix 0  --cipher aes-256-cbc --engine aesni -
```

for the client we obtain the following results (all speeds in Mbps):

	AES128	AES256
X5660 -> i5-560	885	878
i5-560 -> X5660	748	543

(Please note that for all measurement a standard deviation of ~5% applies)

Some initial conclusions:

- the AES-NI capable server CPU clearly shows a huge performance gain: it is nearly capable of saturating the gigabit network.
- it was not possible to run this test with a more capable client CPU. The i5-560M is not capable of keeping up with the X5660, as can be seen from the performance difference between encryption an decryption. In the first entry in the table above the X5660 encrypted all traffic, and the i5-560M decrypted it. The second entry shows the results for the reverse.

For reference: using no encryption

For reference, the above test was also run with encryption and signing disabled:

```
openvpn --dev tun --proto udp --port 11000 --secret secret.key --ifconfig 192.1
  --tun-mtu 9000 --fragment 0 --mssfix 0 --cipher none --auth none
```

Now an **iperf** result of **930 Mbps** is obtained, which shows that performance is not so much limited by the division between kernel-space and user-space processes, but mostly by the encryption and decryption routines as found in the OpenSSL libraries.

10 Gigabit networks

The "plaintext" test (i.e. encryption and signing disabled) was repeated on two machines connected to a 10 Gbps switch.

Again, all results were obtained using 'iperf'.

	MTU	speed
"raw"	1500	8.8 Gbps
"raw", no cksum offloading	1500	3.8 Gbps
via tun	60000	3.6 Gbps
via tun	48000	2.4 Gbps
via tun	9000	1.3 Gbps

This shows the limits of the kernel-space/user-space division that is present in Linux. It also shows how important TCP offloading is: if the TCP checksums need to be calculated in kernel space the performance drops by 50%!

The performance of OpenVPN over this network is not yet understood, as the raw openssl speed of the computers in this network is much lower than that of servers listed above. Unfortunately, it was not possible to use two AES-NI capable PCs on this network.

Last modified on 10/03/18 08:41:54

Attachments (1)