VPN（Virtual Private Network），虚拟专用网络，是一种通过公用网络安全地对企业内部专用网络进行远程访问的连接方式，可有效保障通信的机密性。

传统SSL VPN通过端口代理的方法实现，代理服务器根据应用协议的类型（如http

telnet等）做相应的端口代理，客户端与代理服务器之间建立SSL安全连

接，客户端与应用服务器之间的所有数据传输通过代理服务器转发。这种实现

方式烦琐，应用范围也比较窄：仅适用于用TCP固定端口进行通信的应用，

对每个需要代理的端口进行单独配置；对于每个需要用到动态端口的协议

都必须重新开发，且在代理中解析应用协议才能实现代理，如FTP协议；不能

对TCP以外的其它网络通信协议进行代理；代理服务器前端的防火墙也要根据

代理端口的配置变化进行相应调整。

OpenVPN以一种全新的方式实现了SSL VPN的功能，克服了传统SSL VPN的

一些缺陷，扩展了应用领域，并且防火墙上只需开放TCP或UDP协议的一个端

口。

**虚拟网卡**

在Linux2.4版本以上，操作[系统](http://www.zclw.net/" \t "_blank)支持一个名为tun的设备，tun设备的驱动程序中包含两个部分，一部分是字符设备驱动，一部分是网卡驱动。网卡的驱动把从TCP/IP协议栈收到的数据包结构skb放于tun设备的读取队列，用户进程通过调用字符设备接口read获得完整的IP数据包，字符驱动read函数的功能是从设备的读取队列读取数据，将核心态的skb传递给用户；反过来字符驱动write函数给用户提供了把用户态的数据写入核心态的接口，write函数把用户数据写入核心空间并穿入TCP/IP协议栈。该设备既能以字符设备的方式被读写，作为[系统](http://www.zclw.net/" \t "_blank)的虚拟网卡，也具有和[物理](http://www.zclw.net/" \t "_blank)网卡相同的特点：能够配置IP地址和路由。对虚拟网卡的使用是OpenVPN实现其SSL VPN功能的关键。

**地址池以及路由**

OpenVPN服务器一般需要配置一个虚拟IP地址池和一个自用的静态虚拟IP地址（静态地址和地址池必须在同一个子网中），然后为每一个成功建立SSL连接的客户端动态分配一个虚拟IP地址池中未分配的地址。这样，[物理](http://www.zclw.net/" \t "_blank)网络中的客户端和OpenVPN服务器就连接成一个虚拟网络上的星型结构局域网，OpenVPN服务器成为每个客户端在虚拟网络上的网关。OpenVPN服务器同时提供对客户端虚拟网卡的路由[管理](http://www.zclw.net/" \t "_blank)。当客户端对OpenVPN服务器后端的应用服务器的任何访问时，数据包都会经过路由流经虚拟网卡，OpenVPN程序在虚拟网卡上截获数据IP报文，然后使用SSL协议将这些IP报文封装起来，再经过[物理](http://www.zclw.net/" \t "_blank)网卡发送出去。OpenVPN的服务器和客户端在虚拟网卡之上建立起一个虚拟的局域网络，这个虚拟的局域网对[系统](http://www.zclw.net/" \t "_blank)的用户来说是透明的。

**客户端与服务端安全连接的建立**

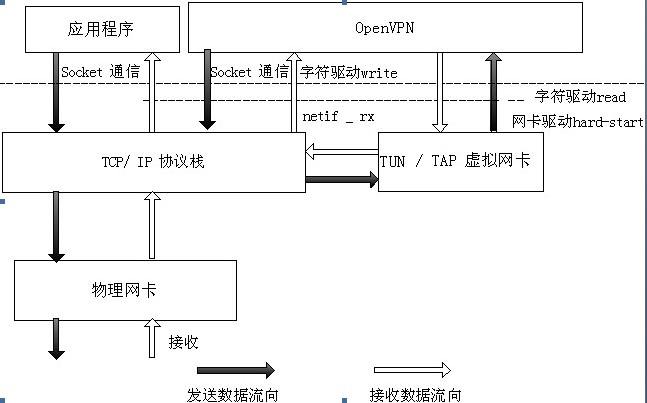
OpenVPN的服务器和客户端支持tcp和udp两种连接方式，只需在服务端和客户端预先定义好使用的连接方式（tcp或udp）和端口号，客户端和服务端在这个连接的基础上进行SSL握手。连接过程包括SSL的握手以及虚拟网络上的[管理](http://www.zclw.net/" \t "_blank)信息，OpenVPN将虚拟网上的网段、地址、路由发送给客户端。连接成功后，客户端和服务端建立起SSL安全连接，客户端和服务端的数据都流入虚拟网卡做SSL的处理，再在tcp或udp的连接上从[物理](http://www.zclw.net/" \t "_blank)网卡发送出去。

**数据包的处理过程**

发送数据流程：

应用层的外出数据，经过[系统](http://www.zclw.net/" \t "_blank)调用接口传入核心TCP/IP层做处理，在TCP/IP经过路由到虚拟网卡，虚拟网卡的网卡驱动发送处理程序hard\_start\_xmit()将数据包加入skb表并完成数据包从核心区到用户区的复制，OpenVPN调用虚拟网卡的字符处理程序tun\_read()，读取到设备上的数据包，对读取的数据包使用SSL协议做封装处理后，通过socket系统调用发送出去。

接收数据流程：  
物理网卡接收数据包，经过核心TCP/IP上传到OpenVPN，OpenVPN通过link\_socket\_read()接收数据包，使用SSL协议进行解包处理，经过处理的数据包OpenVPN调用虚拟网卡的字符处理程序tun\_write()写入虚拟网卡的字符设备，设备驱动程序完成数据从用户区到核心区的复制，并将数据写入skb链表，然后调用网卡netif\_rx()接收程序，数据包再次进入[系统](http://www.zclw.net/" \t "_blank)TCP/IP协议栈，传到上层应用程序。如图1所示。



在linux下，要实现内核空间和用户空间数据的交互，有多种方式：可以通用socket创建特殊套接字，利用套接字实现数据交互；

通过proc文件系统创建文件来进行数据交互；

还可以使用设备文件的方式，访问设备文件会调用设备驱动相应的例程，设备驱动本身就是内核空间和用户空间的一个接口，Tun/tap驱动就是利用设备文件实现用户空间和内核空间的数据交互。

由于半虚拟网卡只是到达全虚拟网卡的过渡，因此这里不考虑半虚拟网卡，另外全虚拟网卡方案中的 VTun 也只是到 OpenVPN 的过渡。而基于 IPSec 的 VPN 受到各方面的限制也不予考虑，再者，VPDN 的应用场合也有限制，也不予考虑。最终我们只剩下了基于虚拟网卡的 SSL VPN，当然正如 OpenVPN 官方网站所说，OpenVPN 相比 IPSec 所欠缺的就是各个操作系统的原生支持，虽然 OpenVPN 的设计非常优秀，但是在标准化方面还是比不过 IPSec，IPSec 的成功，和诸如 Cisco 等公司的推波助澜是分不开的，我们知道，第一流的公司做标准，第二流的公司做品牌，第三流的公司做产品，因此标准化是很重要的。在 Windows 上，如想用 VPDN，基本直接就能支持，然而想使用 OpenVPN，则必须安装客户端。另外效率问题也是 OPenVPN 的劣势，然而这不是根本的劣势，技术上的问题终究是可以得到解决的，我本人就曾经修改了 tun 驱动程序，使其吞吐量大幅提高...

Android VPN service

Vpnservice是安卓提供给开发者用于开发自己的VPN的服务。开发者继承这个Vpnservice，从而实现VPN。手机本身是有一块网卡，安卓虚拟出一个网卡，然后通过NAT，将真实网卡上的出站流量转发到虚拟网卡上，然后Vpnservice获取这个虚拟网卡上的“流量”，并转发给Vpn的服务端。

iOS NetworkExtension

[NetworkExtension](https://developer.apple.com/documentation/networkextension)是苹果提供的用于配置 VPN 和定制、扩展核心网络功能的框架。NE 框架提供了可用于定制、扩展 iOS 和 MacOS 系统的核心网络功能的 API。

Network Extension 最早出现在 iOS 8，不过那个版本不支持虚拟网卡，只能简单调用 iOS 系统自带的 IPSec 和 IKEv2 协议的 VPN。

在 iOS 9 中，开发者可以用 NETunnelProvider 扩展核心网络层，从而实现非标准化的私密VPN技术。最重要的两个类是 NETunnelProviderManager 和 NEPacketTunnelProvider。

WebSocket

WebSocket协议，客户端和服务端都可以主动的推送消息，可以是文本也可以是二进制数据。而且没有同源策略的限制，不存在跨域问题。