UDT:基于 UDP 的数据传输协议

【摘要】

本文档描述了 UDT 数据传输协议。UDT 被设计为一种用于 TCP 数据传输协议不能高效运行场合下的替代方案。UDT 可有效克服 TCP 在高带宽延迟(BDP)网络环境中的低效率传输问题，这也是开发 UDT 的初衷。另一个重要的应用场景是，允许网络研究人员、学生和应用程序开发人员在 UDT 框架下轻松地实现和部署新的数据传输算法和协议。此外，UDT 也可用于更好地支持防火墙穿透。

UDT 完全构建在 UDP 之上。但是，UDT 是有连接的，支持单播（不支持多播），数据收发全双工。它支持可靠的数据流传输和部分可靠的消息传递。拥塞控制模块是一个开放的框架，可以用来实现和部署不同的控制算法。UDT 采用 AIMD 速率拥塞控制算法作为原生、默认的控制算法。

1 概述

TCP 传输控制协议[RFC5681]已经非常成功，很大程度上促进了如今的互联网繁荣。目前 TCP 仍然贡献了大部分的互联网数据传输。

然而，TCP 不是完美的，它不可能适用于所有的应用场合。在过去的几年里，随着科技的飞速发展，光纤网络性能快速提升，BDP（bandwidth-delay product）不断增加，但是 TCP 数据传输效率往往会随着 BDP 的增长而下降。TCP 本身针对拥塞问题的 AIMD(additive increase multiplicative decrease)算法无法很好地在高带宽网络环境中充分利用网络带宽。理论分析证明，随着 BDP 的增加，TCP 更容易产生数据包丢失[LM97]。

开发 UDT 数据传输协议的原始动机是克服 TCP 在高速广域网上运行效率低的问题。虽然现在已经部署 TCP 变体(例如针对 Linux 操作系统的 BiC TCP [XHR04]和 Windows 操作系统的 Compound TCP [TS06])，但是某些问题仍然存在。例如，目前所有的 TCP 变体都不能有效的解决 RTT 不公平问题，具有更短 RTT 的网络连接将占用更多网络带宽。

此外，随着互联网的不断发展，针对传输协议的挑战和需求将不断涌现。网络领域的研究人员需要一个平台来快速开发和测试新的算法和协议。使用 UDT 可以快速实现涉及传输协议的新思路，尤其是拥塞控制算法，并且在真正的网络环境下进行验证。

最后，在某些应用场景中，UDT 相比 TCP 更有优势。例如，基于 UDP 的传输协议更容易穿越 NAT 防火墙。另一个例子是，TCP 的拥塞控制和可靠性控

制在某些应用中并不是必须的，例如 VOIP、无线通信等应用。开发人员可以使用原生 UDT 或改进后的 UDT 来满足实际应用需求。

基于上述原因和动机，我们认为有必要设计和开发一款基于 UDP 的数据传输协议。

顾名思义，UDT 完全构建在 UDP [RFC768]之上。数据报文和控制报文都使用 UDP 传输。为了方便维护拥塞控制，提供良好的可靠性和安全性，UDT 协议是面向连接的，并且不支持多播。最后，数据可以通过 UDT 实现并发双向传输。

UDT 支持可靠的数据传输和部分可靠的消息传递。数据流语义类似于 TCP，而消息传递语义可以看作是 SCTP 的一个子集[RFC4960]。

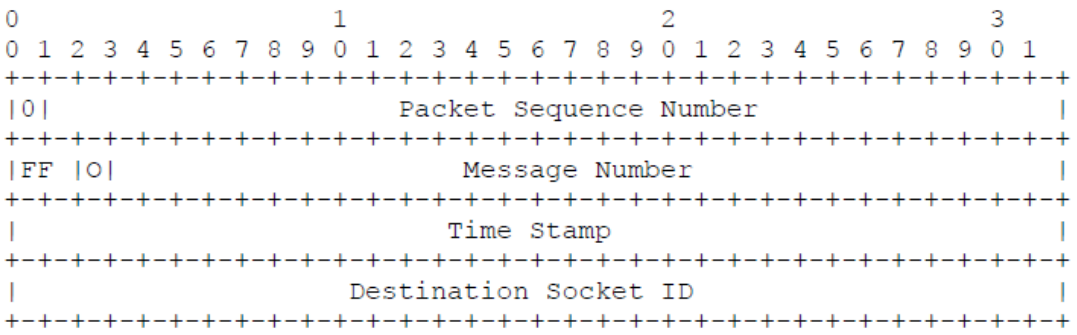
本文档定义了 UDT 的协议规范。详细的功能描述和性能分析可以在[GG07]中找到，在[UDT]中可以找到完整的功能参考实现。

2 数据包结构

UDT 有两种报文:数据报文和控制报文。它们以报文头中第 1 位(标志位)来区分报文类型。

2.1 数据报文

数据报文头部结构如下所示:



数据报文的报头首位为 0。报文序列号（Packet Sequence Number）使用标志位之后的 31 位。UDT 对数据报文进行排序，每发送一个数据报文序列号增加 1。序列号按顺序递增，增加到最大值(2^31- 1)后从 0 开始。

数据报文的报头中第二个长度 32 比特的字段用于消息传递。第一、二位“FF”标记当前报文在一条消息中的位置，“10”标识为首包，“01”标识为尾包，“11”标识为唯一的包，“00”标识为中间的任何包。第三位“O”表示消息传递顺序，“1”

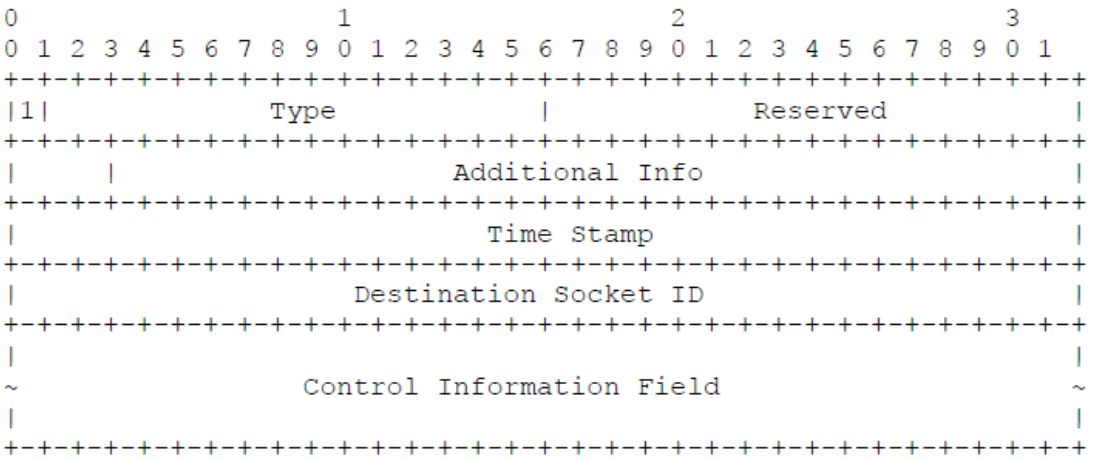
表示该消息必须按顺序方式交付，“0”表示该消息可以按乱序方式交付。其余 29 位是消息编号（Message Number），与包序列号相似（但不相关）。一个 UDT 消息可以由多个 UDT 数据报文组成。

第三个长度 32 比特的字段是时间戳（Time Stamp），用于标识当前数据报文发送时间。时间戳是一个当前时间与建立连接时间的差值。UDT 原生的控制算法不需要使用时间戳信息，在此定义的目的是给用户自定义控制算法提供 32 比特保留字段。

目的套接字 ID（Destination Socket ID）用于 UDP 多路复用。多个 UDT 套接字可以绑定同一个 UDP 端口，UDT 套接字 ID 用于区分不同的 UDT 连接。

2.2 控制报文

如果 UDT 报文的标志位为 1，那么它就是一个控制报文。控制报文数据结构如下所示：



UDT 定义 8 种类型的控制报文，类型（Type）字段由报文头的比特 1-15 标识。后续字段定义取决于控制报文类型。控制报文必须包含前 128 位首部字段，控件信息字段(Control Information Field)可以不存在，具体取决于控制报文类型。

UDT 对 ACK 报文进行排序。每一个 ACK 报文分配一个唯一递增的 32 位编号，独立于数据报文序列号。ACK 编号存放在控制报文中比特位 32-63(“附加信息”)。ACK 编号范围从 0 到 (2^32 - 1)。【原文中存在错误】

类型名称	类型值	附加信息	控制信息
协议 连接 握手	0x0	未定义	1) 32 位:UDT 版本 2) 32 位:Socket 类型（STREAM 或 DGRAM） 3) 32 位:初始数据报文序列号

			4) 32 位:最大报文字节数(包括 UDP/IP 报头) 5) 32 位:最大流量窗口大小 6) 32 位:连接类型(常规、交会) 7) 32 位:socket ID 8) 32 位:SYN cookie 9) 128 位: UDP 套接字的 IP 地址
保活	0x1	未定义	无
应答 (ACK)	0x2	ACK 序列号	1) 32 位:数据报文的序列号，已收到之前的数据包(不包括该序列号) [以下字段是可选的] 2) 32 位:RTT(微秒) 3) 32 位:RTT 方差 4) 32 位:可用的缓冲区大小(以字节为单位) 5) 32 位:报文接收速率(每秒报文个数) 6) 32 位:估计链路容量(每秒报文个数)
否定应答 (NAK)	0x3	未定义	32 位整数数组，包含经过压缩的丢失数据报文信息
关闭	0x5	未定义	无
ACK 应答 (ACK2)	0x6	ACK 序列号	无
消息丢弃请求	0x7	消息 ID	1) 32 位:待丢弃消息的第一个序列号 2) 32 位:待丢弃消息的最后一个序列号
用户 自定义	0x7FFF		根据比特位 16 - 31 解释，为用户定义保留

最后，时间戳和目标套接字 ID 也存在于控制报文中。

3 UDP 多路复用器

UDP 多路复用器用于实现多个并发 UDT 连接共享同一个 UDP 端口。多路复用器根据数据报头中的套接字 ID 将收到的 UDT 数据报文分发给相应的 UDT 套接字。

一个多路复用器用于处理绑定到同一个 UDP 端口的所有 UDT 连接。也就是说，具有不同 UDP 端口的 UDT 套接字将由不同的多路复用器处理。

每个多路复用器维护两个队列：发送队列和接收队列。发送队列中存放至少

有一个报文待发送的套接字。UDT 发送队列中的套接字根据等待发送报文的时间顺序排列。发送队列维持一个高性能的定时器，当队列中的第一个套接字对应的待发送报文等待时间结束，发送数据包并删除该套接字。如果该套接字需要发送更多报文，套接字将被重新插入到队列中。

接收队列读取收到的报文并将它们分派到对应的套接字。如果目标套接字 ID 为 0，则将报文发送到侦听套接字（如果有的话），或发送到一个处于交会连接阶段的套接字（见第 5 节）。

与发送队列类似，接收队列维护一个等待接收数据包的套接字列表。接收队列扫描列表以检查每个套接字是否有计时器过期，扫描时间间隔为 SYN (SYN = 0.01 秒，在第 4 节中定义)。

4 定时器

UDT 使用四个定时器来触发不同的周期性事件。每一个事件有自己的周期，各事件不相关。计时器使用系统时间作为时钟源，如果系统时间更新，定时器也应该进行更新。

对于 UDT 中的某个周期事件 E，假设其时间变量为 ET，周期为 p。如果系统时间 t0（ET = t0）被复位或重置，则在任意时刻 t1 （t1 - ET >= p）事件 E 被触发。

四个定时器分别是 ACK、NAK、EXP 和 SND。定时器 SND 在基于速率进行报文发送时使用（参见第 6.1 节），而其它三个定时器用于报文接收。

定时器 ACK 用于触发 ACK 事件，其定时周期由拥塞控制单元设置。即使拥塞控制单元不需要基于时间的 ACK，UDT 依旧会以周期不大于 0.01 秒的速度发送 ACK 报文。这里，0.01 秒被定义为同步周期（SYN），它会影响 UDT 中使用的其它定时器。

定时器 NAK 用于触发 NAK 事件，其定时周期动态更新，周期值为 4 * RTT+ RTTVar + SYN，其中 RTTVar 是 RTT 样本的方差。

定时器 EXP 用于触发数据报文重传和维护连接状态。其定时周期动态更新，周期值为 N * (4 * RTT+ RTTVar + SYN)，其中 N 为连续超时次数。为了避免不必要的超时，最小阈值（例如 0.5 秒）应该在实现时被使用。

定时器推荐的周期粒度是微秒。除了定时器 SND 之外，其它定时器不需要保持准确的时间。

在本文档后续部分，将使用时间变量的名称表示关联的事件、变量本身或变

量值，具体含义取决于上下文。例如，ACK 既可以表示 ACK 事件，也可以表示 ACK 周期值。

5 连接建立和关闭

UDT 支持两种不同的连接建立方法，传统的客户/服务器模式和交会模式。在交会模式下，双方 UDT 套接字大约在同一时间连接彼此。

UDT 客户端（在交会模式中，两个端点都是客户端）发送握手请求（类型 0 控制报文）到服务器或对等端。握手报文包含以下信息（假设套接字 A 将此握手信息发送给套接字 B）：

1) UDT 版本：用于兼容目的，当前版本是 4。
2) 套接字类型：STREAM（0）或 DGRAM（1）。
3) 初始序号：套接字 A 发送出去的数据报文起始序号，应该为一个随机值。
4) 报文大小：单个传输报文的最大值（含报文头），通常是 MTU 的值。
5) 最大流量窗口大小：这个值不是必须存在的。但是，在当前的参考实现中使用该参数。
6) 连接类型：用于区分连接建立模式和请求/响应。
7) 套接字 ID：客户端 UDT 套接字 ID。
8) Cookie：用于避免 SYN 泛洪攻击[RFC4987]。
9) 对端 IP 地址：B 的 IP 地址。

5.1 客户端/服务器连接建立

一个 UDT 实体首先启动作为服务器（侦听）。服务器接收处理客户端连接请求，为每个新收到的连接请求创建 UDT 套接字。连接到服务器的客户端首先发送一个握手报文，客户端在计时器超时导致建立连接失败前，应该间隔固定的时间不断发送握手报文，直到收到服务器发送的握手响应报文。

当服务器第一次从客户端接收到连接请求时，它根据客户端地址和密钥生成 cookie 值并将其发送回客户端。然后，客户端必须返回相同的 cookie 到服务器。

当服务器接收到握手包并检测到正确的 cookie 后，将报文大小、最大窗口大小与其本身的值进行比较，并将自己的值设置为较小的那个数值。结果值通过响应握手包发送回客户端，同时回传的信息包括服务器版本号和初始序列号。服务器完成此步骤后，就可以开始发送和接收数据。但是，只要接收到同一客户的

进一步握手报文，它就必须返回响应报文。

客户端收到服务器回传的响应信息后，可以开始发送/接收数据，后续收到的握手响应消息应直接忽略。

客户端的连接类型字段应该设置为 1，服务器的响应字段应该设置为-1。

客户端应该确认响应是否来自被请求的目标服务器。

5.2 交会连接建立

在这种模式下，客户端双方在同一时间向对方发送连接请求。初始连接类型设置为 0。客户端一方收到连接请求后，发送应答响应。如果连接类型为 0，则返回响应值为-1；如果连接类型为-1，则返回响应值为-2；如果连接类型为-2，则直接忽略。

握手双方根据 5.1 节定义对握手消息进行相同的检查(包括版本、报文大小、窗口大小等)。此外，参与握手的客户端只接收处理自己发送过连接请求的 IP 地址的报文。常规的 UDT 服务器（侦听）拒绝交会连接请求。

参与握手的客户端在接收到响应值-1 时初始化连接。

当两个端点都部署在防火墙后，不适合开启侦听服务的情况下，采用交会连接建立模式可以提供更好的安全性和可用性。

5.3 关闭

如果连接的 UDT 实体之一被关闭，它将发送关闭消息到对端。对端接收到这个关闭消息后也将被关闭。关闭消息使用 UDP 发送，并且只发送一次，不保证该消息能够被正常收到。如果关闭消息未收到，对端将在连续 16 次 EXP 超时后关闭（见 3.5 节）。然而，16 次 EXP 超时值的总和应该在最小阈值和最大阈值之间。在我们的参考实现中，分别使用 3 秒和 30 秒。

6 数据收发

每个 UDT 实体有两个逻辑单元:发送单元和接收单元。发送单元根据流量控制和拥塞控制发送或重传应用数据。接收单元接收数据报文和控制报文，根据已接收报文和计时器事件发送控制报文。

接收单元负责触发和处理所有的控制事件，包括拥塞控制、可靠性控制以及其它相关的机制。

UDT 总是试图将应用数据打包成固定大小的报文（连接建立过程中协商的最大数据报文大小），除非没有足够的数据可以发送。

我们在[GHG04b]解释了 UDT 数据发送/接收工作原理。

6.1 发送算法

数据结构和变量

1) 发送方丢失列表：用于存放接收方通过 NAK 报文反馈的丢失数据报文序列号，同时也存放由于事件超时而插入的数据报文序列号。报文序列号以递增顺序方式存储。

数据发送算法

1) 如果发送方丢失列表不空，重新发送在列表中第一个报文，并从列表中删除该报文，跳转到 5）。
2) 在消息模式下，如果在发送丢失列表的数据报文超过应用程序指定的生存时间 TTL，发送消息删除请求并从丢失列表中删除所有相关的数据报文，跳转到 1)。
3) 等待直到有应用程序数据需要发送。
4) a)如果未确认的报文数量超过数据流或拥塞窗口大小，等待直到 ACK 到来，跳转到 1）。
   b)封装一个新的数据报文并发送。
5) 如果当前数据包的序列号是 16*n，其中 n 是整数。跳转到 2)。
6) 等待（SND – t）时间，其中 SND 为根据拥塞控制更新的报文发送间隔，t 为 1)至 5)花费的总时间。跳转到 1）。

6.2 接收算法

数据结构和变量

1) 接收方损失列表：它是一个多元组列表，每个表项内包括检测到的丢失数据报文的序列号、最近收到反馈的时间、以及被NAK报文反馈次数k。表项依据数据报文序列号以递增顺序存储。
2) ACK 历史窗口：循环队列记录 ACK 发送时间，空间不足时对最老的表

项进行覆盖处理。

3) PKT 历史窗口：循环队列记录每个数据报文到达时间。
4) 报文对窗口：循环队列记录每个探测报文对之间的时间间隔。
5) LRSN：记录接收数据包的序列号最大值，初始值为初始序列数-1。
6) ExpCount：记录连续 EXP 超时事件次数。

数据接收算法

1) 查询系统时间，检查定时器 ACK、NAK 或 EXP 是否超时。如果存在超时，重置相关的时间变量，按照以下步骤处理超时事件。如果超时事件为 ACK，检查 ACK 报文间隔。
2) 开始 UDP 报文接收，如果固定时间内没有报文到达，转到 1)。
3) 将 ExpCount 重置为 1。如果没有未确认的数据，或者这是一个 ACK 或 NAK 控制报文，重置 EXP 计时器。
4) 检查报文头部的标志位，如果是控制报文，根据其类型信息进行处理，然后跳转到 1）。
5) 如果当前数据报文的序列号是 16n + 1，其中 n 是整数，在报文对窗口中记录当前数据报文与前一数据报文之间的间隔。
6) 在 PKT 历史窗口中记录数据报文到达时间。
7) a)如果当前数据报文的序列号大于 LRSN + 1，把这两个值之间的所有报文序列号写入接收方丢失列表，并且将它们通过 NAK 报文传递给发送方。

   b)如果序列号小于 LRSN，则从接收方的丢失列表中移除。
8) 更新 LRSN，跳转到 1)。

ACK 事件处理

1) 根据规则从收到的所有报文中查找，获取 ACK 序列号。如果接收方丢失列表为空，则 ACK 序列号为 LRSN + 1；否则为接收方丢失列表中最小的序列号。
2) 如果（a） ACK 序列号等于由 ACK2 确认的最大 ACK 序列号，或（b）它等于在最后一个发送 ACK 报文的序列号，并且这两个 ACK 报文之间的时间间隔小于 2 个 RTT，停止（不发送该 ACK 报文）。
3) 给 ACK 报文分配唯一递增的 ACK 编号，写入 ACK 序列号、RTT、RTT 方差、流窗口大小。如果 ACK 事件不是由 ACK 定时器触发的，发送

ACK 报文然后停止。

4) 计算数据报文到达速度。使用 PKT 历史窗口中存储的报文到达时间间隔，计算最近 16 个包到达的中间值（AI），在这 16 个数值中删除大于 AI*8 或不到 AI / 8 的数值。如果还剩下 8 个以上的值，则计算平均值 AI'，数据包到达速度为 1/AI'（每秒包数），否则返回数值 0。
5) 计算估计链路容量（link Capacity）。使用报文对窗口存储的数据报文对间隔时间，计算最后的 16 个数据报文对的中间值（PI），链路容量是 1/PI（每秒包数）。
6) 将报文到达速度和估计链路容量写入 ACK 报文并发送。
7) 在 ACK 历史窗口中记录 ACK 编号、ACK 序列号和 ACK 发送时间。

NACK 事件处理

搜索收件方的丢失列表，找出时间大于 k*RTT 未进行反馈的所有报文，获取报文序列号，其中 k 初始值为 2，此后报文序列号每在 NAK 中反馈一次 k 加 1。压缩（详见第 6.4 节）并将这些序列号通过 NAK 报文传递给发送方。

EXP 事件处理

1) 将所有未确认报文放入发送方丢失列表中。
2) 如果（ExpCount> 16）并且上次 ExpCount 被重置为 1 后过去 3 秒，或者过去 3 分钟，关闭 UDT 连接并退出。
3) 如果发送方的丢失列表是空的，发送一个保活报文给对端。
4) ExpCount 增加 1。

收到 ACK 报文

1) 更新已确认的最大序列号。
2) 发送 ACK2 应答报文，填入收到 ACK 报文的编号。
3) 更新 RTT 和 RTTVar。
4) 将 ACK 和 NAK 周期更新为 4 * RTT + RTTVar + SYN。
5) 更新流窗口大小。
6) 如果这是一个轻量级 ACK，停止。
7) 更新报文到达率:A = (A * 7 + a) / 8，其中 a 为 ACK 报文中的数值。
8) 更新线路预估容量:B = (B * 7 + b) / 8，其中 b 为 ACK 报文中的数值。
9) 更新发送方缓冲区（释放已被应答的缓冲区）。

10) 更新发送方丢失列表（删除已被应答的报文）。

收到 NAK 报文

1) 将 NAK 中的所有序列号添加到发送方丢失列表。
2) 按速率控制更新 SND 周期（详见 3.6 节）。
3) 复位 EXP 时间变量。

收到 ACK2 报文

1) 根据此 ACK2 中的 ACK 编号，在 ACK 历史窗口中定位相关 ACK。
2) 更新已应答最大的 ACK 编号。
3) 根据 ACK2 到达时间和 ACK 发送时间，计算新的 rtt，并将 RTT 值更新为:RTT = (RTT * 7 +rtt)/ 8。
4) 更新 RTTVar 变量值， RTTVar = (RTTVar * 3 + abs(RTT - rtt)) / 4。
5) 更新 ACK 和 NAK 周期为 4 * RTT + RTTVar + SYN。

收到消息丢弃请求

1) 标记在接收缓冲区中属于这个消息的所有数据报文，防止被从接收缓冲区读出。
2) 在接收方丢失列表中删除所有相关报文。

收到“保活”报文

直接忽略。

收到握手/关闭报文

详见第 5 节。

6.3 流控制

流控制窗口最初的大小是 16。
收到 ACK 报文后，流控制窗口大小更新为接收方可用缓冲区大小。

6.4 损失信息压缩方案

NAK 中携带的报文丢失信息是一个 32 位整数数组。如果数组中的整数是一个正常的序列号（第 1 位必须为 0），表示这个序列号的报文丢失；如果第 1 位为 1，意味着所有从该序列号开始到数组中下一个序列号（第 1 位必须为 0）之间的报文丢失，包含这两个序列号指定的报文。

例如，NAK 报文中携带以下信息:

0x00000002, 0x80000006, 0x0000000B, 0x0000000E

表示序列号为 2、6、7、8、9、10、11、14 的报文丢失。

7 可配置拥塞控制(CCC)

UDT 协议中拥塞控制是一个开放的框架，用户易于实现和切换自定义的拥塞控制算法。具体来说，UDT 协议中原生的拥塞控制算法也是在这个框架下实现。

用户定义的算法可以通过调整 UDT 参数来重新定义几个控制例程，当某一事件发生时控制例程将被调用。例如，当接收到 ACK 报文时，控制算法可以增加拥塞窗口的大小。

7.1 CCC 接口

UDT 允许用户访问两个拥塞控制参数：拥塞窗口大小和报文发送间隔。用户可以调整这两个参数来实现基于窗口的控制、基于速率的控制、基于窗口和速率的控制。

此外，还应该公开以下参数：

1) RTT
2) 最大分段/报文大小
3) 估计带宽
4) 最新发送的报文序列号
5) 接收侧报文到达速率

UDT 实现也可以公开其他参数。这些信息可以用于用户定义的拥塞控制算法调整数据报文发送速率。

以下控制事件可以通过 CCC（回调函数）重新定义 。

1) init：UDT 套接字建立。
2) close：UDT 套接字关闭。
3) onACK：收到 ACK。
4) onLOSS：收到 NACK。
5) onTimeout：超时发生。
6) onPktSent：发送数据包。
7) onPktRecv：接收到数据包。

用户还可以在用户定义的控制算法中调整以下参数。

1) ACK 间隔：每个固定数量的数据报文都可以发送一个 ACK，用户可以定义这个时间间隔。如果这个值是-1，那么它意味着不会根据数据报文间隔发送 ACK。
2) ACK 定时器：每隔一定的时间间隔会发送一次 ACK。这在 UDT 中是强制性的。最大、默认的 ACK 时间间隔为 SYN。
3) RTO: UDT 协议中使用 4 * RTT + RTTVar 计算 RTO，用户可以重新定义。

关于 UDT/CCC 的详细描述和讨论详见[GG05]。

7.2 UDT 原生控制算法

UDT 有一个原生、默认的控制算法，如果没有实现和配置用户自定义的算法，可以使用默认的控制算法。UDT 默认的控制算法使用 CCC 实现。

UDT 的原生算法是一种混合拥塞控制算法，因此它可以调整拥塞窗口大小和报文发送间隔。该控制算法使用基于时间的 ACK，ACK 发送间隔为 SYN。

初始拥塞窗口大小为 16，初始报文发送间隔为 0。控制算法从慢启动阶段开始，直到收到第一个 ACK 报文或 NAK 报文。

收到 ACK 包

1) 若当前状态为慢启动阶段，则设置拥塞窗口的大小，CWND = A * (RTT + SYN)，慢启动阶段结束，停止。
2) 拥塞窗口大小(CWND)设置为:CWND = A * (RTT + SYN) +16。
3) 下一个 SYN 周期需要增加发送的数据报文数量(inc)的计算方法为:

   if(B <= C) inc = 1 / PS;

   else inc = max(10^(ceil(log10((B-C)* PS * 8)))*Beta/PS,1/PS);

其中 B 为链路估计容量，C 为当前发送速度。所有这些都被计算为每秒包数。PS 是 UDT 数据报文大小（按字节计数）。Beta 是常数值 0.0000015。

4) SND 周期更新为：

SND = (SND * SYN) / (SND * inc + SYN)

这四个参数用于降低速率，参数名称以及它们的初始值（括号中）:AvgNAKNum(1)、NAKCount(1)、DecCount(1)，LastDecSeq(初始序列号- 1)。

我们将拥塞周期定义为两个 NAK 事件之间的周期，这两个 NAK 中报文丢失序列号最大值应大于 LastDecSeq，其中 LastDecSeq 为最近一次报文发送速率降低时报文序列号中的最大值。

AvgNAKNum 是所有拥塞期间收到 NAK 数量的平均值。

NAKCount 是当前拥塞期间 NAK 的当前数量。

收到 NAK 报文

1) 若处于慢启动阶段，设置报文发送间隔为 1/recvrate，慢启动结束，停止。
2) 如果这个 NAK 开始新的拥塞周期，则增加报文发送间隔为 snd = snd * 1.125；更新 AvgNAKNum，重置 NAKCount 为 1，然后生成 DecRandom（1 和 AvgNAKNum 之间的随机数），更新 LastDecSeq，停止。
3) 如果（DecCount <= 5）&&（NAKCount == DecCount * DecRandom）；
   a) 更新 SND 周期:SND = SND * 1.125;
   b) DecCount 增加 1;
   c) 记录当前发送的最大序列号（LastDecSeq）。

UDT 协议原生控制算法是为高 BDP 网络环境下批量数据传输而设计的。(GHG04a)

8 安全注意事项

UDT 的安全机制与 TCP 类似。TCP 的大部分应对安全攻击的方法也应用在 UDT 协议中。

9 IANA 的考虑

本文档没有针对 IANA 的考虑。

【参考文献】

[RFC768] J. Postel, User Datagram Protocol, Aug. 1980.

[RFC4987] W. Eddy, TCP SYN Flooding Attacks and Common Mitigations.

[GG07] Yunhong Gu and Robert L. Grossman, UDT: UDP-based Data Transfer for High-Speed Wide Area Networks, Computer Networks (Elsevier). Volume 51, Issue 7. May 2007.

[GG05] Yunhong Gu and Robert L. Grossman, Supporting Configurable Congestion Control in Data Transport Services, SC 2005, Nov 12 - 18, Seattle, WA, USA.

[GHG04b] Yunhong Gu, Xinwei Hong, and Robert L. Grossman, Experiences in Design and Implementation of a High Performance Transport Protocol, SC 2004, Nov 6 - 12, Pittsburgh, PA, USA.

[GHG04a] Yunhong Gu, Xinwei Hong, and Robert L. Grossman, An Analysis of AIMD Algorithms with Decreasing Increases, First Workshop on Networks for Grid Applications (Gridnets 2004), Oct. 29, San Jose, CA, USA.

[LM97] T. V. Lakshman and U. Madhow, The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss, IEEE/ACM Trans. on Networking, vol. 5 no 3, July 1997, pp. 336-350.

[RFC5681] Allman, M., Paxson, V. and E. Blanton, TCP Congestion Control, September 2009.

[RFC4960] R. Stewart, Ed. Stream Control Transmission Protocol. September 2007.

[TS06] K. Tan, Jingmin Song, Qian Zhang, Murari Sridharan, A Compound TCP Approach for High-speed and Long Distance Networks, in IEEE Infocom, April 2006, Barcelona, Spain.

[UDT] UDT: UDP-based Data Transfer, URL http://udt.sf.net.

[XHR04] Lisong Xu, Khaled Harfoush, and Injong Rhee, Binary Increase Congestion Control for Fast Long-Distance Networks, INFOCOM 2004.