Contents

[Contents 1](#_Toc157948271)

[HTTP & HTTPs 1](#_Toc157948272)

[基本概念：4/7分层架构,HTTP, TCP, keep-alive, 状态管理 1](#_Toc157948273)

[客户端请求页面流程 4](#_Toc157948274)

[HTTPs加密: 通信加密 (SSL/TLS) 和内容加密 9](#_Toc157948275)

[TCP/IP 17](#_Toc157948276)

[常见概念: 分组交换,TCP,网关, 18](#_Toc157948277)

[数据发送/接收流程 21](#_Toc157948278)

[应用：netsh, tcpdump 29](#_Toc157948279)

# HTTP & HTTPs

Books:

《图解HTTP》

《HTTP权威指南》

## 基本概念：4/7分层架构,HTTP, TCP, keep-alive, 状态管理

协议：计算机与网络设备要相互通信，双方就必须基于相同的方法。如探测到通信目标、由哪一边先发起通信、使用哪种语言进行通信、怎样结束通信等规则都需要事先确定

DNS协议：提供通过域名查找IP地址，或逆向从IP地址反查域名的服务

IP地址：节点被分配到的地址

MAC地址：网卡所属的固定地址

ARP:是一种用以解析地址的协议，根据通信方的IP地址就可以反查出对应的MAC的地址。

路由选择：有点像快递公司的送货过程。

**TCP/IP的分层架构模式：**

应用层(HTTP, FTP, …)

传输层(TCP/UDP): 提供处于网络连接中的两台计算机之间的数据传输

网络层(IP): 处理在网络上流动的数据包

链路层：处理连接网络的硬件部分

首先作为发送端的客户端在应用层(HTTP协议)发出一个想看某个Web页面的HTTP请求

接着，为了传输方便，在传输层(TCP协议)把从应用层处收到的数据（HTTP请求报文）进行分割，并在各个报文上打上标记序号及端口后转发给网络层

在网络层(IP协议)，增加作为通信目的地的MAC地址后转发给链路层

发送端，每通过一层则增加首部；接收端，每通过一层则删除首部

**HTTP是无状态协议**

HTTP协议对发送过的请求或响应都不做持久化处理， 使用HTTP协议，每当有新的请求发送时，就会有对应的新响应产生。协议本身不保留之前一切的请求或响应报文的信息。这是因为更快地处理大量事务，确保协议的保伸缩性，而特意把HTTP协议设计成如此简单

用户场景需要状态，如何处理？

比如用户登录到一家购物网站，即使他跳转到该站的其他页面后，也需要能继续保持登录状态

HTTP/1.1虽然是无状态协议，但为了实现期望的保持状态功能，于是引入了Cookie技术，有了Cookie再用HTTP协议通信。就可以管理状态

**TCP协议的分割和三次握手**

如何更容易传送大数据？

将大块数据分割成以报文段为单位的数据包进行管理。

如何确保数据能到达目标？

TCP协议采用三次握手策略

发送端首先发送一个带SYN标志的数据包给对方。

接收端收到后，回传一个带有SYN/ACK标志的数据包以示传达确认信息

最后，发送端再回传一个带ACK标志的数据包，代表“握手”结束

（若在握手过程中某个阶段莫名中断，TCP协议会再次以相同的顺序发送相同的数据包）

**持久链接 HTTP keep-alive**

每进行一次HTTP通信就要断开一次TCP连接

建立TCP连接 -> HTTP请求/响应 -> 断开TCP连接

若html里含js, css, jpg等，在发送请求访问HTML页面资源的同时，也会请求该HTML页面里包含的其他资源。因此每次的请求都会造成无谓的TCP连接建立和断开，增加通信量的开销

解决办法：采用持久连接，HTTP keep-alive。只要任意一端没有明确提出断开连接，则保持TCP连接状态。在HTTP/1.1中，所有的连接默认都是持久连接

建立TCP连接 -> HTTP请求/响应 -> HTTP请求/响应 … -> 断开TCP连接

持久连接使得多数请求以管线化pipelining方式发送成为可能。也就是说从前发送请求后需要等待并收到响应，才能发送下一个请求。管线化是指能够做到同时并行发送多个请求，而不需要一个接一个地等待响应

建立TCP连接 -> HTTP请求1, HTTP请求2, … -> HTTP响应 … -> 断开TCP连接

**Cookie的状态管理**

假设要求登录认证的web页面本身无法进行状态的管理(不记录已登录的状态)，那么每次跳转新页面不是要再次登录，就是要在请求报文中附加参数来管理登录状态

Cookie技术通过在请求和响应报文中写入Cookie信息来控制客户端的状态。

Cookie会根据从服务端发送的响应报文内的一个叫做Set-Cookie的首部字段信息，通知客户端保存Cookie。当下次客户端再往该服务器发送请求时，客户端会自动在请求报文中加入Cookie值后发送出去

服务器端发现客户端发送过来的Cookie后，会去检查究竟是从哪一个客户端发来的连接请求，然后对比服务端的记录，最后得到之前的状态信息

第一次请求：

客户端：没有Cookie信息状态

服务端：保存请求，生成Cookie，记住是向谁发送的，在响应中添加Cookie后返回

HTTP/1.1 200 OK

Date: ..

Server: Apache

<Set-Cookie: sid=…>

第二次请求：

客户端：存有Cookie信息状态，请求中添加Cookie后发送

GET /img/ HTTP/1.1

Host: hackr.jp

Cookie: sid=…

服务端：检查Cookie,发回响应

**编码提升传输速率**

压缩传输的内容编码，常用的内容编码：gzip, compress, deflate, indentity

**发送多种数据的多部分对象集合(multipart)**

发送的一份报文主体内可含有多类型实体。通常是在图片或文本文件等上传时使用

在web表单文件上传时使用

Content-Type: multipart/form-data; boundary=Aa

--Aa

…

--Aa--

状态码206(Partial Content部分内容)响应报文包含了多少范围的内容时使用

HTTP/1.1 206 Partial Content

Content-Type: multipart/byteranges; boundary=THIS\_STRING\_SEP

-- THIS\_STRING\_SEP

…

-- THIS\_STRING\_SEP—

**恢复下载中断处的下载？**

获取部分内容的范围请求

Range: byes=5001-10000

**内容协商返回最合适的内容**

同一个Web网站有可能存在着多份相同内容的页面，比如英文版，中文版等。当浏览器的默认语言为英语或中文，访问相同URI的web页面时，则会显示对应的web页面。这样的机制称为内容协商(content negotiation)

内容协商机制：是指客户端和服务器端就响应的资源内容进行交涉，然后提供给客户端最为适合的资源。内容协商会以响应资源的语言、字符集、编码方式等作为判断的基准

Accept

Accept-Charset

Accept-Encoding

Accept-Language

Content-Language

## 客户端请求页面流程

客户端

HTTP协议：生成针对目标web服务器的HTTP请求报文， 采用简单的请求-响应模式。一个HTTP请求只处理一个资源, 浏览器如果需要请求图片和视频，它会发送另一个HTTP请求

HTTP协议是一种文本协议。

TCP协议：为了方便通信，将HTTP请求报文分割成报文段，按序号分为多个报文段，把每个报文段可靠地传给对方（通过三次握手）

IP协议：搜索对方的地址，一边中转，一边传送

服务端

TCP协议：从客户端接收到报文段，重组报文段，按序号以原来的顺序重组请求报文

HTTP协议：对Web服务器请求的内容的处理

请求的处理结果也同样利用TCP/IP通信协议向用户进行回传

**URI格式：**

http://user:pass@www.example.jp:80/dir/index.htm?uid=1#ch1

协议方案名://登录信息（认证）@服务器地址：端口号/带层次的文件路径?查询字符串#片段标识符

**HTTP报文：多行数据构成的字符串文本**

请求报文：请求方法，请求URI, 协议版本、可选的请求首部字段和内容实体

HTTP POST请求的格式：

POST /path HTTP/1.1

Header1: Value1

Header2: Value2

Header3: Value3

body data goes here...

example:

POST /form/entry HTTP/1.1

Host: hackr.jp

Connection: keep-alive

Content-type: application/x-www-form-urlencoded

Content-length: 16

name=qzlin&age=35 // body data

响应报文： 协议版本、状态码、用以解释状态码的原因短语、可选的响应首部字段以及实体构成

HTTP响应的格式：

200 OK

Header1: Value1

Header2: Value2

Header3: Value3

body data goes here...

Example:

HTTP/1.1. 200 OK

Date: Tue, 10 Jul 2012

Content-Length: 362

Content-type: text/html

<html> // body data

…

每个Header一行一个，换行符是\r\n,当遇到连续两个\r\n时，Header部分结束，后面的数据全部是Body。

Body的数据类型由Content-Type头来确定，如果是网页，Body就是文本，如果是图片，Body就是图片的二进制数据。

当存在Content-Encoding时，Body数据是被压缩的，最常见的压缩方式是gzip，所以，看到Content-Encoding: gzip时，需要将Body数据先解压缩，才能得到真正的数据。压缩的目的在于减少Body的大小，加快网络传输。

**HTTP状态码**

* 1XX Informational信息性状态码 接收的请求正在处理
* 2XX Success成功状态码 请求正常处理完毕

200 OK

204 No Content 一般在只需要从客户端往服务器发送信息，而对客户端不需要发送新信息内容的情况下使用

206 Partial Content 响应报文中包含由Content-Range指定范围的实体内容

* 3XX Redirection重定向状态码 需要进行附加操作以完成请求

301 Moved Permanently 永久性重定向

302 Found 临时重定向

和301 Moved Permanently区别：302状态码代表的资源不是被永久移动，只是临时性质的。换句话说，已移动的资源对应的URI将来还有可能发生改变

303 See Other

当使用POST方法访问CGI程序，其执行后的处理结果是希望客户端能以GET方法重定向到另一个URI上去时，返回303状态码。

304 Not Modified

该状态码表示客户端发送附带条件的请求时，服务端允许请求访问资源，但未满足条件的情况。304状态码返回时，不包含任何响应的主体部分

307 Temporary Redirect

* 4XX Client Error客户端错误状态码 服务器无法处理请求

400 Bad Request

401 Unauthorized

当浏览器初次接收到401响应，会弹出认证用的对话窗口。表示需要通过HTTP认证

403 Forbidden 未获得文件系统的访问授权

404 Not Found

* 5XX Server Error服务器错误状态码 服务器处理请求出错

500 Internal Server Error

该状态码表明服务器端在执行请求时发生了错误。也有可能是web应用存在的bug或某些临时的故障

503 Service Unavailable

该状态码表明服务器暂时处于超负载或正在进行停机维护，现在无法处理请求

重点：状态码和状况可能不一致，因为不少返回的状态码响应都是错误的，但是用户可能察觉不到这点。比如Web应用程序内部发生错误，状态码依然返回200 OK

**与HTTP协作的web服务器**

HTTP/1.1规范允许一台http服务器搭建多个web站点，是利用了虚拟主机的功能

在相同的ip地址下，由于虚拟主机可以寄存多个不同主机名和域名的web网站，因此在发送http请求时，必须在host首部内完事指定主机名或域名的uri

在客户端和服务器之间，可能存在通信数据转发程序：代理、网关、隧道

代理：接收由客户端发送的请求并转发给服务器，同时也接收服务器返回的响应并转发给客户端。每次通过代理服务器转发请求或响应时，会追加写入via首部信息

使用代理服务器的理由：利用缓存技术减少网络带宽的流量，组织内部针对特定网站的访问控制，以获取访问日志为主要目的，等等

缓存是指代理服务器或客户端本地磁盘内保存的资源副本

即使存在缓存，也会因为客户端的要求、缓存的有效期等因素，向源服务器确认资源的有效性

网关：接收从客户端发送来的请求时，它就像自己拥有资源的源服务器一样对请求进行处理。利用网关能提高通信的安全性，因为可以在客户端与网关之间的通信线路上加密以确保连接的安全

隧道：客户端和服务器之间的中转，并保持双方通信连接的应用程序。一般使用SSL等加密手段进行通信

**HTTP报文首部**

首部内容为客户端和服务器分别处理请求和响应提供所需要的信息

使用首部字段是为了给浏览器和服务器提供报文主要大小、所使用的语文、认证信息等内容

HTTP请求报文: 方法、URI、HTTP版本、HTTP首部字段（请求首部，通用首部+实体首部）

HTTP响应报文: HTTP版本、状态码、HTTP首部字段（响应首部，通用首部+实体首部）

通用首部

Cache-Control: private, max-age=0, no-cache

控制缓存的行为

Connection: Keep-Alive

http/1.1版本的默认连接都是持久连接，客户端会在持久连接上连续发送请求。当服务器端想明确断开连接时，则指定Connection首部字段的值为Close

Date：Tue, …

创建报文的时间

Transfer-Encoding: chunked

传输报文主体时采用的编码方式

Via: 1.0 gw.hackr.jp

为了追踪传输路径

请求首部字段

Accept: type/subtype

通知服务器用户代理能够处理的媒体类型及媒体类型的相对优先级

文本文件：text/html, text/plain, text/css, … application/xhtml+xml, …

图片文件：image/jpeg, image/gif, …

视频文件：video/mpeg, video/quicktime, …

应用程序使用的二进制文件：application/octet-stream, application/zip…

Accept-Charset: Unicode

通知服务器用户代理支持的字符集及字符集的相对优先顺序

Accept-Encoding: gzip, deflate

通知服务器用户代理支持的内容编码及内容编码的优先级顺序

Accept-Language: zh-cn, zh;q=0.7, en-us, en; q=0.3

通知服务器用户代理能够处理的自然语言集及优先级顺序

Authorization: Basic ddeeedd

通知服务器用户代理的认证信息（证书值）

Expect: 100-continue

From: [info@hackr.jp](mailto:info@hackr.jp)

Host: [www.hackr.jp](http://www.hackr.jp/)

因为虚拟主机运行在同一个ip上，因此使用首部字段Host加以区分

请求被发送至服务器时，请求中的主机名会用IP地址直接替换解决。但如果这时，相同的IP地址下部署运行着多个域名，那么服务器就会无法理解究竟是哪个域名对应的请求。因此，就需要使用首部字段Host来明确指出请求的主机名

条件请求

If-Match 若If-Match的值与服务器资源的ETag匹配，则执行请求

If-None-Match

If-Modified-Since

If-Range 若If-Range的值与ETag匹配，则配合Range执行范围请求处理

If-Unmodified-Since

Max-Forwards: 10

使用HTTP协议通信时，请求可能会经过代理等多台服务器，服务器在往下一个服务器转发请求之前，会将Max-Forwards的值减1后重新赋值。当服务器接收到Max-Forwards值为0的请求时，则不再进行转发，而是直接返回响应

Proxy-Authorization: Basic …

认证行为发生在客户端 与代理之间

Range: bytes=5001-10000

会在处理请求之后返回状态码为206 Partial Content的响应。无法处理该范围请求时，则会返回状态码200 OK的响应及全部资源

User-Agent: Mozilla/5.0 …

将创建请求的浏览器和用户代理名称等信息传达给服务器

响应首部字段

Accept-Ranges: bytes

告知客户端服务器是否能处理范围请求，以指定获取服务器端某个部分的资源

Age: 600

告知客户端，源服务器在多久前创建了响应，单位为秒

ETag: “12222…”

告知客户端实体标识。它是一种可将资源以字符串形式做唯一性标识的方式。服务器会为每份资源分配对应的ETag值

Location: [http://www](http://www/)...

几乎所有的浏览器在接收到包含首部字段Location的响应后，都会强制性地尝试对已提示的重定向资源的访问

Proxy-Authenticate: Basic realm=”…”

将代理服务器所要求的认证信息发送给客户端

WWW-Authenticate: Basic realm=”…”

告知客户端访问请求URI指定资源需要的认证方案Basic, Digest, …

状态码401 Unauthorized响应中，肯定带有首部字段WWW-Authenticate

Server: Apache/2.2 (unix) PHP/5.

实体首部字段

Allow: GET, HEAD

通知客户端，服务器支持的HTTP方法

Content-Encoding: gzip

实体主体部分选用的内容编码方式

Content-Language: zh-CN

Content-Length: 15000 实体主体的大小，单位是字节

Content-Location: [http://www](http://www/)...

Content-MD5: dfasfdd…

客户端会对接收的报文主体执行相同的MD5算法获得的128位二进制数，然后经Base64编码，最后与首部字段Content-MD5的字段值比较

Content-Range: bytes 5001-

Content-Type: type/subtype

Expires: Wed, …

告诉客户端该资源失效的日期，若同时存在Cache-Controll的max-age指令时，会优先处理max-age指令

Last-Modified: Wed, …

服务端发送的Set-Cookie

Set-Cookie: status=enable; expires=Tue, …; path=/; domain=.hackr.jp; secure HttpOnly

当expires省略时，其有效期仅限于维持浏览器会话session时间段内，这通常限于浏览器应用程序被关闭之前

Secure属性用地限制web页面伏在https安全连接时才可以发送Cookie

Httponly属性使用cookie不能被javascript脚本访问

客户端带的Cookie

Cookie:status=enable

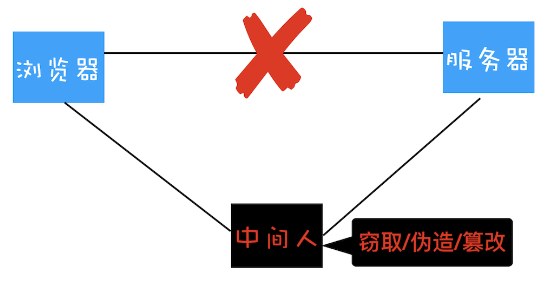
当客户端想获得http状态管理支持时，就会在请求中包含从服务器接收到的cookie

## HTTPs加密: 通信加密 (SSL/TLS) 和内容加密

HTTP缺点：

* 通信使用明文，内容可能会被窃听
* 不验证通信方的身份，因此有可能遭遇伪装
* 无法证明报文的完整性，所以有可能已遭篡改

HTTP报文使用明文方式发送，而TCP/IP协议族的工作机制，通信内容在所有的通信线路上都有可能遭到窥视。窃听相同段上的通信并非难事，只需要收集在互联网上流动的数据包就行了。对于收集来的数据包的解析工作，可交给那些抓包或嗅探器工具

 A blue squares with red text

Description automatically generated

应用层HTTP, FTP,SMTP, POP, Telnet -> SSL/TLS -> TCP/IP

SSL/TLS (标准化SSL): Secure Sockets Layer 安全套接层 Transport Layer Security 传输层安全

介于应用层和TCP层之间，在传输层对网络连接进行加密。强化应用层协议的安全性。

HTTPS = HTTP + SSL/TLS，经常会在web的登录页面和购物结算界面等使用https通信。

客户端通过HTTP请求访问服务端经过三次握手，通过HTTPS七次握手即TCP三层握手 + TLS四次握手。

证书和密钥：

证书是用于验证身份和加密通信的数字凭证；密钥是用于加密解密数据的密码。

证书和密钥通常一起使用，安全通信使用证书验证通信双方身份，通信双方分别持有自己的私钥和对方的公钥（实现加密和解密，从而确保通信双方之间的数据传输过程不被窃取或篡改）

密钥：加密解密 （注意：加密算法是公开的）

对称性密钥 （共享密钥）：使用同一把密钥加密和解密

非对称密钥（公开密钥）：使用公开密钥加密，使用私有密钥解密

应用场景：

* 发送方使用接收方的公钥加密数据，接收方使用自己的私钥解密数据。（注意：不需要发送用解密的私有密钥）
* 发送方使用自己的私钥加密数据（即数字签名），生成唯一标识符并附到数据中。接收方使用发送者的公钥对数据解密(即验证签名)，若解密出来的标识符匹配则真实无篡改。（即数字签名原理，用于验证文档或消息真实性，使用非对称密钥和证书实现）

服务端如何获取证书？

个人或公司申请者如\*.csdn.net向CA（certificate authority数据证书认证机构）提出公开密钥的申请，CA会对申请的公开密钥做数字签名，然后将公开密钥放入公钥证书后绑定在一起，公钥证书也叫做数字证书或证书

证书是由CA机构颁发，包含实体（个人或公司）的身份信息、公钥及其他元数据，通过数字签名确保其真实性和完整性。通过使用证书，可以在网络上进行安全通信。比如HTTPS加密连接就是基于服务器的SSL/TLS证书来建立安全通道。

个人或公司申请者如\*.csdn.net向中间CA去申请证书，根CA是给中间CA做认证。

浏览器验证服务端证书？

输入网页，服务器将公钥证书发送给客户端

浏览器会先验证服务器\*.csdn.net的证书（客户端使用证书里的公钥进行数字签名验证），如果合法再验证中间CA的证书GeoTrust，最后验证根证书DigiCert（若根证书在操作系统里，浏览器就认为根证书合法）。

点击浏览器chrome地址栏左边的小锁如图显示数字证书链：

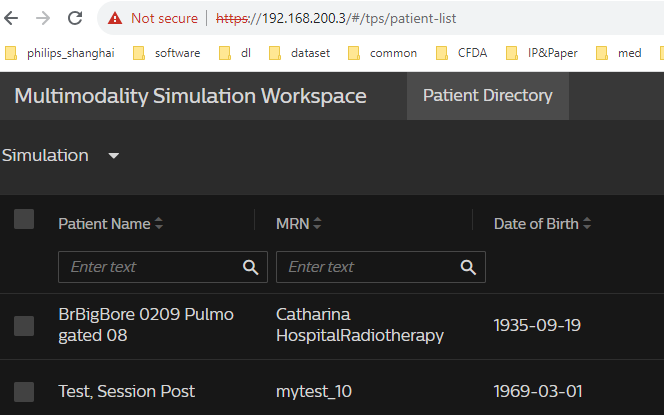
A screenshot of a certificate

Description automatically generated

操作系统里内置的根证书都是通过WebTrust国际安全审计认证。WebTrust认证是电子认证唯一的国际认证标准，各大主流浏览器都要支持。目前通过WebTrust认证的根CA有GeoTrust, DigiCert, Comoda, RapidSsl, SymanTec, Thawte等，都内置在操作系统中。

应用场景：

* **访问自己部署的web server，常常出现Not secure, 如下图所示，如何解决？**



需要web server的证书加入到客户端浏览器

Click `Not secure` -> Certificate Viewer: 161.92.119.174 -> Export…, output `161.92.119.174.crt`

**# install 161.92.119.174.crt**

Double Click 161.92.119.174.crt -> view Details of Certificate: Subject, Subject Alternative Name, …

A screenshot of a certificate

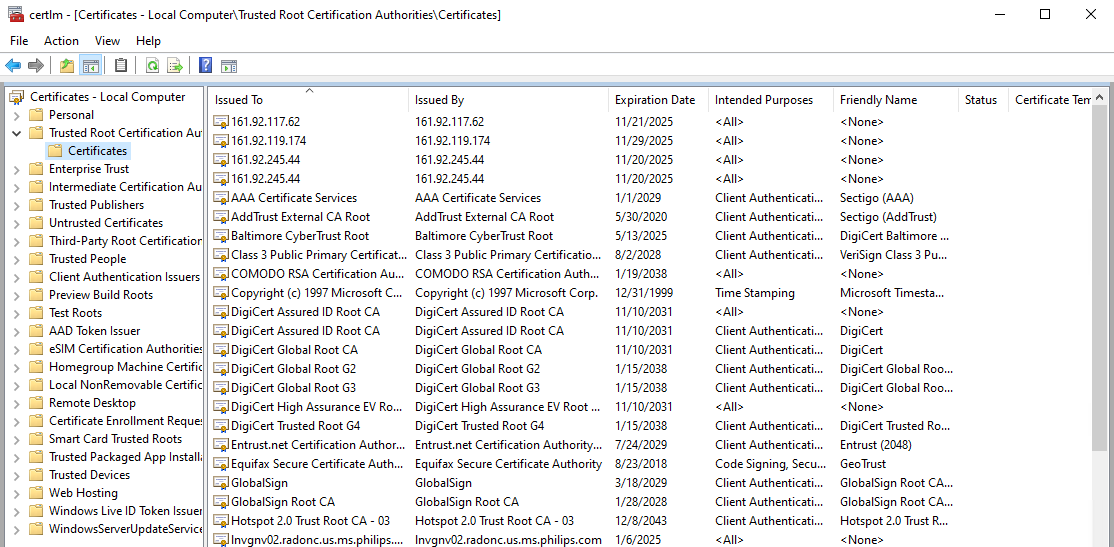
Description automatically generated A screenshot of a computer

Description automatically generated

-> install certificate -> local machine -> Place all certificates in the following storage -> browse -> Trusted CA root certificates

**# Verify the certificate.**

In Window, search: Manage computer certificates



Web browser access to <https://161.92.119.174/>, it will trust this website.  
A screenshot of a computer

Description automatically generated

How to generate 自签名证书?

$ openssl genrsa -out server.key 2048 # generate private key



$ openssl req -new -key server.key -out server.csr # generate CSR

# Generate Certificate：

$ openssl x509 -req

-in server.csr -days 730

-signkey server.key

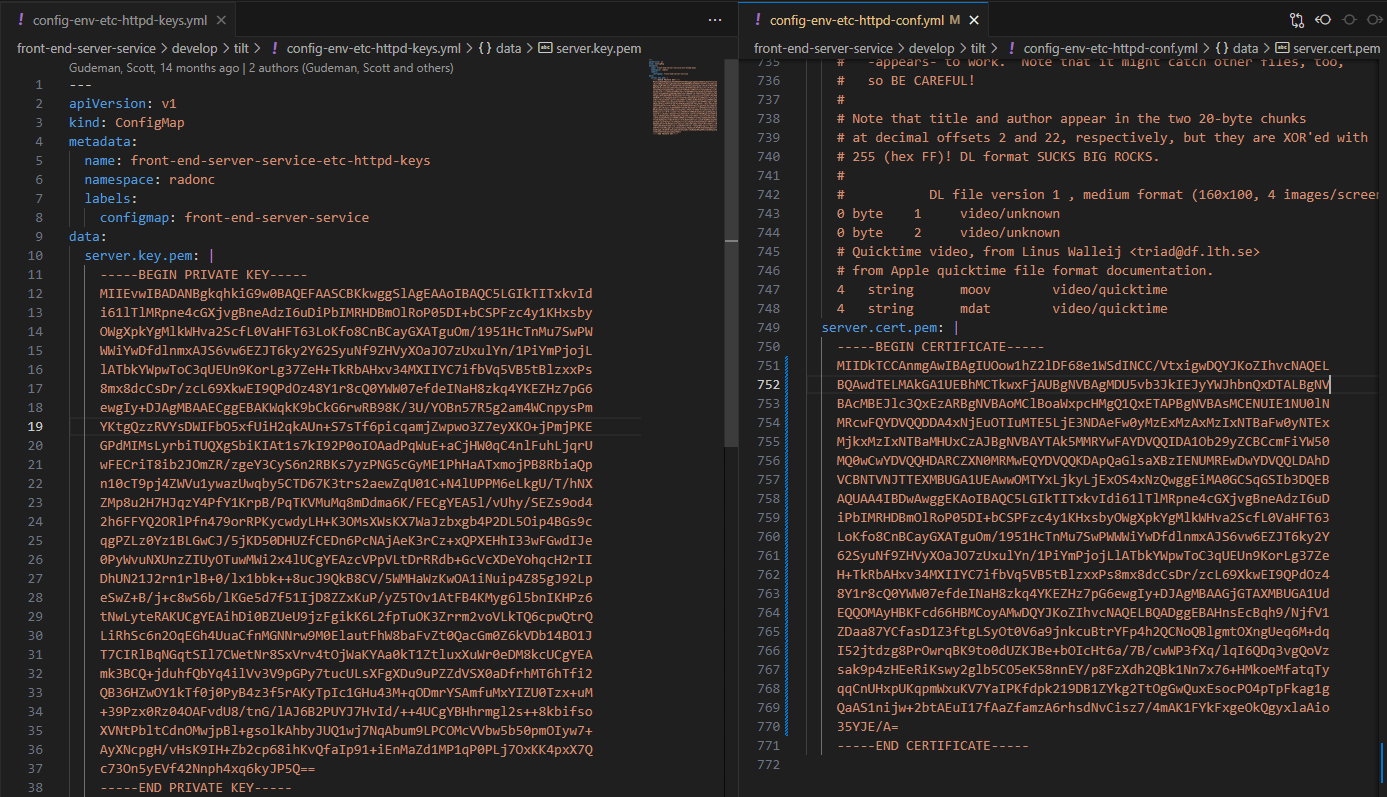
-CAcreateserial

-out server.crt

server.key and server.crt就是web server需要的证书和私钥，如

front-end-server-service/develop/tilt/config-env-etc-httpd-keys.yml

front-end-server-service/develop/tilt/config-env-etc-httpd-conf.yml



确认客户端的客户端证书

https还可以使用客户端证书。以客户端证书进行客户端认证，证明服务器正在通信的对方始终是预料之内的客户端，其作用跟服务器证书如出一辙

例如：银行的网上银行就采用了客户端证书。在登录网银时不仅要求用户确认输入ID和密码，还会要求用户的客户端证书，以确认用户是否从特定的终端访问网银

由自认证机构颁发的证书称为自签名证书

如果使用OpenSSL这套开源程序，每个人都可以构建一套属于自己的认证机构，从而自己给自己颂发服务器证书

独立构建的认证机构叫自认证机构，由自认证机构颁发的“无用”证书也被戏称为自签名证书

确认访问用户身份的认证

核对信息：

密码：只有本人才会知道的字符串信息

动态令牌：仅限本人持有的设备内显示的一次性密码

数字证书：仅限本人（终端）持有的信息

生物认证：指纹和虹膜等本人的生理信息

IC卡等：仅限本人持有的信息

HTTP认证方式：

Basic 基本认证

Digest 摘要认证

SSL 客户端认证

FormBase 基于表单认证

发送请求

GET /private/ HTTP/1.1

Host:…

返回状态码401，需要认证

HTTP/1.1 401 Authorization Required

WWW-Authenticate: Basic realm=”input your ID and password”

浏览器弹出框，要求输入用户ID and password，然后浏览器会组合ID:password字符串，经过Base64编码，把这串字符串写入首部字段Authorization后，发送请求

GET /private/ HTTP/1.1

Host:…

Authorization: Basic dafdfsa….

认证成功返回状态码200,若认证失败返回状态码401

SSL客户端认证采用双因素认证

第一个认证因素是SSL客户端证书用来认证客户端计算机

第二个认证因素是密码用来确定这是用户本人的行为

基本表单认证

通过服务器端的web应用，将客户端发送过来的用户ID和密码与之前登录过的信息做匹配来进行认证

但HTTP是无状态协议，之前已认证成功的用户状态无法通过协议层面保存下来，即无法实现状态管理，因此即使当该用户下一次继续访问，也无法区分他与其他的用户。所以采用Cookie来管理Session, 以弥补http协议中不存在的状态管理功能

步骤：

1. 客户端把用户ID和密码等登录信息放入报文的实体部分，通常是以POST方法把请求发送给服务器。一般使用https通信来进行html表单画面的显示和用户输入数据的发送
2. 服务器验证客户端发送过来的登录信息进行身份认证，然后把用户的认证状态与Session ID绑定后记录在服务器端。向客户端返回响应时，会在首部字段Set-Cookie内写入Session ID。

（可以把Session ID想象成区分用户的等位号，如果Session ID被第三方盗走，对方就可以伪装成你的身份进行恶意操作，所以一般服务器端也需要进行有效期的管理，保证其安全性）

1. 客户端接收到服务器端发来的Session ID后，会将其作为Cookie保存在本地。下次向服务器发送请求时，浏览器会自动发送Cookie,所以Session ID也随之发送到服务器。服务器端可通过验证接收到的Session ID识别用户和其认证状态

基于http的功能追加协议

为了尽可能地实时地显示这些更新的内容，服务器上一有内容更新，就需要直接把那些内容反馈到客户端的界面上。

使用http协议探知服务器上是否有内容更新，就必须频繁地从客户端到服务器端进行确认。如果服务器上没有内容更新，那么就会产生徒劳的通信

**WebSocket**

使用浏览器进行全双工通信

建立在http基础上的协议，发起方仍是客户端，一旦确立websocket通信连接，不论服务器还是客户端，任何一方都可以直接向对方发送报文

主要特点：

推送功能：服务器可直接发送数据，而不必等待客户端的请求

减少通信量：只要建立起websocket连接，就一直保持连接状态

为了实现websocket通信，在http连接建立之后，需要完成一次握手的步骤：

握手请求：

GET /chat HTTP/1.1

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: …

Sec-WebSocket-Protocol: chat, superchat

握手响应：

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WeSocket-Accept: …

Sec-WebSocket-Protocol: chat

成功握手确立WebSocket连接之后，通信时不再使用http的数据帧，而采用WebSocket独立的数据帧

**构建Web内容的技术**

CGI的应用场景

请求 -> Web服务器 -> CGI程序 ->响应

CGI: Common Gateway Interface通用网关接口，指Web服务器在接收到客户端发送过来的请求后转发给程序的一组机制

每次接到请求，程序都要跟着启动一次，因此一旦访问量过大，Web服务器要承担相当大的负载

Servlet的应用场景

多个请求 -> Web服务器，Web容器Servlet, Servlet -> 响应

Servlet与Web服务器相同的进程中，负载小

# TCP/IP

Books:

《图解TCP/IP》

**协议的分层**

四层：应用层HTTP，传输层TCP，网络层IP和数据链路层

七层：应用层，表示层，传话层，传输层，网络层，数据链路层和物理层

OSI参考模型，每个分层都接收由它下一层所提供的特定服务，并且负责为自己的上一层提供特定的服务。上下层之间进行交互时所遵循的约定叫做接口。同一层之间的交互所遵循的约定叫做协议。

分层优点：易于单独实现每个分层的协议，并界定种个分层的具体责任和义务

分层缺点：过分模块化、使处理变得更加沉重以及每个模块都不得不实现相似的处理逻辑等问题

|  |  |  |  |
| --- | --- | --- | --- |
| 分层名称 | 功能 | 概览 |  |
| 应用层 | 针对特定应用的协议 | smtp, telnet, ftp, ssh, http | 应用程序 |
| 表示层 | 各种数据格式 -> 网络标准数据格式 | 接收不同表现形式的信息，如文字流、图像、声音等 |
| 会话层 | 负责建立和断开通信连接 | 如何建立连接，何时断开连接以及保持多久的连接 |
| 传输层 | 负责可靠传输 | TCP/UDP | 操作系统 |
| 网络层 | 寻址与路由选择 | IP |
| 数据链路层 | 互连设备之间传送和识别数据帧 | 数据帧 -> 比特流 | 设备驱动程序与网络接口 |
| 物理层 |  | 比特流 -> 电子信号 |

每个分层上，在处理由上一层传过来的数据时可以附加上当前分层的协议所必须的首部信息。然后接收端对收到的数据进行数据“首部”与“内容”的分离，再转发给上一分层，并最终将发送端的数据恢复为原状

主机A 应用层->表示层->会话层->传输层->网络层->数据链路层->物理层

路由器 网络层<-数据链路层<-物理层

网络层->数据链路层->物理层

主机B 应用层<-表示层<-会话层<-传输层<-网络层<-数据链路层<-物理层

由于数据被转换为通用标准的格式后进行处理，使得异构的机型之间也能保持数据的一致性。这也正是表示层的作用所在。即表示层是进行“统一的网络数据格式”与“某一台计算机或某一款软件特有的数据格式”之间相互转换的分层

TCP/IP分层中的网络层与传输层的功能通常由操作系统提供

连接互联网的所有主机跟路由器必须都实现IP的功能

IP不具有重发机制。即使分组数据包未能到达对端主机也不会重发。因此，属于非可靠性传输协议

## 常见概念: 分组交换,TCP,网关,

**分组交换协议**

分组交换是指将大数据分割为一个个叫做包的较小单位传输的方法

计算机通信也会在每一个分组中附加上源主机地址和目标主机地址送给通信线路。这些发送端地址、接收端地址以及分组序号写入的部分称为‘报文首部’

一个较大的数据被分为多个分组时，为了标明是原始数据中的哪一部分，就有必要将分组的序号写入包中。接收端会根据这个序号，再将每个分组按照序号重新装配为原始数据

通信协议中，通常会规定报文首部应该写入了哪些信息、应该如何处理这些信息。相互通信的每一台计算机则根据协议构造报文首部、读取首部内容等。

**电路交换与分组交换的区别？**

电路交换中，两台计算机是可以独占线路进行数据传输的。其他计算机只能等待这台计算机处理结束以后才有机会使用这条电路收发数据

分组交换可以使多个用户同一时间共享一条通信线路进行通信，从而提高了线路的使用效率，也降低了搭建线路的成本

分组交换，让连接到通信电路的计算机将所要发送的数据分成多个数据包，按照一定的顺序排列之后分别发送。

有了分组交换，数据被细分后，所有的计算机就可以一齐收发数据，这样也就提高了通信线路的利用率。由于在分组的过程中，已经在每个分组的首部写入了发送端和接收端的地址，所以即使同一线路同时为多个用户提供服务，也可以明确区分每个分组数据发往的目的地，以及它是与哪台计算机进行的通信

在分组交换中，由分组交换机（路由器）连接通信线路。发送端计算机将数据分组发送给路由器，路由器收到这些分组数据以后，缓存到自己的缓冲区，然后再转发给目标计算机

在分组交换中，通信线路的速度可能会有所不同。根据网络拥堵的情况，数据达到目标地址的时间有长有短。另外，路由器的缓存饱和或溢出时，甚至可能会发生分组数据丢失、无法发送到对端的情况

单播：早先的固定电话

广播通信：电视播放

多播：电视会议

任播：DNS根域名解析服务器

ICMP: IP数据包在发送途中一旦发生异常导致无法到达对端目标地址时，需要给发送端发送一个发生异常的通知

ARP: 从分组数据包的IP地址中解析出物理地址（MAC地址）的一种协议

计算机内部，通常同一时间运行着多个程序。为此，必须分清是哪些程序与哪些程序在进行通信。识别这些应用程序的是端口号

TCP： 面向有连接的传输层协议。它可以保证两端通信主机之间的通信可达，能够正确处理在传输过程中丢包、传输顺序乱掉等异常情况。为了建立与断开连接，有时它需要至少7次的发包收包，导致网络流量的浪费。此外。为了提高网络的利用率，TCP协议中定义了各种各样复杂的规范，因此不利于视频会议（音频、视频的数据量既定）等场合使用

UDP：面向无连接的传输层协议。UDP不会关注对端是否真的收到了传送过去的数据，如果需要检查对端是否收到分组数据包，或者对端是否连接到网络，则需要在应用程序中实现。UDP常用于分组数据较少或多播、广播通信以及视频通信等多媒体领域

SNMP: simple network management protocol网络管理协议， 使用SNMP管理的主机、网桥、路由器等称为SNMP代理，进行管理的那一端叫SNMP管理器。在SNMP代理端，保存着网络接口的信息、通信数据量、异常数据量以及设备温度等信息。

数据包：帧 -> 数据报 -> 段 -> 消息

帧：表示数据链路层中包的单位。

数据报：是IP和UDP等网络层以上的分层中包的单位

段：表示TCP数据流中的信息

消息：应用协议中数据的单位

在数据包的首部，明确标明了协议应该如何读取数据。反过来说，看到首部，也就能够了解该协议必要的信息以及所要处理内容

TCP首部中包含源端口号，目标端口号（用以识别发送主机跟接收主机上的应用），序号（用以发送的包中哪部分是数据）以及校验和（用以判断数据是否被损坏）

IP首部中包含接收端IP地址以及发送端IP地址，以及TCP/UDP类型信息

以太网首部中包含接收端MAC地址、发送端MAC地址以及标志以太网类型的以太网数据的协议。

经过每个协议分层时，都必须有识别包发送端和接收端的信息。以太网会用MAC地址，IP会用IP地址，而TCP/UDP会用端口号作为识别两端主机的地址，这些地址信息都在每个包经由各个分层时，附加到协议对应的包首部里边

网卡 -> 中继器Repeater -> 网桥Bridge/2层交换机 -> 路由器Router/3层交换机 -> 4-7层交换机 -> 网关(Gateway)

网卡：使计算机连网的设备，也称为Network Interface、网络适配器、LAN卡。一般会被集成到计算机的主板

中继器Repeater:

从物理层上延长网络的设备。只负责将电信号转换为光信号，因此不能传输速度不同的媒介之间转发。

网桥Bridge/2层交换机

识别数据链路层中的数据帧，重构数据帧转发。丢弃错误的数据帧

从数据链路层上延长网络的设备。能够识别数据链路层中的数据帧，并将这些数据帧临时存储于内存，再重新生成信号作为一个全新的帧转发给相连的另一个网段。

网桥会根据地址自学机制来判断是否需要转发数据帧，会记住曾经通过自己转发的所有数据帧的MAC地址，并保存到自己的内存表中。

路由器Router/3层交换机

判断网络路径，转发至目标地址

通过网络层转发分组数据的设备

网桥是根据物理地址MAC地址进行处理，而路由器则是根据IP地址进行处理

路由器可以连接不同的数据链路，如连接两个以太网，或连接一个以太网与一个FDDI

4-7层交换机：处理传输层以上各层网络传输的设备

负责传输层以上的数据转发

应用场景

负载均衡器：为了并发访问，服务器前端访问的入口地址通常只有一个（为了使用者的方便，只会向最终用户开放一个统一的访问URL）。为了通过同一个URL将前端访问分发到后台多个服务器上，可以在这些服务器的前端加上一个负载均衡器。这种负载均衡就是4-7层交换机的一种。还可以通过DNS实现负载均衡，通过对多个IP地址配置同一个名字，每次查询到这个名字的客户得到其中的某一个地址，从而使不同客户访问不同的服务器。该方法也称为循环复用DNS技术

带宽控制，广域网加速器、特殊应用访问加速以及防火墙等

网关(Gateway): 转换协议的设备

负责传输层以上的数据转发及其协议的转换

同4-7层交换机一样，负责处理传输层及以上的数据。负责将从传输层到应用层的数据进行转换和转发的设备。网关不仅转发数据还负责对数据进行转换，它通常会使用一个表示层或应用层网关，在两个不能进行直接通信的协议之间进行翻译，最终实现两者之间的通信

应用场景

为什么连到互联网的电脑与手机之间能够互发电子邮件呢？

互联网与手机之间设置了一道网关。网关负责读取完各种不同的协议后，对它们逐一进行合理的转换，再将相应的数据转发出去。这样一来即使应用的是不同电子邮件的协议，计算机与手机之间也能互相发送邮件

代理服务器：为了控制网络流量以及出于安全的考虑，也是网关的一种，称为应用网关。有了代理服务器，客户端与服务器之间无需在网络层上直接通信，而是从传输层到应用层对数据和访问进行各种控制和处理

防火墙就是一款通过网关通信，针对不同应用提高安全性的产品

带宽：低速数据链路就如同边道较少，高速数据链路就相当于有多个车道。传输速率又称为带宽

吞吐量：主机之间实际的传输速率，吞吐量这个词不仅衡量带宽，同时也衡量主机的CPU处理能力、网络的拥堵程度、报文中数据字段的占有份额等信息

托管主机

数据中心：高并发访问量，托管主机服务，为了减少访问延迟，会集合多个存储于一起，通过连接高速网络，以期提高响应速度。

虚拟化：抽奖网站等对网络资源的需求时刻都在发生变化。虚拟化技术是指当一个网站需要调整运营所使用的资源时，并不增减服务器、存储设备、网络等实际的物理设备，而是利用软件将这些物理设备虚拟化，在有必要增减资源的时候，通过软件按量增减的一种机制。通过此机制实现按需分配、按比例分配，对外提供可靠的服务

利用虚拟化技术，根据使用者的情况动态调整必要资源的机制被人们称作“云”

## 数据发送/接收流程

**数据发送处理流程：**

**应用层**写入的数据

会经过**表示层**格式化编码

再由**会话层**标记发送顺序后才被发送出去

**传输层**为确保所传输的数据到达目标地址，会在通信两端的计算机之间进行确认，如果数据没有到达，它会负责进行重发

**网络层**与数据链路层都是基于目标地址将数据发送给接收端的，但是网络层负责将整个数据发送给最终目标地址，而**数据链路层**则只负责发送一个分段内的数据。

TCP是面向连接型，好比打电话，输入完对方电话号码拨出去之后，只有对端拿起电话才能真正通话，通话结束后将电话机扣上就如同切断电源

UDP是面向无连接型，如同去邮局寄包裹一样。负责处理邮递业务的营业员，不需要确认收件人的详细地址是否真的存在，也不需要确认收件人是否能收到包裹，只要发件人有一个寄件地址就可以办理邮寄包裹的业务

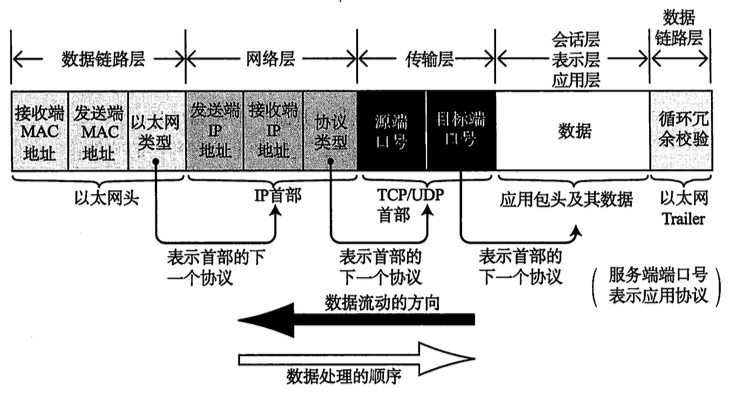
在实际的网络通信当中，每一层的协议所使用的址都不尽相同。TCP/IP通信中使用MAC地址、IP地址、端口号等信息作为地址标识。甚至在应用层中，可以将电子邮件地址作为网络通信的地址

地址特点：唯一性，层次性

MAC地址由设备的制造厂商针对每块网卡进行分别指定。人们可以通过制造商识别号、制造商内部产品编号以及产品通用编号确保MAC地址的唯一性。

IP地址由网络号和主机号两部分组成。通常，同处一个网段的主机也都层于同一个部门或集团组织。网络传输中，每个节点会根据分组数据的地址信息，来判断该报文应该由哪个网卡发送出去。为此，各个地址会参考一个发出接口列表。在这一点上MAC寻址与IP寻址是一样的。只不过MAC寻址中所参考的这张表叫做地址转发表，而IP寻址中所参考的叫做路由控制表

**数据包接收处理流程：**



1. 网络接口（以太网驱动）的处理

主机收到以太网包以后，首先从以太网的包首部找到MAC地址判断是否为发给自己的包，如果是，查找以太网包首部中的类型域从而确定以太网协议所传送过来的数据类型

若数据类型是IP包，就将数据传给处理IP的子程序

若数据类型是ARP协议，就将数据传给ARP处理

1. IP模块的处理

判断IP包首部的IP地址与自己的IP地址匹配，则可接收数据并从中查找上一层协议

如果上一层是TCP，就将IP包首部之后的部分传给TCP处理；如果是UDP，则将IP包首部后面的部分传给UDP处理

若是路由路，接收端地址往往不是自己的地址。此时需要借助路由控制表，查找下一站转发数据

1. TCP模块的处理

首先计算校验和，判断数据是否被破坏。然后检查是否在按照序号接收数据。最后检查端口号，确定具体的应用程序

数据接收完毕后，接收端发送一个“确认回执”给发送端。若回执信息未能达到发送端，那么发送端会认为接收端没有接收到数据而一直反复发送

数据被完整地接收以后，会传给由端口号识别的应用程序

1. 应用程序的处理

接收端应用程序会直接接收发送端的数据。通过解析数据…

**数据链路层：**

指以太网、无线局域网等通信手段

MAC地址用于识别数据链路中互连的节点

MAC地址长48比特，在使用网卡NIC的情况下，MAC地址一般会被烧入到ROM中，因此任何一个网卡的MAC地址都是唯一的，在全世界都不会重复

第１位：单播地址（０）／多播地址（１）

第２位：全局地址（０）／本地地址（１）

第３-24位：表示厂商识别码，由IEEE管理并保证各厂家之间不重复

第25-48位：厂商内部为识别每个网卡而用，由厂商管理并保证产品之间不重复

交换集线器也叫做以太网交换机，以太网交换机就是持有多个端口的网桥。它们根据数据链路层中每个帧的目标MAC地址，决定从哪个网络接口发送数据。这时所参考的、用以记录发送接口的表就叫做转发表

数据链路层的每个通过点在接到包时，会从中将源MAC地址以及曾经接收该地址发送的数据包的接口作为对应关系记录到转发表中

由于MAC地址没有层次性，转发表中的入口个数与整个数据链路中所有网络设备的数量有关

在传输速度不同的情况下，则必须采用那些允许变更速度的设备如网桥、交换集线器或路由器

公共网络

模拟电话线路：利用固定电话线路通信。电话线中的音频带宽用于拨号上网。让计算机与电话线相连需要有一个将数字信号转换为模拟信号的调制解调器，称为猫。猫的传输速率一般只有56kbs左右

ADSL：利用话机到电信局交换机之间这段线路，附加一个叫做分离器的装置，将音频信号（低频信号）和数字信号（高频信号）隔离以免产生噪声干扰

FTTH: fiber to the home，就是一根高速光纤直接连到用户家里或公司的方法，该装置负责在光信号与电子信号之间的转换。使用FTTH可以实现稳定的高速通信

**IP协议：**

网络层的下一层-----数据链路层主要作用是在互连同一种数据链路的节点之间进行包传递。而一旦跨越多种数据链路，就需要借助网络层。网络层可以跨越不同的数据链路，即使是在不同的数据链路上也能实现两端节点之间的数据包传输

主机：配置有IP地址，但不进行路由控制的设备

路由器：即配有IP地址又具有路由控制能力的设备

节点：主机和路由器的统称

数据链路层提供直连两个设备之间的通信功能。与之相比，作为网络层的IP则负责在没有直连的两个网络之间进行通信传输。

IP三大作用模块：IP寻址、路由（最终节点为止的转发）以及IP分包与组包

数据链路的MAC地址：用来标识同一个链路不同计算机的一种识别码

网络层的IP地址：用于在连接到网络中的所有主机中识别出进行通信的目标地址

数据链路实现某一个区间（一跳）内的通信；IP实现直至最终目标地址的通信（点对点）

在一跳的这个区间内，电缆可以通过网桥或交换集线器相连，不会通过路由器或网关相连

当某个IP包到达路由器时，路由器首先查找其目标地址，从而再决定下一步应该将这个包发往哪个路由器

路由控制表

为了将数据包发给目标主机，所有主机都维护着一张路由控制表，该表记录IP数据在下一步应该发给哪个路由器

数据链路的抽象化

不同数据链路有个最大的区别：就是它们各自的最大传输单位（MTU：maximum Transmission Unit）不同。就好像人们在邮寄包裹或行李时有各自的大小限制

MTU的值在以太网中是1500字节，在FDDI中是4352字节，而ATM则为9180字节

IP的上一层可能会要求传送比这些MTU更多字节的数据，为了解决这个问题，IP进行分片处理(IP Fragmentation)，即将较大的IP包分成多个较小的IP包

IP面向无连接，即在发包之前，不需要建立与对端目标地址之间的连接。上层如果遇到需要发送给IP的数据，该数据会立即被压缩成IP包发送出去

面向无连接，即使对端主机关机或不存在，数据包还是会被发送出去，在面向无连接的方式下可能会有很多冗余的通信

IP只负责将数据发给目标主机，TCP负责保证对端主机确实接收到数据

IP地址（IPv4）

32位，每一台主机上的每一块网卡(NIC)都得设置IP地址。通常一块网卡只设置一个IP地址，其实一块网卡也可以配置多个IP地址。此外，一台路由器通常都会配置两个以上的网卡。

IP地址 = 网络地址 + 主机地址

A类地址：0开头，第1-8位是网络地址，后24位主机地址，故一个网段内可容纳主机地址上限为16777214

B类地址：10开头，第1-16位是网络地址，后16位主机地址，故一个网络内可容纳主机上限为65534个

C类地址：110开头，第1-24位是网络地址，后8位主机地址，故一个网络内可容纳主机上限为254个

D类地址：1110开头，第1-32位全是网络地址，无主机地址，主要用于多播

主机地址全为1，表示广播地址

网络标识相同的计算机必须同属于同一个链路

公网IP不足，出现私有IP地址，地址范围如下：

A类 10.0.0.0 ～ 10.255.255.255 (10/8)

B类 172.16.0.0 ～ 172.31.255.255 (172.16/12)

C类 192.168.0.0 ～ 192.168.255.255 (192.168/16)

如何连接私有IP的主机与配有全局IP的互联网主机？

通过NAT技术，互换私有IP与全局IP

现在很多学校、家庭、公司内部采用在每个终端设置私有IP，而在路由器或在必要的服务器上设置全局IP地址，如果配有私有IP的地址主机连网时，则通过NAT进行通信

路由控制

实现IP通信的主机和路由器都必须持有一张这样的表

路由控制表中记录着网络地址与下一步应该发送至路由器的地址。在发送IP包时，首先要确定IP包首部中的目标地址，再从路由控制表中找到与该地址具有相同网络地址的记录，根据该记录将IP包转给相就的下一个路由器。如果路由控制表中存在多条相同网络地址的记录，就选择一个最为吻合的网络地址（即相同位数最多）

默认路由：一般标记为0.0.0.0/0 default

主同路由：IP地址/32，整个IP地址的所有位都将参与路由

环回地址：同一台计算机上的程序之间进行网络通信时所使用的一个默认地址，127.0.0.1, localhost,数据包不会流向网络

$netstat -rn

IP分割处理与再构成处理

数据链路不同，MTU则不同

由于以太网的默认MTU是1500字节，因此4342字节的IP数据报无法在一个帧当中发送完成。这时，路由器将此IP数据报划分成了3个分片进行发送

经过分片之后的IP数据报在被重组的时候，只能由目标主机进行。路由器虽然做分片，但不会进行重组

IP首部中的“片偏移”字段表示分片之后每个分片在用户数据中的相对位置和该分片之后是否还有后续其他分片

路径MTU：指从发送端主机到接收端主机之间不需要分片时最大MTU的大小。即路径中存在的所有数据链路中最小的MTU。很多操作系统都已经实现了路径MTU发现的功能

路径MTU发现的工作原理，见p142

IPv4首部

版本号：4比特

首部长度：4比特

区分服务：8比特，表明服务质量

总长度：16比特，表示IP首部与数据部分合起来的总字节数

标识：16比特，用于分片重组。每发送一个IP包，值逐渐递增

片偏移：13比特，用来标识被分片的每一个分段相对于原始数据的位置

生存时间：8比特，可以中转多少个路由器的意思，每经过一个路由器，TTL减1

协议：8比特，表示IP首部的下一个首部隶属于哪个协议

首部校验和：16比特，只校验数据报的首部，

源地址：32比特

目标地址：32比特

可选项

填充：向字段填充0，调整为32比特的整数倍

数据：将IP上层协议的首部也作为数据进行处理

IP协议相关技术

DNS

在应用中，当用户输入主机名（域名）时，DNS 会自动检索那个注册了主机名和IP地址的数据库，并迅速定位对应的IP地址

域名服务器：管理域名的主机和相应的软件

特点：

各个域的分层都设有一个域名服务器

各层域名服务器都了解该层以下分层中所有域名服务器的IP地址。因此从根域名服务器开始呈树状结构互相连接

由于所有域名服务器都了解根域名服务器的IP地址

DNS解析器：进行DNS查询的主机和软件，用户所使用的工作站或个人电脑都有解析器，一个解析器至少要注册一个以上域名服务器的IP地址。

解析器为了调查IP地址，向域名服务器进行查询处理。接收这个查询请求的域名服务器首先会在自己的数据库进行查找；若没有，则向根域名服务器查找，从而根开始对域名服务器树遍历，直至找到指定的域名服务器，并由空上域名服务器返回想要的数据

解析器和域名服务器将最新了解到的信息暂时保存在缓存里。这样可以减少每次查询时的性能消耗

ARP

解决地址问题的协议，以目标IP地址为线索，用来定位下一个应该接收数据分包的网络设备对应的MAC机制。如果目标主机不在同一个链路上，可以通过ARP查找下一跳路由器的MAC地址

ARP是借助ARP请求与ARP响应两种类型的包确定MAC地址

主机A为了获得主机B的MAC地址，起初要通过广播发送一个ARP请求包。广播包可以被同一个链路上所有的主机或路由器接收，因此ARP的请求包会被同一个链路上所有的主机和路由器进行解析，如果ARP请求包中的目标IP地址与自己的IP地址一致，那么这个节点就将自己的MAC地址塞入ARP响应返回给主机A

ARP缓存表：第一次通过ARP获取到的MAC地址作为IP对MAC的映射关系。一般来说，发送一次IP数据报的主机，继续发送多次IP数据报的可能性会比较高。因此，这种缓存能够有效地减少ARP包的发送。

$arp

RARP （reverse address resolution protocol）

将ARP反过来，从MAC地址定位IP地址的一种协议

需要架设一台RARP服务器，从而在这个服务器上注册设备的MAC地址及其IP地址。然后再将这个设备接入到网络，插电启动设备时，该设备会发送请求信息：xxx MAC地址对应的IP地址是？

$rarp

ICMP

主要功能：确认IP包是否成功送达目标地址，通知在发送过程当中IP包被废弃的具体原因，改善网络设置等

主机A向主机B发送了数据包，由于某种原因，途中的路由器2未能发现主机B的存在，这里路由器2就会向主机A发送一个ICMP包，说明发往主机B的包未能成功。（从路由器2返回的ICMP包会按照往常的路由控制先经过路由器1再转发给主机A。收到该ICMP包的主机A则分解ICMP的首部和数据域以后得知具体发生问题的原因）

$traceroute

$ping

DHCP：　Dynamic Host Configuration Protocol

自动设置IP地址，统一管理IP地址分配

架设DHCP服务器，将DHCP所要分配的IP地址设置到服务器上。此外，还需要将相应的子网掩码、路由控制信息以及DNS服务器的地址等设置到服务器上

NAT: Network Address Translator

用于在本地网络中使用私有地址，在连接互联网时转而使用全局IP地址的技术

只有目标地址、源地址、目标端口、源端口以及协议类型（TCP/UDP）五项内容都一致时才被认为是同一个通信连接

NAT转换表：在TCP的情况下，建立TCP连接首次握手时的SYN包一经发出，就会生成这个表，而后又随着收到关闭连接时发出FIN包的确认应答从表中被删除

**TCP与UDP**

Transmission Control Protocol

User Datagram Protocol

IP首部中有一个协议字段，用来标识网络层IP的上一层所采用的是哪一种传输层协议。根据这个字段的协议号，就可以识别IP传输的数据部分究竟是TCP还是UDP的内容

传输层的TCP和UDP为了识别自己所传输的数据部分究竟应该发给哪个应用程序，设定了端口号

服务端程序在Unix系统当中叫做守护进程。例如HTTP的服务端程序是httpd（HTTP守护进程），而ssh的服务端程序是sshd（SSH守护进程）。在unix中并不需要将这些守护进程逐个启动，而是启动一个可以代表它们接收客户端 请求的inetd(互联网守护进程)服务程序即可。

确认一个请求究竟发给的是哪个服务端 （守护进程），可以通过所收到数据包的目标端口号轻松识别。当收到TCP的建立连接请求时，如果目标端口为22，则转给sshd, 如果是80则转给hhtpd…

TCP是面向连接的可靠的流协议。流就是指不间断的数据结构，你可以把它想象成排水管道中的水流

TCP为提供可靠性传输，实行 “顺序控制”或“重发控制”机制。此外还具备“流控制（流量控制）”、“拥塞控制”、提高网络利用率等众多功能

TCP与UDP区分？

TCP用于在传输层有必要实现可靠传输的情况。由于它是面向有连接并具备顺序控制、重发控制等机制的，所以它可以为应用提供可靠传输

UDP主要用于那些对高速传输和实时性有较高要求的通信或广播通信

TCP充分实现了数据传输时各种控制功能，可以进行丢包时的重发控制，还可以对次序乱掉进行顺序控制。TCP作为一种面向有连接的协议，只有在确认通信对端存在时才会发送数据，从而可以控制通信流量的浪费

套接字Socket: 操作系统提供的api,使用TCP/UDP功能，应用程序利用套接字，可以设置对端的IP地址、端口号，并实现数据的发送与接收

连接：指各种设备、线路、或网络中进行通信的两个应用程序为了相互传递消息而专有的、虚拟的通信线路。TCP负责控制连接的建立、断开、保持等管理工作

UDP经常用于以下几个方面：

* 包总量较少的通信 （DNS, SNMP等）
* 视频、音频等多媒体通信（即时通信）
* 限定于LAN等特定网络中的应用通信
* 广播通信（广播、多播）

TCP特点：

IP数据传输存在问题？数据的破坏、丢包、重复以及分片顺序混乱等问题

解决：TCP通过检验和、序列号、确认应答、重发控制、连接管理以及窗口控制等机制实现可靠性传输

序列号和确认应答

接收端查询接收数据TCP首部中的序列号和数据的长度，将自己下一步应该接收的序号作为确认应答返送回去。就这样，通过序列号和确认应答号，TCP可以实现可靠传输

重发超时

在重发数据之前，等待确认应答到来的那个特定时间间隔。如果超过了这个时间仍未收到确认应答，发送端将进行数据重发。如何确定？

最理想的是，找到一个最小时间，它能保证“确认应答一定能在这个时间内返回”。每次发包时都会计算往返时间及其偏差。在unix and window系统中，超时以0.5秒为单位进行控制。因此重发超时都是0.5秒的整数倍。

连接管理

TCP会在数据通信之前，通过TCP首部发送一个SYN包作为建立连接的请求等待确认应答。可以使用TCP用于控制的字段来管理TCP连接。一个连接的建立需要三次握手，断开需要四个包才能完成

客户端 服务端

SYN（请求建立连接）

 ACK（针对SYN的确认应答）SYN（请求建立连接）

ACK (针对SYN的确认应答) 

连接建立完成，发送数据…..

FIN（请求切断连接）

 ACK (针对FIN的确认应答)

 FIN（请求切断连接）

ACK（针对FIN的确认应答）

TCP以段为单位发送数据

在建立TCP连接的同时，可以确定发送数据包的单位，称为最大消息长度MSS（Maximum Segment Size）。理想情况下，最大消息长度正是IP中不会被分片处理的最大数据长度。MSS是在三次握手的时候，在两端主机之间被计算得出。两端的主机在发出建立连接的请求时，会在TCP首部中写入MSS选项，告诉对方自己的接口能够适应的MSS的大小。然后会在两者之间选择一个较小的值投入使用

TCP首部

源端口号 16位

目标端口号 16位

序列号 32位，指发送数据的位置

确认应答号 32位，发送端收到这个确认应答以后，可以认为在这个序号以前的数据都被正常接收

数据偏移 32位，表示TCP所传输的数据部分应该从TCP包的哪个位开始计算，也可看成TCP首部的长度

保留 4位

控制位 8位，　CWR, ECE, URG, ACK, PSH, RST, SYN, FIN 具体见p223

窗口大小 16位

紧急指针 16位，从数据部分的首位到紧急指针所指示的位置为止为紧急数据

选项 比如时间戳字段选项，用于高速通信中对序列号的管理。若要将几个G的数据高速转发到网络时，32位序列号的值可能会迅速使用完。在传输不稳定的网络环境下，就有可能会在较晚的时间点却收到散布在网络中的一个较早序列号的包。而如果接收端对新老序列号产生混淆就无法实现可靠传输。为了避免这个问题的发生，引入了时间戳这个选项，它可以区分新老序列号

**端口号**

数据链路的地址 MAC地址 用来识别同一链路不同的计算机

IP中的地址 IP地址 用来识别TCP／IP网络中互连的主机和路由器

传输层的地址 端口号 用来识别同一台计算机中进行通信的不同应用程序

TCP/IP(or UDP/IP)通信中通常采用5个信息来识别一个通信。它们分别是源IP地址，目标IP地址，协议号，源端口号，目标端口号。只要其中某一项不同，则被认为是其他通信

**服务端有必要确定监听端口号，客户端（接收服务）没必要确定端口号**。在这种方法下，客户端应用程序可以完全不用自己设置端口号，而全权交给操作系统进行分配。操作系统可以为每个应用程序分配互不冲突的端口号。例如，每需要一个新的端口号时，就在之前分配号码的基础上加1，这样操作系统就可以动态地管理端口号了。

**不同的传输协议可以使用相同的端口号**。因为数据到达IP层后，会先检查IP首部中的协议号，再传给相应协议的模块。那么知名端口号与传输层协议并无关系，只要端口一致都将分配同一种程序进行处理

服务名默认的端口号：

21 ftp

22 ssh

23 telnet

53 DNS

80 http

443 https

110 pop3

# 应用：netsh, tcpdump

* **端口转发(端口映射)**

应用场景：

对外开放vagrant中的服务。

端口转发还可用于将端口从网卡的外部地址，转发到在同一计算机上运行的虚拟机端口

在Linux中，使用iptables可以非常轻松地配置端口重定向。在Windows中可以使用内置网络配置命令行工具NetSH实现端口转发的功能。

netsh: Network Shell 网络配置命令行工具 # windows

# 添加端口转发

$ netsh interface portproxy add v4tov4

listenaddress=<ip> listenport=<port>

connectaddress=<ip> connectport=<port>

v4tov4 ipv4 to ipv4, 还有v4tov6， v6tov4，v6tov6

listenaddress 等待连接的本地IP地址。

listenport 本地侦听TCP端口。

connectaddress 将传入连接重定向到本地或远程IP地址（或DNS名称）。

connectport 一个TCP端口，来自listenport的连接会被转发到该端口。

# 查看已经设置的端口转发

$ netsh interface portproxy show all

# 查看端口是否已经监听

$ netstat -an | find "LISTEN"

# 修改已经设置的端口转发

$ netsh interface portproxy set v4tov4 listenport=<port>

# 删除已经设置的端口转发

$ netsh interface portproxy delete v4tov4 listenport=<port>

# 删除指定的端口转发规则

$ netsh interface portproxy delete v4tov4 listenport=<port> listenaddress=<ip>

# 清除所有当前的端口转发规则

$ netsh interface portproxy reset

Windows10系统开放指定端口

Refer to <https://www.zoneidc.com/news/6883.html>

请确保防火墙（Windows防火墙或通常包含在杀毒软件中的第三方防火墙）允许到新端口的传入连接。

Search: Windows Defender Firewall

->Advanced settings -> Inbound Rules (入站规则)

->New Rule… -> Port, next -> TCP -> Specific local ports: <port1>, <port2> -> Allow the connection: Domain, Private, Public -> Name: <name>

注：

入站规则：别人电脑访问自己电脑;

出站规则：自己电脑访问别人电脑。

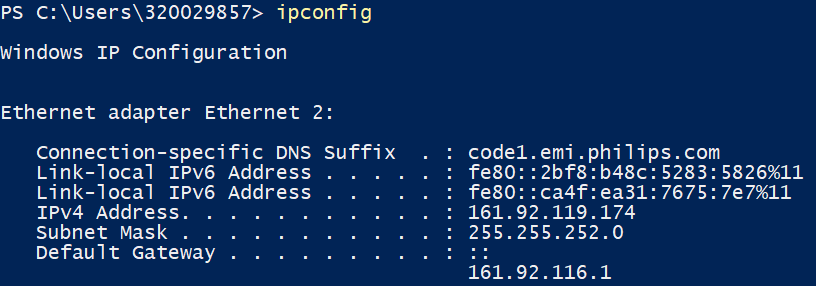
Example:

通过CT console，debug mmsim

* configure port mapping in windows machine.

in windows machine, open Windows PowerShell with administrator

$ ifconfig // get host IP



# Add port mapping

$ netsh interface portproxy add v4tov4

listenaddress=161.92.119.174 listenport=443

connectaddress=192.168.200.3 connectport=443

$ netsh interface portproxy add v4tov4

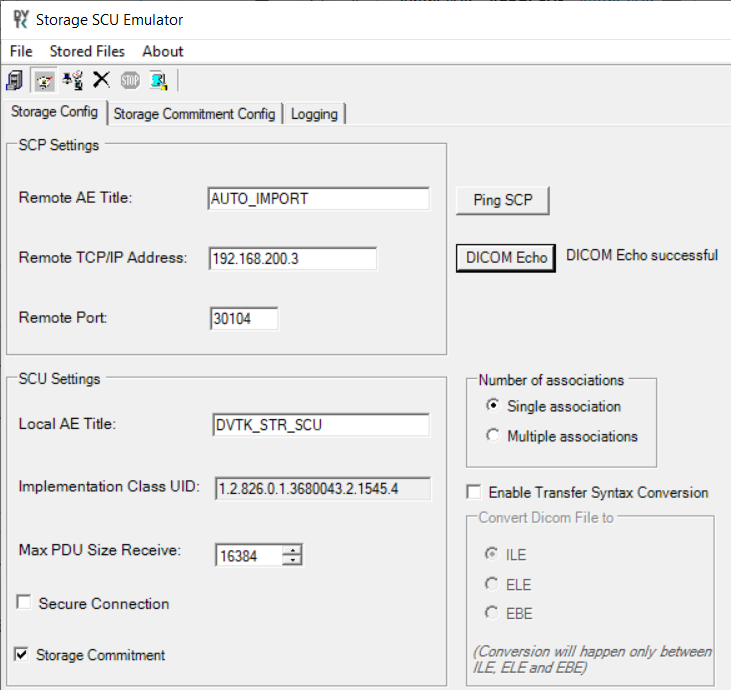
listenaddress=161.92.119.174 listenport=30104

connectaddress=192.168.200.3 connectport=30104

# Verify port 443 via browser

access Patient Dashboard: <https://161.92.119.174/> instead of <https://192.168.200.3/>

# Verify port 30104 via storage SCU Emulator:

 A screenshot of a computer

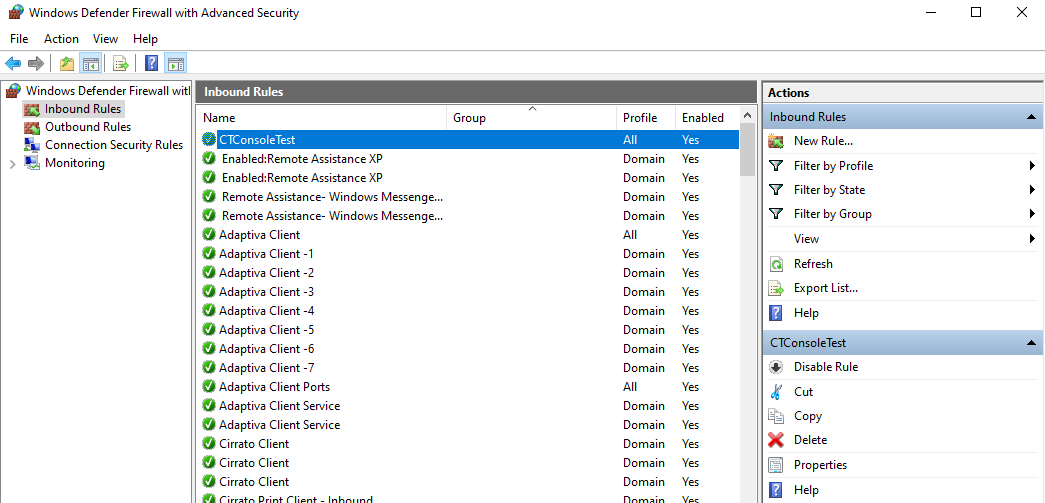
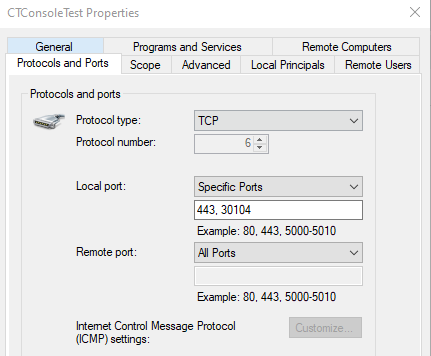
Description automatically generated

Windows10系统开放指定端口，从而允许内网其他电脑可以访问该电脑服务

Search: Windows Defender Firewall

->Advanced settings -> Inbound Rules (入站规则)

->New Rule… -> Port, next -> TCP -> Specific local ports: 443, 30104 -> Allow the connection: Domain, Private, Public -> Name: CTConsoleTest

* **抓取流动在网卡上的数据包：tcpdump， wireshark**

抓包原理：通过注册一种虚拟的底层网络协议来完成对网络报文的处理。

当网卡接收到一个网络报文之后，它会遍历系统中所有已经注册的网络协议，例如以太网协议处理模块来尝试进行报文的解析处理，这一点和文件系统的挂载相似，就是让系统中所有已注册的文件系统来进行尝试挂载，如果哪一个认为自己可以处理，那么就完成挂载。

当抓包模块把自己伪装成一个网格协议的时候，系统在收到报文的时候就会给这个伪协议一次机会，让它来对网卡收到的报文进行一次处理，此时该模块就会趁机对报文进行窥探。

<https://www.tcpdump.org/>

# 在149.59.136.46打开抓包工具，窥探

$ tcpdump -i eth0 // 抓取经过网卡eth0的网络数据

//抓取经过eth0,目的或源地址是130.147.128.227的网络数据

$ tcpdump -i eth0 host 130.147.128.227

// 抓取主机130.147.128.227发送的所有数据

$ tcpdump -i eth0 src host 130.147.128.227

//抓取主机130.147.128.227和主机10.37.63.61 or 10.37.63.95的通信

$ tcpdump host 130.147.128.227 and \(10.37.63.61 or 10.37.63.95 \)

// 抓取主机10.37.63.61接收的所有数据

$ tcpdump -i eth0 dst host 10.37.63.61

// 抓取主机10.37.63.61所有在TCP 80端口的数据包

$ tcpdump -i eth0 host 10.37.63.61 and tcp port 80

e.g.,

截获主机[www.baidu.com](http://www.baidu.com)发送与接收所有的数据包

$ tcpdump -i eth0 host [www.baidu.com](http://www.baidu.com)

A close-up of a computer code

Description automatically generated

$ wget [www.baidu.com](http://www.baidu.com)

A screenshot of a computer code

Description automatically generated

测试130.147.128.227是否给149.59.136.46发送API请求？

$ tcpdump -i eth0 host 130.147.128.227 # 在46号机抓包来自227号机

在130.147.128.227号机，CT console -> launch mmsim

查看46号机stdout，是否有接收到信号