Surge 官方中文指引：理解 Surge 原理

Surge Networks Inc.

2020-08-27

目录

第一章　概述

Surge 是一个在 iOS 和 macOS 平台上的网络工具，其核心能力有四项：

- 接管：可以将设备发出的网络连接进行接管。Surge 支持代理服务和虚拟网卡两种方式接管。
- 处理：可以对被接管的网络请求和响应进行修改。包括 URL 重定向、本地文件映射、使用 JavaScript 自定义修改等多种方式。
- 转发：可以将被接管的网络请求转发给其他代理服务器。可以是全局转发，也可以按照非常灵活的规则系统确定出口策略。
- 截获：可以截获并保存网络请求和响应的具体数据，同时可对 HTTPS 加密流量进行 MITM 解密。

以上四项能力构成了 Surge 的核心工作流。但 Surge 的功能还不仅限于以上四点。比如你可以自定义 DNS 服务器、对全局配置 DNS-over-HTTPS 等。

第二章　接管

要想使 Surge 实现后续的转发、修改和截获等功能，首先需要 Surge 对网络连接进行接管。

在 macOS 和 iOS 下，要想使程序发出的网络连接被另一个程序所接管，而不是直接将数据发送到物理网卡，有以下三种方式：

1. 配置代理：如果系统配置了代理服务器，那么程序在执行网络请求的时候，就不会直接连接目标服务器，而是产生一个发向代理服务器的连接。利用这个特性，可以在本地启动一个代理服务，并配置系统代理为 127.0.0.1 （即本机）的一个端口，这样就可以接管网络请求。

但是，这种方式要求程序自身支持代理机制，系统的代理设置只是告知程序应该使用代理，需要程序自己完成代理的后续逻辑。好在，对于绝大部分带用户界面的程序，由于开发时使用了系统的高层网络框架（Cocoa/Cocoa Touch），开发者不需要进行任何额外的工作就可以支持代理。

而对于命令行程序，由于使用的是 POSIX 接口进行网络请求，该接口并没有对代理服务器提供内嵌支持，所以需要开发者自己完成对代理服务器的支持，这导致各种命令行程序对代理的支持情况和具体行为并不统一。同时由于大部分命令行程序并没有为 macOS 进行特殊处理，所以不会理会系统配置里的代理服务器设置。大部分命令行程序需要通过环境变量 https_proxy 和 http_proxy 去配置代理，还有一部分需要通过修改配置文件进行配置。

还有少量程序由于完全缺乏代理服务器的支持，无法通过这种方式去接管网络连接。

2. 虚拟网卡（Virtual Network Interface，简写为 VIF）：主流操作系统几乎都存在 TUN 和 TAP 两种虚拟网卡接口，原本是为了提供对 VPN 的支持。通过在系统中建立虚拟网卡并配置全局路由表，可以接管所有的网络请求。

这种方式对应程序来说是无感知的，所以并不需要程序主动提供支持，几乎所有程序都可以被这种方式接管网络请求。除非程序主动指定了物理网卡，绕过了默认的虚拟网卡。

3. Socket Filter：这是 macOS 的一项内核特性，可以通过注入一个 Kernel Extension（kext）对所有 socket 调用进行 hook，以此接管请求。

除系统自身的一些程序外，这种方式可以强制接管系统中所有程序的所有网络请求。如 Proxifier 和 Little Snitch 就使用了这种方式接管网络。

这三种方式各有优劣：

1. 方法 1 性能最优，对系统侵入性最小，无奈有部分程序不支持。
2. 方法 2 性能略低，因为截取到的流量是 IP 层的数据包，需要有一个 TCP 协议栈进行重组装，造成了额外的性能开销。
3. 方式 3 最暴力，对系统侵入性高，Kernel Extension 有可能造成整个系统的不稳定，Apple 已确认在未来的 macOS 中将取消对 Socket Filter 的支持。

Surge 主要使用方法 1 接管网络请求。方法 2 作为补充，接管不支持代理的程序。

- 对于 Surge iOS 版本，开启后会将自身注册为代理服务器，同时使用 Network Extension 接口建立了 TUN 虚拟网卡。
- 对于 Surge Mac 版本，开启「设置为系统代理」选项会将自身注册为代理服务器（即方法 1），开启「增强模式」选项将会建立虚拟网卡（即方法 2）。

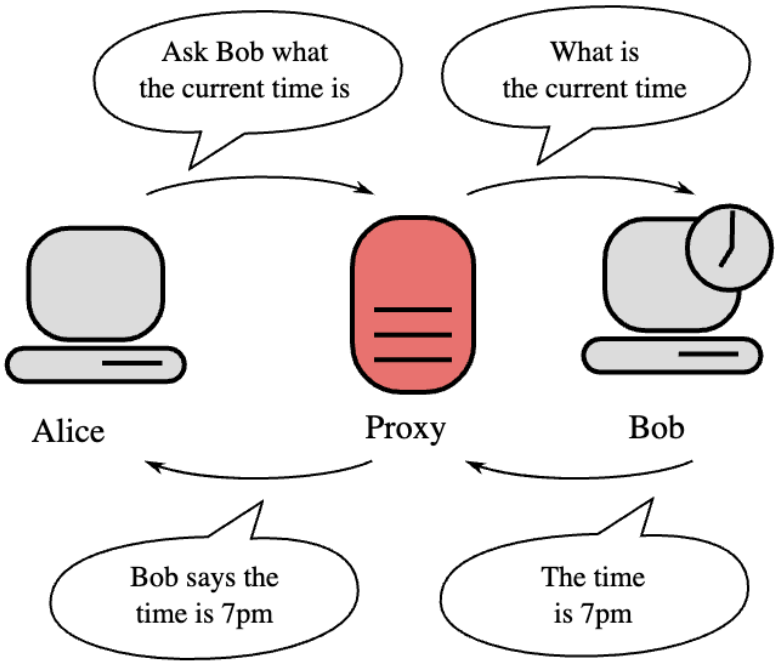
以上的说明针对的是 Surge 对本机程序的接管。当使用 Surge 接管另一个设备的网络请求时：

- 由于 iOS 的系统限制，只能靠使用方法 1 作为代理服务器去接管另一个设备的请求。（修改目标设备的代理服务器设置）
- Surge Mac 除了使用方法 1 外，也可以靠方法 2 接管另一个设备的请求。（修改目标设备的默认路由设置）

2.1 代理服务接管（方法 1）

2.1.1 什么是代理协议

代理是一项从计算机网络诞生就一直存在的机制，简单来说，代理服务器就是一个传话人，将应用和目标服务器间的数据由一个代理服务器中转传递。



当使用代理服务器时，除了发送原始的数据，还需要一些额外的工作：

1. 告知代理服务器，目标服务器的主机名和端口号。
2. 发送鉴权信息，供代理服务器进行身份验证。（可选）
3. 对数据传输进行加密。（可选）

如何进行这三项工作的规范，就是代理协议，有 RFC 规范的代理协议只有 HTTP 代理协议和 SOCKS 代理协议两种。SOCKS 代理协议有 SOCKS4、SOCKS4a、SOCKS5 三个版本。（macOS 使用的是 SOCKS5）

除了有 RFC 规范的代理协议，还有很多自定义的代理协议，如 shadowsocks、Snell 等。但是由于系统和程序没有内嵌对这些协议的支持，需要将他们通过一个客户端程序转换为标准的 HTTP 或 SOCKS5 代理服务供程序和系统使用。Surge 也可以充当这样的转换器，以 HTTP 代理协议和 SOCKS5 代理协议向系统和程序提供代理服务。

HTTPS 和 SOCKS-TLS 代理并没有 RFC 规范，只是在原协议套上了 TLS 层进行加密。

2.1.2 HTTP 代理和 TCP 代理

HTTP 代理仅可以转发 HTTP 协议的请求（除非该 HTTP 代理额外支持了 CONNECT 方法）。当使用一个 HTTP 代理时，向 HTTP 代理发出是一个完整的 HTTP 请求体，代理服务器收到该请求后进行转发，拿到完整的 HTTP 响应，再将该响应转发给客户端。所以 HTTP 代理是会话制，单个 HTTP 代理连接上可以不断地转发不同的 HTTP 请求，这些 HTTP 请求甚至可以不是同一个目标主机。

其他的代理协议都属于 TCP 数据流代理，仅仅是对 TCP 数据流进行了中转，也就是说代理并不关心和理解具体传递的是什么内容，只要是一个基于 TCP 协议的数据流，就可以被代理服务器所转发。

需要注意的是，当我们和目标服务器间使用的是 HTTPS 协议进行连接时，并不可以使用传统的 HTTP 代理协议，我们并不希望代理能够获知转发的内容，所以传递的内容的明文对代理并不可见。为此 HTTP 代理协议增加了 CONNECT 方法，可以将一个 HTTP 代理转变为一个 TCP 数据流代理，用于处理 HTTPS 请求。所以现在 HTTP 代理也可以被用来对任意 TCP 协议进行转发。

2.1.3 不包括简单主机名和忽略这些主机的代理设置

在 macOS 的网络设置中，有「不包括简单主机名」和「忽略这些主机和域的代理设置」两个选项，和代理设置相同，这两个选项也只是「告知」程序应该按照这样的行为去工作，所以需要应用自己提供对这两个选项的支持。

和代理本身的实现一样，绝大部分带用户界面的程序，由于开发时使用了系统的高层网络框架（Cocoa/Cocoa Touch），也自动就获得了对这两个选项的支持。而命令行程序几乎全都不支持这两个设置。

Surge 的 [General] 配置中的 exclude-simple-hostnames 和 skip-proxy 两个设置对应的就是这两个选项，在勾选「设置为系统代理」时会同时应用到系统中。

需要注意的是，如果方法 1 和方法 2 同时启动，那么被这两个设置排除的网络连接，同样会被方法 2 接管。所以该选项一般只用于处理某些特殊的兼容性问题，并不能使请求绕过 Surge。（对于 Surge Mac，如果未开启「增强模式」，确实可以绕过）

2.2 虚拟网卡接管（方法 2）

2.2.1 Fake IP

在 POSIX 规范下，执行网络请求需要先通过 gethostbyname 等类似方法进行 DNS 解析，再通过 connect 去连接获取到的 IP 地址，导致使用方法 2 接管请求时会遇到一个问题：必须要先进行 DNS 解析。

但是，如果该网络请求 Surge 决定交给代理服务器去转发，那么在本地进行的 DNS 查询就是无意义的，而且在某些情况下，该域名可能根本无法在本地进行解析。

为解决这个问题，Surge 的 VIF 在收到一个 DNS 查询时，并不会进行真正的 DNS 查询，而是直接返回一个 Fake IP 地址（通常为 198.18.x.x，该地址段并不会在公网上被使用）。后续收到发往 Fake IP 的 TCP 或 UDP 数据包时，将该虚假 IP 翻译回原始域名进行后续处理。

Surge 返回的 DNS 应答的 TTL 值（Time to live，可粗略理解为有效期）仅为 1 秒，所以该结果即用即抛，不必担心因 Fake IP 造成 Surge 关闭后网络异常。（不过的确观察到一些智能硬件未正确遵循 TTL 重新进行 DNS 查询，一般重启设备即可解决。）

在 Surge 早期的版本中，谨慎起见只会对规则中标记为 force-remote-dns 的主机名返回 fake IP。由于该选项经常对用户产生困扰，现在版本已取消了这个选项，对所有主机名都会返回 Fake IP 地址。配置 [General] 中的 always-real-ip 选项用于覆盖该行为，对于出现在该选项中的主机名，Surge 不会返回 Fake IP，会将该 DNS 查询转发给 DNS 服务器获得真实 IP 地址。

2.2.2 tun-excluded-routes 和 tun-included-routes 选项

在建立虚拟网卡时，Surge 会根据这两个选项加入额外的路由表，tun-excluded-routes 比较好理解，有些用户可能会问为什么会有 tun-included-routes 选项，不应该默认就包含了所有路由吗?

这里需要补充一些网络知识，对于主流操作系统，路由表条目的优先级是按照条目的子网覆盖域决定的，覆盖越小的路由表条目优先级越高，而非按照先后或者上下的顺序。

所以，即使 Surge 的 VIF 配置了 0.0.0.0/0 的默认路由表条目，物理网卡本身存在当前子网的路由表条目（如 192.168.1.0/24），该条目覆盖域小优先级更高。所以所有发往 192.168.1.x 的网络连接依然不会被 Surge 接管。如果配置了 tun-included-routes = 192.168.1.100/32，那么这条路由表条目覆盖域最小优先级最高，使得发往该 IP 的网络连接也能被 Surge 接管。

第三章　处理

在对连接进行处理之前，首先需要对被接管的网络连接进行分类，目前的 Surge 版本中主要有四种类型：

1. HTTP 连接：由 Surge HTTP 代理服务接管的连接。在 Dashboard 中会显示请求的完整 URL 和 HTTP 方法名（GET、POST、HEAD、PUT、DELETE、TRACE、OPTIONS 等）
2. HTTPS 连接：由 Surge HTTP 代理服务接管的，且使用 CONNECT 方法转变为 TCP 数据流的连接。在 Dashboard 中会显示请求的目标主机名和端口号，方法显示为 HTTPS。
3. TCP 连接：由 Surge VIF、Surge SOCKS5 代理服务接管的 TCP 连接。在 Dashboard 中会显示请求的目标主机名和端口号，方法显示为 TCP 和 SOCKS。
4. UDP 会话：由 Surge VIF 接管的 UDP 数据包，发往同一个地址和端口号的数据包构成了一个 UDP 会话。

对于第一类连接，可使用 Surge 的各项能力对请求进行修改、转发和抓取。对于第二类连接，在使用 MITM 进行解密后即暴露为标准 HTTP 连接，也可以使用完整的能力。对于第三类连接，一般情况下只能进行转发。

3.1　区分请求

3.1.1　HTTP

上述第三类连接也有可能是一个 HTTP 连接，但是由于 Surge 并没有方法准确确认，所以不会主动使用 HTTP 引擎去进行处理，否则会由于无法处

理导致连接中断。

有用户曾建议可对数据包进行识别去自动判断具体协议，出于以下两个原因 Surge 并没有这样的设计：

1. 虽然绝大多数基于 TCP 的协议都是由客户端发出第一个数据包，但是按照 TCP 协议标准这不是必须的，像 Telent 等少数协议就是由服务端先发出第一个数据包。

若 Surge 想要通过对数据包进行识别判断连接是否为 HTTP 协议，须等待客户端发出第一个数据包，如果遇上了服务器首先发数据包的协议，则会造成客户端一直等待。（虽然可以通过加入等待超时的方式去处理，但是很不优雅）

2. 存在着一些自定义协议，会发出带有 HTTP 请求头的数据包，但是后续数据并不按照 HTTP 规范进行，从 TCP 协议标准来说这是完全可接受的，但是如果交由 Surge HTTP 引擎处理则会引发异常。

所以，Surge 将选择权交给了用户，提供了 force-http-engine-hosts 参数，对于出现在该参数中的主机名，即使它是一个由 Surge VIF 或 Surge SOCKS5 所接管的第三类连接，也将强制使用 HTTP 引擎去进行处理，所有 HTTP 高级功能都可以被使用。但是如果该连接并非一个使用 HTTP 协议的连接，则会造成连接中断。

另外，如果 Surge 在一个 TCP 连接中发现了 HTTP 请求头或响应头，会在备注中标注：“HTTP request header found in the raw TCP connection.” 和 “HTTP response header found in the raw TCP connection.”，并提取请求和响应头，供简单分析。请注意该功能只会分析 TCP 数据流的双向首个数据包，由于 HTTP 协议通常会进行 TCP 连接复用，所以该功能看不到后续的 HTTP 请求，还是需要手动配置 force-http-engine-hosts 参数去开启完整的抓取。

3.1.2 HTTPS

由 HTTP 代理服务所接管的，使用 CONNECT 方法的连接，Surge 会判定为 HTTPS 连接，但其实该连接也可能是一个除 HTTPS 外的其他 TCP 协

议连接。

所以，默认情况下 Surge 对此类连接仅进行单纯的 TCP 数据流转发，如果目标主机名出现在 [MITM] 的 hostname 配置中，Surge 则会进行 HTTPS 解密，关于 HTTPS 和 MITM 的具体说明请参见后续章节。

3.1.2.1 对 TCP 连接进行 MITM 解密

当配置 [MITM] 段中的 tcp-connection 选项开启时，如果第三类连接的主机名出现在 [MITM] 的 hostname 配置中，Surge 也会进行 HTTPS 解密，并交由 HTTP 引擎处理，可开启所有高级功能。

但是如果该连接并非 HTTPS 协议，则会造成连接中断。

3.2 修改请求和响应

Surge 所有的修改功能都是为 HTTP 协议所设计，未解密的 HTTPS 连接和 TCP 连接将会跳过该步骤。

目前 Surge 提供的修改能力包括：

- 重定向（URL Rewrite，Map Remote）
- 本地文件映射（Map Local，API Mock）
- 请求头和响应头修改（Header Rewrite）
- JavaScript 脚本修改

其中脚本修改的功能最为强大，可通过脚本间接实现其他几项能力。但是由于脚本编写繁琐，运行时开销略大，简单需求还是应该通过其他方式实现。

3.2.1 HTTP 相关词汇中英翻译对照

请求：Request 请求头：Request Header 请求体：Request Body 响应：Response 响应头：Response Header 响应体：Response Body 头字段：Header Field

3.2.2 重定向（URL Rewrite，Map Remote）

Surge 提供两种进行 HTTP 重定向的实现方式：

1. 请求头修改：通过直接修改请求头内容实现，客户端程序对本次重定向无感知。为了保证重定向后的行为正确，URL 被修改后，Surge 会自动使用 URL 的主机名部分覆盖请求头的 Host 字段。脚本进行重定向时不会进行该行为。
2. 返回 302、307 响应：直接向 HTTP 连接返回一个状态码为 302 或 307 的完整 HTTP 响应，客户端需要支持 HTTP 重定向。

3.2.3 本地文件映射（Map Local，API Mock）

根据用途的不同，该功能在不同的软件中有不同的名称，如 Map Local，API Mock 等。但是实际指的都是同一类型的功能，即不进行真正的 HTTP 请求，直接返回一个预设的响应。

Surge 会根据本地文件的扩展名自动选择合适的 Content-Type 作为 HTTP 响应头字段，也可以自定义响应头字段覆盖该行为。

3.2.4 请求头和响应头修改（Header Rewrite）

该功能用于简单的修改请求头的字段，支持 add、del、replace 三种操作。

当使用 add 操作时，如果该字段名已经存在，会追加一个同名字段，这种行为是被 HTTP 标准所允许的，服务端应该将多个同名字段拼接后进行理解。但是由于部分 HTTP 服务器未正确遵循该规范，除非有特殊的需求，一般建议使用 del 和 add 组合操作，先删再加。

3.2.5 JavaScript 脚本修改

脚本修改提供最全面的修改能力，可以随意修改请求和响应的头字段和数据体，但是需要注意，目前 Surge 只支持对 UTF-8 编码的数据体进行脚本读取和修改。

关于脚本的详细使用方法请参见手册。

第四章　转发，代理和规则系统

请求在处理完毕后将被转发。如果 Surge 的出站模式设置为直接连接，那么该请求将被直接发往目标服务器，如果出站模式设置为全局代理，那么将转发给代理服务器。

当出站模式设置被设置为规则判定时，将根据配置的规则决定转发策略。

4.1　规则系统

规则系统中有两个基本概念：策略和规则

1. 策略：描述了 Surge 进行转发的方式，有三种类别：

- 内置策略：DIRECT、REJECT、REJECT-TINYGIF、REJECT-DROP
- 代理策略：每个策略对应一个代理服务
- 策略组：根据一定的规则从子策略中选择一个最终策略。
  - select 策略组：通过 UI 菜单选择一个策略。
  - url-test 策略组：选择延迟最低的策略。
  - fallback 策略组：选择可用的策略中，最靠前的策略。
  - ssid 策略组：根据当前的 Wi-Fi SSID 选择一个策略。
  - load-balance 策略组：随机使用一个子策略，可选进行可用性检查。

2. 规则：规则由四个部分组成：类型、条件、策略和参数。当条件满足时，该规则匹配，使用该规则指定的策略。

4.2 策略

下面具体说明各种策略的用途

4.2.1 内置策略

内置策略由 Surge 提供，不随配置而变化：

- DIRECT：将该请求直接发往目标服务器
- REJECT：拒绝该请求，当连接类型为 HTTP 时，会返回一个错误页面。（该行为可被 show-error-page-for-reject 参数控制）
- REJECT-TINYGIF：拒绝该请求，当连接类型为 HTTP 时，返回一个 1px 的 GIF 图片响应。若为其他类型连接则直接断开。该策略主要用于 Web 广告屏蔽。
- REJECT-DROP：拒绝该请求，与 REJECT 不同的是，该策略将静默抛弃请求。因为部分程序有着十分暴力的重试逻辑，连接失败后会立刻进行重试，导致请求风暴。

由于操作系统对用户空间程序（user-space program）的 socket 并没有提供抛弃的操作，Surge 静默抛弃的实现方式是将该 socket 闲置一段时间后再关闭。

同时，如果发往某主机名的请求短时间内大量触发 REJECT/REJECT-TINYGIF 策略（当前版本的阈值为 30 秒内 10 次），为了避免产生大量资源浪费，Surge 将自动升级策略为 REJECT-DROP 策略。

4.2.2 代理策略

代理策略由用户自己定义，每个策略描述了一个代理服务，当使用该策略时即为通过该代理服务转发请求。

一个简单的代理策略定义行如下：

```
ProxyA = http, 11.22.33.44, 8080, username=user, password=pass
```

其中，ProxyA 为策略名，供规则和策略组所使用。第一个参数为代理协议类型，目前 Surge 支持的代理协议类型有：http,https,socks5,socks5-tls,ss,snell,vmess,trojan，以及两个特殊类型 external 和 direct。第二个参数为代理服务器主机名，第三个参数为代理服务器端口号，后续为 key=value 的参数表，根据协议类型不同需要提供不同的参数。

不同的代理协议类型有其特定的参数，有些参数是所有代理策略通用的，这里单独讲解几个参数，完整参数表请参见手册：

- tfo：开启 TCP Fast Open，TFO 可以使 TCP 握手时顺带传递第一个数据包，以此降低代理协议握手的时间开销。但是由于这是 2014 年才引入的新特性，如果和目标服务器之前的设备有任何一个不支持该特性，则会导致异常，目前观察到大多数 ISP 的网络都有异常产生的机率，所以除非是内网代理服务器，否则不推荐开启。
- underlying-proxy：通过另一个代理使用该代理，该功能在其他软件中可能被称为代理链（Proxy chain）。（目前仅在 iOS beta 版本可用）
- interface：强制指定该代理使用某一网卡进行连接
- allow-other-interface：当 interface 参数指定的网卡不存在时，是否允许使用默认网卡进行连接。如果为 false 且指定的网卡不存在，连接将直接断开。
- always-use-connect：http 类型代理专用参数，如上一章所描述，当使用 HTTP 代理转发 HTTP 请求时，是按照会话制进行转发的，如果 always-use-connect=true，将会把远端 HTTP 代理当做一个标准的 TCP 代理使用。

4.2.2.1 direct 类型 (仅限 Surge Mac)

这是一个特别的类型，严格来说并不是一个代理，用于强制使用某一个网卡进行请求。

```
PolicyName = direct, interface=en2, allow-other-interface=false
```

请注意，由于 Darwin 内核的限制，使用的网卡必须拥有连接的目标主机的路由表条目，否则无法使用该网卡。

举例说明，比如同时连接了有线网络和无线网路，通过 netstat -rn 命令我们可以看到两个网卡 en0 和 en1 都拥有 default 的全覆盖路由表条目，只是优先级不同。

```
Internet:
Destination        Gateway            Flags        Netif Expire
default            192.168.20.1       UGSc           en0
default            192.168.20.1       UGScI          en1
```

这种情况下可以使用 direct 策略自由选择 en0 或者 en1 作为出口。

配合 VPN 使用

某些 VPN 连接后，会加入一个很小的内网路由表，由于覆盖小优先级高，可用于访问特定内网资源。在未开启增强模式时，如果出口策略是 DIRECT，Surge 也会按照这个行为进行本地转发。

但如果开启了增强模式，由于 Surge 的 VIF 已配置为默认路由，为了能将连接从本地物理网卡发出，出口连接将会强制绑定原先的默认网卡，无视路由表。这会导致 VPN 的内网路由表失效，该情况下可使用 direct 类型策略强制绑定 VPN 的 utun 设备解决。

```
[Proxy]
CorpVPN = direct, interface=utun1, allow-other-interface=true

[Rule]
DOMAIN-SUFFIX,internal.corp.com,CorpVPN
```

4.2.2.2 external 类型 (仅限 Surge Mac)

external 类型策略可以让 Surge 与其他代理客户端更方便的协同工作。

该功能目前只能通过直接编辑配置实现，策略定义行为：External = external, exec = “/usr/local/bin/local”, args = “-c”, args = “/usr/local/etc/config.json”, local-port = 1080, addresses = 11.22.33.44

其中 args 和 addresses 参数为选填，其他必填。args 和 addresses 字段可以反复使用进行追加。

当使用到该策略时 Surge 会进行以下工作：1. 使用 exec 和 args 参数启动该外部程序，之后向 SOCKS5 127.0.0.1:[local-port] 转发请求。2. 如果外部进程被终止，当再次使用该策略时会自动进行重启。3. Surge 会在启动增强模式时自动将 addresses 参数中的地址排除在 VIF 路由表外。（请在该字段填写使用的代理服务器 IP 地址）4. 当由 Surge 启动的外部进程的请求被 Surge VIF 处理时，永远使用 DIRECT 策略。（为了应对像 obfs-local 这样的插件请求问题，外部进程的子进程也会被同样处理）5. Surge 退出时会自动关闭所有外部进程，增强模式关闭时会自动清理加入的路由表。

上述 3 和 4 的功能是有重叠的，请尽量使用 addresses 声明使用到的地址以排除 TUN 处理，这样可以减少系统开销，4 的功能是一重额外保护。

4.2.3 策略组

Surge 提供多种不同类型的策略组以满足各种场景的不同需求，在具体讲解各种策略组前，需要先了解连通性测试。

4.2.3.1 连通性测试

Surge 的多个功能会用到连通性测试，测试方式有 3 种

1. ICMP Ping 测试：简单的 Ping 测试，用于反映当前物理网络状况。

Mac 版本首页卡片和网络诊断中的路由延迟为该测试结果。

2. DNS 查询测试：向所有 DNS 服务器并行查询 bing.com 域名的 A 记录，结果为收到响应的最短时间，用于反映当前物理网络状况，同时简单确认具有 Internet 访问。

Mac 版本首页卡片和网络诊断中的 DNS 延迟，Mac 版本主菜单和 iOS 版本通知中心插件的连通性测试延迟为该测试结果。

3. HTTP 测试：向目标 HTTP 服务器发出 HEAD 请求，计算收到响应头的时间，任意响应数据包均判定为有效。测试地址可自定义，建议选择在全球都有节点的 URL。

Mac 版本首页卡片的 Internet 和代理延迟，策略组的判断基准，网络诊断的代理测试为该测试结果。

策略组使用方法 3 作为判断基准而非方法 1 是因为：

1. 代理服务器可能有中转，Ping 测试只能表示到达中转服务器的延迟。
2. 除了与代理服务器间的连通性，代理服务器的 DNS 和出口网络情况也应该进行考量。
3. 某些代理协议因设计欠考虑，会引入额外的延迟开销，如 SOCKS5，也应当被考量。

4.2.3.2 url-test 策略组

并发测试所有子策略，选择延迟最低的策略。有以下几个参数

- url：用于测试的 URL。
- timeout：测试的最长等待时间，超过该时间的策略将标记为失败不再继续等待。
- interval：每次测试的间隔时间。所有类 url-test 组的测试时机为：
  - 首次使用时进行测试。
  - 后续使用该策略组时，如果上次测试的时间间隔已大于 interval 设置时间，则再次触发测试。
  - 当目前选中策略产生不可恢复性错误时，直接触发测试。
  - 网络切换后，将清理之前的测试结果，当策略组被使用时触发首次测试。
- tolerance：容忍度，如果某几个策略测试结果相差不大，那么会导致在这几个策略中频繁切换，如果策略的代理服务器的出口 IP 不同，可能会触发目标网站的风险控制。所以加入了容忍度设计，仅当新一次的测试结果中，最佳策略比原选中策略的延迟差大于容忍度时，才会切换至新的策略。
- evaluate-before-use：默认情况下，在首次使用策略组时将直接使用子策略中的第一个策略，同时触发延迟测试。如果配置了 evaluate-before-use=true，那么首次使用时将等待测试完毕后选择最佳策略。

4.2.3.3 fallback 策略组

与 url-test 组基本一致，区别是只关心子策略是否可用而不关心具体延迟，然后从可用策略中选择靠前的策略。可以通过调小 timeout 参数将缓慢线路也标记为不可用。该类型没有 tolerance 参数。

4.2.3.4 load-balance 策略组

负载均衡组，随机从子策略中选取一个策略使用。

当配置了 url 参数时，会按照 fallback 组的行为进行可用性检查，然后仅从可用的子策略中随机选取。

除 url、timeout、interval 外，还有一个参数：

- persistent：当 persistent=true 时，对于同一目标主机名，将尽量使用同一个策略。避免因出口 IP 不同而触发目标网站的风险控制。但当可用性改变时可能导致策略变化。

4.2.3.5 ssid 策略组

虽然名字依然是 SSID 策略组，但是功能已经扩展，可根据当前网络的 SSID、BSSID、路由 IP 地址等因素选择子策略。iOS 版本还可以为数据网络指定策略。

4.3 规则

Surge 使用规则系统来对选择每个连接的出口策略。规则的匹配方式为**自上而下，逐一测试**。最末尾规则一定是一个 FINAL 规则，当所有规则都不匹配时使用。

4.3.1 域名规则

当连接的目标主机名符合时，匹配该规则。

- DOMAIN：严格匹配某域名。
- DOMAIN-SUFFIX：匹配某域名及其子域名，如 DOMAIN-SUFFIX,apple.com 可以匹配 apple.com 和 www.apple.com，但是不会匹配 anapple.com。
- DOMAIN-KEYWORD：简单的字符串搜索，只要域名包含子串就会匹配。
- DOMAIN-SET：专为大量域名集列表文件设计，支持上万条记录的快速查询。文件中每行为一个域名，如果某行以. 开头则表示匹配所有子域名和该域名本身。可用于广告过滤。

4.3.1.1 域名和主机名

域名其实是主机名的一种形式，Surge 内部对域名和主机名并没有区分，所有文档所提到的域名和主机名所对应的处理逻辑都是一样的。

比如，DOMAIN,1.2.3.4 规则其实也可以用于匹配目标主机为 IP 地址 1.2.3.4 的连接。DOMAIN,MacBook.local 也可用于匹配 Bonjour 主机名。

4.3.2 IP 地址规则

当连接的目标主机的 IP 地址符合时，匹配该规则。包含 IP-CIDR，IP-CIDR6，GEOIP 三种类型。

当目标主机名是一个域名或主机名时，IP 类型规则会触发本地 DNS 解析。根据解析得到的 IP 地址进行判断。当解析失败是：* 如果最终的 FINAL 规则带有 dns-failed 标记，那么将直接匹配 FINAL 规则。* 如果 FINAL 规则不带有 dns-failed 标记，**该请求将直接失败**。

IP 类型规则有一个专有的参数 no-resolve，如果一个 IP 规则带有该参数，那么 1. 如果目标主机名是一个域名，那么将跳过该规则，不触发 DNS 解析。2. 如果目标主机名是 IP 地址，按规则进行判断。3. 如果目标主机名是一个域名，且先前出现的 IP 规则已经触发了 DNS 解析获得了 IP 地址，那么使用该 IP 地址进行判断。

由于 DNS 查询有时间开销，所以在配置规则时，最优的方式是尽量先不触发 DNS 解析，将所有会触发 DNS 解析的规则放在底部。这样应使用代理

策略的请求就完全避免了在本地进行 DNS 解析。

但是也不用刻意的去完全避免解析，因为一旦决定使用 DIRECT 策略，那么最终还是要进行解析。Surge 有完备的 DNS 缓存系统，不必在意短时间内的重复解析。

不过需要注意，如果某目标主机在本地 DNS 不可被解析，那么一定需要加入对应的规则，在触发 DNS 前就决定策略终止匹配。或者对 FINAL 规则加上 dns-failed 标记并使用代理策略。

4.3.3 HTTP 相关规则

仅对 HTTP 请求有效的规则，包含 URL-REGEX 和 USER-AGENT。

比较特殊的是，只有由于只有进行 MITM 解密后才可获取到 URL，所以 URL-REGEX 对未解密的 HTTPS 连接无效。但是 USER-AGENT 规则却对未解密的 HTTPS 也连接有效，因为程序在使用 HTTP 代理时，会在发送 CONNECT 请求时带上自己 User Agent 的明文。

4.3.4 其他规则

- PROCESS-NAME：仅对 Mac 版本有效，可以匹配程序名。
- SRC-IP：可匹配连接来源 IP 地址，接管其他设备连接时可使用。
- IN-PORT：Mac 版本支持多端口监听，可为不同监听端口配置特定的规则。
- DEST-PORT：可匹配目标主机的端口号。
- PROTOCOL：可根据连接的协议进行匹配，取值范围是 HTTP，HTTPS，TCP，UDP。（虽然逻辑关系上 HTTP 和 HTTPS 都是 TCP 的一种特殊形式，但是该规则将按照前一章的分类区别对待。）
- SCRIPT：可以使用 JavaScript 根据各种参数完全自由的选择策略。

4.3.5 规则集

RULE-SET 规则集可以将多个子规则放在一个单独的文件中，便于分享和复用。但是规则集中的规则不可以指定策略，整个规则集指向一个同一个策

略。

另外 Surge 自带了 SYSTEM 和 LAN 两个规则集，规则集包含的具体子规则会随 Surge 更新而有所调整。注意 LAN 规则集会触发 DNS 解析。

4.3.5.1 RULE-SET 和 DOMAIN-SET 的不同

RULE-SET 可包含所有类型的子规则，执行效率和在主配置中的规则没有区别，而 DOMAIN-SET 仅可使用 DOMAIN 和 DOMAIN-SUFFIX 两种形式的内容，使用了特别的逻辑进行优化，在内容非常多时性能有极大的提升。（千条以上，否则两者没有太大的区别）

4.3.6 逻辑规则

可通过 AND，OR，NOT 运算对所有规则类型进行组合使用。如

```
AND,((PROCESS-NAME,Google Chrome),(PROTOCOL,UDP)),REJECT
```

可以拦截 Chrome 发出的 UDP 数据包。

第五章　TLS，HTTPS 与 MITM

本章详细讲解了 TLS 和 HTTPS 的关系，TLS 的作用，以及怎样进行 MITM。本文只阐述方法、流程和高层抽象概念，尽量不涉及具体的算法和技术实现。

5.1　TLS 和 HTTPS 的关系

现代计算机网络体系结构的设计采用的是分层思想，HTTP 是基于 TCP 协议层的应用层协议。

TCP 层所提供的核心功能是可靠传输，上层应用不必担心数据包的构造、拆分、乱序、丢包等实现问题，当使用 TCP 协议与目标主机通信，即有了一个可靠的双向的流式数据通道。

HTTP 协议基于 TCP 协议的基础上，定义了更详细的数据传输标准，将数据流抽象成了会话制，客户端发送一个请求，服务端回应一个响应，一一对应，更方便使用。

HTTPS 则是在 TCP 层和 HTTP 层之间，又插入了一个 TLS 层，TLS 层可以使 TCP 层的数据流得到加密和安全保护。所有基于 TCP 协议的上层协议都可以靠这种方式获得保护，而不必对协议自身进行调整。(如 SMTP)

5.2 TLS 的作用

一般认为 TLS 层的作用就是数据加密，实际上 TLS 层承载了全面的安全特性，至少包括：

1. 私密性（Confidentiality）：即数据加密，哪怕攻击者从链路上自始至终地获取了完整的数据流，也无法解密出原始数据。
2. 真实性（Authenticity）：可以确认目标主机身份，比如当访问 example.com 时，即使整个物理网络遭到了劫持，也能保证访问的是 example.com 的主机而非其他冒牌主机。
3. 数据完整性（Integrity）：保证数据不可被修改，链路上有攻击者对数据流进行了修改一定会引发错误。

5.2.1 数据加密和完整性保护

简单来说，TLS 握手阶段通过非对称加密或密钥协商交换算法生成一个对称加密密钥，后续传输依靠该密钥进行加密和完整性保护，关于 TLS 如何进行数据加密和完整性保护的具体算法细节不在本章的探讨范围，可以查阅相关文献。

5.2.2 确认目标主机身份

讲解 TLS 如何确认目标主机身份前，需要补充一点简单的密码学知识：非对称加密。我们这里不去描述非对称加密的具体数学原理，仅简单描述非对称加密的用法。

5.2.2.1 非对称加密

对称加密，指的是加密与解密用的是同一个密钥。非对称加密，顾名思义，指的是加密与解密用的是不同的密钥，我们把用来加密的密钥称为**公钥（Public Key）**，用来解密的密钥称为**私钥（Private Key）**，当然只有该公钥配套的私钥才能去解密由该公钥加密的内容，这对配套的公钥与私钥称为**密钥对（Key Pair）**。密钥对满足：

- 不可能由公钥直接计算出私钥。
- 暴力破解的算力需求不现实，在密钥长度符合要求的前提下，完全不用担心被暴力破解的可能。

除了用于加解密，密钥对还可以用于签名（Signing），对于一段内容，可以用私钥生成一段签名（一般称作数字签名），公钥可以验证签名是否是由私钥所生成，以此确定该内容是由私钥持有者所认可的内容。同时保证了内容无法被篡改，只要内容有变化数字签名验证一定会失败。

最常见和著名的非对称加密算法是 RSA 算法，也还有其他的非对称加密算法。

5.2.2.2 X.509 证书链

接下来我们简略了解一下什么是证书，证书其实就是一堆键值对（Key-Value）构成的数据体，不同的用途时会有不同的字段内容。

1. 一份证书对应着一个密钥对，公钥是证书的一部分，而私钥由证书的拥有者私密保存。
2. 证书可以由另一个证书所签发，即该证书的内容中包含了来自上级签发证书的数字签名和上级签发者的信息。
3. 上级签发者的证书可以由上上级签发者所签发，构成证书链，一般 TLS 使用的是 3 级证书链，我们把最高级签发者称作根证书（或叫 CA 证书，Certificate Authority），中间的证书称作中级证书（Intermediate Certificate），最末端的证书称作服务器证书（Server Certificate）
4. 操作系统预置了很多 CA 证书，表示操作系统信任这些 CA 证书，中级证书通过出示 CA 证书的数字签名表示受到 CA 证书信任，服务器证书出示中级证书的数字签名表示受到中级证书信任，构成了信任链。

那么这个证书信任关系是怎样对应的真实世界的身份验证呢？通常来说，CA 证书的维护机构需要符合特定的安全审计，守规守法的运作，由他们再去挑选一些代理机构，授予中级证书，当有域名的持有者希望能获得一份表明自己身份的证书时，代理机构先验证申请者的身份（如通过域名的联系人邮箱），然后向申请者颁发证书。

上述体系即为 X.509 证书链的简略描述。

- 证书颁发的过程中，通常由申请者先产生密钥对，然后将公钥封装为 CSR (Certificate Signing Request)，然后发送给代理机构，代理机构确认身份后返回带有其中级证书数字签名的证书，证书的字段中包含申请者的域名。代理机构并不知道该证书的私钥。
- 操作系统会随着系统更新，根据业务需求不断的调整内置的根证书库。
- 有的软件会选择自己维护根证书库，忽略系统的根证书库，如 Firefox。
- 上述提到的最末端的证书更准确的名称是 End-entity Certificate 或 Leaf Certificate，由于本文主要讲述的是服务端证书的验证，为避免混淆故直接写作服务器证书（Server Certificate）。

5.2.2.3 TLS 握手

拥有上述基础知识后，我们可以开始讲解 TLS 是怎样确认目标主机身份的了。来看一下 TLS 握手的具体流程：

1. 客户端通过 SNI，明文告知服务端正在访问 example.com，请提供相应的证书。
2. 服务端确认自己拥有 example.com 的证书，向客户端提供自己的服务器证书和所有中级证书。
3. 客户端收到证书，确认证书的 Common Name（或者 SAN 字段）字段包含 example.com。
4. 确认验证该证书的证书链可被根证书库所信任。

现在我们已经能确认服务器提供的证书，确实是 example.com 的一个真实可信的证书了，唯一还需要确认的是，服务端拥有该证书的私钥。

5. 客户端随机生成一段内容，使用服务器证书的公钥加密后发给服务端。服务端用私钥解密后可通过该内容计算出后续传输阶段的对称加密密钥，称作会话密钥。

如果服务端没有该证书的私钥，那么则不可能计算出正确的会话密钥，也就无法继续和客户端的通讯。

（上述流程有所简化，且根据 TLS 版本与加密方式不同可能略有不一致，可阅读 https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/ 了解更多）

通过上述步骤，客户端终于和 example.com 建立了一个安全信道。可以开始后续的 HTTP 内容传输。

5.3 MITM 攻击

在解释清楚了 TLS 是怎样保证连接安全之后，我们想要解释怎样通过中间人攻击（MITM，Man-In-The-Middle）解密 TLS 流量就太简单了。

中间人攻击顾名思义，就是在客户端 A 和服务端 B 之间，插入一个中间人 C，以此截取明文内容。具体做法是劫持 A 往 B 的连接到 C 上，让 A 以为自己在和 B 通讯，而实际上是在和 C 通讯，C 再建立和 B 的连接，作为中间人在 A 和 B 间转发内容。

要实施 MITM 有两个条件：

1. 有能力劫持 A 的网络，一般代理服务或软件、VPN、ISP、Wi-Fi 提供者等链路拥有者可以轻松实现。（恶意软件也可以通过对操作系统 hook 实现，但是如果恶意软件已获得如此高的权限，也不必再通过 MITM 去解密流量，直接读取对应软件内存即可）
2. 需要突破 TLS 的目标主机身份机制。

根据前文的描述，因为是我们自己处于研究目的想要进行 MITM，所以突破的方式很简单，在系统的根证书库中插入一份自己拥有私钥的证书即可。

5.3.1 Surge 的 MITM 流程

我们再来完整的看一下 Surge 进行 MITM 的流程。

1. 用户配置 MITM 功能，Surge 在本地生成密钥对，产生一个根证书并安装进系统证书库。开启了 example.com 的 MITM 功能

2. 收到向 example.com:433 的 CONNECT 请求，进入 MITM 模式，直接向客户端表示已完成到服务器的 TCP 握手。
3. 客户端开始 TLS 握手，发出 ClientHello 消息。
4. Surge 立刻生成一份 example.com 的服务器证书，并由配置的根证书进行签名，以该证书和客户端完成握手。
5. 客户端进行 HTTP 层通讯，发送真实 HTTP 请求。
6. Surge 收到请求后，按照第二章的描述对请求进行修改，再判定出口策略。使用对应策略向真实的 example.com 发起连接并完成 TLS 握手，转发该请求。

不同的软件在进行 MITM 时有一个细节会有所不同，Surge 是在与客户端握手时，直接凭空生成了一份符合需要的新服务器证书。另一些软件的策略是，先暂缓与客户端的握手，立刻开始与服务端的握手，在完成服务端握手拿到服务端证书后，修改其公钥与签发者信息并用自己的根证书重新签名，再用这个证书与客户端完成握手。

第二种方案的好处是，这样客户端拿到的证书与真实服务器的差异非常小，可以解决一些少见的兼容性问题。Surge 之所以采用第一种方案，是因为 Surge 的规则系统允许以 HTTP 层的特征作为判断依据选择出口策略（如 URL），所以必须先完成握手获得 HTTP 层请求后，才可以去与真实目标服务器建立连接。

5.3.2 公开的根证书

实践中我们发现有些 MITM 工具软件为了减少工作量，没有提供本地生成根证书的功能。取而代之的是直接预置了一份根证书和证书的私钥。

这种做法是很不安全的，如果用户系统信任了该证书，一旦网络遭遇劫持，攻击者便可以利用这份公共证书的私钥进行 MITM 攻击解密流量。

请务必在本地生成自己独特的根证书，并妥善保存私钥。

5.3.3 对抗 MITM 攻击

作为软件开发者，如果不想自己的流量可被 MITM 工具所解密，需要进行 MITM 防御。

方法其实并不复杂，只需要抛弃 X.509 证书链验证逻辑，使用自己的逻辑进行验证即可。X.509 的诞生意义是为了服务浏览器，当用户访问某一个网站时，除了这个网站的域名外，是一无所知的。所以需要依赖证书链对证书的合法性进行验证。但是对于 App 来说并不存在这个限制，在 TLS 握手阶段，直接判断服务器证书的公钥是否为特定值即可，这样 MITM 工具便不可能绕过身份验证，也就无法解密流量。

实践中除了比对服务器证书的公钥，还有很多做法，这里不再展开。如果 App 进行 MITM 防御，想要继续进行 MITM，则必须使用越狱设备修改程序二进制或注入运行时代码，突破自定义的证书验证流程。

5.4 TLS 的其他细节

我们顺便再补充一些与 TLS 相关的细节。

5.4.1 常见的 HTTPS 错误

我们经常在浏览中看到的「不安全」错误，就是上述验证过程中由于各个环节失败所产生的错误，一般有：

- 名称不符：服务器证书的许可域名和正在访问的 URL 并不匹配。
- 证书过期：证书是一定会被设置有效期的，一般为一年，出现该错误一般说明网站维护人忘记更新证书，或者用户设置了错误的系统时间。
- 根证书不可信：说明服务器提供的证书链的根证书并没去在系统的证书库中。

如果是在一个可信网络下（如家庭宽带），遇到上述错误，通常是网站管理员的配置失误造成的，如果在公共网络中遇到以上错误，那么需要加倍小心，可能遇到了劫持。

5.4.2 SNI

之前我们在 TLS 握手中提到了 SNI，这里详细解释下 SNI 的作用。

首先我们需要知道，假设服务器有一个 IP 地址 11.22.33.44，且有多个域名 exampleA.com 和 exmapleB.com 都指向这个 IP，当客户端向服务端发起 TCP 连接时，服务端并不知道客户端是通过 IP 还是 exampleA.com 或者 exmapleB.com 进行访问的，因为 TCP 的元数据中仅包含了 IP 地址，域名在被客户端用来查询获取到 IP 后，完全不参与后续的 TCP 环节。

这在 HTTP 协议的实践中遇到了一个问题，由于 IP 地址的稀缺性，我们有时希望同一个 IP 地址（或者说同一台服务器），能够根据访问者访问的域名，提供不同的内容（即虚拟主机）。为了解决这个问题，浏览器会在发出 HTTP 请求时，在请求头中加上 Host 字段，内容为 URL 中的主机名（域名）部分。这样服务器便可通过 Host 字段区分访问的站点以返回不同内容。

HTTPS 也遇到了同样的问题，TLS 握手时需要根据访问的域名的不同，使用不同的服务器证书。由于 Host 的内容存在于握手后的加密传输中，如果握手都无法完成那么自然无法通过该字段解决问题。所以 TLS 在握手时，客户端（浏览器）会以明文发送客户端所访问的域名的，即 SNI，供服务器选择证书。

但这可能导致隐私泄露，使得链路拥有者可以知道用户访问的网站的域名。但由于用户所访问的 IP 地址是一定能被获知的，所以访问的域名的泄露的影响有多大其实并不好说。

另外对于如果 TLS 客户端是浏览器，由于 TLS 的 SNI 和 HTTP 的 Host 都是 URL 的主机名，所以这两者一定是一致的，但是对于其他 TLS 客户端却并不一定一致。比如 Surge 就支持自定义 TLS 握手的 SNI 内容。

5.4.3 前向安全（Forward Secrecy）

之前在讲解 TLS 握手时有提到，TLS 后续对称加密的所使用的会话密钥，是由服务器证书的私钥加密的随机数据计算而得的。

那么，如果攻击者保存了通讯的密文，假如未来某天服务器的私钥泄漏了，或者计算机科学的发展使得暴力破解私钥成为可能，那么就可以通过私钥解密出会话密钥，从而完全解密保存下来的密文。

为了解决这个缺陷，现在所使用的 TLS 协议在握手时会更复杂一点，不再简单地使用静态的非对称密钥对去传递会话密钥，转而使用密钥协商算法去生成临时的会话密钥。比如目前常见的 DHE 算法。

这里简单描述一下 DHE 的用法：1. 服务端每次都为新连接随机生成一个密钥对：服务端私钥和服务端公钥。2. 客户端每次都为新连接随机生成一个密钥对：客户端私钥和客户端公钥。3. 客户端和服务端通过交换公钥。4. 客户端根据服务端公钥、客户端私钥和客户端公钥，通过算法计算出结果 1。5. 服务端根据客户端公钥、服务端私钥和服务端公钥，通过算法计算出结果 2。6. 算法保证了结果 1 和结果 2 一定是相等的，使用该结果进一步加工即可得到会话密钥。7. 服务端私钥、客户端私钥、结果 1、结果 2、会话密钥均只存在于两端的内存中，连接结束后，这些数据将被彻底抛弃不可恢复。

在这样的密钥交换机制下，由于攻击者仅能保存公开交换的客户端公钥和服务端公钥，任何人都无法再计算出用于会话密钥。

此项特性即称为前向安全（Forward Secrecy），也叫做完美前向安全（Perfect Forward Secrecy）。TLS 在握手时会根据客户端和服务器情况自动选择是否使用具有前向安全的密钥交换算法。

5.4.4 TLS Cipher Suite

综上所述，TLS 协议在握手阶段，服务器和客户端需要协商出数个结果

1. TLS 协议版本
2. 密钥交换算法
3. 签名算法
4. 对称加密算法
5. 哈希算法

协商的方法其实很简单，客户端先告状服务端自己上述 5 个项目所支持的组合，服务端再按照自己所支持的组合，选择尽可能安全的一个结果告知客户端。这个组合称为 TLS Cipher Suite。

以目前最常见的几个 TLS Cipher Suite 举例：

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256：这是一个 TLS 1.2 版本下的组合，密钥交换算法为 ECDHE，签名算法为 RSA，对称加密算法为 AES-128-GCM，哈希算法为 SHA256。

- TLS_AES_256_GCM_SHA384：这是一个 TLS 1.3 版本下的组合，TLS 1.3 标准只支持使用 ECDHE 算法进行密钥交换所以不需要协商，同时由于所使用的加密算法均为 AEAD 算法已自带完整性保护，不再需要进行单独的完整性保护，签名算法也不用再协商。组合中只包含对称加密算法为 AES-256-GCM，哈希算法为 SHA384。

如果你使用了一个基于 TLS 的代理协议，可以在 Surge 的 Dashboard/Recent Requests 的备注中，看到代理连接所协商出的 Cipher Suite。

第六章　DNS

Surge 完全抛弃了系统的 DNS 解析，全部自行实现。

6.1　并发解析

Surge 会同时向所有配置的 DNS 上游服务器进行 DNS 查询，并选取最快的返回结果，以提高性能。该特性和 dnsmasq 的实现一致。

6.2　Optimistic DNS（DNS 乐观解析）

由于现代网络的复杂性，大多数网站会将 DNS 的记录有效期（TTL）配置为很短的时间，如 30 秒。这样可以使得网络管理员修改 DNS 记录后迅速生效，不必再等待所有节点 TTL 超时，利于故障排除和维护。

这是可以理解的，网站和 API 可用性是很多公司的重中之重，如果某个 IP 不可达，修改 DNS 记录后需要 24 小时才能完全生效，造成的损失不可估量，所以运维会选择很短的 TTL。

但这带来了一个问题，客户端会严格按照 TTL 去进行查询，那么每隔很短的时间就会进行再次查询，一次 DNS 查询的时间开销短至几毫秒，然而最长可以要数秒。频繁重复查询会造成不必要的延迟。

为此 Apple 在 WWDC 2018 提出了 Optimistic DNS 的优化方案，在建立新连接时，如果本地 DNS 缓存已经过期，那么也先继续使用旧的结果，同时进行 DNS 查询，如果连接建立失败，则用新的结果重试。

绝大多数情况下，DNS 记录是不变的，这样的方案根本不会影响正常使用，当极小概率遇到 DNS 记录切换时，也只会耽误一两个请求，可以说是很完美的优化。

不过受限于 POSIX 等限制，Apple 也并没有将这个优化应用到所有地方，Surge 则完全应用了这个优化方案，使得所有请求都可以享受到 Optimistic DNS 的优化。

6.3 本地映射

Surge 支持配置本地 DNS 映射，功能和 /etc/hosts 文件基本一致。除了直接指定主机名所对应的 IP 地址，还支持对特定域名自定义特定的 DNS 服务器。或者完全通过脚本去自定义解析逻辑。

6.4 使用系统的解析

Surge 支持配置部分域名回退到系统 DNS 解析（example.com = server:syslib），用于解决一些兼容性问题，比如一些 VPN 会利用 Split DNS 机制在系统中添加用于处理特定域名的 DNS 服务器，Surge 目前还不能支持这种复杂逻辑，可通过对 VPN 相关域名配置回退解决。