There are, in fact, some ambiguities in this choice for applications that want only a lightweight reliable service. For example, an application might want notification of lost messages but not actually care enough to have a transport protocol attempt to redeliver the missing data. For such applications the choice is between using TCP and attempting to reduce the overhead by using as few options as possible (perhaps controlling the behavior of the local TCP implementation through runtime parameters at the sockets API), and using UDP with additional application-level protocol exchanges to provide the level of service that is required.

In the end, the fact that TCP is implemented and readily available on most platforms usually counts for a lot, and the savings in protocol design and application implementation usually means that TCP is chosen whenever there is any doubt.

7.3.11 Protocols That Use TCP

Many application protocols associated with bulk transfer of data use TCP. These include the File Transfer Protocol (FTP), the Hypertext Transfer Protocol (HTTP), and email protocols such as the Simple Mail Transfer Protocol (SMTP) and the Post Office Protocol (POP3).

Telnet is an interesting example of a protocol that commonly transfers small amounts of data but still uses TCP. The command-response nature of Telnet and its immediate visibility to a human user is such that it is essential to ensure that messages are delivered correctly.

TCP is also used by control and routing protocols to transport their data. The Border Gateway Protocol (BGP-4) and the Label Distribution Protocol (LDP) are good examples. The use of TCP makes sense for them because they establish clear and long-lived associations with "adjacent" nodes over which they need to keep exchanging information. Using TCP means that these protocols do not need to include methods to track the data that is exchanged—they are heavily dependent on the reliability of TCP. On the other hand, many control and routing protocols that use TCP need to include their own keep-alive mechanisms to ensure that the TCP connection is still active and to detect connection failures in a timely manner.

7.4 Stream Control Transmission Protocol (SCTP)

TCP is a well-established, proven transport protocol that is used by a substantial number of application protocols. So why invent another transport protocol? As can be seen by the number of RFCs that apply to TCP, it has been necessary to make small tweaks to the protocol over the years as the Internet has evolved and as the requirements on a transport protocol have become clearer. A new class of control protocols has recently started to be used within the Internet to signal Packet Switched Telephone Network (PSTN) connections, and these protocols place a high level of requirements on their transport service provider. Rather than developing still more modifications to TCP, the opportunity was taken to invent a new transport protocol, the Stream Control Transmission Protocol (SCTP). SCTP is defined in RFC 2960.

Although SCTP was designed specifically to meet the transport requirements of PSTN signaling messages over IP networks, it is available as a transport protocol for any application or control protocol. The main features of SCTP are as follows. Many of these will be familiar to those who understand the services provided by TCP.

- SCTP is a reliable connection-oriented transport protocol.
- It operates over a connectionless network protocol such as IP.
- It provides acknowledged, error-free, nonduplicated transfer of user data.
- It can be supplied and can deliver data in large blocks.
- It fragments data to fit within the MTU size.
- It includes sender pacing and congestion avoidance schemes.

In addition, SCTP provides some new, unique features. SCTP facilitates the establishment and maintenance of multiple *streams* between the same pair of end points. This is equivalent to having multiple conversations between two people at the same time on the same phone call, but actually allows for an extra level of hierarchy in the address scheme. To revert to the postal analogy, this is like having a whole family served by a single mailbox with different mail exchanges (streams) going on to different family members. Messages within the SCTP connection may be delivered in strict order of arrival across all streams or may be separated into individual streams for delivery—in either case, SCTP ensures in-order delivery within each stream.

Some moderate performance enhancements are included to allow multiple SCTP messages to be bundled into a single SCTP packet for transmission. This reduces the network overhead of the IP header for small SCTP messages and, more importantly, reduces the processing overhead associated with sending and receiving each SCTP packet.

SCTP also includes some improved network-level fault tolerance through the concept of multihoming. At either or both ends of an SCTP association multiple addresses may be used so that the association can be dynamically moved from one point of attachment to another. This allows distinct routes to be engineered through the network for the different addresses and allows the SCTP association to be redirected around network outages without the need to tear it down and reestablish it. Figure 7.18 shows how an SCTP association is formed to support multiple streams across an IP network using multihoming.

Finally, SCTP also includes some additional security features to provide resistance to flooding and masquerade attacks.

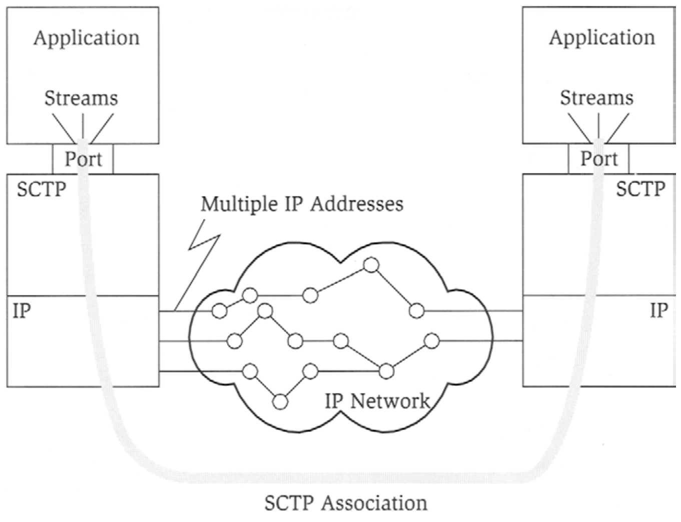

Figure 7.18 An SCTP Association carries multiple streams between a pair of ports on end points that may be identified by multiple IP addresses.

7.4.1 SCTP Message Formats

SCTP communicates end-to-end using SCTP packets that are sent within IP datagrams using the protocol identifier value of 132. Each SCTP packet contains a single SCTP header and one or more SCTP *chunks*. The SCTP header identifies the SCTP association and contains security and verification details. Each chunk may be a control message applicable to the association or one of the streams that run through it, or may be data being exchanged on one of the streams. Figure 7.19 shows how an SCTP packet is constructed.

Just as in UDP and TCP, the primary identifiers in the SCTP header (shown in Figure 7.20) are the source and destination port numbers. Port numbers for SCTP are managed from the same space as they are for the other IP transport protocols and are administered by IANA. This means that an application can be run over any transport protocol without needing to change its port number. The header also includes a Verification Tag assigned during association establishment. The sender of an SCTP packet inserts the receiver's tag as a form of protection

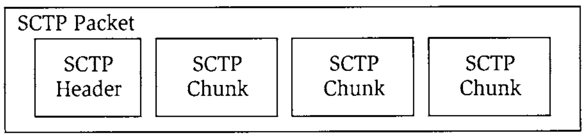

Figure 7.19 An SCTP packet contains a single header and one or more SCTP chunks.

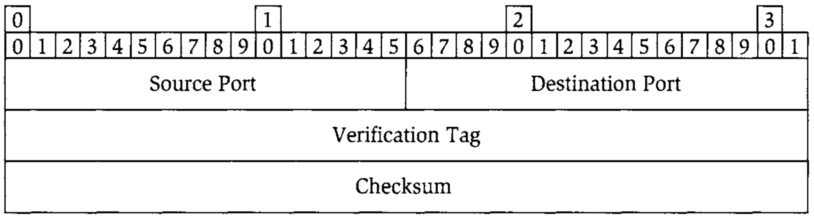

Figure 7.20 The SCTP message header.

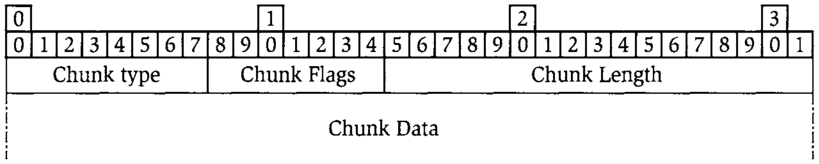

Figure 7.21 SCTP chunks have a common format.

against old SCTP packets being delivered very late and also to help protect the association from security attacks.

The final field in the SCTP header is a 32-bit checksum field. Unlike TCP and UDP, SCTP does not use the standard IP checksum, but instead uses the Adler 32-bit checksum. This checksum is somewhat of an improvement over the 16-bit Fletcher checksum (see Section 5.6.2), allowing fewer corruptions to be missed. Note that SCTP does not use a pseudoheader since the inclusion of the verification tag provides some protection against misdelivery, but also because the multihoming nature of SCTP means that it would not be so simple to pick the correct IP addresses to use in the pseudoheader. The Adler 32-bit checksum is described in RFC 1950.

Note that the SCTP header does not include a length field. The length of the whole packet can be deduced from the size of the reassembled IP fragments.

Each SCTP chunk has a common format, as shown in Figure 7.21. It begins with a type identifier to indicate how the chunk should be interpreted, a set of flags that have distinct meanings according to the chunk type, and a length indicator that shows the length of the entire chunk, including the type, flags, and length fields. Each chunk must start on a 4-byte boundary, so it may be necessary to insert padding between chunks, but the chunk length still reflects the actual length of a chunk without the padding.

After this chunk common header, the contents of the chunk are interpreted according to the chunk type. Most chunk data are built up from a type-dependent header followed by a sequence of chunk parameters. Each chunk parameter is encoded as a type-length-variable (TLV) with 2 bytes assigned to the type and 2 bytes to the length, which is calculated to include the type and length fields. As with chunks, the chunk parameters must start on 4-byte boundaries, and

Table 7.2 The SCTP Chunk Types

Chunk Type	*Chunk Usage*
0	Payload Data
1	Association Initiation
2	Initiation Acknowledgement
3	Selective Acknowledgement
4	Heartbeat Request
5	Heartbeat Acknowledgement
6	Abort
7	Shutdown Request
8	Shutdown Acknowledgement
9	Operation Error Notification
10	State Cookie Echo
11	State Cookie Echo Acknowledgement
12	Explicit Congestion Notification Echo
13	Congestion Window Reduced
14	Shutdown Complete
15 to 255	Reserved by IETF

so padding may need to be inserted between chunk parameters—this padding is not included in the parameter length.

Note that chunks within a single SCTP packet all apply to the same association, but may refer to different streams. It is important that some sense of order be preserved when processing chunks from the same packet—any chunk that applies to the whole association must be processed in order, and chunks for an individual stream must also be kept in sequence.

Table 7.2 lists the defined SCTP chunk types. Note that each chunk is effectively a control message in its own right.

7.4.2 Association Establishment and Management

Figure 7.22 shows some sample chunk exchanges in the life of an SCTP association. In SCTP the end points are perceived as peers rather than as client and server. This is a semantic nicety that allows applications to have less of a master-slave relationship, but does not alter the fact that one end must initiate the association.

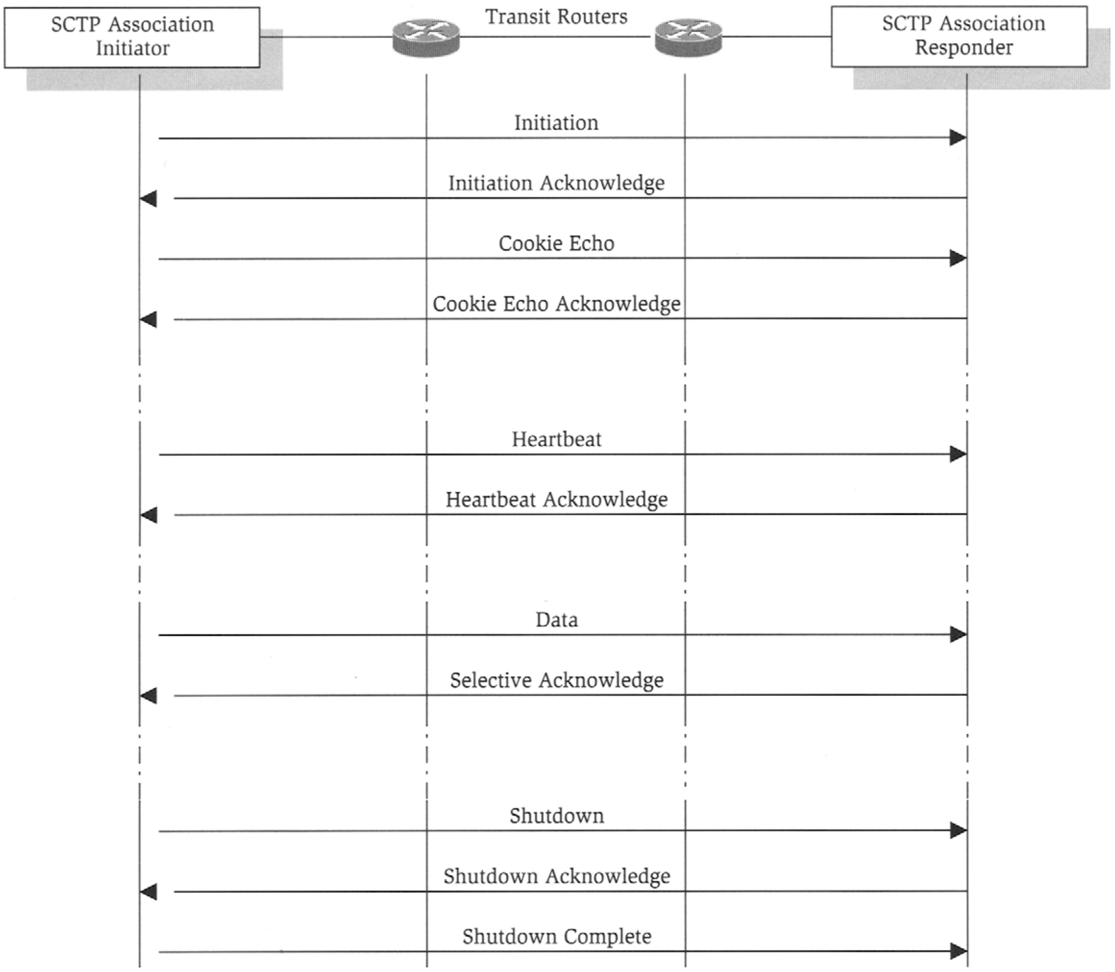

Figure 7.22 SCTP chunk exchanges during the life of an association.

Association establishment is initiated when one SCTP peer sends a packet containing an Association Initiation chunk, shown in Figure 7.23. This chunk carries several common fields to negotiate the terms of the association and a series of optional parameters encoded as chunk parameters. The common fields are listed in Table 7.3. The optional parameters carried in an Association Initiation chunk are listed in Table 7.4.

The remote end point responds to an Association Initiation chunk with an Initiation Acknowledgement chunk, shown in Figure 7.24. This chunk accepts the association and supplies the negotiated values and reverse direction parameters for management of the association. The same common fields are present

Table 7.3 Common Fields in an SCTP Association Initiation Chunk

Association Initiation Chunk Field	*Use*
Initiate Tag	This 32-bit tag is exchanged during association initialization and is placed on every message that applies to the session. It is used to help prevent security breaches and to validate that individual packets apply to this instance of the association. The tag must not have value zero.
Advertised Receiver Window Credit	The initial size of the receiver window—that is, the number of bytes that the sender may send. This value may be modified by Selective Acknowledgement chunks.
Number of Outbound Streams	Defines the maximum number of outbound streams the sender of this chunk wants to create in this association. A value of zero must not be used.
Number of Inbound Streams	Defines the maximum number of inbound streams the sender of this chunk is willing to allow the receiver to create in this association. A value of zero must not be used.
Initial Transmission Sequence Number (TSN)	The initial TSN is the sequence number that identifies the first byte of data that will be sent on the association. Any number in the range 0 to 4,294,967,295 is acceptable. Some implementations randomize this value and set it to the value of the Initiate Tag field.

Table 7.4 Optional Parameters in an SCTP Association Initiation Chunk

Parameter Type	*Parameter Name*	*Use*
5	IPv4 Address	One of the IPv4 addresses that may be used to identify the sender's end of the association. Multiple IPv4 and IPv6 addresses may be present. If no addresses are present, the SCTP application should use the address from the IP datagram that delivered the SCTP packet.
6	IPv6 Address	One of the IPv6 addresses that may be used to identify the sender's end of the association. Multiple IPv4 and IPv6 addresses may be present. If no addresses are present, the SCTP application should use the address from the IP datagram that delivered the SCTP packet.
9	Cookie Preservative	A value in milliseconds by which the sender is suggesting that the cookie timeout value be increased to prevent the cookie expiring again (as it has just done) during association establishment.
11	Host Name Address	A single host name that may be used to identify the sender's end of the association. The host name may not be present along with any IPv4 or IPv6 addresses, and only one host name may be used.
12	Supported Address Types	The address types that the sender supports and from which the receiver may choose addresses for its end of the association. If this parameter is absent, the sender supports all address types.

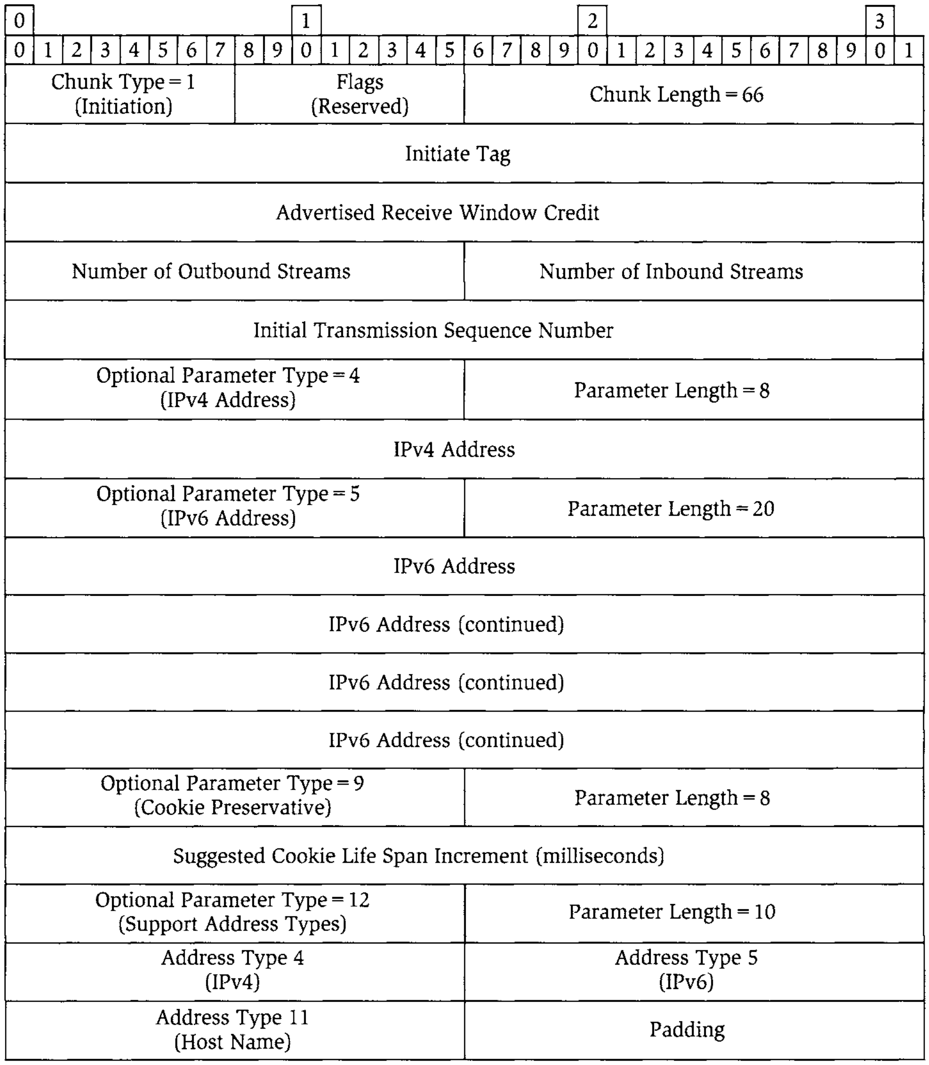
0
1
2
3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Chunk Type = 1 (Initiation)
Flags (Reserved)
Chunk Length = 66
Initiate Tag
Advertised Receive Window Credit
Number of Outbound Streams
Number of Inbound Streams
Initial Transmission Sequence Number
Optional Parameter Type = 4 (IPv4 Address)
Parameter Length = 8
IPv4 Address
Optional Parameter Type = 5 (IPv6 Address)
Parameter Length = 20
IPv6 Address
IPv6 Address (continued)
IPv6 Address (continued)
IPv6 Address (continued)
Optional Parameter Type = 9 (Cookie Preservative)
Parameter Length = 8
Suggested Cookie Life Span Increment (milliseconds)
Optional Parameter Type = 12 (Support Address Types)
Parameter Length = 10
Address Type 4 (IPv4)
Address Type 5 (IPv6)
Address Type 11 (Host Name)
Padding

Figure 7.23 The SCTP Association Initiation chunk.

that were used on the Association Initiation chunk. Some of the same optional parameters may also be present: the IPv4 Address, IPv6 Address, and Host Name Address parameters may all be included. Additionally, the Initiation Acknowledgement chunk must include the State Cookie parameter (type 7). The State Cookie is used to authenticate and correlate the Initiation Acknowledgement chunk with the third stage of the four-way association establishment handshake, the State Cookie Echo chunk. The State Cookie contains all of the

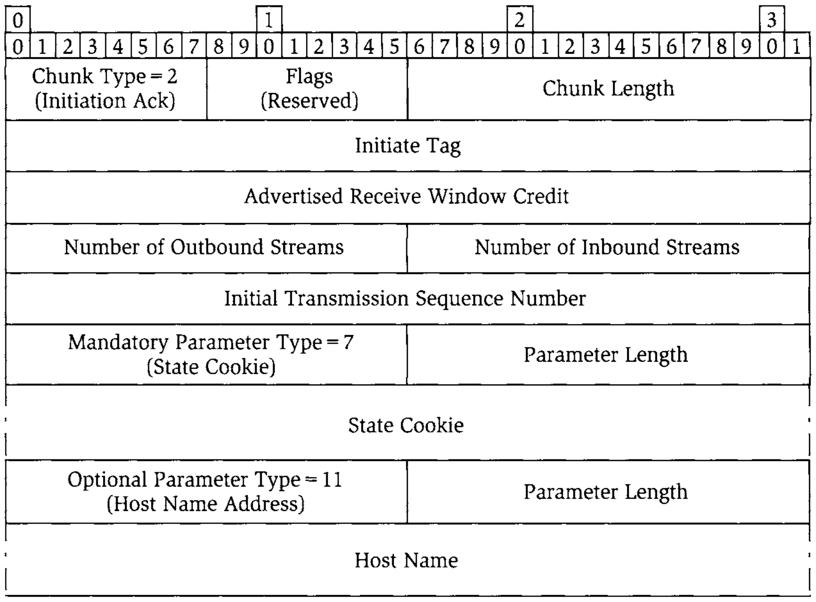
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Chunk Type = 2 (Initiation Ack)
Flags (Reserved)
Chunk Length
Initiate Tag
Advertised Receive Window Credit
Number of Outbound Streams
Number of Inbound Streams
Initial Transmission Sequence Number
Mandatory Parameter Type = 7 (State Cookie)
Parameter Length
State Cookie
Optional Parameter Type = 11 (Host Name Address)
Parameter Length
Host Name

Figure 7.24 The SCTP Initiation Acknowledgement chunk.

information the responder needs to coordinate between the SCTP chunks, and additionally includes a Message Authentication Code (MAC) to provide additional security. When a responder builds a State Cookie and sends it in an Initiation Acknowledgement chunk, it starts a timer to protect itself from leaving around half-open associations. The State Cookie may, therefore, timeout during association establishment. If the sender detects this, it can try to reestablish the association and may present a Cookie Preservative parameter to suggest an amount by which the receiver should increment its timer so that the association will be correctly established.

The Initiation Acknowledgement chunk may also contain an Unrecognized Parameter (type 8), which allows the responder to return any parameters that were seen on the Association Initiation chunk that it does not support. Note that the Association Initiation and the Initiation Acknowledgement chunks may be quite large because of the presence of a potentially large number of addresses and the size of a State Cookie.

If the initiator is happy with the parameters on the Initiation Acknowledgement chunk, it echoes the responder's State Cookie back to it using a State Cookie Echo chunk, which contains just the cookie as received on the Initiation Acknowledgement chunk. As a final handshake, the responder acknowledges the State Cookie Echo chunk with a State Cookie Echo Acknowledgement chunk that contains no data or parameters. At this point the association is up and ready to carry data in both directions.

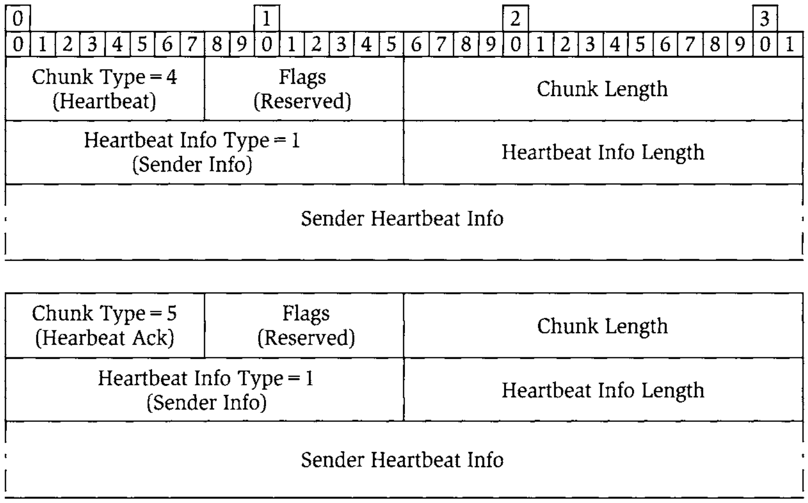

Figure 7.25 The SCTP Heartbeat and Heartbeat Acknowledgement chunks.

Once an association is open, the SCTP implementation may probe it periodically to check that it is still established and active. It does this using the Heartbeat and Heartbeat Acknowledgement chunks shown in Figure 7.25. The Heartbeat chunk contains Sender Heartbeat Information, which is in any format that the sender may choose and is transparent to every node apart from the sender. The information would normally include a timestamp in local format and probably the source and destination addresses used for the SCTP packet that contains the chunk. When a receiver gets a Heartbeat chunk it turns it around as a Heartbeat Acknowledgement chunk and copies the sender information back to the sender. A receiver should send the SCTP packet that contains the Heartbeat Acknowledgement as soon as it can and should not wait for the packet to be filled with other chunks.

An active association can carry data for any of the streams that thread it. The process of data transfer is described further in Section 7.4.3.

Orderly association closure requires a three-way handshake in SCTP. The Shutdown Request chunk is used to begin the process. It contains the sequence number of the last received contiguous byte of data on the association. Note that although acknowledgement of noncontiguous data is allowed (see Section 7.4.3), this facility is not available on association shutdown. Either end point may initiate shutdown and, just as in TCP, the shutdown may be staggered, with the end that begins the shutdown sending no further data but the other end able to continue to send. Eventually, when the remote end has finished sending data, it responds with a Shutdown Acknowledgement chunk and the end that started the shutdown confirms this step with a Shutdown Complete chunk. At this

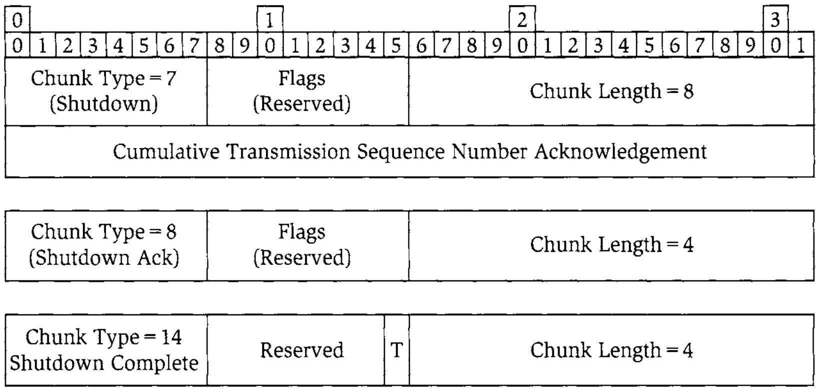

Figure 7.26 The Shutdown Request, Shutdown Acknowledgement, and Shutdown Complete chunks used in graceful association termination in SCTP.

point the association is closed. Figure 7.26 shows the three chunks used in the shutdown sequence.

There is one flag on the Shutdown Complete chunk. The T-flag is used (set to zero) to indicate that the sender of the message found and destroyed a Transmission Control Block (TCB) associated with this association—in other words, that this was a normal shutdown and that state matching the received Shutdown Acknowledgement chunk was found. If the state couldn't be found, the Shutdown Complete chunk should still be sent to help the far end of the (nonexistent) association to clean up—in this case the T-flag should be set to 1.

Two other important SCTP chunks exist. The Abort chunk is used to preemptively abort an association if any error is detected during establishment or even during normal processing. The Abort chunk may contain one or more Cause parameters giving the reason for the chunk and passing associated data. In the same way, a nonfatal error observed during the life of an association can be reported using Cause parameters on an Operation error chunk. Note that an SCTP peer that receives what it considers to be a badly encoded Abort or Operation Error chunk must silently discard the chunk and must not respond with its own Abort or Operation Error chunk since to do so risks a tight loop of message exchanges.

Table 7.5 lists the cause codes defined for Cause parameters carried in Abort or Operation Error chunks. Along with each cause code is listed the additional information passed in the Cause parameter.

Figure 7.27 shows the Abort and Operation Error chunks with their payloads of Cause Parameters. The Abort chunk carries the T-flag with the same meaning as on the Shutdown Complete chunk.

Table 7.5 SCTP Cause Codes and Additional Information

Cause	*Meaning*	*Additional Information*
1	Invalid Stream Identifier	The value of the invalid stream identifier was received in a data chunk.
2	Missing Mandatory Parameter	This is a count of missing mandatory parameters and the parameter type number of each missing parameter.
3	Stale Cookie Error	A cookie has been received in a State Cookie Echo chunk but the cookie has expired by the number of microseconds indicated. Note that this value is in microseconds even though the Suggested Cookie Life Span Increment given by the Cookie Preservative chunk is in milliseconds.
4	Out of Resource	No data is passed when this error is reported.
5	Unresolvable Address	The complete unresolvable address is passed encoded as an SCTP parameter so that its type and length can be seen.
6	Unrecognized Chunk Type	This error returns the chunk type, flags, and length of the unrecognized chunk.
7	Invalid Mandatory Parameter	This error is returned when one of the mandatory parameters on an Association Initiate or Initiate Acknowledgement chunk is set to an invalid value. No data is returned with this error, so it is not possible for the sender to determine which parameter is at fault.
8	Unrecognized Parameters	This error returns the full SCTP parameter that is unrecognized.
9	No User Data	A data chunk (see below) was received with a valid TSN but no data was present. This error returns the TSN that was received.
10	Cookie Received While Shutting Down	No data is passed when this error is reported.

7.4.3 Data Transfer

Data transfer in SCTP is managed, as in TCP, as a single sequenced and numbered flow of bytes on the association. That is, each data chunk contains a Transmission Sequence Number (TSN) that identifies the first byte in the context of the association. The amount of data is indicated by subtracting the Data chunk parameters from the Data chunk length.

One of the most important features of SCTP is that it can multiplex more than one data stream onto the same association. It does this by identifying the data stream to which the data applies through a 16-bit field in the Data chunk parameters, as shown in Figure 7.28. Additionally, a stream sequence number encodes a message number from the sending application so that the data chunks can be reassembled by the receiving application. Finally, a payload protocol identifier is included to help applications that multiplex data from several protocols through the same association.

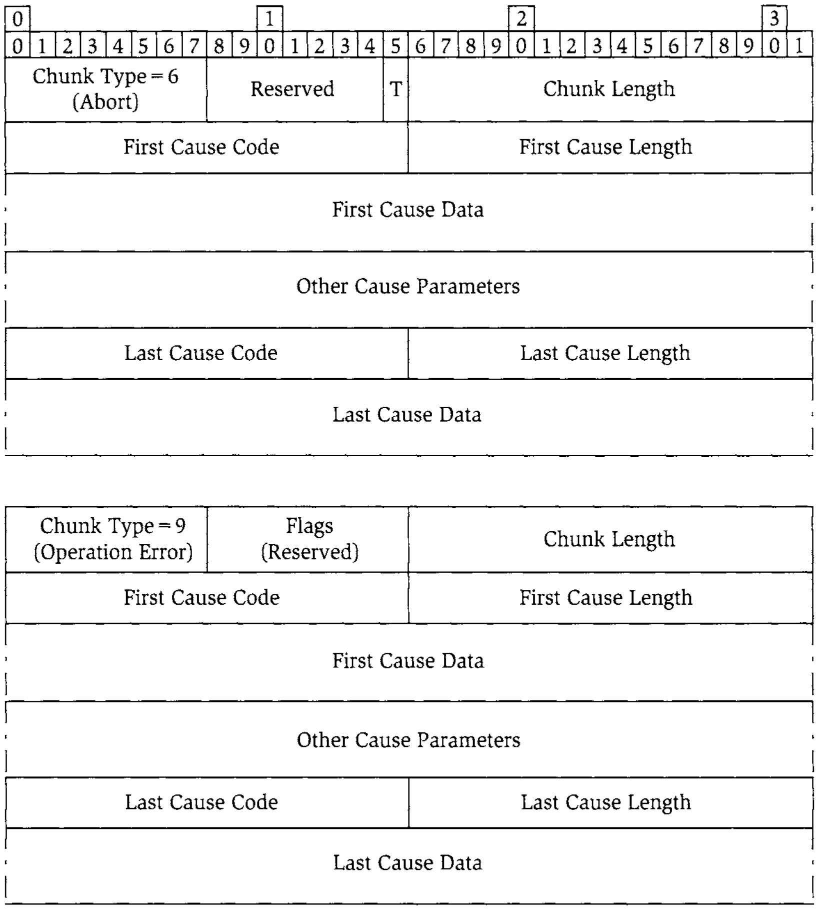

Figure 7.27 The SCTP Abort and Operation Error chunks.

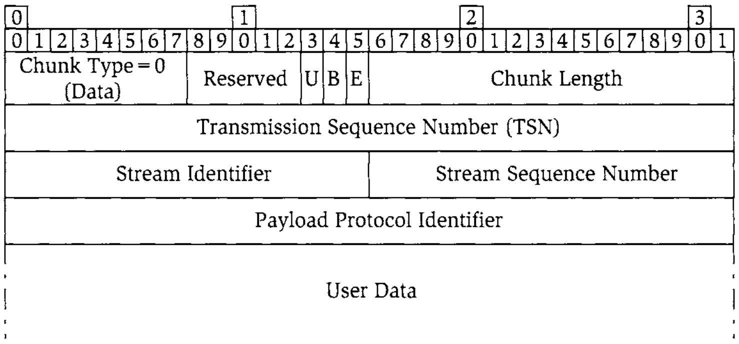

Figure 7.28 The SCTP Data chunk.

Data delivery in SCTP is closely tied to the concept of user data messages. That is, the application delivers data to the SCTP service in discrete lumps (of arbitrary size) and these are transferred by SCTP (which may need to segment them to send them), reassembled, and delivered whole to the application at the remote end. User data messages are given sequence numbers by the application within the context of the stream on which they flow, and SCTP undertakes not only to reassemble the data chunks so that the right data is placed in the right message, but also to deliver the messages in order.

Three SCTP Chunk Flags are used on the Data chunk to help manage this. The B-flag indicates that the data chunk comes from the beginning of a user message. The E-flag indicates that this is the end of a user data message. The B- and E-flags may both be set on a data chunk if the chunk represents the entirety of the user data message. The third flag, the U-flag, indicates that the chunk contains all or part of an unordered data message. Unordered messages do not have valid Stream Sequence Numbers and should be delivered on the stream as soon as they have been reassembled—this makes them behave a little like the receive side of urgent data in TCP.

Figure 7.29 shows how data may be multiplexed from two streams onto a single SCTP association and then demultiplexed for delivery. Note that for simplicity in this example, no acknowledgments of the data are shown. Also for

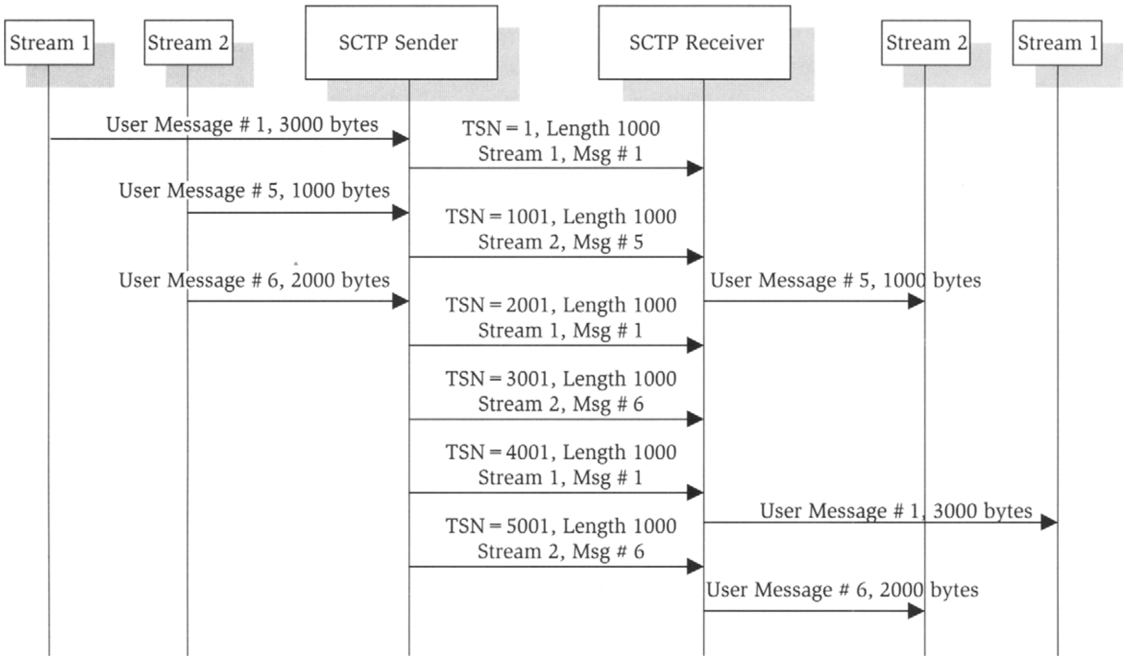

Figure 7.29 Multiplexing and demultiplexing of user data streams on a single SCTP association.

simplicity the SCTP packets shown in Figure 7.29 carry just one Data chunk each—there is no reason, except perhaps for the maximum size of the MTU, for a single packet not to carry multiple Data chunks. If a packet carries more than one Data chunk, each is encoded just as it would be if it were the only chunk in the packet. In particular, the TSN of one chunk follows on from the last byte of the previous chunk.

Acknowledgements in SCTP utilize the understanding of selective acknowledgment gained over the years using TCP. A Selective Acknowledgement chunk in SCTP acknowledges up to a specific TSN (the Cumulative TSN Acknowledgement), indicating that all bytes up to and including that TSN have been successfully received. Note that this is different from TCP, in which the Acknowledgement Number indicates the next expected Sequence Number. It is not necessary to issue multiple Selective Acknowledgement chunks for each stream; a single Selective Acknowledgement chunk can serve the needs of the whole association.

At the same time, the Selective Acknowledgement chunk can indicate blocks of data after the Cumulative TSN Acknowledgement value that have been received, and can acknowledge them selectively. It does this using the perversely named *gap acknowledgement blocks*, which indicate the start and end offsets from the Cumulative TSN Acknowledgement value of each block of received—not missing—data. Consider the sequence of bytes shown in Figure 7.30. All bytes on a shaded background have been successfully received, but those not shaded are missing. In this case, the Selective Acknowledgement chunk should report a Cumulative TSN Acknowledgement of 37 with two gap acknowledgement blocks: {4, 11} and {15, 20}.

The Selective Acknowledgement chunk shown in Figure 7.31 contains the Cumulative TSN as described in the preceding paragraph. It also sets the receiver window just as in TCP, but learning from the lessons of window scaling in TCP, the Receiver Window is a full 32 bits so that scaling is not needed. The next portion of the chunk is used to indicate gap acknowledgement blocks and is preceded by a count of blocks—an implementation that does not support selective

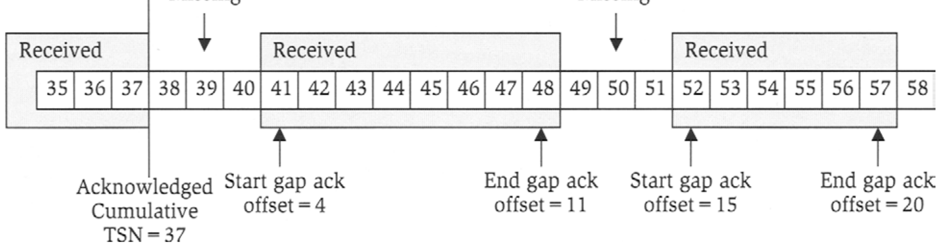

Figure 7.30 SCTP Selective Acknowledgements can acknowledge blocks of data beyond the acknowledged Cumulative Transmission Sequence Number.

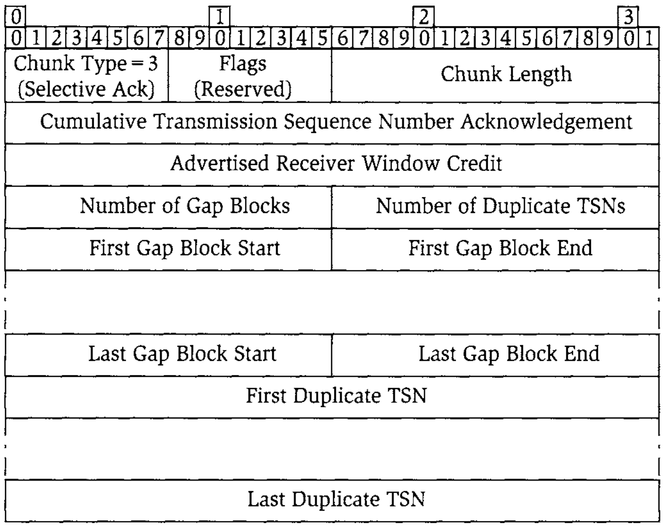

Figure 7.31 The SCTP Selective Acknowledgement chunk.

acknowledgement or which is lucky enough to receive its data in perfect order sets the gap block count to zero and does not supply any start or end offsets.

Also included in the Selective Acknowledgement chunk is a series of duplicate TSNs. Each time a receiver gets a duplicate TSN it lists it in the next Selective Acknowledgement chunk so that the sender knows that its data is being echoed or it is retransmitting too fast. There are a few points to note:

- Only the TSN indicated in the Data chunk is included. It is not the intention that every duplicate byte be listed.
- If a TSN is received multiple times, it should appear in the list multiple times. Only the first receipt is not included.
- Each time a Selective Acknowledgement chunk is sent the counts are reset, and only new duplicates are reported on the next Selective Acknowledgement chunk.
- If no duplicates have been received, the Number of duplicate TSNs is set to zero and no duplicates are included in the chunk.

7.4.4 SCTP Implementation

RFC 2960 does not simply define SCTP as a protocol. It also gives many details on how the protocol should, and in some cases must, be implemented. These details cover the use of timers, retransmission algorithms, flow control and congestion windows, and so forth. The RFC even includes state machines to ensure that there is no mistake in the behavior of an implementation. Lacking the weight of reference material and sample implementations that TCP has, new SCTP implementations should pay close attention to the text in the RFC.

Although the IETF does not normally specify interfaces, it has published a Working Group draft that documents extensions to the sockets API to make the full features of SCTP available to applications in a standardized way. This draft is making its way toward being an RFC, and it describes a mapping of SCTP into a sockets API to provide compatibility for existing TCP applications, access to the new features of SCTP, and a consolidated error and event notification scheme. Implementations of SCTP should aim to provide the sockets interface to make themselves fully useable by existing TCP-based applications.

7.4.5 Choosing Between TCP and SCTP

As yet, SCTP is not a commonly used transport protocol. Perhaps the greatest gating factor to its adoption is simply its lack of availability—TCP is built into most common operating systems as standard, but SCTP is relatively rare. Add to this a natural conservatism among protocol engineers, who would rather stick with the established and proven technology of TCP, whatever its issues, than go out on a limb with a new implementation of a new protocol.

For an existing application to move to using SCTP, both end points must contain an implementation of SCTP and the applications at each end must be migrated to use the new protocol. Given the size of the installed base of applications, this is unlikely to progress quickly. There is no easy way for an application to know whether a remote application with which it wants to communicate supports SCTP, so unless it is specifically configured it will fall back on TCP. The first deployments of SCTP in support of existing applications are likely to be in private networks where it is easier to manage which applications use which transport protocol. It is certainly true that the recent work to enhance the sockets API to allow applications to make use of the features of SCTP will make the process easier.

Nevertheless, SCTP has some distinct advantages and is growing in popularity. New control and application protocols that are developed are free to choose between TCP and SCTP without the weight of history. Since the new protocols require new development and installation, it is less painful for them to also require a new transport protocol, so if the new applications require or can make sensible use of the additional features offered by SCTP they are free to choose it.

7.4.6 Protocols That Use SCTP

IANA lists very few protocols as having registered ports specifically for use with SCTP. However, there is nothing to prevent an application or control protocol that has a registered port number for TCP being successfully run over SCTP. This is increasingly what is happening with some implementations of PSTN protocols such as SIP and MTP2.

7.5 The Real-Time Transport Protocol (RTP)

The Real-Time Transport Protocol (RTP) is a transport protocol in the sense that it provides transport services for its applications for the delivery of end-to-end data across the Internet. On the other hand, as a protocol it is remarkably lightweight, comprising just a single message (the data message) and being so short of features of its own that it must actually run over another transport protocol to achieve the level of function normally expected by an application. RTP should be considered as a top-up transport protocol.

RTP is usually used on top of UDP, although it could actually be run over any other transport protocol from any protocol suite. Since RTP is intended to help manage data delivery for real-time applications (such as video and voice), it is desirable to keep the protocol overheads to a minimum, and UDP is best placed to do that. Of course, RTP could have been designed as a full transport protocol with the features of UDP and capable of standing on its own. There were two reasons not to do this. First, UDP already existed and was readily available—why reinvent or reimplement the same function? Second, to have made RTP a stand-alone and not a top-up protocol would have limited the options for its deployment over other transport protocols and would have lost the ability to build services suitable to different applications.

RTP is accompanied by a management protocol called the Real-Time Transport Control Protocol (RTCP). It should be emphasized that RTCP is not a signaling protocol—it is not used to set up or manage connections, and it is not used to directly control the way data is exchanged by RTP. However, RTCP does allow end points to exchange information about the behavior of data flows between them, and this can be very useful to real-time applications that must act to ensure that the traffic meets the demanding quality of service requirements of voice and video applications. The information exchanged by RTCP may, therefore, be used by RTP applications to change how they present data to RTP and how it is sent over the network.

Since RTP runs over other transport protocols it does not have a registered IP protocol identifier, but it does have registered server port numbers (5004 for RTP and 5005 for RTCP). Although RTP and RTCP are really client–server protocols, these registered port numbers give all nodes ports on which to passively listen for traffic. Source port numbers are taken from the dynamic ports range and must be allocated with RTP using an even port number and RTCP using a port number one greater in value.

The International Telecommunications Union (ITU) standard H.323 mandates the use of RTP, and it is used in products such as Microsoft's NetMeeting.

7.5.1 Managing Data

The primary purpose of RTP is to monitor, and, hence, maintain, the quality of data traffic for real-time applications. The requirements vary somewhat according

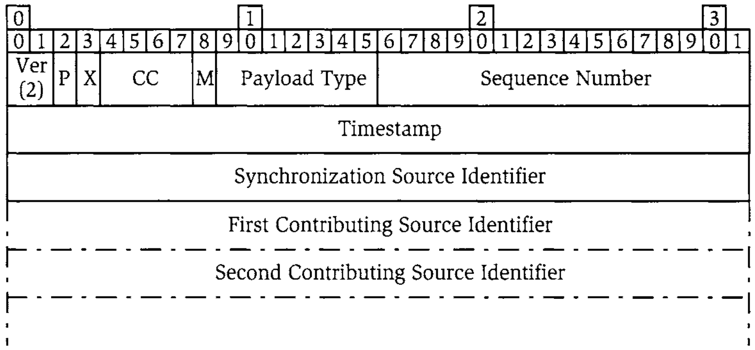

Figure 7.32 The RTP header is at least 12-bytes long.

to application but are not limited simply to timely delivery of in-order, uncorrupted data. It is also important to manage the rate of delivery of data to avoid *jitter*, the distortions of media streams caused by data arriving in bursts with gaps in between.

For this reason, the most important field in the RTP header shown in Figure 7.32 is the timestamp field. The timestamp is a 32-bit integer and contains the timestamp of the generation of the first byte of payload data. The stamp itself is in local format and does not need to be understood by the remote end point. It must, however, be monotonically increasing (that is, the timestamp on one packet is always the same as or greater than that on the previous packet—within the scope of wrapping the 32-bit integer) and must provide sufficient granularity for the application to determine and resolve jitter issues. For example, audio applications would probably implement a clock that incremented by one for each audio sampling period.

The other fields in the header are as follows. A 2-bit field indicates the version number of the protocol. Currently, version two is in use. This is followed by a single bit that indicates whether the packet concludes with 1 or more bytes of padding that are included in the packet length but do not form part of the packet data. If this is the case, the last byte of the packet is a count of the number of bytes of padding (that is, preceding and including the count) that should be ignored. It may be necessary to pad an RTP packet up to a particular block size if multiple packets are concatenated in a single lower-layer protocol packet, or if the data must be presented in well-known units to an encryption algorithm.

The next bit is the X-flag and indicates whether the header contains any private extensions (see below). The Contributing Source Identifier Count (CC) is a count of the number of Contributing Source Identifiers appended to the header and is important in defining the length of the header.

The use of the M-flag is dependent on the value of the Payload Type that follows it. It is most commonly used to help manage the data by providing synchronization events such as end of frame markers. This can allow the receiving application to reset itself without needing to parse the payload.

The Payload Type defines the use to which the data is put, and so controls how the RTP header is used and what RTCP messages should be sent. The current list of registered Payload Types is shown in Table 7.6. Since the M-flag and the Payload Type field of RTP overlap with the Message Type field of RTCP, the Payload Type is chosen from a distinct set to make processing simpler. RTCP message types run from 200 to 204, so RTP message types 72 through 76 are reserved. Most of the older payloads are discussed in RFC 1890.

The Sequence Number is simply a 16-bit packet count that increments for each packet sent and wraps back to zero. This is used to help detect packet loss. The first number used should be chosen at random to help protect against clashes with previous uses of the same ports.

The Synchronization Source Identifier (SSRC) is a random 32-bit identifier that identifies the source of the data within an RTP session. Since multiple nodes may participate in an RTP session (two-way traffic, multicast distribution, and so forth) it is necessary that the SSRC be unique across the session to guarantee identification of individual sources. Random generation of SSRCs is almost, but not quite, sufficient to make this guarantee, and there remains a slight possibility of two nodes picking the same values. SSRC contention is resolved by both sources dropping out of the session and picking new random SSRC values. This process also helps detect data that is sent in a loop.

Contributing Source Identifiers (CSRCs) identify multiple data sources that have been combined to form a single stream. This may be useful when separate applications generate data for the same RTP session (for example, audio and video) and the multiple streams are merged into a single stream. In such cases, the SSRC identifies the *mixer*, that is the application that merges the streams, and the CSRCs identify the individual source applications.

It is also possible for mixers to merge data streams from disjoint nodes. For example, in an audio conference, the mixer acts as a clearing house for all source voice streams, merges them, and sends them out to all of the listeners. The SSRC identifies the mixer and the CSRC identifies the speaker.

RTP headers may be extended to carry information specific to the payload. The presence of a header extension is indicated by setting the X-flag to 1. The format of the extension is dependent on the payload type, but the extension is itself identified by a type (to allow multiple different extensions for a given payload type). An Extension Length field indicates the length of the extension in 32-bit words, not including the identifier or length fields. This is illustrated in Figure 7.33.

Table 7.6 Registered RTP Payload Types

Payload Type	*Name*	*Clock Rate (Hz)*	*Usage*
0	PCMU	8000	Audio
1	1016	8000	Audio
2	G721	8000	Audio
3	GSM	8000	Audio
4	G723	8000	Audio
5	DVI4	8000	Audio
6	DVI4	16000	Audio
7	LPC	8000	Audio
8	PCMA	8000	Audio
9	G722	8000	Audio
10	L16	44100	Audio
11	L16 (2 channel)	44100	Audio
12	QCELP	8000	Audio
13	CN	8000	Audio
14	MPA	90000	Audio
15	G728	8000	Audio
16	DVI4	11025	Audio
17	DVI4	22050	Audio
18	G729	8000	Audio
25	CellB	90000	Video
26	JPEG	90000	Video
28	nv	90000	Video
31	H261	90000	Video
32	MPV	90000	Video
33	MP2T	90000	Audio/Video
34	H263	90000	Video
Dynamic	GSM-HR	8000	Audio
Dynamic	GSM-EFR	8000	Audio
Dynamic	L8	variable	Audio
Dynamic	RED	variable	Audio

Table 7.6 *Continued*

Payload Type	*Name*	*Clock Rate (Hz)*	*Usage*
Dynamic	VDVI	variable	Audio
Dynamic	BT656	90000	Video
Dynamic	H263-1998	90000	Video
Dynamic	MPIS	90000	Video
Dynamic	MP2P	90000	Video
Dynamic	BMPEG	90000	Video

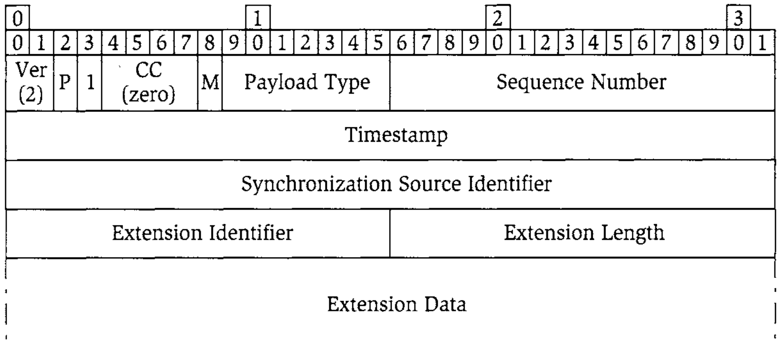

Figure 7.33 An RTP header with no Contributing Source Identifiers, but with a header extension.

7.5.2 Control Considerations

The Real-Time Control Protocol (RTCP) has three purposes.

- It allows participants in an RTP session to register their presence and to leave the session gracefully.
- It is used to monitor RTP data traffic and to feed back information about the quality of service being delivered.
- It can carry application-specific information.

There are five RTCP packet types, which are described in the following paragraphs.

The Sender Descriptor (SDES) packet is sent by an application when it joins an RTP session. The SDES gives the SSRC of the application and supplies additional information such as the host name or the geographical address of the node in Source Description Items. Figure 7.34 shows how the SDES packet is

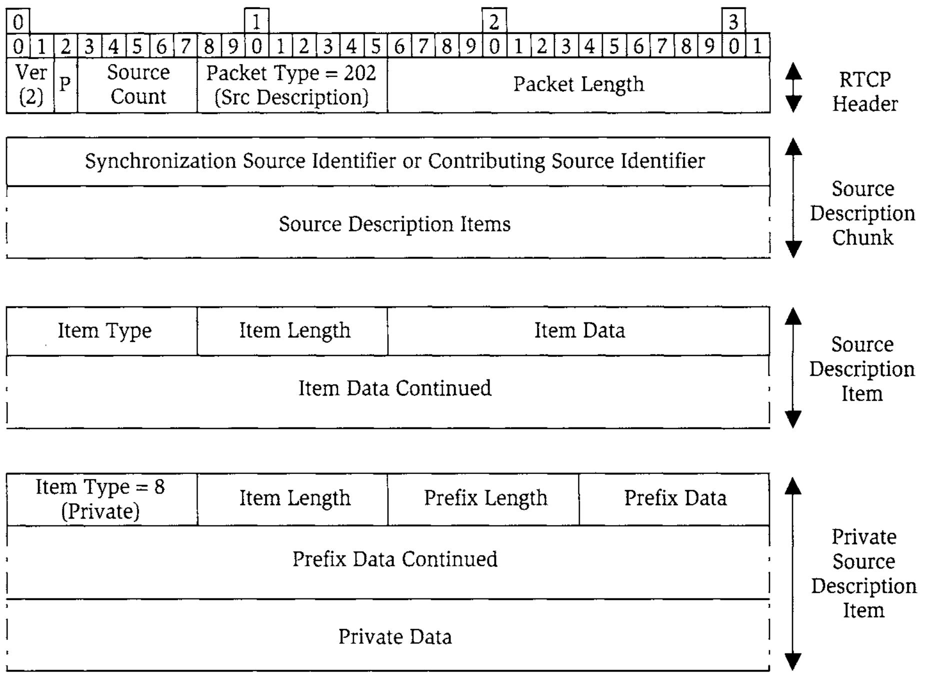

Figure 7.34 The RTCP Source Description packet has a smaller common header and at least one Source Description chunk built up from a sequence of source description items.

built of a common RTCP header followed by a series of Source Description Chunks. The header includes a version number (two is the current version), the P-flag to indicate whether the packet is terminated with padding, a count of the number of Source Description Chunks present, a packet type field, and a packet length, which includes the header fields and is represented as a count of 32-bit words in the packet, *minus one*. This peculiarity in the length field is intended to protect against scanning buffers built up of concatenated RTP packets where there is some corruption and the length field contains zero—in practice, it means that the length field counts the number of 32-bit words that follow the length field.

The Source Description Chunk describes an individual source. It begins with the SSRC of the participant in the session and contains one or more Source Description items. Each Source Description Item has an item type, an item length and item data. Table 7.7 lists the defined item types and gives their purpose. The length of each chunk is given in bytes and does not include the item type or item length fields. Individual items are not padded to reach any special byte boundaries, and text strings are not null terminated. However, each chunk begins on a 4-byte boundary, so there may be null padding at the end of

Table 7.7 The RTCP Source Description Items

Item	*Contents*
0	End of list marker. Length should be zero and data is ignored
1	Persistent transport name (canonical name) of the form "user@host" or "host" where host is either the fully qualified domain name of the host or is the IP address of the host on one of its interfaces presented in dotted notation
2	User name
3	User's email address
4	User's phone number
5	User's geographical location (address)
6	Application name
7	Free-form notes about the source
8	Additional private data. As shown in Figure 7.34, this is comprised of prefix and private data

a chunk. Since the Source Description Chunk does not include a length field or a count of the number of items in the chunk, the chunk is ended with a special item of type zero with a length field of zero.

A mixer, that is a node that merges RTP streams from multiple sources, sends an SDES packet for itself and includes Sender Descriptor Chunks for each of its contributing participants. If there are more than 31 chunks (governed by the size of the Source Count field) the mixer simply sends multiple SDES packets. Similarly, if a new participant joins the session through the mixer, the mixer just sends another SDES.

When an application leaves the session, it sends a BYE packet if it is well behaved. This lets other participants know that it has gone and allows them to free up resources associated with the participant. Similarly, if an application notices a clash in SSRC values between its own SSRC and that of another participant, it sends a BYE packet and immediately selects a new SSRC value and sends a new SDES.

The BYE packet, shown in Figure 7.35, also begins with the standard RTCP header. Like the SDES, the BYE allows for multiple participants to be referenced in one packet; this is useful if a mixer leaves the session. Additionally, the BYE packet includes information encoded as a printable text string about why the participants have left. Only one reason is allowed for all participants identified on a single BYE packet. If no reason is included, the reason length is set to zero.

Traffic monitoring is achieved using the Sender Report (SR) and Receiver Report (RR), shown in Figures 7.36 and 7.37. Periodically, every source of data on the RTP session sends a Sender Report to show the number of bytes and

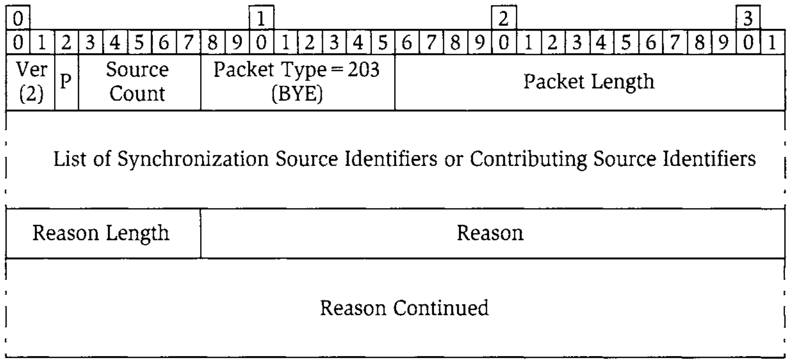

Figure 7.35 The RTCP BYE packet can report on multiple sources leaving the session.

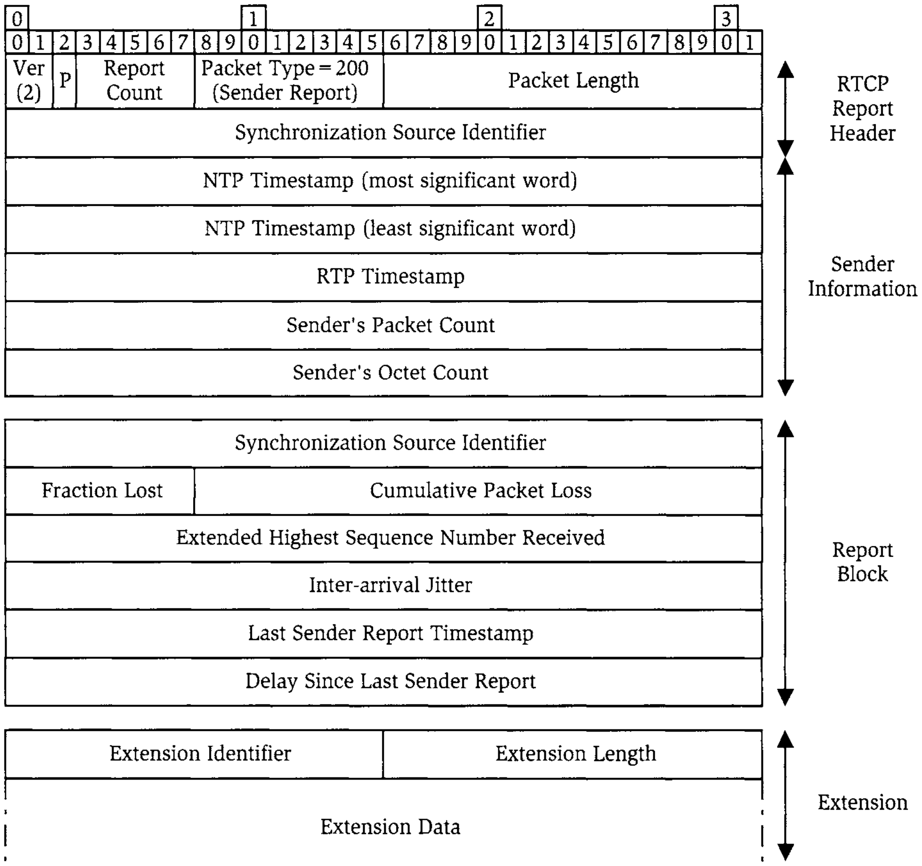

Figure 7.36 The RTCP Sender Report packet has a common header, a mandatory piece of sender information, and at least one report block. It may be followed by a profile-specific extension.

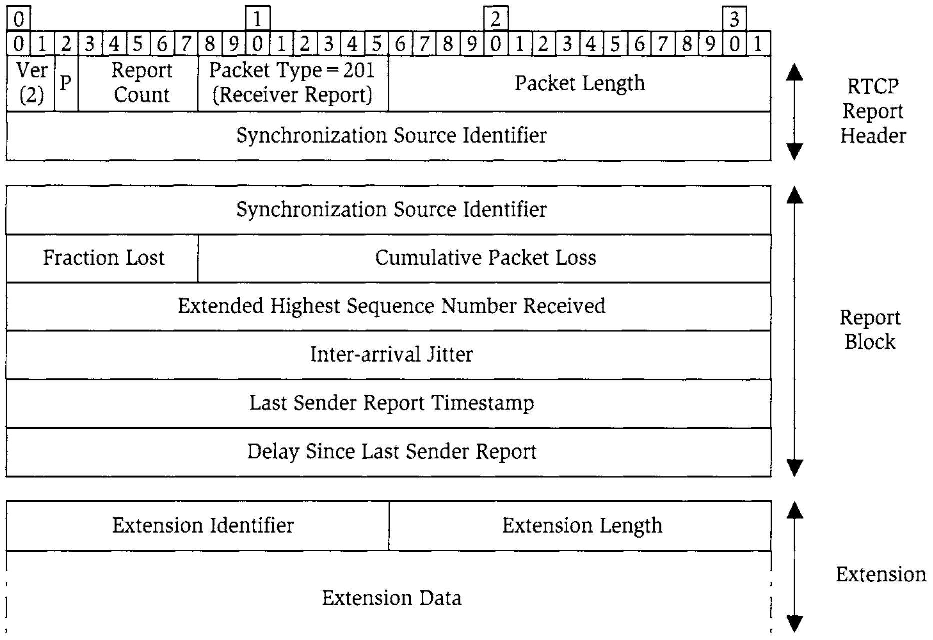

Figure 7.37 The RTCP Receiver Report packet has a common header and at least one report block. It may be followed by a profile-specific extension. It does not carry any sender information.

packets it has sent. It includes an NTP timestamp to correlate the SR across the whole network, and also shows the RTP time at which the packet was generated (using the same clock that is used to generate the timestamps on the RTP packets).

Each SR also includes a Report block for each participant (again allowing for multiple participants on a single SR if the sending node is a mixer). The Report Block quantifies the quality of data received by the sender—which makes sense in the case of bidirectional traffic.

When it receives an SR, a participant responds with a Receiver Report. The RR is similar to an SR in that it contains an SSRC to identify the receiver and a Report Block to describe the received data. The Sender Information is, however, omitted.

Both SRs and RRs may include extensions with interpretation left up to the applications.

A final RTCP packet is defined to allow applications to use RTCP to transfer their own control information. The Application packet shown in Figure 7.38 begins with a standard RTCP header and the SSRC of the sender. This is followed by a 4-byte field that identifies the packet usage in printable text, and application data that is interpreted according to the application.

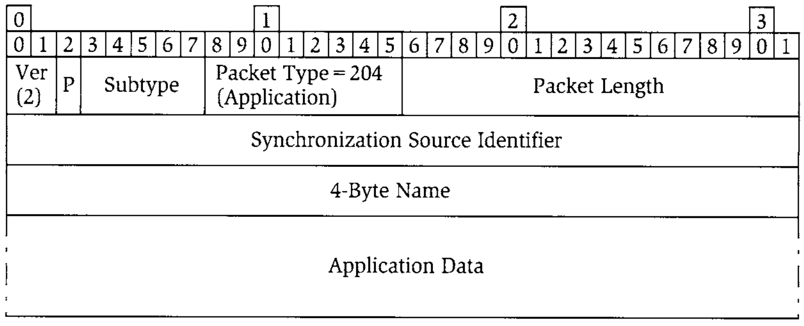

Figure 7.38 RTCP defines the Application packet for private use by an application.

7.5.3 Choosing a Transport for RTP

As mentioned before, RTP is a supplementary transport protocol that needs to run over some other transport protocol to provide the correct level of integrity and delivery. Any transport protocol would do, but in the context of the Internet the choice is between UDP, TCP, and SCTP.

RTP traffic does not require 100 percent reliability. In fact, it is acceptable to lose the odd voice sample and it is only when losses get reasonably high that the listener will notice the degradation. Furthermore, a protocol that attempts 100 percent packet delivery will back off while doing retires—this would be wholly unacceptable to a real-time application. Other back-off mechanisms for flow control and pacing are also ill-suited to real-time applications.

All this makes TCP and SCTP pretty poor choices to underlie RTP. Add to these issues the signaling overhead of TCP and SCTP and the additional cost of maintaining connections and there is really no choice but to use the lighter-weight choice of UDP.

If any further evidence was needed, it should be recalled that RTP is capable of being used by multicast applications. TCP would require a full mesh of end-to-end connections, but UDP is capable of working with IP multicast (see Chapter 3) to address these needs.

7.5.4 Choosing to Use RTP

RTP offers real-time applications ways of monitoring quality of service so that they can take action to deliver the level of function their users need.

There are, however, some issues with RTP and RTCP. The principle concern with RTP is the amount of control information that is transmitted in each data packet. This is illustrated in Figure 7.39, which shows the best case where a total of 40 bytes of overhead are sent before each piece of data. If the data is made up of audio samples, which are typically 16 bytes each, this means that more than 70 percent of the bandwidth is being used for control information.

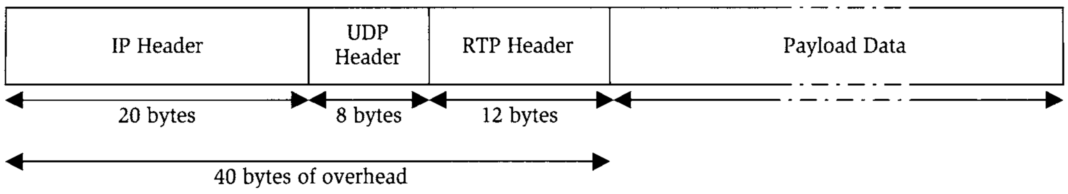

Figure 7.39 RTP over UDP and IP imposes at least a 40-byte overhead in IPv4.

But the worst case is far worse. The IP header might have a few extra bytes for IP options, and the RTP header might include one or more CSRCs. Then security might be adding to the header size, and the IP addresses might use IPv6. Further, there may be considerable overhead from the lower-layer network protocols such as ATM or Ethernet—in fact, the overhead can be so large that several ATM cells are needed to send the header information before any data is actually sent. All this can add up to very poor use of the network bandwidth and, since the whole point of RTP is to enhance the quality of traffic for real-time applications, it is reasonable to wonder whether things might not be better if RTP was abandoned and some other method was used.

The features of RTP (together with RTCP) are still sufficiently important that RTP is used extensively. Header compression techniques are used to reduce the amount of IP, UDP, and RTP information sent in each packet, thus improving the available bandwidth for data. See Chapter 15 for a discussion of header compression.

RTP scales well as the number of participants in the session increases, but RTCP traffic increases exponentially as each participant exchanges control and monitoring information with all other participants. Obviously, this can affect the bandwidth available for data throughput.

To get around this issue, the Sender Reports can be sent out less frequently and the responding Receiver Reports can be delayed a small amount. These delays can be randomized to avoid bunching of traffic, and can be factored according to the number of participants in the session.

More sophisticated schemes can favor Sender Reports from active senders (that is, those that are currently sending RTP packets), giving them immediate Receiver Reports, and delay the Receiver Reports from silent participants. Similarly, silent participants can send Sender Reports less frequently.

7.6 Further Reading

There are numerous books that cover some or all of the common IP transport protocols at a variety of levels ranging from brief overview to thorough in-depth analysis of an individual implementation.

Internetworking with TCP/IP, Vol. 1, by Douglas Comer (1996). Prentice-Hall. This is often considered the ultimate reference for TCP/IP.

TCP/IP Illustrated, Vol. 1, by Richard Stevens (1994), and *TCP/IP Illustrated, Vol. 2*, by Gary Wright and Richard Stevens (1995). Addison-Wesley. These volumes give, respectively a, thorough explanation of TCP and how to implement it, and a detailed, line-by-line explanation of a sample implementation.

TCP/IP Clearly Explained, by Peter Loshin (1999). Academic Press. This has useful chapters on TCP and UDP.

The Design and Implementation of the 4.3 BSD UNIX Operating System, by Leffler, McKusick, Karels, and Quarterman (1989). Addison-Wesley. This gives a useful overview of UDP, TCP, and sockets within the context of an operating system implementation.

Stream Control Transmission Protocol (SCTP): A Reference Guide, by Stewart, Xie, and Allman (2001). Addison-Wesley. The definitive guide to SCTP, written by two of the principal designers of the protocol.

IP Telephony, by Bill Douskalis (2000). Prentice-Hall. This provides an overview of RTP and RTCP together with plenty of technical details about how the protocols can be used to carry voice traffic in the Internet.

The IETF has published multiple RFCs covering the material in this chapter. They can be found through the IETF's web site at www.ietf.org. New work on IP transport protocols is split between two key Working Groups: the Transport Area Working Group (http://www.ietf.org/html.charters/tsvwg-charter.html) and the Signaling Transport Working Group (http://www.ietf.org/html.charters/sigtran-charter.html). Some key RFCs are:

UDP

RFC 768—User Datagram Protocol

TCP

RFC 793—Transmission Control Protocol
RFC 1122—Requirements for Internet Hosts
RFC 1323—TCP Extensions for High Performance
RFC 2018—TCP Selective Acknowledgement Options
RFC 2414—Increasing TCP's Initial Window
RFC 2525—Known TCP Implementation Problems
RFC 2581—TCP Congestion Control

SCTP

RFC 1950—ZLIB Compressed Data Format Specification version 3.3 (contains the definition of the Adler checksum algorithm)
RFC 2960—Stream Control Transmission Protocol
RFC 3257—SCTP Applicability Statement

RTP

RFC 1889—RTP: A Transport Protocol for Real-Time Applications
RFC 1890—RTP Profile for Audio and Video Conferences with Minimal Control