中华人民共和国通信行业标准

流控制传送协议(SCTP)

YD/T 1194-2002

前　言

本标准是根据 RFC 2960(2000)建议制定的，它规定了流控制传送协议(SCTP)所使用的消息格式编码和程序。SCTP 协议主要用于在 IP 网中传送 PSTN 的信令消息，同时 SCTP 协议还可以用于其他的信息在 IP 网内传送。

本标准的附录 A、附录 B 和附录 C 是资料性的附录。

本标准由信息产业部电信研究院提出并归口。

本标准起草单位：信息产业部电信传输研究所
深圳市中兴通讯股份有限公司
华为技术有限公司
上海贝尔有限公司

本标准主要起草人：吕　军　续合元　张　宜　高　峰　连　超　林　铭　吕　严

中华人民共和国信息产业部　2002-06-21 发布　2002-06-21 实施

1　范围

本标准规定了流控制传送协议(SCTP)所使用的消息格式编码和程序，SCTP 协议主要用于在 IP 网中传送 PSTN 的信令消息和 IP 网内的信令消息。

本标准主要适用于完成 No.7 信令与 IP 网互通的信令网关(SG)设备，以及 IP 网用于呼叫控制的软交换(Soft-Switch)交换机等设备的开发、生产、引进和购买。

2　规范性引用文件

下列文件中的条款通过在本标准中引用而成为本标准的条款，凡是注日期的引用文件，其随后所有的修改单(不包括勘误的内容)或修订版均不适用于本部分，然而鼓励根据本部分达成协议的各方研究是否可以使用这些文件的最小版本。凡是不注日期的引用文件，其最新版本适用于本标准。

RFC 793　传送控制协议(TCP)
RFC 1191　发现通路 MTU
RFC 1123　对 Internet 主机的要求—应用和支持
RFC 1700　分配的号码
RFC 1750　出于安全目的的随机建议
RFC 1981　用于 IPv6 的发现通路 MTU
RFC 1982　串号的算法
RFC 2373　IPv6 的地址结构
RFC 2401　Internet 协议的安全结构
RFC 2460　Internet 协议，版本 6
RFC 2481　明确的拥塞通知
RFe 2581　TCP 拥塞控制
RFC 2960　流传送控制协议(SCTP)

3　名词术语和缩略语

3.1　定义

SCTP 偶联：SCTP 偶联实际上是在两个 SCTP 端点间的一个对应关系，它包括了两个 SCTP 端点、以及包括验证标签和传送顺序号码等信息在内的协议状态信息，一个偶联可以由使用该偶联的 SCTP 端点用传送地址来惟一识别，在任何时候两个 SCTP 端点间都不会有多于一个的偶联。

SCTP 端点：SCTP 端点是 SCTP 分组中逻辑的接收方或发送方，在一个多归属的主机上，一个 SCTP 端点可以由对端主机表示为 SCTP 分组可以发送到的一组合格的目的地传送地址，或者是可以收到 SCTP 分组的一组合格的起源传送地址。一个 SCTP 端点使用的所有传送地址必须使用相同的端口号，但可以使用多个 IP 地址。SCTP 端点使用的传送地址必须是唯一的。

流：流是从两个 SCTP 端点建立的一个单向逻辑通道，对于顺序递交业务，在这个通道中所有的用户消息都必须按照顺序进行递交。

传送地址：传送地址是用网络层地址、传送层协议和传送层端口号定义的，当 SCTP 在 IP 上运行时，传送地址就是由 IP 地址和 SCTP 端口号的组合来定义的，这里 SCTP 就充当传送协议。

3.2　缩略语

MAC	消息鉴权码
MTU	最大的传送单元
RTO	重发超时
RTT	双向传播时间
RTTVAR	双向传播时间变化
SCTP	流控制传送协议
SRTT	平滑的双向传输时间
TCB	传送控制块
TCP	传送控制协议
TLV	类型-长度-取值的编码格式
TSN	传送顺序号
ULP	高层协议

4　SCTP 的功能描述

信令传送中应用的 SCTP 协议主要用来在无连接的网络上传送 PSTN 信令消息，该协议可以用来在 IP 网上提供可靠的数据传送协议，SCTP 具有如下功能。

—在确认方式下，无差错、无重复地传送用户数据；

—根据通路的 MTU 的限制，进行用户数据的分段；

—在多个流上保证用户消息的顺序递交；

—将多个用户的消息复用到一个 SCTP 的数据块中；

—利用 SCTP 偶联的机制(在偶联的一端或两端提供多归属的机制)来提供网络级的保证；

—SCTP 的设计中还包含了避免拥塞的功能和避免遭受泛播和匿名的攻击的功能。

4.1　SCTP 的结构

SCTP 位于 SCTP 用户应用和无连接网络业务层之间，这种无连接的网络可以是 IP 网络或者其他的网络。本标准规定的 SCTP 协议主要是运行在 IP 网络上的。SCTP 协议通过在两个 SCTP 端点间的建立的偶联，来为两个 SCTP 用户之间提供可靠的消息传送业务。

SCTP 实际上是一个面向连接的协议，但 SCTP 偶联的概念要比 TCP 的连接具有更广的概念，SCTP 协议提供了在两个 SCTP 端点间的一组传送地址之间建立偶联的方法，通过这些建

立好的偶联，SCTP 端点可以发送 SCTP 分组。一个 SCTP 偶联可以包含用多个可能的起源 / 目的地地址的组合，这些组合包含在每个端点的传送地址列表中。

图 1 给出了 SCTP 偶联在 IP 网络协议中的示意。

图 1　SCTP 偶联的示意

4.2　SCTP 的功能

SCTP 传送业务可以分解成如图 2 所示的如下几个功能块，各功能块的用途在下节介绍。

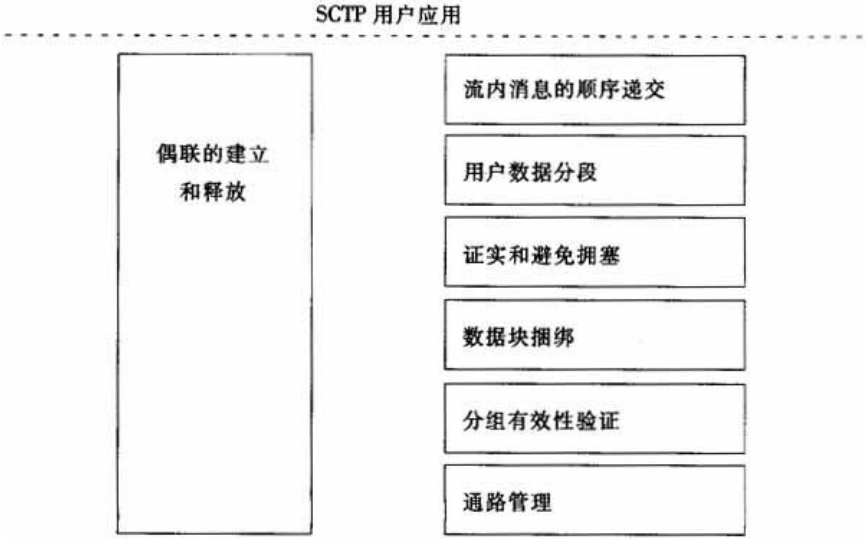

图 2　SCTP 传送业务的功能图

4.2.1　偶联的建立和释放

偶联的建立是由 SCTP 用户发起请求来启动的，出于安全性考虑，为了避免遭受恶意的攻击，在偶联的启动过程中采用了 COOKIE 机制。

SCTP 提供了对激活偶联的正常的关闭程序，它必须根据 SCTP 用户的请求来执行，当然 SCTP 也提供一种非正常的关闭程序(即：中止程序)，中止程序的执行既可以根据用户的请求来启动，也可以由 SCTP 协议检出差错来中止。

SCTP 不支持半打开状态(类似 TCP)，即一端可以在另一端结束后继续发送数据。无论是哪个端点执行了关闭程序后，偶联的两端都应停止接收从用户发来的新数据，并且只传送队列中的数据。

4.2.2　流内消息的顺序递交

SCTP 中的流用来指示需要按顺序递交到高层协议的用户消息的序列，在同一个流中的消息需要按照其顺序进行递交。

SCTP 用户可以在偶联建立时规定在一个偶联中所支持的流的数量，这个数量是可以进行协商的，用户消息通过流号来进行关联。在 SCTP 内部，为每个通过 SCTP 的 SCTP 用户消

息都分配一个流顺序号码。在接收端，SCTP 保证在给定购流中，消息可以按照顺序递交给 SCTP 用户。但当某个流由于等待下一个连续的用户消息造成闭塞时，其他流上的顺序递交不应受影响。

SCTP 也提供非顺序递交的业务，接收到用户消息可以使用这种方式立即递交到 SCTP 用户，而不需要保证其发送时的顺序。

4.2.3 用户数据分段

在需要的时候,SCTP 在发送用户消息时可以对消息进行分段,以确保发送到低层的 SCTP 分组长度挎合通路 MTU 的要求。在接收方，需要把各分段重组成完整的消息后，再把消息递交给 SCTP 用户。

4.2.4 证实和避免拥塞

SCTP 为每个用户数据分段或未分段的消息都分配一个传送顺序号码(TSN)，TSN 的分配是独立于流一级分配的流顺序号码。接收方对所有收到的 TSN 进行证实，尽管此时在接收序列中可能存在接收到的 TSN 不连续。采用这种方式，可以使可靠的递交功能可以与流的顺序递交相分离。

证实和拥塞避免功能可以在规定时间内没有收到证实的时候负责对分组的重发。分组的重发功能可以通过与 TCP 协议类似的拥塞避免程序来调节的。

4.2.5 数据块捆绑

SCTP 分组在发送到低层时要包含一个公共的分组头，其后跟着一个或多个数据块。每个数据块中既可以包含用户数据，也可以包含 SCTP 控制信息。SCTP 用户具有一个选项，可以请求是否把多于一个的用户消息捆绑在一个 SCTP 分组中进行发送。SCTP 的这种数据块捆绑的功能可以在发送端生成千个完整的 SCTP 分组，在接收端负责分解该 SCTP 分组。

当拥塞出现的时候，尽管用户可能请求 SCTP 不必进行捆绑，但 SCTP 的实施仍旧可以执行捆绑功能。用户禁止进行捆绑只会影响到 SCTP 实施，即在传送 SCTP 分组之前产生一个较小的时延。

4.2.6 分组的有效性验证

每个 SCTP 公共分组头中都包含一个必备的验证标签字段和一个 32bit 长的校验字段。验证标签的值由偶联的端点在偶联启动时选择，如果收到的分组中未包含期望的验证标签值，则舍弃该分组。校验码则由 SCTP 分组的发送方设置，以提供附加的保护，用来避免由网络造成的数据差错。接收方对包含无效校验码的 SCTP 分组予以丢弃。

4.2.7 通路管理

发送方的 SCTP 用户能够使用一组传送地址作为 SCTP 分组的目的地。SCTP 通路管理功能可以根据 SCTP 用户的指令和当前合格的目的地集合的可达性状态，为每个发送的 SCTP 分组选择一个目的地传送地址。当用分组业务量不能完全表明可达性时，通路管理功能可以通过心跳消息来监视到某个目的地地址的可达性,并当任何远端传送地址的可达性发生变化时向 SCTP 用户提供指示。通路管理功能也用来在偶联建立时，向远端报告合格的本地传送地址集合，并且把从远端返回的传送地址报告给本地的 SCTP 用户。

在偶联建立后,需要为每个 SCTP 端点都定义一个首选通路,用来在正常情况下发送 SCTP 分组。

在接收端，通路管理功能在处理 SCTP 分组前，用来验证入局的 SCTP 分组属于的偶联是否存在。

5 SCTP 原语定义

SCTP 通过接收高层协议(SCTP 用户)发送的原语请求，为 SCTP 的用户提供服务，并且应当可以根据不同事件由 SCTP 向 SCTP 用户发送通知。

本节章描述的原语和通知可以作为实施 SCTP 协议的一个参考，以下描述的高层协议接口原语功能只是一个说明性的内容，并不要求完全按照以下的说明来实现 SCTP 协议。不同的 SCTP 实施可以具有不同的 ULP 接口，但是 SCTP 必须提供一个最小集的业务，用来保证所有的 SCTP 实施都可以支持相同的协议。

5.1　高层协议(ULP)向 SCTP 发送的请求原语

本节对 ULP 和 SCTP 之间的功能进行归纳，使用的描述方法采用了高级程序语言所用的过程和函数调用方法。

以下描述的 ULP 原语规定了在 SCTP 必须支持的用于互相通信的基本功能。单独的实施必须按照各自格式进行，对原语详细定义，它可以是在一个调用中使用这些基本功能的子集或者是用这些基本功能的组合来实现。

以下的原语描述使用了如下格式：原语名(必备属性，[任选属性])

返回结果：必备属性，[任选属性]

5.1.1　INITIALIZE 原语

原语格式：INITIALIZE　([本端端口]，[本地合格的地址列表])

返回结果：本地 SCTP 实例名

这个原语允许 SCTP 启动其内部的数据结构，并为建立操作环境分配所需的资源，一旦 SCTP 启动后，则高层协议在与其他 SCTP 端点直接通信时就不需要再调用该原语。

SCTP 将向高层协议返回一个本地 SCTP 实例名

必备属性：无

任选属性：以下属性类型可以通过原语进行传递。

—本地端口：SCTP 端口号，如果高层协议(ULP)希望规定；

—本地合格的地址列表，本地 SCTP 端点应当绑定的地址列表。如果未包含地址列表，缺省的，所有分配给主机的所有 IP 地址应当是作为本地端点(注)。

注：如果实施支持这个任选属性，则实施应当保证本端点发送的任何 SCTP 分组中应当包含一个在本地合格的地址列表中规定的 IP 地址。

5.1.2　ASSOCIATE 原语

原语格式：ASSOCIATE(本地 SCTP 实例名，目的地传送地址，出局的流数量)

返回结果：偶联 ID　[，目的地传送地址列表] [，出局的流数量]

该原语用来由高层启动一个到特定端点的偶联。对端点按照在该端点定义的传送地址进行规定，如果本地 SCTP 实例未启动，则认为该原语是一个差错。

用来进行本地处理 SCTP 偶联的偶联 ID，作为返回结果用来返回偶联是否成功建立。如果偶联建立不成功，则返回一个差错。如果偶联成功建立，则返回结果中还应包含到对端的完整的传送地址列表以及本端点的出局的流数量，同时还应从返回的目的地地址中选择一个传送地址作为本地端点向对端发送 SCTP 分组的首选通路。返回的“目的地传送地址列表”可以由高层协议用来改变首选通路，或者是向一个特定传送地址强制发送一个分组。

注：如果 ASSOCIATE 原语用作模块化的功能调用，则偶联成功建立后，ASSOCIATE 原语还可以返回除偶联 ID 之外的其他偶联参数；如果 ASSOCIATE 原语是作为一个非模块化的功能调用，则应当只返回偶联 ID，其他偶联参数应当用 COMMUNICATION UP 进行通知。

必备属性：

—本地 SCTP 实例名：从 INITIALIZE 操作中获得；

—目的地传送地址：规定了建立的偶联的对端点的一个传送地址；

—出局的流数量，高层协议希望向对端点开放的出局的流的数量。

任选属性：无

5.1.3　SHUTDOWN 原语

原语格式：SHUTDOWN(偶联 ID)

返回结果：结果编码

该原语用来正常地关闭一个偶联，任何已在本地发送队列中的用户数据都将被递交到对端。该偶联将在收到所有发送的 SCTP 分组的证实后停止。返回结果用来指示是否成功地关闭了该偶联，如果成功则返回一个成功关闭偶联编码；如果试图关闭偶联失败，则返回一个差错编码。

必备属性：

—偶联 ID：本地处理的 SCTP 偶联。

任选属性：无

5.1.4 ABORT 原语

原语格式：ABORT(偶联 ID [，原因编码])

返回结果：结果编码

该原语用来非正常地关闭(中止)一个偶联，本地发送队列中的用户数据将被丢弃，并发送一个 ABORT 数据块到对端。返回结果用来指示是否成功地中止了该偶联，如果偶联中止成功，则返回一个成功编码；如果试图中止偶联失败，则返回一个差错编码。

必备属性：

一偶联 ID：本地处理的 SCTP 偶联。

任选属性：

—原因编码：传递给对端的中止原因。

5.1.5 SEND 原语

原语格式：SEND(偶联 ID，缓冲区地址，字节计数[，上下文] [，流 ID] [，存活时间][，目的地传送地址] [，无序标志] [，无绑定标志] [，净荷协议 ID])

返回结果：结果编码

SCTP 用户使用该原语通知 SCTP 发送数据，返回结果用来指示是否成功地发送了数据。这是通过 SCTP 发送用户数据的主要方法。

必备属性：

—偶联 ID：本地处理的 SCTP 偶联；

—缓冲区地址：需要发送的用户消息存储的位置；

—字节计数：用户数据的字节数。

任选属性：

—上下文：一个任选的 32bit 整数，如果这个用户消息传送失败，则在向高层协议通知发送失败时携带。

—流 ID：用来指示数据需要在哪个流上发送，如果未规定，则缺省认为流 ID 为 0。

—存活时间：规定用户数据的存活时间，用户数据不应在存活时间之后由 SCTP 发送。这个参数可以避免传送一个过期的用户消息，如果用户数据不能在存活时间内启动传送(即通过 SCTP 的 SEND 原语发送到目的地)，则 SCTP 应当通知高层用户。如果 SCTP 已经尝试在存活时间超时前传送数据块，则用户数据就可以认为是被发送了。

注：为了更好地支持数据块的存活时间选项，发送方可以暂停给一个出局的 DATA 数据块分配 TSN 号码。为了简化实施，一旦 TSN 号码由发送方分配，则认为这个 DATA 数据块已经被发送，并且不受这个 DATA 数据块的存活时间选项的限制了。

—目的地传送地址：规定了分组要传送到已建立的偶联的对端点的一个传送地址，只要可能，SCTP 将使用这个目的地传送地址来发送分组，而不使用当前的首选通路。

—无序的标志：如果这个标志存在，则指示用户允许把数据按照无序的方式递交给对端(即：携带这个这个消息的所有 DATA 数据块的 U 标志位设置为 1)。

—无绑定标志：指示 SCTP 不要把这个用户数据同其他出局的 DATA 数据块捆绑在一个 SCTP 分组中。当网络拥塞时，SCTP 可以不考虑这个标志位是否存在而直接进行捆绑。

—净荷协议 ID：一个 32bit 的无符号整数，它用来指示被传送到对端的净荷协议数据的类型，这个值由 SCTTP 作为透明的数据进行传递。

5.1.6 SET PRIMARY 原语

原语格式：SETPRIMARY(偶联 ID，目的地传送地址，[起源传送地址])

返回结果：结果编码

高层协议用该原语指示本地 SCTP 把给定的目的地传送地址作为发送分组的首选通路。该操作的返回结果用来指示这个操作是否成功执行。

如果规定的目的地传送地址没包含在先前 ASSOCIATE 原语或 COMMUNICATION UP 通知返回的“目的地传送地址列表”中，则返回一个差错。

必备属性：

—偶联 ID，本地处理的 SCTP 偶联；

—目的地传送地址：规定了分组要传送到已建立的偶联的对端点的一个传送地址，这个地址将作为今后发送分组的首选地址，这个地址将修改本地 SCTP 地址维护的当前首选地址信息。

任选属性：

—起源传送地址：一些实施可以允许任选的把放在所有的出局 IP 数据报中的地址设置为缺省起源地址。

5.1.7 RECEIVE 原语

原语格式：RECEIVE(偶联 ID，缓冲区地址，缓冲区容量 [，流 ID])

返回结果：字节计数 [，传送地址] [，流 ID] [，流顺序号码] [，部分标志位] [，递交号码][，净荷协议 ID]

该原语用来把在 SCTP 队列中的可用的用户消息读到由高层协议规定的缓冲区中。所读消息的字节数将作为结果返回，如果有可能根据特定的实施，也可以返回其他信息，如发送方的地址、收到的消息的流 ID，以及是否有消息可以进行恢复等。对于顺序的消息，他们的流顺序号码也可以被返回。

根据实施，如果在调用这个原语时，队列中没有消息可用，则可以返回一个这种情况的指示，或者是先禁止执行该调用进程，直到队列中有新消息可用为止。

必备属性：

—偶联 ID：本地处理的 SCTP 偶联；

—缓冲区地址：高层协议指示的接收的用户消息被存储的内存位置；

—缓冲区容量，将要收到的数据的最太长度，以字节为单位。

任选属性：

—流 ID：用来指示接收到的数据所在的流；

—流顺序号码：由对端 SCTP 发送方分配的流顺序号码；

—部分标志位，如果返回的这个标志位设置为 1，则这个 RECEIVE 原语中只包含了整个消息的一部分，如果这个标志位被设置，则流 ID 和流顺序号码必须也包含在这个 RECEIVE 原语中，如果这个标志伎设置为 0，则表示对这个流顺序号码而言已经没有更多需要递交的内容；

—净荷协议 ID：一个 32bit 的无符号整数，它用来指示收到的对端的净荷协议数据的类型，这个值由 SCTP 作为透明的数据进行传递。

5.1.8 STATUS 原语

原语格式：STATUS(偶联 ID)

返回结果：状态数据

该原语用来要求 SCTP 返回一个包含以下信息的数据块。

—偶联连接状态；

—目的地传送地址表；

—目的传送地址的可达性状态；

—当前的接收方窗口大小；

—当前的拥塞窗口大小；

—未确认的 DATA 数据块的数量；

—收到的 DATA 数据块的数量；

—首选通路；

—首选通路上最近收到的 SRTT；

—首选通路的 RTO；

—其他目的地地址的 SRTT 和 RTO 等。

必备属性：

—偶联 ID：本地处理的 SCTP 偶联。

任选属性：无

5.1.9 CHANGE HEARTBEAT 原语

原语格式：CHANGEHEARTBEAT(偶联 ID，目的地传送地址，新状态 [，周期])

返回结果：结果编码

高层协议用该原语指示本地端点允许或禁止向指定的目的地传送地址发送心跳消息。返回原因用来指示该操作的执行情况。如果可能，当目的传送地址未空闲时，心跳程序也不执行。

必备属性：

—偶联 ID：本地处理的 SCTP 偶联；

—目的地传送地址：规定了到偶联的对端点的一个传送地址；

—新状态：用来指示对该目的地传送地址的心跳状态(允许或禁止)。

任选属性：

—周期：如果该参数存在，且允许对端目的地传送地址进行心跳测试，则用来指示心跳测试的频率，该参数的缺省值为设置值加上到目的地地址的 RTO，这个参数对所有目的地起作用。

5.1.10 REQUEST HEARTBEAT 原语

原语格式：REQUESTHEARTBEAT(偶联 ID，目的地传送地址)

返回结果：结果编码

高层协议用该原语指示本地端点对给定偶联的特定目的地传送地址执行心跳程序，返回结果用来指示传送给目的地地址的 HEARTBEAT 数据块是否成功。

必备属性：

—偶联 ID：本地处理的 SCTP 偶联；

—目的地传送地址：HEABTBEAT 消息需要发送去的偶联传送地址。

5.1.11 GET SRTT REPORT 原语

原语格式：GETSRTTREPORT(偶联 ID，目的地传送地址)

返回结果：SRTT 的取值

高层协议用该原语指示本地 SCTP 报告对给定偶联上规定的目的地传送地址的当前 SRTT 测量值，返回结果应当是一个包含最近 SRTT 的毫秒值(整数)。

必备属性，

—偶联 ID：本地处理的 SCTP 偶联；

—目的地传送地址：需要报告的 SRTT 测量的偶联传送地址。

5.1.12 SET FAILURE THRESHOLD 原语

原语格式：SETFAILURETHRESHOLD(偶联 ID，目的地传送地址，故障门限)

返回结果：结果编码

该原语允许本地 SCTP 定制到给定目的地地址的可达性故障检出的门限“Path.Max.Retrans”，返回结果用来指示该操作是否成功。

必备属性：

—偶联 ID：本地处理的 SCTP 偶联；

—目的地传送地址：设置故障检出门限的偶联传送地址。

—故障门限：对该目的地地址的 Path.Max.Retrans 参数的新取值。

5.1.13 SET PROTOCOL PARAMETERS 原语

原语格式：SETPROTOCOLPARAMETERS(偶联 ID[，目的地传送地址，]协议参数列表)

返回结果：结果编码

该原语允许本地 SCTP 定制协议参数，返回结果用来指示该操作是否成功。

必备属性：

—偶联 ID：本地处理的 SCTP 偶联；

—协议参数列表：SCTP 用户希望定制的协议参数的名称和取值(如：Association.Max.Retrans 等)。

任选属性：

—目的地传送地址：针对每个目的地传送地址可以对相关的协议参数进行设置。

5.1.14 RECEIVE UNSENT MESSAGE 原语

原语格式：RECEIVE_UNSENT(数据恢复 ID，缓冲区地址，缓冲区容量[，流 ID][，流顺序号码][，部分标志位] [，净荷协议 ID])

必备属性：

—数据恢复 ID：在故障(failure)通知中传递绘高层的标识；

—缓冲区地址：高层协议指示的接收的消息所存储的内存位置；

—缓冲区容量：将要收到的数据的最太长度，以字节为单位。

任选属性：

—流 ID：这个返回值被设置用来指示数据需要发送到哪个流；

—流顺序号码：这个值返回用来指示与该消息相关的流顺序号码；

—部分标志位：如果返回的这个标志位设置为 1，则这个 Receive 原语中只包含了整个消息的一部分，如果这个标志位被设置，则流 ID 和流顺序号码必须也包含在这个 Receive 原语中，如果这个标志位设置为 0，则表示对这个流顺序号码而言已经没有更多需要递交的内容；

—净荷协议 ID：一个 32bit 的无符号整数，它用来指示收到的对端的净荷协议数据的类型，这个值由 SCTP 作为透明的数据进行传递。

5.1.15 RECEIVE UNACKNOWLEDGED MESSAGE 原语

原语格式：RECEIVE_UNACKED(数据恢复 ID，缓冲区地址，缓冲区容量[，流 ID) [，流顺序号码][，部分标志位] [，净荷协议 ID])

必备属性：

—数据恢复 ID：在故障(failure)通知中传递给高层的标识；

—缓冲区地址：高层协议指示的接收的消息所存储的内存位置；

—缓冲区容量，将要收到的数据的最大长度，以字节为单位。

任选属性：

—流 ID：这个返回值被设置用来指示数据需要发送到哪个流；

—流顺序号码：这个值返回用来指示与该消息相关的流顺序号码；

—部分标志位：如果返回的这个标志位设置为 1，则这个 Receive 原语中只包含了整个消息的一部分，如果这个标志位被设置，则流 ID 和流顺序号码必须也包含在这个 Receive 原语中，如果这个标志位设置为 0，则表示对这个流顺序号码而言已经没有更多需要递交的内容；

—净荷协议 ID，一个 32bit 的无符号整数，它用来指示收到的对端的净荷协议数据的类型，这个值由 SCTP 作为透明的数据进行传递。

5.1.16 DESTROY 原语

原语格式：DESTROY(本地 SCTP 实例名)

必备属性：

—本地 SCTP 实例名：这个值是在 initialize 原语中传递给应用的，它用来指示哪个 SCTP 实例要进行破坏。

5.2 SCTP 向高层协议发送的通知原语

假定 SCTP 提供了一种由 ULP 异步处理信号的方法，当 SCTP 发一个信号给 ULP 处理时，也需要把特定的信息传递给 ULP。

5.2.1 DATA ARRlVE 通知

当一个用户消息被成功地接收，并且准备向高层递交时，SCTP 将用该原语通知高层协议。

原语格式：DATA ARRIVE ([，偶联 ID] [，流 ID])

返回结果；无

在该通知原语中可以包含以下任选属性。

—偶联 ID：本地处理的 SCTP 偶联；

—流 ID：用来指示数据是从哪个流上收到的。

5.2.2 SEND PAILURE 通知原语

当一个消息不能被递交时，SCTP 将发送该原语通知高层协议。

原语格式：SEND FAILURE([，偶联 ID] [，数据恢复 ID] [，原因编码][，上下文])

返回结果：无

在该通知原语中可以包含以下任选属性。

—偶联 ID：本地处理的 SCTP 偶联；

—数据恢复 ID：用来恢复末发送和未证实数据的标识；

—原因编码：用来指示故障的原因，例如：长度过长、消息存活时间过期等；

—上下文：与该消息相关的任选信息(见 5.1.1 节)。

5.2.3 NETWORK STATUS CHANGE 通知原语

当目的地传送地址被标记为未激活时 (例如，当 SCTP 检测出故障)或标记为激活时(当 SCTP 检出故障恢复)，SCTP 将用该原语通知高层协议。

原语格式：NETWORK STATUS CHANGE(偶联 ID，目的地传送地址，新状态)

返回结果：无

在该通知原语中必须包含以下必备属性。

—偶联 ID：本地处理的 SCTP 偶联；

—目的地传送地址：它用来指示由于状态变化而受影响的对端点的目的地传送地址；

—新状态：用来指示新的状态。

5.2.4 COMMUNICATION UP 通知原语

SCTP 用该原语通知高层协议本地 SCTP 已经准备好发送或接收用户数据，或者是一个丢失通信的端点又已经恢复。

注：如果 ASSOCIATE 是作为一个模块化功能调用，则偶联参数都作为 ASSOCIATE 原语的返回结果，在这种情况下，COMMUNICATION UP 通知原语作为偶联的发起方是任选购。

原语格式：COMMUNICATION UP(偶联 ID，状态，目的地传送地址列表，出局的流数量，入局流数量)

返回结果：无

在该通知原语中必须包含以下必备属性。

—偶联 ID：本地处理的 SCTP 偶联；

—状态：用来指示发生了哪种类型的事件；

—目的地传送地址列表：对端点的一个完整的传送地址集合；

—出局的流数量：偶联高层允许使用的最大的流数量；

—入局流数量：对端点对该偶联所请求的流数量(这个值可以与出局的流数量的取值不同)。

5.2.5 COMMUNICATION LOST 通知原语

当 SCTP 完全丢失了到一个端点的通信时(用心跳消息)，或者是检出端点已经执行了中止操作，则 SCTP 将使用该原语通知高层协议。

原语格式：COMMUNICAT1ON LOST(偶联 ID，状态 [，数据恢复 ID][，最后证实的 TSN][，最后发送的 TSN])

返回结果：无

在该通知原语中必须包括以下必备属性。

—偶联 ID：本地处理的 SCTP 偶联；

—状态：用来指示发生了哪种类型的事件，状态可以指示故障或者是响应 SHUTDOWN 或 ABORT 请求的正常的中止事件。

在该通知原语中可以包括以下任选属性。

—数据恢复 ID：用来恢复未发送或未证实数据的表示；

—最后证实的 TSN：由对端点最后证实的 TSN；

—最后发送的 TSN：发送到对端的最后一个 TSN。

5.2.6 COMMUNICATION ERROR 通知原语

当 SCTP 从对端收到了一个 ERROR 数据块，并且确定需要向高层协议通知时，才使用该通知原语。

原语格式：COMMUNICATION ERROR(偶联 ID，错误信息)

返回结果：无

在该通知原语中必须包括以下必备属性。

—偶联 ID：本地处理的 SCTP 偶联；

—错误信息：用来指示错误类型，并且可以任选地包含一些从 ERROR 数据块中收到的附加信息。

5.2.7 RESTART 通知原语

当 SCTP 检出其对端已经重新启动时，则可以用该原语通知高层协议。

原语格式：RESTART(偶联 ID)

返回结果：无

在该通知原语中必须包括以下必备属性。

—偶联 ID：本地处理的 SCTP 偶联。

5.2.8 SHUTDOWN COMPLETE 通知原语

当 SCTP 已经完成了关闭程序后，则用该原语通知高层协议。

原语格式：SHUTDOWN COMPLETE(偶联 ID)

返回结果：无

在该通知原语中必须包括以下必备属性。

—偶联 ID：本地处理的 SCTP 偶联。

6 SCTP 分组的格式以及参数定义

6.1 SCTP 分组格式

SCTP 分组由公共的分组头和若干数据块组成，每个数据块中既可以包含控制信息，也可以包含用户数据。除了 INIT，INIT ACK 和 SHUTTDOWN COMPLETE 数据块外，其他类型的多个数据块可以捆绑在一个 SCTP 分组中，当然必须要满足偶联对 MTU 的要求。当然这些数据块也可以不与其他数据块捆绑在一个分组中。如果一个用户消息不能放在一个 SCTP 分组中，则这个消息可以被分成若干个数据块。

SCTP 分组的格式如图 3 所示。

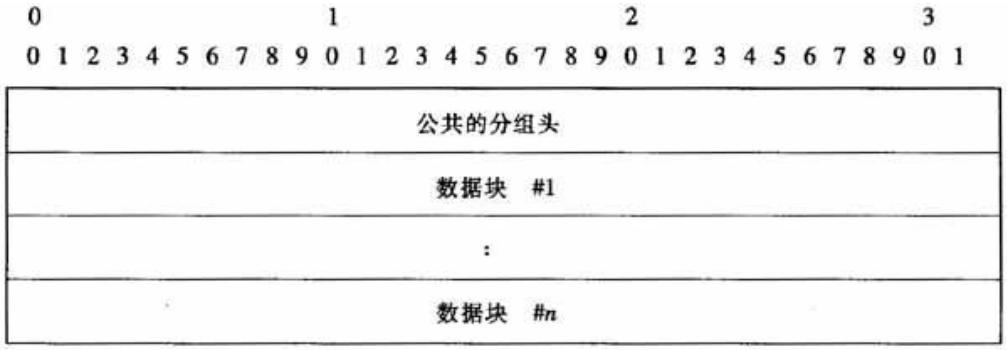

图 3 SCTP 分组的格式

6.1.1 SCTP 公共分组头字段的格式

SCTP 公共分组头字段的格式如图 4 所示。

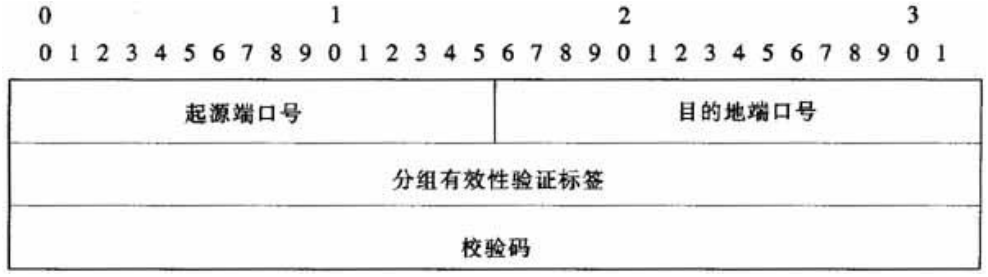

图 4 SCTP 公共分组头的格式

SCTP 公共分组头的各字段含义如下所述。

起源端口号(16bit 的无符号整数)，该端口号用来识别 SCTP 发送方的端口号码，接收方使用起源端口号和起源 IP 地址，以及目的地端口号和可能的目的地 IP 地址来识别属于某个偶联的分组。

目的端口号(16bit 的无符号整数)：该 SCTP 端口号用来确定分组的去向。接收方主机将利用该端口号把 SCTP 分组解复用到正确的接收端点或应用。

验证标签(32bit 的无符号整数)：接收到分组的接收方使用验证标签来判别发送方的这个 SCTP 分组的有效性。在发送方该验证标签必须设置为在偶联启动阶段中从对端点收到的启动标签中的值，以下情况除外。

—包含 INIT 数据块的分组中验证标签必须为 0，携带 INIT 块的 SCTP 的分组中不能包

含其他数据块；

—在包含 SHUTDOWN-COMPLETE 数据块且设置了 T 比特的分组中，验证标签必须要从包含 SHUTDOWN-ACK 数据块的分组中复制；

—在包含 ABORT 数据块的分组中，验证标签必须要从触发这个 ABORT 发送的分组中复制。

校验码(32bit 的无符号整数)，该字段用来传送 SCV 分组的校验码。

6.1.2 数据块字段的格式

SCTP 分组中数据块字段的格式如图 5 所示，每个数据块中都包括数据块类型字段、数据块特定的标志位字段、数据块长度字段和取值字段。

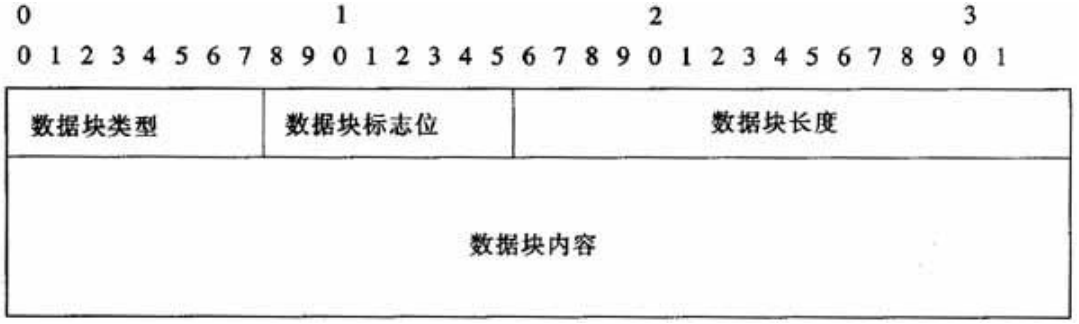

图 5 SCTP 分组中数据块字段的格式

SCTP 分组的数据块字段中各字段含义如下所述。

数据块类型(8bit 无符号整数)：该字段用来表明在数据块中内容字段的信息类型，该参数的取值范围为 0～254，255 留作今后的扩展。

数据块类型字段的编码分配如下：

类型编码 含义

0 —净荷数据(DATA)

1 —启动(INIT)

2 —启动证实 (INIT ACK)

3 —选择证实 (SACK)

4 —Heartbeat 请求(HEARTBEAT)

5 —Heartbeat 证实(HEARTBEAT ACK)

6 —中止 (ABORT)

7 —关闭(SHUTDOWN)

8 —关闭证实 (SHUTDOWN ACK)

9 —操作差错 (ERROR)

10 —状态 Cookie(COOKIE ECHO)

11 —Cookie 证实 (COOKIE ACK)

12 —为明确拥塞通知响应(ECNE)预留

13 —为降低拥塞窗口(CWR)预留

14 —关闭完成 (SHUTDOWN COMPLETE)

15～62 —IETF 预留

63 —IETF 定义的数据块扩展

64～126 —IETF 预留

127 —IETF 定义的数据块扩展

128～190 —IETF 预留

191 —IETF 定义的数据块扩展

192～254 —IETF 预留

255　　—IETF 定义的数据块扩展

采用这种方式编码，可以使用数据块类型字段中高两位的比特用来指示当端点收到不识别该数据块类型时所要采取的特定动作。

00：停止处理该分组并舍弃该分组，不处理分组中任何随后的数据块。

01：停止处理该分组并舍弃该分组，不处理分组中任何随后的数据块，并在 ERROR 的“不识别的数据块类型”字段中报告不识别的数据块。

10：跳过该数据块并继续处理。

11：跳过该数据块并继续处理，在 ERROR 数据块中报告差错原因为不识别的参数类型。

注：ECNE 和 CWR 数据块类型预留给“明确拥塞通知”备用。

数据块标志位(8bit)：这些比特的使用根据数据块类型的取值，除非特殊规定，否则这个字段设置为 0，并在接收方忽略。

数据块长度(16bit 的无符号整数)：该值用来表示包含数据块类型字段、数据块标志位字段、数据块长度字段和取值字段在内的字节数。因此如果数据块取值部分的长度为 0，则该长度字段应设置为 4。数据块长度字段不计算该数据块中最后一个参数中包含的填充字节。

数据块内容(可变长度)：数据块内容字段是在该数据块中传送的真正的信息，该字段的使用和格式取决于数据块类型。

数据块的总长度(包括类型、长度和取值字段)必须是 4 字节的整数情，如果该长度不是 4 字节的整数倍，则发送方应当向数据块中填充全 0 的字节，这些填充的字节不计人数据块长度字段。发送方填充的字节数应不超过 3 个字节，在接收方忽略所有的填充字节。除最后一个参数外，其他参数中的填充字段作为数据块长度进行计算。

6.1.3　任选 / 可变长参数的格式

SCTP 控制数据块的内容取值包含了特定数据块类型所要求的字段，即随后是一个或多个参数。这些包含在一个数据块中的任选 / 可变长参数都是按照参数类型、参数长度和参数取值的方式定义的，其格式如图 6 所示。

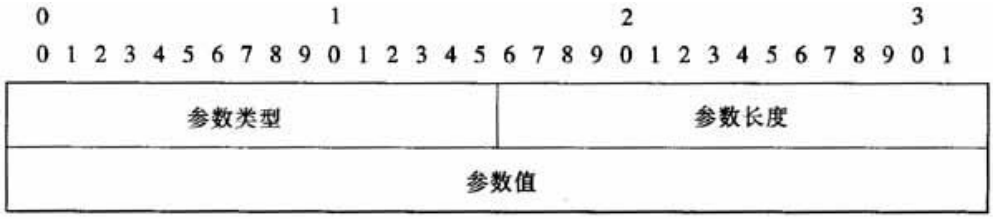

图 6　任选 / 可变长参数格式

数据块的参数类型(16bit 无符号整数)：参数类型字段用来识别参数的类型，取值范围可以从 0 到 65534，65535 预留给 IETF 进行扩展。

数据块的参数长度(16bit 无符号整数)：参数长度字段包含参数类型、参数长度和参数取值字段在内所有字段的字节数，因此一个参数的取值字段为 0，则该长度字段应设置为 4。参数长度字段不计算填充字节。

数据块的参数值：可变长度，参数取值字段包含在该参数中传送的实际信息。

参数的总长度(包括类型、长度和取值字段)必须是 4 字节的整数倍，如果该长度不是 4 字节的整数倍，则发送方应当向参数中填充全 0 的字节，这些填充的字节不计入参数长度字段。发送方填充的字节数应不超过 3 个字节，接收方忽略所有的填充字节。

采用这种方式编码，可以使用参数类型字段中高两位的比特指示当端点收到不识别参数类型时所要采取的特定动作。

00：表示停止处理该分组并舍弃该分组，不处理分组中任何随后的数据块。

01：表示停止处理该分组并舍弃该分组，不处理分组中任何随后的数据块，并在 ERROR 或 INITACK 的“不识别的参数类型”字段中报告不识别的参数类型。

10：跳过该参数并继续处理。

11：跳过该参数并继续处理，在 ERROR 或 INIT ACK 的“不识别的参数类型”字段中报告不识别的参数类型。

6.2 SCTP 数据块的格式

本节定义了 SCTP 协议使用的所有数据块的格式。

6.2.1 净荷数据(DATA)数据块的格式

DATA 数据块的格式如图 7 所示。

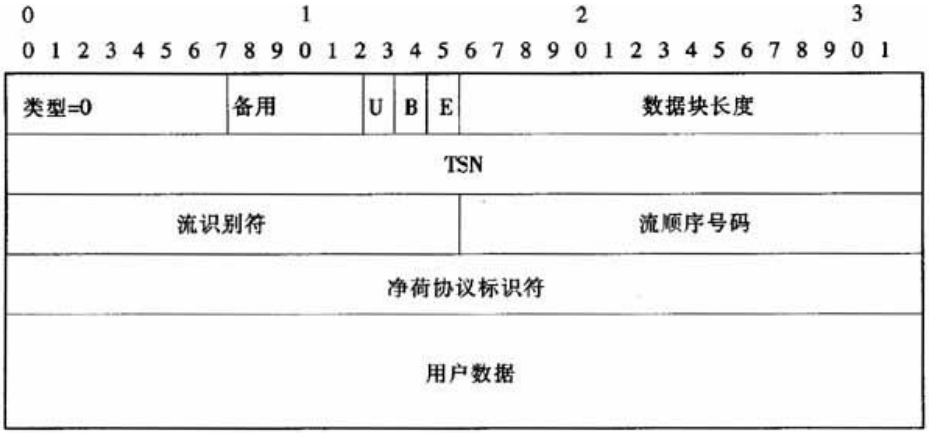

图 7 DATA 数据块格式

DATA 数据块中各字段含义如下所述：

备用比特(5bit)：应当设置为全 0，在接收方忽略。

U 比特(1bit)：称为非顺序比特。如果该比特设置为 1，则指示这是一个非顺序的 DATA 数据块，不需要给该数据块分配流顺序号码，所有接收方必须忽略流顺序号码。在重新组装完成后(如果需要)，非顺序的数据块不需要尝试任何重新排序的过程，可以由接收方直接递交到高层；如果一个非顺序的用户消息被分段，则消息的每个分段中的 U 比特必须被设置为 1。

B 比特(1bit)：称为分段开始比特。如果该比特被设置，则指示这是用户消息的第一个分段。

E 比特(1bit)：称为分段结束比特。如果该比特被设置，则指示这是用户消息的最后一个分段。一个未分段的用户消息应当把所有的 B 和 E 比特设置为 1。如果 B 和 E 比特都设置为 0，则表明这是一个分段的用户消息的一个中间分段。当用户消息被分段到多个数据块中，接收方需要使用 TSN 对消息进行重组，这意味着给分段的用户消息的每个分段都必须要使用连续的 TSN。BE 比特的取值含义如表 1 所述。

表 1 BE 比特的取值含义

B E	表 示 的 含 义
1 0	用户消息的第一个分段
0 0	用户消息的中间分段
0 1	用户消息的最后一个分段
1 1	未分段的消息

长度(16bit 的无符号整数)：该字段用来指示 DATA 数据块从类型字段开始到用户数据字段结束之间的字节数，但不包含任何填充字节，如果 DATA 数据块的用户数据字段为 0，则长度字段设为 16。

TSN(32bit 无符号整数)：该值表示该数据块的 TSN，TSN 的有效值从 0～2 -1。TSN 的值达到 4294967295 后将回转到 0。

流标识符(16bit 无符号整数)：该字段用来识别用户数据属于的流。

流顺序号码(16bit 无符号整数)，该值用来表示所在流中的用户数据的顺序号码。该字段的有效值为 0～65535。当一个用户消息被 SCTP 分段后，则必须在消息的每个分段中都带有相同的流顺序号码。

净荷协议标识符(32bit 无符号整数)：该值表示一个应用(或上层协议)特定的协议标识符。这个值由高层协议传递到 SCTP 并发送到对等层。这个标识符不由 SCTP 使用，但却可以由特定的网络实体或对等的应用来识别在 DATA 数据块中携带的信息类型。甚至在每个分段的 DATA 数据块中也应包含该字段(以确保对网络中间的代理可用)。0 表示高层未对该协议净荷规定应用标识符。其中“M2UA”协议净荷使用编码 2;“M3UA”协议净荷使用编码 3;“SUA”协议净荷使用编码 4;“M2PA”协议净荷使用的编码待定。

用户数据(可变长度),它用来携带用户数据净荷。该字段必须被填充为 4 字节的整数情，发送方填充的字节数应不超过 3 个字节，接收方忽略所有的填充字节。

6.2.2　启动(INIT)数据块的格式

该数据块用来启动两个 SCTP 端点间的一个偶联，INIT 数据块的格式如图 8 所示。

图 8　INIT 数据块格式

INIT 数据块应包含以下参数，除非特别支持，每个参数只能在 INIT 数据块中出现一次。必备参数：

—启动标签；

—通告的接收方窗口信用；

—输入流数量；

—输出流数量；

—初始 TSN。

可变长度参数：	类型值	状态
— IPv4 地址(注 1)	5	任选
— IPv6 地址(注 1)	6	任选
— Cookie Preservative	9	任选
— ECN 能力预留(注 2)	32768 (0x8000)	任选

— 主机名地址(注 3)　　11　　任选

— 支持的地址类型(注 4)　　12　　任选

注 1：INIT 数据块中可以包含多个 IPv4 或 IPv6 的地址或地址组合。

注 2：ECN 能力字段用于今后的明确拥塞通知。

注 3：INIT 数据块中不能包含多于一个的主机名，此外，发送方在 INIT 中不能有主机名地址类型和其他类型地址组合的情况，如果 INIT 数据块中包含有主机名，则接收到 INIT 的一方将忽略其他地址。

注 4：该参数如果存在，则规定了发送方端点能支持的所有地址类型，如果没有这个参数，则表明发送方端点可以支持任意的地址类型。

INIT 数据块标志字段：该字段备用，所有的比特应设为全 0。INIT 中的参数可以接任何顺序进行处理。

启动标签(32bit 无符号整数)：INIT 的接收方(响应端)记录启动标签参数的值。这个值必须被放置到 INIT 的接收方发送的与该偶联相关的每个 SCTP 分组中的验证标签字段中。启动标签允许除 0 以外的的任何值。如果在收到的 INIT 数据块中的启动标签为 0，则接收方必须作为错误处理，并且发送 ABORT 数据块中止该偶联。

通告的接收方窗口信用值(a_rwnd，32bit 的无符号整数)：这个值表示指定的缓冲区的容量，用字节数表示，为 INIT 发送方为偶联预留的窗口大小。在偶联存活期间，这个缓冲区的容量不应减少(即不应把该偶联的专用缓冲区取走)，但端点可以在发送的 SACK 数据块中修改 a_rwnd 的值。

输出流数量(OS，16bit 的无符号整数)，用来定义发送 INIT 数据块的一方希望在该偶联中创建的输出流的数量。该值不允许为 0，接收方收到该参数为 0 的 INIT 数据块后应中止该偶联。

输入流数量(NIS，16bit 的无符号整数)：定义了发送这个 INIT 块的一方允许对端在该偶联中所创建的流的数量。该值不允许为 0，接收方收到该参数为 0 的 INIT 数据块后应中止该偶联。

初始的 TSN(32bit 的无符号整数)：定义发送方将使用的初始的 TSN，该值可以设置为启动标签字段的值。

6.2.2.1　INIT 中的任选 / 可变长参数

本节定义的 INIT 数据块的可变长参数应当在定长参数之后。

— IPv4 地址参数，如图 9 所示。

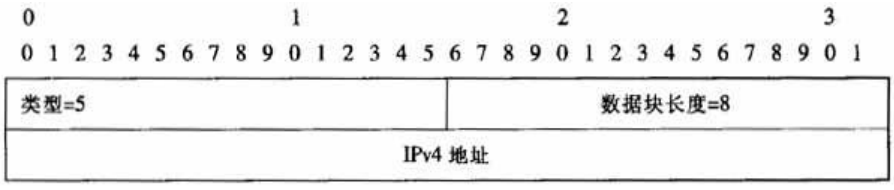

图 9　IPv9 地址参数格式

IPv4 地址（32bit 无符号整数）：包含发送方端点的 IPv4 地址，采用二进制编码。

—IPv6 地址参数，如图 10 所示。

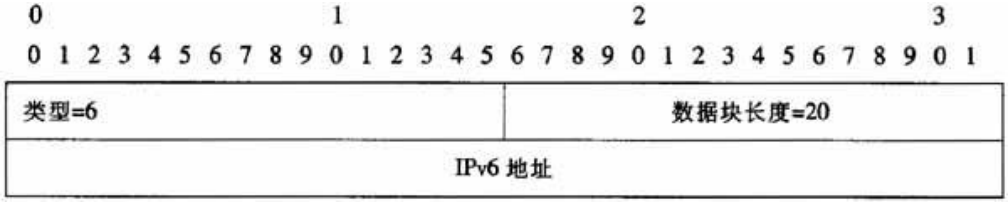

图 10 IPv6 地址参数格式

IPv6 地址(128bit 的无符号整数)：包含发送方端点的 IPv6 地址，采用二进制编码。发送方不必把 IPv4 地址映射到 IPv6 地址中，可以直接在 IPv4 地址参数中使用 IPv4 地址。

与 SCTP 公共分组头中的起源端口号一起，IPv4 或 IPv6 地址参数中的地址可以用来向 INIT 的接收方指示，由 INIT 发送方所支持的传送地址。在该偶联存活期间，这个 IP 地址可以出现在 IP 包中起源地址字段中，由 INIT 的发送者来发送，并且可以由 INIT 的接收者作为 IP 包的目的地地址。当 INIT 的发送方是一个多归属的情况时，多于一个 IP 地址参数可以包含在一个 INIT 数据块中。此外一个多归属的端点可以接入到不同类型的网络，这样多于一个的地址类型能够在一个 INIT 数据块中出现，即 IPv4 和 IPv6 的地址允许出现在同一个 INIT 数据块中。

如果 INIT 中包含了 IP 地址参数,则包含在 INIT 数据块的 IP 数据报中的起源地址和其他附加地址均可以被接收 INIT 的端点作为目的地。如果 INIT 中未包含任何 IP 地址参数，在收到 INIT 的端点必须使用收到的 IP 数据报中的起源地址作为该偶联的目的地地址。

—防止 Cookie 过期参数，如图 11 所示。

INIT 的发送方应使用这个参数来建议 INIT 的接收方提供较长存活跨度的状态 CooKIE。

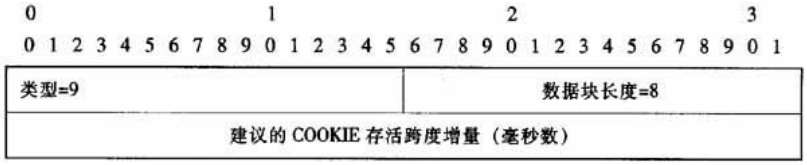

图 11　防止 Cookie 过期参数的格式

建议的 COOKIE 存活跨度增量(32bit 的无符号整数)，该参数用来向接收方指示发送方希望接收方为其缺省的 COOKIE 的存活跨度增加的毫秒数。

由于失效的 cookie 操作差错原因，前一次尝试与对等端建立偶联失败后，又重新尝试偶联建立时，这个任选参数应能由发送方添加到 INIT 数据块中。接收方出于安全的考虑可以选择忽略建议的 COOKIE 存活跨度增量。

—主机名地址(11)，如图 12 所示。

INIT 发送方使用这个参数把其主机名(在其 IP 地址的位置中)传递到对等层。这个对等层负责解析这个主机名，用这个参数可以使偶联工作通过 NAT box 进行工作。

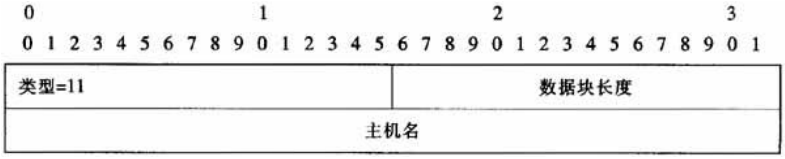

图 12　主机名地址参数格式

主机名：可变长度，该字段包含了按照 RFC1123 规定的“主机名句法”定义的主机名，主机名地址的解析不在本标准中规定,该参数中至少有一个非空的中止符包含在主机名字符串中，并且应包含长度。

—支持的地址类型：INIT 的发送方使用该参数列出其所支持的全部地址类型，如图 13 所示。

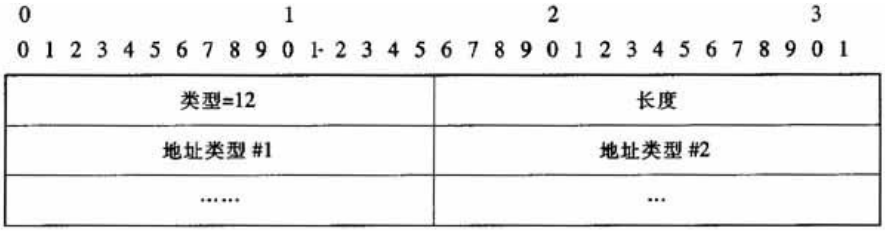

图 13　支持的地址类型参数格式

地址类型(16bit 无符号整数)：该参数使用对应的地址类型的类型值(例如：IPv4=5，IPv6=6，主机名=11)。

6.2.3 启动证实(INIT ACK)数据块的格式

INIT ACK 数据块用来确认 SCTP 偶联的启动。

INIT ACK 的参数部分与 INIT 数据块的参数部分相同，它额外还使用两个的可变长度的参数即：状态 COOKIE(STATE COOKIE)和未识别的参数。

INIT ACK 数据块的格式如图 14 所示。

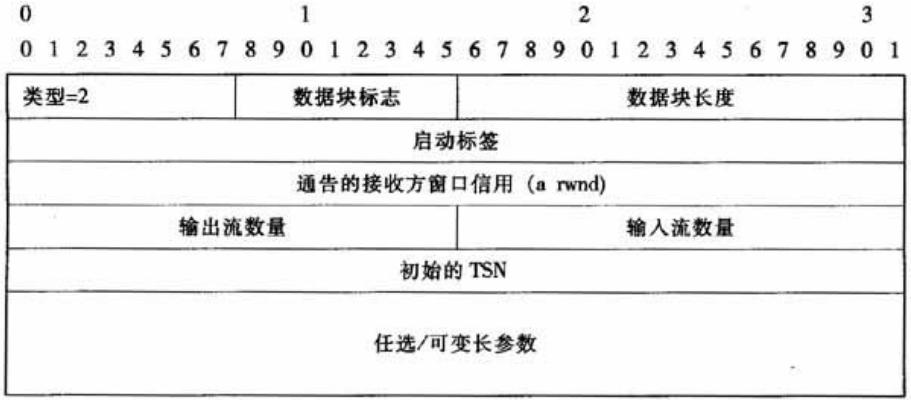

图 14 INIT ACK 数据块格式

INIT ACK 数据块应包含以下参数：

必备参数：

—启动标签；

—通告的接收方窗口信用；

—输入流数量；

—输出流数量；

—初始 TSN。

可变长参数：	类型值	状态
STATECOOKIE	7	必备
IPv4 地址	5	任选
IPv6 地址	6	任选
未识别的参数	9	任选
ECN 能力预留	32768(0x8000)	任选
主机名地址	11	任选

启动标签(32bit 的无符号整数)：INIT ACK 的接收方记录启动标签参数的值，并把该值放到 INITACK 接收方需要在相应的偶联上发送的每个 SCTP 分组中的验证标签中。启动标签不允许为 0。如果收到的 INIT ACK 数据块中的启动标签为 0，则接收方当作错误来处理并通过发送 ABORT 来关闭偶联。

通告的接收方窗口信用值(a_rwnd，32bit 的无符号整数)：这个值表示指定的缓冲区的容量，用字节数表示，是 INIT ACK 发送方为偶联预留的窗口，在偶联存活期间，这个缓冲区的容量不应减少(即不应把该偶联的专用缓冲区取走)。

输出流数量(OS，16bit 的无符号整数)：定义发送 INIT ACK 数据块的一方希望在该偶联中创建的输出流的数量。该值不允许为 0，接收方收到该参数为 0 的 INIT ACK 数据块后应中止该偶联并舍弃 TCB。

输入流数量(NIS，16bit 的无符号整数)：定义发送 INIT ACK 数据块的一方允许对端点在该偶联中所创建的流的最大数量。该值不允许为 0，接收方收到该参数为 0 的 INIT ACK

数据块后应中止该偶联并舍弃该 TCB。

初始的 TSN(32bit 的无符号整数)：定义发送方将使用的初始的 TSN，该值可以设置为启动标签字段的值。

与 SCTP 公共分组头中的起源端口号一起，每个在 INIT ACK 中 IP 地址向 INIT ACK 的接收方指示偶联启动的存活期间 INIT ACK 发送方支持的有效传送地址。如果 INIT ACK 包含了一个 IP 地址参数，则包含在 INIT ACK 数据块的 IP 数据报的起源地址和在 INIT ACK 中提供附加的地址都可以被 INIT ACK 的接收方作为目的地。如果 INIT ACK 未包含任何 IP 地址参数，则 INIT ACK 的接收方必须使用收到包含 INIT ACK 数据块的 IP 数据报的起源地址作为该偶联的惟一的目的地地址。

State Cookie 和未识别的参数也使用类型-长度-内容的格式来编码，INIT ACK 中其他参数的定义可参见 INIT 中相关参数的定义。

State Cookie 参数：该参数类型为 7，为可变长度参数，该参数长度取决于 COOKIE 的长度，该参数值的取值必须包含由 INIT ACK 发送方创建该偶联所需的所有状态和参数信息，连同消息授权码。

不识别的参数，该参数类型为 8，可变长度参数。该参数内容是 INIT 数据块中包含的一个不识别的参数，该参数用来返回给 INIT 数据块的产生者一个指示，这个参数值字段包含了从 INIT 数据块中复制过来的不识别参数的完整的参数类型、长度和参数值。

6.2.4　选择证实(SACK)数据块的格式

这个数据块通过使用 DATA 数据块中的 TSN 用来向对等的端点确认接收到的 DATA 数据块，并通知对等的端点在收到的 DATA 数据块中的间隔。所谓间隔就是指收到的 DATA 数据块的 TSN 不连续的情况。

SACK 必须包含累积的 TSN 证实和通告的接收方窗口信用(a_rwnd)参数。

累积的 TSN 证实参数的值是指收到的 TSN 顺序断开前的最后一个 TSN 号码，下一个 TSN 则是在发送 SACK 的端点尚未收到的 TSN 值。所以这个参数确认已经收到了小于或等于该值的所有 TSN。

SACK 中可以包含 0 个或多个间隔证实块，每个间隔证实块确认了在一个不连续 TSN 后所收到的 TSN 序列，所有通过间隔证实块确认的 TSN 值都应比累积 TSN 证实的值大。

图 15　SACK 数据块格式

SACK 数据块的格式如图 15 所示。

数据块标志位，设为全 0 并由接收方忽略。

SACK 数据块中应包含如下参数：

累积 TSN 证实(32bit 无符号整数)：该参数包含了在收到 TSN 序列的间隔前的最后一个 TSN 值。

通告的接收方窗口信用(a_rwnd，32bit 的无符号整数)：该字段指示修改了 SACK 的发送方的接收缓冲容量的字节数。

间隔证实块的数目(16bit 的无符号整数)：用来指示 SACK 数据块中包含的间隔证实块的数目。

重复的 TSN 的数目(16bit 的无符号整数)，该字段包含了该端点收到的重复的 TSN 的数目。每个重复的 TSN 都列在间隔证实块列表后。

间隔证实块：这个字段中包含了间隔证实块，根据间隔证实块数量字段给出的值，间隔证实块重复若干次。所有 TSN 大于或等于累积 TSN 证实+间隔证实块开始的 DATA 数据块，或者是小于或等于每个间隔证实块的累积TSN证实+间隔证实块结束的DATA数据块都被看作是被正确地接收了。

间隔证实块开始(16bit 无符号整数)：该字段用来指示这个间隔整数块的起始 TSN 偏移，为了计算实际的 TSN 号码必须要用累积 TSN 证实加上偏移号码。计算出的 TSN 标识用来识别第一个在这个间隔证实块中被收到的 TSN。

间隔证实块结束(16bit 无符号整数)：用来指示这个间隔证实块的结束 TSN 偏移，为了计算实际的 TSN 需要把累积 TSN 证实加上这个偏移号码。这个计算出的 TSN 用来识别在这个间隔证实块中最后收到的 DATA 数据块。

重复的 TSN：(32bit 无符号整数)：用来指示一个在上一个 SACK 发送后收到的 TSN 重复的个数。每次一个接收者收到一个重复的 TSN(在发送 SACK 前)，则把这个 TSN 加到重复的 TSN 列表中。每发送一次 SACK 后则把统计重复 TSN 的计数器重新清 0。

6.2.5 HeartBeat 请求(HEARTBEAT)数据块的格式

SCTP 端点通过向对端点发送这个数据块用来检测定义在该偶联上到特定目的地传送地址的可达性。

参数字段包含 HEARTBEAT 信息，它是一个可变长度的非透明数据结构，其信息通常只需要发送方明白即可。格式如图 16 所示。

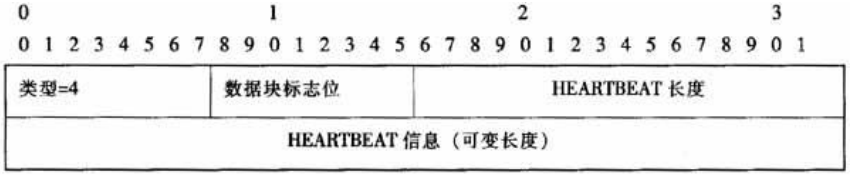

图 16 HEARTBEAT 数据块格式

数据块标志(8bit)：在发送方设置为全 0，并在接收方忽略。

HEARTBEAT 长度(16bit)：设置为数据块长度的字节数，包括数据块头和 HEARTBEAT 信息字段。

HEARTBEAT 数据块包含如下参数。

HEARTBEAT 信息：可变长度，其格式如图 17 所示。当该 HEARTBEAT 数据块发送到目的地传送地址时，发送方特定的 HEARTBEAT 信息字段通常包括关于发送方当前的时间信息。

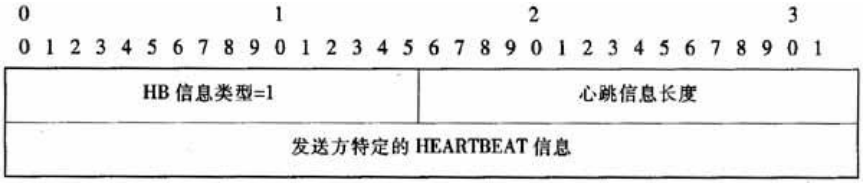

图 17 HEARTBEAT 信息参数格式

6.2.6　HeartBeat 证实(HEARTBEAT ACK)数据块的格式

SCTP 端点在收到对端点发来的 HEARTBEAT 数据块后，则发送该数据块作为响应。HeartBeat 证实总是向包含 HEARTBEAT 数据块的 IP 数据报中的起源 IP 地址发送，来作为对该 HEARTBEAT 数据块的响应。HeartBeat 证实数据块的结构如图 18 所示。

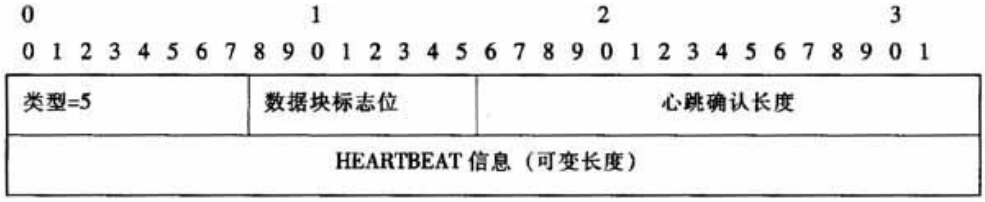

图 18　HEARTBEAT 数据块格式

数据块标志(8bit)：在发送方设置为全 0，并在接收方忽略。

HEARTBEAT 长度(16bit)：设置为数据块长度的字节数，包括数据块头和 HEARTBEAT 信息字段。

HeartBeat 证实数据块应包含如下参数。

HEABTBEAT 信息:可变长度,该字段的内容应当把 HEARTBEAT 请求数据块中的 HEARTBEAT 信息参数作为回送的响应，该参数字段包含一个可变长度的非透明的数据结构。

6.2.7　中止(ABORT)数据块的格式

SCTP 端点发送 ABORT 数据块来中止到对等端的偶联,ABORT 数据块中可以包含原因参数用来通知接收 ABORT 数据块的一方中止该偶联的原因。DATA 数据块不能同 ABORT 数据块捆绑在一个 SCTP 分组中。控制数据块(除 INIT、INIT ACK 和 SHUTDOWN COMPLETE)均可以同 ABORT 进行捆绑在一个 SCTP 分组中，但这些控制块都应放在 SCTP 分组中的 ABORT 数据块之前，否则这些控制数据块将被接收方忽略。

如果一个端点收到了格式错误或与不存在的偶联相关的 ABORT 消息，则应当舍弃该消息。此外，在任何情况下，端点收到一个 ABORT 消息后，都不能通过发送 ABORT 消息作为响应。ABORT 数据块的格式如图 19 所示。

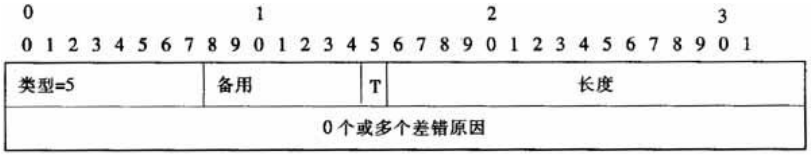

图 19　ABORT 数据块格式

数据块标志(8bit)：其中高 7 比特备用，在发送方设置为全 0，并在接收方忽略。

T 比特(1bit)：当发送方有一个 TCB 被破坏时则该 T 比特设置为 0,如果发送方没有 TCB，则把该比特设置为 1。

长度(16bit 无符号整数)：设置为该数据块的长度，包括数据块头和所有包含的差错原因字段。

ABORT 数据块还可以包含 0 个或多个差错原因参数，这些参数的格式可参见 6.2.10。

6.2.8　关闭偶联(SHUTDOWN)数据块的格式

偶联的端点可以使用这个数据块启动对该偶联的正常关闭程序，SHUTDOWN 数据块的格式如图 20 所示。

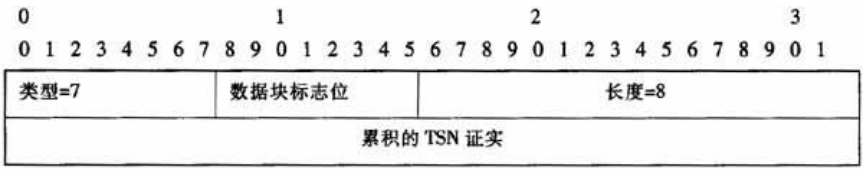

图 20　SHUTDOWN 数据块格式

数据块标志(8bit)：在发送方设置为全 0，并在接收方忽略。

长度(16bit 无符号整数)，用来指示 SHUTDOWN 数据块的长度，该字段设置为 8。

SHUTDOWN 数据块应包含如下参数。

累积的 TSN 证实(32bit 的无符号整数)，该参数包含了在任何间隔前收到的最后一个数据块的 TSN。由于 SHUTDOWN 消息不包含间隔证实块，因此不能用来对收到的失序 TSN 进行证实。

6.2.9　关闭证实(SHUTDOWN ACK)数据块的格式

在完成了关闭程序后必须使用该数据块来确认收到的 SHUTDOWN 数据块。SHUTDOWN ACK 数据块的格式如图 21 所示。

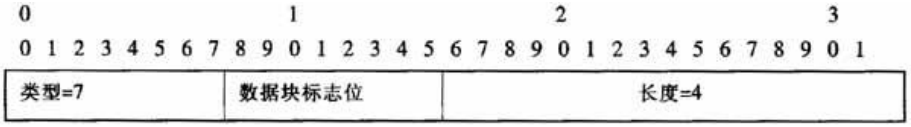

图 21　SHUTDOWN ACK 数据块格式

数据块标志位(8bit)：在发送方设置为全 0，并在接收方忽略。SHUTDOWN ACK 中不再包含其他参数。

6.2.10　操作差错(ERROR)数据块的格式

SCTP 端点发送该数据块向其对端点通知一些特定的差错情况。该数据块中可以包含一个或多个差错原因。一般操作差错不一定被看作是致命的，致命差错情况的报告一般使用 ABORT 数据块。ERROR 数据块格式如图 22 所示。

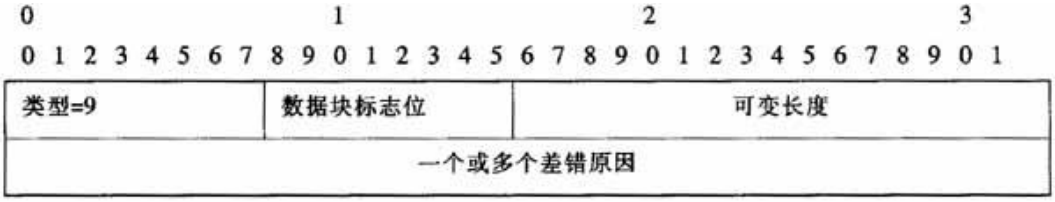

图 22　ERROR 数据块格式

ERROR 数据块包含如下参数。

数据块标志位(8bit)：在发送方设置为全 0，并在接收方忽略。

可变长度(16bit 无符号整数)：设置为该数据块的字节数，包括数据块头和所有包含的差错原因字段的长度。

差错原因参数的格式如图 23 所示。

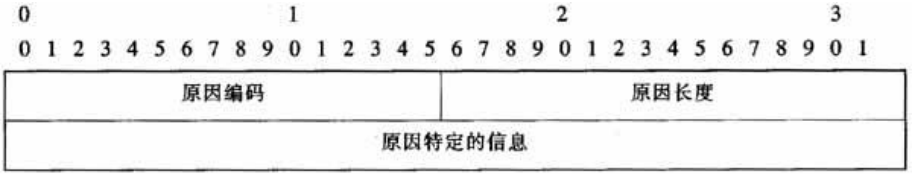

图 23　差错原因参数格式

原因编码(16bit 无符号整数)：定义了被报告的差错情况的类型。

原因编码的含义

1：无效的流标识符

2：丢失必备参数

3：过期的 Cookie 差错

4：资源耗尽

5：无法解析的地址

6：不识别的数据块类型

7：无效的必备参数

8：不识别的参数

9：无用户数据

10：关闭阶段收到 COOKIE

11：使用新的地址重新启动偶联

原因长度(16bit 无符号整数)：设置为该参数的字节数，包括原因编码、原因长度和原因特定的信息字段。

原因特定的信息：可变长度，该字段用来携带差错的详细情况。

6.2.10.1　无效的流标识符原因参数的格式

差错原因无效的流识别符用来指示端点收到了一个关于不存在的流的 DATA 数据块。其格式如图 24 所示。

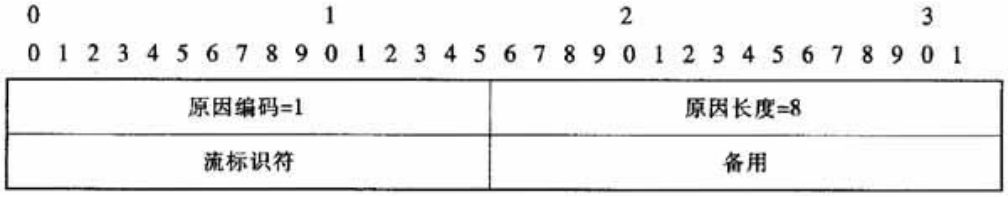

图 24　无效的流识别符原因参数的格式

流识别符：(16bit 无符号整数)：包含了接收购差错的 DATA 数据块的流标识符。

备用字段(16bit)：由发送方设为全 0，在接收方忽略。

6.2.10.2　丢失必备参数原因参数的格式

丢失必备参数差错原因用来指示一个或多个必备的参数在收到的 INIT 或 INIT ACK 数据块中丢失。该原因参数的格式如图 25 所示。

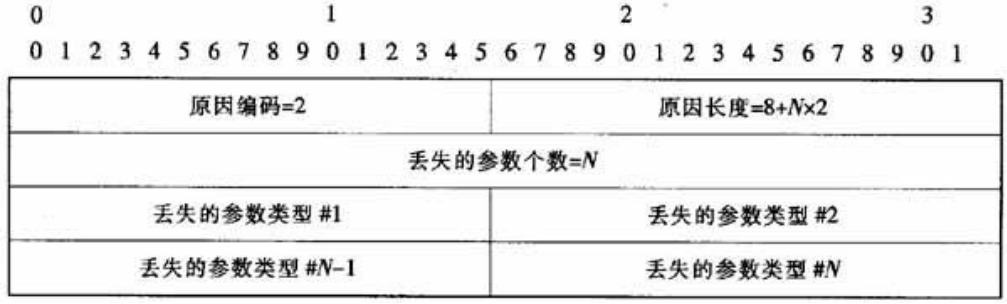

图 25　丢失必备参数原因参数的格式

该原因参数包含如下参数。

丢失的参数个数(32bit 无符号整数)：该字段用来指示丢失的参数个数。

丢失的参数类型(16bit 无符号整数)：每个字段都应包含丢失的必备参数号。

6.2.10.3　过期的 COOKIE 差错原因参数的格式

过期的 COOKIE 差错原因参数用来指示收到的有效的 State Cookie 已经过期了。该原因参数的格式如图 36 所示。

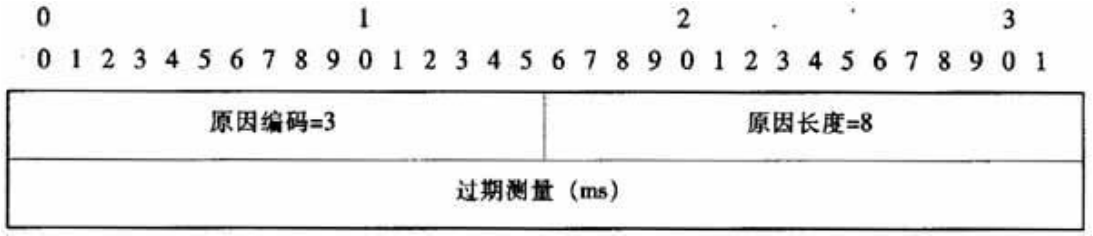

图 26　过期的 COOKE 差错原因参数格式

该原因参数中包含如下参数。

过期测量(32bit 无符号整数)：该字段包含了当前时间和 State Cookie 过期时的时间差值(用毫秒表示)。该差错原因的发送方可以通过在该字段中包含一个非 0 的值来报告 State Cookie 过期了多长时间。如果发送方不希望提供这个信息，则该字段设置为 0。

6.2.10.4 资源耗尽原因参数的格式

资源耗尽差错原因用来指示发送方的资源已经耗尽，通常情况下该查错原因与 ABORT 数据块一起发送，该原因参数的格式如图 27 所示。

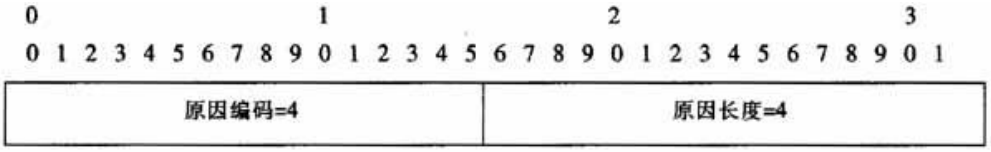

图 27 资源耗尽原因参数的格式

6.2.10.5 不可解析的地址原因参数的格式

不可解析的地址用来指示发送方不能解析特定的地址参数(即发送方不支持该类地址类型)，通常情况下该查错原因与 ABORT 数据块一起发送。该原因参数的格式如图 28 所示。

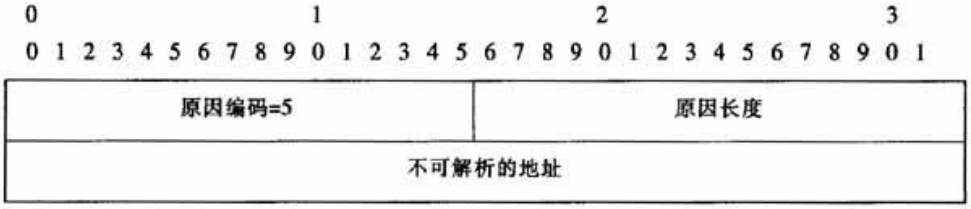

图 28 不可解析的地址差错原因参数格式

不可解析的地址：可变长度，不可解析的地址字段中包括不能解析的完整的地址参数或主机名参数(类型、长度和地址值)。

6.2.10.6 不识别的数据块类型原因参数的格式

如果接收方不理解数据块且数据块类型比特中的高位比特设为 1，则把不识别的数据块类型错误返回给数据块的产生者，该原因参数的格式如图 29 所示。

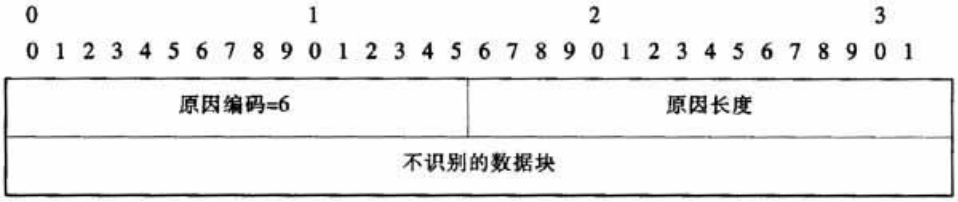

图 29 不识别的数据块差错原因参数格式

不识别的数据块(可变长度)：该字段包含 SCTP 分组中不识别数据块的数据块类型、数据块标志和数据块长度。

6.2.10.7 无效的必备参数原因参数的格式

当一个必备参数被设置成无效值时，则向 INIT 或 INITACK 的生成者返回无效的必备参数差错。该原因参数的格式如图 30 所示。

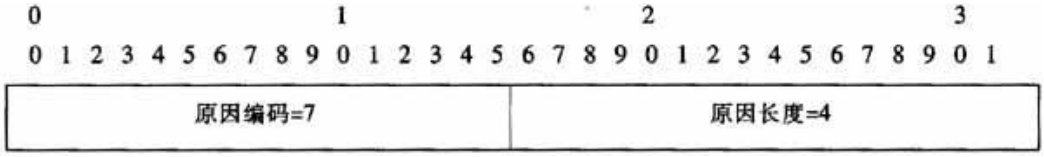

图 30 无效的必备参数原因参数的格式

6.2.10.8 不识别的参数原因参数的格式

如果接收方不能识别 INIT ACK 数据块中一个或多个任选参数，则向 INIT ACK 数据块的

产生者返回不识别的参数的差错原因。该原因参数的格式如图 31 所示。

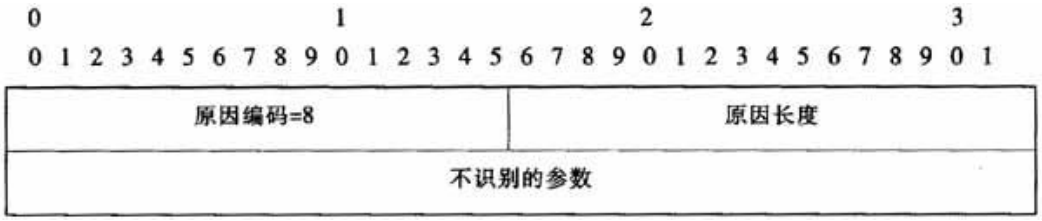

图 31　不识别的参数差错原因参数格式

不识别的参数：可变长度，该参数字段包含了从 INIT ACK 数据块中复制的完整的不识别的参数。当 COOKIE ECHO 数据块的发送者希望报告不识别的参数时，这个参数通常是包含在 ERROR 数据块中与 COOKIE ECHO 数据块捆绑在一起发送作为对 INIT ACK 的响应。

6.2.10.9　无用户数据原因参数的格式

如果收到的 DATA 数据块中未包含用户数据，则把这个差错原因返回给 DATA 数据块的产生者。该原因参数的格式如图 32 所示。

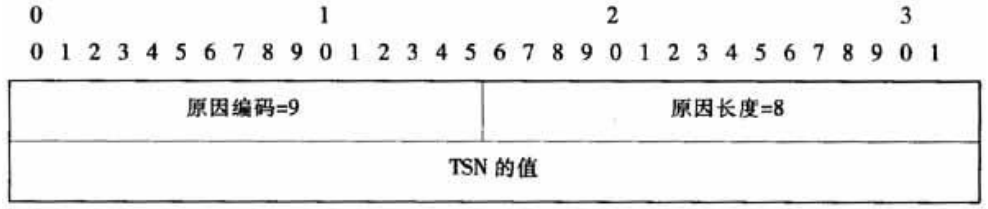

图 32　无用户数据差错原因参数格式

TSN 的值(32bit 无符号整数)：该字段包含接收到的这个没有用户数据的 DATA 数据块的 TSN。这个原因值通常是在 ABORT 数据块中返回的。

6.2.10.10　关闭期间收到 Cookie 原因参数的格式

当端点处于 SHUTDOWIV-ACK-SENT 状态时，又收到 COOKIE ECHO 时则发送该差错原因。返回这个差错原因的 ERROR 数据块通常与重发的 SHUTDOWN ACK 数据块捆绑在一起发送。该原因参数的格式如图 33 所示。

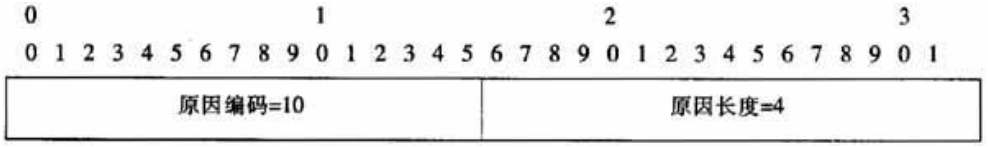

图 33　关闭期间收到 COOKIE 的原因参数的格式

6.2.10.11　使用新地址重新启动偶联

当在现存的偶联上收到了 COOKIE ECHO 数据块，而 COOKIE ECHO 数据块又向该偶联中增加了先前没有的地址，此时使用该错误原因，并把新增加的地址作为差错信息在该参数中传送，这个差错原因通常都在 ABORT 中发送，用来拒绝 COOKIE ECHO 数据块。该原因参数的格式如图 34 所示。

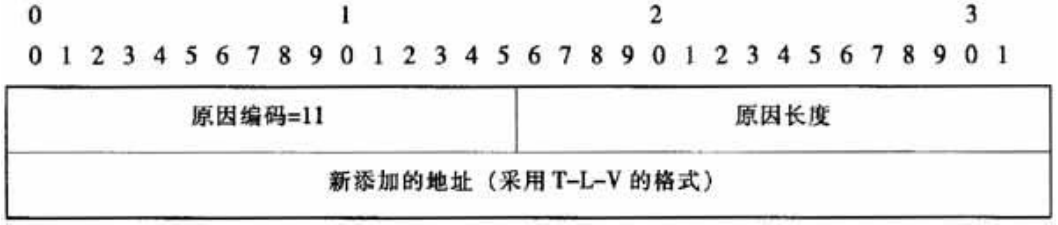

图 34　使用新地址重新启动偶联原因参数的格式

6.2.11　状态 COOKIE(COOKIE ECHO)数据块的格式

该数据块只在启动偶联时使用，它由偶联的发起者发送到对端点，用来完成启动过程。这个数据块必须在该偶联上发送的 DATA 数据块前发送，但可以同其他的 DATA 数据块捆绑到同一个 SCTP 分组中。COOKIE ECHO 数据块的格式如图 35 所示。

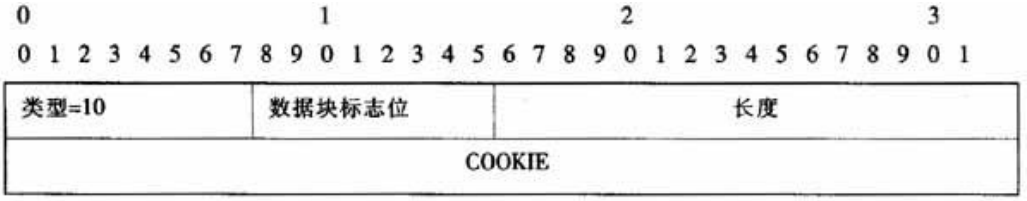

图 35 COOKIE EHCO 数据块的格式

COOKIE EHCO 数据块应包含如下参数。

数据块标志位(8bit)：在发送方设置为全 0，在接收方忽略。

长度：(16bit 的无符号整数)：设置为该数据块长度的字节数，包括 4 字节的数据块头和 COOKIE 的长度。

COOKIE(可变长度)：该字段必须包含从前一个 INIT ACK 数据块的状态 COOKIE 参数中收到的准确的 COOKIE，使用 COOKIE 时应尽可能的小从而保证互操作性。

6.2.12　COOKIE 证实(COOKIE ACK)数据块的格式

这个数据块只在启动偶联时使用，它用来证实收到 COOKIE EHCO 数据块。这个数据块必须在该偶联上发送任何 DATA 或 SACK 数据块前发送，但这个数据块可以与一个或多个 DATA 或 SACK 数据块捆绑在一个 SCTP 分组中发送。COOKIE ACK 数据块中没有其他参数。COOKIE ACK 数据块的格式如图 36 所示。

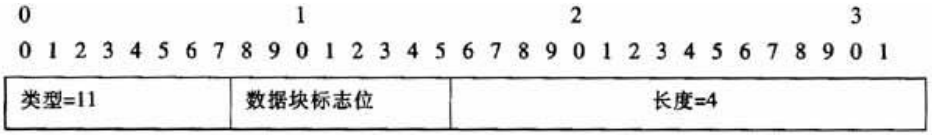

图 36 COOKIEACK 数据块格式

COOKIE ACK 数据块中只包含数据块标志(8bit)，在发送方设置为全 0，并在接收方忽略。

6.2.13　关闭完成(SHUTDOWN COMPLETE)数据块的格式

该数据块在完成关闭程序后用来确认收到 SHUTOOWN ACK 数据块。SHUTDOWN COMPLETE 数据块中不包含其他参数。SHUTDOWN COMPLETE 数据块如图 37 所示。

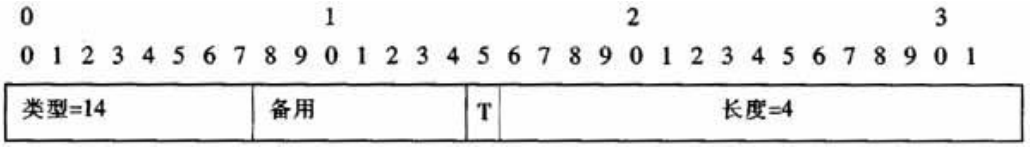

图 37　SHUTDOWN COMPLETE 数据块格式

数据块标志(8bit)：其中高 7 比特备用，备用比特在发送方设为全 0，在接收方忽略。

T 比特(1bit)：当发送方有一个 TCB 被破坏时则该 T 比特设置为 0;如果发送方没有 TCB，则把该比特设置为 1。

7　SCTP 端点的维护的参数和相关建议值

本节中对用于 SCTP 协议实施的参数集进行了说明，但不是作为强制要求。这里提到的 SCTP 参数只是一个有限集合、开发者可以根据具体情况实施优化或增加各自的附加参数。

7.1　对应每个 SCTP 实例所需的参数

偶联：当前偶联的列表，用来对应每个偶联的数据用户，这可以采用 HASH 表的方式来

实现或者使用其他依赖于实施的结构,数据用户可以被作为表示信息来处理,如文件描述符、命名管道指针或者是依赖与 SCTP 实施的表的指针。

密钥：端点使用密钥来计算 MAC，这应当是有足够长度的保证加密质量的随机数，RFC1750 对密钥的选择有一些建设性的说明。

地址列表，这是一组该实例绑定的 IP 地址的列表，这个信息在 INIT 或 INIT ACK 数据块中被传递到对端点。

SCTP 端口：端点绑定的本地 SCTP 端口号。

7.2 对应每个偶联 SCTP 端点所需的参数

对端验证标签：该标签在每个分组中发送，其值应当等于从 INIT 或 INIT ACK 数据块中收到的值。

本地验证标签：该标签期望在每个入局的分组中，在 INIT 或 INIT ACK 数据块中发送。

状态：用来指示偶联所处状态的状态变量，即 COOKIE-WAIT，COOKIE-ECHOED，ESTABLISHED,SHUTDOWN-PENDING,SHUTDOWN-SENT,SHUTDOWN-RECEIVED,SHUTDOWN-ACK-SENT 等。

对端传送地址列表,对端点绑定的一组 SCTP 传送地址列表。这个信息是从 INIT 或 INIT ACK 数据块中得到的，并用来把一个入局的分组同给定的偶联关联起来，通常来讲，这个信息都是用 HASH 表或关键字进行快速查询，并接入 TCB。

首选通路：即对端点当前的首选目的地传送地址，它也可以用来规定所在端点的起源传送地址。

全局差错计数：整个偶联的所有差错计数。

全局差错门限：这个门限用来控制偶联，当全局差错计数达到了这个门限，将导致偶联的关闭或中止。

对端的 Rwnd：对端的 Rwnd 的当前计算值。

下一个 TSN:下一个 TSN 号码被分配给一个新的 DATA 数据块,它可以在 INIT 或 INIT ACK 数据块中发送到对端,并且这个号码每次配给 DATA 数据块(通常的情况是在发送前或者是分段时)后加 1。

最后收到的 TSN：这是最后一个按顺序收到的 TSN，这个值最初是使用对端的初始 TSN 来设定的，并在收到的 INIT 或 INIT ACK 数据块中携带，然后把该值减 1 得到。

映射数组:这是一个以比特或字节定义的数组,它用来指示哪个收到的 TSN 是失序的(相对于最后收到的 TSN)，如果不存在不连续，即没有收到失序的分组，这个数组将被设置为全 0，这个结构可以用一个循环缓冲或比特数组来实现。

Ack 状态:这个标志位用来指示下一个收到的分组是否是应当响应 SACK,其初始值为 0,每收到一个分组，这个值加 1，如果这个值到达 2 或者是更高，则发送一个 SACK，并把这个值重新设置为 0。这个状态只适用于没有 DATA 数据块失序的的情况，当出现了 DATA 数据块失序的情况，则不需要延时发送 SACK。

入局流：一个用来跟踪入局流的数组结构，通常包含下一个希望收到的流顺序号码和可能的流号码。

出局流：一个用来跟踪出局流的数组结构，通常包含下一个希望在某个流上发送的流顺序号码。

重装队列：一个用于对分段数据的重装队列。

本地传送地址列表：该偶联绑定的本地 IP 地址。

偶联的 PMTU：对所有对端点传送地址发现的最小的 PMTU。

7.3 对应每个传送地址所需的参数

对应于从 INIT 或 INITACK 中收到的对端地址列表中的每个目的地传送地址，端点都需

要维护以下这些参数。

差错计数：对该目的地的当前差错计数。

差错门限：对该目的地的当前差错门限，即当差错计数到达该值时，则标记到该目的地的偶联停止。

Cwnd：当前的拥塞窗口。

Ssthresh：当前 ssthresh 的值。

RTO：当前的重发超时取值。

SRTT：当前的平滑双向时延值。

RTTVAR：当前的 RTT 变化。

部分字节证实：在拥塞避免模式下 cwnd 增加的跟踪方法。

状态：目的地的当前状态，即 DOWN、UP、ALLOW-HB、NOHEARTBEAT 等状态。

PMTU；当前已知的通路 MTU。

每个目的地的定时器：针对每个目的地使用的定时器。

RTO-期间：这个标志位用来跟踪发送到该地址的 DATA 数据块是当前用来计算 RTT 的。如果这个标志位为 0，则下一个发送到该目的地的 DATA 数据块将用来计算 RTT 并设置这个标志位，每次 RTT 计算完成后(DATA 数据块被 SACK 证实后)，清除该标志位。

最后使用的时间：这个时间用来指示最后向该目的地发送分组的时间，它也用来确定是否需要发送 HEABTBEAT。

7.4 需要的通用参数

出局队列：出局 DATA 数据块的队列，

入局队列：入局 DATA 数据块的队列。

7.5 SCTP 参数的建议值

RTO 的初始值：3s；

RTO 的最小值：1s；

RTO 的最大值：60s；

RTO Alpha：1/8；

RTO.Beta：1/4；

有效的 COOKIE 寿命：60s；

偶联的最大重传次数：10 次；

通路的最大重传次数：5 次；

INIT 的最大重传次数：8 次；

HB 测试周期：30s。

8 SCTP 的程序

SCTP 的程序包括一些各节描述的内容：偶联的建立、数据的传递、拥塞控制、故障管理和偶联关闭等 5 个部分的内容。此外在 SCTP 的程序中规定了一些安全性的内容。

8.1 偶联的建立程序

为了简化程序描述，对于以下偶联的建立程序，使用 SCTP 端点 A 和 SCTP 端点 Z 来进行描述，其中假定 SCTP 端点 A 试图与 SCTP 端点 Z 建立偶联。

在从 SCTP 端点 A 向 SCTP 端点 Z 能传送第一个用户数据块之前，两个端点必须完成启动程序，以建立它们之间的 SCTP 偶联。

端点的 SCTP 用户应使用 ASSOCIATE 原语来请求启动到另一个 SCTP 端点的偶联。

从 SCTP 用户的观点来看，在没有发起的 ASSOCIATE 原语的情况下，SCTP 偶联可以隐含地打开，通过启动端点发送第一个用户数据到目的地端点的方式来实现。启动 SCTP 将使用

INIT/INIT ACK 中所有必备和任选参数的缺省值。一旦偶联建立起来，在两端就打开了用于数据传送的单向流。

8.1.1　偶联的正常建立

启动程序包括以下步骤(假定 SCTP 端点 A 试图与 SCTP 端点 Z 建立偶联，且 Z 接受了新的偶联)。

1)“A”首先向“Z”发送一个 INIT 数据块。在 INIT 数据块中，“A”必须在启动标签字段里提供它的验证标签(Tag_A)。Tag_A 应当是 1 到 4294967295 的中的一个随机数(见 5.3.1 节关于标签值选择的描述)。A 在发送了 INIT 后，启动 T1-init 定时器并进入 COOKIE-WAIT 状态。

2)“Z”在收到 INIT 数据块后应立即用 INIT ACK 数据块响应。INIT ACK 数据块中的目的地 IP 地址必须设置成 INIT ACK 数据块响应的那个 INIT 数据块的起源 IP 地址。在这个响应数据块中，除了填写其他参数外，“Z”必须将验证标签字段置成 Tag_A，将它自己的启动标签字段置成 Tag_Z。而且“Z”必须产生一个状态 COOKIE，与 INIT ACK 一起发送。

注：在发出带有状态 COOKIE 参数的 INIT ACK 后，Z 不分配任何资源，也不为新偶联保持任何状态。

3)根据从“Z”收到的 INIT ACK，“A”需要停止 T1-init 定时器并离开 COOKIE-WAIT 状态。然后“A”会把从 INIT ACK 数据块收到的状态 Cookie 在 COOKIE ECHO 数据块中发送，A 启动 T1—cookie 定时器并进入 COOKIE-ECHOED 状态。

注：COOKIE ECHO 数据块能够被与任何出局未决的 DATA 数据块捆绑在一个分组中，但 COOKIE ECHO 必须是分组里的第一个数据块。除非收到返回 COOKIE ACK，否则发送者不能给对端发送其他分组。

4)根据收到的 COOKIE ECHO 数据块，端点“Z”创建 TCB 后，转移至 ESTABLISH 状态，然后用一个 COOKIE ACK 数据块响应。一个 COOKIE ACK 数据块可以与任何未决的 DATA 数据块(和 / 或 SACK 数据块)捆绑在一起,但是 COOKIE ACK 数据块必须是分组中的第一个数据块。在接收到的有效 COOKIE ECHO 数据块后,“Z”可以向 SCTP 用户发送 COMMUNICATION UP 通知。

5)根据收到的 COOKIE ACK，端点“A”会从 COOKIE-ECHOED 状态转移至 ESTABLISHED 状态，并停止 T1-cookie 定时器。“A”也可以用 COMMUNICATION UP 通知 ULP 偶联建立成功。

INIT 或 INIT ACK 数据块不能与其他任何数据块捆绑在一起。它们必须是在携带它们的 SCTP 分组中出现的唯一的数据块。端点必须向给它发送 INIT 数据块的 IP 地址发送 INIT ACK。

如果端点收到 INIT,INIT ACK 或 COOKIE ECHO 数据块,但由于在收到的 INIT 和 INIT ACK 中缺少必备参数或者有无效的参数值，或缺少本地资源，端点必须用 ABORT 数据块响应。端点应在 ABORT 数据块中包括错误原因参数来说明中止的原因，例如丢失必备参数的类型等。在发出的包含 ABORT 数据块的 SCTP 分组里，验证标签字段必须被置为对端的启动标签值。

在收到偶联的第一个 DATA 数据块后，端点必须用 SACK 立即响应 DATA 数据块。

当产生了 TCB，每个端点必须将它内部累积 TSN ACK 的值设置为：初始 TSN-1。

8.1.1.1　流参数的处理

在 INIT 和 INIT ACK 数据块中,数据块的发送者将指出在偶联中希望允许的出局的流(OS)的数量，以及它允许的从其他端点接收的最大入局的流数量。

在从对端收到流的配置信息后，每个端点将执行下列检查：如果对端的 NIS 比本端点的 OS 小，意味着对端不能支持本端想要配置的出局流的数量，则本端点要么使用 NIS 作为出局流数量，要么中止偶联并向 ULP 报告对端资源短缺。

在偶联启动后，每个端点发出的流识别符的有效范围应为 0 到本地 OS 和远端 NIS 中的最小值-1。

8.1.1.2　地址参数的处理

在偶联启动的过程中，端点将使用下列规则来发现和收集对端的目的地传送地址。

1)如果在收到的 INIT 或 INIT ACK 数据块中没有出现地址参数，端点将把到达的数据块中的起源 IP 地址提取并记录下来，与 SCTP 源端口号码一起作为对端惟一的目的地传送地址。

2)如果在收到的 INIT 或 INIT ACK 中包含有主机名，端点将主机名解析成一组 IP 地址，并把解析得到的 IP 地址与 SCTP 源端口组合作为对端的传送地址。此时端点必须忽略其他出现在收到的 INIT 或 INIT ACK 中 IP 地址参数。

在 INIT 的接收者解析主机名时，对 SCTP 有潜在的安全影响。如果 INIT 接收者根据收到的数据块中的主机名进行地址解析，而接收者用来解析主机名的机制又有较长时延(如 DNS 查询)，这样接收者可能会在建立 State Cookie 和释放本地资源之前由于等待主机名解析结果时打开本地资源而受到资源攻击。因此，如果主机名翻译可能造成长时延，则 INIT 的接收者必须推迟主机名解析，直到收到对端的 COOKlEECHO 数据块之后再进行地址解析。在这种情况下，INIT 的接收者应该用收到的主机名建立状态 COOKIE
(代替目的地传送地址)，并发送 INIT ACK 给发送 INIT 的源 IP 地址。

INIT ACK 的接收者将立即尝试用收到的数据块解析主机名。在主机名被成功地解析前，INIT 或 INITACK 的接收者不能向对端发送用户数据(单独或捆绑)。

如果主机名解析不成功，端点必须立即用“无法解析的地址”错误发送 ABORT 给对端。ABORT 将发送给源 IP 地址，即发出的最后一个分组的地址。

3)如果在收到的 INIT 或者 INIT ACK 数据块中只有 IPv4/IPv6 地址，则接收方应当提取并记录接收到的数据块中的所有传送地址以及发送 INIT 或 INIT ACK 数据块的起源 IP 地址。传送地址是通过 SCTP 的起源端口(包含在公共分组头中)和在 INIT 和 INIT ACK 数据块中的 IP 地址参数和 IP 数据报中的起源 IP 地址的组合得到的。接收方向对端发送分组时应当只使用这些传送地址作为目的地的传送地址。

在一些情况下(例如，当实施不控制用于发送的起源 IP 地址时)，一个端点可能需要在它发送到对端的 INIT 或 INIT ACK 中包含所有可能的 IP 地址。

当所有的传送地址按照上述规则从 INIT 或 INIT ACK 数据块中得到后，端点将选择一个传送地址作为初始的首选路由。

INIT ACK 必须向包含 INIT 数据块的 IP 数据报的起源地址发送。INIT 的发送方可以在 INIT 数据块中包含“支持的地址类型”参数，指示可接受的地址类型。如果这个参数存在，则 INIT 数据块的接收方必须使用 INIT 数据块的“支持的地址类型”参数中指示的地址类型来响应 INIT 数据块，如果接收方不希望或者是不能使用对端指示的地址类型时，则用“不可解析的地址”差错原因来中止该偶联。

在某些情况下，INIT ACK 的接收方由于不支持地址类型而无法解析出地址参数，则可以中止启动程序，并通过重新发送包含“支持的地址类型”参数的 INIT 数据块给对端指示希望使用的地址类型，尝试重新建立一个偶联。

8.1.1.3 生成状态 COOKlE

当发送的 INIT ACK 响应 INIT 数据块时，INIT ACK 的发送方格创建一个状态 COOKIE，并且在 INITACK 中的状态 COOKIE 参数中发送。在状态 COOKIE 中，发送方应当包含消息鉴权码、状态 COOKIE 创建的时间标记、状态 COOKIE 的寿命以及所有建立该偶联所需的信息。

生成状态 COOKIE 时应使用以下步骤。

1)使用收到的 INIT 以及发出的 INIT ACK 数据块中的信息创建一个偶联 TCB；

2)在 TCB 中，把创建时间设置为当前的时间，并且设置 COOKIE 寿命为协议参数“有效 COOKIE 寿命”的值；

3)从 TCB 中识别并收集重新创建 TCB 所需信息的最小子集，并且用这个子集的信息和密钥生成一个 MAC，并且通过组合信息子集和得到的 MAC 来生成一个状态 COOKIE。

在发送了包含状态 COOKIE 参数的 INIT ACK 后，发送方应当删除 TCB 和与新偶联有关的任何本地资源，这样可以避免资源被恶意占用。

用来生成 MAC 的散列算法对于 INIT 的接收方是专用的。MAC 的使用是为了避免业务受到攻击，因此密钥应当是随机的，也应当经常更换。同时状态 COOKIE 中的时间标记也可以用来确定用来验证 MAC 的密钥。为了保证互操作性，SCTP 实现应当保证 COOKIE 尽可能的小。

8.1.1.4　状态 Cookie 的处理

当一个端点收到带有状态 COOKIE 参数的 INIT ACK 数据块后，则必须立即用收到的状态 COOKIE 向对等端响应一个 COOKIE ECHO 数据块，发送方也可以在 COOKIE ECHO 数据块之后增加任何未决的 DATA 数据块。

在发送了 COOKIE ECHO 数据块后，端点应当启动 T1-COOKIE 定时器，如果定时器超时了，则端点应当重新传送 COOKIE ECHO 数据块并重新启动 T1-COOKIE 定时器。这个过程将一直重复，直到端点接收到一个 COOKIE ACK 数据块，或者是到达了 Max.Init.Retransmits 的门限，此时应标记对端点为不可达[并使该偶联进入关闭(CLOSEC)状态]。

8.1.1.5　切状态 Cookie 的鉴权

当一个端点收到了其他端发来点 COOKIE ECHO 数据块，且偶联尚未建立，则该端点应采取以下动作。

1)使用状态 COOKIE 中携带的 TCB 数据和密钥来计算 MAC，RFC2104 中介绍了如何生成和计算 MAC。

注：状态 COOKIE 中的时间标记可以用来确定使用的密钥。

2)状态 COOKIE 的鉴权是把计算出的 MAC 同在状态 COOKIE 中携带的 MAC 进行比较，如果比较失败，则把这个包含 COOKIE ECHO 和数据块(可能包含)的 SCTP 分组直接丢弃，而不产生任何指示。

3)对状态 COOKIE 中的创建时间标记和当前的本地时间比较，如果封装的时间已经超过了在状态 COOKIE 中携带的寿命值，则这个包含 COOKIE ECHO 和数据块(可能包含)的 SCTP 分组庭当被丢弃，同时这端点必须向对端点传送一个差错原因为“过期的 COOKIE(Stale Cookie)”的 ERROR 数据块。

4)如果状态 COOKIE 有效，则使用 COOKIE ECHO 携带的 TCB 数据信息，向发送 COOKIE ECHO 数据块的发送方创建偶联，并进入到建立(ESTABLISHED)状态。

5)向对端点发送 COOKIE ACK 数据块，用来确认收到了 COOKIE ECHO 数据块。COOKIE ACK 可以同其他未决的 DATA 数据块或者是 SACK 数据块捆绑在一起，但 COOKIE ACK 数据块必须是 SCTP 分组中的第一个数据块。

6)任何与 COOKIE ECHO 捆绑的数据块都可以用 SACK 立即确认(随后的 DATA 数据块的确认应当遵循 8.2.2 节的规定)。如果 SACK 与 COOKIE ACK 捆绑在一个 SCTP 分组中，COOKIE ACK 必须是是第一个数据块。

如果从一个端点收到 COOKIE ECHO 后，在接收的端点发现已经有一个偶联存在，则应遵循 8.1.2 节规定购程序。

8.1.2　对重复的或不期望的 INIT、INIT ACK、COOKIE ECHO 和 COOKIE ACK 的处理

在偶联的存活期间(可以处于任何可能的状态)，端点有可能从其对端点收到以下用于建立偶联的数据块(INIT、INIT ACK、COOKIE ECHO 和 COOKIE ACK)，接收方应当把这些用于建立的数据块当作重复数据块来处理，并按照本节规定的内容进行处理。

注：端点收到数据块只能是发自或来源于与这个端点有偶联的 SCTP 传送地址，因此端点把这类数据块当作当前偶联的一部分。

以下的情况可能导致重复或不期望的数据块。

1)对等端的看机而没有被检测出，或自身重新启动，并发出了一个新的 INIT 数据块试

图用来恢复偶联；

2)两端同时尝试启动偶联；

3)一个用于建立当前偶联或建立已不存在的偶联的过期分组中的数据块；

4)由攻击者生成的错误分组；

5)对端从未收到 COOKIE ACK 并且在重复发送 COOKIE ECHO。

为了能识别并正确处理这些情况，应当采用下节中规定的原则：

8.1.2.1 在 COOKIE-WAIT 或 COOKIE-ECHOED 状态下收到 INIT

这种情况通常用来指示启动冲突，即每个端点同时希望尝试同对端点建立偶联。

在 COOKIE-WAIT 或 COOKIE-ECHOED 状态下收到 INIT，一个端点必须使用 INIT ACK 来响应，在这个 INIT ACK 数据块中，应当包含该节点先前发送的 INIT 数据块中相同参数(包括未变化的验证标签)，和由该节点生成的参数以及从 INIT 数据块中收到参数一起发送过去，端点也应在 INIT ACK 中当产生一个状态 COOKIE，端点使用其先前发送的 INIT 中的参数来计算状态 COOKIE。

在此之后，端点不需要改变其状态，T1-init 定时器应当继续运行，并且不破坏相关的 TCB。TCB 存在时处理状态 COOKIE 的正常程序可以解决对于一个偶联的两个重复的 INIT。

对于一个处于 COOKIE ECHO 状态的端点必须使用它自己和对端的标签信息来移植 Tie-Tags。(参见 8.1.2.2 节中对 Tie-Tag 的描述)

8.1.2.2 在除 CLOSED、COOKIE-ECHOED 和 COOKIE-WAIT 之外的状态下收到不期待 INIT

除非另外指出，在收到对某个偶联的一个不期望的 INIT 时，端点应当产生一个带有状态 COOKIE 的 INIT ACK，发出 INIT ACK 的端点必须把其当前的验证标签和对端的验证标签复制到状态 COOKIE 中的备用部分，这两个位置我们称为 Peers-Tie-Tag 和 Local-Tie-Tag。INIT ACK 必须要包含一个新的验证标签(随机产生的，参见 8.1.3.1 节)，该端点的其他参数应当从现存偶联的参数中复制到 INIT ACK 和 COOKIE 中(例如：输出流的数量)。

在发送完 INIT ACK 之后，端点将不采取任何动作，即当前的偶联包含当前的状态以及相对应的 TCB 都不应发生变化。

注：只有当一个 TCB 存在而偶联不处于 COOKIE-WAIT 状态时才移植 Tie-Tags，对于一个正常偶联的 INIT(即端点处

于 COOKIE-WAIT 状态),Tie-Tags 必须被设置为 0(用来指示不存在前一个 TCB)。INIT ACK 和状态 COOKIE 的移植规定如 8.1.2.1 节。

8.1.2.3 不期望的 INIT ACK

如果一个节点不是在 COOKIE-WAIT 状态下收到 INIT ACK，则端点应当舍弃该 INIT ACK 数据块。一个不期望的 INIT ACK 通常用来指示对一个过期的或者重复的 INIT 的处理。

8.1.2.4 TCB 存在时对 COOKIE ECHO 的处理

当节点在任何状态下收到了对于一个已存在偶联的 COOKIE ECHO 数据块(即未处于关闭状态)时应适用以下原则。

1)按照 8.1.1.5 节步骤 1 的描述来计算 MAC。

2)按照 8.1.1.5 节步骤 2 的描述对状态 COOKIE 进行鉴权(这是上面的 8.1.2 情况 3 或 4)。

3)用当前时间来比较状态 COOKIE 中的时间标记,如果比较结果比状态 COOKIE 中携带的寿命值要大，且状态 COOKIE 中的验证标签又与当前偶联的验证标签不符，则包含 COOKIE ECHO 和任何 DATA 数据块的分组应当被丢弃。

端点也必须发送带有差错原因为“过期的 COOKIE 的”ERROR 数据块给对端点(这是上面的 8.1.2 情况 3 和 4)。

如果验证标签和状态 COOKIE 与当前偶联的验证标签相匹配，尽管超过了寿命值则仍认

为该状态 COOKIE 有效(8.1.2 情况 5)。

4)如果状态 COOKIE 被证实是有效的，则把 TCB 解包到临时的 TCB。

5)可参见表 2 来确定当前要采取的动作。

表 2　TCB 存在时对 COOKIE ECHO 的处理

Local Tag	Peers Tag	Local-Tie-Tag	Peers-Tie-Tag	Action/ Description
X	X	M	M	1)
M	X	A	A	2)
M	O	A	A	2)
X	M	O	O	3)
M	M	A	A	4)

注：

X 表示标签与现存的 TCB 不匹配；

M 表示标签与现存的 TCB 相匹配；

O 表示在 COOKIE 中没有 Tie-Tag (示知)；

A 表示所有情况即 X、M 和 O 的情况。

对表 2 中未给出的情况，应当直接丢弃 COOKIE，而不用给出通知。

采取动作。

1)在这种情况下，对端可能已经重启动了，当一个端点识别出对端可能重启动时，除进行以下动作外，现存的进程被当作在一个 COOKIE EeHo 后收到 ABORT 的情况来处理。

—任何 SCTP 数据块都可以被保留(这是一个实施决定的选项)；

—向高层发送一个 BESTART 指示，而不是发送 COMMUNICAION LOST 指示。

所有与对端点相关的的拥塞控制参数(例如：cwnd 和 ssthresh)必须被复位为初始值(参见 8.2.2.1)。

如果 COOKIE 指示有一个新的地址被添加到重启动的偶联中，则整个 COOKIE 都应被舍弃。此时应发送一个包含差错原因为“使用新地址重新启动偶联”的 ABORT 数据块。同时差错原因中应当给出新添加到重新启动的偶联中的地址列表，这些地址应该是那些在原来偶联中没有的地址。这样就可以保证重启动偶联程序来恢复偶联。

之后这个端点将进入 ESTABLISHED 状态。

如果端点是在 SHUTDOWN-ACK-SENT 状态并且识别出对端重新启动(动作 1)，则它不需要重新建立一个新的偶联，而是向对端重新发送一个 SHUTDOWN ACK，同时在向对端发送一个差错原因为“关闭期间收到 COOKIE”的 ERROR 数据块。

2)在这种情况下，两端都可以同时尝试重新建立偶联，但对端点在响应了本端点的 INIT 后又启动了 INIT，因此，对端点可以在不知道发送的前一标签时选择一个新的验证标签。端点应当进入或停止在 Es-tablished 状态，但必须要用收到的 COOKIB 更新其对端的验证标签，停止任何运行的 init 和 COOKIE 定时器，并发送 COOKIE ACK。

3)在这种情况下，本端的 COOKIE 迟到，在它到达本端点前，发送的 INIT 和收到的 INIT ACK 和最后发出的应带有与对端相同 TAG 的 COOKIE 中使用了自己的一个新的 TAG，此时这个 COOKIE 应当被丢弃，并且不产生任何通知，端点不需要改变状态，并且应让定时器继续运行。

4)当本地和远端的标签能够匹配时，端点也应当进入 Established 状态，它应停止任何运行的 INIT 或 COOKIE 定时器，并且发送一个 COOKIE ACK。

注：“对端的验证标签”是指收到的 INIT 或 INIT ACK 数据块中启动标签字段中的标签。

8.1.2.5　对重复的 COOKIE ACK 的处理

除非是处于 COOKIE ECHOED 状态，否则在其他状态下节点收到 COOKIE ACK 数据块后都直接丢弃，并不产生指示。

8.1.2.6　对过期的 COOKIE 差错处理

收到差错原因为“过期的 COOKIE”的差错数据块，可以用来指示以下几种可能的事件。

1)在发送方发出状态 COOKIE 未处理前，偶联已经不能成功建立；

2)一个过期的状态 COOKIE 在建立完成后被处理；

3)接收到一个过期的状态 COOKIE，而接收方已经对该偶联不感兴趣，并且丢失了 ABORT 数据块。

在处理带有“过期的 COOKIE”差错原因的差错数据块时，端点应当首先检查该偶联是否处于建立状态，即偶联是否处于 COOKIE-ECHOED 状态。如果偶联不是处于该状态，则节点将直接丢弃该差错数据块，并不产生任何指示。

如果偶联是处于 COOKIE-ECHOED 状态，则端点可以选择以下三种方式进行处理。

1)向远端点发送一个新的 INIT 数据块，使远端点生成一个新的状态 COOKIE，尝试重新建立过程。

2)丢弃 TCB，并且向高层报告不能建立偶联。

3)向远端点发送一个新的 INIT 数据块，并且通过增加 Cookie Preservative 参数请求对状态 COOKIE 的存活时间进行延长，在计算延长时间时，实施应当根据先前交换的 COOKIE ECHO/ERROR 得到的 RTT 信息测量，且增加结果不能大于测量的 RTT+1s，这是因为过长的状态 Cookie 的存活时间会导致节点遭到重播攻击。

8.1.3　其他偶联启动的问题

8.1.3.1　选择标签值

启动标签的取值范围可以从 1～(232-1)，启动标签值的随机选择对于帮助防止 man in the middle 和顺序号码攻击十分必要。RFC 1750 中描述的这种方法可以用于启动标签的随机选择,仔细地选择启动标签可以防止把从先前的偶联发来的过期的重复分组错误地当成当前偶联的分组进行处理。

对于一个给定偶联，两个端点间使用的验证标签值在偶联的存活期间不需要改变。只要端点清除了当前偶联，再重新建立到对端的偶联时，则必须重新使用一个验证标签值。

8.2　数据的传递程序

数据的传送只有在 ESTABLISHED、SHUTDOWN-PENDING 和 SHUTDOWN-RECEIVED 这三个状态下才会出现,惟一的例外就是在 COOKIE-WAIT 状态下,DATA 数据块允许同一个出局的 COOKIE ECHO 数据块捆绑在一起发送。

根据以下原则，DATA 数据块只能在 ESTABLISHED、SHUTDOWN-PENDING 和 SHUTDOWN-SENT 状态下被接收，在 CLOSED 状态下突然收到的 DATA 数据块则应按照 8.4 节的内容进行处理。其他状态下收到的 DATA 数据块都应当被丢弃。

在 ESTABLISHED、SHUTDOWN-PENDING 和 SHUTDOWN-RECEIVED 状态下必须要处理 SACK 数据块，一个入局方向上的 SACK 数据块可以在 COOKIE-ECHOED 状态下处理，在 CLOSED 状态下突然收到的 SACK 数据块则应按照 8.4 节的内容进行处理。其他状态下收到的 SACK 数据块都应当被丢弃。

SCTP 的接收方必须能够接收长度最小为 1500 字节的 SCTP 分组，这意味着 SCTP 不必在 INIT 或 INITACK 数据块的初始 a_rwnd 中指示小于 1500 字节的情况。

为了保证传送的有效性，SCTP 定义了对较小的用户消息进行捆绑的机制，对于较大的用户消息进行分段。图 38 中描绘了用户消息通过 SCTP 的流程。

本节中，数据的发送方是指发送 DATA 数据块的端点，而数据的接收方则是指接收 DATA 数据块的端点，数据的接收方将会发送 SACK 数据块。

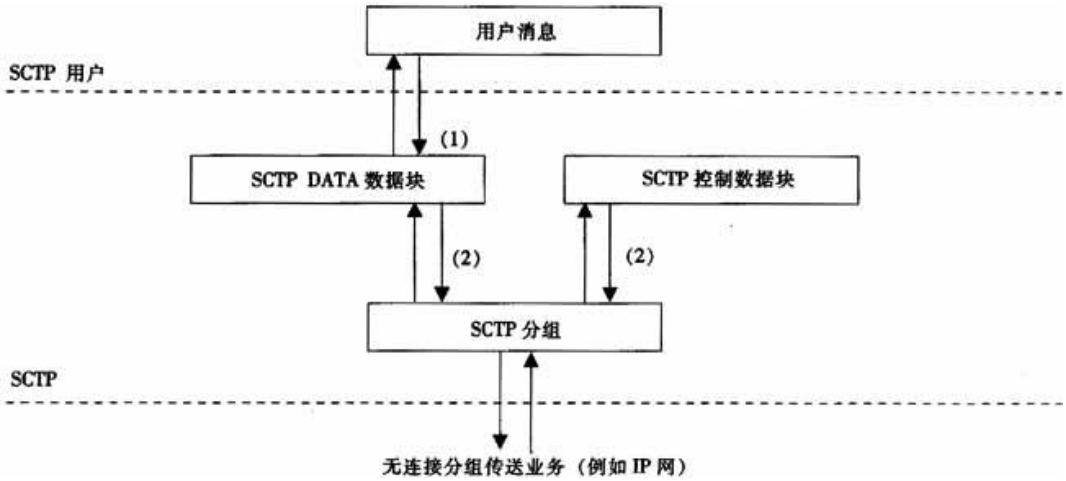

图 38　用户数据传送的示意图

1)当把用户消息转换成 DATA 数据块时，端点应当对长度大于当前偶联通路允许的最大 MTU 的用户消息进行分段，并把他们放在多个 DATA 数据块中，数据的接收方在把在多个数据块中的数据重装成用户消息并递交给用户(详细内容见 8.2.9 节)

2)多个 DATA 数据块和控制数据块可以由发送方进行捆绑，在一个 SCTP 分组中传送。这个 SCTP 分组的最太长度不应大于当前通路的 MTU，接收方在把捆绑的数据块从 SCTP 中恢复成原始的数据块。在 SCTP 分组中控制数据块应当在 DATA 数据块之前。

这种分段和捆绑的机制将在 8.2.9 和 8.2.10 节中予以说明，这种机制对于发送方来说是任选的,但它必须应能由接收方执行,即一个端点必须能够正确处理捆绑的或分段的数据。

8.2.1　DATA 数据块的传送

本标准的规定是假定对每一个目的地传送地址而言只有一个重发定时器,但在具体的实施中可以对每个 DATA 数据块都使用一个重发定时器。

对于数据的发送者而言，在发送或重发 DATA 数据块时必须遵循以下基本原则。

1)在任何时候，当对端的 rwnd 指示对端缓冲器没有空间时，数据的发送方都不能再发送任何新数据(即 rwnd=0，见 8.2.2.1)。如果 cwnd 允许，不管 rwnd 的值等于多少(包括为 0)，数据的发送方总是可以有一个 DATA 数据块正在传送给接收方(参见原则 2)。这个原则可以允许发送方探询 rwnd 的变化,以避免由于数据接收方给发送方的 SACK 在传送过程中丢失而得不到 rwnd 变化的情况。

2)在任何时候，如果到某个传送地址上等待证实的数据(字节数)大于或等于 cwnd，则发送方不能再向此地址发送新的数据。如果等待证实的数据的字节数小于 cwnd，则在新一次的发送中，发送方使等待证实的数据比 CWND 多(PMTU-1)字节。

3)当允许发送方传送数据时，在发送新的 DATA 数据块前，发送方必须要先发送那些未证实的 DATA 数据块，这个过程也可以看作是重发(由当前的 cwnd 来限制)。

4)然后发送方可以根据原则 1 和原则 2 允许的数量来发送新的 DATA 数据块。

多个数据块在一起传送可以捆绑在一个单独的分组中，此外被重发的 DATA 数据块也可以同新的数据块捆绑在一起传送，其捆绑后的长度不应大于通路允许的 MTU。高层协议可以要求不进行捆绑，但只是降低了时延，而将不能充分利用捆绑的高效率，它也不会在发生拥塞和重发时停止捆绑。

在一个端点发送 DATA 数据块前，如果还有收到的 DATA 数据块未被证实(例如：由于延迟证实)，发送方应当创建一个 SACK 数据块并把它同要发送的 DATA 数据块捆绑在一个 SCTP 分组中发送，当然应当保证最后的 SCTP 分组的长度不大于通路当前的 MTU。

注：当窗口满的时候(即根据原则 1 或 2 不允许传送时)，发送方仍可以从高层接收发送

请求，但在收到一些或全部未完成的DATA数据块的确认前或根据原则A或B允许传送前，不能传送任何DATA数据块。

当需要向其他地址发送或重发时，如果该地址的T3-rtx的定时器没有在运行，则发送方必须重新启动该定时器；如果该定时器已经在运行，则当先前发送的未证实的DATA数据块(即最低的TSN)被重新发往该地址时，发送方应重新启动这个定时器。否则数据的发送方不需要重新启动该定时器。

如果已经启动或重新启动了 T3-rtx 定时器，则定时器的取值必须依照 8.2.3.2 和8.2.3.3节定义的定时器原则进行调整。

注：数据发送方不应使用TSN＞(2^{31}-1)作为当前发送窗口的开始TSN。

8.2.2 对接收的DATA数据块的证实

SCTP端点必须要证实接收到的每个有效的DATA数据块。

此处采用了RFC 2581 4.2节定义的延时证实算法，要明确的是至少应对收到两个分组进行证实(不是每两个DATA数据块)，并且应当在收到未证实的DATA数据块200ms之内产生这个证实。在某些情况下，SCTP的证实方法比本文件规定的算法更保守一些会更好。但SCTP在发送证实时绝不能比本文件算法规定的频次更高。

SCTP接收方不必针对每个入局方向的分组产生多于一个的SACK，除非是要更新提供的窗口，作为接收应用需要接收新数据。

注：证实产生允许的最大的时延可以由SCTP的管理者来配置，可以是静态的，也可以是动态的，这主要是为了满足承载协议的特定定时要求。

一个实施不允许把最大时延配置成大于500ms，换句话说，一个实施可以选择比500ms低的值，但不能大于500ms。

除了对高层请求的关闭情况，可以由端点在发送SHUTDOWN数据块时发送证实，其他情况下，证实必须在SACK数据块中发送。一个SACK数据块可以证实接收到的多个DATA数据块，参见6.2.4节对SACK数据块格式的定义。特别是SCTP端点必须设置累积的TSN ACK字段以指示收到的最后连序的(有效数据块的)TSN。任何收到的DATA数据块的TSN大于累积的TSN证实值的情况都应在Gap Ack块字段中报告。

注：SHUTDOWN数据块不包含Gap Ack块字段，所以端点必须使用SACK数据块来证实收到失序的DATA数据块。

当一个分组中有两个重复的DATA数据块，并且又没有其他新的DATA数据块，则端点必须立即返回一个SACK而无需等待。如果分组中包含了重复的DATA数据块与新的数据块，则端点可以立即返回一个SACK。通常来讲，收到重复的DATA数据块只发生在当最初的SACK数据块丢失，且对端RTO超时的情况。重复的TSN号码应当在SACK中作为重复进行报告。

当端点收到一个SACK，它可以使用重复的TSN信息以确认是否有SACK丢失，关于这个数据块的其他应用有待进一步研究。

数据的接收方负责维护其接收缓冲器，数据接收方应当及时通知数据的发送方关于接收数据的能力变化，至于如何由实施管理接收缓冲器则取决于多方面的因素(例如操作系统、内存管理系统和内存的数量等)。

但是8.3.3.1节定义的数据发送方的策略是假定接收方按照如下方式操作的。

1)在偶联的启动阶段，端点告诉对端它已经为这个偶联的INIT或INIT ACK分配了接收缓冲器的容量，端点将根据该值设置a_rwnd。

2)对于已经收到并在缓冲区中保存的DATA数据块，应当根据已收到缓存的字节数降低a_rwnd，这样就可以有效地关闭数据发送方的rwnd，并限制可以传送的数据量。

3)当DATA数据块递交给高层后，并且已经从接收缓冲中释放，则可以根据递交给高层的字节数来增加a_rwnd，这样可以有效地放开数据发送方的rwnd，并且允许发送更多的数

据。数据接收方只有在释放了接收缓冲中这些字节后才可以增加 a_rwnd。例如，如果接收方在重装队列中正保存着一个 DATA 数据块的分段，则不应增加入 a_rwnd。

4)在发送 SACK 时，数据的接收方应当把当前的 a_rwnd 值填到 a_rwnd 字段中。数据接收方应当认为数据发送方不会重发那些已经被累积的 TSN Ack 证实的 DATA 数据块(即发送方已经从重发队列中清除)。

在一些特定的情况中，数据的接收方会需要清除已经收到的 DATA 数据块，而这些数据块还未被接收缓冲器释放(被递交到高层)，这些 DATA 数据块可以在 Gap Ack 块中进行证实。例如，在重装来自对端的用户消息时，数据接收方可以在接收缓冲中保持这些数据，而此时接收方可能又用光了接收缓冲区，此时就可以把这些用 Gap Ack 块证实的 DATA 数据块清除掉。如果数据接收方清除了这些数据块，则在随后的 SACK 中就不必包含这些 Gap Ack 块，除非是又收到了这些重发的数据块。此外端点应当在计算 a_rwnd 时考虑这些清除的数据。

一个端点不应认为 SACK 无效，并丢弃数据。只有在极特殊的情况下一个端点才可以使用这个程序(缓冲器空间耗尽)。数据接收方应当考虑丢掉那些已经在 Gap Ack 块中证实的数据可能导致数据发送方的不理想的重发策略，从而降低性能。

如果一个端点接收到的 DATA 数据块中没有任何用户数据(即长度字段设置为 16)，则必须要返回一个差错原因为“无用户数据”的 ABORT 数据块。同时也应要求端点不发送没有用户数据部分的 DATA 数据块。

8.2.2.1　对收到的 SACK 的处理

端点收到的每个 SACK 中都应包含 a_rwnd 的值，这个值表示数据接收方在发送 SACK 的时候缓冲区空间的数量，即从总的接收缓冲区容量中剩余的部分(总量在 INIT/INIT ACK 中规定)。使用 a_rwnd、累积的 TSN Ack 和 Gap Ack 块，数据发送方可以得到对端接收缓冲空间的使用情况。

数据发送方的一个问题就是在处理 SACK 时，必须要考虑接收的 SACK 可能失序的情况。这样数据接收方发送的 SACK 可能先于一个更早发送的 SACK 被数据发送方收到。如果收到的 SACK 失序，则数据发送方可能会得到对端接收缓冲容量的错误情况。

因为没有一个可以使用的明确的标识来检测失序的 SACK，这样数据的发送方必须要使用启发方式来确定 SACK 是否是最新的。

一个端点可以使用以下原则来计算 rwnd，rwnd 的计算需要使用接收的 SACK 中的 a_rwnd 的值、累积的 TSN Ack 和 Gap Ack 块。

1)在偶联的建立阶段，端点把 rwnd 设置为在 INIT 或 INIT ACK 中规定的通告的接收方窗口信用(a_rwnd)。

2)无论何时向对端发送(或重发)了一个 DATA 数据块，端点则根据数据的大小减少对端的 rwnd 值。

3)无论何时，一个 DATA 数据块被标记为重发(由于 T3-rtx 定时器超时或者是快速重发)，则把这些数据块的大小再加到 rwnd 值中。

注：如果实施是对每个 DATA 数据块都使用了定时器，则只有那些定时器超时的 DATA 数据块被标记为重发。

4)无论何时收到了 SACK，则端点执行以下程序：

—如果收到的累积 TSN Ack 比已有的 TSN Ack 值小，则丢弃该 SACK，这是因为累积的 TSN ACK 是单调递增的，所以如果 TSN Ack 值小于已有的累积 TSN Ack，则表示 SACK 失序。

—把 rwnd 的值设置为最新收到的 a_rwnd 减去正在处理累积 TSN Ack 和 Gap Ack 块之间未证实的字节数。

—如果 SACK 丢失了一个先前已经用 Gap Ack 块证实的 TSN(例如，数据接收方拒绝了数据)，则标记相应的 DATA 数据块可以进行重发，把这个数据块标记为丢失并进行快速重发的

(如 8.3.2.4 节所述)。如果针对这个 DATA 数据块最初发送到的目的地地址，没有重发定时器在运行，则针对该目的地地址启动 T3-rtx 定时器。

8.2.3　重发定时器的管理

SCTP 端点使用重发定时器 T3-rtx 用来保证数据递交时缺少来自对端的反馈的情况。定时器的时长取值可以参见对及 TO 的规定(重发超时)。

当一个端点的对端是一个多归属的,则端点应当针对对端点不同的目的地传送地址分别计算其 RTO。

SCTP 中对 RTO 的计算和管理与 TCP 管理其重发定时器的方法相同,为了计算当前的 RTO,一个端点必须针对一个目的地传送地址保存两个状态变量：即 SRTT(平滑状态下的双向时间)和 RTTVAR(双向时间变化)。

8.2.3.1　RTO 的计算

SRTT、RTTVAR 和 RTO 的计算规则如下。

1)当已经针对一个给定的目的地传送地址进行的 RTT 测量包被发送后,则设置 RTO 为协议参数 BTO.Initial。

2)当第一个 RTT 测量 R 被得到后，则设定 SRTT 为 R,

RTTVAR=R/2，RTO=SRTT+4×RTTVAR

3)如果得到了一个新的 RTT 测量 R'，则设置

RTTVAR=(1-RTO.Beta)×RTTVAR+RTO.Beta×|SRTT-R|

SRTT=(1-RTO.Alpha)×SRTT+RTO.Alpha×R'

注：用来更新 RTTVAR 的 SRTT 值应当是在第二个算式更新 SRTT 之前的值。在计算后得到修正的 RTO=SRTT+4*RTTVAR。

4)如果数据正在传送中，则允许使用下面的规则 5，新的 RTT 测量必须要针对每个双向传输进行。此外，针对给定目的地传送地址双向时延的新的 RTT 测量不应大于一次。这里有两个原因：第一在实际中过于频繁的测量并不会带来明显的好处，第二如果测量过于频繁，则规则 3 中的 RTO.Alpha 和 RTO.Beta 需要进行调整，这样 SRTT 和 RTTVAR 也要用相同的速率进行调整(根据进行的双向时延测试次数反映的取值)，而规则 3 中用到的 RTO.Alpha 和 RTO.Beta 只是一次双向时延测试的结果。

5)Karn's 算法：RTT 测量不必使用重发的分组来进行(这样得到的响应无法反映出是对第一个实例的响应还是随后实例的响应)。

6)在计算 RTO 时，如果它小于 RTO.Min 秒，则把双向时延设置为 RTO.Min 秒，使用这个规则的原因是为了避免由于 RTO 过低而带来不必要的超时。

7)最大值则可以设置为 RTO 提供的 RTO.max 秒。目前对用于计算 RTT 测量和其他状态变量的时钟的离散度 G 尚无要求，除了在计算 RTTVAR 时，如果 RTTVAR=0，则设置 RTTVAR=G。经验总结出认为小于 100ms 的时钟离散度要比粗糙的时钟离散度要好。

8.2.3.2　重发定时器规则

使用以下规则来管理重发定时器。

1)无论何时，只要向一个地址发送了 DATA 数据块(包括重发)，如果这个地址的 T3-rtx 定时器未运行，则启动这个定时器，这个定时器将在到这个地址的 RTO 之后超时，这里使用 RTO 是为了在针对某个目的地地址的前一个 T3-rtx 定时器超后得到 RTO 的双倍值。如规则 8.2.3.3 节的 2)所述。

2)当发送到某个地址的所有未完成的数据都被证实后，则停止对该地址的 T3-rtx 定时器。

3)针对某个地址，当收到的 SACK 证实了那些最早的 DATA 数据块的 TSN，则用该地址当前的 RTO 重新启动 T3-rtx 定时器。

4)当收到的 SACK 中丢失了先前已用 Gap Ack 块证实过的 TSN，如果 T3-rtx 定时器未运行，则针对这个 DTTA 数据块最初发送去的目的地地址启动 T2—nx 定时器。

8.2.3.3　对定时器 T3-rtx 超时的处理

当到一个目的地地址的重发定时器 T3-rtx 超时后，则应采取如下动作。

1)对于定时器超时的目的地地址，按照 8.3.2.3 节规定的内容调整 ssthresh 并把 cwnd 设置为 MTU。

2)对于定时器超时的目的地地址，设置 RTO=RTO×2(定时器后退)。在上面的规则 8.2.3.1 节 7)中讨论过的最大值(RTO.max)可以用来提供这种加倍操作的上边界。

3)对于定时器超时的目的地地址，确定有多少最早(即最低的 TSN)未完成的 DATA 数据块可以放在一个单独的分组中，在重发时不会受到该目的地传送地址的 MTU 限制(这可能与定时器超时的目的地地址不同，见 8.2.4 节)。称这个值为 k，在一个分组中捆绑并重传这 k 个 DATA 数据块到该目的地端点。

4)如果根据 8.2.3.2 节 1)中的规则指示可以这样做的话，则针对启动重发的目的地地址启动重发定时器 T3-rtx。对于启动的 T3-rtx 使用的 RTO 应当是到重发发送的目的地地址，当接收方是多归属的，则这个地址可以与定时器超时的目的地地址不同(参见 8.2.4 节)。

重发之后，一旦得到一个新的 RTT 测量值，这个 RTT 测量值根据 8.2.2.1 节 5)中的原则，只可以用发送的新数据与证实之间的时间，或者是用 HEARTBEAT 消息的测量值进行计算(参见 8.4.3 节)结果。再根据 8.2.3.1 节 2)的步骤进行计算，当然也包括对 RTO 的计算，这个计算结果会使 RTO 返回到加倍之前的值。

注：任何发送到 T3-rtx 定时器超时的目的地地址 DATA 数据块不能满足 MTU 的限制时(上面的规则 3)，则这个 DATA 数据块应标记为重传，并且在 cwnd 允许的情况下立即发送(通常是收到 SACK 后)。

管理重发定时器的最后一个原则与故障结束方式有关。

当一个节点从当前的目的地传送地址切换到另外一个地址时，当前的重发定时器继续运行。如果按照 8.3.3.2 节 1)的规则，允许端点发送一个包含 DATA 数据块的分组到一个新的传送地址，则针对该传送地址也启动定时器，并使用数据要发送到的目的地地址的 RTO。

8.2.4　多归属的 SCTP 端点

如果可以使用多个目的地传送地址作为到一个端点的目的地地址，则这个 SCTP 端点可以被看作是多归属的。

更进一步，端点的高层协议(ULP)应当可以在多个目的地地址中选择一个地址作为到这个多归属 SCTP 点的首选通路(可参见第 5 章和 8.1.1.2 节)。

缺省的情况是，一个端点总是在首选通路上发送数据，除非是 SCTP 用户明确规定了使用的目的地传送地址(也可能是起源传送地址)。

端点应能根据收到的 DATA 或控制数据块中的起源传送地址，并把相同地址的作为发送响应数据块的目的地传送地址(如：SACK、HEARTbEAT ACK 等)。这个规则在端点把 DATA 数据块和响应数据块捆绑传送时应遵守，即此时的目的地传送地址应当是响应数据块应要对应的目的地地址。

但是当要用一个 SACK 证实来自不同起源地址的 DATA 数据块时，SACK 数据块可以使用这些需要被证实的 DATA 或控制数据块发来的起源地址中的一个地址作为目的地传送地址。

当接收方收到了重复的 DATA 数据块，应尽可能不使用重复的 DATA 数据块中的起源地址作为目的地地址，最好选择另外的目的地传送地址向这个多归属的端点发送 SACK 数据块。这样做的原因是在收到从多归属端点发来的重复数据块后，可以指示 SACK 的返回通路已经断开了(根据 DATA 数据块中的起源地址)。

此外，当对端点是一个多归属的端点时，本端在重发时，应当尝试把 DATA 发到另外的

激活的目的地传送地址，该地址应与 DATA 数据块最后一次发送的目的地地址不同。

重发不会影响对未证实的数据的总数统计，但是如果 DATA 数据块重发到不同的目的地地址，对新目的地地址和旧目的地地址(上次数据块发送到的地址)的未证实数据的统计应当同时调整。

8.2.4.1　对未激活的目的地地址的 FAILOVER

一个多归属的 SCTP 端点可能由于出现一些差错或者由于 SCTP 用户的调整，使该端点的一些传送地址处于未激活状态(见 8.4.2)。

当有出局数据要发送，而首选通路又变成未激活时(例如：故障原因)，或者是由 SCTP 用户明确地请求向一个未激活的目的地传送地址发送数据时，在向高层协议报告差错前，如果有其他地址存在，SCTP 端点应当尝试向该目的地的其他激活的目的地传送地址发送数据。

重发数据时，如果该端点是一个多归属的 SCTP 端点，其重发选择策略应当考虑每个起源和目的地地址对。重发的端点应当选择一个与最初的起源和目的地地址对最不同的起源和目的地地址发送该分组。

注：对于选择重发到不同的起源和目的地地址对取决于具体的实施，在本规范中不规定。

8.2.5　流标识符和流顺序号

每个 DATA 数据块必须携带一个有效的流标识符，如果端点收到的 DATA 数据块中包含有无效的流标识符，则应按照正常程序对收到的 DATA 数据块进行确认，并立即发送一个错误原因为“无效的流标识符”的 ERROR 数据块，同时丢弃这个 DATA 数据块。端点可以把 ERROR 数据块同 SACK 捆绑在同一个分组中，但 ERROR 应当在 SACK 数据块之后出现。

在偶联建立时，所有流中的流顺序号码都应当从 0 开始。当流顺序号码到达 65535 后，则接下来的流顺序号码为 0。

8.2.6　有序的和无序的递交

在一个流中，一个端点必须把 U 标志位设置为 0 的 DATA 数据块按照他们的流顺序号码递交给高层，如果到达的 DATA 数据块的顺序号码失序，则端点必须保存这些数据块，直到他们被重新排序后再递交给高层协议。

但是，一个 SCTP 端点可以通过把 DATA 数据块的 U 标志位设置为 1 来指示对于一个特定传送的 DATA 数据块可以采用无序递交的方式。

当一个端点收到的 DATA 数据块的 U 标志位设置为 1，则该端点就不使用排序机制，并立即把这些数据递交给高层(如果由数据发送方对用户数据进行了分段，则应在重装后再递交)。这样就提供了一种用来在给定流中传送“带外”数据的方法。同样的，也把在一个流上发送的所有 DATA 数据块的 U 标志位设置为 1，使这个流作为“无序”的流。

注：当发送无顺序的 DATA 数据块时，实施方可以选择把 DATA 数据块放在发送的分组中，如果可能的话并把它放在出局发送队列的队首先发送。

U 标志位设置为 1 的 DATA 数据块的流顺序号码字段是没有意义的，发送方可以填任意值，但接收方必须忽略这个字段。

注：在传送数据时，端点在发送 U 标志位设置为 1 的 DATA 数据块时不需要增加流顺序号码。

8.2.7　报告收到的 DATA 数据块的 TSN 间隔

在收到新的 DATA 数据块，端点应当首先检查收到的 TSN 的连续性。如果端点检查出了在收到的 DATA 数据块顺序中出现间隔，则应立即发送带有 Gap Ack 块的 SACK 数据。在对收到的 SCTP 分组判别后认定不能填充 TSN 间隔的时候，数据接收方应继续发送 SACK 数据块。

根据收到的 SACK 中包含的 Gap Ack 块，端点可以计算出丢失的 DATA 数据块并决定是否重发这些 DATA 数据块(参见 8.3.2.4 节)。

在一个 SACK 中可以报告多个 TSN 的间隔。当对端点是一个多归属的 SCTP 端点时，SCTP

端点应当总是向相同的目的地地址发送SACK,用来证实从该地址收到的最后的DATA数据块。

在收到SACK后,端点必须从发送队列中清除所有由SACK中累积的TSN Ack证实的DATA数据块。端点必须把那些TSN未包含SACK的在Gap Ack块中的DATA数据块当作丢失的数据块来处理。为了决定重发，对每个报告为丢失的未证实的DATA数据块的号码都应由数据的发送方来记录，详细内容参见8.3.3节。

可以在SACK数据块中报告的Gap Ack块的最大数量是受通路的MTU限制的。如果由于MTU的限制，一个SACK不能完全包含所有需要报告的Gap Ack块，则端点只发送一个SACK，报告的Gap Ack块的顺序应当是从最低的TSN到最高的TSN进行排列，不能包含的最高的TSN号码可以暂不证实。

8.2.8 CRC-32校验码的计算

在发送SCTP分组的时候，一个端点必须能够保证传送的数据的完整性，这就需要对这个分组进行校验码计算。CRC-32校验码的计算如下所述。

在构造分组的时候(包含SCTP的头和一个或多个控制或DATA数据块),发送方应当作以下计算。

1)正确设置SCTP公共分组头中的验证标签并把校验码字段初始化为全0;

2)对整个分组计算CRC-32校验码，包括SCTP公共头和所有的数据块;

3)把得到的结果值填到SCTP公共分组头的校验码字段，其他比特保持不变。

当收到一个SCTP分组后，接收方必须先检查校验码。

1)先保存收到的CRC-32校验码;

2)然后把收到的SCTP分组的校验码字段填充为全0，再对整个分组进行CRC-32校验码的计算;

3)验证计算出的CRC-32校验码与收到的CRC-32校验码是否相同，如果不同，则接收方必须要把这个分组当成一个无效的SCTP分组来处理。通常对无效的SCTP分组的处理就是直接把该分组丢弃，并不产生任何指示。

8.2.9 分段和重装

端点可以不支持对发送 DATA 数据块进行分段，但必须要支持对收到的已分段的 DATA 数据块的重装功能。如果一个端点支持分段，则当需要发送的用户消息使生成的SCTP分组长度大于MTU时，应对用户消息进行分段。如果在实施中不支持对出局的用户消息的分段，则端点必须要向高层返回一个错误指示，同时也不必再尝试发送该消息。

注：对于这种错误情况，发送原语应当需要向高层返回一个差错指示。

如果对端点是一个多归属的端点，则发送端点选择发送的消息长度应当小于偶联通路MTU长度。偶联通路的MTU是指到所有目的地地址中最小的通路MTU。

注：一旦一个消息被分段后，就不能再进行分段了，如果PMTU被再次降低，则必须使用IP分段机制，对于PMTU的检出可以参见8.3.3节。

SCTP的实施在确定分段时必须要考虑SCTP的分组头、DATA数据块的头；如果SACK数据块同DATA数据块捆绑在一起的话，实施还必须要考虑SACK数据块所要求的空间。

分段过程采用如下步骤。

1)数据的发送方必须要把用户消息分割成几个DATA数据块，使得每个数据块加上SCTP的开销后配相同的流顺序号码(SSN)。如果用户指示用户消息可以按照无序的方式递交，则这个用户消息的每个DATA数据块的U标志位都应设置为1。

2)发送方也必须把这一组中的第一个DATA数据块的B/E比特设置为10，把这一组中的最后一个DATA数据块的B/E比特设置为01，其他DATA数据块的B/E比特设置为00。

一个端点必须能通过检查每个收到的DATA数据块的B/E比特,来识别分段的DATA数据块，并且把分段的DATA数据块排队后重装。一旦用户消息被重装好，则SCTP应当把这个重

装的用户消息放在特定的流中重新排序并最终递交到高层。

注：如果数据的接收方缓冲已经耗尽，面此时还在等待一个需要重装的用户消息的其他分段，就可以使用部分递交 API 把这个不完整的人局消息递交上去，释放部分接收缓冲，使之可以继续接收消息的其他部分。

8.2.10　捆绑机制

端点可以通过捆绑机制把多个数据块放在一个出局的 SCTP 分组中来传送。最终生成的 IP 分组的长度(包括 SCTP 分组和 IP 分组头)必须要小于或等于当前通路的 MTU。

如果它的对端点是一个多归属的端点，则发送端点应当选择当前首选通路中最小 MTU 作为分组的最太长度。

当把控制数据块和 DATA 数据块进行捆绑的时候，端点必须把控制数据块放在 SCTP 分组的最前面。发送方在 SCTP 分组重传送 DATA 数据块时，其 TSN 必须是按照升序排列。

注：由于控制数据块必须要放在分组的最前面，而 DATA 数据块必须在 SHUTDOWN 或 SHUTDOWN ACK 之前传送，因此 DATA 数据块不能与 SHUTDOWN 或 SHUTDOWN ACK 捆绑在一起传送。

部分的数据块不能放在一个 SCTP 分组中传送，换句话说，SCTP 中的数据块都应当是完整的。

一个端点必须按照数据块在分组中的顺序来对这些数据块进行处理处理。接收方使用数据块长度字段来确定数据块的开始和结束，并且认为所有的数据块的长度都应是 4 个八位位组的整数倍。如果接收方检测到了部分的数据块(不完整的)，则必须要丢弃该数据块。

一个端点不能把 INIT、INIT ACK 或 SHUTCOMPLETE 同其他数据块捆绑起来。

8.3　拥塞控制程序

拥塞控制是 SCTP 的一个基本功能。对于一些应用，可以为 SCTP 业务量分配足够的资源，以确保能准确地递交那些对时间要求较高的数据。这样在正常情况下，传送就不大可能会经历严重的拥塞状态。但 SCTP 必须能在不正常的情况下工作，这些不正常的情况可以是部分网络故障或者是异常的业务量情况。为了能使数据尽可能地递交到高层，在这种情况下，SCTP 必须要采用拥塞控制机制使得其从拥塞状态下迅速恢复。

在网络没有发生拥塞的时候这些预防性的拥塞控制算法应当不会对网络的协议性能产生任何影响。

注：在满足规定的性能要求的前提下，一个 SCTP 实施总允许采用多种拥塞控制算法，而不单是下面描述的算法。

SCTP 使用的拥塞控制算法是基于 RFC2581 的，本节描述了 RFC2581 定义的算法是如何在 SCTP 中使用的。SCTP 的拥塞控制总是针对整个偶联进行的，而不能针对偶联中的某个流来进行。

8.3.1　SCTP 与 TCP 拥塞控制的区别

在 SCTP 的 SACK 中包含的 Gap Ack 块与 TCP 的 SACK 有着相同的含义。TCP 认为 SACK 中的信息只是一个建议性的信息，SCTP 也认为 SACK 中的 Gap Ack 块包含的信息是建议性的信息。在 SCTP 中任何由 SACK 证实的 DATA 数据块，包括那些在接收方失序的 DATA 数据块，只有在通过累积的 TSN ACK 确认后才认为是把这些 TSN 的 DATA 数据块递交给高层(即用 SACK 中的累积的 TSN ACK 证实的 DATA 数据块)，因此用 cwnd 的值来控制未证实的数据(在无 SACCK 的 TCP 中)要比用最高确认的顺序号码和在拥塞窗口中可以发送的最后一个 DATA 数据块之间上边界要好。SCTP 使用了与无 SACK TCP 不同的快速重发和快速恢复。

SCTP 与 TCP 的最大区别就是一个多归属功能，SCTP 被设计在两个端点间建立一个可靠的通信偶联，每个端点可以使用多个传送地址。两个端点间不同的地址可以产生不同的通路，这样比较理想的拥塞算法就是需要针对不同的通路使用不同的拥塞控制参数集。对于 SCTP

来说，在拥塞控制中对于多归属接收方的处理就是一个新情况，并且需要在今后进行改进。目前的算法使用以下假设。

—除非高层有明确的指示，否则发送方总是使用相同的目的地，但是 SCTP 可以在一个地址被标记为未激活时选择另外一个目的地地址(见 8.2 节)。同样的在重发的过程中也可以使用与最初发送不同的传送地址；

—发送方针对每个可以发送去的目的地地址使用分别的拥塞控制参数集(不是针对每个起源和目的地地址对，而只是对每个目的地地址)，如果到一个地址很长时间不使用，则这些个参数应当予以丢弃(失效)。

对每个目的地地址，一个端点根据第一次的传送地址进行慢启动。

注：在一个单独的 TCP 进程中，TCP 保证把数据按照顺序递交给高层。这意味着当 TCP 发现了在接收购顺序号码中有间隔，则它一直要等待这个间隔被填好后才把顺序号码高于丢失的顺序号码的数据递交给高层。另一方面，尽管在 TSN 中还有间隔，但只要对于特定的流其流顺序号码是连续的(即丢失的 DATA 数据块属于不同的流)，或者是指示采用无序方式，SCTP 则可以把这些数据递交到高层协议。尽管这不会影响 cwnd，但却会影响 rwnd 的计算。

8.3.2 SCTP 的慢启动和避免拥塞

端点必须要使用慢启动和避免拥塞算法以控制进入到网络中的数据数量。SCTP 的拥塞控制是关于偶联进行的，而不是针对一个流进行的。在某些情况下，对于 SCTP 发送方使用更保守的算法要比其他允许的算法更有益。但 SCTP 发送方不必选择比以下算法更严格的算法。

与 TCP 相似，SCTP 端点使用以下三个控制变量来规定其传送速率。

—接收方广播的窗口大小(rwnd，单位：八位位组数)：由接收方根据入局分组的缓冲容量来设置(这个变量是对整个偶联而言的)。

—拥塞控制窗口(cwnd，单位：八位位组数)：这个变量根据发送方观察的网络情况来调整(到每个目的地地址都应维持这样一个变量)。

—慢启动门限(ssthresh，单位：八位位组数)：发送方用这个变量判别是否处于慢启动阶段还是避免拥塞阶段(到每个目的地地址都应维持这样一个变量)。

SCTP 还需要一个附加的控制变量 partial_bytes_acked，这个变量用来在避免拥塞阶段帮助调整 cwnd。与 TCP 不同，SCTP 发送方必须要针对每个端点的目的地地址都维护这组变量(此时这个对端点是一个多归属的点)。

8.3.2.1 慢启动(Slow-Start)

当在一个状态不明的网络或者是空闲了相当长时间的网络上开始传送数据时，要求 SCTP 向网络探询一下以确定其可用的容量。出于这个目的，在开始传送数据时或者是对重发定时器检出的丢失消息的重发时，就使用慢启动算法。

—在数据传送前或者是空闲了足够长的时间后，初始的 cwnd 应当设置为小于或等于 2×MTU。

—在重发定时器超时后，初始的 cwnd 应当不大于 1×MTU。

—ssthresh 的初始制可以是任意的大(例如：实施的时候可以使用接收方广播窗口的大小)。

—只要 cwnd>0，就允许则端点到某个传送地址有 cwnd 字节长的未完成数据。

—当 cwnd≤ssthresh，SCTP 端点就要用慢启动算法来增加 cwnd(假定当前的拥塞窗口已经完全被使用了)。如果一个入局 SACK 提高了累积 TSN Ack 的值，则 cwnd 最多就只能按照 1)和 2)中最小值来增加。

1)先前未证实 DATA 数据块已被证实的总数；

2)到目的地通路的 MTU。

这主要是用来保护在[SAVAGE99]中规定的 ACK-Splitting 的攻击。

当对端点是一个多归属的端点，如果一个端点收到的 SACK 增加了累积 TSN Ack 的值，则该端点应当增加与发送证实数据的目的地地址对应的 cwnd。但如果接收的 SACK 不能提高累积 TSN Ack 的值，则端点不必针对任何目的地地址调整 cwnd。

由于端点的 cwnd 并没有与累积 TSN Ack 相关联，如果收到了两个重复的 SACK，尽管他们可以不改变累积 TSN Ack，但端点仍可以使用他们来记录新数据发送的时间。这样由 SACK 证实的数据降低了传送中的数据数量，并使之小于 cwnd。由此造成 cwnd 的值没有变化，又允许发送新的数据。另一方面，cwnd 的增加必须要跟累积 TSN Ack 的增加相联系。否则在可能拥塞的时间里，重复的 SACK 不仅能记录新数据的时间，而且也能反过来记录比网络留下的更多的新数据。

当端点不向一个给定购传送地址传送数据数据时,则该传送地址的 cwnd 应当在每个 RTO 时间内按照 max(cwnd/2，2xMTU)进行调整。

8.3.2.2　避免拥塞

当 cwnd 大于 ssthresh，如果发送方有与该传送地址对应的 cwnd 或更多未完成的数据，则 cwnd 应当在每个 RTT 中增加 1×MTU。

在实际实施中可以用以下方式获得这个目标。

—partial_bytes_acked 初始为 0。

—当 cwnd 大于 ssthresh，则收到 SACK 提高了累积 TSN Ack 的值，把 partial_bytes_acked 增加上所有用这个 SACK 的累积 TSN Ack 和 Gap Ack 块证实的所有新数据块的八位位组总数。

—当 partial_bytes_acked 等于或大于 cwnd，且在 SACK 到达之前，发送方有 cwnd 或更多未证实的数据(即：表明在 SACK 到达之前，在途中传送的字节数大于或等于 cwnd)，则用 MTU 来增加 cwnd，并把 partial_bytes_acked 设置为 partial_bytes_acke 的当前值减 cwnd。

—与在慢启动中相同，当发送方不向一个给定的传送地址发送数据的时候，该传送地址的 cwnd 应当在每个 RTO 时间内按照 max(cwnd/2 2×MTU)进行调整。

—当所有发送方传送的数据都被接收者证实后，partial_bytes_acked 又被重新初始化为 0。

8.3.2.3　拥塞控制

当根据收到的 SACK 检测到有分组丢失，端点应采取以下动作：

ssthresh=max(cwnd/2，2×MTU)；

cwnd=ssthresh。

基本上，分组的丢失将导致 cwnd 降低一半。

当到某个地址的 T3-rtx 超时，SCTP 将用如下动作进行慢启动。

ssthresh=max(cwnd/2, 2×MTU)；

cwnd=1×MTU。

并应确保在端点收到了数据成功递交给该地址的确认前，不会有多于一个的 SCTP 分组在向该地址传送。

8.3.2.4　根据 GAP 报告的快速重发

在没有数据丢失的时候，端点采用延迟确认方式，但当端点发现到达的 TSN 顺序中出现间隔时，在这个间隔被补上之前，它每收到一个携带数据的分组就后向发送一个 SACK。

当一个端点收到的 SACK 中指示一些 TSN 丢失，在开始采取快速重发动作前，它应当等待至少 3 次或更多次的对相同 TSN 的指示(通过后续的 SACK)。

当从连续第四个 SACK 中收到了相同的指示，这个 TSN 就当作丢失来报告，发送方应采

取以下动作。

1)标记丢失的 DATA 数据块为重发；

2)按照 8.3.2.3 节的公式，针对最后一次发送的丢失的 DATA 数据块的目的地地址调整 ssthresh 和 cwnd。

3)确定有多少个最早的(即：最低的 TSN)标记为重发 DATA 数据块可以放在一个分组中，而不会受到到该分组需要发送到的目的地地址的通路 MTU 的限制。这个值称为足，在一个分组中重传这尺个 DATA 数据块。

4)当且仅当最后的 SACK 确认的最低的未完成的 TSN 的 DATA 数据块发送给该地址后，或者是端点向该地址重发了第一个未证实的 DATA 数据块后，重新启动 T3-rtx 定时器。

注：在进行上述调整之前，如果接收到的 SACK 也确认了新的 DATA 数据块并且提高了累积 TSN ACK 的值，则应按照 8.3.2.1 和 8.3.2.2 节定义的原则先调整 cwnd。

上述描述的一个最直接的实施就是针对每个由 SACK 报告的 TSN 间隔设一个计数器，这个计数器针对每个报告的 TSN 间隔来加 1，在达到 4 之后，则开始重发过程，并把该计数器清 0。

因为 SCTP 中的 cwnd 间接地限制了未完成的 TSN 的数量，TCP 的快速恢复的影响是通过不调整拥塞控制窗口的大小来自动获得的。

8.3.3　发现通路 MTU

RFC 1191 规定了“发现通路 MTU”，端点如何对给定的 Internet 通路的最大传送单元保持评估，来避免在该通路上发送的分组超过 MTU，并可以定期尝试检测通路 MTU 的变化。RFC 1191 全面的讨论了 MTU 的发现机制和策略用来确定当前端到端 MTU 值的设置以及该值变化的检出。RFC1981 对 IPv6 规定了相同的机制，使用 IPv6 的 SCTP 发送方必须使用发现通路 MTD，除非是所有的分组长度都小于 IPv6 的最小 MTU。

端点应当果用这些技术，并且针对每个目的地地址进行这些操作。

SCTP 使用的发现通路 MTU 的方法与 RFC1191 描述的方法有 4 点不同。

1)SCTP 偶联可以包含多个地址，一个端点必须针对远端的不同目的地地址分别进行 MTU 的评估。

2)在文件中其他地方谈到的 MTU 是指与上下文中涉及的目的地地址相关的 MTU。

3)与 TCP 不一样，SCTP 没有“最大分段长度”的概念，因此对每个目的地地址的 MTU 在初始化时不应当大于分组包选路到该目的地地址的本地接口的链路的 MTU。

4)由于 SCTP 传送数据是根据 TSN 来传送的，而不是根据字节数(TCP 的情况)，所以 RFC 1191 第 6.5 节讨论的方法可以采用，当要向远端地址重发一个 IP 数据报，且 IP 数据报的长度又大于到该地址的通路 MTU 时，则 IP 数据报在重发时不用设置 DF 比特，并允许其进行分段，而发送一个新的 IP 数据报，则必须设置 DF 比特。

发送方应当遵循偶联的 PMTU，偶联的 PMTU 是指到对端的所有目的地地址中所发现的最小的 PMTU。当把消息分割成多个部分时，这个偶联的 PMTU 应当用来计算每个分段的长度，这样就允许重发时可以把分组发送到另外的地址而不会遇到 IP 分段。

除了这些区别外，RFC 1191 和 RFC 1981 中讨论的 TCP 使用的发现 MTU 方法均适用于 SCTP 针对每个目的地地址进行计算。

注：对 IPv6 的目的地地址，则不存在 DF 比特，则 IP 数据报的分段必须采用 RFC2460 描述的方法进行分段。

8.4　故障管理程序

8.4.1　端点故障的检出

一个端点应当有一个计数器，用来对连续重发到对端点的分组总数进行统计(如果对端点是一个多归属的节点，则应包括到所有目的地传送地址的重发)。如果这个计数器的值超

过协议参数 Association.Max.Retrans 指示的限制，则本端点认为对端点不可达，并且应当停止向该端点发送任何数据(偶联应当进入到 CLOSE 状态)。此外本地端点还应向高层报告对端点的故障，并且可以报告在出局队列中未证实的用户数据(任选的)。当对端点变为不可达时，则偶联自动的关闭。

当发送到对端点的 DATA 数据块被证实后(收到 SACK)，或者是收到来自对端点的的 HEARTBEAT-ACK 消息，该计数器应当被重新复位。

8.4.2 通路故障的检出

当对端点是一个多归属的端点，则端点应当针对到对端点每个目的地传送地址使用一个差错计数器。

每当到某个地址的 T3-rtx 定时器超时后，或者是发送到一个空闲地址的 HEARTBEAT 消息在 RTO 时间内没有收到证实，则到该目的地地址的差错计数器应当加 1，当差错计数器的值超过了协议参数 Path.Max.Retrans 规定的限制后，端点应当标记该目的地传送地址为“未激活”并且应当向高层进行通知。

当未证实的 TSN 被证实或者是当发送到该地址的 HEARTBEAT 被用 HEARTBEAT ACK 证实后，则端点应当清除该目的地传送地址(数据块或 HEARTBEAT 发送去的地址)的差错计数器。

当对端点是一个多归属的点，且最后一个发送的数据块被重发到另外的地址，则会出现含混，即最后一个数据块发送的地址不知道是否被证实的情况。但这种含混不会对 SCTP 的行为造成有意义的影响。如果这种含混是不期望的，如果最后发送的数据块是一个重发的数据块，则发送方可以选择不清除差错计数器。

注：在配置 SCTP 端点时，用户应当避免让“Association.Max.Retrans”的取值大于到所有远端点目的地“Path.Max.Retrans”的总和。否则当所有的目的地址变为不激活时，而端点仍认为对端点可达，当这种情况出现时，SCTP 选择何种功能是由实施决定的。

当首选通路标记为未激活时(例如：由于大量的重发)，发送方可以自动向另外的存在且激活的目的地地址发送分组。如果当首选通路标记为未激活时，存在多于一个激活的目的地地址时，SCTP 只选择一个传送地址用来作为新的目的地传送地址。

8.4.3 通路的心跳

通常情况下，SCTP 端点通过定期地向目的地传送地址发送心跳消息来监视对端的空闲目的地传送地址的可达性。HeartBeat 的发送可以在偶联进入建立(ESTABLISHED)状态后就开始，并且在发送了 SHUTDOWN 或 SHUTDOWN-ACK 之后停止。在偶联进入 COOKIE-SENT 状态之后到偶联进入关闭状态前，收到 HeartBeat 的一方必须要 HeartBeat ACK 进行响应。

当目的地传送地址没有新的数据块能用来更新通路 RTT(通常包含在第一个传送的 DATA、INIT、COOKIE ECHO 和 HEARTBEAT 数据块中)，并且在当前心跳周期中没有 HEABTBEAT 数据块被发送到该地址，则认为这个目的地地址是空阔的。这适用于激活的或未激活的目的地传送地址。

高层协议可以有选择地启动以下功能。

1)禁止在一个绘定偶联上针对某个特定的目的地传送地址进行心跳测试。

2)改变心跳测试的周期。

3)重新允许在一个给定偶联上针对某个特定购目的地传送地址进行心跳测试。

4)按需地请求在一个给定偶联上针对某个特定的目的地传送地址进行心跳测试。

当向一个地址发送了 HEART7BEAT 数据块后，在 RTO 时间内没有收到证实，则端点应当增加到该目的地地址的差错计数器。

当这个计数器的到达了 Path.Max.Retrans 协议参数的门限时，端点应当标记相关的目的地传送地址为未激活(如果先前未标记的话)，并向高层协议报告到该目的地地址的可达性的变化(该报告取决于实施)。之后，端点应当继续对该目的地地址进行心跳测试，但应当停

止增加到该目的地地址的差错计数器。

发送方的心跳数据块中的Heartbeat信息字段应当包含分组发送的当前时间，以及分组需要发送到的目的地地址。

注：心跳机制的另外一种实施可以是每向一个目的地地址发送HEARTBEAT数据块后，就增加差错计数器的值，当HEARTBEAT ACK数据块收到后，发送方应当清除到该目的地地址的差错计数器，这种方法可以清除先前出现的差错(和其他的差错计数器)。

HEARTBEAT的接收方应当立即用HEARTBEAT ACK数据块进行响应，并且在HEARTBEAT ACK数据块的Heartbeat信息字段中复制在HEARTBEAT数据块中收到的信息。

在收到HEARTBEAT ACK数据块后，HEARTBEAT数据块的发送方应当清除该数据块指示的目的地传送地址的差错计数器，并标记目的地地址为激活(如果先前未标记的话)。由于收到了HEARTBEAT ACK数据块，端点向高层发送未激活的目的地传送地址变为激活的报告是任选购。接收到HEARTBEAT ACK的一方必须同时能清除偶联所有的差错计数器(见8.4.1)。

接收到HEARTBEAT ACK的一方应当能使用HEARTBEAT ACK数据块中携带的时间值对该目的地传送地址进行RTT测量。

对一个空闲的目的地地址可以进行心跳测试，建议HEARTBEAT数据块的发送周期为到该目的地地址的RTO加上HB.interval的参数值，并带有±50%的抖动，如果前一个HEARTBEAT的应答没有收到，则应当按照指数级来降低RTO。

目前为SCTP的用户提供了一个原语，可以用来改变到某个给定目的地地址的HB.interval的值和是否开启到该目的地地址的心跳测试。心跳测试的周期可以由SCTP用户设定的值加上到该目的地地址的RTO的值来决定。当心跳测试周期超时后则只发送一个HEARTBEAT(如果多个目的地地址的空闲)。如何选择空闲的目的地地址进行心跳测试发送HEARTBEAT取决于实施(如果多于一个目的地空闲)

注：如果改变了心跳测试的时长，则只需要考虑一端的影响。当这个值被增加后，即心跳测试周期变长，则检测到丢失的ABORT数据块也应相应延长。如果出于任何原因，对端点希望中止一个偶联，且ABORT数据块丢失，则本端点只能通过发送DATA或HEARTBEAT数据块来发现丢失了ABORT数据块(导致对端发送其他的ABORT数据块)。在改变心跳测试定时器时必须要考虑这个问题。如果心跳测试被禁止，则只有通过在该偶联上发送DATA数据块来发现丢失了从对端发来的ABORT数据块。

8.4.4 对OOTB(Out of the blue)分组的处理

当一个SCTP分组的格式是正确的，即此分组可以通过接收方CRC-32校验码的检查(见8.2.8节)，但接收方不能识别出该分组属于哪个偶联时，则认为这个分组是一个OOTB分组。

接收到OOTB分组的一方应当采取如下动作。

1)如果OOTB分组是一个来自或去往非单播地址的分组，则直接丢弃该分组而不产生任何报告。

2)如果OOTB分组包含了一个ABORT数据块，则接收方必须直接舍弃该OOTB分组而不产生任何报告，也不采取任何动作，否则进入3)。

3)如果分组中包含了一个验证标签为0的INIT数据块，则按照8.1节描述的内容进行处理。

4)如果分组中包含的第一个数据块是一个COOKIE ECHO数据块，则按照8.1节描述的内容进行处理，否则进入5)；

5)如果分组中包含了一个SHUTDOWN ACK数据块，则接收方应当向OOTB分组的发送方相应一个SHUTDOWN COMPLETE数据块。在发送SHUTDOWN COMPLETE数据块时，接收OOTB的这一方必须把发送的分组的验证标签字段设置为从SHUTDOWN ACK中收到的验证标签值，并设置数据块标志的

T 比特用来指示未发现 TCB，否则进入 6)；

6)如果分组中包含了一个 SHUTDOWN COMPLETE 数据块，则接收方应当直接舍弃该分组不产生任何报告，也不采取任何动作，否则进入 7)；

7)如果分组中包含了一个差错原因为“过期的 cookie”的 ERROR 或 COOKIE ACK，则接收方应当直接舍弃该 SCTP 分组不产生任何报告，否则进入 8)；

8)接收方应当用 ABORT 来向发送者响应收到的 OOTB 分组，发送 ABORT 数据块时，接收 OOTB 的这一方必须把发送的分组的验证标签字段设置为从 OOTB 分组中收到的验证标签值，并设置数据块标志的 T 比特用来指示未发现 TCB，在发送 ABORT 数据块后，接收 OOTB 分组的一方应当丢弃 OOTB 分组，也不采取任何动作。

8.4.5　验证标签

本节定义了在发送和接收未包含 INIT、SHUTDOWNCOMPLETE、COOKIE ECHO(见 8.1 节)或 ABORT 数据块的 SCTP 分组时适用验证标签的规则。用来发送和接收包含这些数据块类型的规则在 8.5.1 节中单独讨论。

当发送一个 SCTP 分组时，端点必须把出局分组的验证标签字段的值设置为从对端收到的 INIT 或 INIT ACK 数据块中的启动标签参数的值。

当收到一个 SCTP 分组时，端点必须要保证收到的 SCTP 分组的验证标签字段的值同其标签向匹配。如果收到的验证标签的值与接收方自己的标签的值不同，则接收方应当直接舍弃该分组而不产生任何报告，除非是 8.5.1 节列出的情况，否则也不进行进一步的处理。

8.4.5.1　验证标签规则中的异常情况

1)携带 INIT 的分组的标签规则

—发送方必须设置分组的验证标签为 0。

—当端点收到一个验证标签为 0 的 SCTP 分组，端点应当验证该分组中是否只包含一个 INIT 数据块，否则接收方应当直接舍弃该分组而不产生任何报告。

2)携带 ABORT 的分组的标签规则

—如果知道目的地端点的标签值，发送端点应当总是把要发送的分组的验证标签字段设置为目的地端点的标签值。

—如果 ABORT 是为了响应 OOTB 分组，则端点必须要执行 8.4.4 节规定的流程。

—如果验证标签与本端的标签或者是远端的标签相匹配，则接收方必须接受该分组，否则接收方应当直接舍弃该分组而不产生任何报告。

3)携带 SHUTDOWN COMPLETE 的分组的标签规则

—在发送 SHUTDOWN COMPLETE 时，如果收到 SHUTDOWN ACK 的一方存在 TCB，则应当使用目的地端点的标签，只有当 TCB 不存在时，发送方才使用从 SHUTDOWN ACK 中收到的验证标签。

—当分组的验证标签字段与本端的标签匹配或者是该标签设置为对端的标签且 T 比特也设置为数据块标志，收到 SHUTDOWN COMPLETEE 的一方应当接受该分组，否则接收方应当直接舍弃该分组不产生任何报告，也不采取任何动作。如果端点不是处于 SHUTDOWN-ACK-SENT 状态，则端点必须忽略收到的 SHUTDOWN COMPLETE 数据块。

4)携带 COOKIE ECHO 的分组的标签规则

—在发送 COOKIE ECHO 时，端点必须使用在 INIT ACK 中收到的启动标签值。

—收到 COOKIE ECHO 一方的应当执行第 8.1 节规定的程序。

5)携带 SHUTDOWN ACK 的分组的标签规则

—如果接收方是处于 COOKIE-ECHOED 或 COOKIE-WAIT 状态，则应当执行 8.4.4 节规定的程序，换句话说就是把这个分组当作 OOTB 分组来对待。

8.5　偶联关闭程序

端点退出服务时，应当停止它的偶联。偶联的停止可以使用中止程序和关闭程序。偶联的中止可以在有未证实的数据时就中止，偶联的两端点都舍弃数据，并且不递交到对端。偶联的关切可以认为是一个正常的关闭程序，这样队列中所有的数据都可以递交到对端。但是在关闭的情况下，SCTP 不支持半开放状态(类似 TCP)，即半关闭状态可以由一方继续发送数据，而另一方已经处于关闭状态。任何一个端点执行了关闭程序，则偶联的两端都将停止接受从其用户发来的新数据，并且在发送或接收到 SHUT DOWN 数据块时，把队列中的数据递交给对端。

8.5.1　偶联的中止

当一个端点决定中止一个现存的偶联，则它应向对端点发送 ABORT 数据块，发送方必须把发送的分组中填上对端的验证标签，并且不在 ABORT 数据块中捆绑任何 DATA 数据。

端点不必响应包含 ABORT 数据块的任何分组(参见 8.4.4 节)。

收到 ABORT 的端点应当根据 8.4.5.1 节规定购验证标签规则进行检查。

在检查完验证标签后，接收端点应当从记录中清除该偶联，并且向高层来报告偶联的停止。

8.5.2　偶联的关闭

使用 SHUTDOWN 原语(见 10.1 节)，偶联端点的高层可以正常地关闭，这允许来自偶联关闭启动者的未被完成的所有数据块在偶联结束前可以被递交。

在收到来自高层的 SHUTDOWN 原语，则端点进入 SHUTDOWN-PENDING 状态，并且保持该状态直到所有证实成的数据被对端证实。此时端点不从高层接收新数据，如果需要可以填充丢失的数据，并可以重发数据到远端。

一旦所有未完成的数据被证实后，端点应当发送 SHUTDOWN 数据块给对端，该数据块中的累积 TSN ACK 字段中应包含从对端收到的最后一个连续的 TSN。然后端点应当启动 T2-shutdown 定时器并进入 SHUTDOWN-SENT 状态，如果定时器超时，则端点必须重新发送 SHUTDOWN 数据块，该数据块中应包含更新过的从对端点收到的最后一个顺序的 TSN。

必须遵循第 8.2.3 节的规定来确定适当的 T2-shutdown 定时器取值，为了指示 TSN 中的不连续间隙，端点可以把 SACK 数据块和 SHUTDOWN 数据块捆绑在一个 SCTP 分组中传送。

端点应当根据 Association.Max.Retrans 协议参数来限制 SHUTDOWN 数据块的重发次数，如果超过了这个门限，端点应当破坏这个 TCB 并且必须向高层报告对端点不可达(该偶联进入 CLOSED 状态)。从对端收到的任何分组(即对端发送所有队列中的 DATA 数据块)应当清除端点的重发计数器并且重新启动 T2-Shutdown 定时器，给对端足够的机会发送所有未发送的排队的数据块。

在收到 SHUTDOWN 数据块后，对端点应当作以下操作。

—进入 SHUTDOWN-RECEIVED 状态；

—停止接收从 SCTP 用户来的新数据；

—通过检查数据块的累积 TSN ACK 字段，验证所有未完成的 DATA 数据块已经被 SHUTDOWN 的发送方接收。

—旦端点进入了 SHUTDOWN-RECEIVED 状态，则端点不能通过发送 SHUTDOWN 响应高层协议(ULP)的请求，并且应当丢弃后续的 SHUTDOWN 数据块。

如果还有未完成的 DATA 数据块，SHUTDOWN 的接收方应当按照第 8.2 节定义的程序继续正常的数据传送过程，直到所有未证实的 DATA 数据块被证实后，但 SHUTDOWN 的接收方不能接受从 SCTP 用户发来的新数据。

当处于 SHUTDOWN-SENT 状态，SHUTDOWN 的发送方必须立即用 SACK 和 SHUTDOWN 响应每个收到的包含一个或多个 DATA 数据块的分组，并重新启动 T2-shutdown 定时器。如果已经没有未证实的 DATA 数据块，SHUTDOWN 的接收方应当发送 SHUTDOWN ACK 并启动本端

T2-shutdown 定时器，并且进入 SHUTDOWN-ACK-SENT 状态，如果定时器超时了，端点必须重新发送 SHUTDOWN ACK。

SHUTDOWN ACK 的发送方应当根据 Association.Max.Retrans 协议参数限制重发 SHUT7DOWN ACK 数据块数量，如果超过了这个门限，端点应当破坏这个 TCB 并且必须向高层报告对端点不可达(该偶联进入 CLOSED 状态)。

在收到 SHUTDOWN ACK 后，SHUTDOWN 的发送方应当停止 T2-shutdown 定时器，并向对端点发送 SHUTDOWN COMPLETE 数据块，并清除偶联的所有记录。

在收到 SHUTDOWN COMPLETE 数据块后，端点应当验证是否处于 SHUTDOWN-ACK-SENT 状态，如果不是则舍弃该数据块，如果端点处于 SHUTDOWN-ACK-SENT 状态，则端点应当停止 T2-shutdown 定时器并清除偶联的所有记录(随后偶联进入 CLOSED 状态)。

—个端点应当保证所有未完成的 DATA 数据块在启动关闭程序前都应被证实。

如果端点处于 SHUTDOWN-PENDING 、 SHUTDOWN-SENT 、 SHUTDOWN-RECEIVED 、 或 SHUT-DOWN-ACK-SENT 的状态，则端点应当拒绝任何从高层来的发送新数据请求。

如果端点处于 SHUTDOWN-ACK-SENT 状态，而又收到了带有起源和目的地传送地址的属于该偶联的(可以是在 IP 地址中或者是在 INIT 数据块中)INIT 数据块(例如：如果 SHUTDOWN COMPLETE 丢失)，则端点应当丢弃 INIT 数据块并且重发 SHUTDOWN ACK 数据块。

注：收到的 INIT 数据块中带有相同起源和目的地 IP 地址作为分配给端点的传送地址，但使用了不同的端口号，用来表示重新启动了另一个偶联。

INIT 或 COOKIE 的发送方应当用在一个 SCTP 分组中用单独的 SHUTDOWN COMPLETE 来响应 SHUTDOWN-ACK，这个 SCTP 分组的公共头的验证标签字段应当设置为与收到的 SHUTDOWN ACK 分组的标签相同，这主要是考虑了 8.4.4 节提到的 OOTB 分组的情况。INIT 的发送方可以让 T1-init 定时器继续运行，并继续保持 COOKIE-WAIT 或 COOKIE-ECHOED 状态。正常的 T1-init 定时器超时将使 INIT 或 COOKIE 数据块重发，并重新启动一个新的偶联。

如果在 COOKIE WAIT 或 COOKIE ECHOED 状态下收到了 SHUTDOWN,则应当舍弃该 SHUTDOWN 数据块并且不产生任何报告。

如果一个端点是处于 SHUTDOWN-SENT 状态，并且从对端点收到了 SHUTDOWN 数据块，端点应当立即用 SHUTDOWN ACK 进行响应，并且进入 SHUTDOWN-ACK-SENT 状态，并重新启动 T2-shutdown 定时器。

如果一个端点是处于 SHUTDOWN-ACK-SENT 状态，并且收到了一个 SHUTDOWN ACK，则该端点应当停止 T2-shutdown 定时器，并向对端点发送 SHUTDOWN COMPLETE 数据块，并且清除偶联的所有记录。

附录 A

(资料性附录)

SCTP 的状态转移图

在 SCTP 偶联的存活期内，SCTP 端点的偶联程序从一个状态转移到另一状态以响应不同的事件。能潜在影响偶联状态的事件包括：

—SCTP 用户原语呼叫，如[ASSOCIATE]、[SHUTDOWN]和[ABORT]等；

—收到 INIT、COOKIE ECHO、ABORT、SHUTDOWN 等控制数据块；

—一些超时事件。

图 A.1 和 A.2 所示的状态图解释了状态的变化、引起变化的事件和产生的结果。注意一些错误条件没有在状态图中示出。所有特殊情况的充分描述可在下文找到。

注：数据块的名字用大写字母给出，而参数名的第一个字母是大写，例如 COOKIE ECHO

数据块类型对 State Cookie 参数。如果发生了不止一个事件 / 消息，它们导致的状态转移用标签(A)，(B)标出。

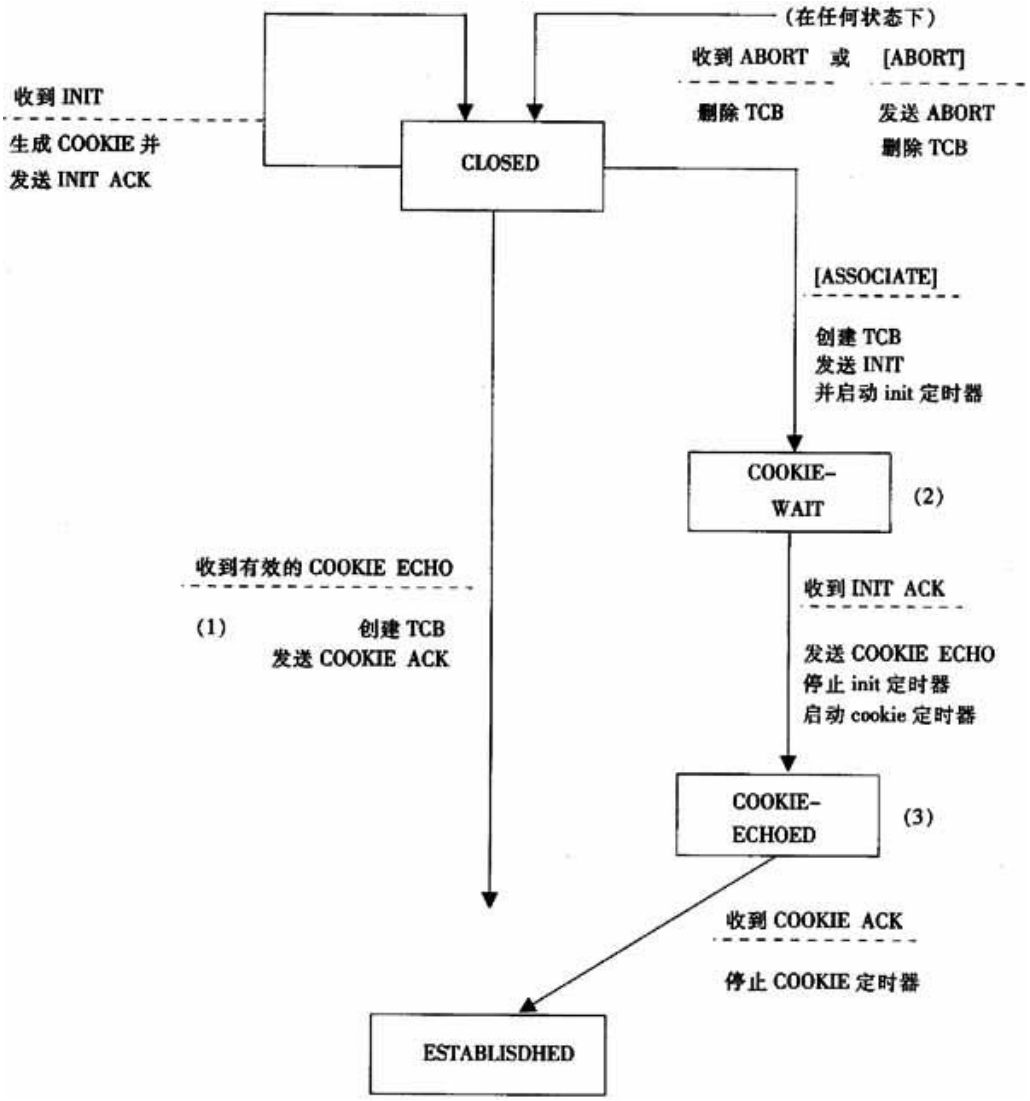

图 A.1　SCTP 的状态转移图 1

图中步骤的含义如下。

(1)如果在收到的 COOKIE ECHO 里的 State Cookie 是无效的(即没有通过一致性检查)，接收者必须丢弃这个分组，并不产生任何指示。或者，如果收到的 State Cookie 超时(见 8.1.1.5 节)，接收者必须返回 ERROR 数据块。在这两种情况下，接收者都停留在 CLOSED 状态。

(2)如果 T1-init 定时器超时，端点必须重传 INIT 并重启 T1-init 定时器，而不改变状态。这个过程必须被重复 Max.Init.Retransmits 次。在这之后，端点必须中止启动程序并向 SCTP 用户报错。

(3)如果 T1-cookie 定时器超时，端点必须重传 COOKIE ECHO 并重启 T1-cookie 定时器而不改变状态。这个过程必须被重复 Max.Init.Retransmits 次。在这之后，端点必须中止启动程序并向 SCTP 用户报错。

(4)在 SHUTDOWN-SENT-SENT 状态，端点必须无时延地确认任何收到的数据块。

(5)在 SHUTDOWN-RECEIVED 状态，端点不能接受任何本地 SCTP 用户新的发送请求。

(6)在 SHUTDOWN-RECEIVED 状态，端点必须传送或重传数据，直到队列中的所有数据发

送完后离开这个状态。

(7)在 SHUTDOWN-ACK-SENT 状态，端点不能接收任何本地 SCTP 用户新的发送请求。

CLOSED 状态用于指出偶联没有被创建(即不存在)。

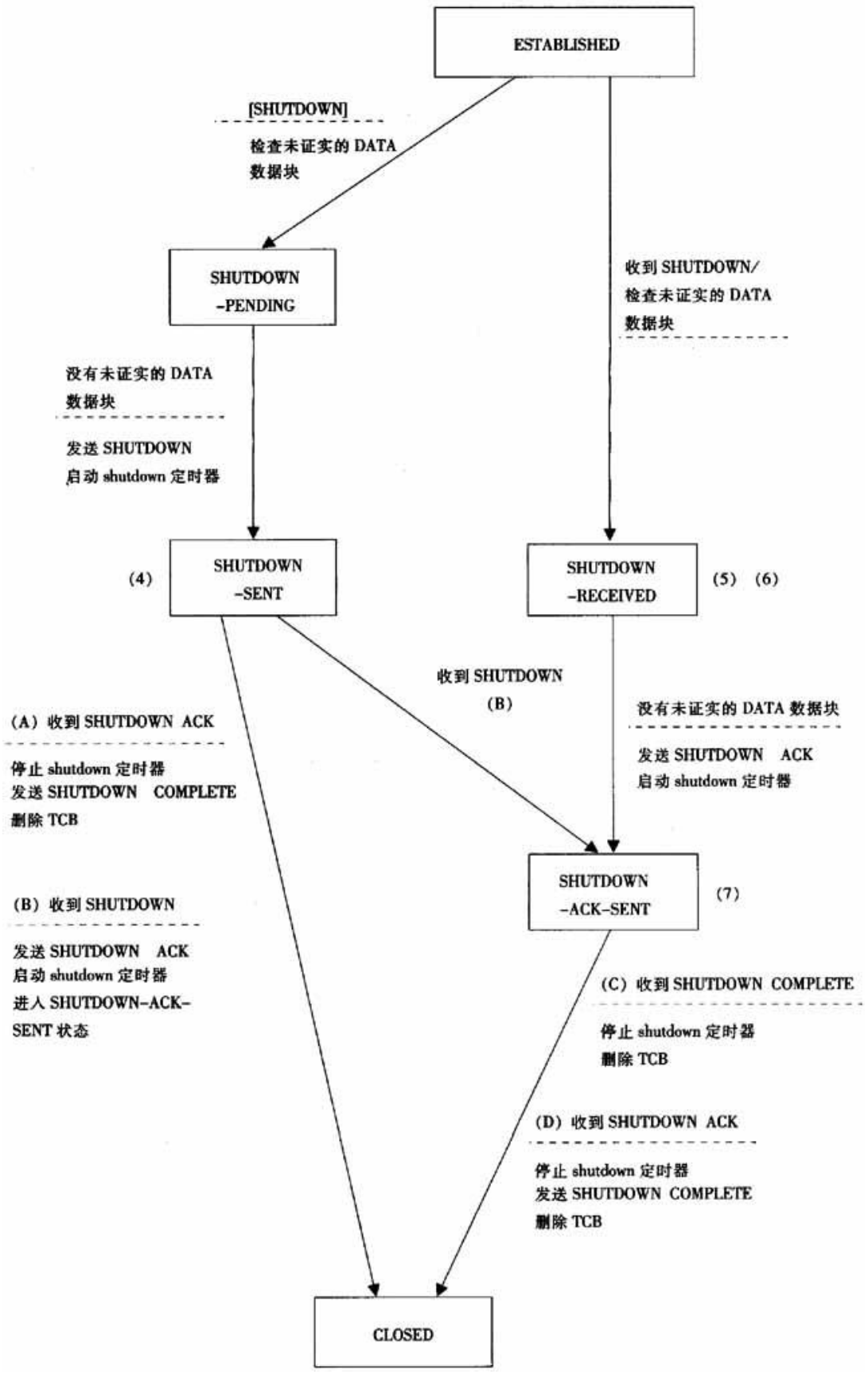

图 A.2　SCTP 的状态转移图 2

附录 B

（资料性附录）

SCTP 程序示例

为了更直观地说明 SCTP 的应用程序，本附录采用消息流程图的方式分别给出了正常的偶联建立、偶联重新启动、数据块延迟证实、定时器使用原则和使用 SACK 来报告接收数据间隔的程序示例，这里的程序示例只用于说明目的，具体的实施不必与示例中完全相同。

B.1　正常偶联建立的示例

在图 B.1 的示例中，A 点用来启动偶联建立，并向 Z 发送一个用户消息，随后后 Z 向 A 发送两个用户消息(假定这些消息没有捆绑和分段)。

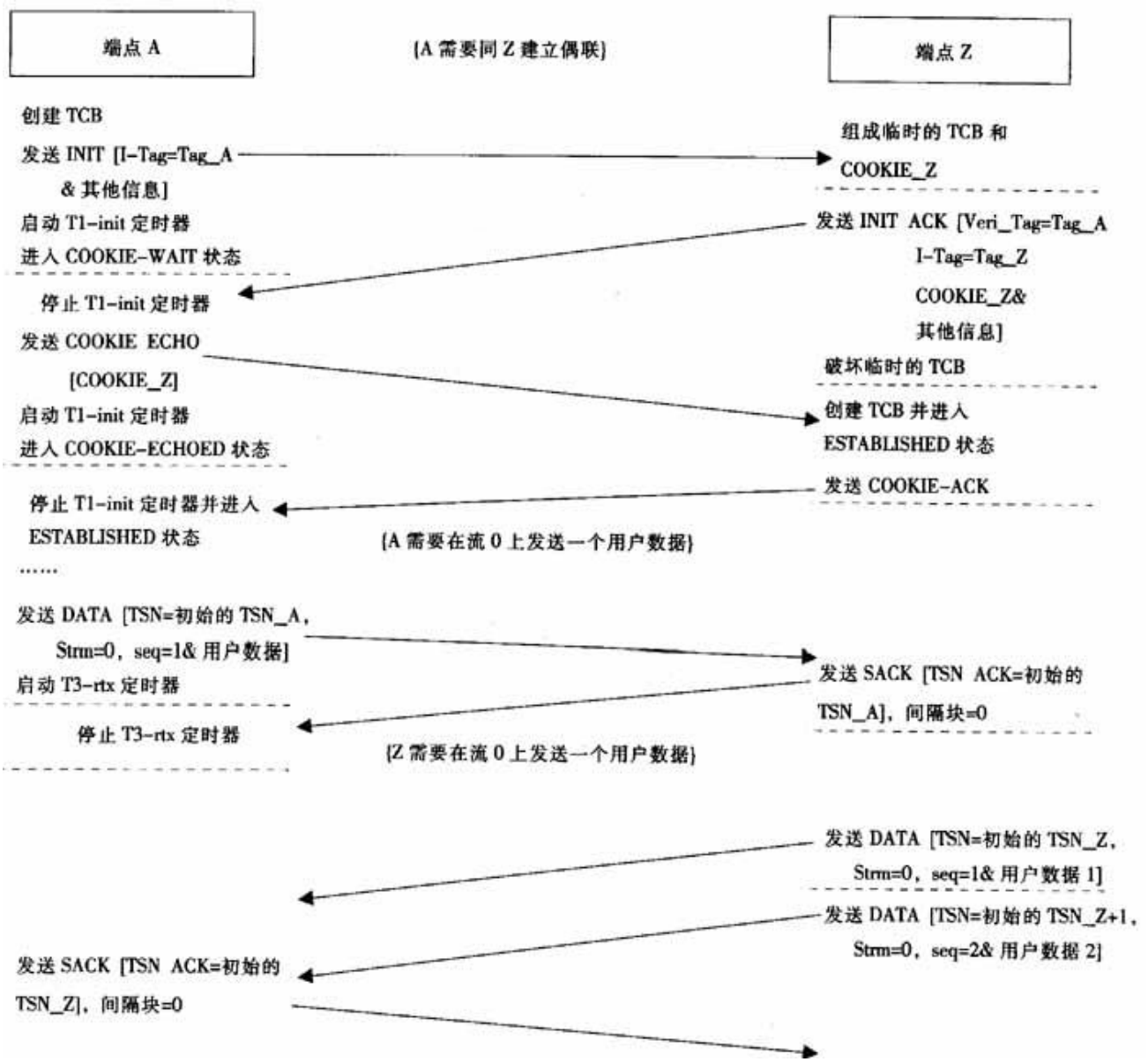

图 B.1　偶联正常建立和数据发送的示例

如果 A 点在发送完 INIT 或 COOKIE ECHO 数据块后 T1-init 定时器，则需要重新发送带有相同启动标签(即 TAG_A)或状态 COOKIE 的 INIT 或 COOKIE ECHO 数据块，这个过程也将应重复 Max.Init.Retransmits 次，直到 A 认为 Z 不可达，同时向高层发送故障报告(此时偶联进入关闭状态)。在重发 INIT 时，端点必须根据 8.2.3 节定义的原则来确定适当的定时器值。

B.2　偶联重启动的示例

在图 B.2 的示例中，在发生重启动事件后 A 点启动偶联，端点 Z 直到收到交换的消息后才知道重启动发生(即心跳消息尚未检测出 A 点的故障)，图 B.2 假定没有捆绑和分段。

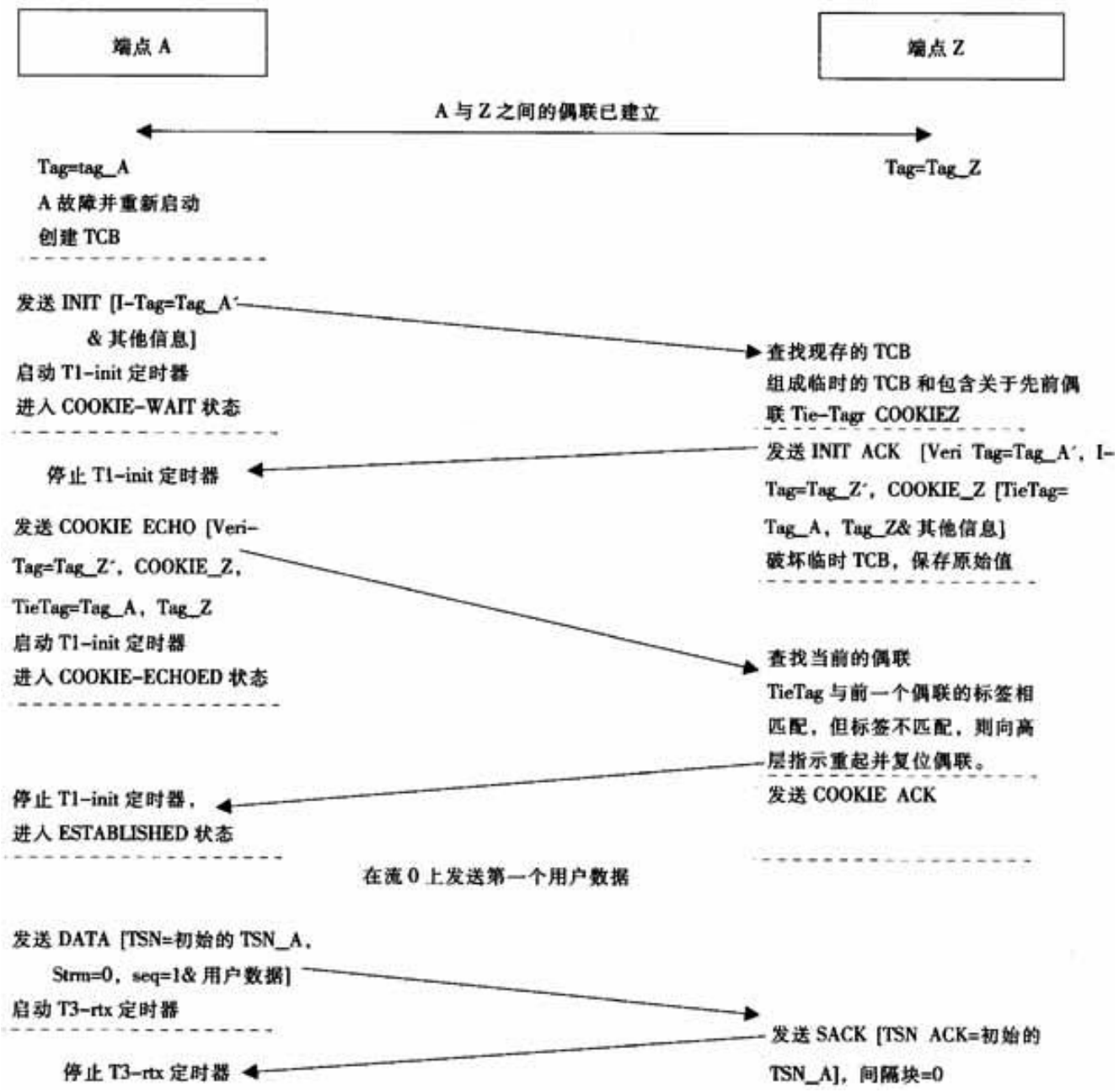

图 B.2　偶联重新启动的示例

B.3　延时证实的示例

图 B.3　延时证实的示意

B.4　使用 SACK 报告间隔的示例

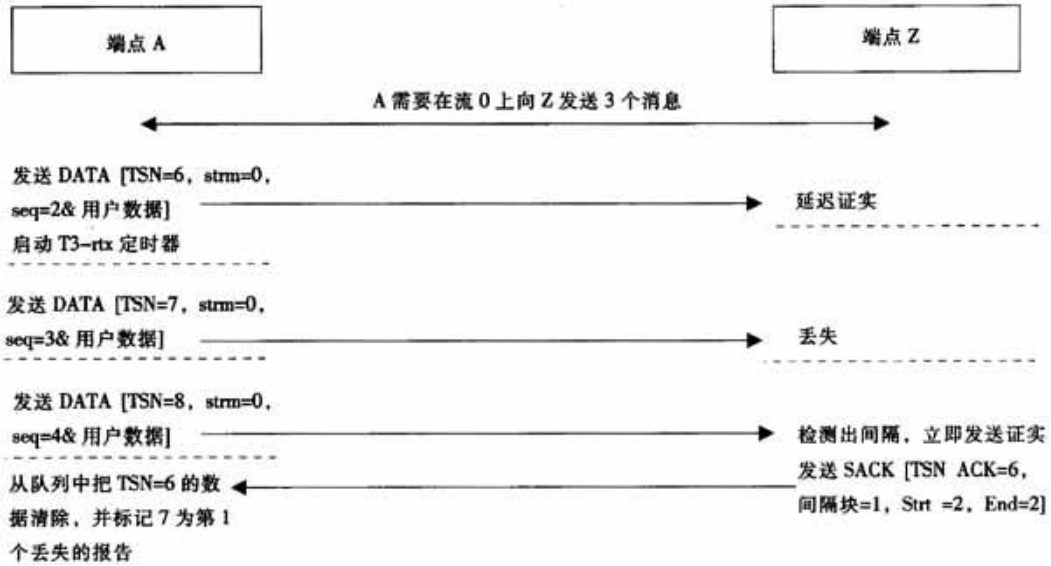

图 B.4　使用 SACK 来报告接收数据间隔

B.5　延时证实的示例

图 B.5 的实例说明了使用定时器的不同原则(假定这个示例中接收方使用的是延时证实方面)。

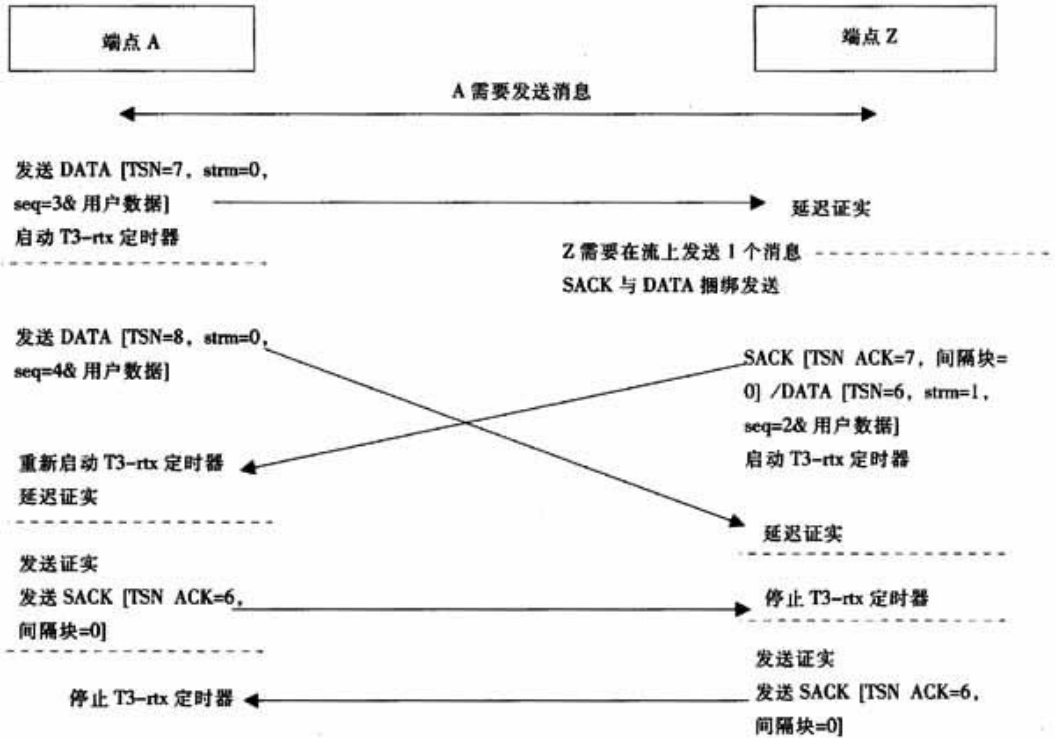

图 B.5　定时器应用示例

附录 C

(资料性附录)

明确的拥塞通知

明确的拥塞通知功能在 RFC 2481 建议中进行了定义，在 RFC 2481 建议中详细描述了除用数据报丢失确定拥塞之外的一种获知拥塞状态的算法。对于本标准定义的 SCTP 协议，这个功能特征是任选的，同时 RFC 2481 建议规定的内容也适用于 SCTP 协议。如果在实施 SCTP 的过程中需要使用明确的拥塞通知功能，则应参照本附录的说明，注意在 SCTP 上实施明确的拥塞通知功能时与 RFC 2481 规定的明确的拥塞通知功能的区别。

协商：RFC 2481 详细描述了在 TCP 连接的 SYN 和 SYN ACK 阶段对 ECN 的协商，发送 SYN 的一方在 TCP 的标志中设置两个比特，发送 SYIV ACK 的一方则只设置一个比特，这是为了保证两方都可以正确地使用 ECN 能力。对于 SCTP 而言，则不必这样做。为了指示一个端点能够支持 ECN，端点应当在 INIT 或 INIT ACK 数据块中增加一个为 ECN 预留的 TLV 参数，这个 TLV 中不包括任何参数值，其格式如图 C.1 所示。

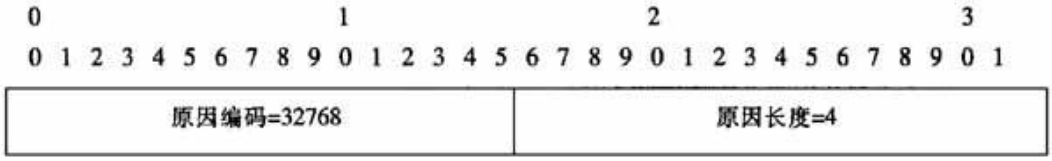

图 C.1　ECN 能力参数的格式

ECN 响应(ECN-Echo)：在 RFC 2481 建议中详细描述了由接收方在 TCP 证实中使用的一个特定比特，回送给发送方用来通知在网络中经历了拥塞。对 SCTP 而言，在 ECNE 数据块中

也包含了相同的指示，这个数据块只包含了一个数据单元，即标记了 CE 比特的 IP 数据报的最低的 TSN，其格式如图 C.2 所示。

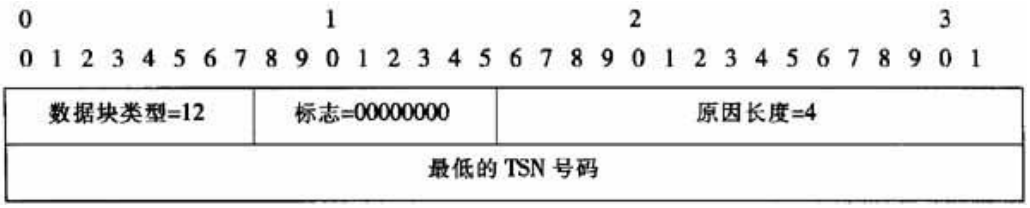

图 C.2　ECNE 数据块的格式

注：ECNE 数据块被看作是控制数据块。

CWR：在 RFC 2481 建议中详细描述了由发送方在下一个出局 TCP 段的头中使用一个特定比特，向它的对端指示它已经降低了拥塞窗口，这个比特称为 CWR 比特，其格式如图 C.3 所示。对于 SCTP，在 CWR 数据块中也包含了相同的指示。这个数据块中也包含一个数据单元，即在 ECNE 数据块中发送的 TSN 号码。这个单元表示标记了 CE 比特数据报的最低 TSN。

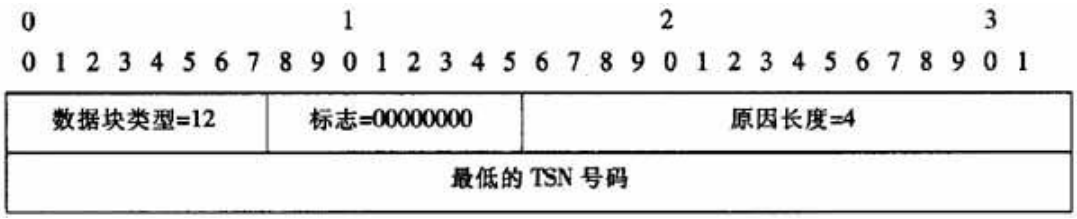

图 C.3　CWR 数据块的格式

注：CWR 数据块被看作是控制数据块。