Network Working Group
Request for Comments: 1693
Category: Experimental

T. Connolly
P. Amer
P. Conrad
University of Delaware
November 1994

An Extension to TCP : Partial Order Service

Status of This Memo

This memo defines an Experimental Protocol for the Internet community. This memo does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited

IESG Note:

Note that the work contained in this memo does not describe an Internet standard. The Transport AD and Transport Directorate do not recommend the implementation of the TCP modifications described. However, outside the context of TCP, we find that the memo offers a useful analysis of how misordered and incomplete data may be handled. See, for example, the discussion of Application Layer Framing by D. Clark and D. Tennenhouse in, "Architectural Considerations for a New Generation of Protocols", SIGCOM 90 Proceedings, ACM, September 1990.

Abstract

This RFC introduces a new transport mechanism for TCP based upon partial ordering. The aim is to present the concepts of partial ordering and promote discussions on its usefulness in network communications. Distribution of this memo is unlimited.

Introduction

A service which allows partial order delivery and partial reliability is one which requires some, but not all objects to be received in the order transmitted while also allowing objects to be transmitted unreliably (i.e., some may be lost).

The realization of such a service requires, (1) communication and/or negotiation of what constitutes a valid ordering and/or loss-level, and (2) an algorithm which enables the receiver to ascertain the deliverability of objects as they arrive. These issues are addressed here - both conceptually and formally - summarizing the results of research and initial implementation efforts.

The authors envision the use of a partial order service within a connection-oriented, transport protocol such as TCP providing a further level of granularity to the transport user in terms of the type and quality of offered service. This RFC focuses specifically on extending TCP to provide partial order connections.

The idea of a partial order service is not limited to TCP. It may be considered a useful option for any transport protocol and we encourage researchers and practitioners to investigate further the most effective uses for partial ordering whether in a next-generation TCP, or another general purpose protocol such as XTP, or perhaps within a "special purpose" protocol tailored to a specific application and network profile.

Finally, while the crux of this RFC relates to and introduces a new way of considering object ordering, a number of other classic transport mechanisms are also seen in a new light - among these are reliability, window management and data acknowledgments.

Keywords: partial order, quality of service, reliability, multimedia, client/server database, Windows, transport protocol

Table of Contents

1. Introduction and motivation

Current applications that need to communicate objects (i.e., octets, packets, frames, protocol data units) usually choose between a fully ordered service such as that currently provided by TCP and one that does not guarantee any ordering such as that provided by UDP. A similar "all-or-nothing" choice is made for object reliability - reliable connections which guarantee all objects will be delivered verses unreliable data transport which makes no guarantee. What is more appropriate for some applications is a partial order and/or partial reliability service where a subset of objects being communicated must arrive in the order transmitted, yet some objects may arrive in a different order, and some (well specified subset) of the objects may not arrive at all.

One motivating application for a partial order service is the emerging area of multimedia communications. Multimedia traffic is often characterized either by periodic, synchronized parallel streams of information (e.g., combined audio-video), or by structured image streams (e.g., displays of multiple overlapping and nonoverlapping windows). These applications have a high degree of tolerance for less-than-fully-ordered data transport as well as data loss. Thus they are ideal candidates for using a partial order, partial reliability service. In general, any application which communicates parallel and/or independent data structures may potentially be able to profit from a partial order service.

A second application that could benefit from a partial order service involves remote or distributed databases. Imagine the case where a database user transmitting queries to a remote server expects objects (or records) to be returned in some order, although not necessarily total order. For example a user writing an SQL data query might specify this with the "order by" clause. There exist today a great number of commercial implementations of distributed databases which utilize - and thus are penalized by - an ordered delivery service.

Currently these applications must use and pay for a fully ordered/fully reliable service even though they do not need it. The introduction of partial services allows applications to lower the demanded quality of service (QOS) of the communication assuming that such a service is more efficient and less costly. In effect, a partial order extends the service level from two extremes - ordered and unordered - to a range of discreet values encompassing both of the extremes and all possible partial orderings in between. A similar phenomenon is demonstrated in the area of reliability.

It is worth mentioning that a TCP implementation providing a partial order service, as described here, would be able to communicate with a non-partial order implementation simply by recognizing this fact at connection establishment - hence this extension is backward compatible with earlier versions of TCP. Furthermore, it is conceivable for a host to support the sending-half (or receiving-half) of a partial order connection alone to reduce the size of the TCP as well as the effort involved in the implementation. Similar "levels of conformance" have been proposed in other internet extensions such as [Dee89] involving IP multicasting.

This RFC proceeds as follows. The principles of partial order delivery, published in [ACCD93a], are presented in Section 2. The notion of partial reliability, published in [ACCD93b], is introduced in Section 3 followed by an explanation of "reliability classes". Then, the practical issues involved with setting up and maintaining a Partial Order Connection (POC) within a TCP framework are addressed in Section 4 looking first at connection establishment, and then discussing the sender's role and the receiver's role. Section 5 provides insights into the expected performance improvements of a partial order service over an ordered service and Section 6 discusses some future directions. Concluding remarks are given in Section 7.

2. Partial Order Delivery

Partial order services are needed and can be employed as soon as a complete ordering is not mandatory. When two objects can be delivered in either order, there is no need to use an ordered service that must delay delivery of the second one transmitted until the first arrives as the following examples demonstrate.

2.1 Example 1: Remote Database

Simpson's Sporting Goods (SSG) has recently installed a state-of-the-art enterprise-wide network. Their first "network application" is a client/server SQL database with the following four records, numbered {1 2 3 4} for convenience:

	SALESPERSON	LOCATION	CHARGES	DESCRIPTION
1	Anderson	Atlanta, GA	$4,200	Camping Gear
2	Baker	Boston, MA	$849	Camping Gear
3	Crowell	Boston, MA	$9,500	Sportswear
4	Dykstra	Wash., DC	$1,000	Sportswear

SSG employees running the client-side of the application can query the database server from any location in the enterprise net using standard SQL commands and the results will be displayed on their

screen. From the employee's perspective, the network is completely reliable and delivers data (records) in an order that conforms to their SQL request. In reality though, it is the transport layer protocol which provides the reliability and order on top of an unreliable network layer - one which introduces loss, duplication, and disorder.

Consider the four cases in Figure 1 - in the first query (1.a), ordered by SALESPERSON, the records have only one acceptable order at the destination, 1,2,3,4. This is evident due to the fact that there are four distinct salespersons. If record 2 is received before record 1 due to a network loss during transmission, the transport service can not deliver it and must therefore buffer it until record 1 arrives. An ordered service, also referred to as a virtual circuit or FIFO channel, provides the desired level of service in this case.

At the other extreme, an unordered service is motivated in Figure 1.d where the employee has implicitly specified that any ordering is valid simply by omitting the "order by" clause. Here any of 4! = 24 delivery orderings would satisfy the application, or from the transport layer's point of view, all records are immediately deliverable as soon as they arrive from the network. No record needs to buffered should it arrive out of sequential order. As notation, 4 ordered objects are written 1;2;3;4 and 4 unordered objects are written using a parallel operator: 1||2||3||4.

Figures 1.b and 1.c demonstrate two possible partial orders that permit 2 and 4 orderings respectively at the destination. Using the notation just described, the valid orderings for the query in 1.b are specified as 1;(2||3);4, which is to say that record 1 must be delivered first followed by record 2 and 3 in either order followed by record 4. Likewise, the ordering for 1.c is (1||2);(3||4). In these two cases, an ordered service is too strict and an unordered service is not strict enough.

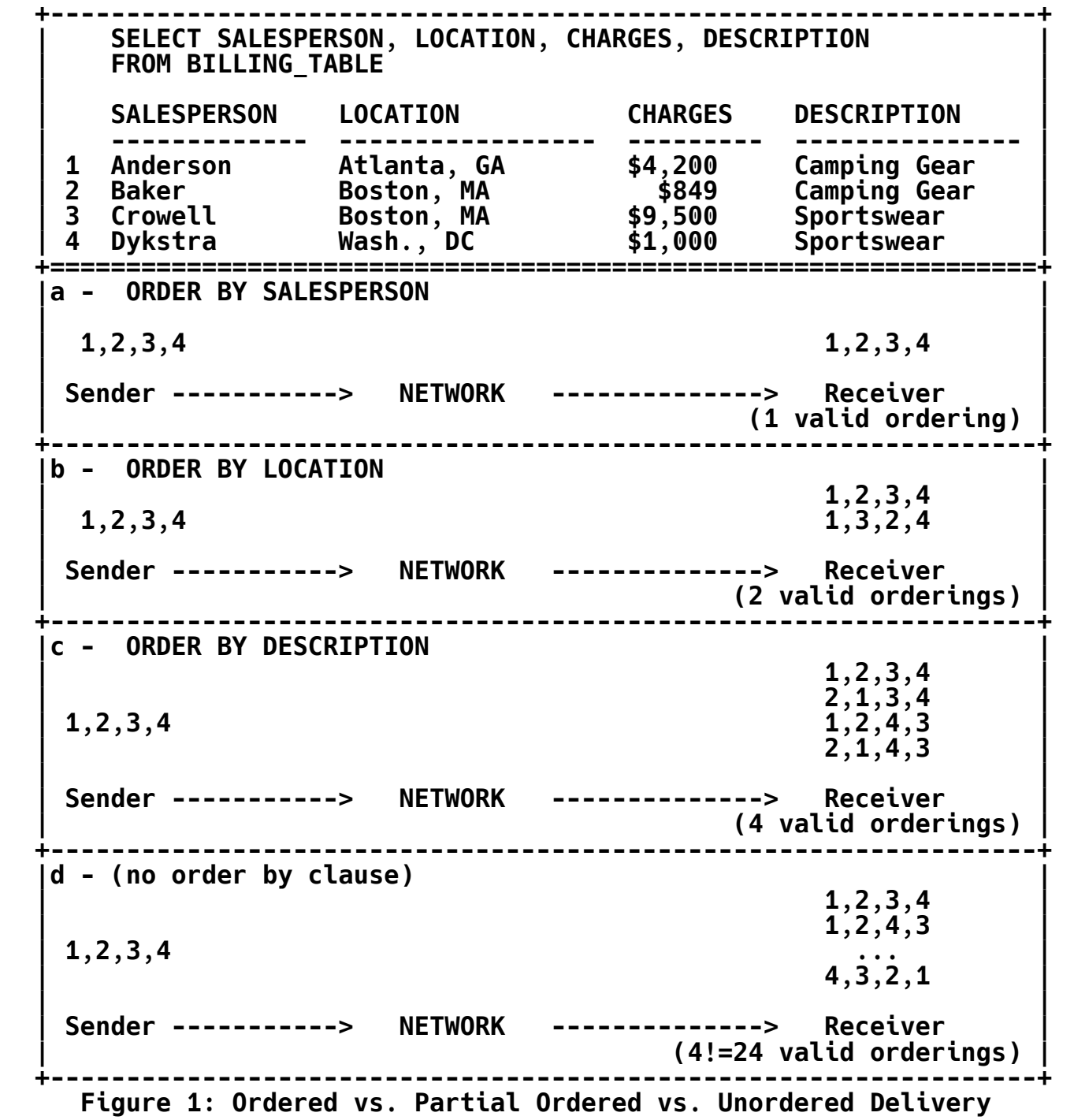


Figure 1: Ordered vs. Partial Ordered vs. Unordered Delivery

It is vital for the transport layer to recognize the exact requirements of the application and to ensure that these are met. However, there is no inherent need to exceed these requirements; on

the contrary, by exceeding these requirements unecessary resources are consumed. This example application requires a reliable connection - all records must eventually be delivered - but has some flexibility when it comes to record ordering.

In this example, each query has a different partial order. In total, there exist 16 different partial orders for the desired 4 records. For an arbitrary number of objects N, there exist many possible partial orders each of which accepts some number of valid orderings between 1 and N! (which correspond to the ordered and unordered cases respectively). For some classes of partial orders, the number of valid orderings can be calculated easily, for others this calculation is intractable. An in-depth discussion on calculating and comparing the number of orderings for a given partial order can be found in [ACCD93a].

2.2 Example 2: Multimedia

A second example application that motivates a partial order service is a multimedia broadcast involving video, audio and text components. Consider an extended presentation of the evening news - extended to include two distinct audio channels, a text subtitle and a closed-captioned sign language video for the hearing impaired, in addition to the normal video signal, as modeled by the following diagram.

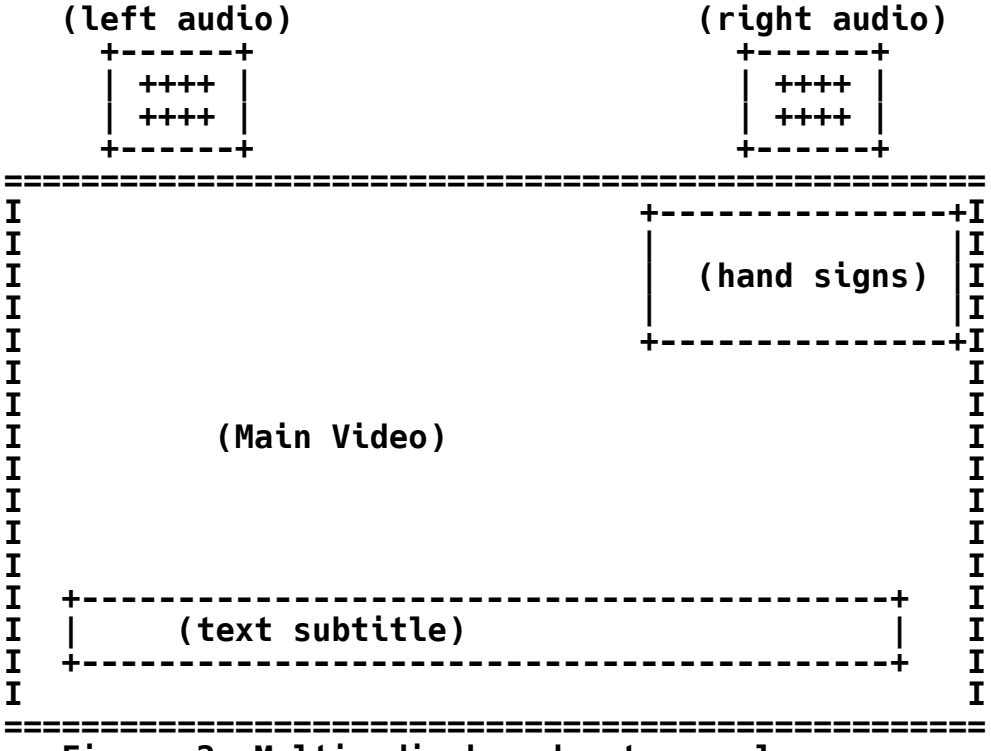


Figure 2: Multimedia broadcast example

The multimedia signals have differing characteristics. The main video signal may consist of full image graphics at a rate of 30 images/sec while the video of hand signs requires a lower quality, say 10 images/sec. Assume the audio signals are each divided into 60 sound fragments/sec and the text object each second consists of either (1) new text, (2) a command to keep the previous second of text, or (3) a command for no subtitle.

During a one-second interval of the broadcast, a sender transmits 30 full-motion video images, 10 closed-captioned hand sign images, 60 packets of a digitized audio signal for each of the audio streams and a single text packet. The following diagram then might represent the characteristics of the multimedia presentation in terms of the media types, the number of each, and their ordering. Objects connected by a

horizontal line must be received in order, while those in parallel have no inherent ordering requirement.

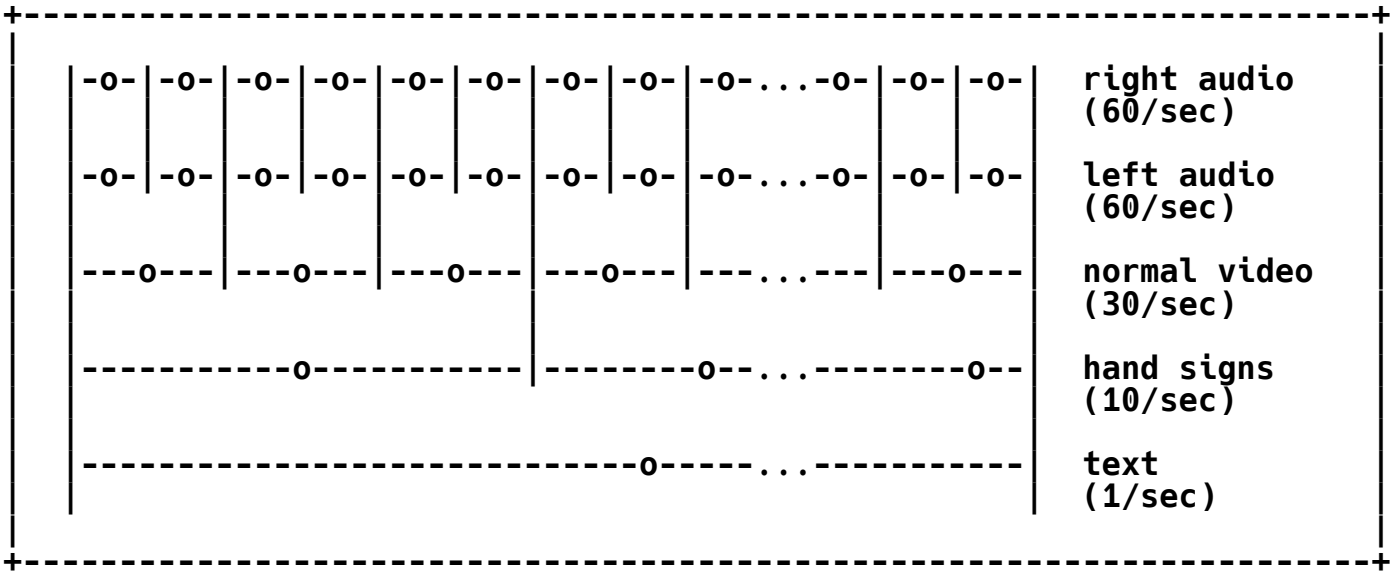


Figure 3: Object ordering in multimedia application

Of particular interest to our discussion of partial ordering is the fact that, while objects of a given media type generally must be received in order, there exists flexibility between the separate "streams" of multimedia data (where a "stream" represents the sequence of objects for a specific media type). Another significant characteristic of this example is the repeating nature of the object orderings. Figure 3 represents a single, one-second, partial order snapshot in a stream of possibly thousands of repeating sequential periods of communication.

It is assumed that further synchronization concerns in presenting the objects are addressed by a service provided on top of the proposed partial order service. Temporal ordering for synchronized playback is considered, for example, in [AH91, HKN91].

2.3 Example 3: Windows Screen Refresh

A third example to motivate a partial order service involves refreshing a workstation screen/display containing multiple windows from a remote source. In this case, objects (icons, still or video images) that do not overlap have a "parallel" relationship (i.e., their order of refreshing is independent) while overlapping screen objects have a "sequential" relationship and should be delivered in order. Therefore, the way in which the windows overlap induces a partial order.

Consider the two cases in Figure 4. A sender wishes to refresh a remote display that contains four active windows (objects) named {1 2 3 4}. Assume the windows are transmitted in numerical order and the receiving application refreshes windows as soon as the transport service delivers them. If the windows are configured as in Figure 4a, then there exist two different orderings for redisplay, namely 1,2,3,4 or 1,3,2,4. If window 2 is received before window 1, the transport service cannot deliver it or an incorrect image will be displayed. In Figure 4b, the structure of the windows results in six possible orderings - 1,2,3,4 or 1,3,2,4 or 1,3,4,2 or 3,4,1,2 or 3,1,4,2 or 3,1,2,4.

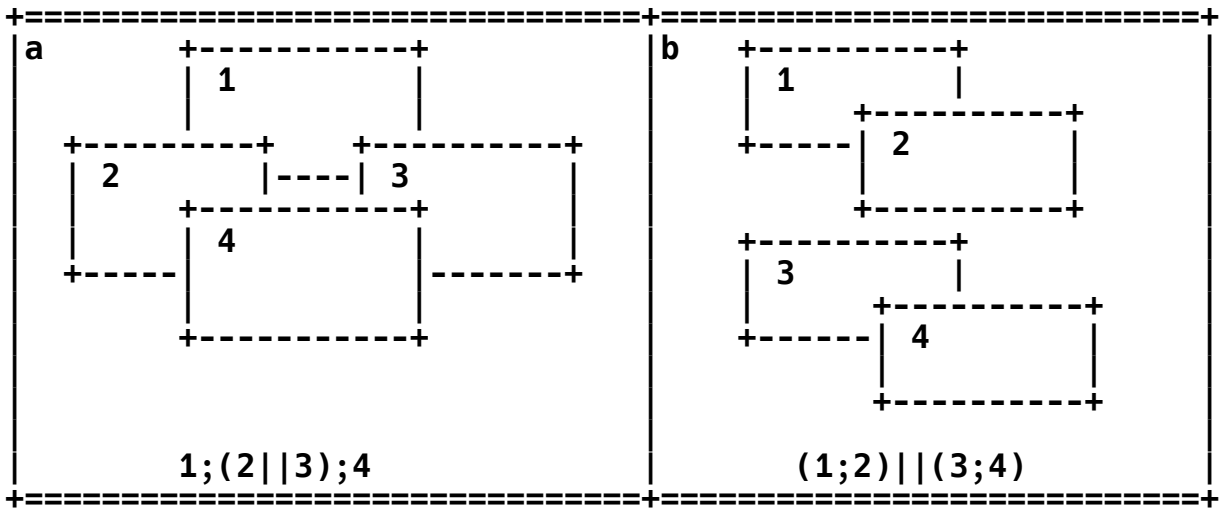


Figure 4: Window screen refresh

2.4 Potential Savings

In each of these examples, the valid orderings are strictly dependent upon, and must be specified by the application. Intuitively, as the number of acceptable orderings increases, the amount of resources utilized by a partial order transport service, in terms of buffers and retransmissions, should decrease as compared to a fully ordered transport service thus also decreasing the overall cost of the connection. Just how much lower will depend largely upon the flexibility of the application and the quality of the underlying network.

As an indication of the potential for improved service, let us briefly look at the case where a database has the following 14 records.

	SALESPERSON	LOCATION	CHARGES	DESCRIPTION
1	Anderson	Washington	$4,200	Camping Gear
2	Anderson	Philadelphia	$2,000	Golf Equipment
3	Anderson	Boston	$450	Bowling shoes
4	Baker	Boston	$849	Sportswear
5	Baker	Washington	$3,100	Weights
6	Baker	Washington	$2000	Camping Gear
7	Baker	Atlanta	$290	Baseball Gloves
8	Baker	Boston	$1,500	Sportswear
9	Crowell	Boston	$9,500	Camping Gear
10	Crowell	Philadelphia	$6,000	Exercise Bikes
11	Crowell	New York	$1,500	Sportswear
12	Dykstra	Atlanta	$1,000	Sportswear
13	Dykstra	Dallas	$15,000	Rodeo Gear
14	Dykstra	Miami	$3,200	Golf Equipment

Using formulas derived in [ACCD93a] one may calculate the total number of valid orderings for any partial order that can be represented in the notation mentioned previously. For the case where a user specifies "ORDER BY SALESPERSON", the partial order above can be expressed as,

(1||2||3);(4||5||6||7||8);(9||10||11);(12||13||14)

Of the 14!=87,178,291,200 total possible combinations, there exist 25,920 valid orderings at the destination. A service that may deliver the records in any of these 25,920 orderings has a great deal more flexibility than in the ordered case where there is only 1 valid order for 14 objects. It is interesting to consider the real possibility of hundreds or even thousands of objects and the potential savings in communication costs.

In all cases, the underlying network is assumed to be unreliable and may thus introduce loss, duplication, and disorder. It makes no sense to put a partial order service on top of a reliable network. While the exact amount of unreliability in a network may vary and is not always well understood, initial experimental research indicates that real world networks, for example the service provided by the Internet's IP level, "yield high losses, duplicates and reorderings of packets" [AS93,BCP93]. The authors plan to conduct further experimentation into measuring Internet network unreliability. This information would say a great deal about the practical merit of a partial order service.

3. Reliability vs. Order

While TCP avoids the loss of even a single object, in fact for many applications, there exists a genuine ability to tolerate loss. Losing one frame per second in a 30 frame per second video or losing a segment of its accompanying audio channel is usually not a problem. Bearing this in mind, it is of value to consider a quality of service that combines a partial order with a level of tolerated loss (partial reliability). Traditionally there exist 4 services: reliable-ordered, reliable-unordered, unreliable-ordered, and unreliable-unordered. See Figure 5. Reliable-ordered service (denoted by a single point) represents the case where all objects are delivered in the order transmitted. File transfer is an example application requiring such a service.

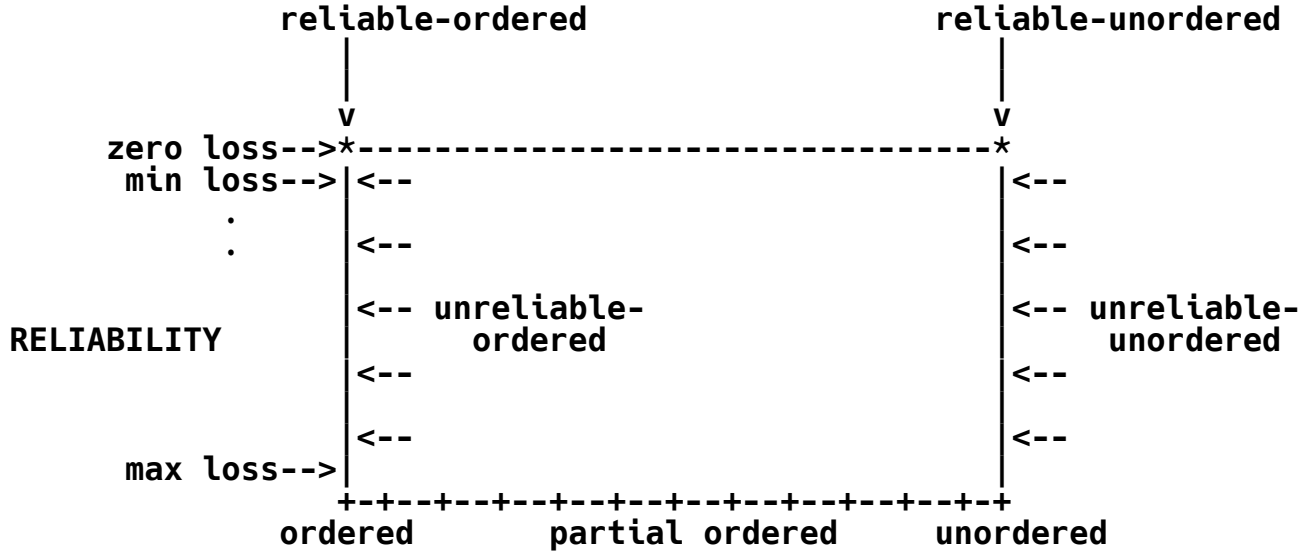


Figure 5: Quality Of Service: Reliability vs. Order - Traditional Service Types

In a reliable-unordered service (also a single point), all objects must be delivered, but not necessarily according to the order transmitted; in fact, any order will suffice. Some transaction processing applications such as credit card verification require such a service.

Unreliable-ordered service allows some objects to be lost. Those that are delivered, however, must arrive in relative order (An "unreliable" service does not necessarily lose objects; rather, it may do so without failing to provide its advertised quality of

service; e.g., the postal system provides an unreliable service). Since there are varying degrees of unreliability, this service is represented by a set of points in Figure 5. An unreliable-ordered service is applicable to packet-voice or teleconferencing applications.

Finally unreliable-unordered service allows objects to be lost and delivered in any order. This is the kind of service used for normal e-mail (without acknowledgment receipts) and electronic announcements or junk e-mail.

As mentioned previously, the concept of a partial order expands the order dimension from the two extremes of ordered and unordered to a range of discrete possibilities as depicted in Figure 6. Additionally, as will be discussed presently, the notion of reliability is extended to allow for varying degrees of reliability on a per-object basis providing even greater flexibility and improved resource utilization.

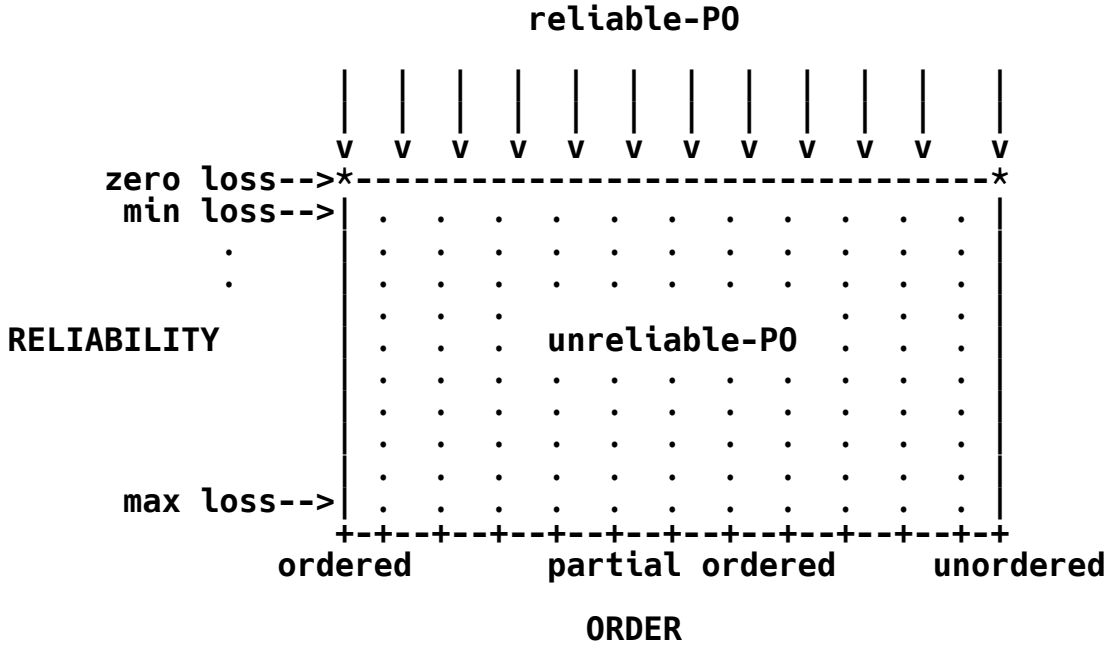


Figure 6: Quality Of Service: Reliability vs. Order - Partial Order Service

3.1 Reliability Classes

When considering unreliable service, one cannot assume that all objects are equal with regards to their reliability. This classification is reasonable if all objects are identical (e.g.,

video frames in a 30 frame/second film). Many applications, such as multimedia systems, however, often contain a variety of object types. Thus three object reliability classes are proposed: BART-NL, BART-L, and NBART-L. Objects are assigned to one of these classes depending on their temporal value as will be show presently.

BART-NL objects must be delivered to the destination. These objects have temporal value that lasts for an entire established connection and require reliable delivery (NL = No Loss allowed). An example of BART-NL objects would be the database records in Example 2.1 or the windows in the screen refresh in Example 2.3. If all objects are of type BART-NL, the service is reliable. One possible way to assure eventual delivery of a BART-NL object in a protocol is for the sender to buffer it, start a timeout timer, and retransmit it if no ACK arrives before the timeout. The receiver in turn returns an ACK when the object has safely arrived and been delivered (BART = Buffers, ACKs, Retransmissions, Timers).

BART-L objects are those that have temporal value over some intermediate amount of time - enough to permit timeout and retransmission, but not everlasting. Once the temporal value of these objects has expired, it is better to presume them lost than to delay further the delivery pipeline of information. One possibility for deciding when an object's usefulness has expired is to require each object to contain information defining its precise temporal value [DS93]. An example of a BART-L object would be a movie subtitle, sent in parallel with associated film images, which is valuable any time during a twenty second film sequence. If not delivered sometime during the first ten seconds, the subtitle loses its value and can be presumed lost. These objects are buffered-ACKed-retransmitted up to a certain point in time and then presumed lost.

NBART-L objects are those with temporal values too short to bother timing out and retransmitting. An example of a NBART-L object would be a single packet of speech in a packetized phone conversation or one image in a 30 image/sec film. A sender transmits these objects once and the service makes a best effort to deliver them. If the one attempt is unsuccessful, no further attempts are made.

An obvious question comes to mind - what about NBART-NL objects? Do such objects exist? The authors have considered the notion of communicating an object without the use of BART and still being able to provide a service without loss. Perhaps with the use of forward error correction this may become a viable alternative and could certainly be included in the protocol. However, for our purposes in this document, only the first three classifications will be considered.

While classic transport protocols generally treat all objects equally, the sending and receiving functions of a protocol providing partial order/partial reliability service will behave differently for each class of object. For example, a sender buffers and, if necessary, retransmits any BART-NL or BART-L objects that are not acknowledged within a predefined timeout period. On the contrary, NBART-L objects are forgotten as soon as they are transmitted.

4. Partial Order Connection

The implementation of a protocol that provides partial order service requires, at a minimum, (1) communication of the partial ordering between the two endpoints, and (2) dynamic evaluation of the deliverability of objects as they arrive at the receiver. In addition, this RFC describes the mechanisms needed to (3) initiate a connection, (4) provide varying degrees of reliability for the objects being transmitted, and (5) improve buffer utilization at the sender based on object reliability.

Throughout the discussion of these issues, the authors use the generic notion of "objects" in describing the service details. Thus, one of the underlying requirements of a partial order service is the ability to handle such an abstraction (e.g., recognize object boundaries). The details of object management are implementation dependent and thus are not specified in this RFC. However, as this represents a potential fundamental change to the TCP protocol, some discussion is in order.

At one extreme, it is possible to consider octets as objects and require that the application specify the partial order accordingly (octet by octet). This likely would entail an inordinate amount of overhead, processing each octet on an individual basis (literally breaking up contiguous segments to determine which, if any, octets are deliverable and which are not). At the other extreme, the transport protocol could maintain object atomicity regardless of size - passing arbitrarily large data structures to IP for transmission. At the sending side of the connection this would actually work since IP is prepared to perform source fragmentation, however, there is no guarantee that the receiving IP will be able to reassemble the fragments! IP relies on the TCP max segment size to prevent this situation from occurring[LMKQ89].

A more realistic approach given the existing IP constraints might be to maintain the current notion of a TCP max segment size for the lower-layer interface with IP while allowing a much larger object size at the upper-layer interface. Of course this presents some additional complexities. First of all, the transport layer will now have to be concerned with fragmentation/reassembly of objects larger

than the max segment size and secondly, the increased object sizes will require significantly more buffer space at the receiver if we want to buffer the object until it arrives in entirety. Alternatively, one may consider delivering "fragments" of an object as they arrive as long as the ordering of the fragments is correct and the application is able to process the fragments (this notion of fragmented delivery is discussed further in Section 6).

4.1 Connection Establishment

By extending the transport paradigm to allow partial ordering and reliability classes, a user application may be able to take advantage of a more efficient data transport facility by negotiating the optimal service level which is required - no more, no less. This is accomplished by specifying these variables as QOS parameters or, in TCP terminology, as options to be included in the TCP header [Pos81].

A TCP implementation that provides a partial order service requires the use of two new TCP options. The first is an enabling option "POC-permitted" (Partial Order Connection Permitted) that may be used in a SYN segment to request a partial order service. The other is the "POC-service-profile" option which is used periodically to communicate the service characteristics. This second option may be sent only after successful transmission and acknowledgment of the POC-permitted option.

A user process issuing either an active or passive OPEN may choose to include the POC-permitted option if the application can benefit from the use of a partial order service and in fact, in cases where the viability of such service is unknown, it is suggested that the option be used and that the decision be left to the user's peer.

For example, a multimedia server might issue a passive <SYN> with the POC-permitted option in preparation for the connection by a remote user.

Upon reception of a <SYN> segment with the POC-permitted option, the receiving user has the option to respond with a similar POC-permitted indication or may reject a partial order connection if the application does not warrant the service or the receiving user is simply unable to provide such a service (e.g., does not recognize the POC-permitted option).

In the event that simultaneous initial <SYN> segments are exchanged, the TCP will initiate a partial order connection only if both sides include the POC-permitted option.

A brief example should help to demonstrate this procedure. The following notation (a slight simplification on that employed in RFC 793) will be used. Each line is numbered for reference purposes. TCP-A (on the left) will play the role of the receiver and TCP-B will be the sender. Right arrows (-->) indicate departure of a TCP segment from TCP-A to TCP-B, or arrival of a segment at B from A. Left arrows indicate the reverse. TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of the line). Liberties are taken with the contents of the segments where only the fields of interest are shown.

```
      TCP-A                                                  TCP-B

   1. CLOSED                                                 LISTEN

   2. SYN-SENT    --> <CTL=SYN><POC-perm>             --> SYN-RECEIVED

   3. ESTABLISHED <-- <CTL=SYN,ACK><POC-perm>         <-- SYN-RECEIVED

   4. ESTABLISHED --> <CTL=ACK>                       --> ESTABLISHED
```

Figure 7. Basic 3-Way handshake for a partial order connection

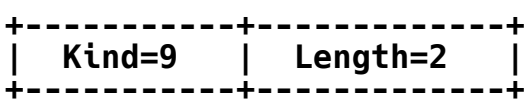
In line 1 of Figure 7, the sending user has already issued a passive OPEN with the POC-permitted option and is waiting for a connection. In line 2, the receiving user issues an active OPEN with the same option which in turn prompts TCP-A to send a SYN segment with the POC-permitted option and enter the SYN-SENT state. TCP-B is able to confirm the use of a PO connection and does so in line 3, after which TCP-A enters the established state and completes the connection with an ACK segment in line 4.

In the event that either side is unable to provide partial order service, the POC-permitted option will be omitted and normal TCP processing will ensue.

For completeness, the authors include the following specification for both the POC-permitted option and the POC-service-profile option in a format consistent with the TCP specification document [Pos81].

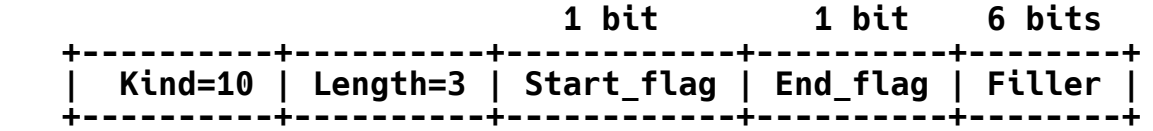
TCP POC-permitted Option:

Kind: 9 Length: - 2 bytes



TCP POC-service-profile Option:

Kind: 10 Length: 3 bytes



The first option represents a simple indicator communicated between the two peer transport entities and needs no further explanation. The second option serves to communicate the information necessary to carry out the job of the protocol - the type of information which is typically found in the header of a TCP segment - and raises some interesting questions.

Standard TCP maintains a 60-byte maximum header size on all segments. The obvious intuition behind this rule is that one would like to minimize the amount of overhead information present in each packet while simultaneously increasing the payload, or data, section. While this is acceptable for most TCP connections today, a partial-order service would necessarily require that significantly more control information be passed between transport entities at certain points during a connection. Maintaining the strict interpretation of this rule would prove to be inefficient. If, for example, the service profile occupied a total of 400 bytes (a modest amount as will be confirmed in the next section), then one would have to fragment this information across at least 10 segments, allocating 20 bytes per segment for the normal TCP header.

Instead, the authors propose that the service profile be carried in the data section of the segment and that the 3-byte POC-service-profile option described above be placed in the header to indicate the presence of this information. Upon reception of such a segment, the TCP extracts the service profile and uses it appropriately as will be discussed in the following sections.

The option itself, as shown here, contains two 1-bit flags necessary to handle the case where the service profile does not fit in a single TCP segment. The "Start_flag" indicates that the information in the data section represents the beginning of the service profile and the "End_flag" represents the converse. For service profiles which fit completely in a single segment, both flags will be set to 1. Otherwise, the Start_flag is set in the initial segment and the End_flag in the final segment allowing the peer entity to reconstrcut the entire service profile (using the normal sequence numbers in the segment header). The "Filler" field serves merely to complete the third byte of the option.

Note that the length of the service profile may vary during the connection as the order or reliability requirements of the user change but this length must not exceed the buffering ability of the peer TCP entity since the entire profile must be stored. The exact makeup of this data structure is presented in Section 4.2.

4.2 Data Transmission

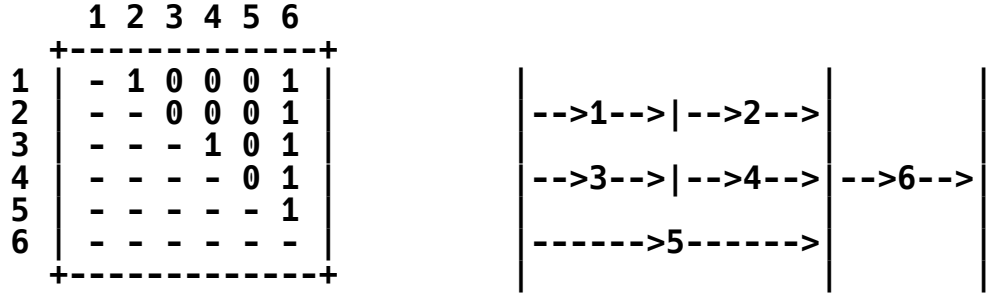
Examining the characteristics of a partial order TCP in chronological fashion, one would start off with the establishment of a connection as described in Section 4.1. After which, although both ends have acknowledged the acceptability of partial order transport, neither has actually begun a partial order transmission - in other words, both the sending-side and the receiving-side are operating in a normal, ordered-reliable mode. For the subsequent discussion, an important distinction is made in the terms sending-side and receiving-side which refer to the data flow from the sender and that from the receiver, respectively.

For the partial ordering to commence, the TCP must be made aware of the acceptable object orderings and reliability for both the send-side and receive-side of the connection for a given set of objects (hereafter referred to as a "period"). This information is contained in the service profile and it is the responsibility of the user application to define this profile. Unlike standard TCP where applications implicitly define a reliable, ordered profile; with partial order TCP, the application must explicity define a profile.

The representation of the service profile is one of the concerns for the transport protocol. It would be useful if the TCP could encode a partial ordering in as few bits as possible since these bits will be transmitted to the destination each time the partial order changes. A matrix representation appears to be well-suited to encoding the partial order and a vector has been proposed to communicate and manage the reliability aspects of the service. Temporal values may be included within the objects themselves or may be defined as a function of the state of the connection [DS93]. Using these data structures, the complete service profile would include (1) a partial order matrix, (2) a reliability vector and (3) an object_sizes vector which represents the size of the objects in octets (see [ACCD93a,CAC93] for a discussion on alternative structures for these variables).

Throughout this section, we use the following service profile as a running example. Shown here is a partial order matrix and graphical representation for a simple partial order with 6 objects - ((1;2)||(3;4)||5);6. In the graphical diagram, arrows (-->) denote sequential order and objects in parallel can be delivered in either

order. So in this example, object 2 must be delivered after object 1, object 4 must be delivered after object 3, and object 6 must be delivered after objects 1 through 5 have all been delivered. Among the 6 objects, there are 30 valid orderings for this partial order (each valid ordering is known as a linear extension of the partial order).



PO Matrix PO Graph

In the matrix, a 1 in row i of column j denotes that object i must be delivered before object j. Note that if objects are numbered in any way such that 1,2,3,...,N is a valid ordering, only the upper right triangle of the transitively closed matrix is needed [ACCD93a]. Thus, for N objects, the partial order can be encoded in (N*(N-1)/2) bits.

The reliability vector for the case where reliability classes are enumerated types such as {BART-NL=1, BART-L=2, NBART-L = 3} and all objects are BART-NL would simply be, <1, 1, 1, 1, 1, 1>. Together with the object_sizes vector, the complete service profile is described.

This information must be packaged and communicated to the sending TCP before the first object is transmitted using a TCP service primitive or comparable means depending upon the User/TCP interface. Once the service profile has been specified to the TCP, it remains in effect until the connection is closed or the sending user specifies a new service profile. In the event that the largest object size can not be processed by the receiving TCP, the user application is informed that the connection cannot be maintained and the normal connection close procedure is followed.

Typically, as has been described here, the service profile definition and specification is handled at the sending end of the connection, but there could be applications (such as the screen refresh) where the receiving user has this knowledge. Under these circumstances the receiving user is obliged to transmit the object ordering on the

return side of the connection (e.g., when making the request for a screen refresh) and have the sender interpret this data to be used on the send side of the connection.

Requiring that the sending application specify the service profile is not an arbitrary choice. To ensure proper object identification, the receiving application must transmit the new object numbering to the sending application (not the sending transport layer). Since the sending application must receive this information in any case, it simplifies matters greatly to require that the sending application be the only side that may specify the service profile to the transport layer.

Consider now the layered architecture diagram in Figure 8 and assume that a connection already is established. Let us now say that UserA specifies the service profile for the sending-side of the connection via its interface with TCP-A. TCP-A places the profile in the header of one or more data packets (depending upon the size of the service profile, the profile may require several packets), sets the POC-service-profile option and passes it to IP for transmission over the network. This packet must be transmitted reliably, therefore TCP-A buffers it and starts a normal retransmit timer. Subsequently, the service profile arrives at the destination node and is handed to TCP-B (as indicated by the arrows in Figure 8). TCP-B returns an acknowledgment and immediately adopts the service profile for one direction of data flow over the connection. When the acknowledgment arrives back at TCP-A, the cycle is complete and both sides are now able to use the partial order service.

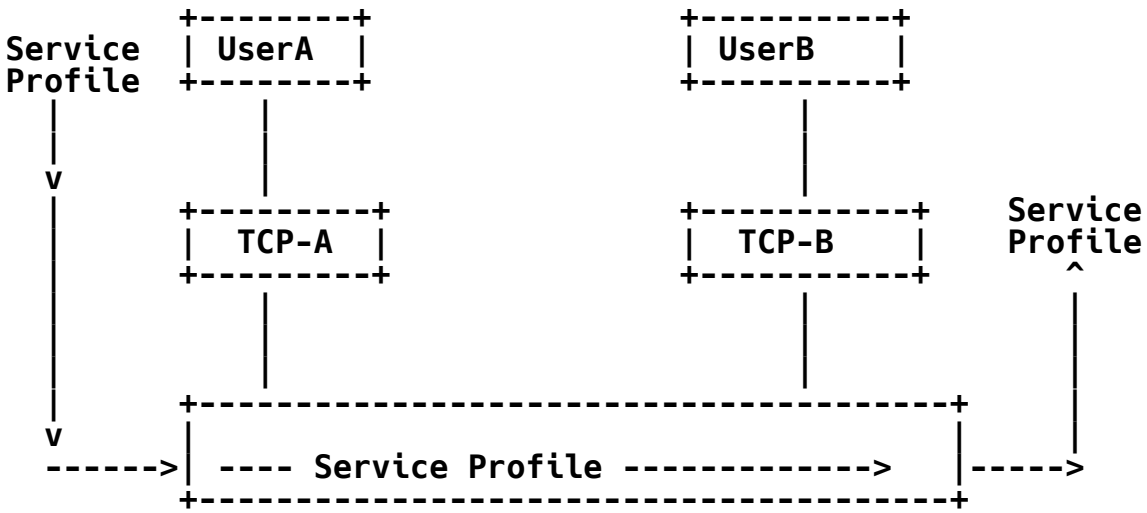


Figure 8. Layered Communication Architecture

Note that one of the TCP entities learns of the profile via its user interface, while the other TCP entity is informed via its network interface.

For the remaining discussions, we will assume that a partial order profile has been successfully negotiated for a single direction of the connection (as depicted in Figure 8) and that we may now speak of a "sending TCP" (TCP-A) and a "receiving TCP" (TCP-B). As such, TCP-A refers to the partial order data stream as the "send-side" of the connection, while TCP-B refers to the same data stream as the "receive-side".

Having established a partial order connection, the communicating TCPs each have their respective jobs to perform to ensure proper data delivery. The sending TCP ascertains the object ordering and reliability from the service profile and uses this information in its buffering/retransmission policy. The receiver modifications are more significant, particularly the issues of object deliverability and reliability. And both sides will need to redefine the notion of window management. Let us look specifically at how each side of the TCP connection is managed under this new paradigm.

4.2.1 Sender

The sender's concerns are still essentially four-fold - transmitting data, managing buffer space, processing acknowledgments and retransmitting after a time-out - however, each takes on a new meaning in a partial order service. Additionally, the management of the service profile represents a fifth duty not previously needed.

Taking a rather simplistic view, normal TCP output processing involves (1) setting up the header, (2) copying user data into the outgoing segment, (3) sending the segment, (4) making a copy in a send buffer for retransmission and (5) starting a retransmission timer. The only difference with a partial order service is that the reliability vector must be examined to determine whether or not to buffer the object and start a timer - if the object is classified as NBART-L, then steps 4 and 5 are omitted.

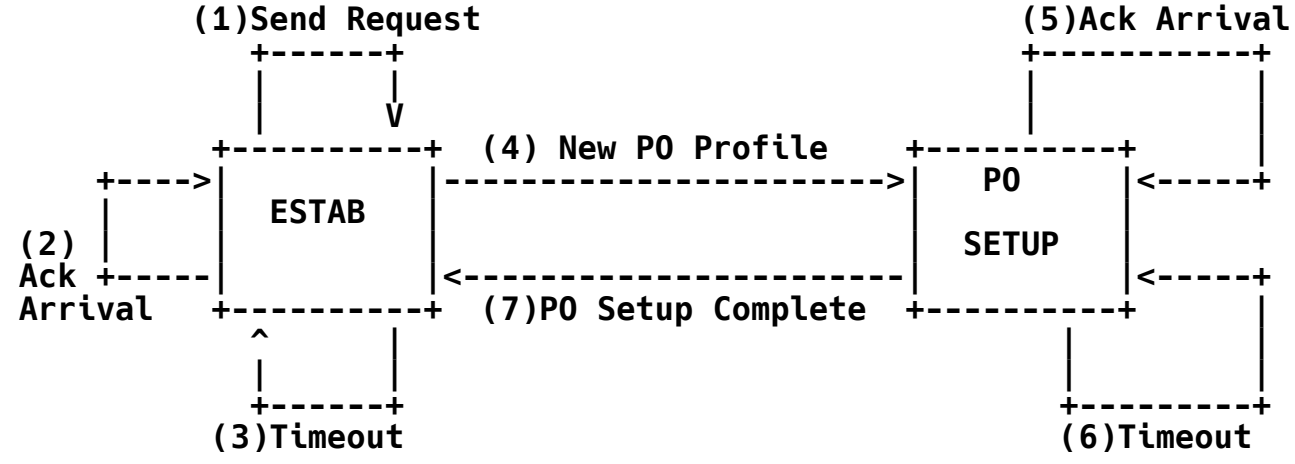
Buffer management at the sending end of a partial order connection is dependent upon the object reliability class and the object size. When transmitting NBART-L objects the sender need not store the data for later possible retransmission since NBART-L objects are never retransmitted. The details of buffer management - such as whether to allocate fixed-size pools of memory, or perhaps utilize a dynamic heap allocation strategy - are left to the particular system implementer.

Acknowledgment processing remains essentially intact - acknowledgments are cumulative and specify the peer TCP's window advertisement. However, determination of this advertisement is no longer a trivial process dependent only upon the available buffer space (this is discussed further in Section 4.2.2). Moreover, it should be noted that the introduction of partial ordering and partial reliability presents several new and interesting alternatives for the acknowledgment policy. The authors are investigating several of these strategies through a simulation model and have included a brief discussion of these issues in Section 6.

The retransmit function of the TCP is entirely unchanged and is therefore not discussed further.

For some applications, it may be possible to maintain the same partial order for multiple periods (e.g., the application repeats the same partial order). In the general case, however, the protocol must be able to change the service profile during an existing connection. When a change in the service profile is requested, the sending TCP is obliged to complete the processing of the current partial order before commencing with a new one. This ensures consistency between the user applications in the event of a connection failure and simplifies the protocol (future study is planned to investigate the performance improvement gained by allowing concurrent different partial orders). The current partial order is complete when all sending buffers are free. Then negotiation of the new service profile is performed in the same manner as with the initial profile.

Combining these issues, we propose the following simplified state machine for the protocol (connection establishment and tear down remains the same and is not show here).



```
   Event (1) - User Makes a Data Send Request
   =========
      If Piggyback Timer is set then
           cancel piggyback timer
      Package and send the object (with ACK for receive-side)
      If object type = (BART-L,BART-NL) then
           Store the object and start a retransmit timer
      If sending window is full then
           Block Event (1) - allow no further send requests from user

   Event (2) - ACK Arrives
   =========
      If ACKed object(s) is buffered then
           Release the buffer(s) and stop the retransmit timer(s)
      Extract the peer TCP's window advertisement
      If remote TCP's window advertisement > sending window then
           Enable Event (1)
      If remote TCP's window advertisement <= sending window then
           Block Event (1) - allow no further send requests from user
      Adjust sending window based on received window advertisement

   Event (3) - Retransmit Timer Expires
   =========
      If Piggyback Timer is set then
           cancel piggyback timer
      Re-transmit the segment (with ACK for receive-side)
      Restart the timer

   Event (4) - PO Service Profile Arrives at the User Interface
   =========
      Transition to the PO SETUP state
      Store the Send-side PO service profile
      Package the profile into 1 or more segments, setting the
           POC-Service-Profile option on each
      If Piggyback Timer is set then
           cancel piggyback timer
      Send the segment(s) (with ACK for receive-side)
      Store the segment(s) and start a retransmit timer

   Event (5) - ACK Arrival
   =========
      If ACKed object(s) is buffered then
           Release the buffer(s) and stop the retransmit timer(s)
      Extract the peer TCP's window advertisement
      If all objects from previous service profile have been ACKed and
      the new service profile has been ACKed then enable Event (7)
```

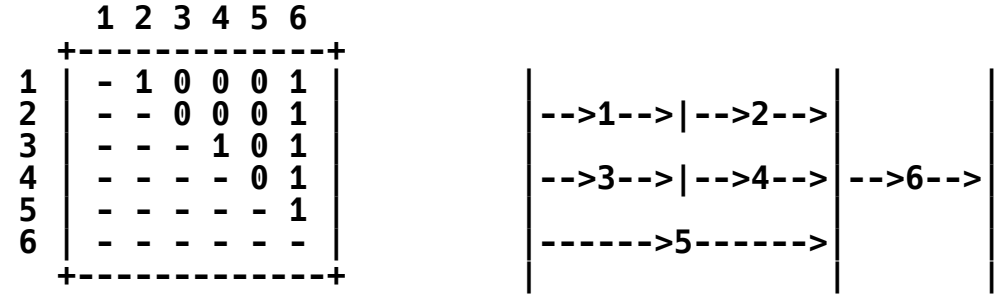
Event (6) - Retransmit Timer Expires
=========
If Piggyback Timer is set then
cancel piggyback timer
Re-transmit the segment (with ACK for receive-side)
Restart the timer

Event (7) - PO Setup Completed
=========
Transition to the ESTAB state and begin processing new service profile

4.2.2 Receiver

The receiving TCP has additional decisions to make involving object deliverability, reliability and window management. Additionally, the service profile must be established (and re-established) periodically and some special processing must be performed at the end of each period.

When an object arrives, the question is no longer, "is this the next deliverable object?", but rather, "is this ONE OF the next deliverable objects?" Hence, it is convenient to think of a "Deliverable Set" of objects with a partial order protocol. To determine the elements of this set and answer the question of deliverability, the receiver relies upon the partial order matrix but, unlike the sender, the receiver dynamically updates the matrix as objects are processed thus making other objects (possibly already buffered objects) deliverable as well. A check of the object type also must be performed since BART-NL and BART-L objects require an ACK to be returned to the sender but NBART-L do not. Consider our example from the previous section.



PO Matrix PO Graph

When object 5 arrives, the receiver scans column 5, finds that the object is deliverable (since there are no 1's in the column) and immediately delivers the object to the user application. Then, the

matrix is updated to remove the constraint of any object whose delivery depends on object 5 by clearing all entries of row 5. This may enable other objects to be delivered (for example, if object 2 is buffered then the delivery of object 1 will make object 2 deliverable). This leads us to the next issue - delivery of stored objects.

In general, whenever an object is delivered, the buffers must be examined to see if any other stored object(s) becomes deliverable. CAC93 describes an efficient algorithm to implement this processing based on traversing the precedence graph.

Consideration of object reliability is interesting. The authors have taken a polling approach wherein a procedure is executed periodically, say once every 100 milliseconds, to evaluate the temporal value of outstanding objects on which the destination is waiting. Those whose temporal value has expired (i.e. which are no longer useful as defined by the application) are "declared lost" and treated in much the same manner as delivered objects - the matrix is updated, and if the object type is BART-L, an ACK is sent. Any objects from the current period which have not yet been delivered or declared lost are candidates for the "Terminator" as the procedure is called. The Terminator's criterion is not specifically addressed in this RFC, but one example might be for the receiving user to periodically pass a list of no-longer-useful objects to TCP-B.

Another question which arises is, "How does one calculate the send and receive windows?" With a partial order service, these windows are no longer contiguous intervals of objects but rather sets of objects. In fact, there are three sets which are of interest to the receiving TCP one of which has already been mentioned - the Deliverable Set. Additionally, we can think of the Bufferable Set and the Receivable Set. Some definitions are in order:

Deliverable Set: objects which can be immediately passed up to the user.

Buffered Set: objects stored in a buffer awaiting delivery.

Bufferable Set: objects which can be stored but not immediately delivered (due to some ordering constraint).

Receivable Set: union of the Deliverable Set and the Bufferable Set (which are disjoint) - intuitively, all objects which are "receivable" must be either "deliverable" or "bufferable".

The following example will help to illustrate these sets. Consider our simple service profile from earlier for the case where the size of each object is 1 MByte and the receiver has only 2 MBytes of buffer space (enough for 2 objects). Define a boolean vector of length N (N = number of objects in a period) called the Processed Vector which is used to indicate which objects from the current period have been delivered or declared lost. Initially, all buffers are empty and the PO Matrix and Processed Vector are as shown here,

```
     1 2 3 4 5 6
   +-------------+
 1 | - 1 0 0 0 1 |
 2 | - - 0 0 0 1 |
 3 | - - - 1 0 1 |
 4 | - - - - 0 1 |
 5 | - - - - - 1 |      [ F F F F F F ]
 6 | - - - - - - |        1 2 3 4 5 6
   +-------------+

      PO Matrix        Processed Vector
```

From the PO Matrix, it is clear that the Deliverable Set = {(1,1),(1,3),(1,5)}, where (1,1) refers to object #1 from period #1, asssuming that the current period is period #1.

The Bufferable Set, however, depends upon how one defines bufferable objects. Several approaches are possible. The authors' initial approach to determining the Bufferable Set can best be explained in terms of the following rules,

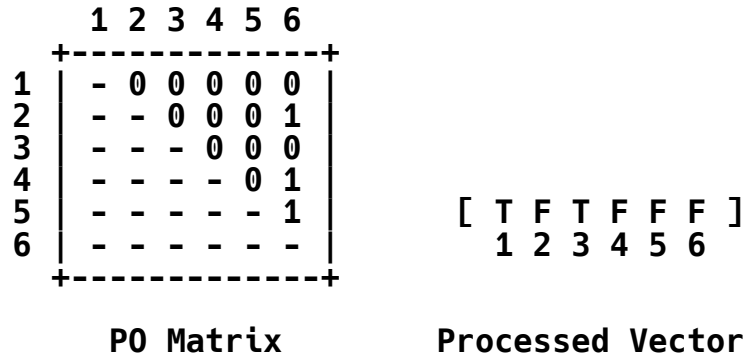
Rule 1: Remaining space must be allocated for all objects from period i before any object from period i+1 is buffered

Rule 2: In the event that there exists enough space to buffer some but not all objects from a given period, space will be reserved for the first objects (i.e. 1,2,3,...,k)

With these rules, the Bufferable Set = {(1,2),(1,4)}, the Buffered Set is trivially equal to the empty set, { }, and the Receivable Set = {(1,1),(1,2),(1,3),(1,4),(1,5)}.

Note that the current acknowledgment scheme uses the min and max values in the Receivable Set for its window advertisement which is transmitted in all ACK segments sent along the receive-side of the connection (from receiver to sender). Moreover, the "piggyback_delay" timer is still used to couple ACKs with return data (as utilized in standard TCP).

Returning to our example, let us now assume that object 1 and then 3 arrive at the receiver and object 2 is lost. After processing both objects, the PO Matrix and Processed Vector will have the following updated structure,



PO Matrix Processed Vector

We can see that the Deliverable Set = {(1,2),(1,4),(1,5)}, but what should the Bufferable Set consist of? Since only one buffer is required for the current period's objects, we have 1 Mbyte of additional space available for "future" objects and therefore include the first object from period #2 in both the Bufferable and the Receivable Set,

Deliverable Set = {(1,2),(1,4),(1,5)}

Bufferable Set = {(1,6),(2,1)}

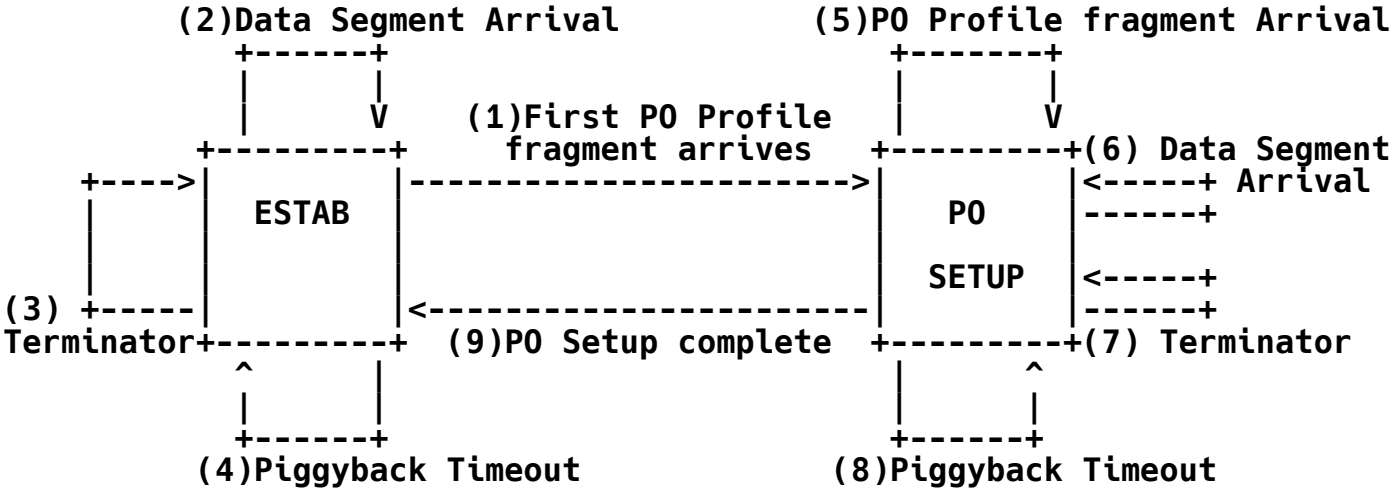
Buffered Set = { }

Receivable Set = {(1,2),(1,4),(1,5),(1,6),(2,1)}

In general, the notion of window management takes on new meaning with a partial order service. One may re-examine the classic window relations with a partial order service in mind and devise new, less restrictive relations which may shed further light on the operation of such a service.

Two final details: (1) as with the sender, the receiver must periodically establish or modify the PO service profile and (2) upon processing the last object in a period, the receiver must re-set the PO matrix and Processed vector to their initial states.

Let us look at the state machine and pseudo-code for the receiver.



Event 1 - First PO Service Profile fragment arrives at network interface

```
   Transition to the PO SETUP state
   Store the PO service profile (fragment)
   Send an Acknowledgement of the PO service profile (fragment)
```

Event 2 - Data Segment Arrival

```
   If object is in Deliverable Set then
        Deliver the object
        Update PO Matrix and Processed Vector
        Check buffers for newly deliverable objects
        If all objects from current period have been processed then
             Start the next period (re-initialize data structures)
        Start piggyback_delay timer to send an ACK
   Else if object is in Bufferable Set then
        Store the object
   Else
        Discard object
        Start piggyback_delay timer to send an ACK
```

Event 3 - Periodic call of the Terminator

```
   For all unprocessed objects in the current period do
        If object is "no longer useful" then
             Update PO Matrix and Processed Vector
             If object is in a buffer then
                  Release the buffer
             Check buffers for newly deliverable objects
```

```
                If all objects from current period have been processed
                then Start the next period (re-initialize data
                structures)

   Event 4 - Piggyback_delay Timer Expires
   =======
      Send an ACK
      Disable piggyback_delay timer

   Event 5 - PO Service Profile fragment arrives at network interface
   =======
      Store the PO service profile (fragment)
      Send an Acknowledgement of the PO service profile (fragment)
      If entire PO Service profile has been received then enable Event
      (9)

   Event 6 - Data Segment arrival
   =======
      (See event 2)

   Event 7 - Periodic call of the terminator
   =======
      (See Event 3)

   Event 8 - Piggyback_delay Timer Expires
   =======
      (See Event 4)

   Event 9 - PO Setup Complete
   =======
      Transition to the ESTAB state

   Note that, for reasons of clarity, we have used a transitively closed
   matrix representation of the partial order.  A more efficient
   implementation based on an adjacency list representation of a
   transitively reduced precedence graph results in a more efficient
   running time [CAC93].
```

5. Quantifying and Comparing Partial Order Services

While ordered, reliable delivery is ideal, the existence of less-than-ideal underlying networks can cause delays for applications that need only partial order or partial reliability. By introducing a partial order service, one may in effect relax the requirements on order and reliability and presumably expect some savings in terms of buffer utilization and bandwidth (due to fewer retransmissions) and shorter overall delays. A practical question to be addressed is, "what are the expected savings likely to be?"

As mentioned in Section 2, the extent of such savings will depend largely on the quality of the underlying network - bandwidth, delay, amount and distribution of loss/duplication/disorder - as well as the flexibility of the partial order itself - specified by the PO matrix and reliability vector. If the underlying network has no loss, a partial order service essentially becomes an ordered service. Collecting experimental data to ascertain realistic network conditions is a straightforward task and will help to quantify in general the value of a partial order service [Bol93]. But how can one quantify and compare the cost of providing specific levels of service?

Preliminary research indicates that the number of linear extensions (orderings) of a partial order in the presence of loss effectively measures the complexity of that order. The authors have derived formulae for calculating the number of extensions when a partial order is series-parallel and have proposed a metric for comparing partial orders based on this number [ACCD93b]. This metric could be used as a means for charging for the service, for example. What also may be interesting is a specific head-to-head comparison between different partial orders with varying degrees of flexibility. Work is currently underway on a simulation model aimed at providing this information. And finally, work is underway on an implementation of TCP which includes partial order service.

6. Future Direction

In addition to the simulation and implementation work the authors are pursuing several problems related to partial ordering which will be mentioned briefly.

An interesting question arises when discussing the acknowledgment strategy for a partial order service. For classic protocols, a cumulative ACK of object i confirms all objects "up to and including" i. But the meaning of "up to and including" with a partial order service has different implications than with an ordered service.

Consider our example partial order, ((1;2)||(3;4)||5);6). What should a cumulative ACK of object 4 confirm? The most logical definition would say it confirms receipt of object 4 and all objects that precede 4 in the partial order, in this case, object 3. Nothing is said about the arrival of objects 1 or 2. With this alternative interpretation where cumulative ACKs depend on the partial order, the sender must examine the partial order matrix to determine which buffers can be released. In this example, scanning column 4 of the matrix reveals that object 3 must come before object 4 and therefore both object buffers (and any buffers from a previous period) can be released.

Other partial order acknowledgment policies are possible for a protocol providing a partial order service including the use of selective ACKs (which has been proposed in [JB88] and implemented in the Cray TCP [Chang93]) as well as the current TCP strategy where an ACK of i also ACKs everything <= i (in a cyclical sequence number space). The authors are investigating an ACK policy which utilizes a combination of selective and "partial-order-cumulative" acknowledgments. This is accomplished by replacing the current TCP cumulative ACK with one which has the partial order meaning as described above and augmenting this with intermittent selective ACKs when needed.

In another area, the notion of fragmented delivery, mentioned in the beginning of Section 4, looks like a promising technique for certain classes of applications which may offer a substantial improvement in memory utilization. Briefly, the term fragmented delivery refers to the ability to transfer less-than-complete objects between the transport layer and the user application (or session layer as the case may be). For example, a 1Mbyte object could potentially be delivered in multiple "chunks" as segments arrive thus freeing up valuable memory and reducing the delay on those pieces of data. The scenario becomes somewhat more complex when multiple "parallel streams" are considered where the application could now receive pieces of multiple objects associated with different streams.

Additional work in the area of implementing a working partial order protocol is being performed both at the University of Delaware and at the LAAS du CNRS laboratory in Toulouse, France - particularly in support of distributed, high-speed, multimedia communication. It will be interesting to examine the processing requirements for an implementation of a partial order protocol at key events (such as object arrival) compared with a non-partial order implementation.

Finally, the authors are interested in the realization of a network application utilizing a partial order service. The aim of such work is threefold: (1) provide further insight into the expected performance gains, (2) identify new issues unique to partial order transport and, (3) build a road-map for application designers interested in using a partial order service.

7. Summary

This RFC introduces the concepts of a partial order service and discusses the practical issues involved with including partial ordering in a transport protocol. The need for such a service is motivated by several applications including the vast fields of distributed databases, and multimedia. The service has been presented as a backward-compatible extension to TCP to adapt to

applications with different needs specified in terms of QOS parameters.

The notion of a partial ordering extends QOS flexibility to include object delivery, reliability, and temporal value thus allowing the transport layer to effectively handle a wider range of applications (i.e., any which might benefit from such mechanisms). The service profile described in Section 4 accurately characterizes the QOS for a partial order service (which encompasses the two extremes of total ordered and unordered transport as well).

Several significant modifications have been proposed and are summarized here:

(1) Replacing the requirement for ordered delivery with one for application-dependent partial ordering

(2) Allowing unreliable and partially reliable data transport

(3) Conducting a non-symmetrical connection (not entirely foreign to TCP, the use of different MSS values for the two sides of a connection is an example)

(4) Management of "objects" rather than octets

(5) Modified acknowledgment strategy

(6) New definition for the send and receive "windows"

(7) Extension of the User/TCP interface to include certain QOS parameters

(8) Use of new TCP options

As evidenced by this list, a partial order and partial reliability service proposes to re-examine several fundamental transport mechanisms and, in so doing, offers the opportunity for substantial improvement in the support of existing and new application areas.

8. References

[ACCD93a] Amer, P., Chassot, C., Connolly, T., and M. Diaz, "Partial Order Transport Service for Multimedia Applications: Reliable Service", Second International Symposium on High Performance Distributed Computing (HPDC-2), Spokane, Washington, July 1993.

[ACCD93b] Amer, P., Chassot, C., Connolly, T., and M. Diaz, "Partial Order Transport Service for Multimedia Applications: Unreliable Service", Proc. INET '93, San Francisco, August 1993.

[AH91] Anderson, D., and G. Homsy, "A Continuous Media I/O Server and its Synchronization Mechanism", IEEE Computer, 24(10), 51-57, October 1991.

[AS93] Agrawala, A., and D. Sanghi, "Experimental Assessment of End-to-End Behavior on Internet," Proc. IEEE INFOCOM '93, San Francisco, CA, March 1993.

[BCP93] Claffy, K., Polyzos, G., and H.-W. Braun, "Traffic Characteristics of the T1 NSFNET", Proc. IEEE INFOCOM '93, San Francisco, CA, March 1993.

[Bol93] Bolot, J., "End-to-End Packet Delay and Loss Behavior in the Internet", SIGCOMM '93, Ithaca, NY, September 1993.

[CAC93] Conrad, P., Amer, P., and T. Connolly, "Improving Performance in Transport-Layer Communications Protocols by using Partial Orders and Partial Reliability", Work in Progress, December 1993.

[Chang93] Chang, Y., "High-Speed Transport Protocol Evaluation -- the Final Report", MCNC Center for Communications Technical Document, February 1993.

[Dee89] Deering, S., "Host Extensions for IP Multicasting," STD 5, RFC 1112 Stanford University, August 1989.

[DS93] Diaz, M., and P. Senac, "Time Stream Petri Nets: A Model for Multimedia Synchronization", Proceedings of Multimedia Modeling '93, Singapore, 1993.

[HKN91] Hardt-Kornacki, S., and L. Ness, "Optimization Model for the Delivery of Interactive Multimedia Documents", In Proc. Globecom '91, 669-673, Phoenix, Arizona, December 1991.

[JB88] Jacobson, V., and R. Braden, "TCP Extensions for Long-Delay Paths", RFC 1072, LBL, USC/Information Sciences Institute, October 1988.

[JBB92] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, LBL, Cray Research, USC/Information Sciences Institute, May 1992.

[LMKQ89] Leffler, S., McKusick, M., Karels, M., and J. Quarterman, "4.3 BSD UNIX Operating System", Addison-Wesley Publishing Company, Reading, MA, 1989.

[OP91] O'Malley, S., and L. Peterson, "TCP Extensions Considered Harmful", RFC 1263, University of Arizona, October 1991.

[Pos81] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification," STD 7, RFC 793, DARPA, September 1981.

Security Considerations

Security issues are not discussed in this memo.

Authors' Addresses

Tom Connolly
101C Smith Hall
Department of Computer & Information Sciences
University of Delaware
Newark, DE 19716 - 2586

EMail: connolly@udel.edu

Paul D. Amer
101C Smith Hall
Department of Computer & Information Sciences
University of Delaware
Newark, DE 19716 - 2586

EMail: amer@udel.edu

Phill Conrad
101C Smith Hall
Department of Computer & Information Sciences
University of Delaware
Newark, DE 19716 - 2586

EMail: pconrad@udel.edu