

基于 UDP 的可靠简单请求响应协议

一种专门为基于 UDP 的 RPC 通信设计的上层协议，在 UDP 的基础上提供了 ACK 确认、请求重传、重组排序、心跳功能！仅在 UDP 基础上提供可靠服务，是一种两阶段协议，分为请求和响应两个阶段，并且不需要三次握手建立链接过程，主要包括协议头字段、协议流程、相关设置。

1.1 协议头字段

简单请求响应协议（Simple Request-Response Protocol，SR2P 协议），是一种两阶段协议，专门为 RPC 定制，分为请求和响应两个阶段不需要建立链接。

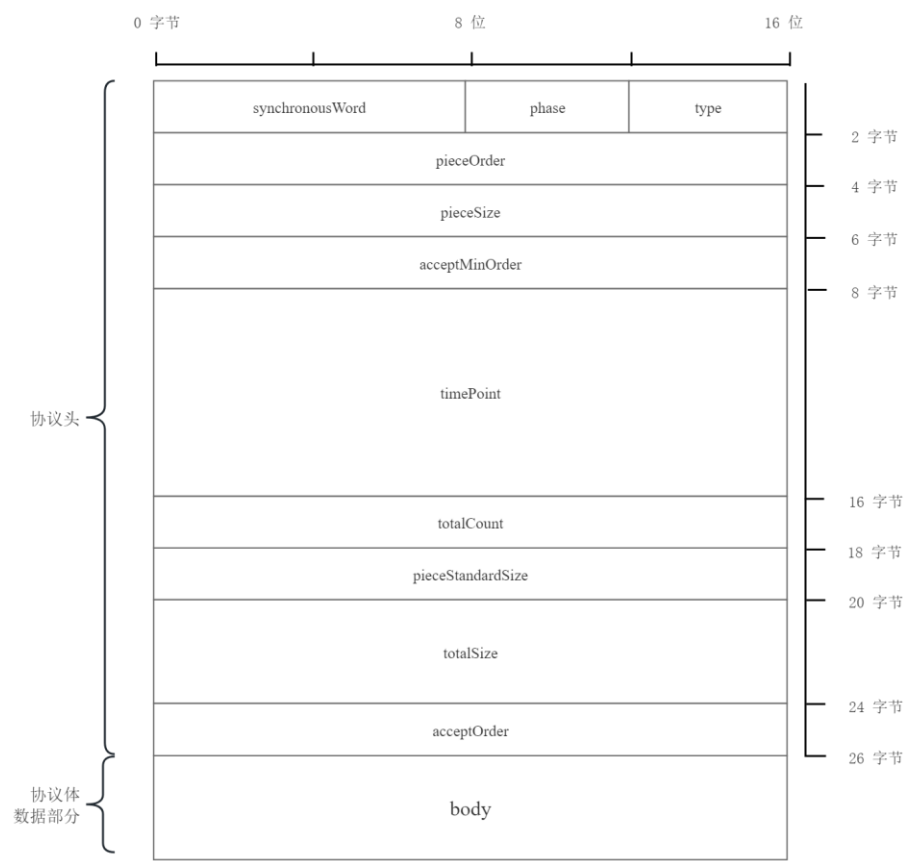


图 1-1 协议字段

协议字段如下所示，协议头是 26 字节，字段字节序采用大端序，数据部分是小端序，由于 MTU 的限制，网络标准 MTU 为 576，考虑 UDP 协议头(8 字节)、IP 协议头(20 字节),最后数据部分最大为 522 字节。如果网络所有节点都支持 MTU

为 1500，数据部分最大为 1472，设数据部分极限值为 defaultPieceLimitSize，简称 dPLS，推荐值为 522，适应任何网络环境。

1.1.1 synchronousWord

同步字，值为 0x11110000，一个字节大小，表示一个报文的开头。

1.1.2 phase

全称为 CommunicationPhase，四位。

- 值为 0x0000，表示 Request 阶段，即请求阶段，该阶段客户端发送请求数据到服务端，服务端负责接受数据，可以理解的 RPC 请求的请求调用阶段。
- 值为 0x0001，表示 Response 阶段，即响应阶段，该阶段服务端发送响应数据到客户端，客户端接受数据，可以理解的 RPC 响应后向客户端返回结果值阶段。

1.1.3 type

全称为 ProtocolType，四位。

- 值为 0x0000，表示 RequestSend，该报文为纯数据报文，意味着协议体里面有数据。
- 值为 0x0001，表示 HearBeat，心跳请求，询问对端是否在线。
- 值为 0x0010，表示 ReceiverACK，接收方确认 ACK，acceptOrder 字段的值告诉发送方已经接受了那些数据分片。
- 值为 0x0011，表示 RequestACK，发送方由于超时未收到 ACK 确认，请求接收端重写发送 ACK。
- 值为 0x0100，表示 TimedOutRequestHeartbeat，如果 RPC 请求需要较长的时间处理，以此来确认对端处于处理状态，结果正在生成，而不是已经离线。
- 值为 0x0101，表示 TimedOutResponseHeartbeat，TimedOutRequestHeartbeat 请求返回此值，表示还在处理中。
- 值为 0x0110，表示 UnsupportedNetworkProtocol，表示协议字段错误，不符合逻辑。
- 值为 0x0111，表示 StateReset，表示状态已经重置，服务端不存在关于次报文的记录，用于回应重放攻击。

- 值为 0x1000，表示 TheServerResourcesExhausted，意味着服务器资源已经耗尽，无法及时处理。

和 Phase 字段配合的交互过程如下表，在 Request 阶段都是 Client 请求 Server，然后 Server 做出反应，Client 会发送如下类型的协议头，Server 端回应类型如下表所示：

阶段	Client(发送方)	Server(接收方)
Request	RequestSend	ReceiverACK
	RequestACK	ReceiverACK
	任何类型但服务端资源耗尽	TheServerResourcesExhausted
	非法数据	UnsupportedNetworkProtocol
	超时未发送剩余数据	StateReset
	重放旧数据	StateReset
	HearBeat	HearBeat

在 Response 阶段都是主要是 Server 将调用结果序列化为报文数据返回 Client，但是如果 RPC 的处理时间较长，甚至是超过预定的等待时间，Client 会发送一个 TimedOutRequestHeartbeat 心跳报文，如果收到 TimedOutResponseHeartbeat，那么客户端会持续等待下一个超时时间，否则会尝试三次请求，无果后，回收资源，向上层表示调用失败，其他情况都是服务端主动向客户端发送数据。

阶段	Client (发送方)	Server(接收方)
Response	TimedOutRequestHeartbeat	TimedOutResponseHeartbeat
	TimedOutRequestHeartbeat	RequestSend

阶段	Server(发送方)	Client(接收方)
Response	RequestSend	ReceiverACK
	RequestACK	ReceiverACK

1.1.4 pieceOrder

分片序号，两字节，表示当前发送的 UDP 报文属于整个数据报的哪个部分。

1.1.5 pieceSize

协议体数据大小，两字节，表示当前分片数据大小，单位为字节。意味着 dPLS 最大值小于 65535。

1.1.6 acceptMinOrder

发送数据端希望收到的 ACK 确认号，2 个字节。

1.1.7 timePoint

唯一确定一个完整报文，表示当前数据属于哪一个数据报文，和客户端 IP/端口号组成一个三元组存储在服务器，唯一确定一次用户请求数据报文，需要客户端维护一个不重数，一般使用时间戳表示。

1.1.8 totalCount

分片数量，两字节，表示一个上层数据包，会被切成多少个分片挨个传输。如果一个 UDP 报文超过 576 字节，那么就需要切片发送，不然不可靠 UDP 传输期间因为网络波动就存在丢包问题，导致报文无法解析。如果报文大小为 S 。如果 $S \leq dPLS$ ，那么 *totalCount* 值就是 1。否则其值采用如下公式 1-1 计算得到。

$$totalCount = S/dPLS + (((S \% dPLS) == 0) ? 0 : 1) \quad (1-1)$$

1.1.9 pieceStandardSize

一个标准分片大小，2 字节，即 dPLS 的值。522 或者 1472。

1.1.10 totalSize

一个完整报文的大小，4 字节。

1.1.11 acceptOrder

ACK 分片确认序号，表示已经收到连续分片的最大分片序号。如收到分片序号 0,1,2,3,4,6,7,9。此时返回 5，表示下一个期待收到的分片序号为 5。

1.2 协议交互流程

协议交互过程分为多种情况，首先是双方都是正常没有差错，还包括有服务端离线、中间分片丢失，心跳超时等。

1.2.1 确定发送分片数量

SR2P 协议会根据发生数据的多少决定每次发生几个数据报，数量规则如下所示：

- $totalCount \leq 3$ ，单次发送一个分片。

- $3 < totalCount$, $totalCount \leq 10$, 单次发送 2 分片。
- $10 < totalCount \leq 36$, 单次发送 3 分片。
- $36 < totalCount$, $totalCount \leq 125$, 单次发送 5 分片。
- $125 < totalCount$, 单次发送 7 分片。

1.2.2 正常流程

正常流程为, 上层收到一个数据包后, 根据其大小等信息, 计算得到各个字段的值, 生成协议头。然后按照 dPLS 值, 将其切片为 $totalCount$ 个 UDP 报文进行传输, 在根据数量规则确定单次发送分片个数。然后向服务端发送分片数据, 然后等待服务端发送 ReceiverACK 确认。

如图 1.2 所示, 开始发送时处于 Request 阶段, 该阶段的超时时间设为 Request_timeout, 由于报文被切分为五个分片。单次发送两个分片数据, 然后等待 $2 * Request_timeout$, 此时不再发送数据, 而是等待 ReceiverACK 确认, 在此期间收到两个 ReceiverACK 确认, 然后发送第三个分片、第四个分片数据, 然后继续等待 ReceiverACK 确认。收到 ReceiverACK 4 以后, 发送最后一个分片数据。最后一个分片数据的 ReceiverACK 收到后, 自动转变为 Reponse 阶段。该阶段的超时时间设为 Reponse_timeout。

此处假设服务器需要花费多一点时间处理请求, 在等待 Reponse_timeout 时间后没有收到回应, 就发送 TimedOutRequestHeartbeat 报文确认是对端无法访问, 还是正在处理数据。如果收到 TimedOutResponseHeartbeat, 那么就再等待 Reponse_timeout 时间, 依此类推。如果服务端已经准备好了响应数据, 那么服务端就不会返回 TimedOutResponseHeartbeat 报文, 而是 RequestSend 报文, 里面会包含响应数据总大小, 分片数量, 当前分片大小等信息, 协议体中包含第一个分片的数据, 图 1-2 中显示总共包含三个分片。

客户端收到后, 给服务端发送 ReceiverACK 确认, 指导服务端发送下一个分片报文, 最后获取全部的数据, 服务端收到 ReceiverACK 3 以后, 等待一个超时时间以后就清楚该次请求相关状态。

解释(一次主动): 在 Reponse 阶段, 一旦服务端完成处理, 生成响应数据以后, 会主动发送 RequestSend 报文给客户端, 但是只会主动一次, 此后的响应数据都需要客户端通过 TimedOutRequestHeartbeat 和 ReceiverACK 去请求服务端将响应数据发送给客户端, 服务端会定时清理响应数据, 每一次客户端, 这是为了防止在服务端放置大量的定时事件, 降低服务端处理能力。

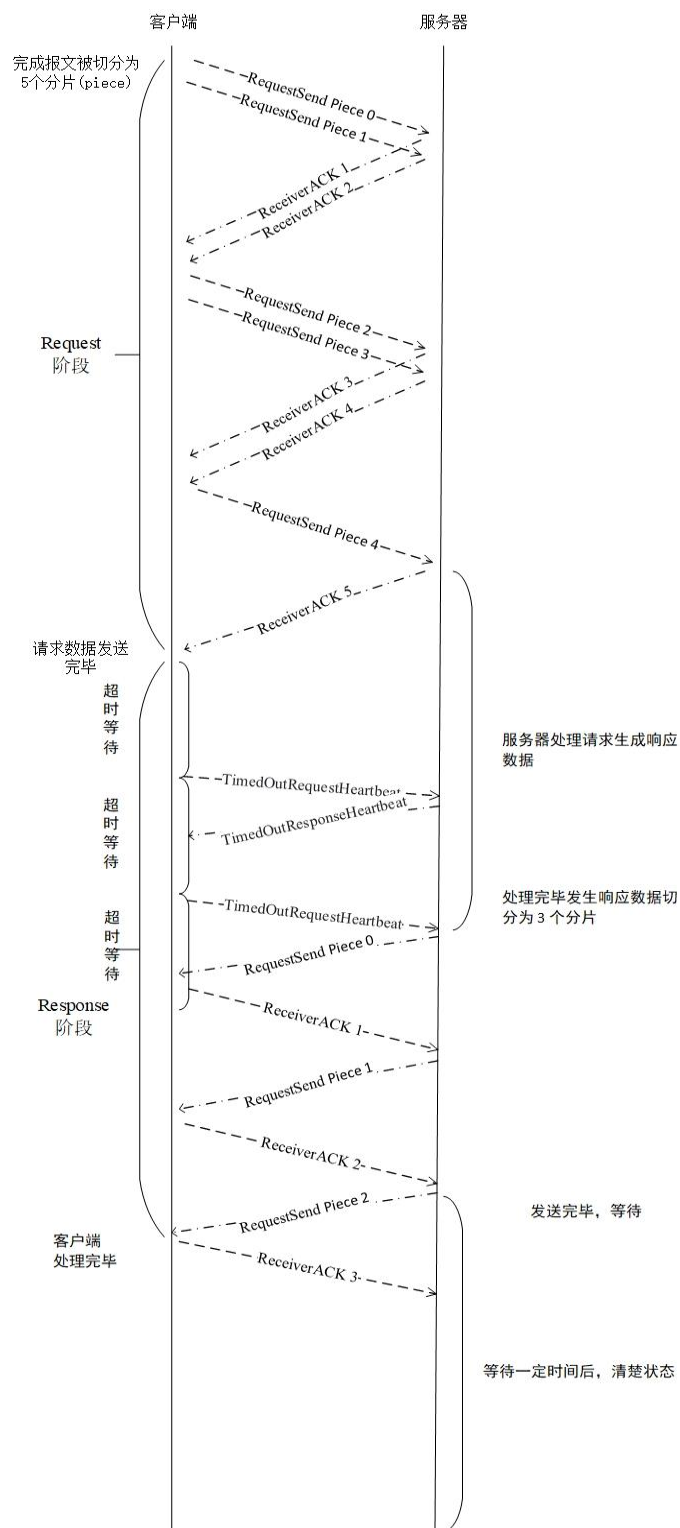


图 1-2 正常流程

服务端完成处理，生成响应数据后，会设置一个定时事件，在一段长为 ResponseClearTimeout 时间后，如果没有来自客户端的 TimedOutRequestHeartbeat

和 ReceiverACK 报文, 就会清除响应数据。如果有, 则会延长 ResponseClearTimeout 时间后再执行清除任务。

1.2.3 服务端不可达、消息丢失

有时候客户端请求的服务端不可达、存在消息丢失, 默认会进行 tryTimes 次请求尝试, 默认情况 tryTimes 等于 3。如果多次请求都无返回, 将直接向上层报错。

如果服务器没有接受到 Piece0、1 的数据, 但是收到 ReqeustACK, 服务端返回 StateReset, 此时客户端从分片 0 重新发送数据。如果接收到了分片数据, 则按规则返回 ReceiverACK。

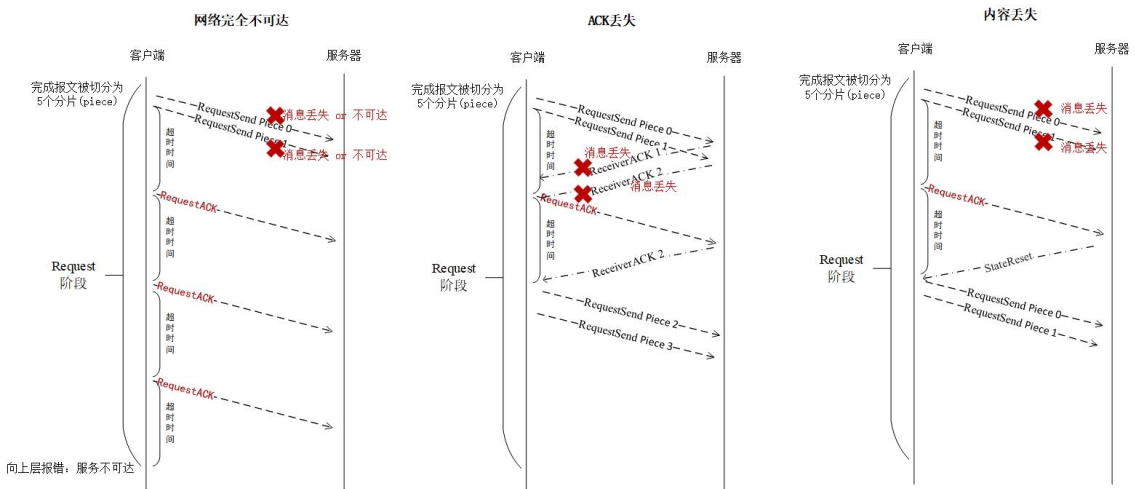


图 1-3 服务端不可达

1.2.4 ACK 丢失

部分 ReceiverACK 丢失可以通过发送方主动请求得以解决, 可能导致冗余数据发送问题, 但是有些 ReceiverACK 丢失并不会有什么影响。如图 1-4 所示即使 ReceiverACK1 丢失了, 也不影响正常发送流程。

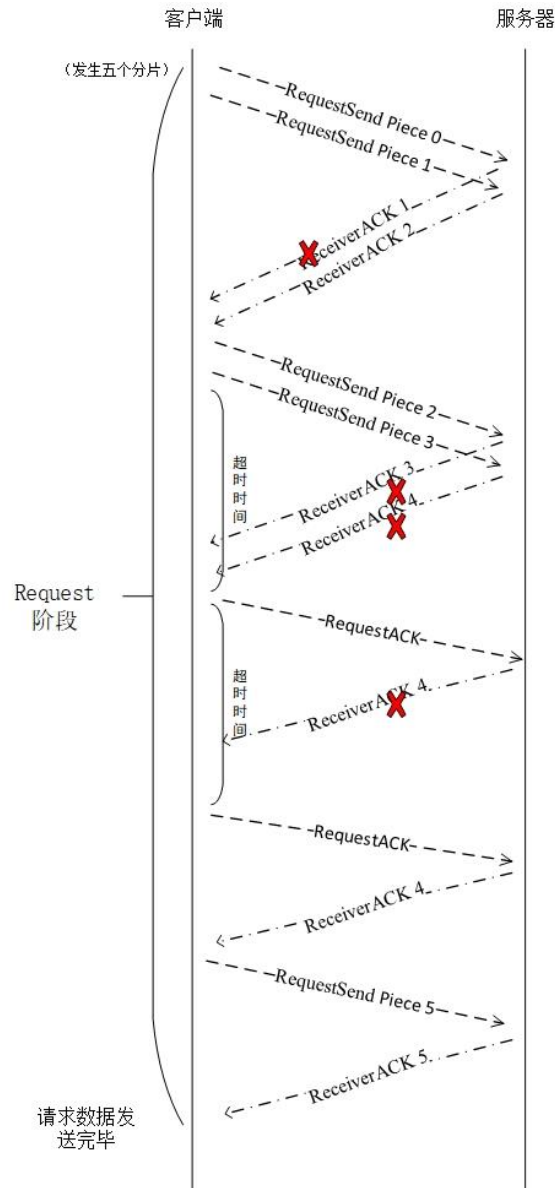
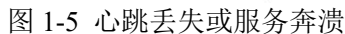


图 1-4 ACK 丢失

1.2.5 TimedOutRequestHeartbeat 超时

在响应阶段,如果服务端处理时间长,需要定期使用 `TimedOutRequestHeartbeat` 进行请求检查,但是可能存在服务奔溃等不可达的问题。如果客户端仍然正常,将会尝试 `tryTimes` 次 `TimedOutRequestHeartbeat` 请求尝试,默认情况 `tryTimes` 等于 3。全部失败后,向上层报错。



如果服务端资源已经耗尽，无法再申请到内存资源，直接返回 `TheServerResourcesExhausted` 报文。

如果协议字段不合理，解析失败，直接返回 `UnsupportedNetworkProtocol`。

报文 `timePoint`、客户端 IP 和端口号组成的三元组标识一个用户请求，

1.3.1 acceptMinOrder

1.3.2 timePoint

客户端 IP/端口号组成一个三元组存储在服务器，唯一确定一次用户请求数据报文。也就是说客户端需要保证在一段时间内发送给服务端不同的请求具有不同

的 timePoint，一般采取递增时间戳实现或者客户端每次启动随机生成一个 64 位整数，然后递增，赋予每一个请求一个独特的 timePoint。

1.3.3 acceptOrder

ACK 确认号，接收端总是返回连续的最大的确认号。如果已经收到分片 0, 1,2,3,4 ,6,7。当收到分片 8 时，acceptOrder 值为 5，如果收到分片 4，返回发送端的 acceptOrder 值同样为 5。