

对象存储&视频云

35条实战秘籍



众多阿里技术专家层层筛选
汇总35篇对象存储&视频云经典实录
从问题原因到适用产品
快速提升您的工单运维效率





阿里云开发者“藏经阁”

海量电子书免费下载

前言

本书从众多技术专家日积月累的典型问题中，经过层层筛选，为大家提供了 35 个经典问题答疑实录，内容颇具深度，具有深厚的技术底蕴，是理论和实践的完美结合。

通篇从“问题描述”到“排查思路&过程”，再到“问题原因”、“解决方案”、“适用产品”，层层探讨和剖析，相信每个经典问题都能让同学们从中挖掘到自己真正的信息需求，整合应用并输出，为后续处理类似疑难工单提升运维效率，降低工单处理时长。

无论你是哪个年龄段、身处哪个职位，希望本书能够让所有对“对象存储&视频云”感兴趣的朋友有所获有所得，对大家的日常生活和工作需求都能有所帮助。书中难免有笔误、差错或遗漏等问题，希望大家一起沟通交流个人见解，共同进步。

目录

CDN 更改源站后生效慢	6
全站加速 WebSocket 偶发访问失败-劫持行为	7
CDN 页面优化不生效排查遇到的坑	11
证书链不完整导致 SSL 握手失败	18
一次 HTTPS 访问慢的案例分析	22
CDN 访问 508 问题	28
CDN 加速 OSS 后未响应 Content-MD5	32
源站不支持 range 导致请求 CDN 异常断开	36
CDN 回源带宽突增	38
跨域问题-The 'Access-Control-Allow-Origin' header has a value 'xxx' that is not equal to the supplied origin	41
413-GET 请求不支持携带 body	44
通过阿里云 CDN 系列产品访问与直接访问源站得到的结果不一样	46
DNS 劫持导致访问目标地址失败并且解析主机后的 IP 不是 CDN 节点 IP	49
劫持问题导致通过 CDN 请求资源没有返回实际资源文件	53
TCP 劫持导致某地移动 CDN 节点 HTTPS 访问失败	56
预热文件卡住-海外 L2 回源国内源站跨境链路	60
设置了缓存规则但每次访问还是 MISS	61

抽丝剥茧定位一个 CDN 访问慢的案例	63
长连接访问的场景下偶发出现 CDN 主动发 FIN 包	70
开启 ossftp 以后端口不通导致连接失败	74
PythonSDK-从 OSS 下载文件报错 InconsistentError	79
通过 ffplay 播放 OSS 上的 mp3 文件会断开	82
跨境访问 OSS 失败	88
OSS 线上业务回调失败	90
SDK 上传 OSS 报错 413 状态码	93
OSS 服务端签名后直传报错 FieldItemTooLong	98
OSS 控制台列举文件超时	101
点播 HLS 加密转码以后生成的 m3u8 只有一个 ts 切片	104
视频直播流播放无声音	108
视频直播播放 ts 切片存在 404	110
视频直播播放 RTS 低延迟直播流报错 1000 错误码	111
一个 FLV 直播流起播异常慢的案例	113
推流音频声道变化导致录制文件 H5 上播放异常	117
视频直播录制文件长度不足	119
视频直播截图失败-服务角色问题	121

CDN 更改源站后生效慢

作者：胡夫

问题描述

CDN 更改源站后生效慢，还是存在回源老的源站的情况。

解决方案

概率性数据同步慢导致，一般是因为域名的访问量级比较小有关，如果访问量大那么刷新速度会比较快。这种情况可以考虑增加一个回原头强制更新 dns 。在 CDN 控制台添加添加"自定义回源请求头"，增加如下配置： Ali-Tproxy-Dns-Update: sync



适用于

CDN

DCDN

全站加速 WebSocket 偶发访问失败-劫持行为

作者：胡夫

问题描述

客户使用 DCDN 的 WebSocket 业务，偶发出现无法访问的情况，线上有 1%左右的失败率。根据客户沟通，该问题并不是必现的，目前是其中一位同事家里的 wifi 网络下能复现问题，而且重启家里的光猫以后能正常，但一段时间以后又会出现问题。

问题排查

1. 客户端信息

尝试让客户复现问题，提供客户端的 Netwrok 信息和客户端抓包信息。从抓包信息看，是源站响应了 500。进一步跟客户确认源站逻辑，根据客户的反馈是源站会先校验 Upgrade: websocket 这个请求头，如果没有这个请求头就会返回 500。但是奇怪的是，根据抓包文件看，客户端明明是带了 Upgrade: websocket 请求头的，因此怀疑是 DCDN 节点转发的问题。

根据 WebSocket 的 RFC 协议文档可以知道：

1. Websocket 请求必须包含一个 Upgrade header 字段，它的值必须包含"websocket"。
2. WebSocket 请求必须包含一个 Connection header 字段，它的值必须包含"Upgrade"。

有问题的请求，是这样的

2. 模拟访问测试

根据抓包信息，使用 curl 指定到对应的 DCDN 请求模拟发 WebSocket 请求测试，发现在客户侧能稳定复现的情况下，我们 curl 测试并不能复现，排查来看又跟节点没有关系。

```
curl -sv 'http://xxx.xxx.com/' -H'Connection: Upgrade' -H'Upgrade: websocket' -H'Sec-WebSocket-Key: pChMykJwUTQB14FCbP2Beg==' -H'Sec-WebSocket-Version: 13' -H'Origin:http://xxx.xxx.com' -H'Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits' -H'Accept-Encoding: gzip, deflate' -x 116.253.2.9.212:80
```

3. 源站打印

尝试让客户在源站侧加一下打印，将收到的请求头打印出来，从源站的打印来看，出问题的时候的确是没有收到 Upgrade: websocket 头。而如果是正常的请求，源站是能打印出 Upgrade: websocket 头的。如果尝试在源站侧去掉这个 Upgrade 的校验，虽然可以请求成功，但是无法进行 WebSocket 通信，因为整条链路并不是走的 WebSocket。

4. 排查 CDN 日志

排查后台 CDN 的日志，发现正常请求的 hit_info 字段显示的是 WS，也就是走的 WebSocket 请求。而异常请求的 hit_info 字段是 dynamic，也就是走的动态加速，并不是 WebSocket。从现象分析来看，怀疑是 L1 收到的请求里就没有 Upgrade: websocket 相关的头，所以 L1 认为这不是 WebSocket 请求，走的是普通的动态加速，转发回源站也不会带 Upgrade 头。

hit_info	-,-WS CHARGE NOTLAST
hit_info	-,-DYNAMIC CHARGE NOTLAST

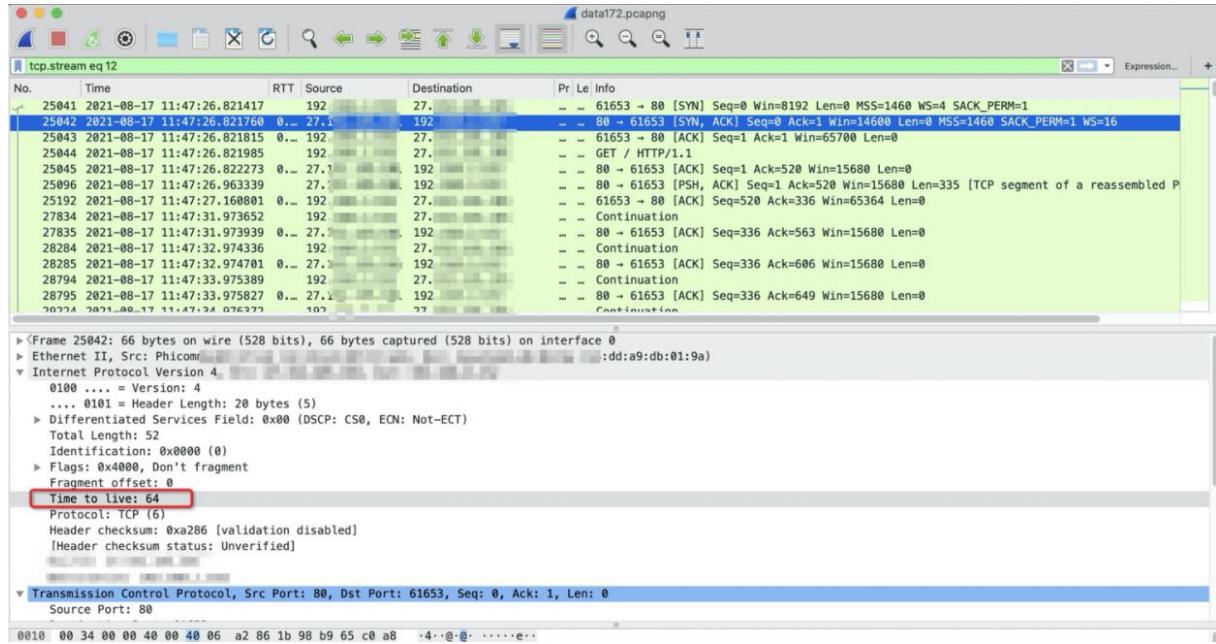
5. DCDN 侧增加 debug 日志

由于日志里并没有记录收到的客户端 Upgrade 字段，因此需要后台加一下 debug 日志，打印下该字段。

6. 重新复现问题

重新让客户复现了一次问题，并提供客户端抓包数据。从抓包数据看，客户端请求确实带了 Upgrade 头，而从 DCDN 的日志来看，确实是没有收到 Upgrade 头的，因此走的也是普通动态加速

请求。同时从客户端收到的来自 DCDN 的报文看，这个 TTL 是 64，这明显是不正常的。正常情况由于 DCDN 节点发出的报文 TTL 原始值是 64，中间每经过一个网络路由节点，TTL 减 1，客户端收到的报文 TTL 不应该是 64。因此判断是客户端存在劫持行为，走了代理，应该是客户端发出请求以后客户端层面做了异常转发，按照普通的 HTTP 请求转发，没有转 WebSocket 相关头导致的。



问题解决

配置 HTTPS 证书，客户端走 wss 请求以后，客户侧能复现问题的现场就正常了，无法再复现问题，而且线上错误率明显改善。

适用产品

DCDN

CDN 页面优化不生效排查遇到的坑

作者：胡夫

背景描述

阿里云 CDN 提供了[页面优化](#)功能，当开启页面优化功能时，CDN 可以自动清除 HTML 页面冗余的注释和重复的空白符，缩小文件体积，提升页面可阅读性。本案例遇到的一个问题是，按照文档开启了 CDN 的页面优化功能，但是访问 HTML 页面实际并没有优化效果。

Gzip 压缩引发

在排查测试的过程中，发现直接用 curl URL 的方式查看响应结果，是已经被页面优化了，但是直接在浏览器上访问 HTML 却没有被优化。进一步对比 curl 请求以及浏览器 Network 下的 Request Header 以及 Response Header，发现浏览器的 Request Header 带了 Accept-Encoding: gzip, deflate，Response Header 返回了 Content-Encoding: gzip，如下图所示：

```
▼ Request Headers      view source
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng
n/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cache-Control: no-cache
Connection: keep-alive
Cookie: Hm_lvt_e933c796aa676cbb0afa2bf2bee228fd=1602845308; Hm_lpvt_e933c796aa676cbb0afa2bf2be
Host: www.51jiaoxi.com
Pragma: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Ge
121 Safari/537.36
```

The screenshot shows the 'Headers' tab in a browser's developer tools. Under 'Response Headers', the following headers are listed:

- Access-Control-Allow-Credentials: true
- Access-Control-Allow-Headers: Accept, x-xsrf-token, DNT, X-Mx-ReqToken, Keep-Alive, User-Agent-Since, Cache-Control, Content-Type, Authorization
- Access-Control-Allow-Methods: GET, POST, OPTIONS, PUT, DELETE
- Access-Control-Expose-Headers: Content-Disposition
- Cache-Control: no-cache, private
- Connection: keep-alive
- Content-Encoding: gzip** (highlighted in blue)
- Content-Type: text/html; charset=UTF-8
- Date: Tue, 20 Oct 2020 06:06:27 GMT
- EagleId: 6e5084a516031739873566665e
- Server: Tengine

我们知道 HTTP 协议上的 GZIP 编码是一种用来改进 WEB 应用程序性能的技术，大流量的 WEB 站点常常使用 GZIP 压缩技术来让用户感受更快的速度。从测试来看，浏览器请求头带了 Accept-Encoding: gzip 表示浏览器支持解码 gzip 压缩后的文件，如果服务器支持 gzip 压缩，那么在收到这个请求头以后，就会返回 gzip 压缩以后的文件，在 Response Header 里以 Content-Encoding: gzip 的形式体现。直接用如下 curl 命令带 Accept-Encoding 请求头测试，发现返回结果也是带了 gzip 的，且页面也是没有优化的。

```
curl -I 'http://测试 URL' -H 'Accept-Encoding: gzip'
```

综上可以暂时得出结论，返回结果带了 Accept-Encoding: gzip 以后，CDN 的页面压缩不生效，返回结果不带 Accept-Encoding: gzip，CDN 的页面压缩生效。

Gzip 压缩是在哪里发生的

由于请求的链路是：客户端 Client -->CDN -->源站

这个 Gzip 压缩，可能是 CDN 产生的，也可能是源站产生的。这个比较容易验证，直接用 curl 命令绑定到源站 IP 去测试即可得出结论：

```
curl -I 'http://测试 URL' -H 'Accept-Encoding: gzip' -x '源站 IP:80'
```

测试来看，返回结果果然带了 Accept-Encoding: gzip，说明是源站 gzip 压缩导致的。产生这个现象的原因逐渐浮出水面，**整个过程如下：**

当用户在浏览器发起请求时，浏览器默认带了 Accept-Encoding: gzip 这个请求头，CDN 作为一个代理服务器，在回源请求源服务器的时候转发了这个来自真实客户端（浏览器）的请求头，源服务器由于开了 Gzip 压缩，因此在收到这个请求头以后返回了 Gzip 压缩以后的内容给 CDN，由于 CDN 不具备 Gunzip 功能，因此无法对 Gzip 压缩以后的内容去做页面优化，因此导致了页面优化功能不生效。

Gzip 配置参数

以上基本定位问题，但是为了更明确，我搭建了一个基于 Nginx 的 Web 服务器用做 CDN 的源站，并在 Nginx 的配置文件 nginx.conf 里开启了 Gzip 压缩，配置如下图：

```
gzip on;
gzip_min_length 1k;
gzip_buffers 4 16k;
gzip_http_version 1.1;
gzip_comp_level 2;
gzip_types    text/plain application/javascript application/x-javascript text/css application/xml application/xml+rss;
gzip_vary on;
gzip_proxied expired no-cache no-store private auth;
gzip_disable "MSIE [1-6]\.";
```

但是测试发现，通过 CDN 访问，并且发了 Accept-Encoding: gzip 请求头以后，CDN 依然可以完成页面压缩，这就比较奇怪。为了定位问题，直接在 Web 服务器上抓了包，看一下 CDN 和 Web 服务器的交互请求，发现一个很奇怪的现象：CDN 请求 Web 服务器的时候转发了 Accept-Encoding: gzip，但是 Web 服务器并没有响应 Content-Encoding: gzip，报文如下图：

```

GET /doc-5737684.html?abc=1 HTTP/1.1
Host: shj.pier39.cn
Via: cn508.l1, cache8.cn508, l2cn1801.l2, cache14.l2cn1801
Eagleeye-Traceid: 7d4da71916031270515047779e
Ali-Swift-Log-Host: [REDACTED]
Ali-Swift-Stat-Host: level2.[REDACTED]
X-Forwarded-For: 101.71.38.193
X-Client-Scheme: http
Ali-Cdn-Real-Ip: 101.71.38.193
User-Agent: curl/7.68.0
Accept: */
Accept-Encoding: gzip, deflate
Ali-Swift-5Xx-No-Retry: on

HTTP/1.1 200 OK
Server: nginx
Date: Mon, 19 Oct 2020 17:04:11 GMT
Content-Type: text/html
Content-Length: 63482
Last-Modified: Mon, 19 Oct 2020 15:06:46 GMT
Connection: keep-alive
Vary: Accept-Encoding
ETag: "5f8dab86-f7fa"
Expires: Wed, 18 Nov 2020 17:04:11 GMT
Cache-Control: max-age=2592000
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: X-Requested-With
Access-Control-Allow-Methods: GET,POST,OPTIONS
Accept-Ranges: bytes

```

根据这个现象，去查了一下 Nginx 官网对于 `ngx_http_gzip_module` 模块的配置说明，可以看到该模块有如下配置参数：

Learn how to embed security in your DevOps pipeline.
Download the Free Ebook on Web Application Security.

Module `ngx_http_gzip_module`

[Example Configuration](#)

[Directives](#)

- [gzip](#)
- [gzip_buffers](#)
- [gzip_comp_level](#)
- [gzip_disable](#)
- [gzip_http_version](#)
- [gzip_min_length](#)
- [gzip_proxied](#)
- [gzip_types](#)
- [gzip_vary](#)

[Embedded Variables](#)

The `ngx_http_gzip_module` module is a filter that compresses responses using the “gzip” method. This often helps to reduce the size of transmitted data by half or even more.

[english](#)
[русский](#)

[news](#)
[about](#)
[download](#)
[security](#)
[documentation](#)
[faq](#)
[books](#)
[support](#)

[trac](#)
[twitter](#)
[blog](#)

其中有一个 `gzip_proxied` 参数引起了注意。这个参数的含义，可以解释如下：

语法: gzip_proxied [off|expired|no-cache|no-store|private|no_last_modified|no_etag|auth|any] ...

默认值: gzip_proxied off

作用域: http, server, location

Nginx 作为反向代理的时候启用, 开启或者关闭后端服务器返回的结果, 匹配的前提是后端服务器必须要返回包含”Via”的 header 头。

off – 对于所有来自代理服务器的请求, 都关闭压缩

expired – 如果响应 header 头中包含 “Expires” 头信息则启用压缩

no-cache – 如果响应 header 头中包含 “Cache-Control:no-cache” 头信息则启用压缩

no-store – 如果响应 header 头中包含 “Cache-Control:no-store” 头信息则启用压缩

private – 如果响应 header 头中包含 “Cache-Control:private” 头信息则启用压缩

no_last_modified – 如果响应 header 头中不包含 “Last-Modified” 头信息则启用压缩

no_etag – 如果响应 header 头中不包含 “ETag” 头信息则启用压缩

auth – 如果响应 header 头中包含 “Authorization” 头信息则启用压缩

any – 无条件启用压缩, 也就是对任何来自代理服务器的请求, 都返回压缩的内容

Enables or disables gzipping of responses for proxied requests depending on the request and response. The fact that the request is proxied is determined by the presence of the “Via” request header field. The directive accepts multiple parameters:

```
off
    disables compression for all proxied requests, ignoring other parameters;
expired
    enables compression if a response header includes the “Expires” field with a value that disables caching;
no-cache
    enables compression if a response header includes the “Cache-Control” field with the “no-cache” parameter;
no-store
    enables compression if a response header includes the “Cache-Control” field with the “no-store” parameter;
private
    enables compression if a response header includes the “Cache-Control” field with the “private” parameter;
no_last_modified
    enables compression if a response header does not include the “Last-Modified” field;
no_etag
    enables compression if a response header does not include the “ETag” field;
auth
    enables compression if a request header includes the “Authorization” field;
any
    enables compression for all proxied requests.
```

由于 Nginx 配置里配置了 gzip_proxied expired no-cache no-store private auth

因此相当于启用了 gzip_proxied 参数，当 Web 服务器发现来自代理服务器的请求时（在这里就是来自 CDN 的请求），Web 服务器会去校验 gzip_proxied 参数，当发现服务器的 Response Header 里没有返回 Expires、"Cache-Control:no-cache"等类似响应头时，服务器就返回了不带 Gzip 压缩的数据。如果 Gzip 配置模块是按照如下配置的话，那么任何来自代理服务器的请求，服务器都会返回 Gzip 压缩的内容。

```
gzip_proxied any
```

如何判断请求是来自代理服务器的

那么问题来了，服务器是如何判断这个请求是来自代理服务器的，而不是真实客户端呢。这里就涉及到 Via 这个 HTTP Header 了。关于 Via 的介绍，可以参考 HTTP 协议关于 [Via 的文档](#)。Via 是一个通用首部，是由代理服务器添加的，适用于正向和反向代理，在请求和响应首部中均可出现。这个消息首部可以用来追踪消息转发情况，防止循环请求，以及识别在请求或响应传递链中消息发送者对于协议的支持能力。在这里，CDN 作为代理服务器，去请求源服务器的时候，请求头里会带上 Via 头（这点在上面的抓包截图里也可以看到），而服务器就是根据请求头里的 Via 得知该请求是来自上游代理服务器的。

HTTP 服务器的问题是知道代理本身是否能够处理压缩响应。传入请求中的接受编码头(也就是 Accept-Encoding: gzip)很可能是由原始客户机请求提供的，但这并不能表明它所经过的代理或网关的能力，也就是说，服务器并不知道上游代理服务器能否处理 Gzip 压缩以后的内容。因此，在此场景中，服务器采用最安全的选项，并选择不压缩它发回的响应，这也是合理的。关于 Via 这个 Header 对于 Gzip 压缩的影响，可以参考[这篇 Akamai 的文章](#)，有详细的介绍。

一个新的想法

既然源站响应了 gzip 内容会导致 CDN 的页面优化不生效，那只能源站响应未压缩过的内容，

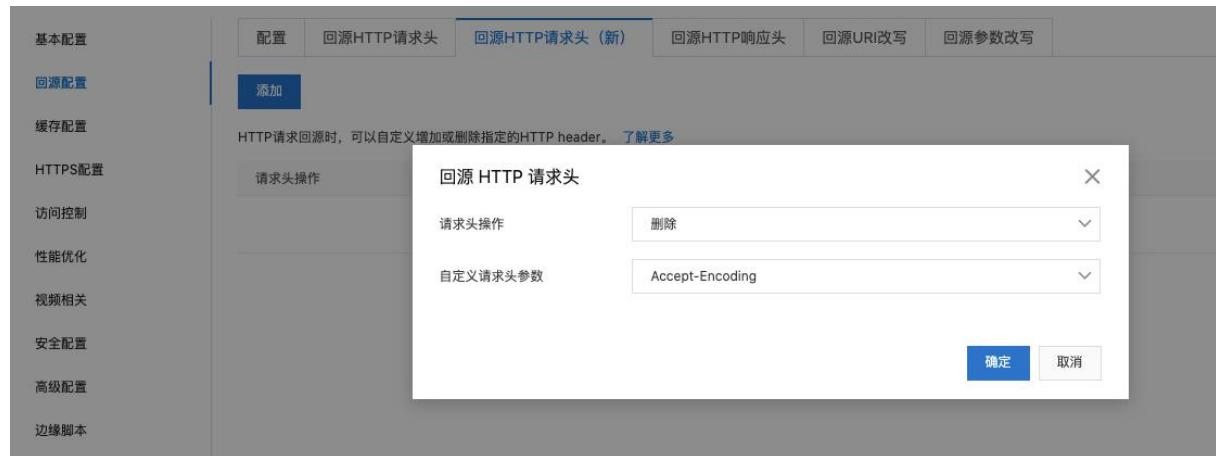
但是这样的话，最终对于客户端来说，请求到的文件没有经过有效压缩，还是会消耗客户端带宽，从而影响 Web 页面的访问性能。而 CDN 除了提供页面优化的功能外，还提供了 Gzip 的功能，那能不能在 CDN 层面去做页面优化和 Gzip 压缩呢？

通过测试发现，在 CDN 开启页面优化的同时，无论是开启 Gzip 压缩还是 Br 压缩，只要请求头带了 accept-encoding: gzip, deflate, br 头，则页面优化不生效。由此可见，在 CDN 层面，智能压缩的优先级比较高，压缩以后的内容无法页面优化，看来这个方案暂时也不可行。当然，这部分的策略有待产品层面去改良。

总结和解决方案

综上，该问题可以总结如下：

- (1) 如果源站响应了 Gzip 压缩的内容，CDN 会因为无法 Gunzip 导致页面优化功能不生效
- (2) 如果希望与 CDN 去智能压缩和页面优化，因此 CDN 层面智能压缩的优先级比页面优化的优先级高，也会导致页面优化不生效
- (3) 如果期望使用 CDN 的页面优化，那么需要确保源站服务器关闭 Gzip 压缩。如果源站服务器是 Nginx，通过修改 Nginx 配置文件里 ngx_http_gzip_module 模块的 gzip_proxied 参数，设定来自代理服务器的请求，不返回 Gzip 压缩的内容来实现。
- (4) 另外还有一种比较实现方案，是可以在 CDN 层面配置删除 Accept-Encoding 这个回源 HTTP 请求头。



证书链不完整导致 SSL 握手失败

作者：胡夫

问题描述

使用 OSS 的 NodeJS SDK，开启 CNAME 的情况下，HTTPS 请求自定义域名，报错 SSL 握手失败。

```
let client = new OSS({
  bucket: '<BucketName>',
  // region 以杭州为例 (oss-cn-hangzhou)，其他 region 按实际情况填写。
  endpoint: 'https://resource.xxxx.com',
  // 阿里云主账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 账号进行 API
  // 访问或日常运维，请登录 RAM 控制台创建 RAM 账号。
  accessKeyId: '<AccessKeyId>',
  accessKeySecret: '<AccessKeySecret>',
  cname:'true'
});
```

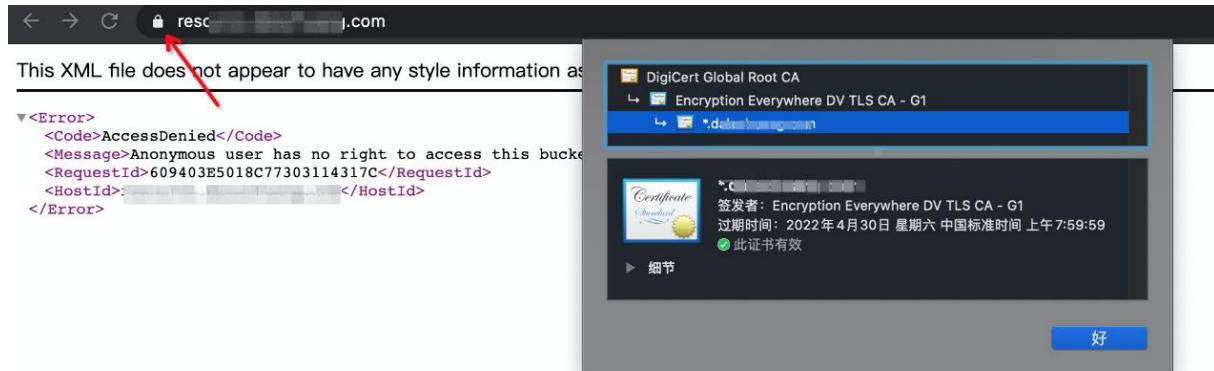
```
sh-3.2# node put.js
Error [ResponseError]: write EPROTO 4570844608:error:14094410:SSL routines:ssl3_read_bytes:sslv3 alert handshake failure:.../deps/openssl/openssl/ssl/rec_layer_s3.c:15
44:SSL alert number 40
, PUT https://res...com/wstest.heic -1 (connected: true, keepalive socket: false, agent status: {"createSocketCount":1,"createSocketErrorCount":0,"closeSocketCount":0,"errorSocketCount":0,"timeoutSocketCount":0,"requestCount":0,"freeSockets":{},"sockets":{"resource.dalezhuang.com:443:::::::::::1"},"requests":{}}, socketHandledRequests: 1, socketHandledResponses: 0)
headers: {}
  at Client.requestError (...) /Downloads/node_modules/ali-oss/lib/client.js:324:13)
  at Client.request (...) /Downloads/node_modules/ali-oss/lib/client.js:203:22)
    status: -1,
    code: 'ResponseError'
}
sh-3.2#
```

排查过程

1. 浏览器访问

从报错信息来看，ssl3_read_bytes:sslv3 alert handshake failure，基本上是可以判断 SSL 握

手失败了。但是通过浏览器访问却可以正常访问，查看证书状态正常。



2. 抓包看

抓包来看确实在 SSL 握手的时候，客户端发完 Client Hello，服务端直接发了 FIN 包。

No.	Time	RTT	Source	Destination	Protocol	Length	Info
1	2021-05-06 20:47:44,153677		30.43.112.25	120.77.167.169	TCP	66	50781 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	2021-05-06 20:47:44,182316	0.028632...	0.000209...	120.77.167.169	TCP	66	443 → 50781 [SYN, ACK] Seq=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
3	2021-05-06 20:47:44,182519	0.000209...	30.43.112.25	120.77.167.169	TCP	54	50781 → 443 [ACK] Seq=1 Ack=1 Win=131328 Len=0
4	2021-05-06 20:47:44,183233		30.43.112.25	120.77.167.169	TLSv1.2	431	Client Hello
5	2021-05-06 20:47:44,214066	0.030833...	120.77.167.169	30.43.112.25	TCP	60	443 → 50781 [ACK] Seq=1 Ack=378 Win=30720 Len=0
6	2021-05-06 20:47:44,214199		120.77.167.169	30.43.112.25	TLSv1.2	61	Alert (Level: Fatal, Description: Handshake Failure)
7	2021-05-06 20:47:44,214337		120.77.167.169	30.43.112.25	TCP	60	443 → 50781 [FIN, ACK] Seq=8 Ack=378 Win=30720 Len=0
8	2021-05-06 20:47:44,214430	0.000093...	30.43.112.25	120.77.167.169	TCP	54	50781 → 443 [ACK] Seq=378 Ack=9 Win=131328 Len=0
9	2021-05-06 20:47:44,217372		30.43.112.25	120.77.167.169	TCP	54	50781 → 443 [FIN, ACK] Seq=378 Ack=9 Win=131328 Len=0
10	2021-05-06 20:47:44,245456	0.028084...	120.77.167.169	30.43.112.25	TCP	60	443 → 50781 [ACK] Seq=9 Ack=379 Win=30720 Len=0

Frame 7: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
 ▷ Ethernet II, Src: RuijieNE_94:5d:39 (00:74:9c:94:5d:39), Dst: IntelCor_20:c3:6c (50:eb:71:20:c3:6c)
 ▷ Internet Protocol Version 4, Src: 120.77.167.169, Dst: 30.43.112.25
 ▷ Transmission Control Protocol, Src Port: 443, Dst Port: 50781, Seq: 8, Ack: 378, Len: 0

3. 查看证书链

基于 1 和 2 的初步判断，有可能会误以为是 SDK 的问题从而误导排查，其实遇到这种情况很有可能就是证书链不完整导致的。可以先用 openssl 工具检查下证书或者用相关证书检测平台检测下证书链，比如 <https://myssl.com/> 或者 https://www.sslceshi.com/ssl_check/

以下三个截图是本案例中分别用上面这两个网站以及用 openssl 检查的结果，可以看到确实是证书链不完整导致的。

MySSL.com | 亞洲最值得信赖的 SSL/TLS 测试平台

检测部署SSL/TLS的服务是否符合行业最佳实践，PCI DSS支付卡行业安全标准，Apple ATS规范。

评级: A+ (绿色)

ATS: 合规

PCI DSS: 合规

降级原因:

- 证书链不完整，降级为B。具体详情可参考《缺少证书链的问题和解决办法》。

配置指南:

证书详情

IP: [REDACTED]

主域名: [REDACTED]

包含域名: [REDACTED]

域名是否匹配: 匹配

证书有效时间: 2021/4/29上午8:00:00 - 2022/4/30上午7:59:59 剩余358天 未过期

证书指纹(SHA1): [REDACTED]

私钥长度: 2048

算法: sha256WithRSA

根证书: Encryption Everywhere DV TLS CA - G1

颁发机构: Encryption Everywhere DV TLS CA - G1

可信颁发机构: 否

验证状态信息: 不能验证签名，因为证书链只包含一个证书

证书链详情

```

PSK identity: None
PSK identity hint: None
SRP username: None
TLS session ticket lifetime hint: 300 (seconds)
TLS session ticket:
0000 - 9a ...Am.5.=....Y..] Xg...r...9.....
0010 - 58 ...lb>.X..H.....$..lb>.X..H.....
0020 - 24 ...Q.....$.vY.#..Q.....$.vY.#..
0030 - 51 1e 13 fe 86 00 24 d3-ba 00 76 59 f2 23 05 f7 Q.....$.vY.#..Q.....$.vY.#..
0040 - c1 8c 84 03 b2 ca f4 60-93 4a fc 2d c4 07 6a fe .....`..J.-..j. ....`..J.-..j.
0050 - 9a 71 f5 33 52 af 4d 0c-a0 4c 4f 1a 92 b1 6d ea .q.3R.M..L0...m. .q.3R.M..L0...m.
0060 - 41 60 d9 26 43 c5 8f a7-54 a4 57 9a ec 8e d0 63 A`.&C...T.W....c A`.&C...T.W....c
0070 - 9c 47 e6 fc 5b 4d 50 f8-1f 7f 4a dc d5 7a 20 27 .G..[MP...J..z ' .G..[MP...J..z '
0080 - b0 da d7 b6 df 25 d0 d8-e5 bd 55 08 35 c9 8c f0 .....%....U.5... (....C.,#X....Q..
0090 - 28 f0 16 81 43 f4 2c 23-58 c7 1e be 0a 51 16 d2 ...;07..e.....(....C.,#X....Q..
00a0 - 8a a4 b6 3b 30 37 e0 e5-65 f0 db e5 d8 1e a9 c2 ...;07..e.....(....C.,#X....Q..
00b0 - 58 1b bd 5a 46 cf dd ba-f6 65 b1 65 d1 81 7e e2 X..ZF....e.e..~. ...;07..e.....(....C.,#X....Q..
00c0 - 27 64 99 6a 75 9a 2f 63-3d 59 d9 01 e0 0c fc 7c 'd.ju./c=Y.....| X..ZF....e.e..~.

Start Time: 1620308311
Timeout : 7200 (sec)
Verify return code: 21 (unable to verify the first certificate)
Extended master secret: no

```

解决方案

补全证书链，可以参考这个文档里介绍的补全证书链的方案。

适用于

存储 OSS

CDN

一次 HTTPS 访问慢的案例分析

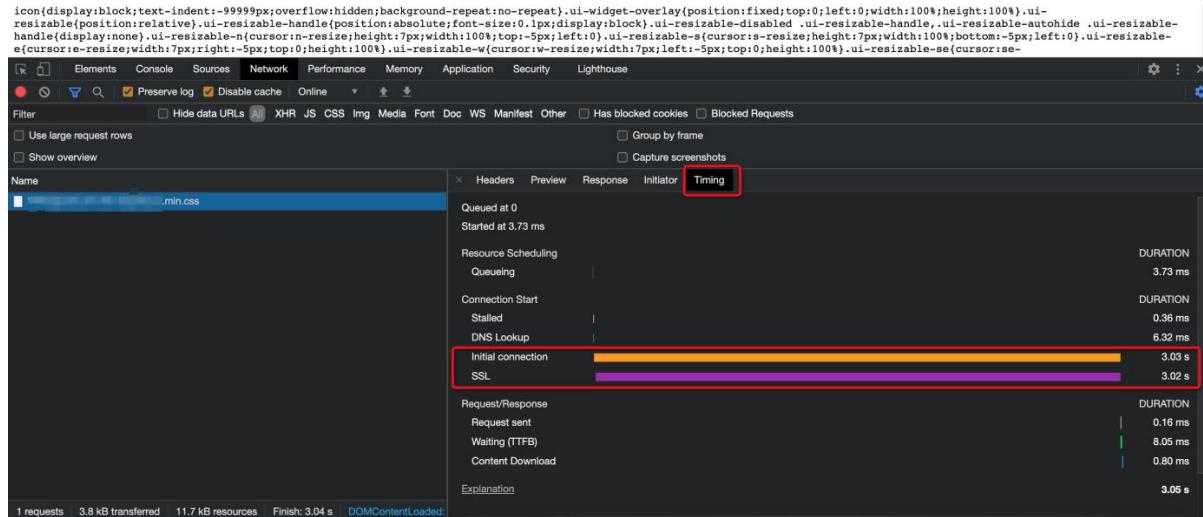
作者：胡夫

问题描述

用户反馈一个通过 CDN 加速的 CSS 资源，访问速度很慢，测试在 PC 上访问超过了 3 秒。

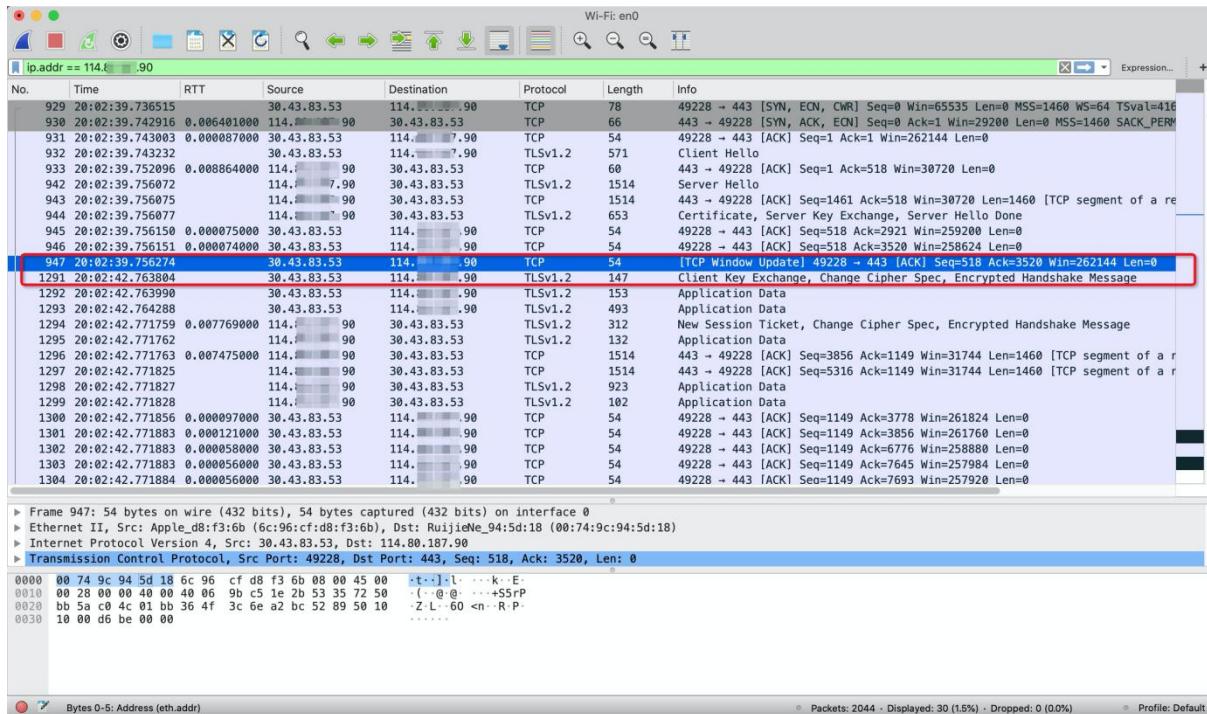
Netwrok-Timing 分析

Chrome 复现了一下问题，通过 Network 下的 Timing 信息分析这一次慢请求主要耗时在哪个时间点。通过以下图可以明显看到，这一次请求总耗时 3.05s，但是有 3.02s 是耗时在了 SSL 握手上。



抓包分析

我们知道 HTTPS 请求，在完成 TCP 三次握手以后，需要通过 TLS 协议来完成 SSL 握手，那么这次 SSL 握手的时间为什么耗时这么久呢？于是抓了一个异常包，根据 CDN 节点 IP 过滤了一下交互报文，具体如下图。



通过这个报文可以看到#944 号报文 Server 端响应了 Server Hello Done，返回了证书信息给客户端，客户端响应 ACK 以后在#947 号报文更新 TCP Window 滑动窗口，然后在#1291 报文才继续发 Client Key Exchange 给 Server 端。从#947 号包到#1291 包，时间也从 20:02:39 到了 20:02:42，耗时了 3 秒左右。从现象来看，这个耗时主要是“卡”在客户端这了，那么客户端这个时间到底干啥去了呢？

证书校验机制

要解释上面问题，还需要知道证书校验的机制。对于一个可信任的 CA 机构颁发的有效证书，在证书到期之前，只要 CA 没有把其吊销，那么这个证书就是有效可信任的。有时，由于某些特殊原因（比如私钥泄漏，证书信息有误，CA 有漏洞被黑客利用，颁发了其他域名的证书等等），需要吊销某些证书。那浏览器或者客户端如何知道当前使用的证书已经被吊销了呢，通常有两种方式：CRL (Certificate Revocation List，证书吊销列表) 和 OCSP (Online Certificate Status Protocol，在线证书状态协议)。

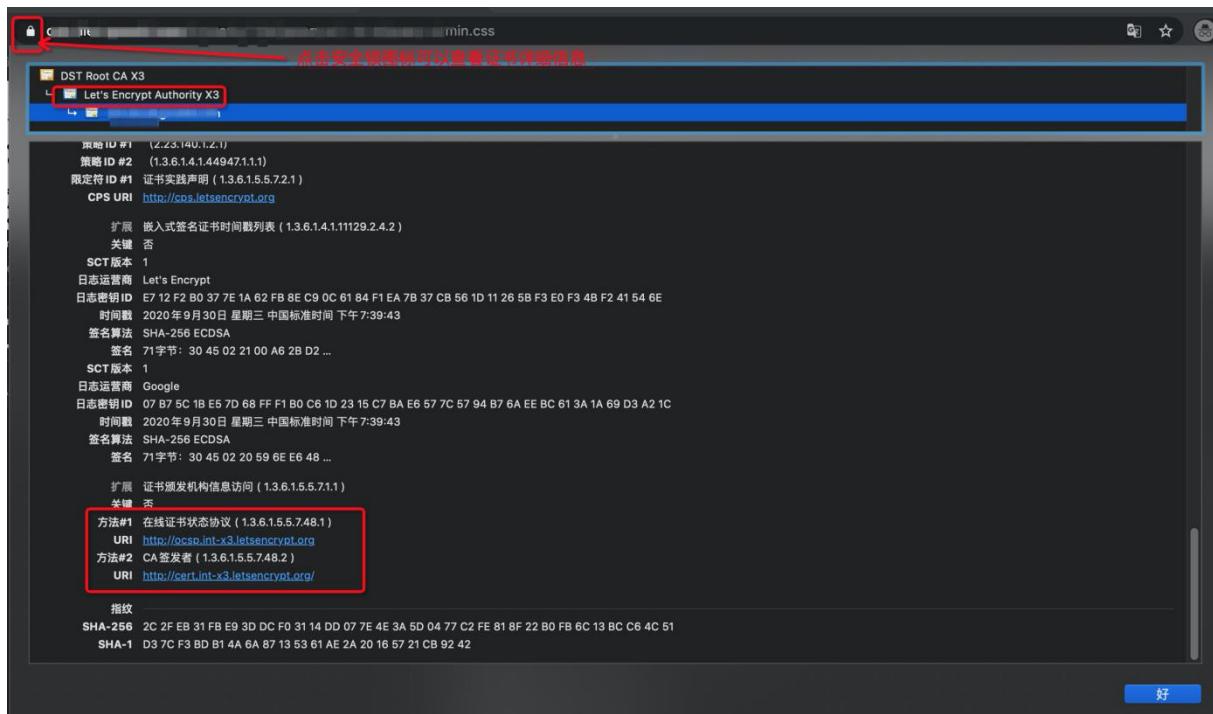
(1) CRL: CRL 是由 CA 机构维护的一个列表，列表中包含已经被吊销的证书序列号和吊销时间。浏览器可以定期去下载这个列表用于校验证书是否已被吊销。可以看出，CRL 只会越来越大，而且当一个证书刚被吊销后，浏览器在更新 CRL 之前还是会信任这个证书的，实时性较差。在每个证书的详细信息中，都可以找到对应颁发机构的 CRL 地址。

(2) OCSP: OCSP 是一个在线证书查询接口，它建立一个可实时响应的机制，让浏览器可以实时查询每一张证书的有效性，解决了 CRL 的实时性问题，但是 OCSP 也引入了一个性能问题，某些客户端会在 SSL 握手时去实时查询 OCSP 接口，并在得到结果前会阻塞后续流程，这对性能影响很大，严重影响用户体验。（OCSP 地址也在证书的详细信息中）

通过浏览器上点击安全锁的小图标，可以看到证书的详细信息，如下图，我们可以看到几个关键信息。

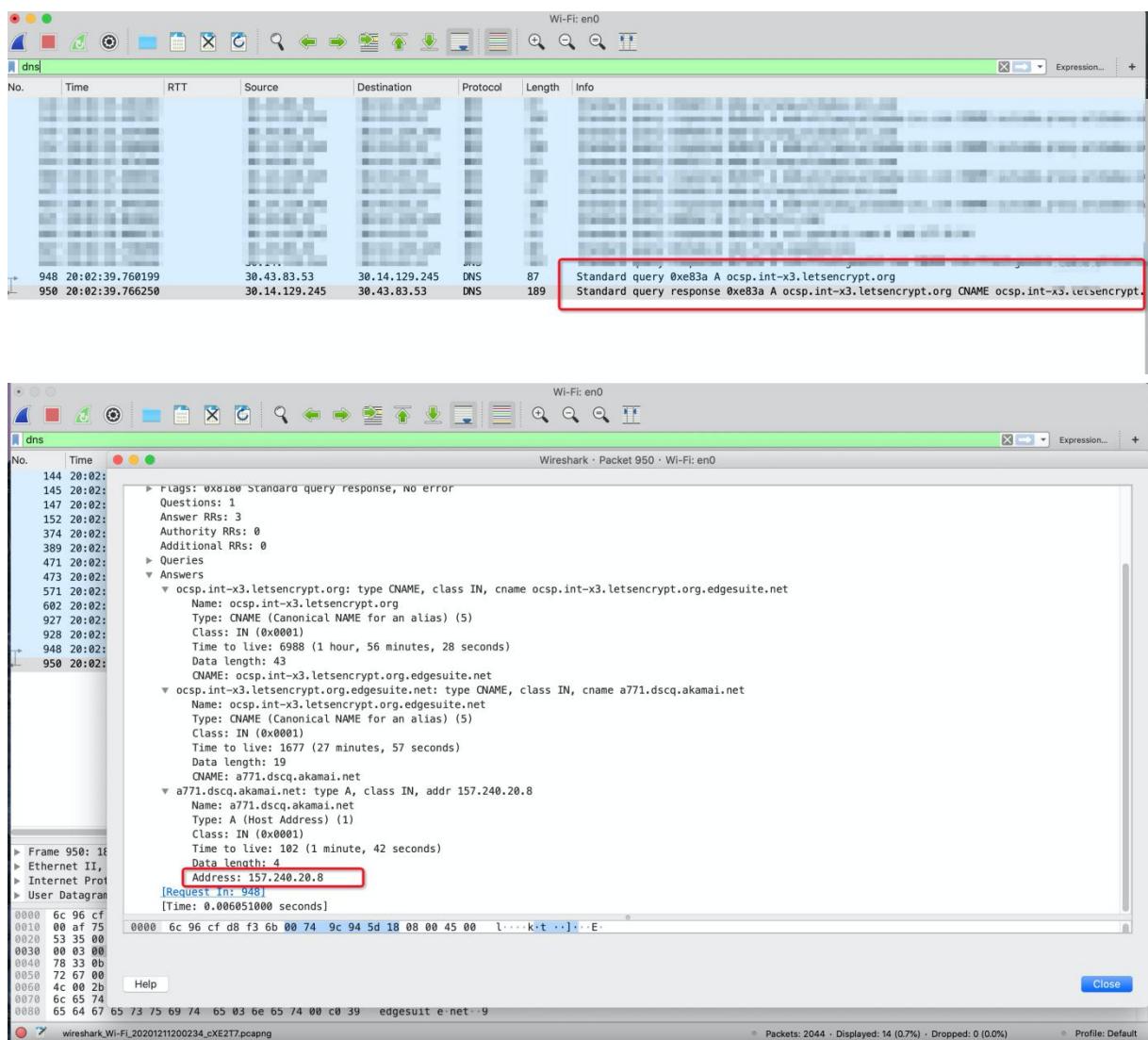
证书签发者：Let's Encrypt Authority X3

OCSP_ServerURL: <http://ocsp.int-x3.letsencrypt.org>

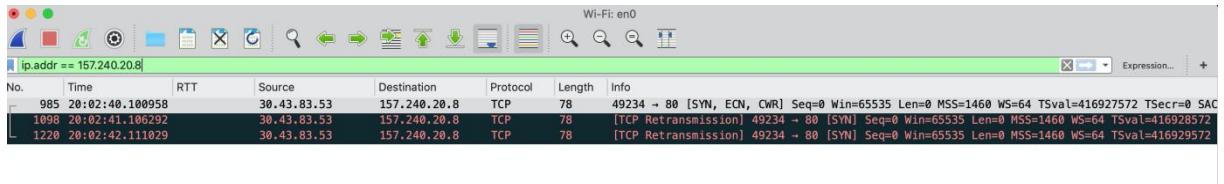


OCSP 校验超时

通过了解证书校验机制以后可以得知，这个问题有可能是慢在证书校验这一块。过滤了一下 DNS 报文，可以看到客户端确实有 OCSP Server 域名 `ocsp.int-x3.letsencrypt.org` 的 DNS 查询请求，该域名 CNAME 解析到了 `a771.dscq.akamai.net`，最终解析到的 IP 是 `157.240.20.8`



过滤了客户端跟 OCSP Server 端的交互包，发现客户端发起 TCP 三次握手，发送 SYN 包以后 Server 端未响应，客户端超时 1 秒以后继续发送 SYN 包，发送了三次均无响应，客户端证书校验失败，这个 3 秒刚好跟目前的耗时情况吻合。



通过一些探测网站，在国内探测了一下这个 OCSP Server 的地址确实表现也不是很好，大部分地区均无法访问，看来这个问题收到运营商链路问题的影响。

共178个点	12个	共44个	共22个独立IP	共22个独立节点	有非200状态	0.747s	0.104s	0.610s	0.000s	0.746s			791.500KB/s		
安徽芜湖市联通	联通	安徽	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
吉林白山市移动	移动	吉林	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
天津天津市联通	联通	天津	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
陕西榆林市电信	电信	陕西	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
澳大利亚国外	国外	澳大利亚	23.214.88.19	澳大利亚新南威尔士悉尼Akamai	200	0.076s	0.041s	0.058s	0.000260s	0.076s	0	0	0	查看	GET PING Trace Dig
河南周口市电信	电信	河南	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
四川成都市电信	电信	四川	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
重庆重庆市联通	联通	重庆	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
四川泸州市移动	移动	四川	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
甘肃天水市电信	电信	甘肃	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
山西太原市电信	电信	山西	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
浙江湖州市联通	联通	浙江	访问超时	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig
江苏徐州市联通	联通	江苏	无法连接到主机	*	0	*	*	*	*	*	*	*	*	查看	GET PING Trace Dig

问题发生的完整过程

通过以上排查分析，总结一下这个问题发生的完整过程：

客户端发起一次 HTTPS 请求，在 SSL 握手的过程中，CDN 服务端将证书信息发给了客户端，客户端根据证书信息，发起在线证书查询来校验证书合法性，但是由于证书校验服务地址不可达导致证书校验失败，客户端间隔一秒继续重试 2 次均失败，耗时 3 秒以后客户端停止校验，继续完成后续的 SSL 握手，造成最终访问慢的现象。

如何优化

OCSP 的原理决定了其存在隐私和性能问题：

- (1) 浏览器直接去请求第三方 CA(Certificate Authority, 数字证书认证机构), 会暴露网站的访客(CA 机构会知道哪些用户在访问我们的网站);
- (2) 浏览器进行 OCSP 查询会降低 HTTPS 性能(访问网站会变慢) OCSP 实时查询会增加客户端的性能开销, 本案例就是这种情况。

针对这种情况, OCSP Stapling 出现了。OCSP Stapling 可以将原本需要客户端实时发起的 OCSP 请求转嫁给服务端, Web 端将主动获取 OCSP 查询结果, 并随证书一起发送给客户端, 以此让客户端跳过自己去寻求验证的过程, 提高 TLS 握手效率, 从而提高 HTTPS 性能。阿里云 CDN 也支持 OCSP Stapling 功能, 可以由 CDN 服务器查询 OCSP 信息, 从而降低客户端验证请求延迟, 减少等待查询结果的响应时间。具体配置方法和原理介绍可以参考 [设置 OCSP Stapling](#)。

不过本案例中, 验证过开启 CDN 的 OCSP Stapling 功能也无法解决, 主要的问题是 OCSP Server 端在国内访问整体质量均不佳, 主要是由于跨境链路和运营商的问题造成的, 因此导致 CDN 去请求 OCSP 一样质量不佳。对于这种情况, 如果是直接访问的 Nginx, 可以通过一些方式生成 OCSP Stapling 文件部署到 Nginx 中 (这里不展开介绍)。如果是通过 CDN 等代理服务器方式, 由于不能单独配置 OCSP Stapling 文件, 最好的方式还是更换证书来解决。

使用阿里云 CDN 的情况下, 可以直接使用阿里云 CDN 的免费证书, 或者到阿里云 SSL 证书服务里去申请免费证书 or 购买付费证书, 详细请戳[这里](#)。

CDN 访问 508 问题

作者：胡夫

问题场景

用户使用阿里云 CDN 加速服务以后出现 508 错误。

The screenshot shows a network request in a browser's developer tools. The request URL is redacted. The method is POST, and the status code is 508. The response headers include:

- content-length: 0
- content-type: application/octet-stream
- date: Fri, 13 Dec 2019 15:01:05 GMT
- eagleid: 249ed89915762492648765095e, 249ed89915762492648765095e
- server: Tengine
- status: 508
- timing-allow-origin: *, *
- via: cache22.12et2[,0], cache26.12nu16-1[46,0], kunlun8.cn547[72,0], cache13.12et2[800,0], kunlun5.cn1568[1123,0]
- x-alicdn-da-ups-status: end0s,0,508

The Request Headers section shows:

- :method: POST
- :path: J0eXAiOjJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6I1ZJRSIsImVCI6MTU3NjgzMTQyNn0.ZVbiFMfyu2s7PSpwMCZ4xadKa_fZRJXkAlavdCMupuk
- :scheme: https

问题排查

1. 508 定义

我们先来看一下，Http_code 508 的定义：

The server terminated an operation because it encountered an infinite loop while processing a request with "Depth: infinity". This status indicates that the entire operation failed.

Source: RFC5842 Section 7.2

也就是说，服务器在处理请求时陷入死循环，此状态表示整个操作失败。

2. Response header 分析

从上面的 CDN 的 508 错误里，可以看到 CDN 返回的 Response Headers 里，x-alicdn-da-ups-status 响应了 508，eagleid 响应了两个值。通常情况下，一次通过 CDN 加速的请求，eagleid 只会记录一个值，用于唯一标识这次请求，像这种情况，eagleid 记录了多个值，说明这个请求在 CDN 里循环了。造成这个现象的可能原因一般有如下两种情况：

- CDN 的源站也是在阿里 CDN 上加速的，比如 CDN 的源站域名也是用了一个阿里 CDN 域名。
- CDN 的源站有一些代理的逻辑，会将 CDN 加速域名的请求代理到另外一个走阿里 CDN 加速的域名上。

这里请注意，因为数据的走向是：客户端-->CDN-->源站。如果源站也是走了 CDN 的话，就会变成：客户端-->CDN-->源站-->CDN-->源站...以此循环，这种情况就可能导致回环的情况，出现 508。CDN 产品本身有限制，要求 CDN 的源站不能是走阿里 CDN 加速，这是一种非标准化的操作。

另外还有一种定位方法，可以直接绑定 Host 到源站去测试，也就是不通过 CDN 访问，然后看返回的 Response headers 响应头，是否带有 CDN 的特殊 header 头，一般就是看 Via 头就可以了（阿里云 CDN 返回的 header 里会带有 Via 头），可以参考下图。如果直接访问源站的时候，源站返回的 header 头却有 CDN 特有的 header 头，那就说明源站有请求 CDN 的逻辑。

Headers Preview Response Cookies Timing

General

Request URL: [REDACTED]

Request Method: POST

Status Code: 508

Remote Address: [REDACTED]

Referrer Policy: no-referrer-when-downgrade

Response Headers

content-length: 0

content-type: application/octet-stream

date: Fri, 13 Dec 2019 15:01:05 GMT

eagleid: 249ed89915762492648765095e, 249ed89915762492648765095e

server: Tengine

status: 508

timing-allow-origin: *, *

via: cache22.12et2[,0], cache26.12nu16-1[46,0], kunlun8.cn547[72,0], cache13.12et2[800,0], kunlun5.cn1568[1123,0]

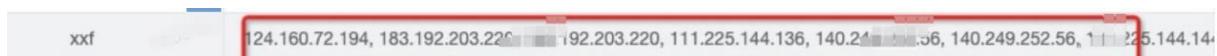
x-alicdn-da-ups-status: end0s,0,508

Request Headers

:method: POST

3. 日志分析

如果去查对应 508 的 sunwell 日志，在 xff 字段里可以看到有多个 cdn 的 ip，也可以说明该请求在 CDN 系统里回环了。



特别说明

CDN 添加域名的时候，CDN 控制台会做判断，如果用户填写的源站域名是阿里 CDN 加速域名的话，会直接报错。但是有两种情况目前 CDN 层面还无法避免，需要用户侧去修改：

- 用户配置加速域名 A 的源站域名是域名 B，此时域名 B 未走 CDN 加速，因此域名 A 的源站可以配置成功。但是配置成功以后，用户将域名 B 添加到 CDN 上做加速。

- 用户配置的源站不走 CDN，但是源站会做一些代理，例如方向代理，把 CDN 加速域名的请求代理到另外一个 CDN 加速域名，这种情况 CDN 也是无法提前监测到的。

适用于

CDN

DCDN

CDN 加速 OSS 后未响应 Content-MD5

作者：胡夫

问题描述

使用 CDN 加速 OSS 以后发现返回的响应头里没有带 Content-MD5 头了，而直接通过 OSS 域名访问是会返回 Content-MD5 头的。刷新 CDN 缓存以后部分节点会返回 Content-MD5，但是还是会有些部分节点不返回 Content-MD5。

```
< HTTP/1.1 200 OK
< Server: Tengine
< Content-Type: video/mp4
< Content-Length: 17369965
< Connection: keep-alive
< Date: Mon, 07 Jun 2021 11:33:58 GMT
< x-oss-request-id: 60BE042629F7AC37327E477F
< Vary: Origin
< x-oss-cdn-auth: success
< Accept-Ranges: bytes
< ETag: "D1EF74CEC4DE1F89F03A30C4647869F4"
< Last-Modified: Wed, 26 May 2021 04:10:19 GMT
< x-oss-object-type: Normal
< x-oss-hash-crc64ecma: 1958374103060741818
< x-oss-storage-class: Standard
< x-oss-version-id: null
< x-oss-server-time: 4
< Ali-Swift-Global-Savetime: 1623065639
< Via: cache25.l2cn3037[0,0,200-0,H], cache47.l2cn3037[1,0], cache47.l2cn3037[1,0], vcache39.cn2038[20,20,200-0,M], vcache33.cn2038[22,0]
< Age: 0
< X-Cache: MISS TCP_MISS dirm:-2:-2
```

```
< X-Swift-SaveTime: Mon, 07 Jun 2021 12:16:47 GMT
< X-Swift-CacheTime: 2592000
< Access-Control-Allow-Origin: *
< Timing-Allow-Origin: *
< EagleId: 73eec0b516230682076361320e
<
{ [2920 bytes data]
100 16.5M 100 16.5M    0      0   766k      0  0:00:22 0:00:22 --:--:-- 914k
* Connection #0 to host tutor-trans-video-online.fbcontent.cn left intact
```

排查过程

1. 排查 CDN 的日志

虽然 X-Cache 是 MISS，没有命中缓存回源的，但是通过以下的 Via 信息可以看到是命中 L2 的缓存了，L2 节点是 l2cn3037。

```
Via: cache25.l2cn3037[0,0,200-0,H], cache47.l2cn3037[1,0], cache47.l2cn3037[1,0], vcache39.cn2038[20,20,200-0,M], vcache33.cn2038[22,0]
```

通过 Ali-Swift-Global-Savetime: 1623065639 可以知道该文件缓存到 CDN 节点的时间，unix 时间戳 1623065639 转换成北京时间是 2021-06-07 19:33:59。于是查对应时间段 l2cn3037 这个节点的日志信息，发现该 L2 上记录的状态码都是 206，同时带了 http_range 字段，说明 L2 是 Range 回源 OSS 的，因此 OSS 响应了 206。

2. 查 OSS 日志

查对应时间段 OSS 的实时日志，发现 OSS 日志字段记录的 content_md5 为"-", 说明 OSS 确实没有响应 content_md5 字段。测试发现直接 range 请求 OSS，OSS 也是不会响应 Content-MD5 的，经确认 OSS 的策略的确如此。

3. 查 CDN 的 range 回源配置

确认 CDN 的 range 回源配置，发现配置的是 on，也就是开启了 range 回源。



Range回源设置	具体描述	示例
开启	<p>当您需要访问资源文件指定范围内的部分内容时，为了提高资源响应效率，则需要开启Range回源。开启Range请求回源后，源站需要依据Range响应文件的字节范围，同时CDN节点也会向客户端响应相应字节范围的内容。</p> <p>说明 如果客户端是Range请求，则回源也是Range请求。只有第一次请求是按照客户端请求里的Range大小回用户源站，后面的请求全部强制按照512 KB分片大小回用户源站。</p>	<p>如果客户端向CDN请求中含有 <code>range: 0-100</code>，则源站收到的请求中也会含有 <code>range:0-100</code>。源站响应CDN节点，CDN节点响应客户端字节范围为0~100，共101个字节。</p>
关闭	<p>当您需要访问资源文件的全部内容时，则需要关闭Range回源。关闭Range回源后，CDN上层节点会向源站请求全部的文件，由于客户端收到Range定义的字节后自动断开HTTP连接，请求的文件没有缓存到CDN节点上，最终导致缓存命中率较低，且回源流量较大。</p>	<p>如果客户端向CDN请求中含有 <code>range: 0-100</code>，则源站收到的请求中没有Range这个参数。源站响应CDN节点完整文件，CDN节点响应客户端的是101个字节，由于链接断开，会导致该文件没有缓存到CDN节点上。</p>
强制	<p>参数请求强制回源站。您选择Range回源为强制后，任何分片请求都会强制分片回源。无论客户端的请求是否是Range请求，都将按照512 KB的分片大小回用户源站，请确保源站支持参数Range。</p>	无

问题原因

整个过程是 19:33:59 有一个客户端发起了 range 请求，由于 CDN 域名配置了 range 回源，因此 CDN 会按照 512KB 的大小 range 回源 OSS，OSS 接到 range 请求以后响应状态码 206 并且不响应 Content-MD5，这份响应被 CDN 缓存下来。后续客户端请求到对应的 CDN 节点，不管是否是 range 请求，由于 CDN 已经有缓存，就会直接返回之前缓存的不带 Content-MD5 的 Response 信息。

PS：为什么当时刷新缓存以后通过 CDN 去访问可以看到有 Content-MD5？

答：因为当时刷新缓存以后，重新去访问的时候，客户端发起的不是 range 请求，而是普通的请求；与此同时，由于 CDN 上配置的 range 回源是"on",并不是"force"，也就是说不是强制 range 回源，因此 CDN 这种情况下回源的时候并没有发起 range 请求，OSS 是会响应 Content-MD5 的。

解决方案

OSS 会返回 CRC，如果要做文件一致性校验的话可以通过 x-oss-hash-crc64ecma 字段获取 CRC 来做校验。

适用于

CDN

DCDN

对象存储 OSS

源站不支持 range 导致请求 CDN 异常断开

作者：胡夫

问题描述

通过 CDN 访问资源异常断开，无法读取完整的数据。

问题排查

1. 日志分析

L2 的日志分析看，接入层 access 日志显示后端主动断开连接，由于接入层的后端是缓存模块 swift，即是 swift 断开了连接。而 swift 的后端回源模块没有断开连接，由此可见这个请求就是 L2 的 swift 断开的。这里还有一个奇怪的地方，access 日志的状态码是 206，而回源模块的状态码是 200。进一步发现，实际这个请求发起的是一个 range 请求，范围 bytes=1572864-2097151，但是源站却没有按照标准响应 206 状态码，而是响应了 200 状态码，导致 L2 一直在向源站读数据。

2. 为什么会发起 range 请求

由于用户的需求是需要读取整个文件，查询 L1 的日志发现客户端确实是没有发起 range 请求的，但是 L2 回源确发起了 range 请求。查询域名配置发现域名开启了强制 range 回源。

问题原因

整个过程如下：

1. 客户端发起一个完整读取文件的请求，由于域名开了强制分片回源，L2 在回源的时候发起了 range 请求，range 请求 bytes=1572864-2097151
2. L2 发起 range 请求回源，但是源站响应了 200，而不是 206，这就会导致 L2 会读取完整的整个文件的数据，而不仅仅是 range 请求的数据。
3. 由于这个文件非常大，导致 L2 一直在读数据中，L2 的 swift 由于已经读完了 range 请求的数据，但是还在读后面的数据，但发现来自 L1 的 range 请求数据已经有缓存了，就直接返回了缓存数据，同时也断开了连接。

解决方案

源站优化，针对 range 请求需要响应 206 状态码，而不是 200 状态码。如果源站对 range 请求无法正确响应，那么 CDN 只能关闭 range 请求，但是关闭 range 请求的情况下对于大文件的访问效果和命中效果不佳，容易造成比较大的回源流量。

适用于

CDN

DCDN

CDN 回源带宽突增

作者：胡夫

问题描述

CDN 控制台显示 05-24 日 CDN 回源带宽突增，同时访问有异常，有很多 5xx 错误。



排查过程

1. 监控查询

控制台查看监控信息，发现一周内同比对比，访问带宽并没有突增，跟周围几天的业务量保持一致，说明业务侧并没有明显的上量，但是回源带宽在异常时间确实有一个突增，而且命中率突降。

2. 日志查询

下载对应时间段 CDN 的日志，发现有大量的 httpcode 为 206 的 range 日志，请求的是 zip 后缀的大文件。分析日志查询前 10 的 clientip，例如 top1 的 clientIP 在 1 小时日志里有上万条请求，都是 range 请求 zip。

```
cat log_file | awk '{print $3}' | sort|uniq -c|sort -nr |head -10
```

```
sh-3.2# cat /.../jje/U...nds/fs.smyfinancial.com_2021_05_24_100000_110000 | awk '{print $3}' | sort|uniq -c|sort -nr |head -10
11096 61.190.124.200
1617 119.54.154.149
1517 117.136.77.130
957 39.162.245.145
953 211.94.195.73
934 60.25.216.150
868 123.123.220.75
797 221.212.23.6
786 117.136.50.73
720 223.104.212.244
5.24日10:00~11:00之间这个IP就发了1万多个请求，基本上都是range请求这个zip文件
sh-3.2# cat /.../jje/U...nds/fs.smyfinancial.com_2021_05_24_100000_110000 | awk '{print $3}' | sort|uniq -c|sort -nr |head -10
11096 61.190.124.200
1617 119.54.154.149
1517 117.136.77.130
957 39.162.245.145
953 211.94.195.73
934 60.25.216.150
868 123.123.220.75
797 221.212.23.6
786 117.136.50.73
720 223.104.212.244
```

3. 查询配置

查询 range 配置，发现 CDN 上并没有开启 range 回源。同时直接 range 请求源站，发现源站并不支持 range 请求。由此问题基本明确是 range 原因产生。

The screenshot shows a configuration interface with a sidebar and several tabs. The sidebar includes '功能概述', '批量复制', '设置报警', '标签管理', '基本配置', '回源配置', '缓存配置', 'HTTPS配置', '访问控制', '性能优化', '视频相关', '概述', and two highlighted sections: '配置Range回源' and '配置拖拽播放'. The main content area has three tabs: '开启' (Enabled), '关闭' (Disabled), and '强制' (Forced). The '开启' tab contains a note about Range requests. The '关闭' tab contains a note about Range requests and their impact on cache命中率. The '强制' tab contains a note about Range requests and their impact on source requests. A red arrow points from the text '5.24日10:00~11:00之间这个IP就发了1万多个请求，基本上都是range请求这个zip文件' to the '开启' tab.

功能	描述
开启	<p>当您需要访问资源文件指定范围内的部分内容时，为了提高资源响应效率，则需要开启Range回源。开启Range请求回源后，源站需要依据Range响应文件的字节范围，同时CDN节点也会向客户端响应相应字节范围的内容。</p> <p>说明 如果客户端是Range请求，则回源也是Range请求。只有第一次请求是按照客户端请求里的Range大小回用户源站，后面的请求全部强制按照512 KB分片大小回用户源站。</p>
关闭	<p>当您需要访问资源文件的全部内容时，则需要关闭Range回源。关闭Range回源后，CDN上层节点会向源站请求全部的文件，由于客户端收到Range定义的字节后自动断开HTTP连接，请求的文件没有缓存到CDN节点上，最终导致缓存命中率较低，且回源流量较大。</p>
强制	<p>参数请求强制回源站。您选择Range回源为强制后，任何分片请求都会强制分片回源。无论客户端的请求是否是Range请求，都将按照512 KB的分片大小回用户源站，请确保源站支持参数Range。</p>
无	

问题原因

在 5.24 日上午的时候，07:00~11:00 期间，有一些客户端在请求一些大文件，比如类似日志里分析的 zip。由于文件比较大，客户端发的是 range 请求，range 的形式分片去请求。而这个文件由于在 CDN 上没有缓存，因此 CDN 需要去回源。另外由于 CDN 上没有配置开启 range 回源，因此虽然客户端请求的时候带了 range，但是 CDN 回源请求源站的时候是不带 range 的，CDN 是向源站请求完整的数据然后返回给客户端。但是由于客户端拿完他该拿的数据部分（range 的部分）就断开了，客户端的断开导致 CDN 跟源站的连接也断开了，这种情况下这个文件并没有缓存到 CDN 上。然后客户端继续发 range 请求，CDN 继续做同样的动作，因为一直缓存不住，而客户端又一直在 range 请求，CDN 就会一直回源，造成回源带宽增加，源站的压力增大，产生了一些 504。

解决方案

这种情况建议 CDN 上开启 range 回源，这样 CDN 也会 range 的形式请求源站，并且把 range 到的部分缓存到 CDN 上，不过前提是需要源站支持 range 请求。不过现在直接测试源站，发 range 请求，源站返回的是完整部分，因此源站不支持 range 请求，这样即使 CDN range 回源也达不到效果，因此需要源站开启 range 功能，然后 CDN 开启 range 回源来优化这个场景。

适用产品

CDN

全站加速

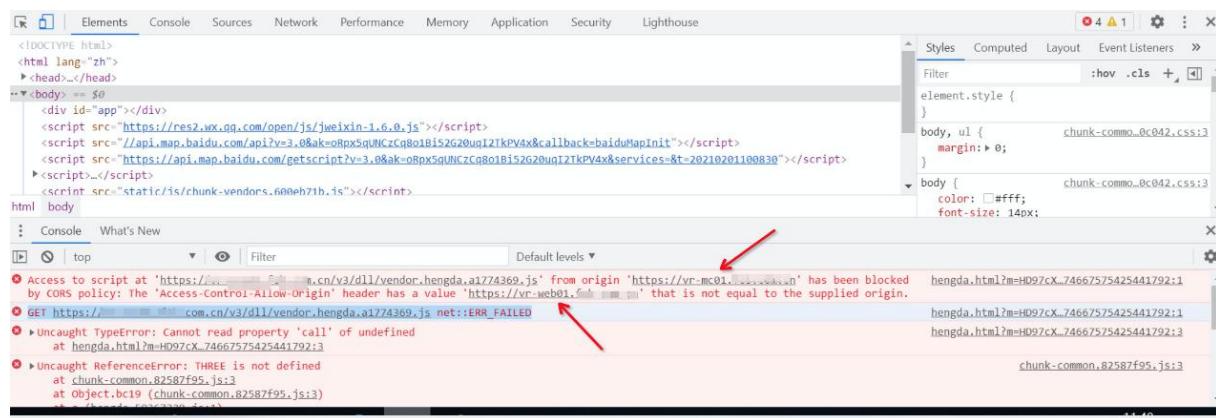
跨域问题-The 'Access-Control-Allow-Origin' header has a value 'xxx' that is not equal to the supplied origin

作者：胡夫

问题描述

请求 CDN 加速的 URL 出现跨域报错

The 'Access-Control-Allow-Origin' header has a value 'xxx' that is not equal to the supplied origin



问题分析

从问题看，CDN 响应的跨域头 Access-Control-Allow-Origin 的 Value 值，跟客户端请求的 Origin 跨域头不一致，因此浏览器 Block 这个请求了。例如，请求跨域头为'Origin:http://域名 A'，但是响应跨域头为'Access-Control-Allow-Origin:http://域名 B'

这个问题有以下几种情况：

1. CDN 确实响应错了

这种情况需要修改 CDN 的跨域头配置，配置成和客户端请求的 Origin 一致。但是目前 CDN 默认只能配置一个跨域头，如果实际业务里存在多个 Origin 的情况下，CDN 的跨域 Value 可以配置成“*”，或者通过后台 PE 修改配置为按照客户端请求的 Origin 去分发匹配的跨域头。

2. 源站配置了跨域头被 CDN 缓存了

第一次客户端请求的时候，请求跨域头为'Origin:http://域名 A'，CDN 没有命中缓存，按照规则回源，由于源站配置了跨域头，响应了'Access-Control-Allow-Origin:http://域名 A'；第二次客户端请求的时候，请求跨域头为'Origin:http://域名 B'，但是由于 CDN 之前已经缓存了，直接返回了缓存数据，而缓存数据里跨域响应头为'Access-Control-Allow-Origin:http://域名 A'，这就导致响应的 Value 值和客户端不匹配，导致被浏览器 Block。针对这种情况有两种解法：

方案一

源站不用配置跨域头，直接在 CDN 上配置跨域头即可。然后刷新下 CDN 的缓存，把历史缓存都清除掉。

方案二

CDN 上配置 Access-Control-Allow-Origin 的时候选择“不允许重复”，这种情况下当源站响应跨域头时，CDN 会去掉源站的响应头，按照 CDN 的跨域响应头规则响应给客户端。

自定义HTTP响应头



响应头操作

增加



自定义响应头参数

Access-Control-Allow-Origin



响应头值

请输入响应头值

是否允许重复

不允许



确定

取消

3. 浏览器缓存

第一次客户端请求的时候，请求跨域头为'Origin:http://域名 A'，CDN 响应了'Access-Control-Allow-Origin:http://域名 A'；第二次客户端请求的时候，请求跨域头为'Origin:http://域名 B'，但是由于浏览器缓存了上一次请求的结果，直接返回了缓存数据，而缓存数据里跨域响应头为'Access-Control-Allow-Origin:http://域名 A'，这就导致响应的 Value 值和客户端不匹配，导致被浏览器 Block。

这种情况需要强制刷新浏览器的缓存，另外为了避免这种情况的发生，建议在 CDN 上配置 Cache-Control 为 no-cache，强制浏览器不缓存。

413-GET 请求不支持携带 body

作者：胡夫

问题描述

请求 CDN 图片时返回 httpcode=413 的问题，浏览器请求正常，但是 App 端 GET 请求会报错 413。

问题排查

测试发现客户端发起的 GET 请求里是带了 body 的，resp headers 返回了 x-swift-error:invalid request 表示请求不合法。

```
* Using Stream ID: 1 (easy handle 0x5617d932b740)
} [5 bytes data]
> GET /smartapp/article/1526372_1618889032843/webp_360.jpg HTTP/2
> Host: [REDACTED]
> User-Agent: curl/7.61.1
> Accept: /*
> Content-Length: 3
> Content-Type: application/x-www-form-urlencoded
>
{ [5 bytes data]
* Connection state changed (MAX_CONCURRENT_STREAMS == 128)!
} [5 bytes data]
* We are completely uploaded and fine ←
{ [5 bytes data]
< HTTP/2 413
< server: Tengine
< content-type: image/jpeg
< content-length: 0
< x-swift-error: invalid request
< via: Kunlun10.cn1956[0,0,0-0,M], Kunlun6.cn1956[1,0]
< access-control
< timing-allow
< eagleid: 71e5fc9a16189955905386223e
<
{ [0 bytes data]
100      3      0      0    100      3      0      20  --:--:-- --:--:-- --:--:--  20
* Connection #0 to host [REDACTED] left intact
```

问题原因

GET 请求如果携带 body 会返回 413 情况，因为 swift 不支持这样请求。

解决方案

修改 App 请求图片的逻辑，GET 请求的时候不要带 body 体。

适用于

CDN

通过阿里云 CDN 系列产品访问与直接访问源站得到的结果不一样

作者：胡夫

问题描述

用了 CDN 加速以后，请求资源得到的结果和不通过 CDN 加速直接访问源站得到的结果不一致。

问题原因

当客户端请求 CDN 节点以后，如果没有命中缓存，CDN 会回源请求源站。在请求源站的时候，CDN 除了会透传客户端的请求头，还会加上一些 CDN 特有的一些请求头去请求源站，比如类似如下 的请求头。

```
Via: cn2836.l1, vcache10.cn2836, l2cn2635.l2, cache28.l2cn2635
Eagleeye-Traceid: 24689aa416179799458162753e
Ali-Swift-Log-Host: test.xxx.cn
Ali-Swift-Stat-Host: level2.test.xxx.cn
X-Forwarded-For: 58.101.142.41
X-Client-Scheme: http
Ali-Cdn-Real-Ip: 58.101.142.41
Ali-Swift-5Xx-No-Retry: on
Cdn-Src-Ip: 127.0.0.1
Ali-Swift-Range-Cache: on
```

其中某些头是特定功能有关，比如 Ali-Swift-Range-Cache 是关于 Range 回源的，如果 CDN 上开了 Range 回源，就会加上这个头。还有一些头是固定都会有的，比如 Via 头是表示的代理服务器，记录的字段主要表示这个请求经过的 CDN 节点，Ali-Cdn-Real-Ip 记录的是客户端真实 IP，X-Forwarded-For 字段就是标准的 HTTP XFF 字段。具体这些请求头，在源站转包分析就能看到来自 CDN

的请求头到底有哪些。

某些情况下，源站对于这里的一些请求头，有一些不一样的表现，大部分请求下都是因为 Via 这个请求头导致的。源站接受到一个请求的时候，对于请求头是否带了 Via 头会有不一样的表现，因为有些源站的配置会判断是否有 Via 头以此来判断该请求是否来自代理服务器，然后做出不一样的响应。

如何验证

如何验证这个问题是跟 CDN 的 Via 请求头有关，或者跟另外的请求头有关，这个可以通过在客户端构造 HTTP 请求，带上 Via 头去请求源站，看源站的响应表现。例如使用 curl 工具绑定源站，并带 Via 请求头的测试命令如下

```
curl -voa 'http://xxxx' -x <源站 IP>:80 -H 'Via:xxx'
```

如果带了 Via 和不带 Via 的请求话，源站服务器的表现不一样，即可定位是源站对于 Via 字段有不一样的表现，因此就可以解释为什么通过 CDN 以后会有不一样的表现。

当然，如果不是 Via 字段引起的，也可以带上其他 CDN 的特有请求头继续追踪验证，看具体是因为哪个请求头引起的。

如何解决

如果定位到是由于 CDN 的某一个特定的头导致源站有不一样的表现，可以参考以下两种解决方案去处理：

- (1) 源站去检查服务器配置，更改源站配置。
- (2) 目前 CDN 已经支持删除回源请求头，可以直接到 CDN 控制台去配置删除相关的回源请求头。

目前 DCDN 暂时还未支持用户自定义配置回源 HTTP 请求头，需要提交工单后台去配置。



适用于

CDN 系列产品，包括 CDN、DCDN、SCDN

DNS 劫持导致访问目标地址失败并且解析主机后的 IP 不是 CDN 节点 IP

作者：胡夫

问题描述

用户使用 CDN 下载资源失败，并且通过 DNS 解析主机后得到的 IP 地址也不是 CDN 节点的 IP 地址。

排查过程

DNS 劫持是网络上一种干扰用户正常访问服务的行为。一般情况下，DNS 劫持的方式分为攻击 DNS 服务器和伪造 DNS 服务器两种。DNS 劫持会导致以下两种情况发生：

- 将目标网站域名解析到错误的 IP 地址从而实现用户无法访问目标网站。
- 蓄意或恶意要求用户访问指定网站。

当使用 CDN 下载资源失败时，可以参考以下内容确认是否发生了 DNS 劫持：

1. 使用基调听云工具进行测拨，当用户侧访问目标网站后，发现部分通过 DNS 解析主机后的 IP 地址不是 CDN 节点的 IP 地址。

时间	错误代码	接入方式	监测城市	监测运营商	监测点ID	监测点IP	DNS服务器	操作系统	浏览器	主机IP	主机城市	主机运营商	平均返回(ms)	最大返回(ms)	最小返回(ms)
2021-01-23 16:08:20	-	LastMile	乌鲁木齐市	中国联通	8185203	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 11	20[REDACTED].99	北京市	中国电信	67	78	65
2021-01-23 16:08:02	-	LastMile	乌鲁木齐市	中国联通	8011560	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 11	20[REDACTED].99	北京市	中国电信	45	46	45
2021-01-23 16:07:55	-	LastMile	乌鲁木齐市	中国移动	7794040	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 8	20[REDACTED].99	北京市	中国移动	61	65	60
2021-01-23 16:07:55	-	LastMile	乌鲁木齐市	中国联通	8154906	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 9	20[REDACTED].99	北京市	中国电信	53	54	52
2021-01-23 16:07:51	-	LastMile	乌鲁木齐市	中国联通	7398849	[REDACTED]	[REDACTED]	Windows 10	Internet Explorer 11	[REDACTED]	唐山市	中国联通	52	56	52
2021-01-23 16:06:50	Completely Lost Package	LastMile	乌鲁木齐市	中国联通	8310628	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 11	[REDACTED]	北京市	中国电信	0	0	2000
2021-01-23 16:06:50	Completely Lost Package	LastMile	乌鲁木齐市	中国联通	8338430	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 11	[REDACTED]	北京市	中国电信	0	0	2000
2021-01-23 16:06:49	Completely Lost Package	LastMile	乌鲁木齐市	中国移动	8337840	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 11	[REDACTED]	北京市	中国电信	0	0	2000
2021-01-23 16:07:47	-	LastMile	乌鲁木齐市	中国电信	7814125	[REDACTED]	[REDACTED]	Windows 10	Internet Explorer 11	[REDACTED]	西安市	中国电信	46	46	46
2021-01-23 16:07:47	-	LastMile	乌鲁木齐市	中国移动	7275585	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 11	[REDACTED]	咸阳市	中国移动	43	47	42
2021-01-23 16:07:47	Completely Lost Package	LastMile	乌鲁木齐市	中国移动	8367008	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 11	[REDACTED]	北京市	中国电信	0	0	2000
2021-01-23 16:07:44	-	LastMile	乌鲁木齐市	中国电信	8277153	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 11	[REDACTED]	西安市	中国电信	45	48	44
2021-01-23 16:07:41	-	LastMile	乌鲁木齐市	中国电信	8011772	[REDACTED]	[REDACTED]	Windows 7 sp1	Internet Explorer 11	[REDACTED]	西安市	中国电信	48	51	46

2. 在用户侧使用第三方探测工具，例如，dig 工具测试用户 DNS 的工作状态，以下是 dig 测试访问目标网站后的返回图。可以发现在 ANSWER 区域中，本次解析仅有 A 解析，并没有 CNAME 解析的相关信息，因此判断发生了 DNS 劫持。

```
; [HEADER]
;     qid : 57024
;     qr : RESPONSE(1)
;     opcode : QUERY(0)
;     aa : NON-AUTHORITATIVE(0)
;     tc : NOT-TRUNCATED(0)
;     rd : RECURSION-DESIRED(1)
;     ra : RECURSION-ALLOWED(1)
;     rcode : NOERROR(0)

; [QUESTION:1]
group.oss.baticq.com. IN A

; [ANSWER:1]
group.oss.baticq.com. 3561 IN A [REDACTED]
```

解决方案

收集下列信息并向运营商获取协助处理：

- 具体的用户信息
- CDN 域名
- 故障发生区域和运营商
- 客户端出口 IP 和出口 DNS
- 说明：关于获取客户端出口 IP 和出口 DNS，可参考更多信息。
- 被劫持的 IP 地址

更多信息

关于如何获取客户端出口 IP 和出口 DNS，可根据您的客户端操作系统参考以下内容。

Linux 系统

1. 确认已安装 curl 工具，然后执行以下命令，获取出口 IP 地址。

```
curl icanhazip.com
```

2. 执行以下命令，获取出口 DNS 信息。

```
cat /etc/resolv.conf
```

系统返回类似如下。

```
[root@... ~]# cat /etc/resolv.conf
options timeout:2 attempts:3 rotate single-request-reopen
; generated by /usr/sbin/dhclient-script
nameserver 192.168.1.136
nameserver 192.168.1.88
```

Windows 系统

进入网页浏览器，在地址栏输入以下内容然后访问网页。

```
http://nstool.netease.com/
```

浏览器返回类似如下。

您好，尊敬的网易用户

您的IP地址信息: [REDACTED] 浙江省杭州市阿里云/电信/联通/移动/铁通/教育网

您的DNS地址信息: [REDACTED] 浙江省杭州市阿里云/电信/联通/移动/铁通/教育网

您的DNS设置正确

适用于

CDN

劫持问题导致通过 CDN 请求资源没有返回实际资源文件

作者：胡夫

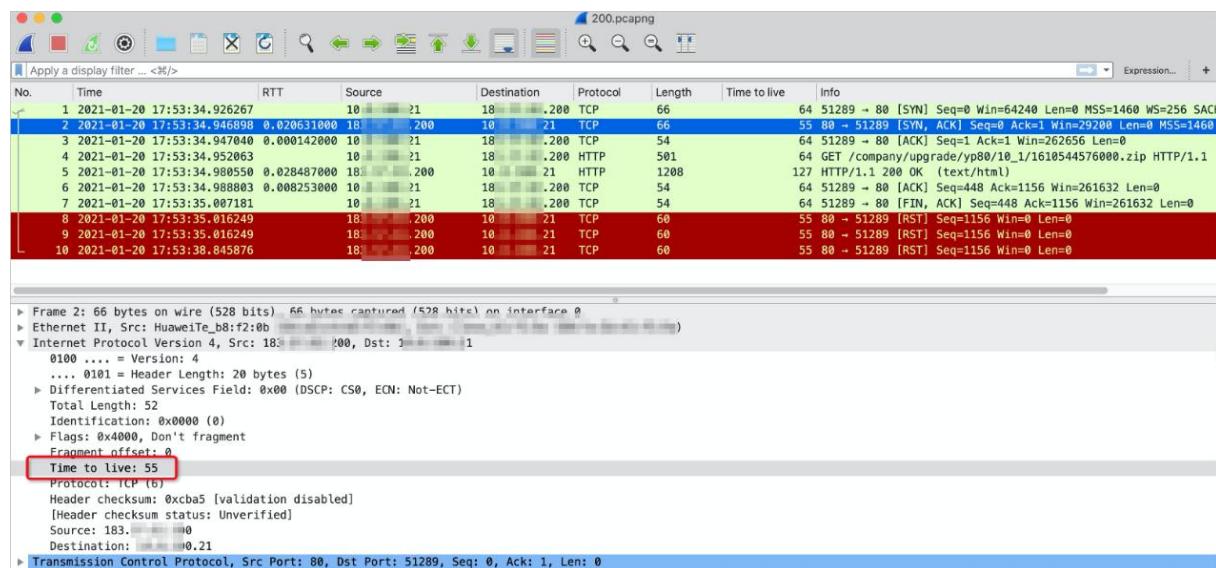
问题描述

通过 CDN 请求网站资源，有时会没有返回实际的资源文件，而是返回其他类型的资源，例如 text/html 类型的内容。

排查过程

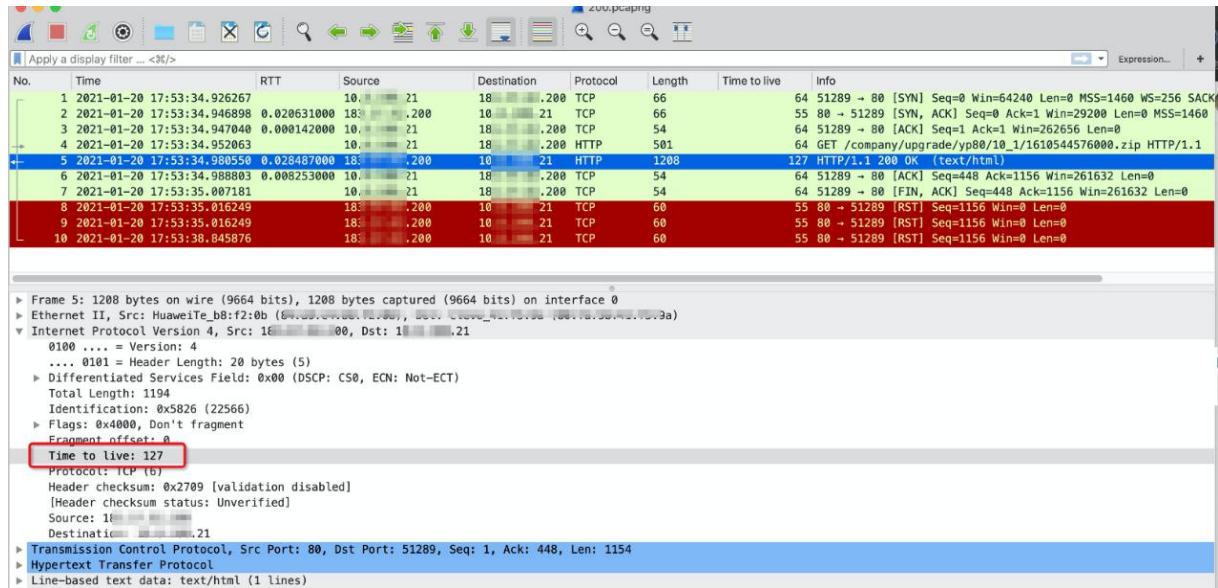
从问题描述看，可以初步判断是由于网站资源被劫持导致，您可以按照以下示例抓包进行确认：

1. 使用 Wireshark 抓取请求资源时的数据包，详情请参见网络异常时如何抓取数据包。
2. 2 号包是 TCP 三次握手的第 2 次握手包，单击第 2 次握手包，确认 Time to live (TTL) 值在正常的范围。

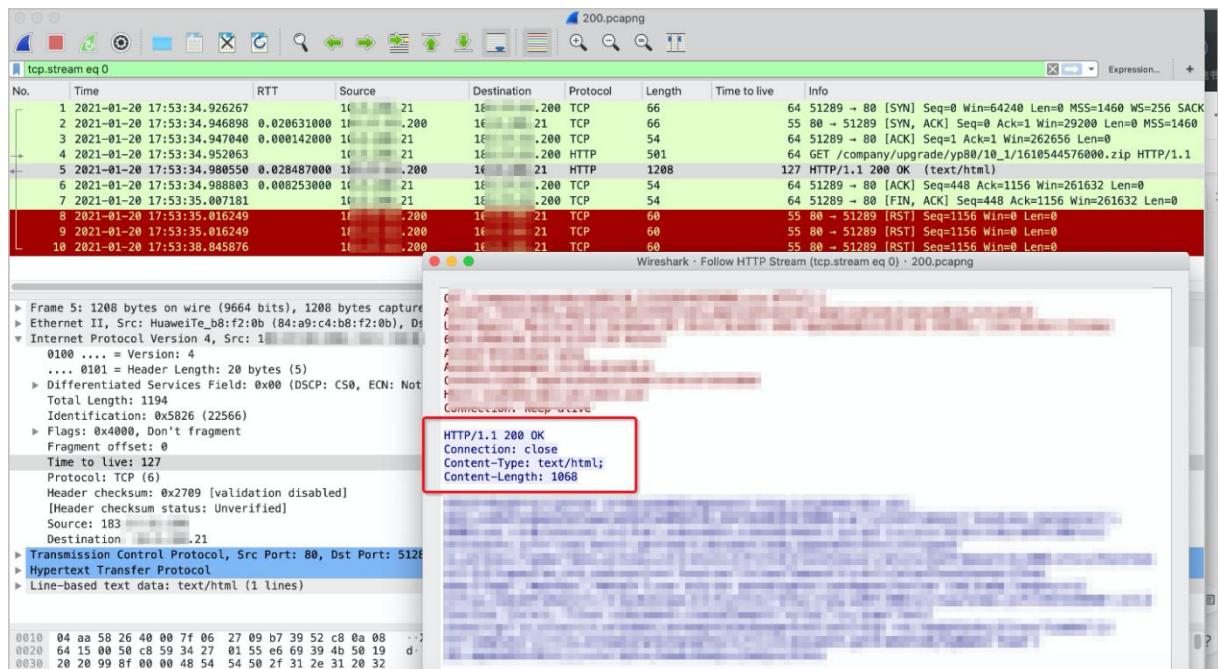


3. 5 号包是服务端响应 HTTP 请求的报文，TTL 变为 127，和上一步查看的 TTL 值不一样，说明为异常情况。

说明：正常情况下 Linux 系统实例的 TTL 都是小于 64。



在 Wireshark 的 Follow HTTP Stream 中可以看到响应的 Response Header 明显不正常，没有 CDN 特有的响应头，例如 Via、Eagled，说明这个请求不是从 CDN 节点响应的。



4. 1 到 3 号包显示客户端跟 CDN 节点是正常 TCP 握手报文，4 号包显示在客户端握手以后正常发起 HTTP 请求，但是请求被劫持，由中间网络的其他链路节点响应了这个 HTTP 请求，6 号包显示客户端以为收到了正常的响应，7 号包正常发起 FIN 包，实际这个 FIN 包到了 CDN 节点，8 到 10 号包可以看到报文的 TTL 是正常的，说明 CDN 节点响应了 RST 包。整个过程中，CDN 节点始终都没有真正的收到来自客户端的 HTTP 请求。

解决方案

- 建议使用 HTTPS 访问资源，提高访问安全，可以有效防止此类内容劫持。如何配置 CDN 的 HTTPS 访问资源，详情请参见配置 HTTPS 证书。
- 如果无法解决问题，可能是由以下两个原因导致，此时需要获取运营商协助。
 - CDN 节点被运营商封了，或者节点本身有问题。
 - 用户域名被运营商封了。

适用于

CDN

TCP 劫持导致某地移动 CDN 节点 HTTPS 访问失败

作者：胡夫

问题描述

在使用阿里云 CDN 服务时，在业务侧做的监控报警显示某地移动业务不可用。根据提供的 URL 和 CDN 节点 Vip 测试发现，会出现 SSL 握手失败。

```
sh-3.2# curl -vvo https://img.████.cn/v2/image/yz_fc.ico --resolve img.████.cn:443:120.████.████.224
* Added img.████.cn:443:120.████.████.224 to DNS cache
* Hostname img.████.cn was found in DNS cache
* Trying 120.████.████.224:443...
* TCP_NODELAY set
  % Total    % Received % Xferd  Average Speed   Time     Time      Current
               Dload  Upload   Total Spent  Left  Speed
  0          0        0       0      0   0 --:--:-- --:--:-- --:--:--   0* Connected to img.████.cn (120.████.████.224) port 443 (#0)
* ALPN, offering http/1.1
* Server aborted the SSL handshake
  0          0        0       0      0   0 --:--:-- --:--:-- --:--:--   0
* Closing connection 0
curl: (35) Server aborted the SSL handshake
```

解决方案

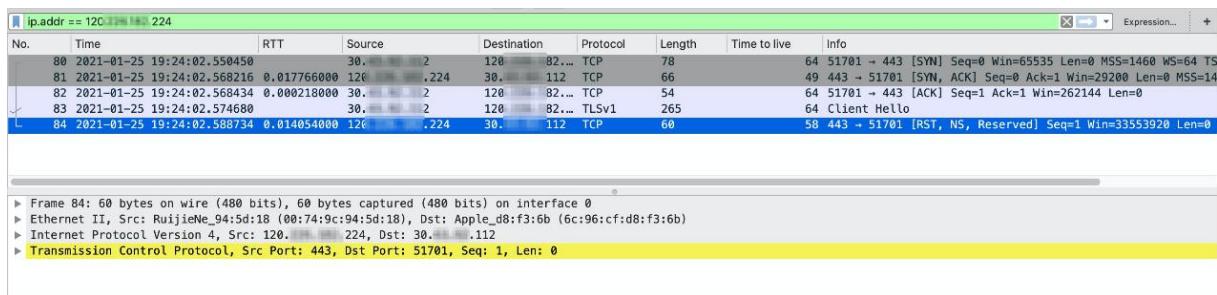
基础分析

进行问题的基础分析，发现以下几个现象：

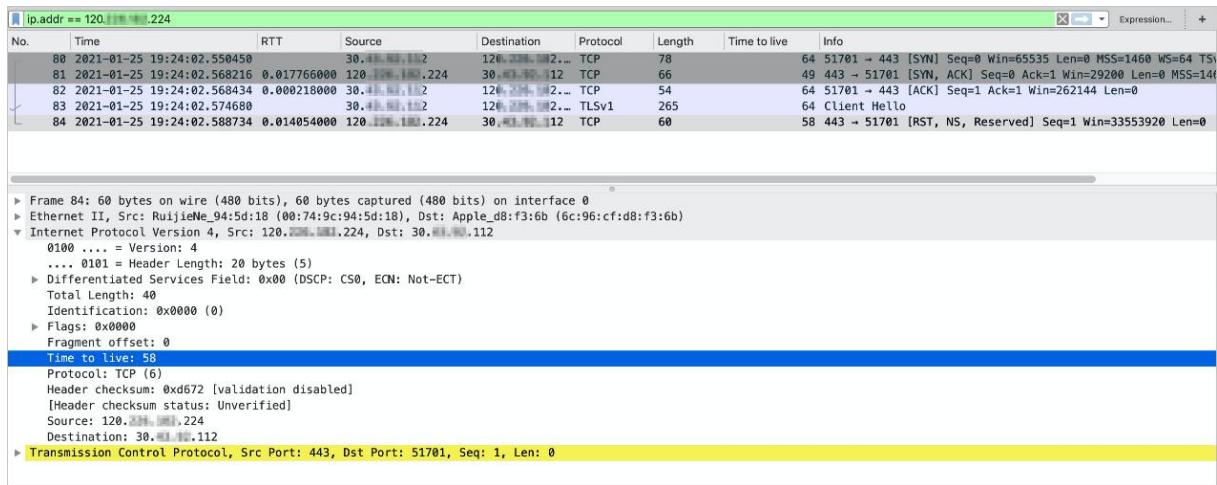
- 在节点系统中探测该 URL，发现其他 CDN 节点均正常，只有探测到这个某地移动的 CDN 节点时有异常。
- 如果用 HTTP 请求到这个 CDN 节点，则是正常的，只有 HTTPS 访问有问题。
- 基于以上两点分析，推测该 CDN 节点可能是历史不支持 SNI 的老节点。于是找到其他的 CDN 域名绑定到该节点，用 HTTPS 访问，却是正常的，因此 SNI 的问题也可排除。

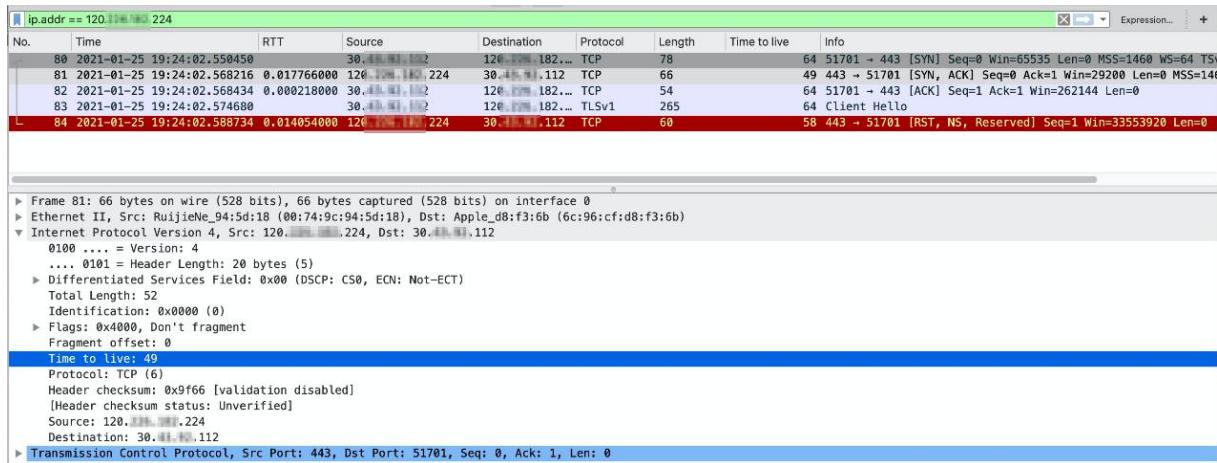
抓包分析

由于绑定节点能直接复现问题，因此直接在客户端侧进行抓包，通过 ip.addr 过滤该 Vip 的报文如下：



80、81、82 报文客户端和服务端完成了 TCP 三次握手，接下来 83 号报文，客户端发起了 SSL 的握手请求，但是 84 号报文服务端直接发了 RST。由于 CDN 在正常情况下不可能出现这种 RST 包，已经可推断是劫持问题。进一步查看这个 RST 报文（84），发现 TTL（Time to live）的值为 58，Identification 的值为 0；而查看 TCP 三次握手服务端的报文（81），显示 TTL 的值为 49，Identification 的值为 0。





判断依据

通常情况下，链路劫持的判断依据会从以下几个方面进行判断分析：

- TTL：表现为 TCP 报的 TTL 不一致甚至抖动很大。
- 一种情况：是跟正常包的 TTL 相差明显。
- 另一种情况：是通过 TTL 来判断操作系统类型，进而间接判断数据包是否有异常。
- Identification：出现不符合 RFC 标准的情况。对于给定地址和协议的 IP 包，它的 Identification 应该是公差为 1 的单调递增数列。每一个 IP 封包都有一个 16 位的唯一识别码。当程序产生的数据要通过网络传送时，都会被拆散成封包形式发送，当封包要进行重组的时候，这个 ID 就是依据。标识字段唯一地标识主机发送的每一份数据报，通常每发送一份消息它的值就会加 1。

说明：其中关于 Identification 的描述，可以参见 RFC 文档。

以下截图是选自 RFC 文档对于 Identification 的描述。

1 . Introduction

In IPv4, the **Identification** (ID) field is a 16-bit value that is unique for every datagram for a given source address, destination address, and protocol, such that it does not repeat within the maximum datagram lifetime (MDL) [[RFC791](#)] [[RFC1122](#)]. As currently specified, all datagrams between a source and destination of a given protocol must have unique IPv4 ID values over a period of this MDL, which is typically interpreted as two minutes and is related to the recommended reassembly timeout [[RFC1122](#)]. This uniqueness is currently specified as for all datagrams, regardless of fragmentation settings.

- Banner 信息：表现为 Response Header 中响应的 Server 字段不对，例如 CDN 是 Tengine，如果显示的响应头为 Tomcat 或者其他，那么很明显这个请求不是 CDN 响应。

问题分析总结

根据抓包分析，可以发现以下几个点：

- TCP 握手包、服务端响应包的 TTL 为 49，而 RST 包的 TTL 为 58，这个抖动很大。正常情况 TCP 握手成功以后，在 TCP 断开前，这条链路应该是稳定的，TTL 基本会维持固定的值，不会马上出现这么大的变动。
- Identification 明显不符合 RFC 标准。如果 81 和 84 号包都是同一个 CDN 节点响应的，那么 84 号包应该有一个具体的 Identification Value 值，而不是和 81 号包保持一致，都是 0。

综上所述，基本上可以判断是运营商劫持行为，且是针对这个域名做的劫持，CDN 作为一个服务端来说确实无法处理该类问题。因此，提供相关的抓包证明，给相关运营商报故障处理。整个 HTTPS 的请求过程如下：

- 客户端发起一次 HTTPS 请求，首先客户端和服务端完成了 TCP 握手，而这个 TCP 握手确实是和 CDN 节点握手的。
说明：一般情况下握手的包不会被劫持。
- 握手成功以后客户端发起 SSL 握手请求，这个请求最终却没有到达真正的 CDN 节点，而是被劫持到中间某一个路由节点给直接 RST。

更多信息

如果要确定一个请求到底是否被劫持，最好的方案是在客户端发起请求以后，客户端和服务端同时双向抓包确认。但是这往往需要有相关权限的工程师登录 CDN 节点进行抓包，操作起来有一定难度。有时候可以通过客户端抓包，看到明显的异常 TTL、Identification 信息进行推断，也能基本定位。

适用于

CDN

预热文件卡住-海外 L2 回源国内源站跨境链路

作者：苍柏

问题描述

用户 2021-01-09 16:40:48 +0800 CST 提交了一个 zip 文件的预热，完成时间是 2021-01-09 16:40:57 +0800 CST，用户源站 100M 带宽，18:11:31 反馈卡在 78.57%，一小时以后查看，进度还是卡在 78.57%。

排查过程

1. 一般原因

预热卡住一般是因为某一个或者某几个节点，从客户源站下载数据时候速度较慢，或者超时等导致。

2. 国内源站

查看用户的源站是国内的源站。海外的 L2 节点请求国内的源站，走跨境链路，可能会受到跨境链路影响，这部分问题 CDN 层面也很难处理。

解决方案

预热可以支持国内和海外的，建议分开预热，先预热国内节点，再预热海外的。控制台目前没有开放分区域预热的功能，目前可以使用 API 或者脚本的方式预热，参考官网 API 预热接口 Push ObjectCache 文档。

设置了缓存规则但每次访问还是 MISS

作者：苍柏

CDN 上配置了缓存规则但每次访问还是 MISS 的原因说明。

问题描述

客户在 CDN 上配置了缓存规则，且客户源站并未响应 no-cache 的 cache-control，但每次访问之后仍然是 MISS，无法实现 HIT。需要排查具体原因。

原因排查

1. 确认问题现象

访问 URL: <http://xxxx.test.com/> 查看浏览器开发者工具下的 response Header，确认到无论是 L1 还是 L2 都是 MISS，且无论 F5 刷新多少次页面，每次都还是 MISS，但比较奇怪的是：x-swift-cachetime 并非 0，而是一个很长的数字。

2. 确认控制台上的的缓存配置是否有特殊性

检查控制台配置，发现客户配置了针对根目录下的 7776000 的缓存时长配置，和 x-swift-cache-time 时长一致，回源请求头以及 ER、ES 等均不存在配置。

3. 测试客户源站发现问题

绑定客户源站，发现客户的源站响应头如下，存在 Vary:Accept-Encoding, Cookie。

我们知道，如果想要访问通过 CDN 加速的站点，实现 PC 端访问展示 PC 端内容，mobile 端访问展示 mobile 内容，则可以通过源站增加：vary: user-agent 来实现此需求；即针对不同的 UA 缓存不同的源站内容；

同样：vary: Accept-Encoding 表示，针对客户端不同的 Accept-Encoding 值，缓存并响应不同的编码内容，常用于 Gzip，Br 等压缩各响应不同内容的需求中。

同理，对 vary: cookie，则表示对请求头中的 cookie，不同的值，缓存并响应不同的内容。

所以，本例中，因为请求头中包含 cookie，且每次访问客户端请求头中的 cookie 的值均不相同，相当于每次访问到 CDN 上都是一个全新的内容，都是第一次访问，第一次缓存，所以每次都是 MISS，但又因为 CDN 上配置了全站缓存 7776000 秒，所以每一个的 x-swift-cachetime 都是 7776000 如此长的时间。

4. 解决问题

源站去掉 vary: cookie 的 response Header 之后，此问题解决。

抽丝剥茧定位一个 CDN 访问慢的案例

作者：胡夫

问题描述

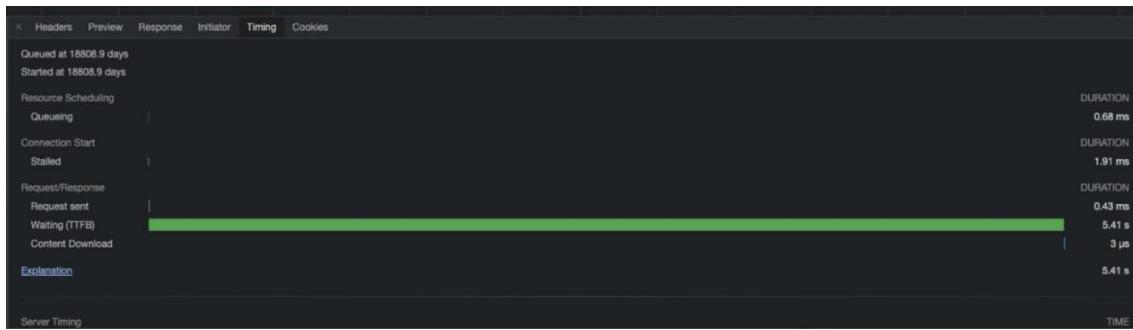
某客户反馈使用 CDN 以后下载时间很慢，TTFB 很长，需要定位原因。

The screenshot shows the Network tab in Chrome DevTools. A specific request for 'info' is selected. The Headers section shows the following details:

- Request URL: <https://www.wuage.com/index/auth/info>
- Request Method: GET
- Status Code: 200
- Remote Address: 10.10.10.10

In the Response Headers section, several headers are listed:

- x-swift-global-savetime: 1625110780787
- content-length: 78
- content-type: application/json; charset=utf-8
- date: Thu, 01 Jul 2021 03:39:47 GMT
- eagleid: 6a75f3a816251107821855912e
- server: Tengine
- set-cookie: JSESSIONID=0bb2e9b4-ac8a-4be2-b08b-555d5aa3b7ca; Domain=.wuage.com; Path=/
- vary: Accept-Encoding
- x-cache: L2cn2641[117,116,200-0,M], cache44.L2cn2641[118,0], vcache43.cn2620[5336,5335,200-1281,M], vcache38.cn2620[5352,0]
- x-application-context: info_web:8080
- x-cache: MISS TCP_MISS dir:-2/-2
- x-swift-cachetime: 0
- x-swift-error: orig response 5xx error
- x-swift-savetime: Thu, 01 Jul 2021 03:39:47 GMT



排查过程

1. 分析 CDN 日志

根据 Network 下的信息后台查到 CDN 的日志信息，发现访问慢是因为中间有过 504 重试，具体过程如下：

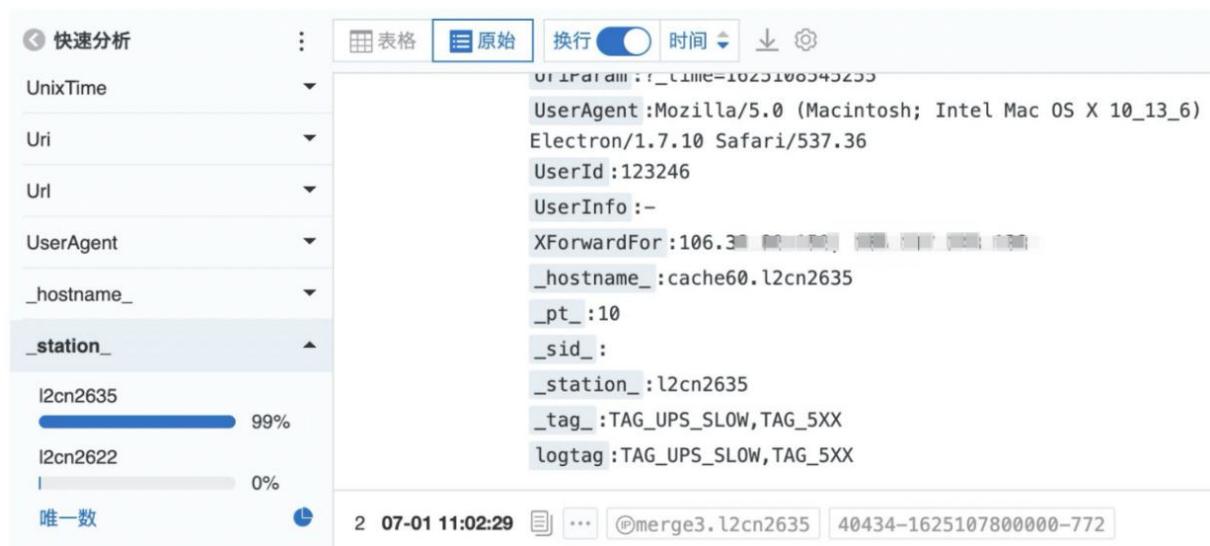
- (1) 客户端请求到 L1 (cn2620)，L1 回源到 cache18.l2cn2635 以后，cache18.l2cn2635 回源 504，RT 时间是 5182ms
- (2) cache18.l2cn2635 响应 504 以后，触发 L1 重试到 cache12.l2cn2641，回源成功，RT 时间是 137ms

所以虽然最终现象看起来是访问成功了，只是速度比较慢，实际的过程中是因为有 504 并发生了重试，耗时主要是第一次回源 504 的耗时。同时查看但是客户端提供的 Network 下的响应头也可以进一步确认此问题，当时虽然响应了 httpcode 200，但是响应头里响应了 x-swift-error: orig response 5xx error

日志列表						
RT(ms)	Fbt(ms)	状态码	错误码	错误信息	tr...	via
5182	5182	504	1281	orig response 5xx error swift回源响应5XX错误	6a...	cache18.l2cn2635[5179,5179,504-1281,M], cache36.l...
137	137	200	1281	orig response 5xx error swift回源响应5XX错误	6a...	cache12.l2cn2641[117,116,200-0,M], cache44.l2c...

2. 错误日志分析

日志服务 SLS 查看 L2 的错误日志分析，504 错误基本就是聚焦在 l2cn2635 节点，且 rt 时间是 5s 左右。



3. 听云探测

根据前面的分析，其他 L2 节点到源站首字节都正常，日志也正常，唯独这个 L2 节点到源站会 504，而且首字节时间是 5k ms，因此怀疑是源站做了相关安全策略，或者该节点到源站的网络问题。但是用户的源站是阿里云的 SLB，且用户反馈 SLB 层面并没有做安全策略。由于该 L2 是唐山的一个 L2 节点，因此想到用听云到唐山地区用三大运营商探测下源站，发现也一切正常。

时间	监测城市	监测运营商	监测点ID	监测点IP	带宽(估计)	DNS服务器	操作系统	浏览器	主机IP	主机城市	主机运营商	总下载时间
2021-07-01 10:53:40	唐山市	中国移动	8266995	183.197.177.126	>= 20 Mbps	111.11.1.1	Windows 7 sp1	Internet Explorer 11	139.140.141.142	上海市	阿里巴巴	0.18
2021-07-01 10:53:27	唐山市	中国联通	7333336	101.22.70.50	>= 10 Mbps	202.99.160.68	Windows 7 sp1	Internet Explorer 11	139.140.141.143	上海市	阿里巴巴	0.438
2021-07-01 10:53:25	唐山市	中国联通	6764480	60.2.249.91	>= 10 Mbps	202.99.160.68	Windows 7 sp1	Internet Explorer 9	139.140.141.144	上海市	阿里巴巴	0.193
2021-07-01 10:53:25	唐山市	中国联通	8568992	110.231.52.151	>= 40 Mbps	202.99.160.68	Windows 7 sp1	Internet Explorer 11	139.140.141.145	上海市	阿里巴巴	0.236
2021-07-01 10:53:22	唐山市	中国联通	7083768	60.2.78.122	>= 40 Mbps	202.99.160.68	Windows 7 sp1	Internet Explorer 11	139.140.141.146	上海市	阿里巴巴	0.328
2021-07-01 10:53:14	唐山市	中国电信	6916333	111.227.97.165	>= 40 Mbps	222.222.202.202	Windows 7 sp1	Internet Explorer 11	139.140.141.147	上海市	阿里巴巴	0.404
2021-07-01 10:53:14	唐山市	中国电信	7279328	106.116.227.172	>= 100 Mbps	222.222.222.222	Windows 7 sp1	Internet Explorer 11	139.140.141.148	上海市	阿里巴巴	0.293
2021-07-01 10:53:12	唐山市	中国移动	4913007	211.143.66.28	>= 10 Mbps	111.11.1.1	Windows 7 sp1	Internet Explorer 11	139.140.141.149	上海市	阿里巴巴	0.382
2021-07-01 10:53:12	唐山市	中国联通	8442323	120.4.182.3	>= 100 Mbps	202.99.160.68	Windows 10	Internet Explorer 11	139.140.141.150	上海市	阿里巴巴	0.167
2021-07-01 10:53:09	唐山市	中国电信	8000961	111.227.17.112	>= 100 Mbps	222.222.202.202	Windows 7 sp1	Internet Explorer 11	139.140.141.151	上海市	阿里巴巴	0.403
2021-07-01 10:53:07	唐山市	中国电信	8195688	111.227.17.112	>= 100 Mbps	222.222.202.202	Windows 10	Internet Explorer 11	139.140.141.152	上海市	阿里巴巴	0.225
2021-07-01 10:53:02	唐山市	中国移动	8549441	120.211.1.88	>= 20 Mbps	111.11.1.1	Windows 7 sp1	Internet Explorer 8	139.140.141.153	上海市	阿里巴巴	0.263
2021-07-01 10:53:01	唐山市	中国移动	8564357	120.211.1.88	>= 10 Mbps	111.11.1.1	Windows 7 sp1	Internet Explorer 11	139.140.141.154	上海市	阿里巴巴	0.272
2021-07-01 10:52:51	唐山市	中国移动	8102629	120.211.1.88	>= 20 Mbps	111.11.1.1	Windows 7 sp1	Internet Explorer 8	139.140.141.155	上海市	阿里巴巴	0.281

4. 为什么会 504?

CDN 回源源站的时候产生了 504，但是客户源站 SLB 的日志并没有查到 504 的日志，看起来这个 504 日志不是源站响应的，那么就有可能 CDN 回源超时导致的。但是根据 rt 时间看 5s 就 504 了，看起来并没有达到默认 CDN 约定的回源超时时间，默认 CDN 回源的超时时间是：建联超时 10s（也就是说 tcp 连接 10s 超时），读客户端请求头超时时间：30s（也就是说应用层 30s 超时）。至此，一共有如下几个疑问需要确认：

- (1) 看起来 rt 时间 5s 并没有达到超时时间，看起来不是超时导致，那么有可能是源站直接响应 504。
- (2) 回源 504 日志显示的错误码是"0"，也就是 CDN 认为是正常请求，并不认为是连接不上或超时断开这种情况，这也进一步印证了源站直接响应 504 的情况。
- (3) 但是现在的问题是源站查不到 504 日志，那么有没有可能是客户反馈的源站日志排查的结果不可靠？

日志列表									
类别	时间	节点	域名	RT(ms)	Fbt(ms)	状态码	错误码	错误信息	操作
access	07-01 11:39:47	l2cn2635	www.wuage.com	5180	5180	504	0	请求正常	6a751 [详细]
tproxy	07-01 11:39:47	l2cn2635	www.wuage.com	5179	5179	504	0	请求正常	6a751 [详细]
swift	07-01 11:39:47	l2cn2635	www.wuage.com	5179	5179	504	1281	orig response 5xx err...	6a751 [详细]

5. 源站抓包

为了验证到底是不是源站吐的 504，因此考虑让用户在源站侧抓包，然后客户端去访问复现，出现问题以后提供 chrome 的 har 文件和源站的抓包文件。根据日志分析，第一次是 cache18.l2cn2635 回源 504，回源的 IP 是 101.x.xx.xx；然后 L1 重试到 cache12.l2cn2641 回源成功，回源的 IP 是 115.xx.xx.xx。从源站的抓包文件看，可以看到来自 115.xx.xx.xx 的报文，但是过滤不到来自 101.xx.xx.xx 的报文，看起来就是 101.xx.xx.xx 的请求没有到源站。

No.	Time	RTT	Source	Destination	Protocol	Length	Window size sc	Time to live	Info
14584	2021-07-01...		115.5	172.18.0.85	TCP	66		128	64 52503 -> 443 [SYN] Seq=0 Win=13594 Len=0 MSS=1460 SACK_PERM=...
14586	2021-07-01...	0.000479000	115.5	172.18.0.85	TCP	54		128	64 52503 -> 443 [ACK] Seq=1 Ack=1 Win=13696 Len=0
14587	2021-07-01...		115.5	172.18.0.85	TLSv1.2	355		128	64 Client Hello
14590	2021-07-01...		115.5	172.18.0.85	TCP	54		128	64 52503 -> 443 [ACK] Seq=302 Ack=1461 Win=16640 Len=0
14591	2021-07-01...		115.5	172.18.0.85	TCP	54		128	64 52503 -> 443 [ACK] Seq=302 Ack=2921 Win=19456 Len=0
14600	2021-07-01...	0.000533000	115.5	172.18.0.85	TCP	54		128	64 52503 -> 443 [ACK] Seq=302 Ack=3283 Win=22400 Len=0
14601	2021-07-01...		115.5	172.18.0.85	TLSv1.2	147		128	64 Client Key Exchange, Change Cipher Spec, Encrypted Handshak...
14608	2021-07-01...	0.001221000	115.5	172.18.0.85	TLSv1.2	1408		128	64 Application Data
14689	2021-07-01...	0.040274000	115.5	172.18.0.85	TCP	54		128	64 52503 -> 443 [ACK] Seq=1749 Ack=3951 Win=28288 Len=0
17533	2021-07-01...		115.5	172.18.0.85	TCP	54		128	64 52503 -> 443 [FIN, ACK] Seq=1749 Ack=3951 Win=28288 Len=0
17533	2021-07-01...	0.000514000	115.5	172.18.0.85	TCP	54		128	64 52503 -> 443 [ACK] Seq=1750 Ack=3952 Win=28288 Len=0
14585	2021-07-01...	0.000009000	172.1	115.56.242...	TCP	66		128	64 443 -> 52503 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460...
14588	2021-07-01...	0.000007000	172.1	115.56.242...	TCP	54		128	64 443 -> 52503 [ACK] Seq=1 Ack=302 Win=30336 Len=0
14592	2021-07-01...		172.1	115.56.242...	TLSv1.2	3336		128	64 Server Hello, Certificate, Server Key Exchange, Server Hell...
14604	2021-07-01...	0.000230000	172.1	115.56.242...	TLSv1.2	296		128	64 New Session Ticket, Change Cipher Spec, Encrypted Handshake...
14621	2021-07-01...	0.003348000	172.1	115.56.242...	TLSv1.2	480		128	64 Application Data
17534	2021-07-01...	0.000058000	172.1	115.56.242...	TCP	54		128	64 443 -> 52503 [FIN, ACK] Seq=3951 Ack=1750 Win=33024 Len=0

6. CDN 抓包

目前情况看起来有点复杂了，源站 SLB 上没有抓到来自 CDN 回源 IP 的报文，且源站日志没有记录该 504 请求的 CDN 回源 IP 的日志，看现象就是 CDN 回源源站的回源链路有问题。为了进一步验证这个问题，只能到发生 504 的 cache18.l2cn2635 这台机器上去抓包。从抓包结果来看，的确是源站直接响应了 504，而且看响应 504 的报文里，TTL 和 SYN 握手报文的 TTL 一致，看起来也不是被劫持导致，也不像是网络问题。而且只有两台 CDN 机器回源有 504 问题，其他机器回源一切正常。

No.	Time	RTT	Source	Destination	Protocol	Length	Info
1	2021-07-01 17:05:29.883804		101.158	139.198	TCP	66	60322 -> 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PE
2	2021-07-01 17:05:29.909212	0.025408...	139.198	101.158	TCP	66	80 -> 60322 [SYN, ACK] Seq=0 Ack=1 Win=13594 Len=0 MSS=1
3	2021-07-01 17:05:29.909234	0.000022...	101.158	139.198	TCP	54	60322 -> 80 [ACK] Seq=1 Ack=1 Win=29696 Len=0
4	2021-07-01 17:05:29.909306	0.025403...	101.158	139.198	HTTP	196	GET http://www.wuage.com/index/auth/info HTTP/1.1
5	2021-07-01 17:05:29.934709	0.025403...	139.198	101.158	TCP	60	80 -> 60322 [ACK] Seq=1 Ack=143 Win=14848 Len=0
6	2021-07-01 17:05:34.934639	139.198	101.158	139.198	TCP	343	80 -> 60322 [PSH, ACK] Seq=1 Ack=143 Win=14848 Len=289
7	2021-07-01 17:05:34.934662	0.000023...	101.158	139.198	TCP	54	60322 -> 80 [ACK] Seq=143 Ack=290 Win=30720 Len=0
8	2021-07-01 17:05:34.934752	139.198	101.158	139.198	HTTP	150	HTTP/1.1 504 Gateway Time-out (text/html)
9	2021-07-01 17:05:34.934762	0.000010...	101.158	139.198	TCP	54	60322 -> 80 [ACK] Seq=143 Ack=386 Win=30720 Len=0

> Frame 8: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits)
> Ethernet II, Src: IETF-VRRP-VRID_01
> Internet Protocol Version 4, Src:

```

0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x14 (DSCP: Unknown, ECN: Not-ECT)
Total Length: 136
Identification: 0x9e21 (40481)
> Flags: 0x4000, Don't fragment
Fragment offset: 0
Time to live: 49
Protocol: TCP (6)
Header checksum: 0xe8c8 [validation disabled]
[Header checksum status: Unverified]
Source: 139.198.101.158
Destination: 101.158
> Transmission Control Protocol, Src Port: 80, Dst Port: 60322, Seq: 290, Ack: 143, Len: 96
> [2 Reassembled TCP Segments (385 bytes): #6(289), #8(96)]
> Hypertext Transfer Protocol
> Line-based text data: text/html (1 lines)

```

7. 源站配置检查

各种证据表明，越来越像是源站有安全策略拦截了，但是源站是 SLB，看 SLB 的一些访问控制的策略，比如 IP 黑白名单，即使拦截了应该也能查到 403 的日志，而不会是 504。检查配置发现

SLB 开启了"透明 WAF"功能。进一步了解该功能原理，SLB 开启透明 WAF 功能以后不需要更改 DNS 解析，请求 SLB 的流量都先走 TWAF（透明 WAF），TWAF 在 CDN 和 SLB 之间充当了一个透明代理。TWAF 和 client 建连时借用了 SLB 的 IP，TWAF 和 SLB 建连时借用了 client 的 IP。那么根据这个现象看，极有可能是 CDN 的请求被 TWAF 拦截了（在下图中，CDN 的回源 IP 就是 client）。

报文流向

假设client ip为1.1.1.1, eip为2.2.2.2, 访问源站eip的80端口, 会涉及两个连接:

Client到waf (对应1、2、3)

1.1.1.1:123 -> 2.2.2.2:80

Waf到源站 (对应4、5)

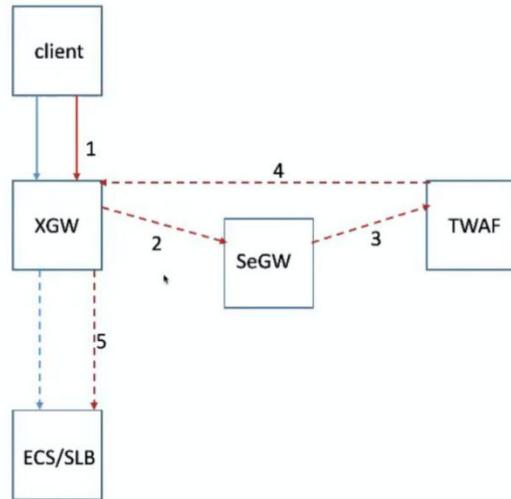
1.1.1.1:124 -> 2.2.2.2:80

标红部分是透明waf和cname-waf在代理模式上的差别，仅此而已。

蓝色部分为未被WAF引流时的报文流向；
红色部分为被WAF引流后的报文流向；

实线表示非vxlan报文

虚线表示vxlan报文



8. TWAF 拦截记录

去查了 TWAF 的拦截记录，发现果然有拦截记录，CDN 的 tproxy IP（回源 IP）被 WAF 识别成攻击 IP，被用户的访问控制策略加入到了 WAF 的黑名单里。致此，终于可以结案！

<input type="checkbox"/>	101.72.10.107	2021.6.26河北	删除
<input type="checkbox"/>	101.72.10.108	2021.6.26河北	删除
<input checked="" type="checkbox"/>	101.72.10.109	2021.6.26河北	删除
<input type="checkbox"/>	101.72.10.110	2021.6.26河北	删除

问题原因

由于业务请求走了 CDN，CDN 又通过汇聚式的 L2 节点回源，而源站配置了透明 WAF，WAF 配置的安全规则认为这些频繁回源的 CDN 节点 IP 是攻击 IP，被访问控制策略加入到黑名单，而 TWAF 通过响应 504 的形式拒绝了来自 CDN 的回源请求。根据上面的 TWAF 实现原理图可以知道，这个 504 的请求对于 SLB 来说完全是没有感知的，所以 SLB 上肯定没有日志，需要查 WAF 的拦截日志才能发现。

解决方案

从黑名单里移除该 CDN 的回源 IP 即可。但是由于目前配置的策略，很有可能还会继续把 CDN 的回源 IP 当成攻击 IP 处理，因此需要调整 WAF 拦截策略，或者把 CDN 的回源 IP 加入到白名单。但是这也有风险，因为 CDN 回源时会智能分配节点访问源站，回源的节点 IP 是不固定的，因此不建议将源站的回源策略白名单设置为固定的节点 IP 列表，这样有可能因为 CDN 回源 IP 变更，新增 IP 由于未在白名单导致被拦截继而发生回源失败的情况。

如果有强诉求，也可以通过调用 CDN 的 DescribeL2VipsByDomain 接口获取 CDN 回源的节点 IP 列表并添加到源站服务器的白名单中。该接口仅支持日峰值带宽为 1Gbps 以上的用户调用，如果符合该条件，可以提交工单，申请该接口的调用权限。

适用于

CDN

全站加速

长连接访问的场景下偶发出现 CDN 主动发 FIN 包

作者：胡夫

问题描述

用户在测试长连接场景，发现频率不高的情况下，服务端(CDN) 也会主动发 FIN 包，并且 FIN 包之前会先发一个 encrypted alert。

No.	Time	RTT	Source	Destination	Protocol	Length	Info
107	2021-04-20 18:25:01.039852		47.164.12.100	164.12.100.47	TLSv1.2	1003	Application Data
108	2021-04-20 18:25:01.039881	0.000029	164.12.100.47	47.164.12.100	TCP	54	24685 → 443 [ACK] Seq=16914 Ack=12338 Win=196 Len=0
109	2021-04-20 18:25:01.195030		164.12.100.47	47.164.12.100	TLSv1.2	1355	Application Data
110	2021-04-20 18:25:01.195630	0.0000608	164.12.100.47	47.164.12.100	TCP	60	443 → 24685 [ACK] Seq=12338 Ack=18215 Win=358 Len=0
117	2021-04-20 18:25:01.195630		164.12.100.47	47.164.12.100	TLSv1.2	1003	Application Data
118	2021-04-20 18:25:01.725664	0.000034	164.12.100.47	47.164.12.100	TCP	54	24685 → 443 [ACK] Seq=18215 Ack=13287 Win=197 Len=0
121	2021-04-20 18:25:01.913459		164.12.100.47	47.164.12.100	TLSv1.2	1355	Application Data
122	2021-04-20 18:25:01.914144	0.0000685	164.12.100.47	47.164.12.100	TCP	60	443 → 24685 [ACK] Seq=13287 Ack=19516 Win=358 Len=0
127	2021-04-20 18:25:02.126496		164.12.100.47	47.164.12.100	TLSv1.2	1003	Application Data
128	2021-04-20 18:25:02.126527	0.000031	164.12.100.47	47.164.12.100	TCP	54	24685 → 443 [ACK] Seq=19516 Ack=14236 Win=199 Len=0
131	2021-04-20 18:25:02.312007		164.12.100.47	47.164.12.100	TLSv1.2	1355	Application Data
132	2021-04-20 18:25:02.312699	0.0000692	164.12.100.47	47.164.12.100	TCP	60	443 → 24685 [ACK] Seq=14236 Ack=20817 Win=358 Len=0
135	2021-04-20 18:25:02.521504		164.12.100.47	47.164.12.100	TLSv1.2	987	Application Data
136	2021-04-20 18:25:02.521543	0.000035	164.12.100.47	47.164.12.100	TCP	54	24685 → 443 [ACK] Seq=20817 Ack=15169 Win=201 Len=0
137	2021-04-20 18:25:02.521608		164.12.100.47	47.164.12.100	TLSv1.2	123	Encrypted Alert
138	2021-04-20 18:25:02.521611	0.000003	164.12.100.47	47.164.12.100	TCP	54	24685 → 443 [ACK] Seq=20817 Ack=15238 Win=201 Len=0
139	2021-04-20 18:25:02.521637		164.12.100.47	47.164.12.100	TCP	60	443 → 24685 [FIN, ACK] Seq=15238 Ack=20817 Win=358 Len=0
143	2021-04-20 18:25:02.561335	0.039698	164.12.100.47	47.164.12.100	TCP	54	24685 → 443 [ACK] Seq=20817 Ack=15239 Win=201 Len=0
144	2021-04-20 18:25:02.580129		164.12.100.47	47.164.12.100	TCP	54	24685 → 443 [FIN, ACK] Seq=20817 Ack=15239 Win=201 Len=0
145	2021-04-20 18:25:02.580741	0.0000612	164.12.100.47	47.164.12.100	TCP	60	443 → 24685 [ACK] Seq=15239 Ack=20818 Win=358 Len=0

▶ Frame 139: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
 ▶ Ethernet II, Src: JuniperH [00:0c:29:00:00:00], Dst: 443 [00:00:00:00:00:43]
 ▶ Internet Protocol Version 4, Src: 164.12.100.47, Dst: 164.12.100.1
 ▶ Transmission Control Protocol, Src Port: 443, Dst Port: 24685, Seq: 15238, Ack: 20817, Len: 0

问题排查

1. encrypted alert 确认

SSL 通信在断开连接时均为发送 Encrypted Alert 信息给客户端告知要关闭 ssl 会话了，同步会发生 FIN,ACK 报文从 TCP 层面断开链接：

Since we are already in an encrypted connection, the only way to really know what is being sent within packets is to make Wireshark or similar tools aware of the keys used in the transmission. Even though this is possible, I think for the purpose of this analysis it is enough to know that the client sends an alert message when the connection is asked to be closed actively by the client or server. The type of this Alert message should be CloseNotify (type 0), but we won't be able to see it from the raw data. In this case, the client is the sender of the following Alert message:

Secure Sockets Layer

TLSv1.2 Record Layer: Encrypted Alert

Content Type: Alert (21)

Version: TLS 1.2 (0x0303)

Length: 26

Alert Message: Encrypted Alert

encrypted alert 只是一个告警，“encrypted alert”是 https 连接断开时（即 tengine 的 finalize）必会发出的，就是四层需要挥手，tls 层需要 encrypted alert，是应用层 finalize 后的必须过程。因此这个问题不在这里，根本的原因需要看下 tengine 为什么会 finalize，也就是为什么会断开连接。

2. 日志分析

排查 CDN 的日志，从日志侧没有发现异常。因为本身这是属于 tcp 长连接的断开，并不是属于异常断开，不会记录 499 或者其他相关错误码，日志的相关字段都是正常的。

3. 节点侧抓包

尝试在 CDN 的回源节点上抓包，在复现问题的时候，发现源站会返回 connection close 头部，导致 tengine 断开。

4. 模拟复现

由于当时用户的请求是一个 POST 请求，因此可以写一个脚本，本地 host 指向源站，用脚本发起长连接请求，再连续请求的过程中也会出现断开的情况，示例代码如下：

```
import json
import requests
client=requests.session()
headers = {'Content-Type': 'application/json', 'Connection': 'keep-alive', 'User-Agent': 'volc-sdk-java/v1.0.0', 'Host': 'xxx.xxx.com', 'X-Date': '20210422T030241Z', 'Authorization': 'HMAC-SHA256 Credential=AKLTZTQ1MmRhYWZkMGRINDAyNDhmOGUzZmY0NTE0Mjk2YWE/20210422/cn-north-1/AdBlocker/request, SignedHeaders=content-type;host;x-content-sha256;x-date, Signature=1f2axxxd'}
i=1
while i<=120:
    url="http://xxx.xxx.com/?Action=AsyncAdBlocker&Version=2021-04-16"
    body = json.dumps({"Parameters": "{\"account_id\":\"sg_07461406093a\", \"chat_text\":\".....\\\\\\...13799275992....\", \"channel_type\": \"world\", \"server_id\": \"9313002\", \"operate_time\": 1619060561, \"ip\": \"192.168.0.56\", \"sender_role_id\": \"72464\", \"senderNickname\": \"uBmMdLxw\", \"senderSignature\": \"\", \"senderRoleLevel\": 2, \"senderRoleCe\": 232, \"senderPayTotal\": 0, \"senderRoleCreateTime\": 1619060257, \"receiverAccount_id\": \"0\", \"receiverNickname\": null, \"receiverRoleCe\": 0, \"receiverPayTotal\": 0}", "Service": "chat", "AppId": 222719})
    r=client.post(url,headers=headers, data=body)
    print r.status_code
    print i
    i+=1
```

问题原因

客户端发起长连接请求，请求头 Connection: keep-alive。刚开始交互一直是 Connection: keep-alive，一段时间之后源站会主动响应 Connection: close 头，导致 cdn tengine 主动断开和客户端的连接。

解决方案

源站修改响应 Connection: close 头的策略去优化。

适用于

CDN

DCDN

开启 ossftp 以后端口不通导致连接失败

作者：胡夫

问题描述

通过 ftp 客户端去连接 ossftp 连接失败。

状态: 正在连接 123... 07:2048...

状态: 尝试连接"ECONNREFUSED - 连接被服务器拒绝"失败。

错误: 无法连接到服务器

状态: 正在等待重试...

排查过程

1. 基本信息确认

首先需要确保服务器上已经安装 ossftp 工具并按照文档启动 ossftp 服务。其次需要确保服务器上安全组已经开启相关的服务端口。如果只是开放了端口，但是 ossftp 服务并没有启动，那么肯定是连接不通这个端口。

2. 为什么启动了 ossftp 还是无法连接

启动 ossftp 以后直接 telnet 测试服务器的公网 2048 端口还是不通，如图 2-1。从服务器上看 ossftp 确实启动起来的，如图 2-2。通过 netstat -anpt 命令查看状态发现 Web 服务端口 8192 已经有监听，但是 FTP 服务端口 2048 端口并没有监听，如图 2-3。

2048 端口作为 FTP 服务端口，用于接收 FTP 请求；8192 端口作为 Web 服务端口，用于打开 ossftp 的图形化管理界面。

```
sh-3.2# telnet 123.57.129.207 2048
Trying 123.57.129.207...
telnet: connect to address 123.57.129.207: Connection refused
telnet: Unable to connect to remote host
```

图 2-1 telnet 测试 2048 端口不通

```
[root@iz2zeggdgw7gg2wvzwrsaqq ossftp]# bash start.sh
Apr 17 17:32:05 - [DEBUG] start confirm_ossftp_exit
Apr 17 17:32:06 - [INFO] start accounts time cost 0
['/usr/bin/python2', '/www/wwwroot/ossftp/ossftp/ftpserver.py', u'--listen_address=123.57.129.207', '--port=2048', '--passive_ports_start=51000', '--passive_ports_end=53000', u'--logLevel=INFO', u'--bucket_endpoints=98pic.oss-cn-hangzhou.aliyuncs.com']
Apr 17 17:32:06 - [INFO] ossftp started
Apr 17 17:32:06 - [INFO] start ossftp time cost 33
Apr 17 17:35:50 - [DEBUG] launcher web_control 39.107.126.72:48145 GET /img/fixed-width.png
Apr 17 17:35:50 - [DEBUG] launcher web_control 39.107.126.72:48157 GET /img/logo.png
Apr 17 17:55:00 - [DEBUG] launcher web_control 58.101.142.41:40300 GET /css/bootstrap.css
Apr 17 17:55:00 - [DEBUG] launcher web_control 58.101.142.41:40305 GET /css/style.css
Apr 17 17:55:00 - [DEBUG] launcher web_control 58.101.142.41:40302 GET /css/bootstrap-responsive.css
Apr 17 17:55:00 - [DEBUG] launcher web_control 58.101.142.41:40304 GET /css/ladda-themeless.min.css
Apr 17 17:55:00 - [DEBUG] launcher web_control 58.101.142.41:40306 GET /js/jquery-1.11.2.min.js
Apr 17 17:55:00 - [DEBUG] launcher web_control 58.101.142.41:40303 GET /css/flat-ui.css
Apr 17 17:55:00 - [DEBUG] launcher web_control 58.101.142.41:40307 GET /js/bootstrap.min.js
Apr 17 17:55:00 - [DEBUG] launcher web_control 58.101.142.41:40308 GET /js/flat-ui.min.js
```



图 2-2 bash start.sh 启动 ossftp

```
[root@iz2zeggdgw7gg2wvzwrsaqq ~]# netstat -anpt
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp        0      0 0.0.0.0:21            0.0.0.0:*           LISTEN     1115/pure-ftpd (SER
tcp        0      0 0.0.0.0:22            0.0.0.0:*           LISTEN     2282/sshd
tcp        0      0 0.0.0.0:8888          0.0.0.0:*           LISTEN     2178/python
tcp        0      0 0.0.0.0:888           0.0.0.0:*           LISTEN     1858/nginx: master
tcp        0      0 0.0.0.0:443           0.0.0.0:*           LISTEN     1858/nginx: master
tcp        0      0 0.0.0.0:8192           0.0.0.0:*           LISTEN     28478/python2
tcp        0      0 127.0.0.1:6379          0.0.0.0:*           LISTEN     1841/redis-server 1
tcp        0      0 0.0.0.0:80            0.0.0.0:*           LISTEN     1858/nginx: master
tcp        0      0 127.0.0.1:22           127.0.0.1:44180    ESTABLISHED 16756/sshd: root@no
tcp        0      0 127.0.0.1:47712          127.0.0.1:8888    TIME_WAIT   -
tcp        0      0 172.17.169.188:80         5.188.210.227:62044 TIME_WAIT   -
tcp        0      0 127.0.0.1:55576          127.0.0.1:3306    TIME_WAIT   -
tcp        0      0 127.0.0.1:44180          127.0.0.1:22     ESTABLISHED 2178/python
tcp        0      0 172.17.169.188:51554        100.100.18.120:443 TIME_WAIT   -
tcp        0      0 127.0.0.1:46084          127.0.0.1:22     ESTABLISHED 2178/python
tcp        0      0 127.0.0.1:22           127.0.0.1:44178    ESTABLISHED 16709/sshd: root@no
tcp        0      0 172.17.169.188:22         58.101.142.41:40579 ESTABLISHED 25899/sshd: root@pt
tcp        0      0 127.0.0.1:44178          127.0.0.1:22     ESTABLISHED 2178/python
tcp        0      0 172.17.169.188:60866        100.100.30.25:80   ESTABLISHED 1917/AliYunDun
tcp        0      0 127.0.0.1:22           127.0.0.1:46084    ESTABLISHED 17125/sshd: root@no
tcp        0      36 172.17.169.188:22         58.101.142.41:39921 ESTABLISHED 24705/sshd: root@pt
tcp6       0      0 ::1:21                 ::*:               LISTEN     1115/pure-ftpd (SER
tcp6       0      0 ::1:3306              ::*:               LISTEN     1840/mysqld
```

图 2-3 netstat -anpt 查看端口监听

3. VPC 机器需要监听到 0.0.0.0

由于用户 ECS 用的是 VPC 网络，VPC 的机器 eth0 是内网 IP，公网 IP 是配置在 xgw 网关上形成映射关系的。因此 VPC 网络的机器 FTP 端口（2048 端口）需要监听 0.0.0.0 或者网卡的私网 IP 地址。通过服务器公网 IP+8192 端口登录 ossftp 的 Web 服务界面，将监听地址由公网 IP 改成 0.0.0.0，然后保存配置并重启 ossftp，如图 3-1。



图 3-1 监听地址改成 0.0.0.0

在图像界面更改监听端口并重启 ossftp 以后，在 ECS 的控制台界面可以看到 ossftp 重新启动的打印信息，并且可以看到监听到了 0.0.0.0，如图 3-2（这里也可以跟图 2-2 去对比下）。

```

Apr 17 17:55:12 - [INFO] start accounts time cost 0
['/usr/bin/python2', '/www/wwwroot/98ppt.tbwx.cn/ossftp/ftpserver.py', '--listen_address=0.0.0.0', '--port=2048', '--passive_ports_start=51000', '--passive_ports_end=53000', '--loglevel=INFO', '--bucket_endpoints='98pic.oss-cn-hangzhou.aliyuncs.com']
Apr 17 17:55:12 - [INFO] ossftp started
Apr 17 17:55:12 - [INFO] start ossftp time cost 12
Apr 17 17:55:16 - [DEBUG] launcher web_control 58.101.142.41:40331 GET /config?cmd=get_config
Apr 17 17:55:16 - [DEBUG] launcher web_control 58.101.142.41:40331 GET /favicon.ico

```

图 3-2 监听地址变更

重新通过 netstat -anpt 去查看可以看到 2048 端口监听到 0.0.0.0 上了，如图 3-3。

```
[root@iz2zeaadaw7aa2wvzwrsaaz ~]# netstat -anpt
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp      0      0 0.0.0.0:21              0.0.0.0:*            LISTEN    1115/pure-ftpd (SER
tcp      0      0 0.0.0.0:22              0.0.0.0:*            LISTEN    2282/sshd
tcp      0      0 0.0.0.0:8888            0.0.0.0:*            LISTEN    2178/python
tcp      0      0 0.0.0.0:888             0.0.0.0:*            LISTEN    1858/nginx: master
tcp      0      0 0.0.0.0:443             0.0.0.0:*            LISTEN    1858/nginx: master
tcp      0      0 0.0.0.0:2048            0.0.0.0:*            LISTEN    9282/python2
tcp      0      0 0.0.0.0:8192             0.0.0.0:*            LISTEN    9282/python2
tcp      0      0 127.0.0.1:6379            0.0.0.0:*            LISTEN    1841/redis-server 1
tcp      0      0 0.0.0.0:80               0.0.0.0:*            LISTEN    1858/nginx: master
tcp      0      0 127.0.0.1:22              127.0.0.1:44180      ESTABLISHED 16756/sshd: root@no
tcp      0      0 127.0.0.1:44180            127.0.0.1:22         ESTABLISHED 2178/python
tcp      0      0 127.0.0.1:46084            127.0.0.1:22         ESTABLISHED 2178/python
tcp      0      0 127.0.0.1:22              127.0.0.1:44178      ESTABLISHED 16709/sshd: root@no
tcp      0      0 172.17.169.188:2048          58.101.142.41:40335  ESTABLISHED 9282/python2
tcp      0      0 172.17.169.188:47714          123.129.198.197:443  TIME_WAIT   -
tcp      0      0 127.0.0.1:47968             127.0.0.1:8888       TIME_WAIT   -
tcp      0      0 172.17.169.188:22              58.101.142.41:40579  ESTABLISHED 25899/sshd: root@pt
tcp      0      0 127.0.0.1:44178             127.0.0.1:22         ESTABLISHED 2178/python
tcp      0      0 172.17.169.188:60866            100.100.30.25:80      ESTABLISHED 1917/AliYunDun
tcp      0      0 127.0.0.1:22              127.0.0.1:46084      ESTABLISHED 17125/sshd: root@no
tcp      0      36 172.17.169.188:22             58.101.142.41:39921  ESTABLISHED 24705/sshd: root@pt
tcp6     0      0 :::21                  ::::*                LISTEN    1115/pure-ftpd (SER
tcp6     0      0 ::::3306                ::::*                LISTEN    1840/mysql
```

图 3-3 netstat -anpt 查看端口监听状态

此时再通过 telnet 命令测试服务器公网的 2048 端口已经正常，如图 3-4。

```
sh-3.2# telnet 123.129.198.207 2048
Trying 123.129.198.207...
Connected to 123.129.198.207.
Escape character is '^]'.
220 oss ftpd ready.
^C
```

图 3-4 telnet 测试端口是否正常

解决方案

如果 ECS 是 VPC 网络的，需要把 FTP 监听端口监听到 0.0.0.0 或者私有 IP 上。

适用于

对象存储 OSS

PythonSDK-从 OSS 下载文件报错 InconsistentError

作者：胡夫

问题描述

用 Python SDK 从 OSS 上下载文件，会报错 {'status': -3, 'x-oss-request-id': '600E562B45F1FB3738772972', 'details': 'InconsistentError: IncompleteRead from source'}，文件下载不完整。

排查过程

1. 初步分析

PythonSDK 默认开启了 CRC 校验，如果客户端计算的 CRC 值与服务端返回的 CRC 值不一致，则会抛出 InconsistentError 异常，具体可以参考 OSS 官方文档--数据安全性。

2. 抓包分析

为了确认为什么会出现下载异常的原因，需要客户端抓包来分析。过滤出客户端跟 OSS 的交互报文，然后通过 `tcp.flags.fin == 1` 过滤 FIN 包，可以看到前面的 FIN 包都是客户端下载完成先发起 FIN，然后 OSS 响应 FIN，符合预期，例如#1027 和#1028、#1945 和#1946 等等。另外，从抓包看，OSS 的 IP 是 100.118.28.50，这是一个内网 IP，说明客户端还是通过内网从 OSS 上下载数据的，由于走内网，网络影响的可能性非常小。但是#69874 包可以看到，这个是 OSS 先发的 FIN 包，断开了连接。

No.	Time	RTT	Source	Destination	Protocol	Length	Time to live	Info
1027	2021-01-25 13:16:48.875166		192.168.22.138	100.118.28.50	TCP	56	64	32806 - 80 [FIN, ACK] Seq=1195 Ack=2597468 Win=5267968 Len=
1028	2021-01-25 13:16:48.876787	0.001541000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 32806 [FIN, ACK] Seq=2597468 Ack=1196 Win=32768 Len=0
1040	2021-01-25 13:16:50.675306		192.168.22.138	100.118.28.50	TCP	56	64	32814 - 80 [FIN, ACK] Seq=1195 Ack=2133843 Win=4341248 Len=
1040	2021-01-25 13:16:50.676722	0.001416000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 32814 [FIN, ACK] Seq=2133843 Ack=1196 Win=32768 Len=0
3335	2021-01-25 13:16:57.980096		192.168.22.138	100.118.28.50	TCP	56	64	32854 - 80 [FIN, ACK] Seq=1195 Ack=5939984 Win=5267456 Len=
3335	2021-01-25 13:16:57.981678	0.001582000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 32854 [FIN, ACK] Seq=5939984 Ack=1196 Win=32768 Len=0
9101	2021-01-25 13:17:22.418868		192.168.22.138	100.118.28.50	TCP	56	64	32972 - 80 [FIN, ACK] Seq=1195 Ack=2165116 Win=6396416 Len=
9102	2021-01-25 13:17:22.420458	0.001590000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 32972 [FIN, ACK] Seq=2165116 Ack=1196 Win=32768 Len=0
13875	2021-01-25 13:17:40.131020		192.168.22.138	100.118.28.50	TCP	56	64	33066 - 80 [FIN, ACK] Seq=1195 Ack=1112390 Win=2295808 Len=
13876	2021-01-25 13:17:40.132368	0.001348000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 33066 [FIN, ACK] Seq=1112390 Ack=1196 Win=32768 Len=0
13882	2021-01-25 13:17:43.820590		192.168.22.138	100.118.28.50	TCP	56	64	33072 - 80 [FIN, ACK] Seq=1195 Ack=496772 Win=4614656 Len=
13883	2021-01-25 13:17:43.821956	0.001366000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 33072 [FIN, ACK] Seq=496772 Ack=1196 Win=32768 Len=0
13884	2021-01-25 13:17:46.928297		192.168.22.138	100.118.28.50	TCP	56	64	32974 - 80 [FIN, ACK] Seq=1195 Ack=907694 Win=6850048 Len=
13885	2021-01-25 13:17:46.929369	0.001392000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 32974 [FIN, ACK] Seq=907694 Ack=1196 Win=32768 Len=0
13886	2021-01-25 13:17:51.428978		192.168.22.138	100.118.28.50	TCP	56	64	33074 - 80 [FIN, ACK] Seq=1195 Ack=6996288 Win=4617728 Len=
13889	2021-01-25 13:17:51.422486	0.001516000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 33074 [FIN, ACK] Seq=6996288 Ack=1196 Win=32768 Len=0
13891	2021-01-25 13:17:55.874778		192.168.22.138	100.118.28.50	TCP	56	64	32966 - 80 [FIN, ACK] Seq=1195 Ack=13764337 Win=6851584 Len=
13892	2021-01-25 13:17:55.876584	0.001806000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 32966 [FIN, ACK] Seq=13764337 Ack=1196 Win=32768 Len=0
25760	2021-01-25 13:18:47.225881		192.168.22.138	100.118.28.50	TCP	56	64	33218 - 80 [FIN, ACK] Seq=1195 Ack=31881365 Win=5259264 Len=
25761	2021-01-25 13:18:47.227341	0.001460000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 33218 [FIN, ACK] Seq=31881365 Ack=1196 Win=32768 Len=0
25766	2021-01-25 13:19:06.242837		192.168.22.138	100.118.28.50	TCP	56	64	33260 - 80 [FIN, ACK] Seq=1195 Ack=40034503 Win=4615168 Len=
25767	2021-01-25 13:19:06.244796	0.001959000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 33260 [FIN, ACK] Seq=40034503 Ack=1196 Win=32768 Len=0
54069	2021-01-25 13:22:25.200629		192.168.22.138	100.118.28.50	TCP	56	64	34262 - 80 [FIN, ACK] Seq=1195 Ack=143984122 Win=8387584 Len=
54010	2021-01-25 13:22:25.202159	0.001530000	100.118.28.50	192.168.22.138	TCP	56	63	80 - 34262 [FIN, ACK] Seq=143984122 Ack=1196 Win=32768 Len=
69874	2021-01-25 13:24:43.557383		100.118.28.50	192.168.22.138	TCP	5752	63	80 - 34656 [FIN, PSH, ACK] Seq=105902723 Ack=1195 Win=32768
69876	2021-01-25 13:24:48.946250		192.168.22.138	100.118.28.50	TCP	56	64	34656 - 80 [FIN, ACK] Seq=1195 Ack=105908420 Win=4629504 Len=

> Frame 69874: 5752 bytes on wire (46016 bits), 5752 bytes captured (46016 bits)
> Linux cooked capture
> Internet Protocol Version 4, Src: 100.118.28.50, Dst: 192.168.22.138
> Transmission Control Protocol, Src Port: 80, Dst Port: 34656, Seq: 105902723, Ack: 1195, Len: 5696

3. ZeroWindow

点击#69874 包包，Follow TCP Stream 过滤这路 TCP 流，可以看到以下数据包。由于客户端接收缓冲区已满，无法正常处理 OSS 方发送的数据，OSS 发出的在途字节都未得到客户端的确认，当待发送的数据为 0，也就是发了出去，但是都没有确认，允许发送的大小为 0。也就是说，在途字节数等于对方的接收窗口，这个时候，Wireshark 打上【TCP window Full】标记，表示“我不能再发送数据”了。与此同时客户端发送的数据包 Win=0 表示接收窗口为 0，表示客户端不能再接收数据了，wireshark 标记了【TCP ZeroWindow】标记。从#69804 号包，13:23:52 开始客户端一直发送 Win=0，直到 13:24:43 的#69818 号包客户端才重新更新 Win=397312，wireshark 标记为【TCP Window Update】。在 13:23:52~13:24:43 期间的时间段内，一直处于 OSS 发送数据而客户端无法正常接收的状态。

No.	Time	RTT	Source	Destination	Protocol	Length	Ti	Info
69798	2021-01-25 13:23:52.260316		100.118.28.50	192.168.22.138	TCP	1480	- 80 - 34656 [ACK] Seq=105341667 Ack=1195 Win=32768 Len=1424 [TCP segment of a connection]	
69799	2021-01-25 13:23:52.260333		100.118.28.50	192.168.22.138	TCP	1480	- 80 - 34656 [ACK] Seq=105343091 Ack=1195 Win=32768 Len=1424 [TCP segment of a connection]	
69800	2021-01-25 13:23:52.260360		100.118.28.50	192.168.22.138	TCP	12872	- 80 - 34656 [ACK] Seq=105344515 Ack=1195 Win=32768 Len=12816 [TCP segment of a connection]	
69801	2021-01-25 13:23:52.260374		100.118.28.50	192.168.22.138	TCP	8600	- 80 - 34656 [ACK] Seq=105357331 Ack=1195 Win=32768 Len=8544 [TCP segment of a connection]	
69802	2021-01-25 13:23:52.260518	0.000144...	192.168.22.138	100.118.28.50	TCP	56	- 34656 - 80 [ACK] Seq=1195 Ack=105365875 Win=1024 Len=0	
69803	2021-01-25 13:23:52.464794		100.118.28.50	192.168.22.138	TCP	1880	- [TCP Window Full] 80 - 34656 [PSH, ACK] Seq=105365878 Ack=1195 Win=32	
69804	2021-01-25 13:23:52.505573	0.040779...	192.168.22.138	100.118.28.50	TCP	56	- [TCP ZeroWindow] 34656 - 80 [ACK] Seq=1195 Ack=105366899 Win=0 Len=0	
69805	2021-01-25 13:23:52.714785		100.118.28.50	192.168.22.138	TCP	56	- [TCP Keep-Alive] 80 - 34656 [ACK] Seq=105366898 Ack=1195 Win=32768 Len=0	
69806	2021-01-25 13:23:52.714794		192.168.22.138	100.118.28.50	TCP	56	- [TCP ZeroWindow] 34656 - 80 [ACK] Seq=1195 Ack=105366899 Win=0 Len=0	
69807	2021-01-25 13:23:53.132797		100.118.28.50	192.168.22.138	TCP	56	- [TCP Keep-Alive] 80 - 34656 [ACK] Seq=105366898 Ack=1195 Win=32768 Len=0	
69808	2021-01-25 13:23:53.966814		100.118.28.50	192.168.22.138	TCP	56	- [TCP Keep-Alive] 80 - 34656 [ACK] Seq=105366898 Ack=1195 Win=32768 Len=0	
69809	2021-01-25 13:23:53.966826		192.168.22.138	100.118.28.50	TCP	56	- [TCP ZeroWindow] 34656 - 80 [ACK] Seq=1195 Ack=105366899 Win=0 Len=0	
69810	2021-01-25 13:23:55.634792		100.118.28.50	192.168.22.138	TCP	56	- [TCP Keep-Alive] 80 - 34656 [ACK] Seq=105366898 Ack=1195 Win=32768 Len=0	
69811	2021-01-25 13:23:55.634802		192.168.22.138	100.118.28.50	TCP	56	- [TCP ZeroWindow] 34656 - 80 [ACK] Seq=1195 Ack=105366899 Win=0 Len=0	
69812	2021-01-25 13:23:58.066810		100.118.28.50	192.168.22.138	TCP	56	- [TCP Keep-Alive] 80 - 34656 [ACK] Seq=105366898 Ack=1195 Win=32768 Len=0	
69813	2021-01-25 13:23:58.966825		192.168.22.138	100.118.28.50	TCP	56	- [TCP ZeroWindow] 34656 - 80 [ACK] Seq=1195 Ack=105366899 Win=0 Len=0	
69814	2021-01-25 13:24:05.638803		100.118.28.50	192.168.22.138	TCP	56	- [TCP Keep-Alive] 80 - 34656 [ACK] Seq=105366898 Ack=1195 Win=32768 Len=0	
69815	2021-01-25 13:24:05.638814		192.168.22.138	100.118.28.50	TCP	56	- [TCP ZeroWindow] 34656 - 80 [ACK] Seq=1195 Ack=105366899 Win=0 Len=0	
69816	2021-01-25 13:24:18.982813		100.118.28.50	192.168.22.138	TCP	56	- [TCP Keep-Alive] 80 - 34656 [ACK] Seq=105366898 Ack=1195 Win=32768 Len=0	
69817	2021-01-25 13:24:18.982827		192.168.22.138	100.118.28.50	TCP	56	- [TCP ZeroWindow] 34656 - 80 [ACK] Seq=1195 Ack=105366899 Win=0 Len=0	
69818	2021-01-25 13:24:43.320602		192.168.22.138	100.118.28.50	TCP	56	- [TCP Window Update] 34656 - 80 [ACK] Seq=1195 Ack=105366899 Win=397312	
69819	2021-01-25 13:24:43.321522		100.118.28.50	192.168.22.138	TCP	456	- 80 - 34656 [PSH, ACK] Seq=105366899 Ack=1195 Win=32768 Len=400 [TCP segment of a connection]	
69820	2021-01-25 13:24:43.321536	0.000014...	192.168.22.138	100.118.28.50	TCP	56	- 34656 - 80 [ACK] Seq=1195 Ack=105367299 Win=397312 Len=0	
69821	2021-01-25 13:24:43.321547		100.118.28.50	192.168.22.138	TCP	22840	- 80 - 34656 [ACK] Seq=105367299 Ack=1195 Win=32768 Len=22784 [TCP segment of a connection]	
69822	2021-01-25 13:24:43.321602		100.118.28.50	192.168.22.138	TCP	24264	- 80 - 34656 [PSH, ACK] Seq=105390883 Ack=1195 Win=32768 Len=24208 [TCP segment of a connection]	
69823	2021-01-25 13:24:43.321608		100.118.28.50	192.168.22.138	TCP	11448	- 80 - 34656 [ACK] Seq=105414291 Ack=1195 Win=32768 Len=11392 [TCP segment of a connection]	

> Frame 69818: 56 bytes on wire (448 bits), 56 bytes captured (448 bits)
 > Linux cooked capture
 > Internet Protocol Version 4, Src: 192.168.22.138, Dst: 100.118.28.50
 > Transmission Control Protocol, Src Port: 34656, Dst Port: 80, Seq: 1195, Ack: 105366899, Len: 0

解决方案

由于客户端长时间不读服务端数据，而 OSS 服务端配置的 send timeout 是 30s，超过这个时间服务端会主动断开连接，因此会出现这种数据没有下载完成和 OSS 断开的情况，于是就出现了 CRC 校验失败导致报错的情况。这种情况需要客户端层面去检查客户端的性能。

适用于

对象存储 OSS

通过 ffplay 播放 OSS 上的 mp3 文件会断开

作者：胡夫

问题描述

用户的 mp3 视频存在在 OSS 上，使用 ffplay 播放的时候，播放途中会出现播放断开，报错如下：

```
Error in the pull function. The specified session has been invalidated for some reason.
```

```
ffplayproc 55. 7.100 / 55. 7.100
Input #0, mp3, from 'https://...o.oss-cn-shenzhen.aliyuncs.com/live/teacherFile
_test/ed9a3775-1210-4f73-aee7-b7a70c531525/8311033988141615780914724.mp3':
Duration: 00:10:48.75, start: 0.025057, bitrate: 80 kb/s
Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 80 kb/s
Metadata:
encoder          : LAME3 98r
[tls @ 000001ac265cf6c0] Error in the pull function.= 0B f=0/0
[tls @ 000001ac265cf6c0] The specified session has been invalidated for some rea
son.
[mp3float @ 000001ac2d3e7400] invalid new backstep -1 0B f=0/0
 90.21 M-A: 0.000 fd= 0 aq= 0KB vq= 0KB sq= 0B f=0/0
```

初步分析

1. 尝试复现

根据用户提供的 mp3 音频的 URL，尝试在本地复现，发现测试播放一直正常，并没有出现用户说的播放中途断开的问题，而用户却几乎百分百必现该问题，因此怀疑还是客户端网络环境问题。

2. 传输加速正常

让用户尝试使用 OSS 的传输加速来播放，测试发现使用传输加速的地址去播放，就可以完整地播放完。根据以上测试，初步判断跟客户的网络环境有关。不过用户显然不认可这个结论，首先用

户反馈同样把这个 mp3 音频放到另外一个云厂家，在同样的客户端上去播放就一切正常，这不得不怀疑这个问题可能还是跟 OSS 有关。其次使用 OSS 的传输加速是需要费用的，客户并不想为此买单。

3. 对比测试

让用户尝试直接用 chrome 浏览器播放或者用 VLC 播放器去播放，播放效果还可以，可以顺利播放完，但是用 ffplay 播放却一直有问题。这个测试的结论，似乎又推翻了上一个用传输加速测试的结果，因为同样的环境下用别的播放器却可以完整播放。同时，如果 wget 下载也是可以正常下载完，并不会存在断开的问题，由此分析这个问题还是跟播放器流式请求有关。

抓包分析-OSS

1. TCP ZeroWindow

为了更直观定位问题，尝试让用户用 http 去播放复现并提供一份抓包数据。通过过滤对应 tcp 流从下往上看，可以看到在 19:21:48.605622 时间点，NO1965 包客户端因为接收窗口满了，Wireshark 打了 TCP ZeroWindow 的标记，随后 OSS 发 TCP Keep-Alive 探测帧探测，直到 19:21:59.621757 时间点，NO1997 包客户端重新更新接收窗口，Wireshark 标记 TCP Window Update，OSS 才重新发送数据，中间间隔了 11s，见下图 3-1。同时看整个数据包，存在比较多的这种情况。

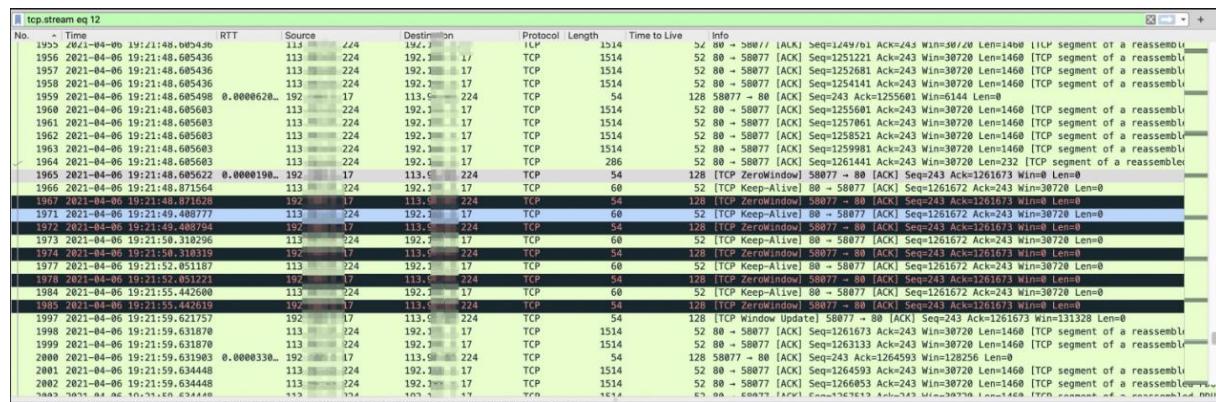


图 3-1 TCP ZeroWindow

2. OSS 发送了 FIN

从 NO2152 包开始 OSS 继续发送数据给客户端，直到最后 NO2177 包 OSS 突然发送了一个 FIN 包断开了连接，通过更改 Wireshark 对于 Time 的显示格式发现从开始播放到 OSS 发了 FIN 包断开连接，整个过程才持续了 100 多秒，而整个音频有 10 分钟。这里从抓包的现象来看的确跟用户描述的现象一致，在播放的过程中停止播放了，而且从抓包结果来看还是因为 OSS 突然发了 FIN 包导致播放器断开连接，无法读取到新的视频 Packet 导致停止播放，见下图 3-2：

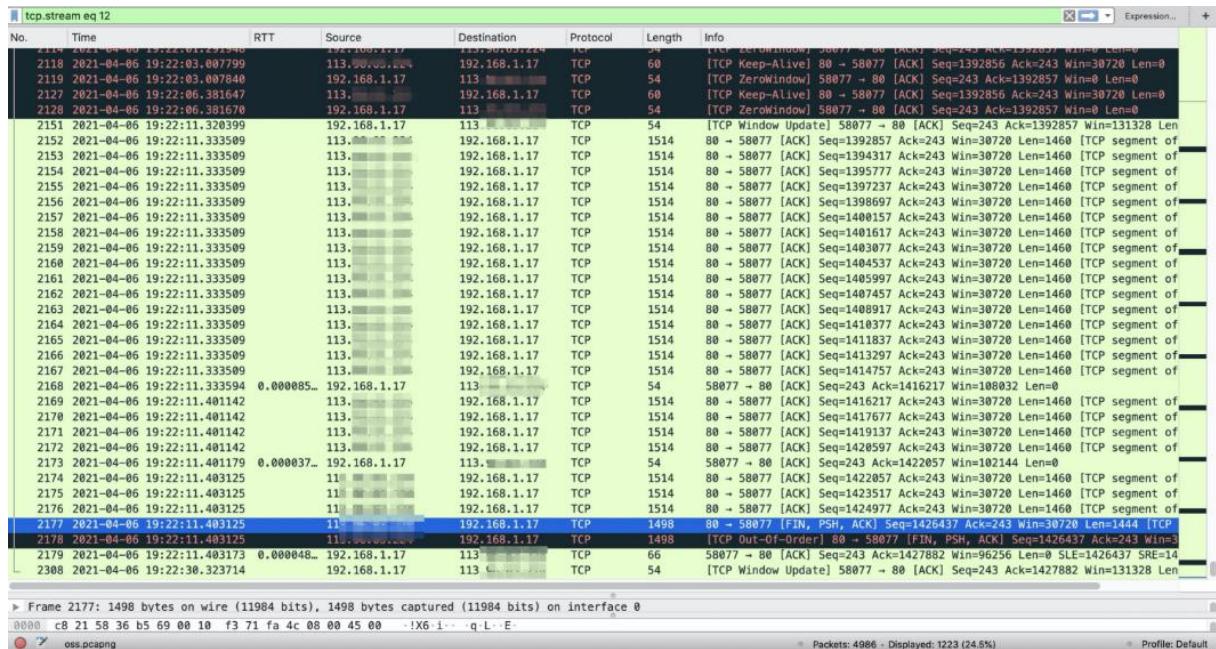


图 3-2 OSS 发送了 FIN 包

3. OSS 单连接最低下载速度

以上现象看比较像一个 OSS 最低下载速度的限制，我们先来说下这个最低下载速度的理解。

一个正常 C/S 架构的系统，通常会有很多 Buffer，会设置很多超时时间，针对 Tengine (OSS 基于 Tengine) 会有 send_timeout, recv_timeout, keepalive_timeout 等各种超时限制。这就会造成系统会有一个最小下载速度的限制。通过基于 wget 的 --limit-rate 功能进行二分测试可以得到 OSS 最低下载速度的区间，单连接持续 5k 以内速度必然出问题(一般持续 30s+出问题)，单连接持续 5 ~10k 以内速度随机出问题，看系统状况 (比较具有偶然性) 单连接持续 10k+ 基本不出

问题。

4. 分析下载速度

看抓包文件的 Conversations 信息，发现下载速率很低，132s 才请求了 1695KB 的数据，平均速度也就差不多 10KB/s 的速度，见下图 3-3：

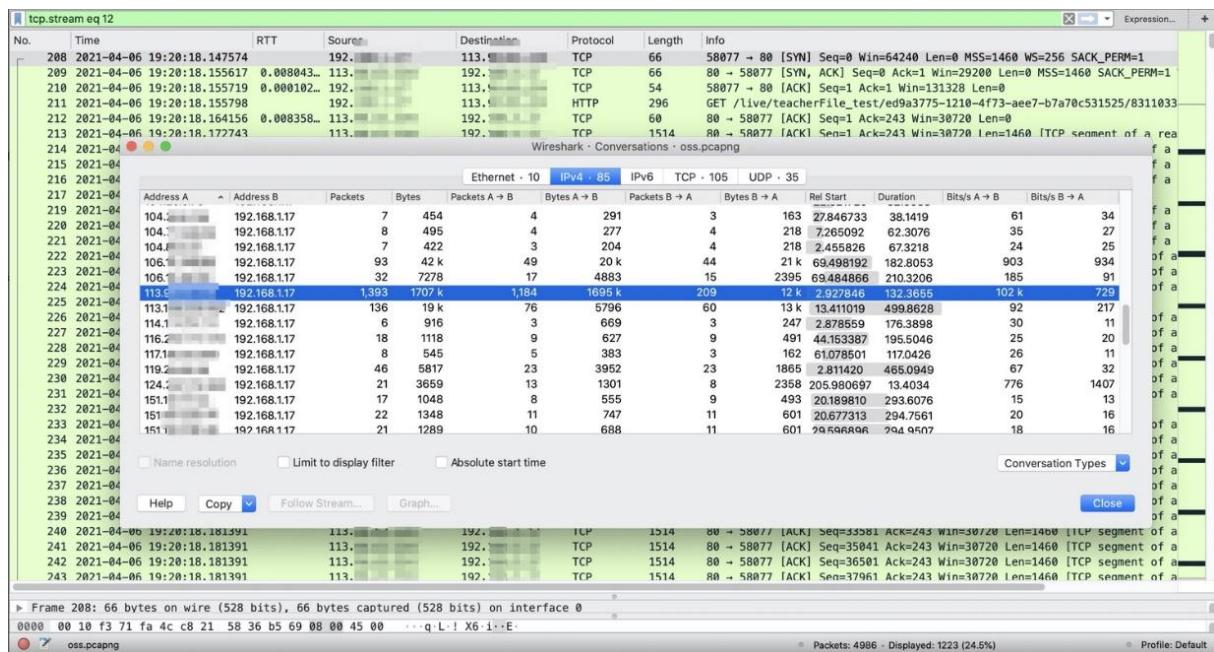


图 3-3 分析下载速度

分析源文件 mp3 文件，大小是 6559630Bytes，整个音频时长 648s，见下图 3-4。按照音频大小和音频时长可以计算得出平均每秒 10123Bytes，略微大于 10KB/s。但网络并不能保证每一秒的数据都一定大于 10KB，在这个过程中有多次存在下载速度比较低的情况，低于 OSS 的最低下载速度，就会被 OSS 断掉连接，这也可以解释为什么 OSS 会突然发 FIN 包，同时也可以解释为何客户端一直出现 TCP ZeroWindow 的问题。OSS 要高于这个最低下载速度发包，但播放速度以及码率限制了播放器缓冲区能够接收的数据，所以就会将下载速度降下来，多次之后就会出现后续 OSS 不再发送数据。基于以上分析，要解决这个问题，需要给这个音频转码，提高码率以此来保证提高下载速度（因为播放时长是不可能缩短的）。文件不大的情况下也可以考虑一次性全读出来，放内存或者本地再慢慢处理。

```

sh-3.2# ffplay "https://oss-cn-shenzhen.aliyuncs.com/live/teacherFile_test/ed9a3775-1210-4f73-aee7-b7a70c531525/8311033988141615780914724.mp3"
ffplay version 4.0.2 Copyright (c) 2003-2018 the FFmpeg developers
built with Apple LLVM version 9.0.0 (clang-900.0.39.2)
configuration: --prefix=/usr/local/Cellar/ffmpeg/4.0.2 --enable-shared --enable-pthreads --enable-version3 --enable-hardcoded-tables --enable-avresample --cc
=clang --host-cflags= --host-ldflags= --enable-gpl --enable-ffplay --enable-libmp3lame --enable-libx264 --enable-libxvid --enable-opencl --enable-videotoolbox
--disable-lzma
libavutil      56. 14.100 / 56. 14.100
libavcodec     58. 18.100 / 58. 18.100
libavformat    58. 12.100 / 58. 12.100
libavdevice     58.  3.100 / 58.  3.100
libavfilter     7. 16.100 /  7. 16.100
libavresample   4.  0.  0 /  4.  0.  0
libswscale      5.  1.100 /  5.  1.100
libswresample   3.  1.100 /  3.  1.100
libpostproc    55.  1.100 / 55.  1.100
Input #0, mp3, from 'https://oss-cn-shenzhen.aliyuncs.com/live/teacherFile_test/ed9a3775-1210-4f73-aee7-b7a70c531525/8311033988141615780914724.mp3':
Duration: 00:10:48.75 start: 0.025057, bitrate: 80 kb/s
  Stream #0:0: Audio: mp3, 44100 Hz, stereo, fltp, 80 kb/s
    Metadata:
      encoder : LAME3.98r
[ 18.80 M-A: 0.000 fd= 0 aq= 16KB vq= 0KB sq= 0B f=0/0

```

图 3-4 ffplay 分析音频 Duration 时长信息

为什么使用传输加速正常

传输加速 sndbuf 稍大，对于这个文件本身恰好没问题而已，换个大点的文件还是会有一样的问题，传输加速并不能从根本上解决这个问题。

为什么使用 chrome 播放正常

Chrome 播放是通过 range 的形式去 GET 请求，需要多少数据就从服务器端 RangeGet 读多少，因此并不会产生这个最低下载速度的问题。

为什么我测试正常

我们的网络环境质量较好，因此在请求的过程中始终保持在 10KB/s + 的速度，并没有触发 OSS 的最低下载速度。而客户那边网络质量不佳，刚好到了这个临界点。

CDN 加速架构

1. CDN 加速也有问题

由于客户的业务就是需要 ffplay 流式读取播放，以上说的方案并不能满足用户需求。考虑到 CDN 加速 OSS，客户端可以就近读取缓存，有更好的加速效果，用户尝试使用了 CDN 加速的方式来播放，但是问题依旧反复出现。

2. 抓包分析-CDN

抓了一份 ffplay 播放通过 CDN 加速的 URL 复现问题的包，发现也是存在 CDN 在发送数据的时候突然发送了 FIN 包，见下图 NO2306 包。由于 CDN 也是基于 Tengine，看起来也是有跟 OSS 类似的问题。

3. 增加写客户端超时

由于 CDN 针对某一个域名是可以调整写客户端超时时间调整的，这个案例中可以加大写客户端超时时间，加大写客户端超时时间可以降低系统的最低下载速度，由此来解决这个问题。这个案例里，我们将用户的 CDN 域名的写客户端时间调整到 900s（默认 30s）以后，用户反馈可以完整的播放完 mp3 了。

适用于

对象存储 OSS

CDN

跨境访问 OSS 失败

作者：胡夫

问题描述

中国大陆地区访问香港 Bucket 资源访问失败。

排查过程

由于 ICMP 层面是通的，在客户端能复现的情况下先客户端抓包，抓包可以看到#9 号显示是服务端发了 RST 包。从 TTL 看，#6 号包是 TCP 的 SYN/ACK，TTL 是 39 符合预期。而#9 号 RST 包的 TTL 是 178，明显不符合预期（OSS 发出的 TTL 包是 64，每经过一条递减，不可能到 178），说明该 RST 包不是 OSS 发送，这种跨境访问的异常基本可以判断是公网链路有关。

No.	Time	RTT	Source	Destination	Protocol	Length	Ti: Info
5	2021-01-21 17:46:56.394264		192.168.2.75	47.***.***.78	TCP	66	- 58973 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
6	2021-01-21 17:46:56.430722	0.036458	47.***.***.78	192.168.2.75	TCP	66	- 80 → 58973 [SYN, ACK] Seq=0 Ack=1 Win=29280 Len=0 MSS=1444 SACK_PERM=1 WS=5
7	2021-01-21 17:46:56.430796	0.000074	192.168.2.75	47.***.***.78	TCP	54	- 58973 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
8	2021-01-21 17:46:56.431068		192.168.2.75	47.***.***.78	HTTP	484	- GET />...>.jpg HTTP/1.1
9	2021-01-21 17:46:56.478928	0.039860	47.***.***.78	192.168.2.75	TCP	66	- 80 → 58973 [RST] Seq=1 Win=262656 Len=0


```

> Frame 6: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: VMware_Bf:75:7e (00:50:56:8f:75:7e), Dst: Microsoft_02:7a:c0 (00:15:5d:02:7a:c0)
> Internet Protocol Version 4, Src: 47.***.***.78, Dst: 192.168.2.75
    0100 ... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x04 (DSCP: Unknown, ECN: Not-ECT)
        Total Length: 52
        Identification: 0x0000 (0)
        Flags: 0x4000, Don't fragment
        Fragment offset: 0
        Time to live: 39
        Protocol: TCP (6)
        Header checksum: 0x4e68 [validation disabled]
        [Header checksum status: Unverified]
        Source: 47.***.***.78
        Destination: 192.168.2.75
> Transmission Control Protocol, Src Port: 80, Dst Port: 58973, Seq: 0, Ack: 1, Len: 0

```

No.	Time	RTT	Source	Destination	Protocol	Length	Traffic Info
5	2021-01-21 17:46:56.394264		192.168.2.75	47.70.100.100	TCP	66	- 58973 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
6	2021-01-21 17:46:56.430722	0.036458	47.70.100.100	192.168.2.75	TCP	66	- 80 → 58973 [SYN, ACK] Seq=0 Ack=1 Win=2900 Len=0 MSS=1444 SACK_PERM=1 WS=5
7	2021-01-21 17:46:56.430796	0.000074	192.168.2.75	47.70.100.100	TCP	54	- 58973 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
8	2021-01-21 17:46:56.431068		192.168.2.75	47.70.100.100	HTTP	484	- GET / HTTP/1.1
9	2021-01-21 17:46:56.470928	0.039860	47.70.100.100	192.168.2.75	TCP	60	- 80 → 58973 [RST] Seq=1 Win=262656 Len=0

> Frame 9: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
 > Ethernet II, Src: VMware_Bf:75:7e (00:58:56:8f:75:7e), Dst: Microsoft_B2:7a:c0 (00:15:5d:02:7a:c0)
 > Internet Protocol Version 4, Src: 47.70.100.100, Dst: 192.168.2.75
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 > Differentiated Services Field: 0x04 (DSCP: Unknown, ECN: Not-ECT)
 Total Length: 40
 Identification: 0x874b (34635)
 Flags: 0x0000
 Fragment offset: 0
 Time to live: 178
 Protocol: TCP (6)
 Header checksum: 0x7c28 [validation disabled]
 [Header checksum status: Unverified]
 Source: 47.70.100.100
 Destination: 192.168.2.75
 > Transmission Control Protocol, Src Port: 80, Dst Port: 58973, Seq: 1, Len: 0

解决方案

这种场景由于需要走国际互联网出口，可能会受到跨境链路的影响。

- 如果是 http 访问有问题，可以考虑用 https 访问
- 如果是 https 访问有问题，可以考虑用 http 访问
- 开启传输加速，用传输加速域名去访问
- CDN/DCDN + OSS 的架构

适用于

对象存储 OSS

OSS 线上业务回调失败

作者：胡夫

问题描述

客户反馈线上业务大量回调失败，具体报错如下。但客户反馈本地去测试向回调服务发起请求，回调服务是可以正常接收请求的，但回调服务器上找不到 OSS 发起的回调日志。

```
com.aliyun.oss.OSSException: Error status : 502  
[ErrorCode]: CallbackFailed  
[RequestId]: 6131E16DA8654B3338371A32  
[HostId]: xxx.oss-cn-shenzhen.aliyuncs.com
```

问题排查

1. 初步分析

根据报错来看，是 OSS 向回调服务发起回调的时候失败了。502 错误可能是回调服务直接响应了 502，也可能是 OSS 向回调服务地址发起回调时候直接连不上或者会阻断，导致没有走到七层。初步判断是 OSS 到回调服务器之间的网络问题。

2. 基调探测

在阿里网站运维监测平台发起基调探测，探测客户的回调地址，发起有很多地区都存在连不上的请求，请求被 reset (connection reset by peer)。从探测结果来看，也怀疑是回调服务器的问题。

河南郑州移动	121.***.227	上海中国电信	500	92ms	20ms	27ms	1ms	92ms	0.04KB	38.00KB/s	查看	PING	DNS	路由追踪
黑龙江哈尔滨电信	121.***.227	上海中国电信	500	142ms	27ms	48ms	1ms	142ms	0.04KB	38.00KB/s	查看	PING	DNS	路由追踪
黑龙江哈尔滨联通	-	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
黑龙江佳木斯联通	-	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
湖北荆州电信	121.***.227	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
湖北武汉电信	-	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
湖北武汉联通	-	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
湖北襄阳移动	121.***.227	上海中国电信	500	108ms	35ms	27ms	1ms	108ms	0.04KB	38.00KB/s	查看	PING	DNS	路由追踪
湖南长沙电信	-	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
湖南长沙联通	-	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
吉林通化电信	-	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
吉林长春联通	-	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
江苏南京联通	-	-	-	-	-	-	-	-	-	-	查看	PING	DNS	路由追踪
江苏南京移动	121.***.227	上海中国电信	500	46ms	5ms	13ms	1ms	46ms	0.04KB	38.00KB/s	查看	PING	DNS	路由追踪
江苏无锡电信	121.***.227	上海中国电信	500	49ms	12ms	9ms	1ms	48ms	0.04KB	38.00KB/s	查看	PING	DNS	路由追踪

3. 抓包确认

本地模拟测试发一个 post 请求到客户回调服务确实是正常的，没有复现到问题。因此尝试到基调平台上抓包，同时由于阿里网站运维检平台目前暂时不支持抓包，因此先考虑用第三方基调"听云"平台去探测抓包。因为 OSS 用的深圳区域，因此直接在听云上用深圳三大运营商去发起探测，发现也是存在部分请求被重置（reset）的情况。

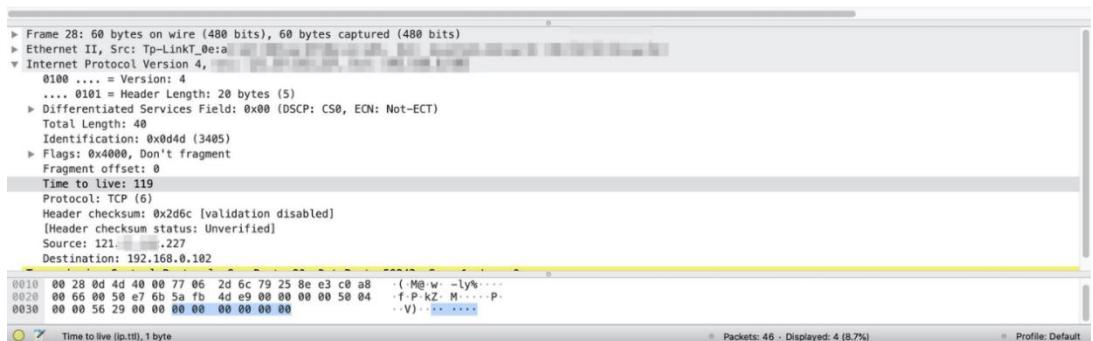
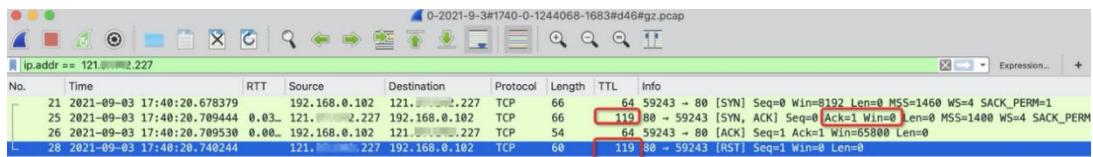
数据详情													
时间	错误代码	接入方式	监测城市	监测运营商	监测点ID	带宽(估计)	DNS服务器	操作系统	浏览器
2021-08-02 17:00:40	与服务器的连接被重置(页面)	LastMile	深圳市	中国联通	7503091	203.172	>= 40 Mbps	210.21.4.130	Windows 10	Internet Explorer 11			
2021-08-02 17:00:40	与服务器的连接被重置(页面)	LastMile	深圳市	中国联通	85680102	40	>= 40 Mbps	116.116.116.116	Windows 7 sp1	Internet Explorer 8			
2021-08-02 17:00:40	HTTP/1.1 500 Internal Server Error(页面)	LastMile	深圳市	中国移动	8579114	11	>= 100 Mbps	211.136.192.6	Windows 7 sp1	Internet Explorer 9			
2021-08-02 17:00:40	HTTP/1.1 500 Internal Server Error(页面)	LastMile	深圳市	中国联通	8520356	1.225	>= 100 Mbps	210.21.196.6	Windows 8.1	Internet Explorer 11			
2021-08-02 17:00:40	HTTP/1.1 500 Internal Server Error(页面)	LastMile	深圳市	中国移动	8599896	88.114	>= 20 Mbps	211.136.192.6	Windows 10	Internet Explorer 11			
2021-08-02 17:00:40	与服务器的连接被重置(页面)	LastMile	深圳市	中国联通	8271626	0.8	>= 40 Mbps	210.21.196.6	Windows 8.1	Internet Explorer 11			
2021-08-02 17:00:40	与服务器的连接被重置(页面)	LastMile	深圳市	中国电信	8453238	1.27	>= 100 Mbps	114.114.114.119	Windows 10	Internet Explorer 11			
2021-08-02 17:00:40	HTTP/1.1 500 Internal Server Error(页面)	LastMile	深圳市	中国移动	8425576	103	>= 10 Mbps	120.196.165.24	Windows 7 sp1	Internet Explorer 8			
2021-08-02 17:00:40	与服务器的连接被重置(页面)	LastMile	深圳市	中国电信	6705144	10.163	>= 40 Mbps	202.96.128.86	Windows 7 sp1	Internet Explorer 11			
2021-08-02 17:00:40	与服务器的连接被重置(页面)	LastMile	深圳市	中国电信	8148923	88.25	>= 1 Mbps	202.96.128.166	Windows 7 sp1	Internet Explorer 11			
2021-08-02 17:00:40	HTTP/1.1 500 Internal Server Error(页面)	LastMile	深圳市	中国移动	8578291	105	>= 10 Mbps	211.136.192.6	Windows 7 sp1	Internet Explorer 9			
2021-08-02 17:00:40	与服务器的连接被重置(页面)	LastMile	深圳市	中国电信	8568213	24.29	>= 40 Mbps	202.96.128.68	Windows 7 sp1	Internet Explorer 11			
2021-08-02 17:00:40	HTTP/1.1 500 Internal Server Error(页面)	LastMile	深圳市	中国移动	8643851	12.10	>= 100 Mbps	120.196.165.24	Windows 7 sp1	Internet Explorer 8			
2021-08-02 17:00:40	HTTP/1.1 500 Internal Server Error(页面)	LastMile	深圳市	中国电信	7240234	08.38	>= 20 Mbps	202.96.128.166	Windows 7 sp1	Internet Explorer 11			

抓包分析发现有以下几个点：

(1) 21、25、26 三个包是三次握手包，25 号是回调服务器的握手 sync 包，通告的 win 窗口居然是 0。

(2) 三次握手成功以后, 28 号包直接发了 reset 包, 从报文看这是服务端发了 reset 包。

(3) 从 TTL 看，服务端握手 sync 包(25 号)包的 TTL 是 119，服务端 reset 包(28 号)包的 TTL 也是 119，说明这种现象不太可能是劫持行为。因为如果不是服务端发的 reset 包，而是被网络链路的其他节点劫持以后发的 reset 包，那么 TTL 不会刚好也是 119。基于以上，基本上可以确认是回调服务器的原因造成的。



问题原因

经过确认，原因是客户自己回调服务器的 EIP（弹性公网 IP）有自带 DDoS 的防护功能，防护阈值是固定的，比如 200Mb（入带宽）。在客户业务量上涨后，客户虽然扩容了带宽，但 DDoS 访问阈值没有跟着调整，导致业务量上涨后触发了 DDoS 防护机制，开始流量清洗。这次是 Sync 访问的清洗，服务端通过回 Sync+Ack+Win=0 来阻断这次应用链接。

适用产品

对象存储 OSS

SDK 上传 OSS 报错 413 状态码

作者：胡夫

问题描述

某客户使用 OSS 的 SDK 上传到 OSS，上传超过 300M 的大文件时会报错。报错如下：

The requested resource does not allow request data with the requested method or the amount of data provided
in the request exceeds the capacity limit

排查过程

1. 初步分析

从客户提供的错误截图来看，可以得到几个关键信息：

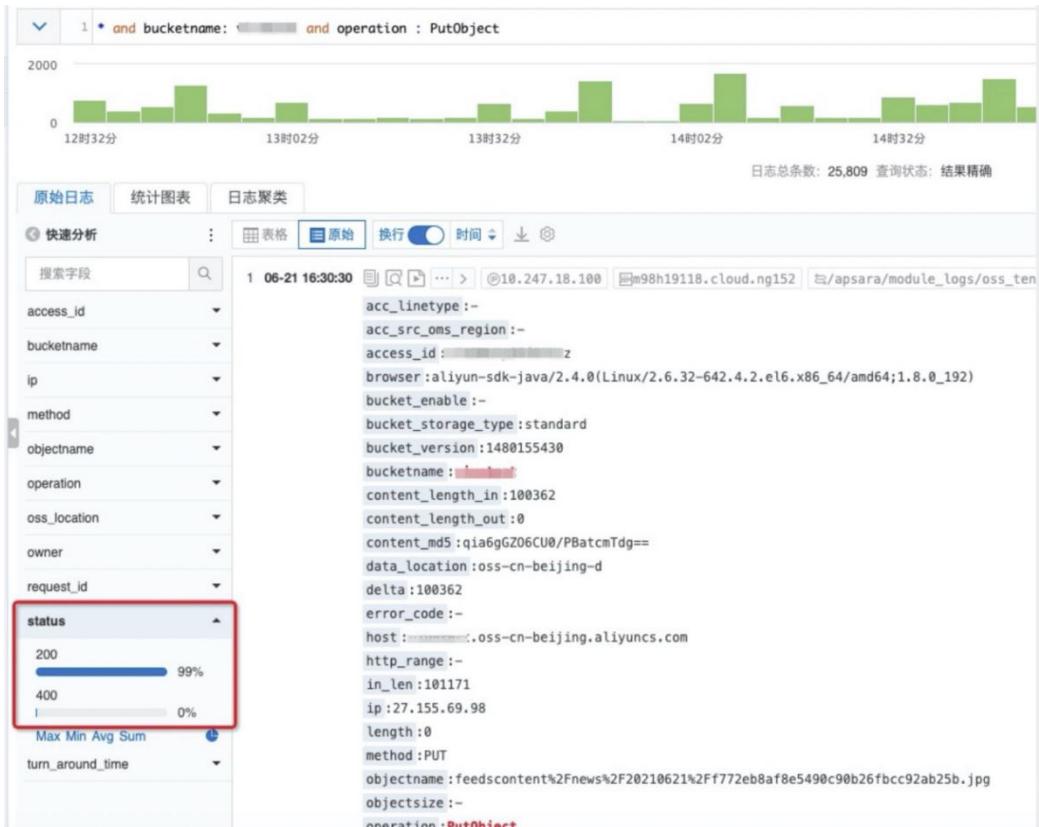
- (1) 错误 ErrorMessage 看，错误 code 是 413，而且这个错误的意思是上传超过大小限制了。
- (2) RequestId 打印的是 null，说明返回错误里的 RequestId 是空值，说明很有可能并不是 OSS 返回的，可能是中间走了代理。因为只要是 OSS 返回的，正常情况都会有 OSS 的 RequestId 的。
- (3) 但是从错误看是 OSSEException，看起来是服务端返回的，但返回的是 Tengine。
- (4) 从错误代码看，调用的是 JavaSDK 的 PutObject 接口，而该接口的上传大小限制是 5GB，而客户反馈的上传文件是 300+MB。

```
[ErrorCode] InvalidResponse
[RequestId] null
[HostId] null
[ResponseError]:
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head><title>413 Request Entity Too Large</title></head>
<body>
<h1>413 Request Entity Too Large</h1>
<p>The requested resource does not allow request data with the requested method or the amount of data provided in the request exceeds the capacity limit.<br>Powered by Tengine</p>
</body>
</html>

at com.aliyun.oss.common.utils.ExceptionFactory.createOSSEException(ExceptionFactory.java:109)
at com.aliyun.oss.common.utils.ExceptionFactory.createInvalidResponseException(ExceptionFactory.java:91)
at com.aliyun.oss.common.utils.ExceptionFactory.createInvalidResponseException(ExceptionFactory.java:81)
at com.aliyun.oss.internal.OSSErrorResponseHandler.handle(OSSErrorResponseHandler.java:71)
at com.aliyun.oss.common.comm.ServiceClient.handleResponse(ServiceClient.java:248)
at com.aliyun.oss.common.comm.ServiceClient.sendRequestImp(ServiceClient.java:130)
at com.aliyun.oss.common.comm.ServiceClient.sendRequest(ServiceClient.java:68)
at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:94)
at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:149)
at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:113)
at com.aliyun.oss.internal.OSSObjectOperation.writeObjectInternal(OSSObjectOperation.java:774)
at com.aliyun.oss.internal.OSSObjectOperation.putObject(OSSObjectOperation.java:144)
at com.aliyun.oss.OSSClient.putObject(OSSClient.java:557)
at com.aliyun.oss.OSSClient.putObject(OSSClient.java:539)
```

2. 日志分析

SLS 日志服务查询 OSS 日志，发现对应时间段只有 200 和 400 的 httpcode，并没有客户截图反馈的 413 的 httpcode。



3. 模拟测试

模拟测试 PUT 上传 OSS，设置很大的 content-length 可以复现类似的 ErrorMessage，但是有区别的是 OSS 返回的服务端信息是 AliyunOSS。这个错误是"Powered by AliyunOSS"，而客户返回的错误截图里显示是" Powered by Tengine"。集合前面分析的几点，基本可以判断是请求没有直接到 OSS，而是走了代理。

```
curl -X PUT 'BucketName.oss-cn-beijing.aliyuncs.com' -H 'content-length:444444444444' -v
*   Trying 47.xx.xx.xx:80...
* TCP_NODELAY set
* Connected to BucketName.oss-cn-beijing.aliyuncs.com (47.xx.xx.xx) port 80 (#0)
> PUT / HTTP/1.1
> Host: BucketName.oss-cn-beijing.aliyuncs.com
> User-Agent: curl/7.68.0
> Accept: */*
> content-length:444444444444
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 413 Request Entity Too Large
< Server: AliyunOSS
< Date: Tue, 22 Jun 2021 07:52:22 GMT
< Content-Type: text/html
< Content-Length: 377
< Connection: close
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head><title>413 Request Entity Too Large</title></head>
<body bgcolor="white">
<h1>413 Request Entity Too Large</h1>
<p>The requested resource does not allow request data with the requested method or the amount
of data provided in the request exceeds the capacity limit.<hr/>Powered by AliyunOSS</body>
</html>
* Closing connection 0
```

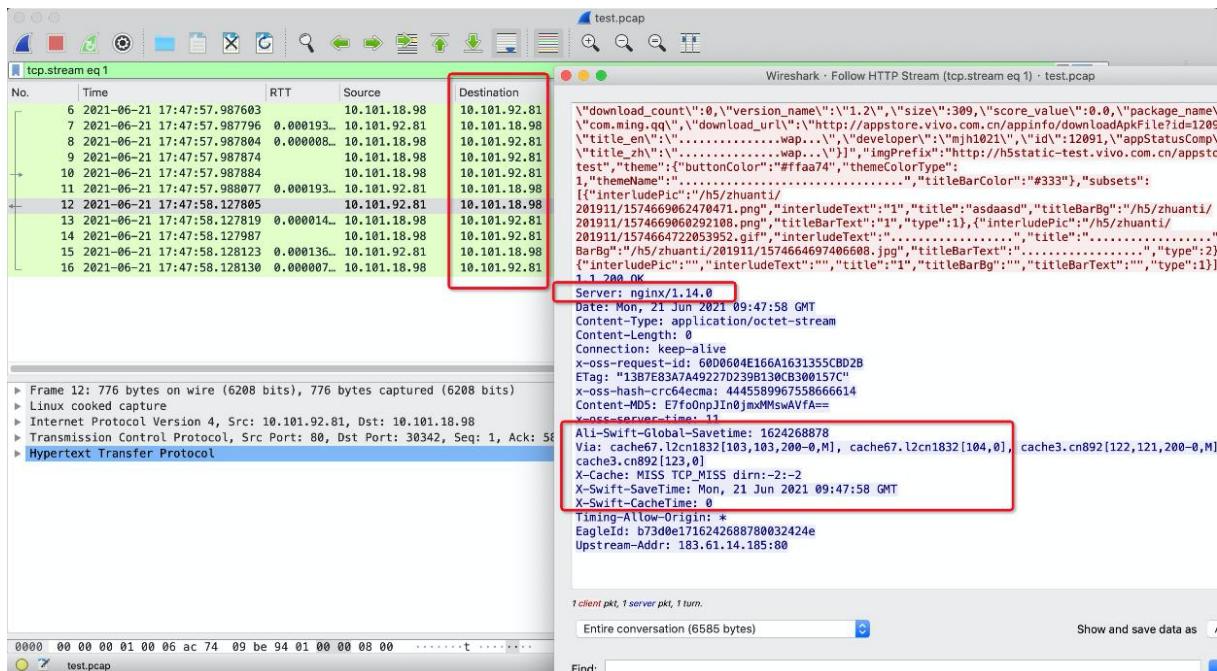
```
sh-3.2# curl -X PUT 'http://[REDACTED].oss-cn-beijing.aliyuncs.com' -H 'content-length:444444444444' -v
* Trying 47.111.111.111...
* TCP_NODELAY set
* Connected to [REDACTED].oss-cn-beijing.aliyuncs.com ([REDACTED]) port 80 (#0)
> PUT / HTTP/1.1
> Host: [REDACTED].oss-cn-beijing.aliyuncs.com
> User-Agent: curl/7.68.0
> Accept: */*
> content-length:444444444444
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 413 Request Entity Too Large
< Server: AliyunOSS
< Date: Tue, 22 Jun 2021 07:52:22 GMT
< Content-Type: text/html
< Content-Length: 377
< Connection: close
<
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head><title>413 Request Entity Too Large</title></head>
<body bgcolor="white">
<h1>413 Request Entity Too Large</h1>
<p>The requested resource does not allow request data with the requested method or the amount of data provided in the request exceeds the capacity limit.<br/>>Powered by AliyunOSS</p>
</body>
</html>
* Closing connection 0
```

4. 抓包确认

由于客户侧的开发者一直坚持请求是直接请求 OSS，集成 JavaSDK 的客户端代码里，设置的 Endpoint 是 OSS 的地址，是直接请求 OSS 的，而从结果来看请求应该是走了代理的，因此只能从客户端上去抓包确认。抓包命令：

```
tcpdump -i eth0 host BucketName.oss-cn-beijing.aliyuncs.com -s0 -w xxx.pcap
```

从抓包结果看，客户端抓到的包里，目的地址是 10 段开头的内部地址，并不是 OSS 的地址。同时返回信息的 Server 信息是 nginx/1.14.0，这并不是 OSS 的（OSS 的信息是 Server: AliyunOSS）。同时返回头里的 Ali-Swift-Global-Savetime、Via、X-Cache、X-Swift-SaveTime、X-Swift-CacheTime、Eagled 等这些响应头都是阿里 CDN 响应头的特征，由此可见这个请求是客户端先请求到客户侧机房网络的 Nginx 代理服务机，Nginx 代理服务机再把请求转发到阿里 CDN 上，CDN 再去请求 OSS。



5. CDN 限制

经过返回头的 Eagled 和 Via 头去查 CDN 日志可以查到日志，请求的确是走了 CDN，而 CDN 有上传大小限制，上传文件的大小限制为单个文件最大不超过 300 MB，具体可以参考使用限制文档。

解决方案

经过客户确认，机房网络的确是配置了代理，请求走了 CDN。CDN 主要是做静态加速，适用场景是下载业务。上传业务，PUT/POST 请求，都是需要回源的，并没有明显的加速效果，且有上传大小限制，最大支持 300MB。因此建议客户业务侧去做区分，上传业务不走代理和 CDN，而是直接请求 OSS。

适用于

对象存储 OSS

CDN

OSS 服务端签名后直传报错 FieldItemTooLong

作者：胡夫

问题描述

客户参考 OSS 官网 Web 端 PostObject 直传实践的服务端签名后直传的最佳实践，报错如下：

```
<Error>
  <Code>FieldItemTooLong</Code>
  <Message>Your value of form field is too long.</Message>
  <RequestId>60AF5B340FF4C13132EAF670</RequestId>
  <HostId>BucketName.oss-cn-beijing.aliyuncs.com</HostId>
  <MaxSizeAllowed>2097152</MaxSizeAllowed>
  <ProposedSize>2620061</ProposedSize>
</Error>
```

排查过程

1. 查看错误码

由于服务端签名直传是封装 OSS 的 PostObject 接口，查看官网的 PostObject 接口文档，对照错误码 FieldItemTooLong 是表单域的 key 或 value 的大小超出限制。因此，需要分析具体这次异常上传请求的表单域的信息。

The screenshot shows the Aliyun OSS developer console. On the left, there's a sidebar with navigation links like '对象存储 OSS', '基础操作', 'PutObject', etc. The main area has a table titled '错误码' (Error Codes) with columns '错误码' (Code), 'HTTP状态码' (Status Code), and '描述' (Description). One row is highlighted with a red border: 'FieldItemTooLong' (400) - '表单域的key或value的大小超出限制' (The size of the key or value in the form field exceeds the limit). To the right of the table, there are several tabs: '本页目录', '响应头', '响应元素', '示例', '错误码' (which is selected), '附录: Post Policy', '附录: Post Signature', and '更多参考'.

2. 抓包分析

此类由于用户是移动端 App 上传的失败，因此分析表单域的信息还需要从移动端的抓包分析。

通过 Charles 抓包可以看到抓包文件如下。初步看并没有发现问题，在仔细对照后发现表单域里上传文件的域写成了"files"，而不是 file。

This screenshot shows a Charles network traffic capture. It displays an incoming POST request from an application. In the 'Contents' tab, under the 'files' section, there is a file named '6BCEDA6C-784B-4253-9022-B820C34DA490.png'. The file size is listed as 4.90 MB (5,194,408 bytes). Below the file information, the raw XML content of the multipart form-data is shown:

```

<?xml version="1.0" encoding="UTF-8"?>
<Error>
<Code>FieldItemTooLong</Code>
<Message>Your value of form field is too long.</Message>
<RequestId>60AF5B340FF4C13132EAF670</RequestId>

```

This screenshot shows the same Charles network traffic capture, but the 'files' field has been corrected to 'file'. The raw XML content now shows:

```

<?xml version="1.0" encoding="UTF-8"?>
<Error>
<Code>FieldItemTooLong</Code>
<Message>Your value of form field is too long.</Message>
<RequestId>60AF5B340FF4C13132EAF670</RequestId>

```

而根据接口文档的表单域参数说明以及提供的示例都说明了传的字段应该是"file"，而不是"files"



对象存储 OSS					
基础操作		x-oss-meta-*	字符串	否	默认值：无 如果请求中携带以x-oss-meta-为前缀的表单域，则视为user meta，例如 x-oss-meta-location。 ② 说明一个Object可以有多个类似的参数，但所有的user meta总大小不能超过8KB。
DeleteMultipleObjects		x-oss-security-token	字符串	否	如果本次访问是使用STS临时授权方式，则需要指定该项为SecurityToken的值，同时OSSAccessKeyId需要使用与之配对的临时AccessKey Id，计算签名时，与使用普通AccessKey Id签名方式一致。 默认值：无 建议STS服务的具体操作请参见开发指南中的 使用STS临时访问凭证访问OSS 。您可以调用STS服务的 AssumeRole 接口或者使用 各语言STS SDK 来获取临时访问凭证。
PostObject		file	字符串	是	文件或文本内容。浏览器会自动根据文件类型来设置Content-Type，并覆盖用户的设置。OSS一次只能上传一个文件。 默认值：无 注意 file必须是表单中的最后一个域。



```

--9431149156168
Content-Disposition: form-data; name="success_action_status"
200
--9431149156168
Content-Disposition: form-data; name="Content-Disposition"
content_disposition
--9431149156168
Content-Disposition: form-data; name="x-oss-meta-uuid"
uuid
--9431149156168
Content-Disposition: form-data; name="x-oss-meta-tag"
metadata
--9431149156168
Content-Disposition: form-data; name="OSSAccessKeyId"
44CF959006BF252F707
--9431149156168
Content-Disposition: form-data; name="policy"
eyJlcmFnPj0wAwhuJoiMjAxMy0xMi0wMjQxNjowNDowMfoiLCJjb25kaXRpb25zIjpWYjB250ZW50LWxlbmd0a
--9431149156168
Content-Disposition: form-data; name="Signature"
k2oYRv66bmc10+dcGRw5x2PRrk=
--9431149156168
Content-Disposition: form-data; name="file"; filename="Myfilename.txt"
Content-Type: text/plain
abcdefg
--9431149156168
Content-Disposition: form-data; name="submit"
Upload to OSS
--9431149156168--

```

解决方案

表单域文件或文本内容的字段应该是 file，用户传的字段变成了 files，导致真正上传文件的二进制内容在 files 字段里了，因此导致表单大小超了限制。

适用于

对象存储 OSS

OSS 控制台列举文件超时

作者：胡夫

问题描述

OSS 控制台列举文件超时 (listobjects)

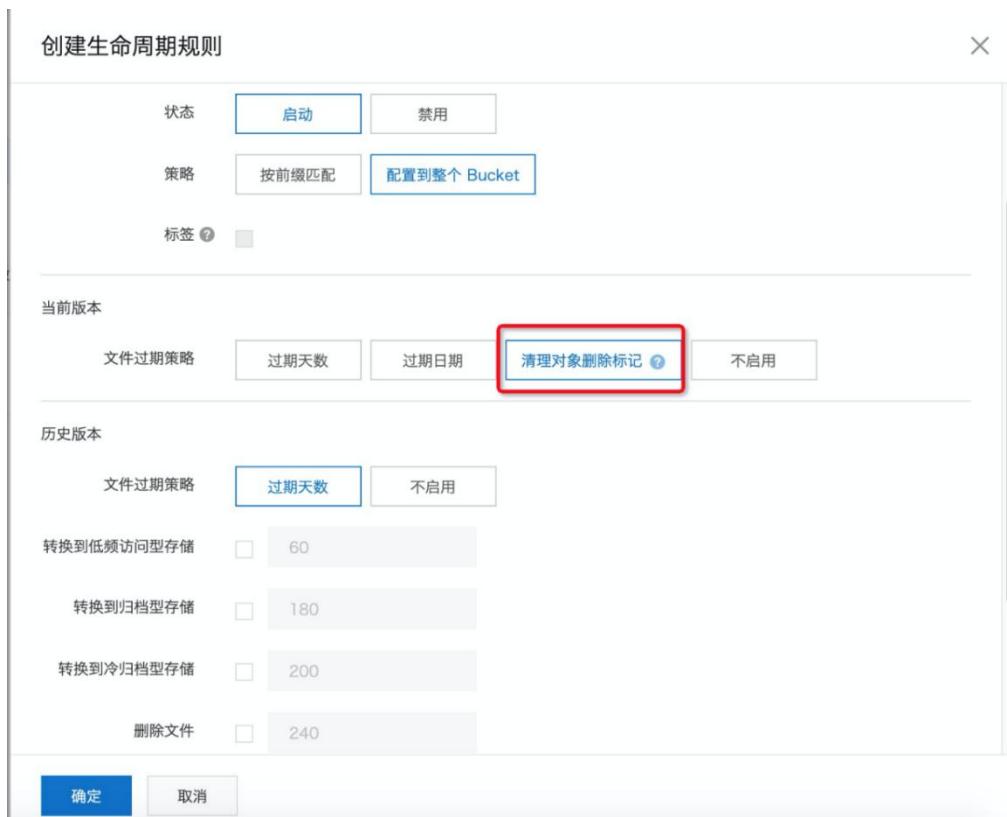


问题原因

确认该 Bucket 是开启了多版本，且存在多个 delete_markar 删除标记，而控制台调用的是 ListObjects 接口，在这种场景下响应速度会变慢（包括 ossbrowser 也有类似问题），参考官网的 FAQ 文档：https://help.aliyun.com/document_detail/188781.html

解决方案

通过 OSS 的生命周期策略去清理对象删除标记



特别说明

生命周期清理对象删除标记的策略，仅用于清理没有历史版本的 DeleteMarker，也称悬空 DeleteMarker。

如果在历史版本没有过期的情况下，则无法过期 DeleteMarker。举例说明如下：

(1) 下图第一个 object

这个 object 只有一个删除标记，没有历史版本，则该 object 的删除标记可以通过生命周期配置的清理对象删除标记策略来删除。

(2) 下图第二个 object

这个 object 除了删除标记以外还有历史版本，则生命周期无法直接删除该删除标记，只有历史版本被删除回收了，生命周期才能删除这些删除标记。这种情况下，如果不需要比较久的历史版本，可以在配置生命周期的时候，配上历史版本的过期删除策略去执行。

Object Name	Last Modified	Size	Storage Class	Operations
111.txt	2021年3月21日16:37:44 (删除标记)	0.0KB	标准存储	详情 彻底删除
1616126489_183.ts	2021年3月21日12:28:52 (删除标记) 2021年3月21日12:28:36 2021年3月21日12:28:29 (删除标记) 2021年3月21日12:17:05	0.0KB 1.373MB 0.0KB 1.373MB	标准存储	详情 彻底删除

另外，如果在生命周期执行期间，又产生了大量删除动作，那么这些删除动作会导致产生了新的“删除标记”。如果你配置了生命周期去删除“删除标记”，但是发现“删除标记”不减少反而增加了，那么可能就是这种情况。

适用于

对象存储 OSS

点播 HLS 加密转码以后生成的 m3u8 只有一个 ts 切片

作者：胡夫

问题描述

片源是 mp4 的视频，时长 8 分钟，可以正常播放。通过点播的 HLS 加密转码以后，生成的 m3u8 里只包含了一个 ts 切片，且播放的时候只播放了几十秒就结束了。

ffprobe

排查这个问题，需要借助 FFmpeg 工具的 ffprobe 来分析视频，看看有什么异常。可以借助 show_packets 来查看多媒体数据包信息或者 show_frames 查看的视频文件中的帧信息。

1. show_packets

```
ffprobe -show_packets input.mp4
```

通过 show_packets 查看的多媒体数据包信息使用标签括起来，其中包含的信息主要为：

字段	说明
codec_type	多媒体类型, 如视频包、音频包等
stream_index	多媒体的stream索引
pts	多媒体的显示时间值
pts_time	根据不同格式计算过后的多媒体的显示时间
dts	多媒体的解码时间值
dts_time	根据不同格式计算过后的多媒体解码时间
duration	多媒体占用的时间值
duration_time	根据不同格式计算过后的多媒体所占用的时间值
size	多媒体包的大小
pos	多媒体所在的文件偏移位置
flags	多媒体包标记, 如关键包与非关键包的标记

2. show_frames

```
ffprobe -show_frames input.mp4
```

通过 show_frames 查看的视频文件中的帧信息使用标签括起来，其中包含的信息主要为：

属性	说明	值
media_type	帧的类型 (视频、音频、字幕等)	video
stream_index	帧所在的索引区域	0
key_frame	是否为关键帧	1
pkt_pts	Frame包的pts	23100
pkt_pts_time	Frame包的pts的时间显示	38.500000
pkt_dts	Frame包的dts	80
pkt_dts_time	Frame包的dts的时间显示	0.080000
pkt_duration	Frame包的时长	20
pkt_duration_time	Frame包的时长时间显示	0.03333
pkt_pos	Frame包所在文件的偏移位置	344
width	帧显示的宽度	544
height	帧显示的高度	960
pix_fmt	帧的图像色彩格式	yuv420p
pict_type	帧类型	I 帧/ P 帧 / B 帧
pkt_size	每帧的大小	

问题分析

1. 分析视频

通过命令分析该视频，并通过-print_format json 保存为 json 文件，示例命令和 json 格式如下：

```
ffprobe -i input.mp4 -show_frames -print_format json > test_frame.json
```

2. 分析 json

通过命令分析 video 帧和 audio 帧，发现有 22911 音频帧，但是只有 4 个视频帧。

```

1  {
2      "frames": [
3          [
4              {
5                  "media_type": "audio",
6                  "stream_index": 1,
7                  "key_frame": 1,
8                  "pkt_pts": 0,
9                  "pkt_pts_time": "0.000000",
10                 "pkt_dts": 0,
11                 "pkt_dts_time": "0.000000",
12                 "best_effort_timestamp": 0,
13                 "best_effort_timestamp_time": "0.000000",
14                 "pkt_duration": 1024,
15                 "pkt_duration_time": "0.023220",
16                 "pkt_pos": "93",
17                 "pkt_size": "550",
18                 "sample_fmt": "fltp",
19                 "nb_samples": 1024,
20                 "channels": 2,
21                 "channel_layout": "stereo"
22             },
23             {
24                 "media_type": "audio",
25                 "stream_index": 1,
26                 "key_frame": 1,
27                 "pkt_pts": 1024,
28                 "pkt_pts_time": "0.023220",
29                 "pkt_dts": 1024,
30                 "pkt_dts_time": "0.023220",
31                 "best_effort_timestamp": 1024,
32                 "best_effort_timestamp_time": "0.023220",
33                 "pkt_duration": 1024,
34                 "pkt_duration_time": "0.023220",
35                 "pkt_pos": "643",
36                 "pkt_size": "1121",
37                 "sample_fmt": "fltp",
38                 "nb_samples": 1024,
39                 "channels": 2,
40                 "channel_layout": "stereo"
41             }
42         ]
43     }
44 }
```

原因总结

源 mp4 视频只有 4 个 video nalu，转码写包时候，音视频包数量差距较大，无法均匀封装在一起，转码时无法正常交织，导致切片显示时间有问题。

视频直播流播放无声音

作者：胡夫

问题描述

视频直播流可以看到画面但是没有声音，需要确认没有声音原因。

排查过程

1. 查看音频头

在直播控制台输入域名、AppName、StreamName 查看实时监控信息，发现没有音频头，只有视频头。

The screenshot shows the 'Real-time Monitoring' section of the live streaming control console. The left sidebar lists various monitoring categories like Direct Broadcast Management, Stream Management, and Resource Monitoring. Under Resource Monitoring, the 'Real-time Monitoring' button is highlighted with a red box. The main content area displays a chart titled '主播到CDN节点的接收音视频帧率' (Video and audio frame rate received by the host to the CDN node), which shows '暂无数据' (No data). At the bottom of the chart area, there is a button labeled '接受头次数' (Accept header count) also highlighted with a red box.

2. 确认编码格式

后台查看监控发现源流的编码信息里，音频编码格式是 G.711,同时跟用户确认音频编码格式，确认推流端的确用的 G.711 音频编码。

```
rtmp_pusher_start1[1950]: cur video info:  
    encode type: H264  
    frame rate: 15  
    bit rate: 4096  
    gop: 4  
    width: 2304  
    height: 1296  
rtmp_pusher_start1[1952]: cur audio info:  
    encode type: G711A  
    sample rate: 8000  
    bit width: 1
```

问题原因

直播服务端目前不支持 G.711 的音频编码格式，支持的格式和相关限制如下。目前直播支持的视频格式为 H.264、H.265,音频格式为 aac。对于 rtmp 协议原生支持的编解码格式,默认也支持 rtmp 推流播放和 fv 播放,但是 hls 不支持,例如音频 mp3。

解决方案

推流端修改音频编码格式。

适用于

视频直播

视频直播播放 ts 切片存在 404

作者：胡夫

问题描述

视频直播流没有断过，但是播放的过程中存在 ts 切片 404 的情况。

排查过程

1. 视频直播域名默认 ts 切片在边缘节点上是保存 300s，可以通过请求 ts 切片地址，返回的 X-Swift-CacheTime 字段确认缓存时间，同时通过 Last-Modified 字段确认 ts 切片的生产时间。

```
< Range: bytes=0-1619165329
< Ali-Swift-Global-Savetime: 1619165329
< Age: 101
< X-Cache: HIT TCP_MEMORY_HIT dirn:-2:-2
< X-Swift-SaveTime: Fri, 23 Apr 2021 08:08:49 GMT
< X-Swift-CacheTime: 300
< Access-Control-Allow-Methods: *
< Access-Control-Allow-Headers: *
< Access-Control-Allow-Origin: *
< Timing-Allow-Origin: *
< EagleId: b788e72f16191654305162445e
<
```

2. 捞取对应时间段 ts 切片的日志，发现同一个 ts 切片，200 的请求和 404 的请求并不在同一个 CDN 节点上。404 的这条请求当时是回源到直播中心返回的 404，由于直播中心的 ts 缓存时间是 60s，而当时请求的时间已经跟 ts 的生成时间超过 60s 了，导致请求到直播中心失败。

解决方案

这种情况是符合预期的，需要了解的知识点就是默认 ts 切片在直播中心缓存 60s，在边缘节点缓存 300s。

适用于

视频直播

视频直播播放 RTS 低延迟直播流报错 1000 错误码

作者：胡夫

问题描述

使用 Web 端 RTS 低延迟播放器播放 artc 协议的低延迟直播流报错，错误码 1000。

artc://push.xxx.com/AppName/StreamName_opus-RTS

排查过程

登陆直播控制台，在数据监控下查看实时监控，发现该路流是纯音频流，没有视频流信息，没有视频头。也就是说，这个源流是纯音频流，而播放器请求该播放地址订阅的流，是同时订阅了音频和视频流。因为视频流不存在，所以视频流订阅超时了，超时时间为 5 秒。

The screenshot shows the 'Real-time Monitoring' section of the live control console. On the left, there's a sidebar with various management options like '直播管理', '工具箱', '域名管理', etc., and 'Real-time Monitoring' is currently selected. The main panel has tabs for '推流监控', '流量带宽', and '质量监控'. It displays a chart for '主播到CDN节点的接收音视频帧率' (Frame rate from host to CDN node receiving audio and video frames), which shows '暂无数据' (No data). Below the chart, there's another section for '接受头次数' (Number of received headers) with the same message.

解决方案

参考官网帮助文档"纯音频与纯视频直播"，通过在播放地址后加上 subvideo=no 参数，用类似如下的地址去播放纯音频流即可。artc://push.xxx.com/AppName/StreamName_opus-RTS?subvideo=no

适用于

视频直播

一个 FLV 直播流起播异常慢的案例

作者：胡夫

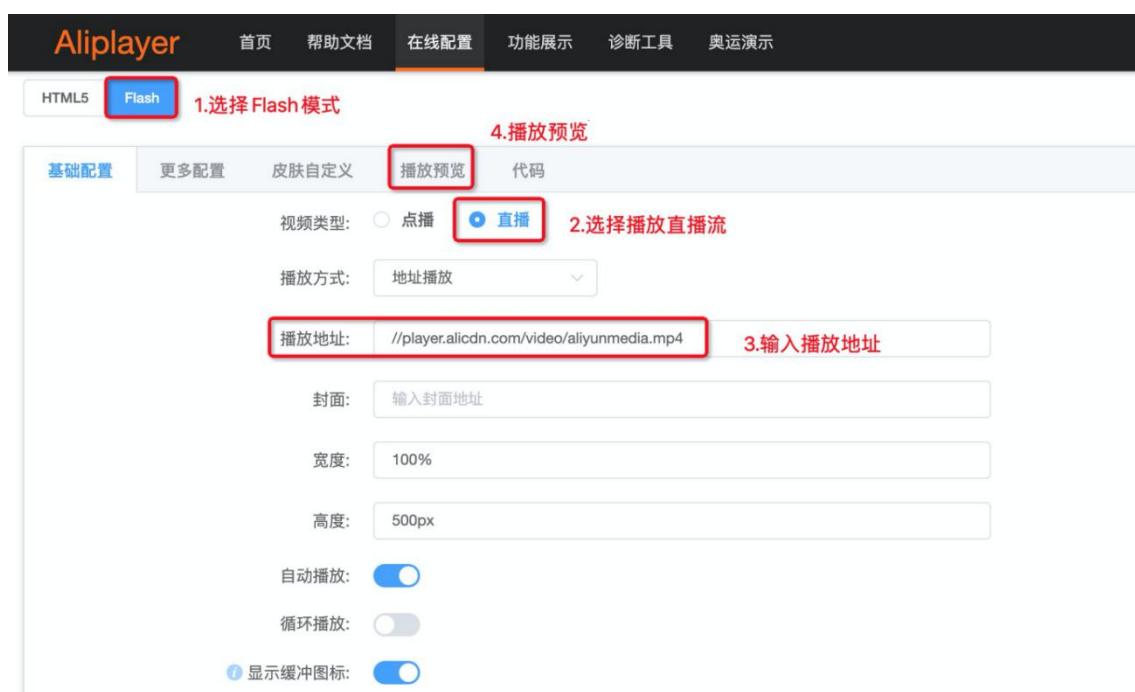
案例现象-H5 播放 FLV 起播慢

阿里云视频直播服务提供了 RTMP/FLV/HLS 三种协议格式的播放地址。本案例遇到的问题是，同一路直播流，使用 H5 播放器播放 RTMP 和 HLS 流正常，而播放 FLV 流则起播速度非常慢，超过 2 分钟才能播放出画面。

使用 Flash 播放器正常

由于 RTMP 和 HLS 是正常的，因此可以排除是 GOP 引起的起播慢。为了对比测试，用 Flash 去测试播放 FLV 流，发现速度非常快。这里推荐两个 Flash 播放器：

(1) 阿里云 Web 播放器 Aliplayer



(2) 第三方 Flash 播放器, 示例 HTML 代码以及播放效果如下:

```
<object>
<embed src="http://www.cutv.com/demo/live_test.swf" width="100%" height="100%"></embed>
</object>
```



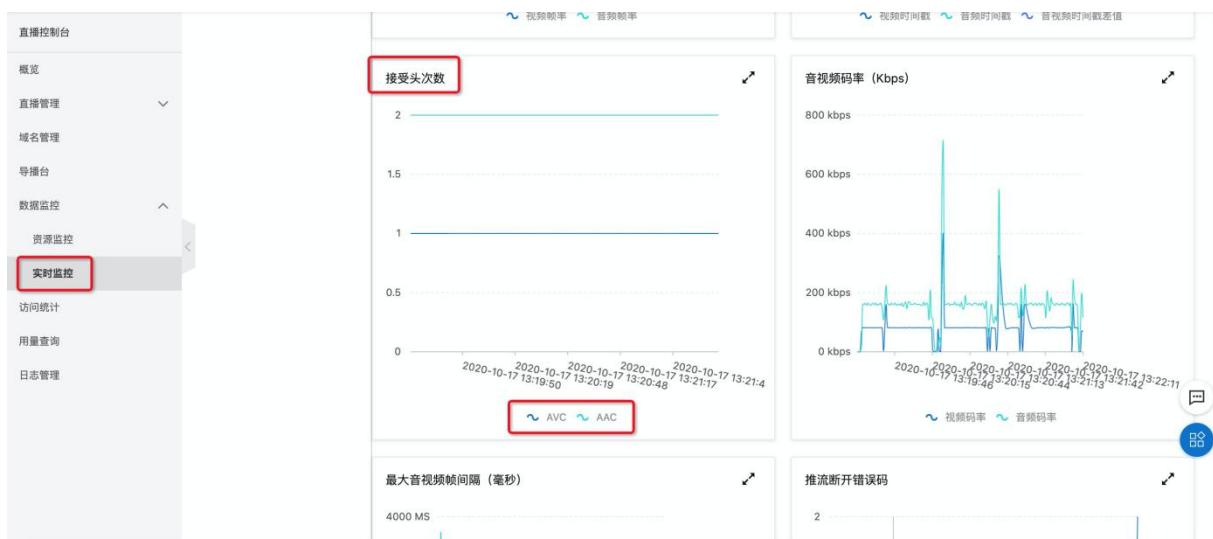
使用 FFplay 播放器异常

为了增加可比性以及分析下直播流音视频格式,因此用 FFMPEG 的播放工具 FFplay 去播放测试,发现同样是播放 FLV 流速度很慢。播放 RTMP 原始流查看 Meta 信息发现该路直播流只有视频信息,无音频信息,具体结果如下图所示:

```
sh-3.2# ffplay rtmp://
ffplay version 4.0.2 Copyright (c) 2018 the FFmpeg developers
  built with Apple LLVM version 9.0.0 (clang-900.0.39.2)
configuration: --prefix=/usr/local/Cellar/ffmpeg/4.0.2 --enable-shared --enable-pthreads --enable-version3 --enable-hardedited-tables --enable-avresample --cc=clang
--host-cflags= --host-ldflags= --enable-gpl --enable-ffplay --enable-libmp3lame --enable-libx264 --enable-libxvid --enable-opencn --enable-videotoolbox --disable-lzma
libavutil      56. 14.100 / 56. 14.100
libavcodec     58. 18.100 / 58. 18.100
libavformat    58. 12.100 / 58. 12.100
libavdevice    58.  3.100 / 58.  3.100
libavfilter     7. 16.100 / 7. 16.100
libavresample   4.  0.  0 / 4.  0.  0
libswscale      5.  1.100 / 5.  1.100
libswresample   3.  1.100 / 3.  1.100
libpostproc    55.  1.100 / 55.  1.100
Input #0, flv, from 'rtmp://'
Metadata:
  displayWidth : 352
  displayHeight: 288
  fps          : 25
  profile       :
  level         :
  Server        : Tengine
  videocodecreal: 0
  author        : bxt
  copyright     :
  description   :
  keywords      :
  rating        :
  title         :
  presetname    : Custom
  creationdate  : Tue Feb 24 17:41:07 2009
  videodevice   : USB 00Z0
  avclevel      : 21
  avcprofile    : 100
  videokeyframe_frequency: 25
  audiodevice   :
  audiochannels : 0
  audioinputvolume: 80
Duration: 00:00:00.00 start: 286027.474000 bitrate: N/A
Stream #0:0: Video: h264 (High), yuv420p(progressive), 352x288, 32 kb/s, 25 tbr, 1k tbn, 50 tbc
[80049.56 M-V: 0.000 TD= 0 AQ= 0 KBS VQ= 11KB SQ= 0B T=0/0 可以看到视频Meta信息，但是没有音频信息]
```

直播实时监控分析

直播服务控制台提供了实时监控，参考阿里云官网帮助文档[查看直播推流实时监控](#)，可以看到直播流的音频头和视频头信息。其中 AAC 是音频头，AVC 是视频头，如果推流端没有推音频和音频头，那么这里的 AAC 监控项数据就会是 0，本案例就是这种情况。



监控项	含义	场景
AVC	服务器端采集的接受主播端推流的 AVC sequence header 的累计次数。	监控主播端推的 AVC sequence header 次数。
AAC	服务器端采集的接受主播推流的 AAC sequence header 的累计次数。	监控主播端推的 AAC sequence header 次数。

结论和解决方案

通过以上测试结果和相关排查可以确认该问题跟不同播放器解码特性有关，结合相关资料进一步分析确认，大部分播放器默认会等 FLV 流的音频，本案例中由于 FLV 流没有音频，导致播放器一直在“傻等”，因此起播非常慢。

目前阿里云直播支持[纯音频和纯视频直播](#)，可以通过在播放 URL 后加 `onlyvideo=1` 参数来实现纯视频直播，加 `onlyaudio=1` 参数来实现纯音频直播。增加 `onlyvideo=1` 以后，阿里云直播服务在 FLV 封装格式中会明确声明没有音频，可以避免播放“傻等”。

推流音频声道变化导致录制文件 H5 上播放异常

作者：苍柏

问题描述

客户使用视频直播，进行推流并产生录制 mp4 文件。但是录制下来的 mp4 文件在 chrome 浏览器上播放时候，拖拽之后会暂停，如果再点击播放的话，会跳转到片头并不会在原位置继续播放。

排查过程

1、搜集客户相关信息

推流地址：rtmp://push-c1.xxxxx.net/recordf/c6541966831622424156053d8b6

录制文件地址：https://oss-live-1.xxxxx.net/record/recordf/c6541966831622424156053d8b6/2021-06-07-14-15-33_2021-06-07-16-05-09.mp4

2、分别使用 VLC 以及 ffplay 播放此 mp4 文件，发现只有在浏览器端播放此 Mp4 文件才会复现客户所说问题。

3、一开始，怀疑此 mp4 文件可能是 meta 存在于文件尾部导致，故使用 mediaCodec 软件排查分析，发现 mp4 结构正常。

4、使用 chrome://media-internals 抓包，然后另外一个标签页拖拽视频，开始复现问题，排查到拖拽暂停时候，chrome H5 播放器下有异常抛出，具体如下：

00:00:01.445	pipeline_buffering_state	{"for_suspended_start":false,"state":"BUFFERING_HAVE_ENOUGH"}
00:00:13.753	info	"Selected video track: []"
00:00:13.754	pipeline_buffering_state	("for_suspended_start":false,"state":"BUFFERING_HAVE_ENOUGH")
00:00:18.256	info	"Selected video track: [1]"
00:00:18.264	pipeline_buffering_state	("for_suspended_start":false,"state":"BUFFERING_HAVE_ENOUGH")
00:00:20.187	event	"kPause"
00:00:20.188	seek_target	2713.350589
00:00:20.188	pipeline_state	"kSeeking"
00:00:20.188	pipeline_state	"kPlaying"
00:00:20.188	video_buffering_state	[{"state":"BUFFERING_HAVE_NOTHING"}]
00:00:20.269	debug	" Detected midstream configuration change PTS:2712927500 Sample Rate: 48000 vs 48000, ChannelLayout: 2 vs 3, Channels: 1 vs 2"
00:00:20.269	error	"Unsupported midstream configuration change! Sample Rate: 48000 vs 48000, Channels: 1 vs 2"
00:00:20.269	error	"audio error during playing, status: PIPELINE_ERROR_DECODE"
00:00:20.269	pipeline_error	"PIPELINE_ERROR_DECODE"
00:00:20.269	pipeline_state	"kStopping"
00:00:20.271	pipeline_state	"kStopped"
00:00:20.311	event	"kPlay"
00:00:38.007	info	"Selected video track: []"

在网上结合 chromium 内核播放端代码来看，可能跟播放过程中声道变化有关，刚开始是双声道，变成了单声道。

结论说明

客户有两段推流，两段推流前后音频声道有变化，导致录制下来的 mp4 文件在 H5 上播放有兼容性问题。

视频直播录制文件长度不足

作者：胡夫

问题描述

用户使用视频直播服务推流 2 个小时，但是录制生成的 mp4 文件只有 7 秒，录制生成的 m3u8 文件正常。

排查过程

rtmp 推流到直播中心以后，HLS 模块会解析每一个 rtmp 数据包，提取出 h264 nal 数据，然后按照 ts 封装格式封装到 ts 文件里面，将 ts 索引到 m3u8 提供直播和录制。如果是录制成 mp4，是通过转封装服务将 m3u8 的 ts 切片转成 mp4 封装。因此 m3u8 录制时长正常，mp4 时长不正常的场景，通常都是 ts 切片有异常导致转封装失败了。进一步分析 m3u8 例索引的 ts 切片，发现录制文件中前两个 ts 只有音频帧，没有视频帧，这种情况会导致转封装 mp4 异常。

解决方案

把这两个异常的 ts 切片删除，然后重新发起转封装任务。后续需要确认推流设备，为什么会在开始推流阶段推的流没有视频帧的情况，推流端需要避免这种情况。

补充说明

1. 直播流的 ts 切片，就是对源流转封装的，编码信息没有改过，编码信息仍旧遵循的源流的，转封装只改变 container，不改变源流信息。因此上文中录制的 m3u8 里的前两个 ts 切片没有视频帧，就是因为源流没有视频帧导致的。

2. 直播录制文件转 mp4 的时候，也只是转了封装，没有转编码信息。
3. 除非用户有配置了直播转码，转码流才涉及到转编码信息。

适用于

视频直播

视频直播截图失败-服务角色问题

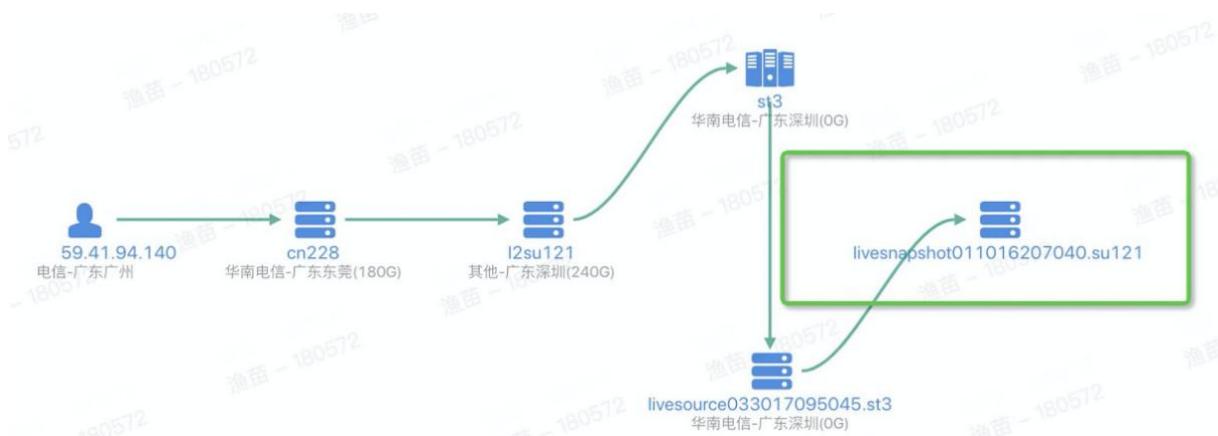
作者：胡夫

问题描述

用户创建了直播截图，但是未产生直播截图文件。

排查过程

1. 查看全链路监控有触发截图



2. 分析天眼 ols 日志

查询命令如下。查到的日志来看 msg=snapshot fail&code=UPLOAD_FAIL_ACCOUNT_EXCEPTION 是上传截图到 OSS 失败。

DomainName and StreamName and livesnapshot

7	2021-03-1 6 16:24:52	msg=uploaded to oss fail &code=UPLOAD_FAIL_ACCOUNT_EXCEPTION&inputUrl=rtmp://33.17.95.45/live/live.yougeqiu.com_14?vhost=live.yougeqiu.com&ali_ols_play=on&aliive_clientnode=livesnapshot011016206208.su121&ossObject=live/14.jpg
8	2021-03-1 6 16:25:02	msg=upload to oss fail &code=UPLOAD_FAIL_ACCOUNT_EXCEPTION&inputUrl=rtmp://33.17.95.45/live/live.yougeqiu.com_14?vhost=live.yougeqiu.com&ali_ols_play=on&aliive_clientnode=livesnapshot011016206208.su121&ossObject=live/14.jpg
9	2021-03-1 6 16:25:02	msg=snapshot fail &code=UPLOAD_FAIL_ACCOUNT_EXCEPTION&taskId=a2bca057-71ec-39bb-a1f9-99c2823c9e2c&subTaskId=f24dd610a3abeab170b4bcc44092b86a&ossObject=live/14.jpg &exceptionMsg=/home/admin/livesnapshot/temp/c1089d39-e624-4a25-8b76-207f57f65de5.jpg
1	2021-03-1 6 16:25:12	msg=upload to oss fail &code=UPLOAD_FAIL_ACCOUNT_EXCEPTION&inputUrl=rtmp://33.17.95.45/live/live.yougeqiu.com_14?vhost=live.yougeqiu.com&ali_ols_play=on&aliive_clientnode=livesnapshot011016206208.su121&ossObject=live/14.jpg
1	2021-03-1 6 16:25:23	msg=upload to oss fail &code=UPLOAD_FAIL_ACCOUNT_EXCEPTION&inputUrl=rtmp://33.17.95.45/live/live.yougeqiu.com_14?vhost=live.yougeqiu.com&ali_ols_play=on&aliive_clientnode=livesnapshot011016206208.su121&ossObject=live/14.jpg
1	2021-03-1 6 16:25:33	msg=upload to oss fail &code=UPLOAD_FAIL_ACCOUNT_EXCEPTION&inputUrl=rtmp://33.17.95.45/live/live.yougeqiu.com_14?vhost=live.yougeqiu.com&ali_ols_play=on&aliive_clientnode=livesnapshot011016206208.su121&ossObject=live/14.jpg
2	2021-03-1 6 16:25:33	msg=upload to oss fail &code=UPLOAD_FAIL_ACCOUNT_EXCEPTION&inputUrl=rtmp://33.17.95.45/live/live.yougeqiu.com_14?vhost=live.yougeqiu.com&ali_ols_play=on&aliive_clientnode=livesnapshot011016206208.su121&ossObject=live/14.jpg

解决方案

以上排查基本可以定位是角色权限问题，若要将视频直播的录制文件和截图文件保存到用户 OSS 的 Bucket 中，需要对直播服务 Live 授权访问 OSS，使用的是 AliyunMTSDefaultRole 角色。可以直接参考官网文档里提供的 权限管理概述 链接去点击授权。

(特别注意需要检查下角色权限对不对，有些情况下可能客户虽然授权过，但是自己修改过这个角色的权限，导致最终还是失败了)

The screenshot shows the 'Video Live' service page. On the left sidebar, under 'Development Guide', 'Authorization Overview' is selected. The main content area displays the following information:

以Describe开头的接口	
AliyunMTSFullAccess	管理媒体转码服务（MTS）的权限 媒体转码服务所有控制台操作和API

Below this, there is a note about custom authorization strategies and a link to 'Authorization Examples'. A callout box highlights the 'Grant Authorization' button for the 'AliyunMTSDefaultRole' role.

适用于

视频直播