

Before jumping into the Network Extension framework, you need to know a bit of theory about VPN and networking in general. If you are already familiar with networking and VPN, feel free to skip this article. If you are not, this article has a quick overview/refresher on the network protocols, it outlines the problem that VPN is aimed to solve, and then analyzes one of the existing VPN protocols.



### Prerequisites

There are no prerequisites for going through this article, but the general knowledge of the networking protocols is going to be a plus. This article makes heavy use of Wireshark. If you are not familiar with this tool, I would suggest installing it (it is free), and inspecting one or two sample captures.

## Networking

What happens when you open your browser and type in a URL? Let's open Wireshark and find out.

### DNS, HTTP (Application Layer)

First, the browser needs to send a DNS request to resolve an IP address for the given domain name, for example `www.example.com`.

DNS Request

DNS Response

The browser sends a DNS query with a domain name `www.example.com`.

The image shows a Wireshark packet capture of a DNS query. The packet list table at the top shows a DNS query from source 192.168.0.13 to destination 65.19.96.252. The packet details pane shows the query for 'www.example.com' with transaction ID 0xd51c. The packet bytes pane shows the raw data of the DNS query, with a yellow box highlighting the domain name 'www.example.com' in the query section.

**IP address of the DNS server configured by DHCP**

No.	Time	Source	Destination	Protocol	Length	Info
23...	9.699123	192.168.0.13	65.19.96.252	DNS	75	Standard query 0xd51c A www.example...
23...	9.712215	65.19.96.252	192.168.0.13	DNS	91	Standard query response 0xd51c A www...
23...	9.712955	192.168.0.13	93.184.216.34	TCP	78	56624 → 80 [SYN] Seq=0 Win=65535 Len...
23...	9.713365	192.168.0.13	93.184.216.34	TCP	78	56625 → 80 [SYN] Seq=0 Win=65535 Len...
23...	9.728383	93.184.216.34	192.168.0.13	TCP	74	80 → 56624 [SYN, ACK] Seq=0 Ack=1 Wi...
23...	9.728474	192.168.0.13	93.184.216.34	TCP	66	56624 → 80 [ACK] Seq=1 Ack=1 Win=131...
23...	9.734553	93.184.216.34	192.168.0.13	TCP	74	80 → 56625 [SYN, ACK] Seq=0 Ack=1 Wi...
23...	9.734679	192.168.0.13	93.184.216.34	TCP	66	56625 → 80 [ACK] Seq=1 Ack=1 Win=131...
23...	9.735495	192.168.0.13	93.184.216.34	HTTP	421	GET / HTTP/1.1
23...	9.752002	93.184.216.34	192.168.0.13	TCP	66	80 → 56625 [ACK] Seq=1 Ack=356 Win=6...

Frame 2359: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface en0, id 0

Ethernet II, Src: Apple\_2a:43:67 (f0:18:98:2a:43:67), Dst: Netgear\_0a:a4:6a (10:0c:6b:0a:a4:6a)

Internet Protocol Version 4, Src: 192.168.0.13, Dst: 65.19.96.252

User Datagram Protocol, Src Port: 63802, Dst Port: 53

Domain Name System (query)

Transaction ID: 0xd51c

Flags: 0x0100 Standard query

Questions: 1

Answer RRs: 0

Authority RRs: 0

Additional RRs: 0

Queries

www.example.com: type A, class IN

Name: www.example.com

[Name Length: 15]

[Label Count: 3]

Type: A (Host Address) (1)

01----- (0-0000)

0000 10 0c 6b 0a a4 6a f0 18 98 2a 43 67 08 00 45 00 ...k..j...\*Cg..E.

0010 00 3d 42 e1 00 00 ff 11 16 0a c0 a8 00 0d 41 13 ...=B.....A.

0020 60 fc f0 3a 00 35 00 29 83 ee d5 1c 01 00 00 01 ...:..5.).....

0030 00 00 00 00 00 00 03 77 77 77 07 65 78 61 6d 70 .....w ww.examp

0040 6c 65 03 63 6f 6d 00 00 01 00 01 le.com... ..

**DNS datagram encapsulated in UDP packet**

Query Name (dns.qry.name), 17 bytes

Packets: 2510 · Displayed: 2510 (100.0%)

Profile: Default

**The Domain Name System (DNS)** is the phonebook of the Internet. Humans access information through domain names, such as `apple.com` or `nshipster.com`. Browsers interact through IP addresses. DNS translates domain names to IP addresses<sup>1</sup>.

A browser has to know an IP address to send requests to `www.example.com`. How was it able to send the requests to the DNS server? Typically, an IP address (or addresses) of the DNS server is provided to the computer by DHCP when connecting to the router.

Now that the browser has an IP address, it is ready to start sending HTTP requests to `www.example.com (93.184.216.34)` .

HTTP Request

HTTP Response

The browser takes the resolved IP address and initiates an HTTP request to fetch the content of the page at the root path ( `/` ).

example-com.pcapng

Apply a display filter ... <%%/>

No.	Time	Source	Destination	Protocol	Length	Info
23...	9.712955	192.168.0.13	93.184.216.34	TCP	78	56624 → 80 [SYN] Seq=0 Win=65535 Len=0
23...	9.713365	192.168.0.13	93.184.216.34	TCP	78	56625 → 80 [SYN] Seq=0 Win=65535 Len=0
23...	9.728383	93.184.216.34	192.168.0.13	TCP	74	80 → 56624 [SYN, ACK] Seq=0 Ack=1 Win=
23...	9.728474	192.168.0.13	93.184.216.34	TCP	66	56624 → 80 [ACK] Seq=1 Ack=1 Win=13171
23...	9.734553	93.184.216.34	192.168.0.13	TCP	74	80 → 56625 [SYN, ACK] Seq=0 Ack=1 Win=
23...	9.734679	192.168.0.13	93.184.216.34	TCP	66	56625 → 80 [ACK] Seq=1 Ack=1 Win=13171
23...	9.735495	192.168.0.13	93.184.216.34	HTTP	421	GET / HTTP/1.1
23...	9.752902	93.184.216.34	192.168.0.13	TCP	66	80 → 56625 [ACK] Seq=1 Ack=356 Win=676
23...	9.752906	93.184.216.34	192.168.0.13	HTTP	1088	HTTP/1.1 200 OK (text/html)
23...	9.752996	192.168.0.13	93.184.216.34	TCP	66	56625 → 80 [ACK] Seq=356 Ack=1023 Win=

▶ Internet Protocol Version 4, Src: 192.168.0.13, Dst: 93.184.216.34

▶ Transmission Control Protocol, Src Port: 56625, Dst Port: 80, Seq: 1, Ack: 1, Len: 355

▼ Hypertext Transfer Protocol

▼ GET / HTTP/1.1\r\n

▶ [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]

Request Method: GET

Request URI: /

Request Version: HTTP/1.1

Host: www.example.com\r\n

Upgrade-Insecure-Requests: 1\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8\r\n

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Vers

Accept-Language: en-us\r\n

Accept-Encoding: gzip, deflate\r\n

Connection: keep-alive\r\n

\r\n

[Full request URI: http://www.example.com/]

HTTP request 1/11

0050	0d 0a 48 6f 73 74 3a 20	77 77 77 2e 65 78 61 6d	Host: www.exam
0060	70 6c 65 2e 63 6f 6d 0d	0a 55 70 67 72 61 64 65	ple.com Upgrade
0070	2d 49 6e 73 65 63 75 72	65 2d 52 65 71 75 65 73	-Insecure-Req
0080	74 73 3a 20 31 0d 0a 41	63 63 65 70 74 3a 20 74	ts: 1 Accept: t
0090	65 78 74 2f 68 74 6d 6c	2c 61 70 70 6c 69 63 61	ext/html , applica
00a0	74 69 6f 6e 2f 78 68 74	6d 6c 2b 78 6d 6c 2c 61	tion/xht ml+xml,a
00b0	70 70 6c 69 63 61 74 69	6f 6e 2f 78 6d 6c 3b 71	pplicati on/xml;q

HTTP Host (http.host), 23 bytes

Packets: 2510 · Displayed: 2510 (100.0%)

Profile: Default

Let’s now spend too much time talking about the application layer protocols. I’m assuming you are already familiar with HTTP and DNS. The reason I used these two examples is that HTTP typically works on top of TCP<sup>2</sup>, and DNS on top of UDP<sup>3</sup>.

https://kean.blog/post/networking-101

3/12

i If you want to learn more, MDN has a great section on HTTP, and Cloudflare – on DNS. If you want to dig a bit deeper, I would recommend reading RFCs. You can find the list of HTTP and DNS RFCs on the respective wiki pages.

## UDP, TCP (Transport Layer)

HTTP and DNS solve specific application-level problems. But when a computer receives a message, how does it know which application to deliver it to? This is where *transport layer* protocols, such as User Datagram Protocol (UDP) and Transmission Control Protocol (TCP), come in.

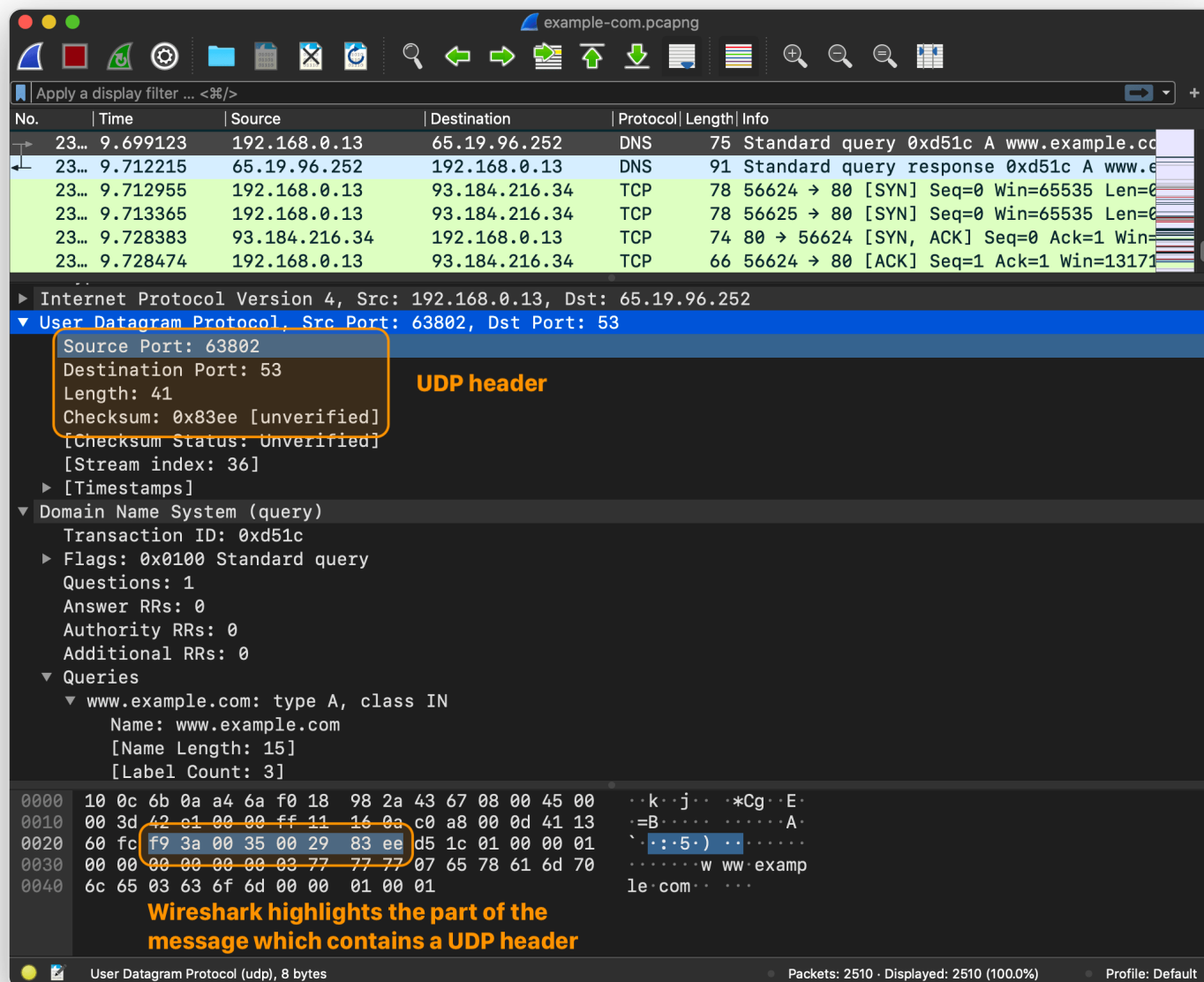
When a server is started, it typically creates a **socket**, an internal endpoint addressable using a combination of an IP address and a **port number** (a 16-bit unsigned integer ranging from 0 to 65535), and starts *listening* for messages arriving on this socket.

When a browser needs to send a message, it also needs a socket and creates one using one of the system APIs.

i On Apple platforms, there is a variety of APIs for working with ports/sockets, ranging from the Unix sockets, **CFSocket** wrapper and to recently introduced **Network** framework.

Transport layer protocols, such as TCP and UDP, transfer data using protocol data units. For TCP, the PDU is a **segment**, and a **datagram** for UDP. Both protocols use a header field for recording the source and destination port number.

The DNS messages are typically sent using UDP (other options exist, even including things like DNS over HTTPS). The UDP datagram header is extremely simple. When a system needs to send a DNS message, it takes the message (the protocol of the message is irrelevant, for DNS it's just raw bytes), and prepends 4 bytes of the UDP datagram header. If we go back to our browser example, here is what the UDP header looks like:



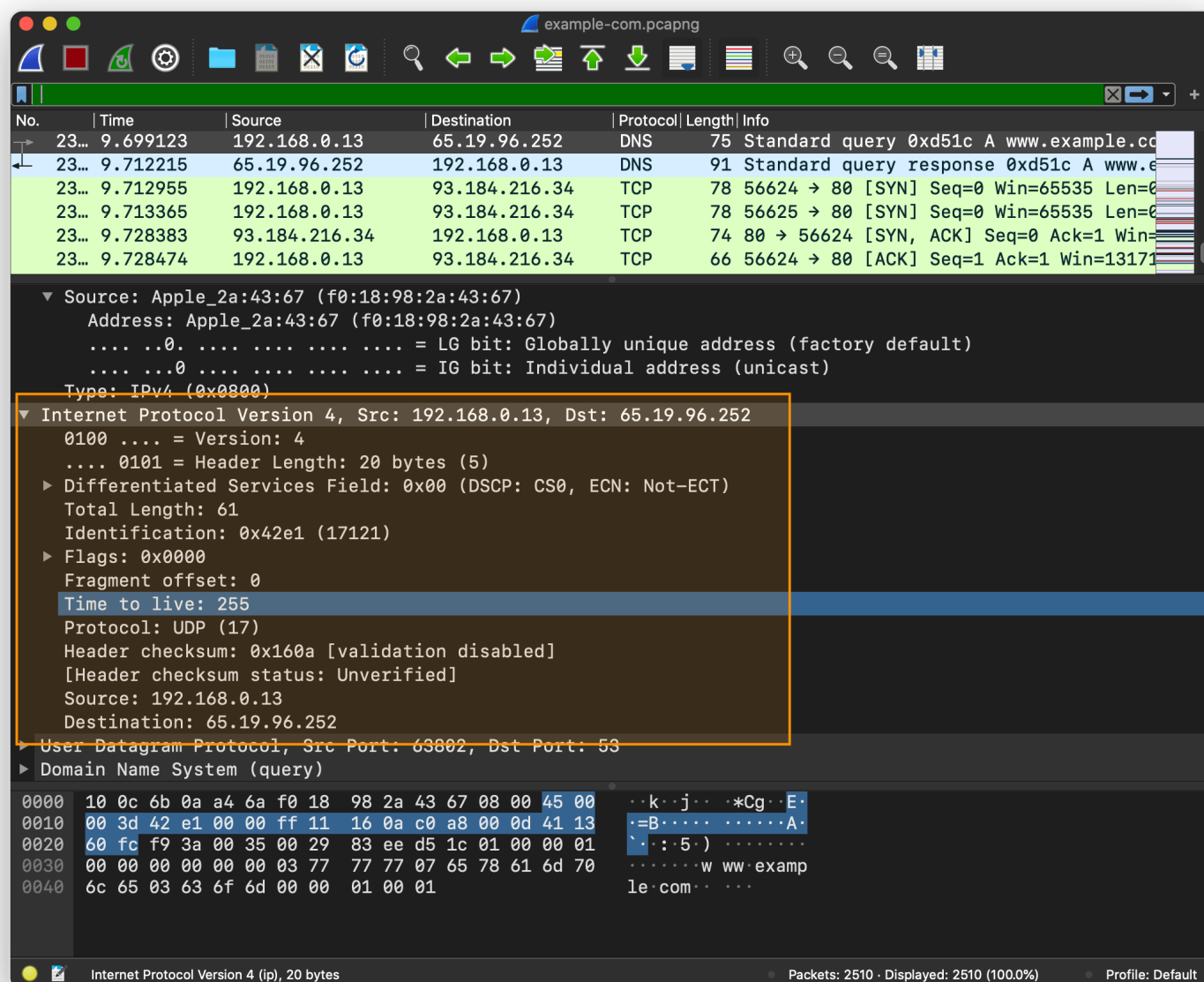
The *source* port is `63802`, which is the port that the browser reserved for itself, and the *destination* port is `53`, which is one of the well-known ports.

Obviously, UDP and TCP are completely different. TCP protocol has its own segment structure. TCP is connection-oriented, it provides reliable, ordered, and error-checked delivery of a stream of octets. UDP, on the other hand, uses a simple connectionless communication model, it's very minimalistic.

## IP (Internet Layer)

In the previous section, we established that a socket is addressable using a combination of an IP address and a port number. We know what the port is for. But there are no IP addresses to be found in UDP/TCP headers. To know which computer

to send the messages to, you need an IP address and IP protocol. Let's get back to our Wireshark capture to see what information an IP header contains.



All of the information that you saw captured using Wireshark was recorded from my `en0` network interface (WiFi).

**Network interface** is a point of connection between the computer and the network. A computer can have multiple network interfaces: Wi-Fi, Ethernet, etc.

There are *physical* network interfaces, such as `en0` (which is your WiFi module on macOS). You can list all of those using `networksetup -listallhardwareports` command. Each physical interface has an associated Ethernet Address and a hardware device, for example:



```
> networksetup -listallhardwareports
```

```
Hardware Port: Wi-Fi
```

```
Device: en0
```

```
Ethernet Address: f0:18:98:2a:43:67
```

```
...
```

And then there are *virtual* network interfaces, such as `lo0` (a loopback interface which computer uses to communicate with itself) and `utun` (which are typically used by VPN, more on this later). You can use `ifconfig` to see all of the network interfaces, including both physical and virtual.

```
> ifconfig
```

```
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
```

```
options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP
```

```
inet 127.0.0.1 netmask 0xff000000
```

```
...
```

All of the information that you saw from the captures in this section is also visible to my ISP and to any servers between me and the destination. They saw all my DNS queries, they saw my IP address, and in case of HTTP (unencrypted), they saw all my HTTP communications too<sup>4</sup>. Using this information anyone can identify and track me. And these are the kinds of problems that VPN is designed to solve.

## VPN

What is VPN (Virtual Private Network)? Originally, VPN was designed to establish secure connections between a computer and a protected or private corporate network, using an otherwise insecure public network. This typically involved “making” a computer part of the private network so that the computers within the network could address it.

In recent years, VPNs also blown up in popularity in the consumer market as a way to stay secure and anonymous when going online, and also enjoying unrestricted network access by hiding the source of the traffic. Business and personal VPN are

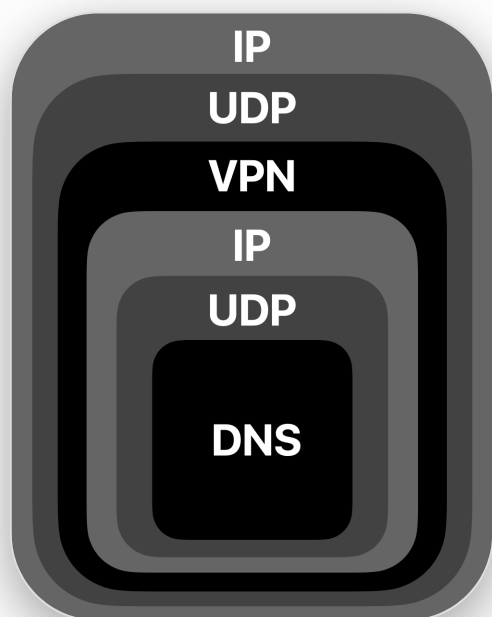
typically very different products. I'm going to focus primarily on personal VPN in this series, as it involves slightly less complexity to deal with.

## How Does It Works?

The key idea behind VPN is surprisingly simple:

- Take all of the IP packets<sup>5</sup> coming from the system
- Encrypt them
- Encapsulated encrypted data blobs in the VPN packets
- Send the VPN packets to the VPN server (over UDP/TCP)

The IP packets end up being wrapped in another IP packets, a networking version of a Russian nesting doll.



This is a simple idea, yet it achieves all of our goals. The original IP packets are fully encrypted. You can see neither the application, nor even the internet layer contents (IP addresses). The only thing ISP or other observers are going to see is some blob of encrypted data going between a VPN client and a VPN server.

Now, how do you implement something like this? This is what I'm going to cover in the upcoming articles. It is going to involve direct IP packets manipulations,



encryption, virtual network interfaces, and more. Exciting stuff!

## Example: OpenVPN

The idea behind VPN is simple, yet designing and implementing a protocol is far from easy. Before designing a VPN protocol of our own, it is worth exploring an existing VPN protocol, such as OpenVPN.

The description of the protocol is surprisingly short and simple. As a first step, the client **establishes the session** with the server and exchanges cryptographic information.

**Reset Client**   **Reset Server**   **Acknowledge**

The client sends a *control* ( `P_CONTROL` ) packet with `P_CONTROL_HARD_RESET_CLIENT_V2` type which prepares a new session.

Wireshark presents you information in a human-readable format, but OpenVPN, as most of the other networking protocols, is a binary protocol

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.103	192.168.56.102	OpenVPN	84	MessageType: P_CONTROL_HARD_RESET_CLIENT_V2
2	0.003189	192.168.56.102	192.168.56.103	OpenVPN	96	MessageType: P_CONTROL_HARD_RESET_SERVER_V2
3	0.004815	192.168.56.103	192.168.56.102	OpenVPN	92	MessageType: P_ACK_V1
4	0.005863	192.168.56.103	192.168.56.102	OpenVPN	184	MessageType: P_CONTROL_V1 (Message fragment 1...
5	0.005873	192.168.56.103	192.168.56.102	OpenVPN	184	MessageType: P_CONTROL_V1 (Message fragment 2...
6	0.005880	192.168.56.103	192.168.56.102	TLSv1	110	Client Hello
7	0.006143	192.168.56.102	192.168.56.103	OpenVPN	92	MessageType: P_ACK_V1
8	0.007420	192.168.56.102	192.168.56.103	OpenVPN	92	MessageType: P_ACK_V1
9	0.048018	192.168.56.102	192.168.56.103	OpenVPN	196	MessageType: P_CONTROL_V1 (Message fragment 1...
10	0.048038	192.168.56.102	192.168.56.103	OpenVPN	184	MessageType: P_CONTROL_V1 (Message fragment 2...

Destination Port: 1194  
Length: 50  
Checksum: 0x53b4 [unverified]  
[Checksum Status: Unverified]  
[Stream index: 0]  
[Timestamps]

▼ OpenVPN Protocol

▼ Type: 0x38 [opcode/key\_id]  
0011 1... = Opcode: P\_CONTROL\_HARD\_RESET\_CLIENT\_V2 (0x07)  
.... 000 = Key ID: 0  
Session ID: 9311214641221158445  
HMAC: de86734d2cbff151b2b1231b61e42308a272818e  
Packet-ID: 1  
Net Time: Jan 22, 2013 18:52:12.000000000 EST  
Message Packet-ID Array Length: 0  
Message Packet-ID: 0

0000 08 00 27 4a be 45 08 00 27 bb 22 84 08 00 45 00 ... 'J·E· · · · · E·  
0010 00 46 00 00 40 00 40 11 48 89 c0 a8 38 67 c0 a8 ·F·@·@·H· ·8g·  
0020 38 66 81 ae 04 aa 00 32 53 b4 38 81 38 14 62 1d 8f· · · · · 2 S·8·b·  
0030 67 46 2d de 86 73 4d 2c bf f1 51 b2 b1 23 1b 61 gF· · · · · sM, · ·Q· ·#·a  
0040 e4 23 08 a2 72 81 8e 00 00 00 01 50 ff 26 2c 00 ·g· ·r· · · · · P·&·  
0050 00 00 00 00 · · · · ·

OpenVPN Protocol (openvpn), 42 bytes   Packets: 440 - Displayed: 440 (100.0%)   Profile: Default

When the session is ready, the client and the server need to exchange cryptographic data to agree on a key. The exchange, called the “handshake” in TLS<sup>6</sup>.

## Control Message      TLS Payload

The client sends another control ( P\_CONTROL ) packet, this time with a payload.

The image shows a Wireshark packet capture of an OpenVPN session. The top pane displays a list of packets. Packets 4, 5, and 6 are highlighted with an orange box, indicating they are fragments of a single OpenVPN message. Packet 7 is also highlighted, showing the Client Hello message. The middle pane shows the details of the selected packet (P\_CONTROL\_V1), including the session ID, HMAC, and message fragments. The bottom pane shows the raw packet data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.103	192.168.56.102	OpenVPN	84	MessageType: P_CONTROL_HARD_RESET_CLIENT_V2
2	0.003189	192.168.56.102	192.168.56.103	OpenVPN	96	MessageType: P_CONTROL_HARD_RESET_SERVER_V2
3	0.004815	192.168.56.103	192.168.56.102	OpenVPN	92	MessageType: P_ACK_V1
4	0.005863	192.168.56.103	192.168.56.102	OpenVPN	184	MessageType: P_CONTROL_V1 (Message fragment 1...
5	0.005873	192.168.56.103	192.168.56.102	OpenVPN	184	MessageType: P_CONTROL_V1 (Message fragment 2...
6	0.005880	192.168.56.103	192.168.56.102	TLSv1	110	Client Hello
7	0.006143	192.168.56.102	192.168.56.103	OpenVPN	92	MessageType: P_ACK_V1

**A single OpenVPN message split across three UDP datagram**

**Wireshark automatically reassembles messages from multiple datagrams for you**

OpenVPN Protocol

Type: 0x20 [opcode/key\_id]  
 0010 0... = Opcode: P\_CONTROL\_V1 (0x04)  
 .... .000 = Key ID: 0  
 Session ID: 9311214641221158445  
 HMAC: 4d8bfbb13d3bc971a68a4a82950e8240786861b2  
 Packet-ID: 5  
 Net Time: Jan 22, 2013 18:52:12.000000000 EST  
 Message Packet-ID Array Length: 0  
 Message Packet-ID: 3  
 Message fragment (26 bytes)

[3 Message fragments (226 bytes): #4(100), #5(100), #6(26)]  
 [Frame: 4, payload: 0-99 (100 bytes)]  
 [Frame: 5, payload: 100-199 (100 bytes)]  
 [Frame: 6, payload: 200-225 (26 bytes)]  
 [Message fragment count: 3]  
 [Reassembled message length: 226]

Transport Layer Security

0010 00 60 00 00 40 00 40 11 48 6f c0 a8 38 67 c0 a8 ...@.@ Ho 8g  
 0020 38 66 81 ae 04 aa 00 4c 13 38 20 81 38 14 62 1d 8f ...L 8 8 b  
 0030 67 46 2d 4d 8b fb b1 3d 3b c9 71 a6 8a 4a 82 95 gF-M ... ; q J  
 0040 0e 82 40 78 68 61 b2 00 00 00 05 50 ff 26 2c 00 @xha ... P &  
 0050 00 00 00 03 05 00 12 00 13 00 01 00 02 00 03 00 .....  
 0060 0f 00 10 00 11 00 23 00 00 00 0f 00 01 01 .....# .....

Frame (110 bytes)      Reassembled Message (226 bytes)

OpenVPN Protocol (openvpn), 68 bytes      Packets: 440 · Displayed: 440 (100.0%)      Profile: Default

I'm not going to go into detail regarding the TLS exchange. The Wireshark session only shows the first messages of the TLS “handshake”. The entire handshake requires a few round-trips. The point is that after a few control messages, the client and the server are ready to encrypt and send messages to the server, regardless of which version of TLS and which encryption model was used. And this is what P\_DATA\_V1 message type is for – data channel packet containing actual tunnel data ciphertext.

OpenVPN\_UDP\_tls-auth.pcapng

Apply a display filter ... <%%/>

No.	Time	Source	Destination	Protocol	Length	Info
363	10.098535	192.168.56.104	192.168.56.102	OpenVPN	92	MessageType: P_ACK_V1
364	10.147384	192.168.56.104	192.168.56.102	OpenVPN	92	MessageType: P_ACK_V1
365	11.991671	192.168.56.103	192.168.56.102	OpenVPN	95	MessageType: P_DATA_V1
366	13.140540	192.168.56.102	192.168.56.103	OpenVPN	95	MessageType: P_DATA_V1
367	13.604953	192.168.56.104	192.168.56.102	OpenVPN	167	MessageType: P_DATA_V1
368	13.605505	192.168.56.102	192.168.56.104	OpenVPN	167	MessageType: P_DATA_V1

Time to live: 64  
Protocol: UDP (17)  
Header checksum: 0x487e [validation disabled]  
[Header checksum status: Unverified]  
Source: 192.168.56.103  
Destination: 192.168.56.102

▼ User Datagram Protocol, Src Port: 33198, Dst Port: 1194  
Source Port: 33198  
Destination Port: 1194  
Length: 61  
Checksum: 0x135d [unverified]  
[Checksum Status: Unverified]  
[Stream index: 0]  
▶ [Timestamps]

▼ OpenVPN Protocol  
Type: 0x30 [opcode/key\_id]  
0011 0... = Opcode: P\_DATA\_V1 (0x06)  
....000 = Key ID: 0

▼ Data (52 bytes)  
Data: d7b233d32203d9b4f27ce4a943ded9b8d51b867e5c40f65b...

**Data messages exchanged between the client and the server**

**Wireshark is not longer able to identify the payload because it is now ciphertext (encrypted IP packets)**

OpenVPN Protocol (openvpn), 53 bytes

Packets: 440 · Displayed: 440 (100.0%)

Profile: Default

And this wraps up an overview of OpenVPN. There is no need to fully understand it, but the big picture is clear. The main takeaways are:

- OpenVPN is a binary protocol that has multiple message types: control ( `P_CONTROL_V1` ), acknowledgment ( `P_ACK_V1` ), and data ( `P_DATA_V1` ).
- Before a client and a server can start exchanging data ( `P_DATA_V1` ), they need to establish a session and exchange cryptographic information.
- The messages are sent over UDP<sup>7</sup>. The control messages ( `P_CONTROL_V1` ) have a delivery guarantee and require acknowledgment ( `P_ACK_V1` ). The data messages ( `P_DATA_V1` ) don't require an acknowledgment and don't have a delivery guarantee, the underlying protocol of the original messages is responsible for providing the needed guarantees.

This is enough information to start working on our toy VPN protocol.

[Continue Reading »](#)

---

## References

1. Wireshark, **OpenVPN Sample Captures**
  2. Cloudflare, **What is DNS?**
  3. Mozilla, **HTTP**
  4. OpenVPN, **OpenVPN Protocol**
- 

1. You could memorize an IPv4 address, but try memorizing `2400:cb00:2048:1::c629:d7a2` (IPv6). ↩
2. HTTP/3 seems to be replacing TCP with QUIC, a new transport layer protocol based on UDP ↩
3. That's not always the case and it depends on your browser. For example, Firefox now uses DNS over HTTPS by default. ↩
4. If you are using HTTPS, you are generally good. Only the destination server is going to be able to read HTTP headers and body of your requests. ↩
5. The packets that are configured to go through the VPN tunnel. A *split* tunnel can be configured, such that only packets with the predefined destination addressed are routed via the tunnel, and the rest go through the normal physical network interfaces. ↩
6. OpenVPN supports two modes: static key (using a pre-shared key) and TLS (using SSL/TLS and certificates for authentication and key exchange). In this example, you see the later. ↩
7. OpenVPN uses UDP by default, but also supports TCP. ↩