Network Working Group
Request for Comments: 955

R. Braden
UCLA OAC
September 1985

Towards a Transport Service for Transaction Processing Applications

STATUS OF THIS MEMO

This RFC is concerned with the possible design of one or more new protocols for the ARPA-Internet, to support kinds of applications which are not well supported at present. The RFC is intended to spur discussion in the Internet research community towards the development of new protocols and/or concepts, in order to meet these unmet application requirements. It does not represent a standard, nor even a concrete protocol proposal. Distribution of this memo is unlimited.

1. INTRODUCTION

The DoD Internet protocol suite includes two alternative transport service [1] protocols, TCP and UDP, which provide virtual circuit and datagram service, respectively [RFC-793, RFC-768]. These two protocols represent points in the space of possible transport service attributes which are quite "far apart". We want to examine an important class of applications, those which perform what is often called "transaction processing". We will see that the communication needs for these applications fall into the gap "between" TCP and UDP -- neither protocol is very appropriate.

We will then characterize the attributes of a possible new transport-level protocol, appropriate for these ill-served transaction-processing applications.

In writing this memo, the author had in mind several assumptions about Internet protocol development.

* Assumption 1: The members of the Internet research community now understand a great deal about protocols, and given a list of consistent attributes we can probably generate a reasonable protocol to meet that specification.

  This is not to suggest that design of good protocols is easy. It does reflect an assumption (perhaps wrong) that the set of basic protocol techniques we have invented so far is sufficient to give a good solution for any point in the attribute space, and that we can forsee (at least in a general way) many of the consequences of particular protocol design choices.

* Assumption 2: We need to develop appropriate service requirements for a "transaction processing protocol".

  The classifications "virtual circuit" and "datagram" immediately define in our minds the most important attributes of TCP and UDP. We have no such immediate agreement about the services to be provided for transaction processing. The existing and proposed transaction-oriented protocols show a number of alternative choices [e.g., Cour81, BiNe84, Coop84, Cher85, Crow85, Gurw85, Mill85].

Many of the ideas discussed here are not new. For example, Birrell and Nelson [BiNe84] and Watson [Wats81] have described transport-level protocols appropriate for transactions. Our purpose here is to urge the solution of this problem within the Internet protocol family.

2. TRANSACTION PROCESSING COMMUNICATIONS

We begin by listing the characteristics of the communication patterns typical in "transaction processing" applications.

* Unsymmetrical Model

  The two end points of the communication typically take different roles, generally called "client" and "server". This leads to an unsymmetrical communication pattern.

  For example, the client always initiates a communication sequence or "transaction". Furthermore, an important subclass of applications uses only a simple exchange of messages, a "request" to the server followed by a "reply" to the client.

  Other applications may require a continuing exchange of messages, a dialog or "conversation". For example, a request to read a file from a file server might result in a series of messages, one per file block, in reply. More complex patterns may occur.

* Simplex Transfers

  Regardless of the pattern, it always consists of a series of SIMPLEX data transfers; at no time is it necessary to send data in both directions simultaneously.

* **Short Duration**

  Transaction communication sequences generally have short duration, typically 100's of milliseconds up to 10's of seconds, but never hours.

* **Low Delay**

  Some applications require minimal communication delay.

* **Few Data Packets**

  In many applications, the data to be sent can be compressed into one or a few IP packets. Applications which have been designed with LAN's in mind are typically very careful to minimize the number of data packets for each request/reply sequence.

* **Message Orientation**

  The natural unit of data which is passed in a transaction is an entire message ("record"), not a stream of bytes.

3. EXAMPLE: NAME SERVERS

To focus our ideas, we will now discuss several particular types of distributed applications which are of pressing concern to members of the Internet research community, and which require transaction-oriented communication.

First, consider the name server/name resolver system [RFC-882, RFC-883] which is currently being introduced into the (research) Internet. Name servers must use TCP and/or UDP as their transport protocol. TCP is appropriate for the bulk transfers needed to update a name server's data base. For this case, reliability is essential, and virtual-circuit setup overhead is negligible compared to the data transfer itself. However, the choice of a transport protocol for the transaction traffic -- queries and responses -- is problematic.

* TCP would provide reliable and flow-controlled transfer of arbitrary-sized queries and responses. However, TCP exacts a high cost as a result of its circuit setup and teardown phases.

* UDP avoids the overhead of TCP connection setup. However, UDP has two potentially-serious problems -- (1) unreliable communication, so that packets may be lost, duplicated, and/or

reordered; and (2) the limitation of a data object (query/response) to the 548-byte maximum in a single UDP packet.

At present, name servers are being operated using UDP for transaction communication. Note that name server requests have a special property, idempotency; as a result, lost, duplicated, or reordered queries do not prevent the name-server system from working. This would seem to favor the use of UDP.

However, it seems quite likely that the defects of UDP will make it unusable for an increasing fraction of queries.

* The average size of individual replies will certainly increase, as the more esoteric mail lookup features are used, as the host population explodes (resulting in a logarithmic increase in domain name sizes), and as the number of alternate acceptable answers increases. As a result, a single response will more often overflow a single UDP packet.

* The average end-to-end reliability will decrease as some of the flakier paths of the Internet are brought into use by name resolvers.

  This will lead to a serious problem of choosing an appropriate retransmission timeout. A name resolver using UDP cannot distinguish packet loss in the Internet from queueing delay in the server. As a result, name servers we have seen have chosen long fixed timeouts (e.g., 30 seconds or more). This will result in long delays in name resolution when packets are lost.

  One might think that delays in name resolution might not be an issue since most name lookups are done by a mailer daemon. However, ARPANET experience with user mail interfaces has shown that it is always desirable to verify the correctness of each host name as the user enters the "To:" and "CC:" addresses for a message. Hence, delays due to lost UDP packets will be directly visible to users.

More generally, the use of UDP violates sound communication system design in two important ways:

* The name resolver/server applications have to provide timeouts and retransmissions to protect against "errors" (losses) in the communication system. This certainly violates network transparency, and requires the application to make decisions for which it is not well-equipped.

As a general design principle, it seems that (Inter-) network properties, especially bad properties, ought to a large extent to be hidden below the transport-service boundary [2].

* The name resolver/server applications must know the maximum size of a UDP datagram.

  It is clearly wrong for an application program to contain knowledge of the number 576 or 548! This does not imply that there cannot be a limitation on the size of a message, but any such limitation should be imposed by the particular application-level protocol, not the transport or internetwork level.

It seems that the TCP/UDP choice for name servers presents an ugly dilemma. We suggest that the solution should be a new transaction-oriented transport protocol with the following features:

* Reliable ("at-least-once") Delivery of Data;

* No Explicit Connection Setup or Teardown Phases;

* Fragmentation and Reassembly of Messages;

* Minimal Idle State in both Client and Server.

4. ANOTHER EXAMPLE: DISTRIBUTED OPERATING SYSTEMS

Distributed operating systems represent another potential application for a transaction-oriented transport service. A number of examples of distributed operating systems have been built using high-speed local area networks (LAN's) for communication (e.g, Cronus, Locus, V-System). Typically, these systems use private communication protocols above the network layer, and the private transport-level protocol is carefully designed to minimize latency across the LAN. They make use of the inherent reliability of the LAN and of simple transactions using single-packet exchanges.

Recently there have been efforts to extend these systems to operate across the Internet [Cher85, Shel85]. Since these are not "open" systems, there is no requirement that they use a standard transport protocol. However, the availability of a suitable transport protocol for transactions could considerably simplify development of future distributed systems.

The essential requirement here seems to be packet economy. The same

minimal two-packet exchange used over the LAN should be possible across the Internet. This leads to two requirements for supporting distributed operating systems:

* No Explicit Connection Setup or Teardown Phases;

* Implicit ("piggy-backed") Acknowledgments Whenever Possible.

  This implies that the response packet will serve as an implicit acknowledgment to the request packet (when timing makes this possible). Similarly, a new request (for the same pair of addressable entities) would implicitly acknowledge the previous response, if it came soon enough.

The nature of the application imposes two other requirements:

* Reliable ("at-most-once"), Ordered Delivery

  However, it should be possible to relax the reliability to take advantage of special cases like an idempotent request.

* Multicast Capability

  The transport service should mesh cleanly with the proposed Internet multicast facility, using host groups [ChDe85].

5. OBJECTIVES FOR A PROTOCOL

We believe that it is possible to design a new transport protocol for the Internet which is suitable for a wide variety of transaction-oriented applications. Such a transport protocol would have the following attributes:

* Reliable Delivery

  Data will be delivered reliably, i.e., exactly once, or the sender will be informed. The protocol must be able to handle loss, duplication, and reordering of request and response packets. In particular, old duplicate request packets must not cause erroneous actions.

  It should also be possible for the application programs to request that the reliability be relaxed for particular transactions. This would allow communication economies in the case of idempotent requests or of notification without reply.

* Minimum Number of Packets in Simple Cases

  In the simplest case (small messages, no packet losses, and the interval between requests and replies between the same pair of addressable entities shorter than applicable timeouts), a simple two-packet exchange should result.

* No Explicit Connection Setup or Teardown Phases

  The protocol will not create virtual circuits, but will provide what is sometimes (confusingly) called "reliable datagram" service.

  However, in order to provide a minimum two-packet exchange, there must be some implicit state or "soft" virtual circuit between a pair of addressable entities. In recent discussions this has been dubbed a "conversation", to distinguish it from a connection.

* Minimal Idle State

  When a server is not processing a transaction, there will be no state kept (except enough to recognize old duplicate packets and to suppress unneeded ACK packets).

* Fragmentation/Reassembly of Large Messages

  There is a range of possible objectives here. The minimum requirement is that the application not have to know the number 576, 548, etc. For example, each application might establish its own "natural" upper limit on the size of a message, and always provide a buffer of that size [3].

  At the other extreme, the protocol might allow very large messages (e.g., a megabyte or more). In this case, the proposed protocol would, in the large-message limit, be performing the bulk data transfer function of TCP. It would be interesting to know whether this is possible, although it is not necessarily a requirement.

  The introduction of multi-packet messages leads to the complex issues of window sizes and flow control. The challenge is to handle these efficiently in the absence of connection setup.

* Message Orientation

The basic unit of communication will be an entire message, not a stream of bytes. If a message has to be segmented, it will be delivered in units of segments or buffers, not bytes.

* Multicast Capability

Based on this discussion, we can suggest some of the key issues and problems in design of this protocol.

* Choice of Addressable Entity

  What will be the addressable entity? It must be unique in space; must it be unique in time (even across system crashes) ?

* Timeout Dynamics

  Timeouts must be the key to operation of this protocol. Experience with TCP has shown the need for dynamic selection of an appropriate timeout, since Internet delays range over four decimal orders of magnitude.

  However, the absence of connection setup and the typically-short duration of a single interaction seem to preclude the dynamic measurement of delays.

* Multi-Packet Messages

  How can flow control be provided for multi-packet messages, to provide reasonable throughput over long-delay paths without overrun with short-delay paths, when there is no virtual circuit setup?

* Implementation Efficiency

  The protocol should lend itself to efficient (which probably implies simple) implementations, so that hosts will be willing to use it over LAN's as well as for general Internet communication.

We believe further study is needed on these questions.

The reader may wonder: how is the proposed protocol related to an RPC (Remote Procedure Call) facility? The intent is that RPC facilities and message-passing IPC facilities will be built on top of the proposed transport layer. These higher-level mechanisms will need to address a number of additional issues, which are not relevant to the communication substrate:

1. Application Interface

   This includes binding and stub generators.

2. Structured Data Encoding

3. Server Location and Binding

4. Authentication and Access Control

6. CONCLUSION

Distributed processing and distributed data bases will underlie many of the future computer system research projects and applications based upon the Internet. As a result, transaction-based communication will be an increasingly important activity on the Internet. We claim that there is a pressing need for an appropriate transport protocol for transaction processing. In this memo, we have given examples to support this claim, and have outlined the service which such a new transport protocol would provide.

This memo is based upon discussions within the New End-to-End Protocols taskforce, and it is a pleasure to acknowledge the participation and sagacity of the members of that group. I want to thank Dave Clark, an ex officio taskforce member, for his contribution to these discussions, and Robert Cole for very helpful suggestions.

NOTES:

[1] For the purposes of this RFC, in fact, the reader may consider "transport service" to be defined as that protocol layer which contains TCP and UDP, as in Figure 1 of RFC-791. Alternatively, we may use the ISO definition -- the transport service is the lowest layer providing end-to-end service which is essentially independent of the characteristics of the particular (Inter-) network used to support the communication.

[2] This idea is implicit in the ISO definition of a transport service.

[3] It would be reasonable for the name server definition to specify an upper bound on the size of a single query or response; e.g., 2K bytes. This would imply (large) limits on the number of RR's that could be returned per response. If that limit is exceeded, we are doing something wrong!

REFERENCES

BiNe84 Birrell, Andrew D. and Nelson, Bruce Jay, "Implementing Remote Procedure Calls". ACM TOCS, Vol. 2, No. 1, February 1984.

ChDe85 Cheriton, David R. and Deering, Steven, "Host Groups: a Multicast Extension for Datagram Networks". To be presented to 9th Data Communication Symposium.

Cher85 Cheriton, David R., "V Message Transaction Protocol". Private communication, July 1985.

Cour81 Xerox Corp., "Courier: The Remote Procedure Call Protocol". XSIS 038112, Xerox Corp., Stamford, Conn., December 1981.

Coop84 Cooper, Eric C., "Circus: a Replicated Procedure Call Facility". Proc. 4th Symposium on Reliability in Distributed Software and Database Systems, October 1984.

Crow85 Crowcroft, Jon, "A Sequential Exchange Protocol". Internal Note 1688, Computer Science Department, University College London, June 1985.

Gurw85 Gurwitz, Robert F., "Object Oriented Interprocess Communication in the Internet". Private communication, April 1985.

Mill85 Miller, Trudy, "Internet Reliable Transaction Protocol -- Functional and Interface Specification". RFC-938, February 1985.

Shel85 Sheltzer, Alan B. , "Network Transparency in an Internetwork Environment", PhD Thesis, UCLA Department of Computer Science, July 1985.

Wats81 Watson, Richard W., "Timer-based Mechanisms in Reliable Transport Protocol Connection Management". Computer Networks, Vol. 5, pp47-56, 1981 (also distributed as IEN-193).