第二部分：WebSocket 协议

扫码试看/订阅极客时间

《Web协议详解与抓包实战》视频课程

Wireshark 过滤器

- **捕获过滤器**
  - 用于减少抓取的报文体积
  - 使用 BPF 语法，功能相对有限

- **显示过滤器**
  - 对已经抓取到的报文过滤显示
  - 功能强大

BPF 过滤器：Wireshark 捕获过滤器

- Berkeley Packet Filter，在设备驱动级别提供抓包过滤接口，多数抓包工具都支持此语法
- expression 表达式：由多个原语组成

Expression 表达式

- primitives 原语：由名称或数字，以及描述它的多个限定词组成
  - qualifiers 限定词
    - Type：设置数字或者名称所指示类型，例如 host www.baidu.com
    - Dir：设置网络出入方向，例如 dst port 80
    - Proto：指定协议类型，例如 udp
    - 其他
- 原语运算符
  - 与：&& 或者 and
  - 或：|| 或者 or
  - 非：! 或者 not
- 例如：src or dst portrange 6000-8000 && tcp or ip6

限定词

Type：设置数字或者名称所指示类型

- host、port
- net ，设定子网，net 192.168.0.0 mask 255.255.255.0 等价于 net 192.168.0.0/24
- portrange，设置端口范围，例如 portrange 6000-8000

限定词

Dir：设置网络出入方向

- src、dst、src or dst、src and dst
- ra、ta、addr1、addr2、addr3、addr4（仅对 IEEE 802.11 Wireless LAN 有效）

限定词

Proto：指定协议类型

- ether、fddi、tr、 wlan、 ip、 ip6、 arp、 rarp、 decnet、 tcp、udp、icmp、igmp、icmp、igrp、pim、ah、esp、vrrp

限定词

其他

- gateway：指明网关 IP 地址，等价于 ether host *ehost* and not host *host*
- broadcast：广播报文，例如 ether broadcast 或者 ip broadcast
- multicast：多播报文，例如 ip multicast 或者 ip6 multicast
- less, greater：小于或者大于

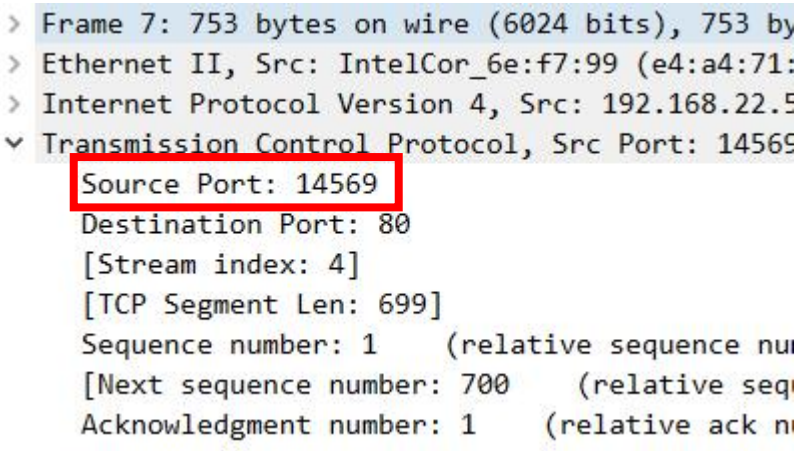
基于协议域过滤

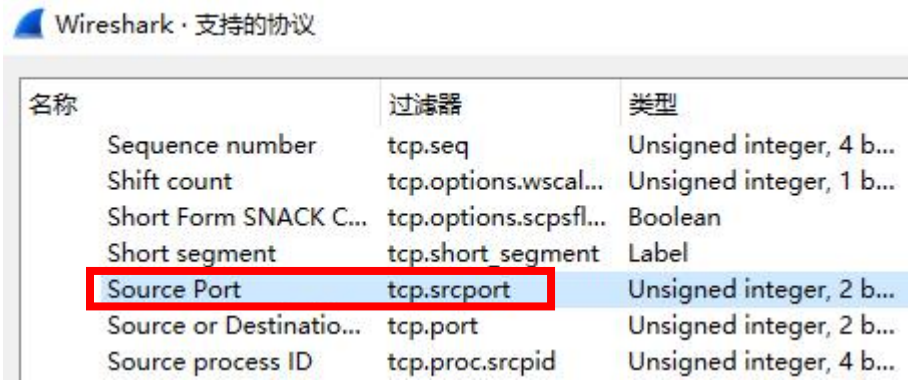
- 捕获所有 TCP 中的 RST 报文
  - tcp[13]&4==4
- 抓取 HTTP GET 报文
  - port 80 and tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420
    - 注意：47455420 是 ASCII 码的 16 进制，表示"GET "
    - TCP 报头可能不只 20 字节，data offset 提示了承载数据的偏移，但它以 4 字节为单位

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

显示过滤器的过滤属性

- 任何在报文细节面板中解析出的字段名，都可以作为过滤属性
  - 在视图->内部->支持的协议面板里，可以看到各字段名对应的属性名
    - 例如，在报文细节面板中 TCP 协议头中的 Source Port，对应着过滤属性为 tcp.srcport





过滤值比较符号

英文	符号	描述及示例
eq	==	等于. ip.src==10.0.0.5
ne	!=	不等于. ip.src!=10.0.0.5
gt	>	大于. frame.len > 10
lt	<	小于. frame.len < 128
ge	>=	大于等于. frame.len ge 0x100
le	<=	小于等于. frame.len ⇐ 0x20
contains		包含. sip.To contains "a1762"
matches	~	正则匹配.host matches "acme\.(org|com|net)"
bitwise_and	&	位与操作. tcp.flags & 0x02

过滤值类型

- **Unsigned integer**：无符号整型，例如 ip.len le 1500
- **Signed integer**：有符号整型
- **Boolean**：布尔值，例如 tcp.flags.syn
- **Ethernet address**：以:、-或者.分隔的 6 字节地址，例如 eth.dst == ff:ff:ff:ff:ff:ff
- **IPv4 address**：例如 ip.addr == 192.168.0.1
- **IPv6 address**：例如 ipv6.addr == ::1
- **Text string**：例如 http.request.uri == "https://www.wireshark.org/"

多个表达式间的组合

英文	符号	意义及示例
and	&&	AND 逻辑与. ip.src==10.0.0.5 and tcp.flags.fin
or	||	OR 逻辑或. ip.scr==10.0.0.5 or ip.src==192.1.1.1
xor	^^	XOR 逻辑异或. tr.dst[0:3] == 0.6.29 xor tr.src[0:3] == 0.6.29
not	!	NOT 逻辑非. not llc
[…]		见 Slice 切片操作符.
in		见集合操作符.

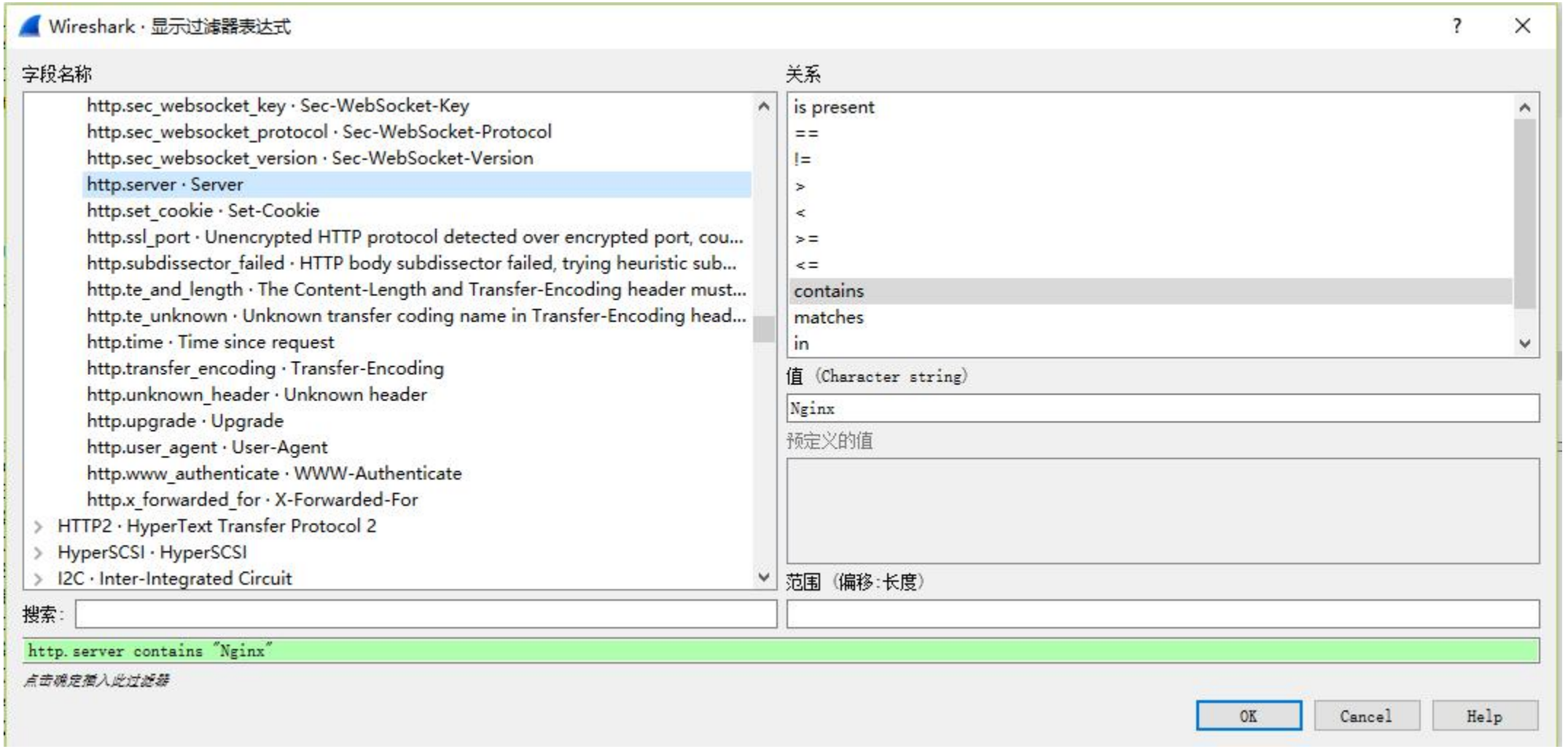
其他常用操作符

- 大括号{}集合操作符
  - 例如 tcp.port in {443 4430..4434}，实际等价于 tcp.port == 443 || (tcp.port >= 4430 && tcp.port <= 4434)
- 中括号[]Slice 切片操作符
  - [n:m]表示 n 是起始偏移量，m 是切片长度
    - eth.src[0:3] == 00:00:83
  - [n-m]表示 n 是起始偏移量，m 是截止偏移量
    - eth.src[1-2] == 00:83
  - [:m]表示从开始处至 m 截止偏移量
    - eth.src[:4] == 00:00:83:00
  - [m:]表示 m 是起始偏移量，至字段结尾
    - eth.src[4:] == 20:20
  - [m]表示取偏移量 m 处的字节
    - eth.src[2] == 83
  - [,]使用逗号分隔时，允许以上方式同时出现
    - eth.src[0:3,1-2,:4,4:,2] ==00:00:83:00:83:00:00:83:00:20:20:83

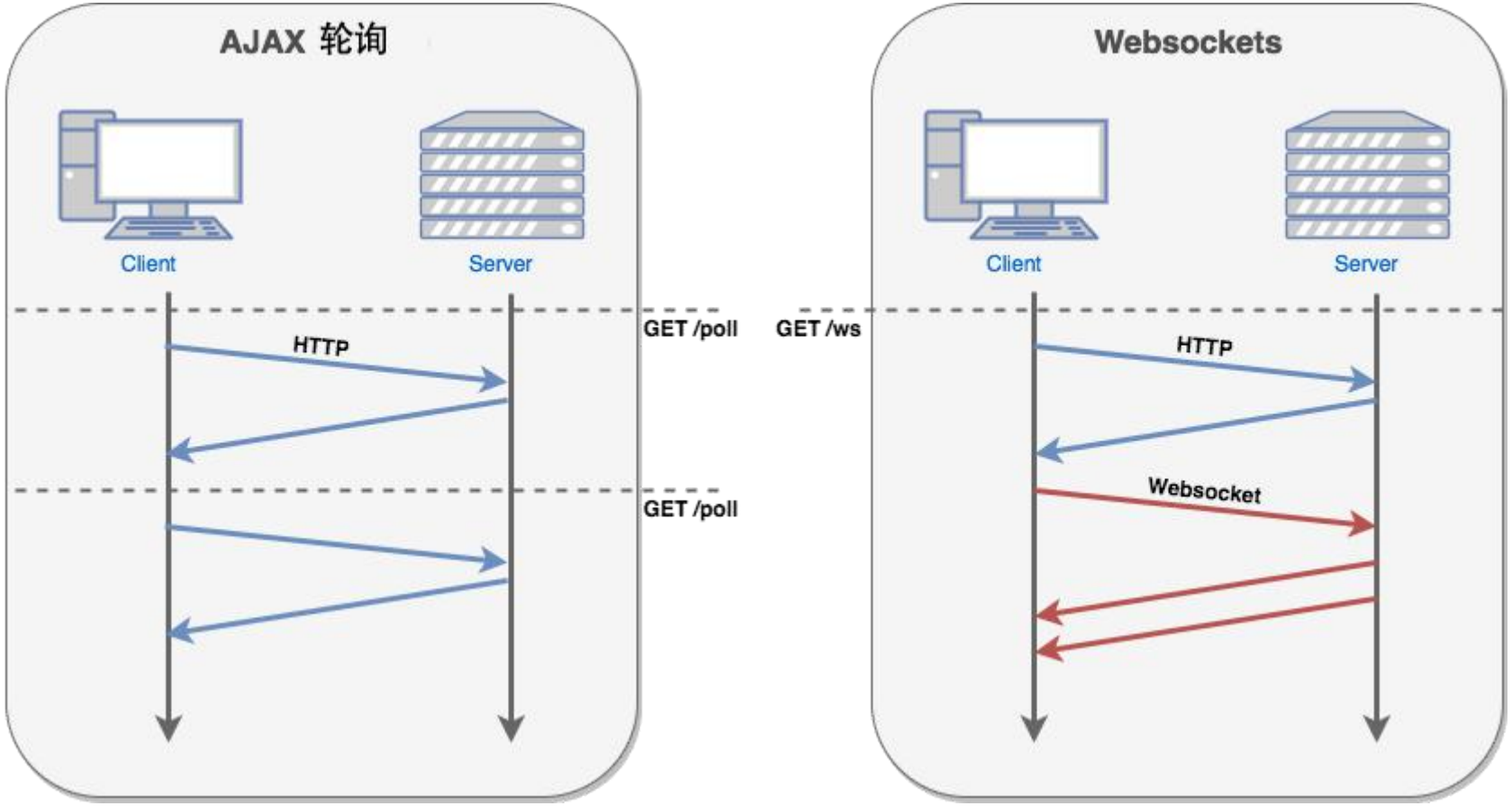
可用函数

upper	Converts a string field to uppercase.
lower	Converts a string field to lowercase.
len	Returns the byte length of a string or bytes field.
count	Returns the number of field occurrences in a frame.
string	Converts a non-string field to a string.

显示过滤器的可视化对话框



如何及时获得更新？从轮询到通知



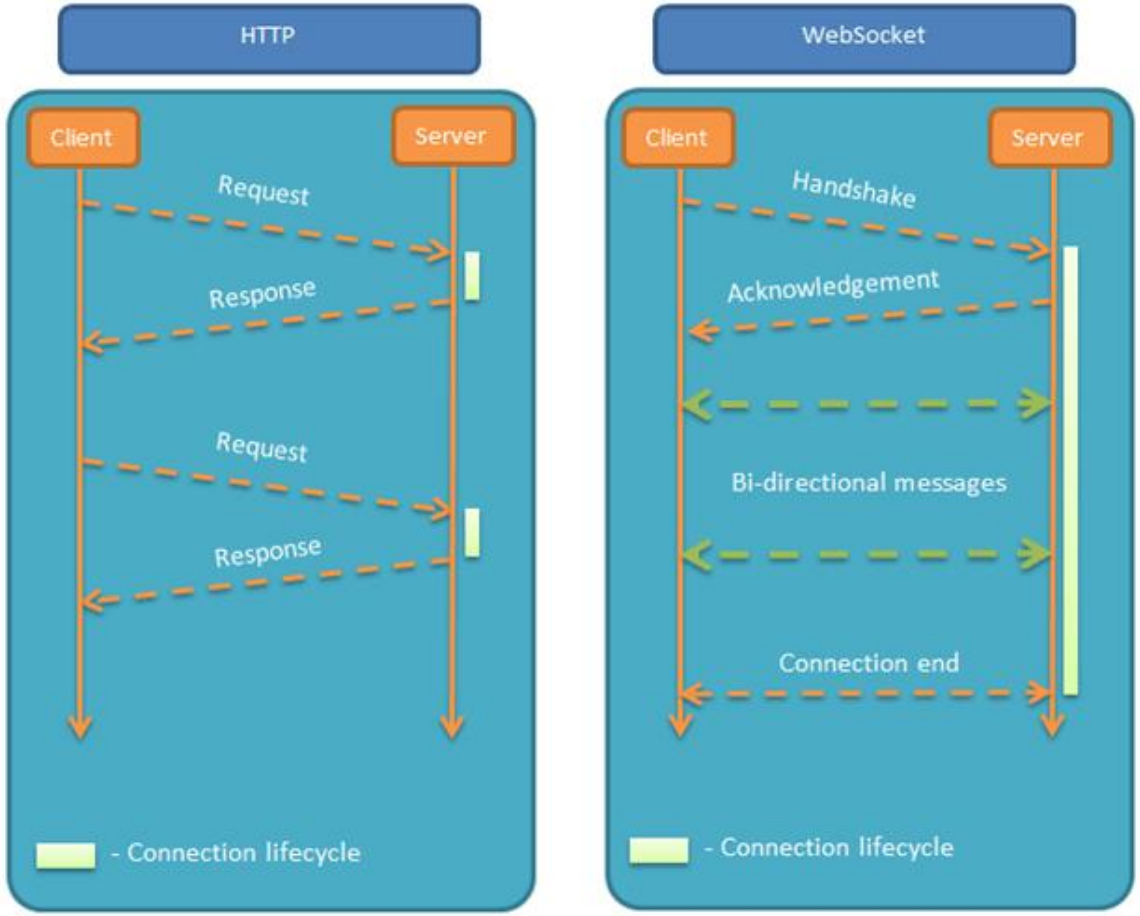
Chrome 请求列表：分析 WebSocket

极客时间

- 过滤器
  - 按类型：WS
  - 属性过滤：is: running
- 表格列
  - Data： 消息负载。 如果消息为纯文本，则在此处显示。 对于二进制操作码，此列将显示操作码的名称和代码。 支持以下操作码：Continuation Frame、Binary Frame、Connection Close Frame、Ping Frame 和 Pong Frame。
  - Length： 消息负载的长度（以字节为单位）。
  - Time： 收到或发送消息的时间。
- 消息颜色
  - 发送至服务器的文本消息为浅绿色。
  - 接收到的文本消息为白色。
  - WebSocket 操作码为浅黄色。
  - 错误为浅红色。

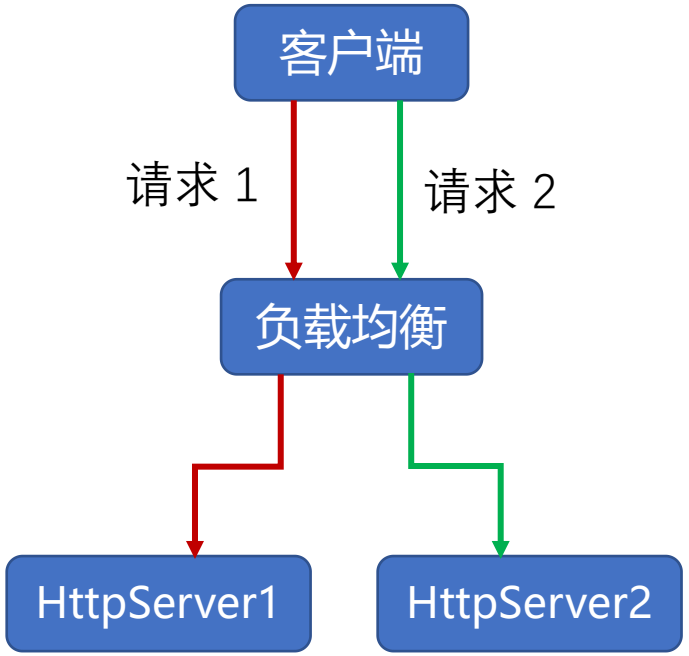
支持双向通讯的 WebSocket

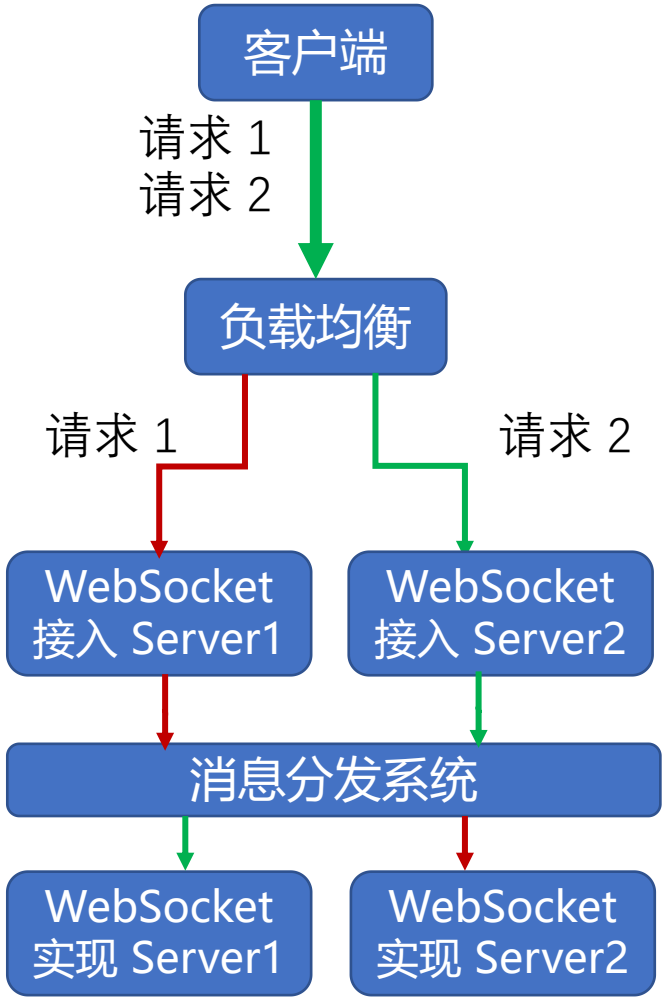
- rfc6455（2011.12）
- 双向通讯的优劣?
- 如何管理会话?
- 如何维持长连接?
- 兼容 HTTP 协议
  - 端口复用
- 支持扩展
  - 如 permessage-deflate 扩展



WebSocket 的成本

- 实时性与可伸缩性
  - 牺牲了简单性
- 网络效率与无状态：请求 2 基于请求 1
  - 牺牲了简单性与可见性





长连接的心跳保持

- HTTP 长连接只能基于简单的超时（常见为 65 秒）
- WebSocket 连接基于 ping/pong 心跳机制维持

兼容 HTTP 协议

- 默认使用 80 或者 443 端口
- 协议升级
- 代理服务器可以简单支持

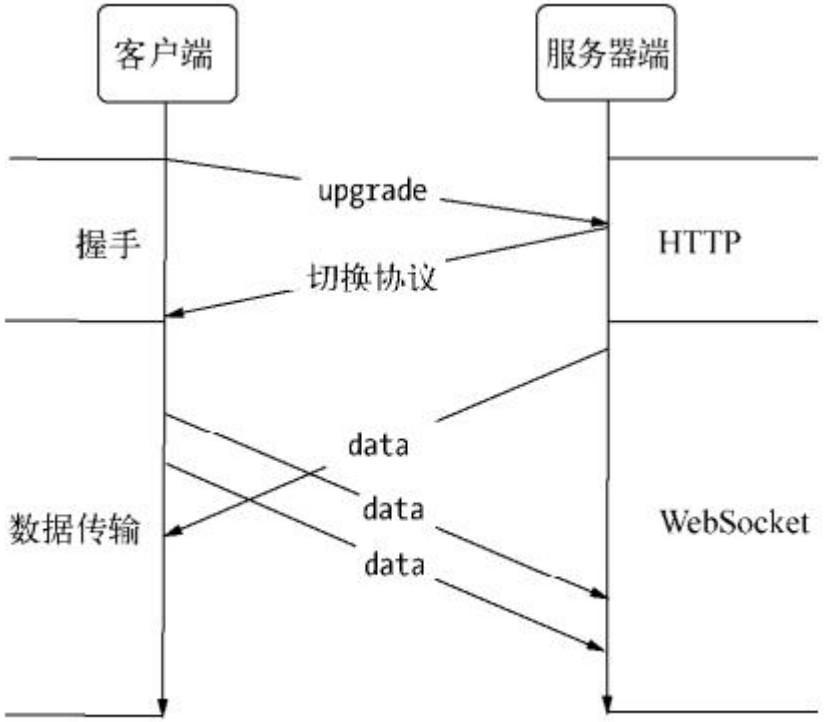


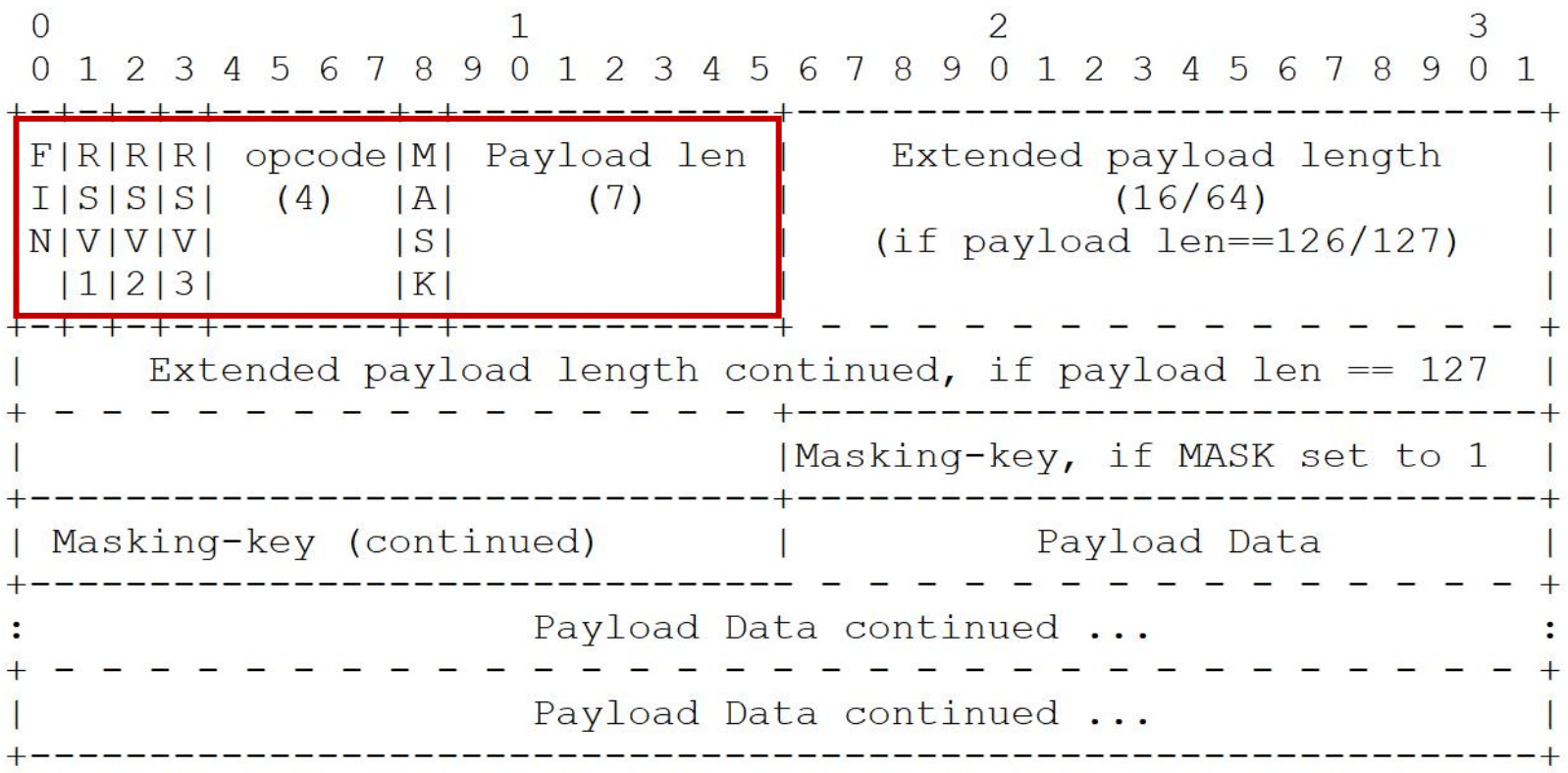
图7-6　协议升级过程示意图

设计哲学：在 Web 约束下暴露 TCP 给上层

- **元数据去哪了？**
  - 对比：HTTP 协议头部会存放元数据
  - 由 WebSocket 上传输的应用层存放元数据
- **基于帧：不是基于流（HTTP、TCP）**
  - 每一帧要么承载字符数据，要么承载二进制数据
- **基于浏览器的同源策略模型（非浏览器无效）**
  - 可以使用 Access-Control-Allow-Origin 等头部
- **基于 URI、子协议支持同主机同端口上的多个服务**

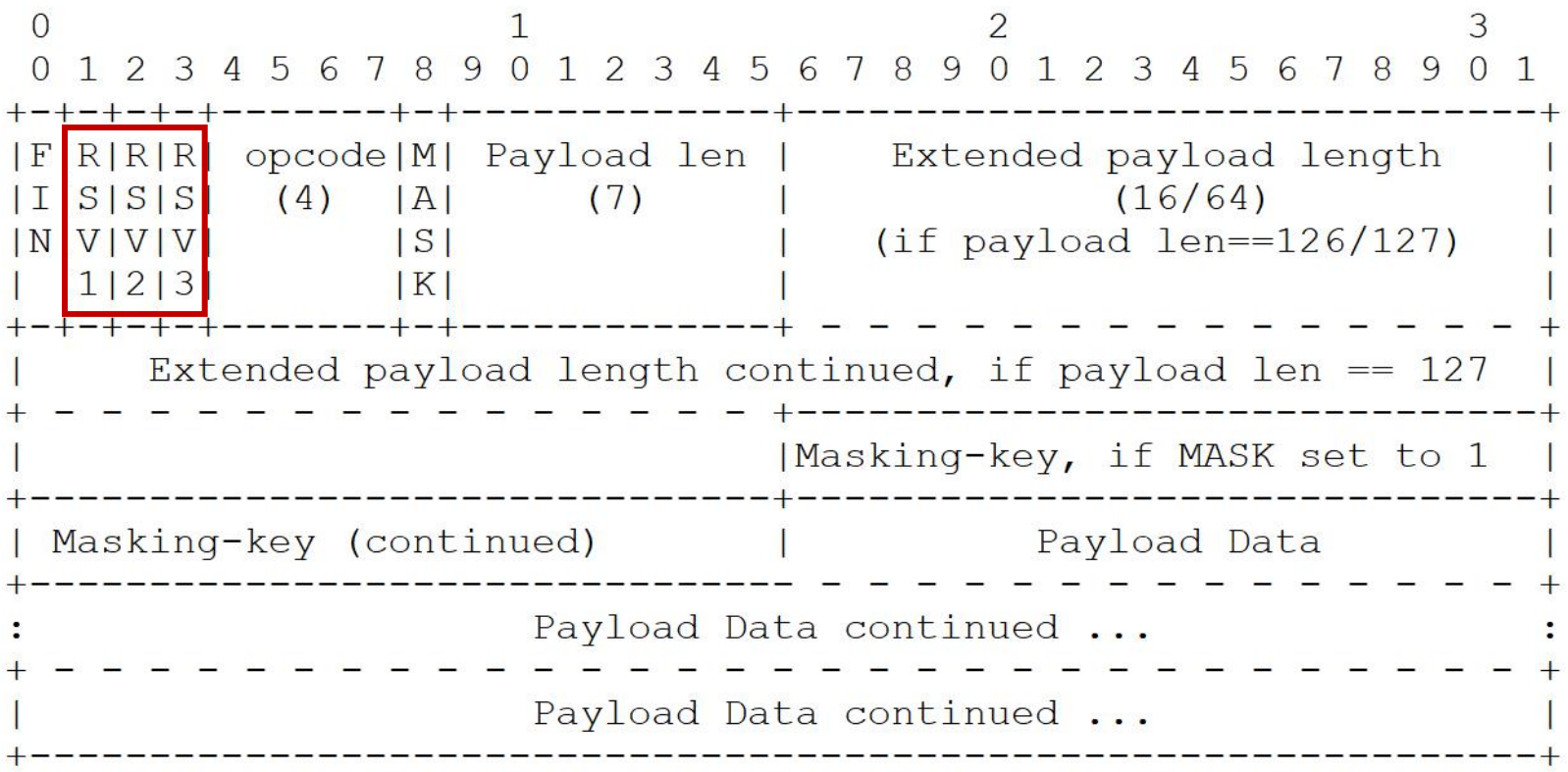
帧格式示意图

- 红色是 2 字节必然存在的帧首部



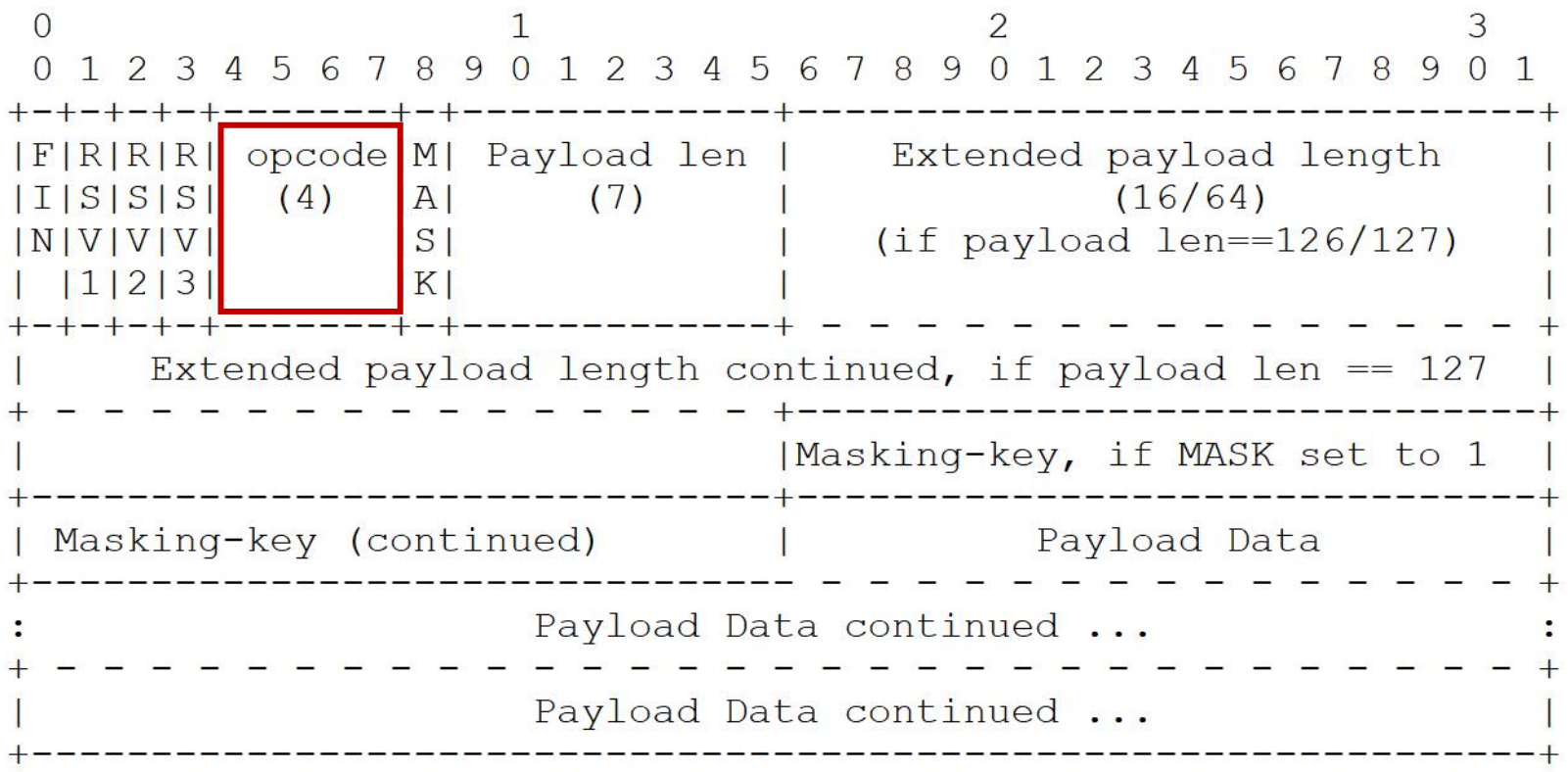
数据帧格式：RSV 保留值

RSV1/RSV2/RSV3：默认为 0，仅当使用 extension 扩展时，由扩展决定其值



数据帧格式：帧类型

- 持续帧
  - 0：继续前一帧
- 非控制帧
  - 1：文本帧（UTF8）
  - 2：二进制帧
  - 3-7：为非控制帧保留
- 控制帧
  - 8：关闭帧
  - 9：心跳帧 ping
  - A：心跳帧 pong
  - B-F：为控制帧保留



ABNF 描述的帧格式

- ws-frame = frame-fin ; 1 bit in length

  frame-rsv1 ; 1 bit in length

  frame-rsv2 ; 1 bit in length

  frame-rsv3 ; 1 bit in length

  frame-opcode ; 4 bits in length

  frame-masked ; 1 bit in length

  frame-payload-length ; 3 种长度

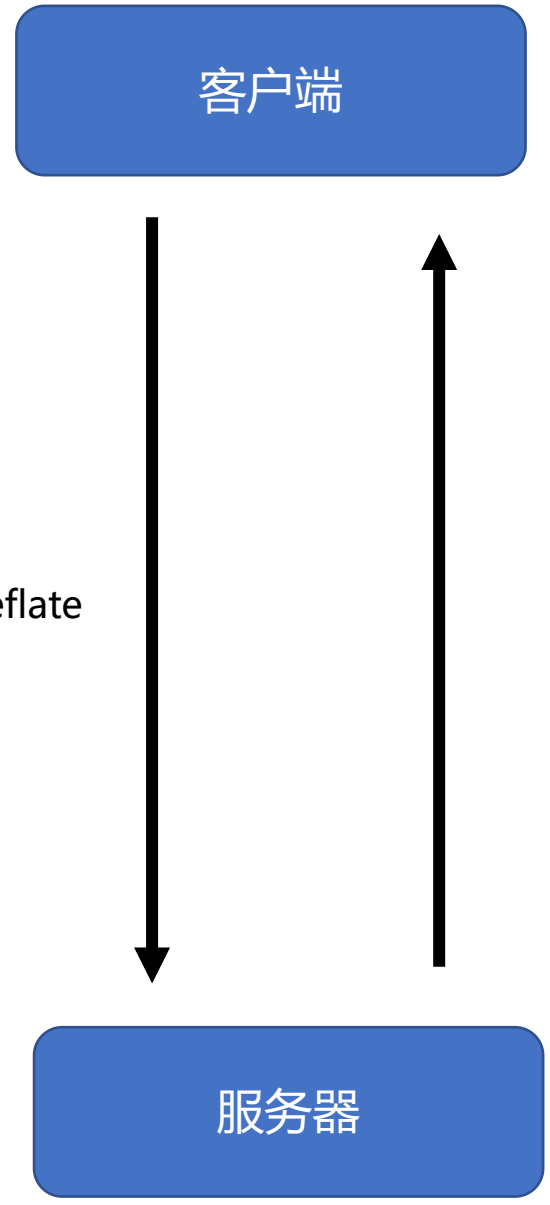
  [frame-masking-key] ; 32 bits in length

  frame-payload-data ; n*8 bits in ; length, where ; n >= 0

URI 格式

- ws-URI = "ws:" "//" host [":" port] path ["?" query]
  - 默认 port 端口 80
- wss-URI = "wss:" "//" host [":" port] path ["?" query]
  - 默认 port 端口 443
- 客户端提供信息
  - host 与 port：主机名与端口
  - shema：是否基于 SSL
  - 访问资源：URI
  - 握手随机数：Sec-WebSocket-Key
  - 选择子协议： Sec-WebSocket-Protocol
  - 扩展协议： Sec-WebSocket-Extensions
  - CORS 跨域：Origin

建立握手



GET /?encoding=text HTTP/1.1
Host: websocket.taohui.tech
Accept-Encoding: gzip, deflate
Sec-WebSocket-Version: 13
Origin: http://www.websocket.org
Sec-WebSocket-Extensions: permessage-deflate
Sec-WebSocket-Key:
c3SkgVxVCDhVCp69PJFf3A==
Connection: keep-alive, Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket

HTTP/1.1 101 Web Socket Protocol Handshake
Server: openresty/1.13.6.2
Date: Mon, 10 Dec 2018 08:14:29 GMT
Connection: upgrade
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: content-type
Access-Control-Allow-Headers: authorization
Access-Control-Allow-Headers: x-websocket-extensions
Access-Control-Allow-Headers: x-websocket-version
Access-Control-Allow-Headers: x-websocket-protocol
Access-Control-Allow-Origin: http://www.websocket.org
Sec-WebSocket-Accept:
yA9O5xGLp8SbwCV//OepMPw7pEI=
Upgrade: websocket

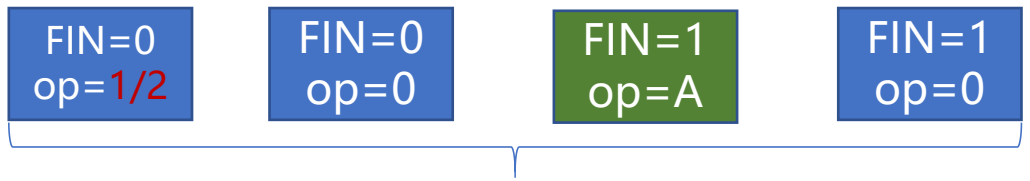
如何证明握手被服务器接受？预防意外

- 请求中的 Sec-WebSocket-Key 随机数
  - 例如 Sec-WebSocket-Key: A1EEou7Nnq6+BBZoAZqWlg==
- 响应中的 Sec-WebSocket-Accept 证明值
  - GUID（RFC4122）：258EAFA5-E914-47DA-95CA-C5AB0DC85B11
  - 值构造规则：BASE64(SHA1(Sec-WebSocket-KeyGUID))
    - 拼接值：A1EEou7Nnq6+BBZoAZqWlg==258EAFA5-E914-47DA-95CA-C5AB0DC85B11
    - SHA1 值：713f15ece2218612fcadb1598281a35380d1790f
    - BASE 64 值：cT8V7OIhhhL8rbFZgoGjU4DReQ8=
    - 最终头部：Sec-WebSocket-Accept: cT8V7OIhhhL8rbFZgoGjU4DReQ8=

消息与数据帧

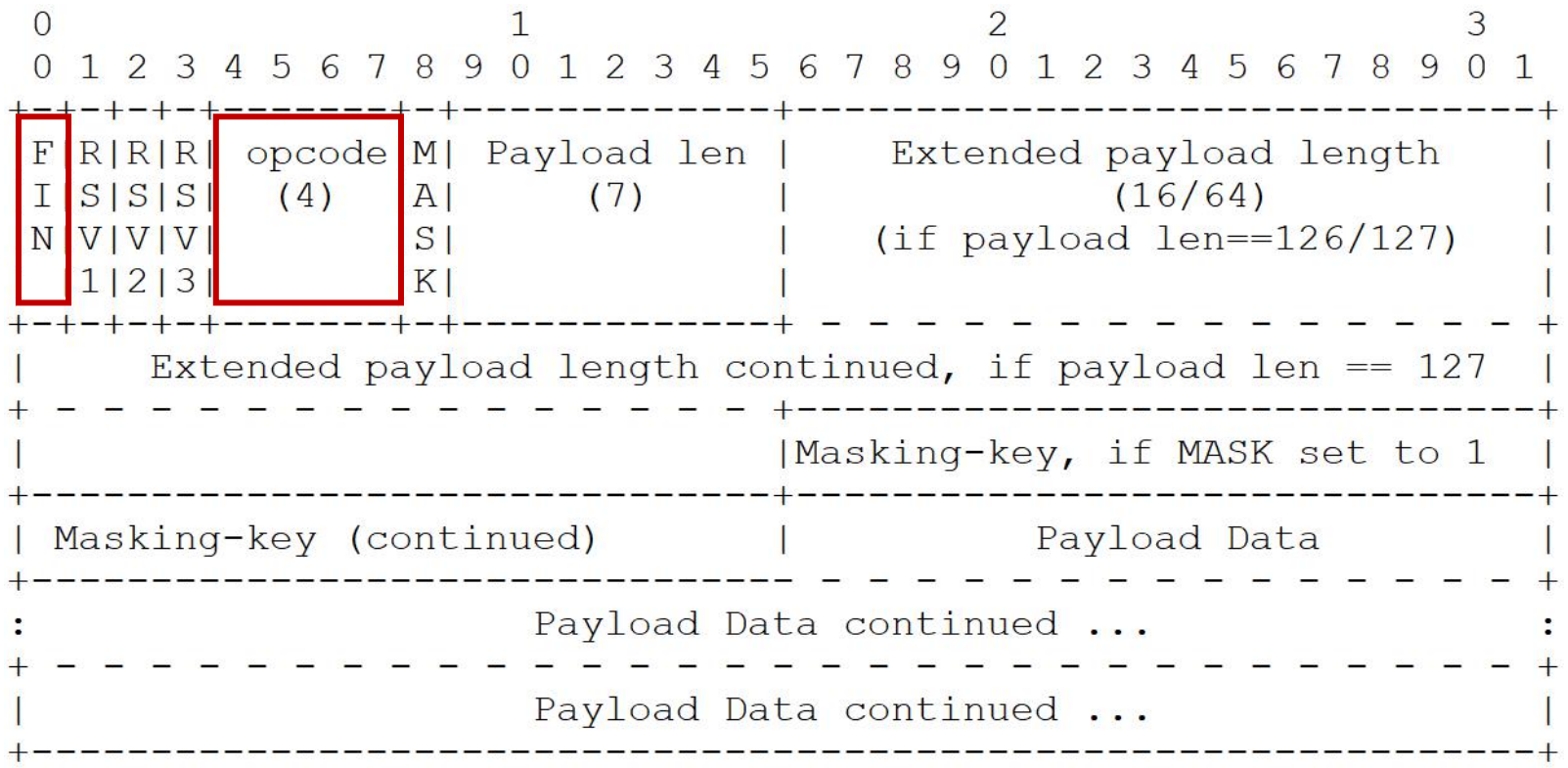
- Message 消息
  - 1 条消息由 1 个或者多个帧组成，这些数据帧属于同一类型
  - 代理服务器可能合并、拆分消息的数据帧
- Frame 数据帧
  - 持续帧
  - 文本帧、二进制帧

非控制帧的消息分片：有序



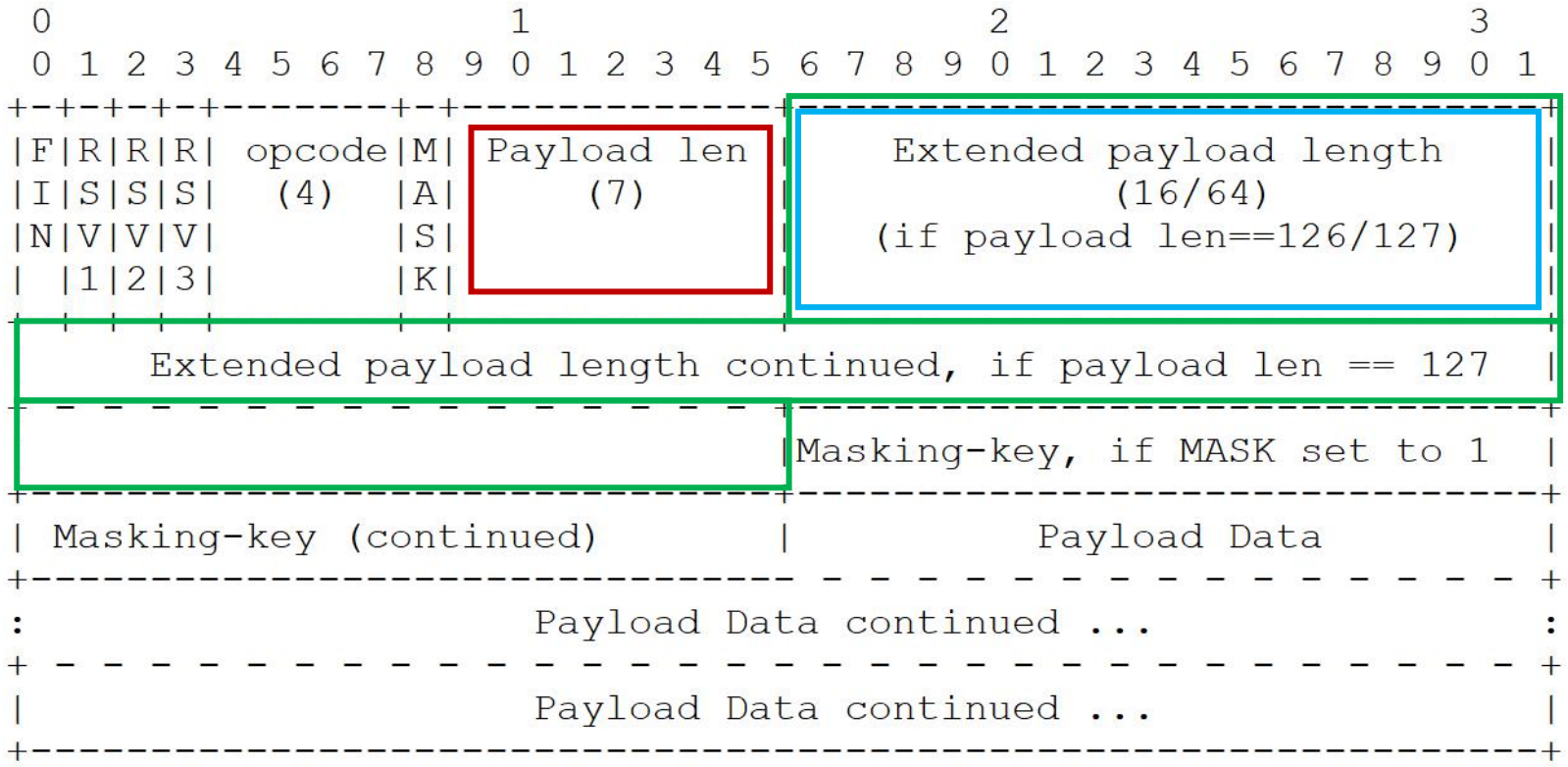
1 条消息由 1 个或者多个数据帧构成

1 条消息由 1 个数据帧构成



数据帧格式：消息内容的长度

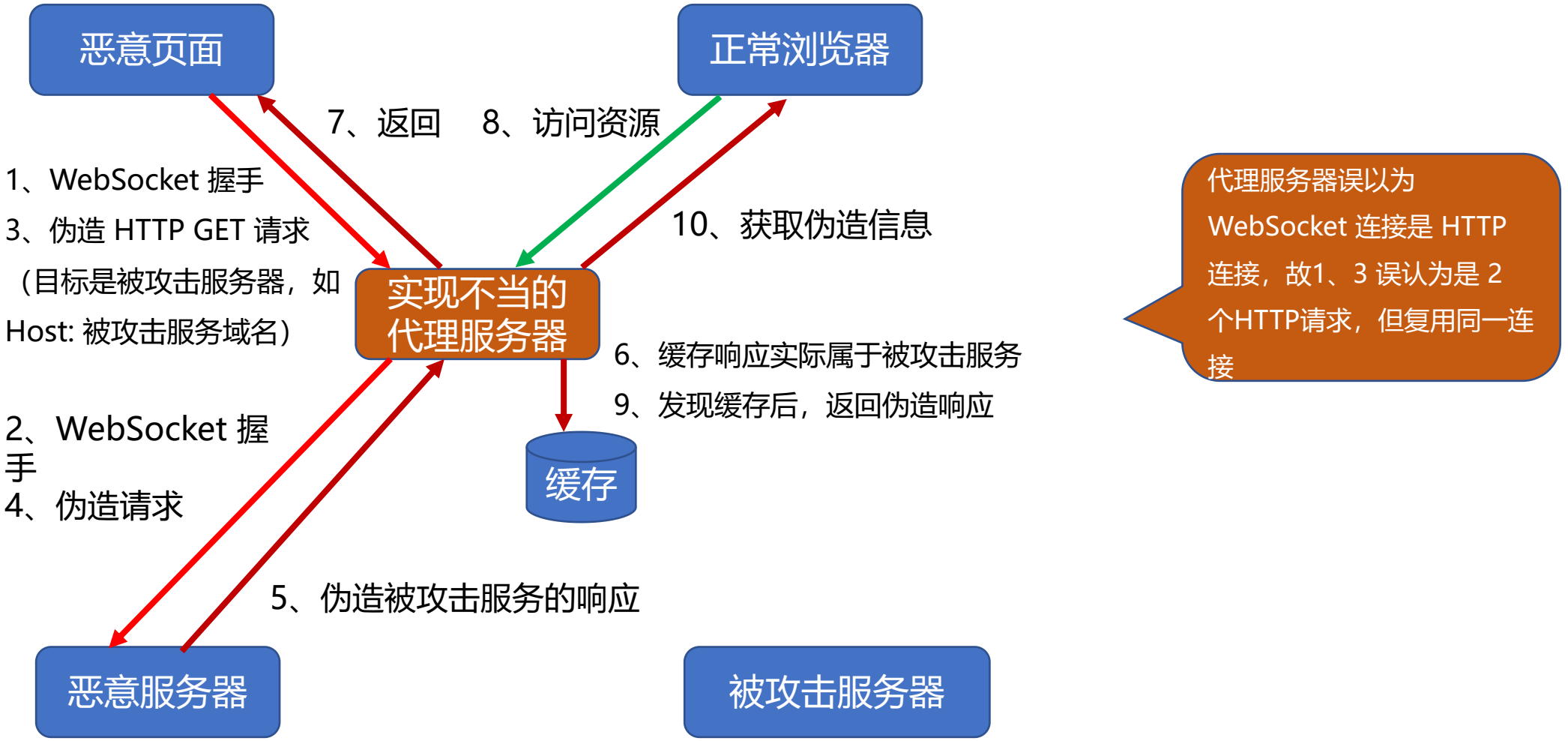
- 消息内容长度组成
  - 应用消息长度
  - 扩展数据长度
- <=125 字节
  - 仅使用 Payload len
- 126 至 2^16-1
  - Payload len 值为 126
  - Extended payload length16 位表示长度
- 2^16 至 2^64-1
  - Payload len 值为 127
  - Extended payload length 共 8 字节 64 位表示长度



发送消息

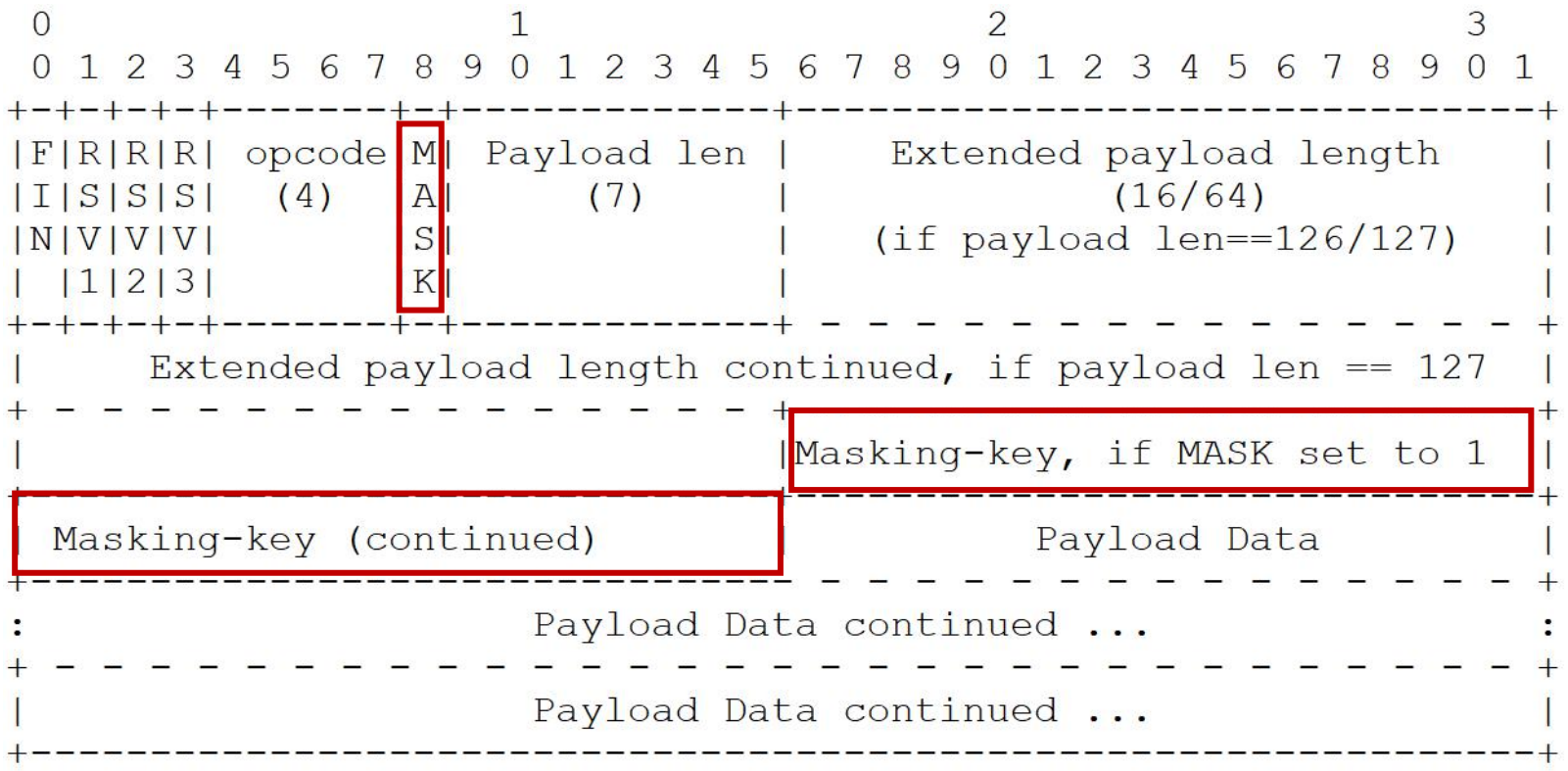
- 确保 WebSocket 会话处于 OPEN 状态
- 以帧来承载消息，一条消息可以拆分多个数据帧
- 客户端发送的帧必须基于掩码编码
- 一旦发送或者接收到关闭帧，连接处于 CLOSING 状态
- 一旦发送了关闭帧，且接收到关闭帧，连接处于 CLOSED 状态
- TCP 连接关闭后，WebSocket 连接才完全被关闭

针对代理服务器的缓存污染攻击



frame-masking-key 掩码

- 客户端消息：MASK 为 1（包括控制帧），传递 32 位无法预测的、随机的 Masking-key
- 服务器端消息：MASK 为 0



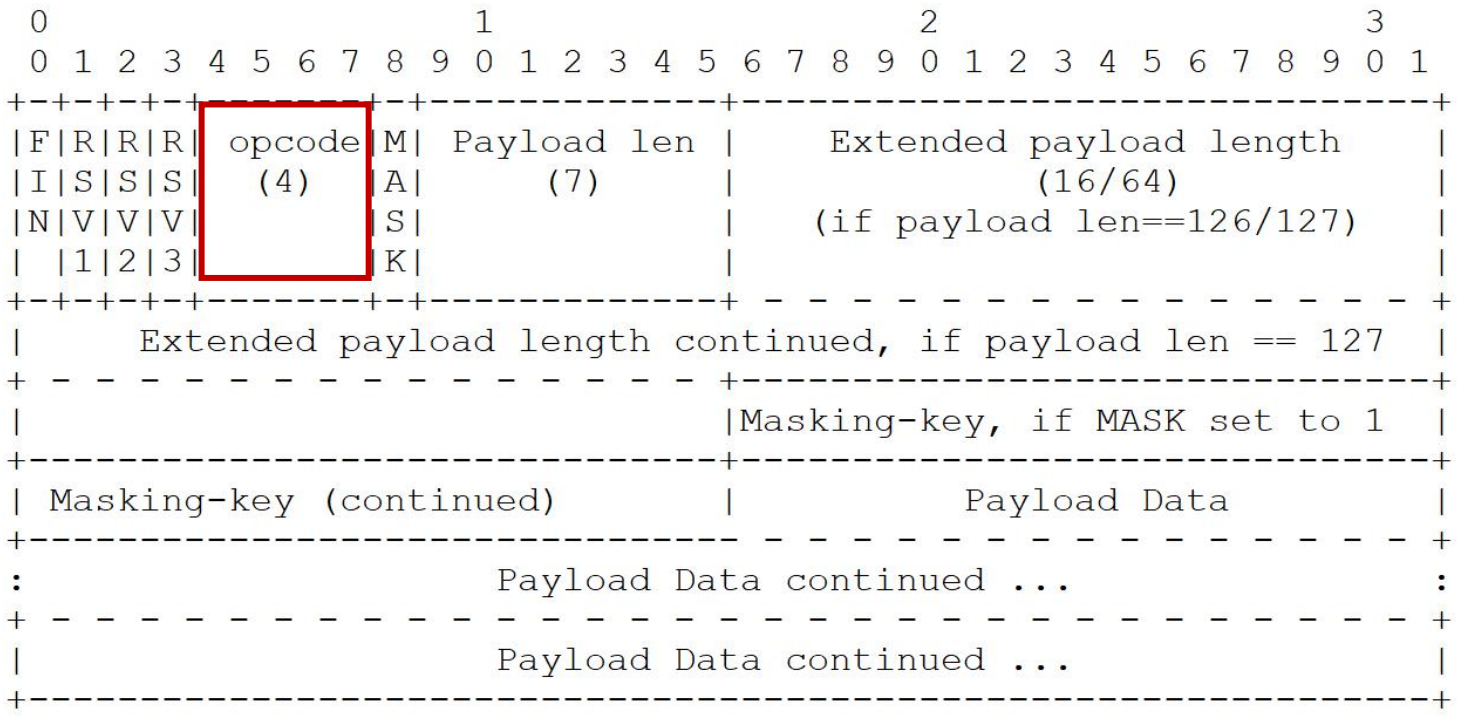
掩码如何防止缓存污染攻击？

- 目的：防止恶意页面上的代码，可以经由**浏览器**构造出合法的 GET 请求，使得代理服务器可以识别出请求并缓存响应

- 强制浏览器执行以下方法：
  - 生成随机的 32 位 frame-masking-key，不能让 JS 代码猜出（否则可以反向构造）
  - 对传输的包体按照 frame-masking-key 执行可对称解密的 XOR 异或操作，使代理服务器不识别
  - 消息编码算法：
    - j = i MOD 4
    - transformed-octet-i = original-octet-i XOR masking-key-octet-j

心跳帧

- 心跳帧可以插在数据帧中传输
  - ping 帧
    - opcode=9
    - 可以含有数据
  - pong 帧
    - opcode=A
    - 必须与 ping 帧数据相同



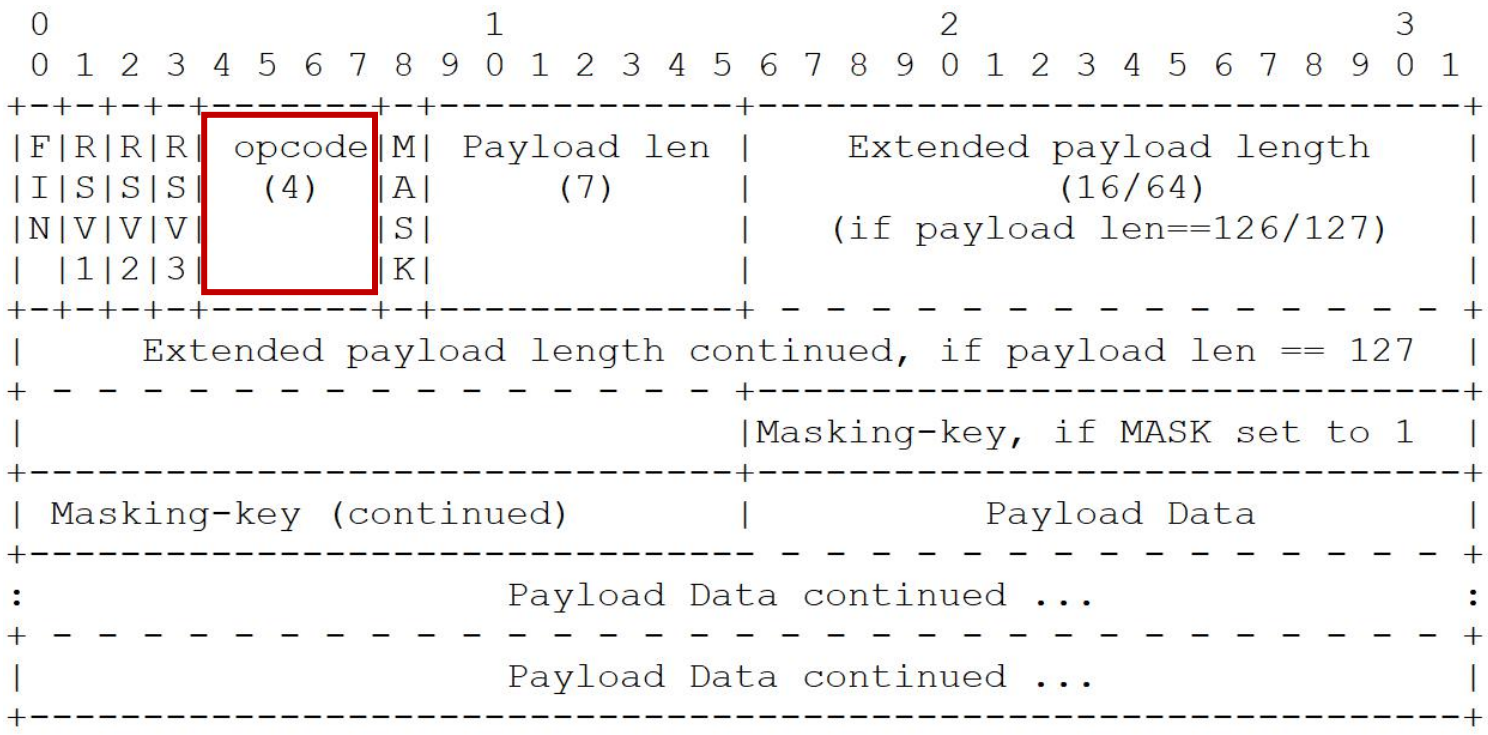
关闭会话的方式

- 控制帧中的关闭帧：在 TCP 连接之上的双向关闭
  - 发送关闭帧后，不能再发送任何数据
  - 接收到关闭帧后，不再接收任何到达的数据

- TCP 连接意外中断

关闭帧格式

- opcode=8
- 可以含有数据，但仅用于解释关闭会话的原因
  - 前 2 字节为无符号整型
  - 遵循 mask 掩码规则



关闭帧的错误码

错误码	含义
1000	正常关闭
1001	表示浏览器页面跳转或者服务器将要关机
1002	发现协议错误
1003	接收到不能处理的数据帧（例如某端不能处理二进制消息）
1004	预留
1005	预留（不能用在关闭帧里），期望但没有接收到错误码
1006	预留（不能用在关闭帧里），期望给出非正常关闭的错误码
1007	消息格式不符合 opcode（例如文本帧里消息没有用 UTF8 编码）
1008	接收到的消息不遵守某些策略（比 1003、1009 更一般的错误）
1009	消息超出能处理的最大长度
1010	客户端明确需要使用扩展，但服务器没有给出扩展的协商信息
1011	服务器遇到未知条件不能完成请求
1015	预留（不能用在关闭帧里），表示 TLS 握手失败