HTTPDNS Android接入文档

HTTPDNS原理介绍

HTTPDNS服务的详细介绍可以参见文章全局精确流量调度新思路-HttpDNS服务详解

总的来说，HTTPDNS作为移动互联网时代DNS优化的一个通用解决方案，主要解决了以下几类问题：

- LocalDNS劫持/故障
- LocalDNS调度不准确

HTTPDNS的Android SDK，主要提供了基于HTTPDNS服务的域名解析和缓存管理能力：

- SDK在进行域名解析时，优先通过HTTPDNS服务得到域名解析结果，极端情况下如果HTTPDNS服务不可用，则使用LocalDNS解析结果
- HTTPDNS服务返回的域名解析结果会携带相关的TTL信息，SDK会使用该信息进行HTTPDNS解析结果的缓存管理

HTTPDNS SDK接入步骤

文件拷贝

将HttpDnsLibs目录下的aar包及jar拷贝至项目工程中libs相应位置

HttpDnsLibs目录下包含两个包：

- 文件名以HTTPDNS为前缀的aar包（HTTPDNS_Android_xxxx.aar）为HTTPDNS SDK
- 文件名以beacon为前缀的jar包（beacon-android-xxxx.jar）为灯塔SDK
    - HTTPDNS SDK使用灯塔SDK进行数据上报

aar引入配置

在App module的build.gradle文件中, 添加如下配置

```
android {

    // ...

    repositories {
        flatDir {
            dirs 'libs'
        }
    }
}

dependencies {

    // ...

    implementation(name: 'HTTPDNS_Android_xxxx', ext: 'aar')
}
```

网络安全配置兼容

App targetSdkVersion >= 28(Android 9.0)情况下，系统默认不允许HTTP网络请求，详细信息参见 Opt out of cleartext traffic，Protecting users with TLS by default in Android P

这种情况下，业务侧需要将HTTPDNS请求使用的IP配置到域名白名单中：

- AndroidManifest文件中配置

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application
android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

- xml目录下添加network_security_config.xml配置文件

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="false">182.254.116.117</domain>
        <domain includeSubdomains="false">119.29.29.29</domain>
    </domain-config>
</network-security-config>
```

反混淆配置

```
# 灯塔
-keep class com.tencent.beacon.** {*;}
```

接口调用

以下仅提供简单的接入演示，SDK接口的具体说明请参考接口文档（HttpDnsDoc目录），使用请参考使用Sample（HttpDnsSample目录）

初始化

```
// 以下代码片段演示新版本接口的简单使用，SDK也初步兼容了之前版本的接口，但仍需要进行少量改动，具体说明请参考接口文档（HttpDnsDoc目录）

// 初始化HTTPDNS SDK
// NOTE: ***** 更多配置项及具体说明请参考文档(HttpDnsDoc目录)及使用Sample(HttpDnsSample目录) *****
DnsConfig
    .Builder()
    // 设置日志输出等级
    .logLevel(LOG_LEVEL)
    // 设置AppId，进行数据上报时用于区分业务
    // 从<a href="https://console.cloud.tencent.com/HttpDNS">腾讯云官网</a>申请获得
    .appId(APP_ID)
    // 设置UserId，进行数据上报时区分用户，出现问题时，依赖该Id进行单用户问题排查
    .userId(USER_ID)
    // 自动初始化内置上报通道，即灯塔
    .initBuiltInReporters()
    // 设置DnsId，即HTTPDNS服务的授权Id
    // 从<a href="https://console.cloud.tencent.com/HttpDNS">腾讯云官网</a>申请获得
    .dnsId(DNS_ID)
    // 设置DnsKey，即HTTPDNS服务的授权Id对应的加密密钥
    // 从<a href="https://console.cloud.tencent.com/HttpDNS">腾讯云官网</a>申请获得
    .dnsKey(DNS_KEY)
    // 设置域名解析请求超时时间，单位ms
```

```
        // 默认为1000
        .timeoutMills(TIMEOUT_MILLS)
        .build()
        .let {
            // 初始化HTTPDNS SDK
            DnsService.init(context, it)
        }
```

域名解析

```
// 进行域名解析
// NOTE: ***** 域名解析接口是耗时同步接口，不应该在主线程调用 *****
// useHttp即是否通过HTTP协议访问HTTP服务进行域名解析
// useHttp为true时通过HTTP协议访问HTTP服务进行域名解析，否则通过UDP协议访问HTTP服务进行域名解析
// ipSet即解析得到的IP集合
// ipSet.v4Ips为解析得到IPv4集合，可能为null
// ipSet.v6Ips为解析得到IPv6集合，可能为null
val ipSet = DnsService.getAddrsByName(/* hostname */hostname, /* useHttp */false)
// NOTE: useHttp默认为false
val ipSet = DnsService.getAddrsByName(/* hostname */hostname)
```

接入验证

日志验证

通过观察日志输出，可以确定域名解析接口具体的解析情况

初始化HTTPDNS SDK时，可以调用

```
DnsConfig.Builder /* DnsConfig.Builder. */logLevel(int logLevel);
```

接口来设置日志输出等级

- SDK默认将日志通过logcat输出，日志tag统一使用HTTPDNS
- 日志输出等级设为Log.VERBOSE时，SDK会输出上报数据的信息
    - key为ldns_ip的是LocalDNS的解析结果
    - key为hdns_ip的是HTTPDNS A记录的解析结果
    - key为hdns_4a_ips的是HTTPDNS AAAA记录的解析结果
    - key为a_ips的是域名解析接口返回的IPv4集合
    - key为4a_ips的是域名解析接口返回的IPv6集合

模拟LocalDNS劫持

模拟LocalDNS劫持情况下，如果App能够正常工作，可以证明HTTPDNS已经成功接入

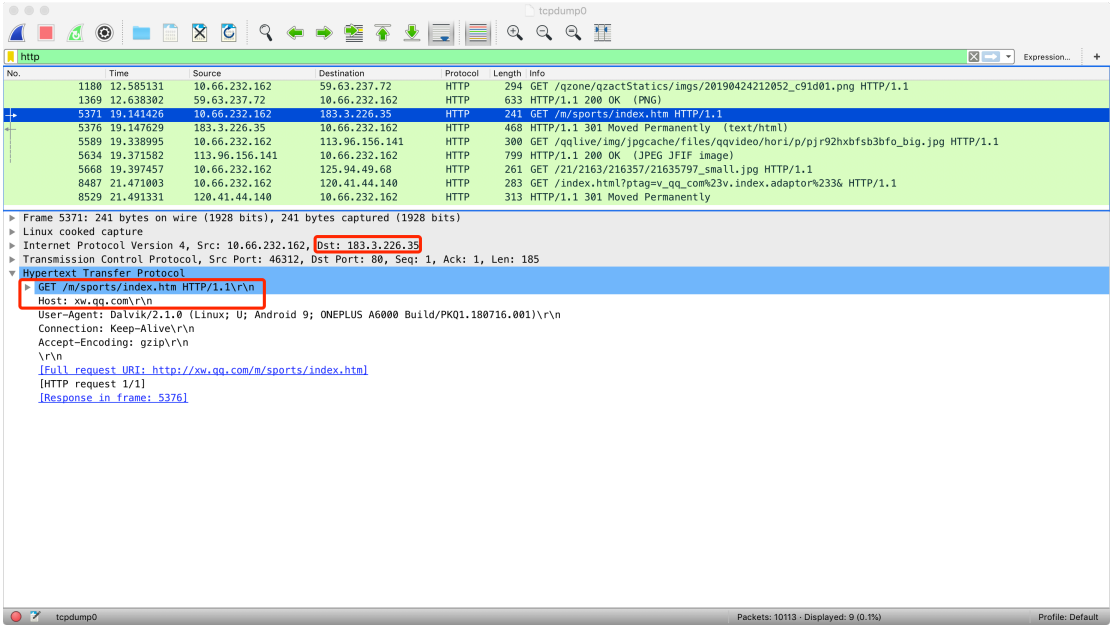
注意：由于LocalDNS存在缓存机制，模拟LocalDNS进行接入验证时，请尽量保证LocalDNS的缓存已经被清理，可以通过重启机器，切换网络等方式，尽量清除LocalDNS的解析缓存；验证时，请注意对照启用LocalDNS和启用HTTPDNS的效果

- 修改机器Hosts文件
  - LocalDNS优先通过读取机器Hosts文件方式获取解析结果
  - 通过修改Hosts文件，将对应域名指向错误的IP，可以模拟LocalDNS劫持
  - Root机器可以直接修改机器Hosts文件
- 修改DNS服务器配置
  - 通过修改DNS服务器配置，将DNS服务器指向一个不可用的IP（如局域网内的另一个IP），可以模拟LocalDNS劫持
  - 机器连接WiFi情况下，在当前连接的WiFi的高级设置选项中修改IP设置为静态设置，可以修改DNS服务器设置（不同机器具体的操作路径可能略有不同）
  - 借助修改DNS的App来修改DNS服务器配置（通常是通过VPN篡改DNS包的方式来修改DNS服务器配置）

抓包验证

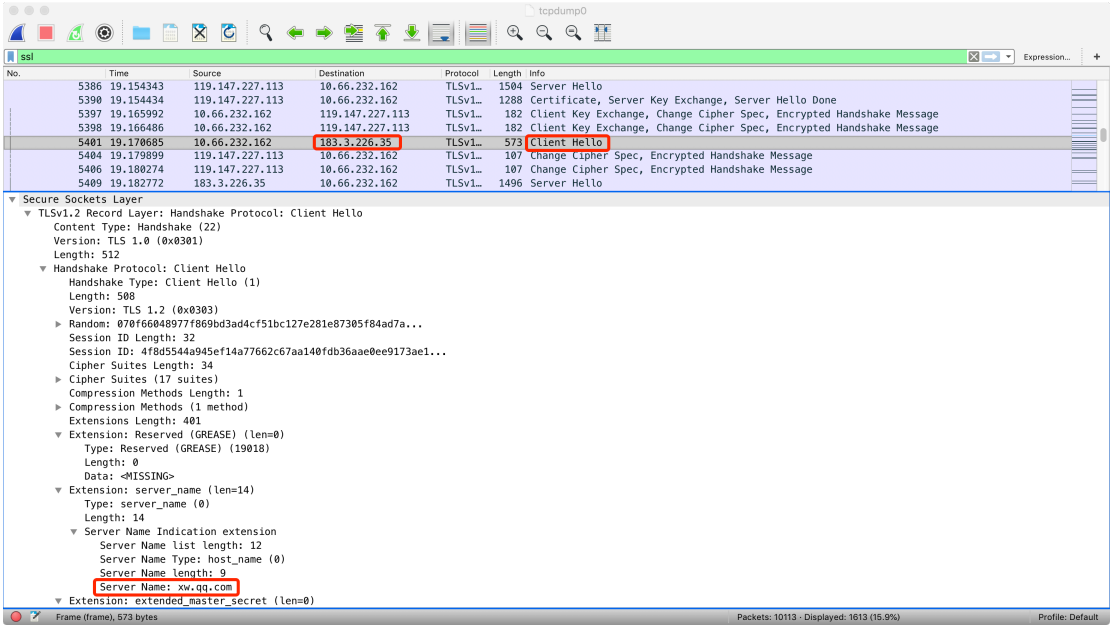
以下以接入HTTP网络访问为例进行说明：

- 使用**tcpdump**进行抓包
  - **注意**，常用的移动端HTTP/HTTPS抓包工具如Charles/Fiddler是通过HTTP代理方式进行抓包，不适用于抓包验证HTTPDNS服务是否生效，相关说明祥见本地使用HTTP代理
  - Root机器可以通过tcpdump命令抓包
  - 非Root机器上，系统可能内置有相关的调试工具，可以获取抓包结果（不同机器具体的启用方式不同）
- 通过WireShark观察抓包结果
  - 对于HTTP请求，我们可以观察到明文信息，通过对照日志和具体的抓包记录，可以确认最终发起请求时使用的IP是否和SDK返回的一致



如上图，从抓包上看，xw.qq.com的请求最终发往了IP为183.3.226.35的服务器

- 对于HTTPS请求，TLS的握手包实际上是明文包，在设置了SNI扩展（详见HTTPS兼容）情况下，通过对照日志和具体的抓包记录，可以确认最终发起请求时使用的IP是否和SDK返回的一致



如上图，从抓包上看，xw.qq.com的请求最终发往了IP为183.3.226.35的服务器

HTTPDNS SDK接入HTTP网络访问实践

将HTTPDNS SDK的域名解析能力接入到业务的HTTP（HTTPS）网络访问流程中，总的来说可以分为两种方式：

- 替换URL中的Host部分得到新的URL，使用新的URL进行网络访问
  - 这种实现方案下，URL丢掉了域名的信息，对于需要使用到域名信息的网络请求，需要做比较多的兼容性工作
- 将HTTPDNS的域名解析能力注入到网络访问流程中，替换掉原本网络访问流程中的LocalDNS实现
  - 这种实现方案下，不需要逐个对请求的URL进行修改，同时由于没有修改URL，不需要做额外的兼容性工作；但需要业务侧使用的网络库支持DNS实现替换
  - 单纯针对DNS替换这个思路，也可以通过Hook系统域名解析函数的方式来实现。但是HTTPDNS SDK内部已经使用了系统的域名解析函数，如果Hook系统域名解析函数可能会造成递归调用直到栈溢出

不同网络库具体的接入方式，可以参见对应的接入文档（当前目录下）及参考使用Sample（HttpDnsSample目录）

替换URL接入方式兼容

如前文所述，对于需要使用到域名信息的网络请求（一般是多个域名映射到同一个IP的情况），我们需要进行额外兼容。以下从协议层面阐述具体的兼容方式，具体的实现方式需要视网络库的实现而定

HTTP兼容

对于HTTP请求，我们需要通过指定报文头中的Host字段来告知服务器域名信息。Host字段详细介绍参见Host

HTTPS兼容

- 我们知道，HTTPS是基于TLS协议之上的HTTP协议的统称，因此对于HTTPS请求，我们同样需要设置Host字段
- 在HTTPS请求中，我们需要先进行TLS的握手。TLS握手过程中，服务器会将自己的数字证书发给我们用于身份认证，因此，在TLS握手过程中，我们也需要告知服务器相关的域名信息。在TLS协议中，我们通过SNI扩展来指明域名信息。SNI扩展的详细介绍参见Server Name Indication

本地使用HTTP代理

本地使用HTTP代理情况下，建议**不要使用**HTTPDNS进行域名解析 以下区分两种接入方式进行分析：

替换URL接入方式

根据HTTP/1.1协议规定，在使用HTTP代理情况下，客户端侧将在请求行中带上完整的服务器地址信息。详细介绍可以参见origin-form 这种情况下（本地使用了HTTP代理，业务侧使用替换URL方式接入了HTTPDNS SDK，且已经正确设置了Host字段），HTTP代理接收到的HTTP请求中会包含服务器的IP信息（请求行中）以及域名信息（Host字段中），但具体HTTP代理会如何向真正的目标服务器发起HTTP请求，则取决于HTTP代理的实现，可能会直接丢掉我们设置的Host字段使得网络请求失败

替换DNS实现方式

以OkHttp网络库为例，在本地启用HTTP代理情况下，OkHttp网络库不会对一个HTTP请求URL中的Host字段进行域名解析，而只会对设置的HTTP代理的Host进行域名解析。这种情况下，启用HTTPDNS没有意义

判断本地是否使用HTTP代理

判断代码如下：

```
val host = System.getProperty("http.proxyHost")
val port = System.getProperty("http.proxyPort")
if (null != host && null != port) {
    // 本地使用了HTTP代理
}
```