LNTP - An Efficient Transport Protocol for Local Area Networks

Samuel T. Chanson, K. Ravindran & Stella Atkins

February 1985
Technical Report 85-4

ABSTRACT

As interests in local area networks (LANs) grow so is the demand for protocols that run on them. It is convenient and a common practice to adopt existing transport protocols that had been designed for long haul networks (LHNs) for use in LANs. DARPA's Transmission Control Protocol/Internet Protocol (TCP/IP) for example, is available in 4.2 BSD UNIX for interface to Ethernet and other LANs. This is not desirable from a performance standpoint as the control structure is usually much more complex than is necessary, and LANs and LHNs have very different characteristics. Though there exists simpler transport protocols such as the User Datagram Protocol (UDP), most do not provide adequate flow control which, because of the much higher channel speed is critical in the LAN environment.

This paper discusses the unique characteristics and requirements of LANs and describes a new transport level protocol (LNTP) specifically designed for use on LANs. The fundamental philosophy in the design of LNTP is simplicity. Any features irrelevant to the LAN environment is not included. As well, LNTP uses a simple but effective deferred flow control mechanism which is activated only when the traffic intensity exceeds some value. This protocol has been implemented and runs under 4.2 BSD UNIX in place of TCP/IP. Detailed comparisons between LNTP, TCP and a few other protocols are given in the paper. Measurement data showed an improvement in network throughput rate of at least 30% over that of TCP. The problem of internet communication is also addressed.

LNTP - an Efficient Transport Protocol for Local Area Networks

Samuel T. Chanson, K. Ravindran & Stella Atkins

Dept. of Computer Science,
University of British Columbia.
Vancouver, B.C., Canada V6T 1W5

1.0 Introduction.

Local area networks (LANs) have soared in popularity in the last few years. As a consequence, there is increased demand for protocols that run on LANs. It is convenient and a common practice to use transport level protocols designed for long haul networks (LHNs) in a LAN environment. For example, the Transmission Control Protocol (TCP), which is a transport layer protocol designed for ARPANET is available in 4.2 BSD UNIX for interface to Ethernet and other local area network technologies. This is not desirable from a performance standpoint since the control structure is usually much more complex than is necessary for LANs resulting in unnecessary overhead. This is because LHNs often have to deal with unreliable and hostile subnets. Furthermore, the underlying network is of low bandwidth (typically 9.6 Kb/sec) constituting the slowest component in a transport connection. Thus the packet transfer rate is critically dependent on the capacity of the physical transport media rather than the transport protocol. As a consequence performance is seldom a major consideration in designing protocols for LHNs.

LANs, on the other hand are characterized by high channel speed (typically 10 MB/Sec.), low transmission error rate and a controllable environment. Measurements on large scale Ethernet installations [16, 17] indicate that the underlying network is seldom a bottleneck. Since the bottleneck now shifts to the higher level protocols and LAN users are usually less tolerant of packet delay, the efficiency of these high level protocols becomes an important design issue.

We have carried out extensive measurement experiments on the performance of TCP as implemented in BSD 4.2 UNIX running on VAXes and SUN workstations connected by an Ethernet[11]. We concluded that much of the protocol overhead and error recovery mechanisms are unnecessary or unsuitable for local area communication and that the throughput and mean packet delay time can be significantly improved if the protocol is simplified. We designed and implemented a highly efficient protocol called LNTP (Local Network Transport Protocol) which takes into consideration the characteristics of LANs. LNTP runs under 4.2 BSD UNIX replacing TCP/IP for local communication. When packets are destined for other networks supporting TCP or some other protocol, the protocol can be implemented at the gateway. Since the majority of the packets in a LAN are for local consumption [16], this scheme greatly improves the network throughput rate as well as the mean packet delay time.

This paper describes the new protocol LNTP and compares it with TCP/IP and HDLC. Measured performance characteristics of LNTP and TCP are also presented.

2.0 Design philosophy of LNTP:

The fundamental philosophy in the design of LNTP was simplicity. In addition to improving understandability and ease of maintenance, the objective was to reduce the protocol processing time. Thus we set out to design a new protocol that only included features strictly required in a single LAN operating environment. The design was guided by two considerations:

(i) What protocol features are *not* to be included ?

Our decision was that any feature irrelevant to a single LAN environment should not be included. Functions that are needed only in rare occasions, particularly if they can be easily achieved by the application programs are

also not to be included.

(ii) What features are to be included and/or emphasised ?

Our decision was that the characteristics of the underlying network and that of the application environment are to be exploited.

In the following discussions on LNTP, features that are included and/or excluded are cited in light of the above considerations. A complete description of the protocol in terms of specifications and state diagrams is given in appendix-A.

2.1 LNTP supported features

Broadly speaking, internet congestion and error control, routing, service options provided to high level protocols and certain functions that handle damaged packets found in a typical LHN protocol [4, 5] are not included as they are irrelevant in a LAN environment. Even checksumming is specified as an option since the error rates of the communication medium are negligible [8, 9, 10]. The only mandatory error control feature of LNTP is the selective retransmission scheme to handle packet loss due to buffer overflows. Consistent with the characteristics of a single LAN environment, the peer address (16-bit address space) does not include internet component. Furthermore, since the probability of out-of-order delivery and duplicates is negligible [14], the sequence number space is made small (4-bits).

To reduce the state information required to maintain a connection (which simplifies the control structure leading to reduced processing overhead), the sender and receiver are logically separated and the send and receive channels are completely decoupled. A consequence of this decision is that a receiver is unable to piggyback control information on reverse data packets to the sender, a feature commonly supported by LHN protocols [4,6] to conserve communication bandwidth. However, network bandwidth is not a scarce resource in a LAN.

Moreover, our measurement results show piggybacking is rare due to the unavailability of reverse data packet at the right time. Thus in view of the simplicity and reduced processing overhead, LNTP is asymmetric in send and receive.

2.2 Flow control

Since LAN bandwidth is several orders of magnitude higher than that of LHN, network load seldom exceeds 40% of the network capacity even for very large scale LANs with heavy usage. Measured figures on an Ethernet installation [16] consisting of 120 Alto machines, two time-sharing systems (TENEX), gateways, file and print servers showed that the maximum load as observed over a 6-minute interval was 7.9% and the average load was 0.8% of the Ethernet capacity of 2.94 Mb/sec. The corresponding figures over an observation interval of 1 second were 32.4% and 2.7% respectively. [17] reported measurement data on an existing university time-sharing environment extrapolated to that for a large scale Ethernet-based LAN supporting the same environment. The network utilization for 1000 users with heavy network disk usage was no more than 30%.

Thus the physical transport channel itself is almost never the bottleneck. The only delay is virtually the propagation delay which is small compared to protocol processing times. Thus flow control is a strong requirement for LAN end-to-end protocols. Without it, it is easy for a fast machine to swamp a slower machine. The flow control in LNTP was designed with this in mind and forms an integral part of the protocol specification. In contrast, flow control is seldom a central design issue in LHN protocols.

Our measurements on TCP/IP over Ethernet [11] also indicated that the sender was frequently blocked for want of window space; furthermore, the amount of control packet traffic was high (up to 50% of the packets were control packets). The flow control mechanism in LNTP maximises the parallel operations of the sender and receiver, and minimises the number of control packets

that has to be exchanged. The provision to send/accept packets even in the PRMPT_SENT/HOLD_SENT (see appendix-A for LNTP specifications) states reduces the probability of choking the protocol. The concept of threshold in the window space reduces the control traffic (no control before the threshold is exceeded). Because of the deterministic nature of packet delays, a mathematical model can be formulated for a proper threshold to be set as a function of the system parameters to maximise the network throughput rate.

2.3 Timer structure

Every outgoing data packet initiates a timer which is reset either by the next outgoing data packet (the timer is also restarted) or by the arrival of an acknowledgement. When the timer expires (T_s), the packet is retransmitted. There is an associated timer at the receiver which is reset and restarted by every incoming packet; on time out (T_r), an explicit acknowledgement is sent. The timer values T_s and T_r must be chosen to satisfy the inequality

$$T_s > T_r + E_{tt(dat)} + E_{tt(ack)}$$

where E_{tt} is the packet end-to-end transfer time. This timer scheme has lower overhead than the traditional one in most protocols. For stream traffic, the next outgoing/incoming packet resets T_s/T_r so that no extra network traffic is generated. For interactive type traffic, the timers guard against possible deadlocks that might arise due to packet loss. This simpler scheme is not suitable for LHNs because the variance of E_{tt} is high in such systems necessitating the choice of a large value for T_s which makes the scheme inefficient. In any case, a tight inequality is less meaningful in multihop networks. Another motivation behind the design of this timer structure is that management of multiple logical timers though usually realised by a single physical timer is more complex than managing a single timer. This is because each timeout event results in a complex sequence of actions [3]. Thus LNTP implements a single logical timer; in contrast, almost

all other protocols specify a separate timer for each outstanding packet though some implementations cut corners in this.

3.0 Comparing TCP and LNTP.

This section describes the major differences in between TCP/IP and LNTP. The differences directly lead to a simpler structure and lower protocol overhead for LNTP.

3.1 Process addressing.

TCP [4] was designed to handle internetwork communications. The address of the communicating processes reflect this, being characterized by the triple

<process port number, host address, network address>.

The local address component (process port number) is 16 bits while the internet component is 32 bits for a 48-bit address. LNTP uses a 16-bit process address that is assigned uniquely to communicating processes across the entire LAN. The underlying address mapping mechanisms might either subdivide this address into

< Local process port number, Host number >

or use this address as an index into a table providing the above information; the subdivision is installation-dependent. For example, in a LAN consisting of 30 nodes, a 5-bit host number field could be used which means that up to 2048 process ports may be assigned for LNTP in a single node.

However, in the current implementation of LNTP under UNIX 4.2 BSD, a 24-bit address field is used consisting of

< 16-bit process port number, 8-bit host number >

to ensure compatibility with UNIX's assignment of 16-bit process port ids.

3.2 Packet fragmentation and reassembly.

The Internet Protocol (IP) [5] which provides network layer support to TCP offers network-dependent data fragmentation to match the underlying network, and the subsequent reassembly procedures at the TCP input level. The reassembly procedures involve examination of the IP header fields to identify the fragment numbers to facilitate in-place assembly, detection and elimination of reassembly deadlocks, and coalescing the fragments to enable high level processing.

For internal communication within a single LAN, the underlying physical network characteristics are known and fixed. Hence, for efficiency reason the fragmentation/reassembly feature was not included in LNTP where segments are generated directly in sizes that match those of the physical network.

3.3 Sequence number management.

TCP uses a byte level sequence number scheme which entails two types of overhead:

(i) A large sequence number field (32 bits) in the segment.

(ii) High overhead in sequence number management.

Typically, when a segment arrives, checks are made to determine if the sequence number space of the segment overlaps that of the previous one or if there is a gap between the two sequence number spaces. Furthermore, the segment length implied by the sequence number must tally with that specified by the IP level call.

The above overhead is eliminated in LNTP by providing small, packet level sequence numbers (4-bit sequence number). This replacement is justified on the following grounds:

(i) The advantage of a large sequence number space in TCP is to provide a large recycling interval to easily identify duplicates and out-of-order deliveries common in multihop networks, but highly unlikely in a single LAN [14].

(ii) Since a TCP segment can potentially be fragmented at the local IP layer and the intervening gateways and since TCP does not provide segment level identification, a TCP segment can potentially be delivered as multiple segments at the remote peer. So a byte level identification makes the process of coalescing the segments into a stream easier. In a single LAN, such features are seldom needed.

(iii) Byte level sequence number based windowing might lead to small windows resulting in excessive fragmentation of the stream [11] unless a reasonable lower limit is placed on the segment size.

We note, however, that byte level sequence numbering maps well to provide a byte stream interface to the client. A packet level sequence numbering scheme such as that used in LNTP requires mapping of the packets into byte streams and vice versa at the client - protocol interface should the client require a stream interface. However, this mapping is performed at one point only and the overhead involved is relatively low.

3.4 Error Control.

The error rate in LANs is extremely low, of the order of 1 in 10^{11} to 10^{12} [9, 10]. The only common type of packet loss is the discarding of packets due to buffer overflows. LNTP provides selective retransmission based on the RETXMT control packet which handles this type of packet loss efficiently. On the other

hand, TCP provides a more general timeout-based retransmission scheme which is more complex.

TCP/IP uses double checksums, one in the TCP and one in the IP layer. The TCP level checksum is provided for end-to-end error control while the IP level checksum enhances internet datagram reliability. LNTP uses an optional 16-bit checksum that can provide end-to-end error detection capability between the transport level entities should the system designer choose to include it in the packet structure.

TCP/IP also provides a self-destruction mechanism for each packet in the form of a time-to-live (TTL) option. This option prevents a packet sent long ago from suddenly reappearing, fouling up the current data stream. A TTL field is specified (typically two minutes) when a segment is created , and is decremented at each network hop. When this field reaches zero, the packet is destroyed. Such a feature is useful only in a multihop LHN, and so is not incorporated in LNTP.

3.5 Data stream options.

TCP provides two options (URG and PUSH) to handle specialized data streams. Applications requiring these facilities occur infrequently in a local area network environment. Furthermore, clients requiring such service can easily build his own. None of these options are incorporated in LNTP.

TCP/IP also provides for options offering different qualities of service. This is specified in terms of traffic priority, delay and throughput characteristics that can be tolerated by the user. Based on these service options, packet handling and service scheduling at the intermediate hops are determined. These features are relevant only in LHNs and are not incorporated in LNTP.

3.6 Flow control.

In contrast to the elaborate window management scheme used in TCP [4], LNTP provides three flow control packets whose functions are simple and straightforward:

PRMPT: request information on the latest in-sequence packet received.

HOLD(n): stop sending (acknowledges up to packet n-1).

RESUME(n): start sending (acknowledges up to packet n-1).

TCP uses a data segment in an indirect way to solicit an acknowledgement from the receiver. It sends a dummy segment by deliberately placing the sequence number of the octet outside the receive window. On discovering that the sequence number falls outside of its window, the receiver sends a window update to "correct" the sender. These complexities are avoided by the clean and tight semantics attached to the LNTP control packets.

The flow control of LNTP works as follows. Thresholds are defined in the windows of the sender and the receiver. On the sender side, as long as the number of outstanding packets is less than its threshold (which should be true most of the time in normal operation) the sender does not initiate control. Once the threshold is exceeded at the availability of the next data packet, the sender initiates control by piggybacking the PRMPT control on the data packet. Similarly on the receiver side, as long as the number of packets waiting to be consumed is below its threshold, the receiver does not acknowledge (unless an out-of-order packet arrives, which causes a RETXMT(n) to be sent, or unless the receiver is prompted by the sender). This *deferred flow control* scheme reduces the volume of control packets flowing in the network. When its threshold is exceeded, the receiver sends a HOLD(n) to suspend transmission. The receiver accepts any packets that might be in transit before the HOLD control arrives at its destination. This feature maximizes the parallel operations of the sender and the receiver and minimizes packet loss. Furthermore, the ability of a heavily loaded receiver (by unrelated host activities) to directly suspend the sender from sending more packets provides a certain degree of adaptation to the load at the

receiver's end. When the receiver exits from the flow-controlled region, a hysteresis is provided in the sequence number space to absorb the sudden surge of packets that might be sent by the sender when it is unblocked. This eliminates the possibility of the receiver oscillating in and out of the flow-controlled region, a phenomenon commonly known as the ping-pong effect.

Such explicit measures for flow control are notably absent from TCP. In fact, the receiving TCP silently discards packets when its receive window is full unless the sender solicits a window update.

4.0 Comparison of LNTP and HDLC (High Level Data Link Control).

Even though LNTP and HDLC, level 2 of X.25 as adopted by CCITT [6] operate at different levels in the ISO reference model of Open Systems Interconnection [12] (LNTP at the transport level and HDLC at the link level), there are common grounds with respect to their internal structures on which comparisons can be made.

HDLC provides an error-free point-to-point link on top of an imperfect physical channel by providing link level error and flow control. This link is used by the packet layer (level 3 of X.25) to manage logical channels. As a link level protocol, HDLC does not provide logical channel numbers, and the packet layer extends HDLC to provide logical channels between nodes. However, the error and flow control functions as seen by the network client (e.g., transport layer) are in fact built upon those of HDLC with the packet layer providing its own channel level flow control. A transport layer should operate on top of X.25 to provide end-to-end error and flow controls for each virtual circuit independently since an X.25 packet could traverse a series of nodes before reaching the transport station. Thus, in effect, there are multiple levels of error and flow control, both stepwise and end-to-end, in X.25 based systems. This is a significant difference between the X.25 and LAN environments.

In a LAN environment, most of the packets reach the receiving transport station in a single hop except for a small number of internet packets. A single layer of error and flow control at the transport level is therefore sufficient to maintain end-to-end reliability.

Having pointed out the structural differences between X.25 and LAN-based systems, let us compare HDLC and LNTP with respect to their internal structures (and not their functional services). (N.B., X.25 and HDLC are used interchangeably in this paper as X.25 merely extends the services provided by HDLC.)

(i) Packet size.

X.25 uses 128 or 256-byte packet length whereas LNTP packets can be up to 2048 bytes (the length is specified as part of the header). The larger packet size is used since packet transport efficiency is directly related to the packet size [11].

(ii) Error control.

X.25 error control is built upon timer-based and receiver-initiated retransmissions. The arrival of an out-of-sequence packet provokes a REJ(n) frame from the receiver which triggers retransmissions of *all* frames from sequence number n. A sender timeout associated with a frame 'n' again triggers retransmissions. Though the earlier versions of HDLC supported SREJ(n) (selective reject) the CCITT-adopted version does not support it. LNTP supports a timer-based retransmission and a selective retransmission (by means of the RETXMT(n) packet) schemes.

(iii) Flow control.

HDLC uses the standard sliding window scheme for flow control. It does not exchange windows at any time but uses fixed window sizes at the sender and receiver (typically 2 and 1 respectively). LNTP too, as can be inferred from Section 3.6, uses a form of the sliding window scheme but takes the full sequence number space as the window sizes at both the sender and receiver. Windows (range of valid sequence numbers)

are not explicitly exchanged even though the dispatch of a RESUME after a HOLD would allow the sender to estimate the receiver's window. Note however that the window of acceptable sequence numbers increases as packets are consumed by the client at the receiver and, that unlike most sliding window schemes, the sender is not constrained by the receiver's window size that *was* indirectly available to it when a RESUME arrived; instead, the activities initiated by the sender are solely based on its local window. This scheme increases the degree of parallel operations at the sender and receiver nodes. The threshold scheme also allows hysteresis to be implemented easily by setting a lower threshold when the state goes from HOLD_SENT to NORMAL. This avoids frequent ping-ponging between the two states.

5.0 A word about NCP (Network Control Protocol).

NCP [7] is a transport level protocol that was used in the initial versions of ARPANET. It is no longer being supported. NCP assumes an underlying reliable subnet on which it provides the notion of connections between end points. It does not use any acknowledgement, and assumes that the packets injected into the subnet will arrive error-free and in the right order to the end stations. Significantly, it lacks flow control. (The buffer allocation mechanism is done on a host-to-host and not on a per connection basis.)

In a LAN environment, though damaged packets can be practically ruled out, flow control is an important requirement. A fast sender, for example, can overflow a slow receiver. Thus any possible adaptation of NCP for LANs requires a flow control mechanism to be added.

6.0 LNTP implementation

6.1 Implementation structure

The protocol describes the one way flow of information from one peer to another. The complete transport layer consists of an implementation of two complementary instances of the protocol running almost independently with the interactions between the two occurring in the disconnect phase. The bidirectional session level interactions are handled by the two instances independently in terms of error and flow control. In other words, there are two logical channels below the session level interface each handling data flow in one direction.

On a close-down request at the session level, the protocol instance that handles the sending at the local end of the connection enters the DISCONNECTING state after a series of operations described in section A.2.3, then sends a DREQ to the remote peer. The receiver at the other end sends a DCONF, and the protocol moves to a STOP state. At the same time, the remote sender starts a disconnect operation going through identical steps. When the other protocol instance also moves to a STOP state, the session level disconnect is confirmed.

The generalised implementation structure is given in Figure 6.1. The client resides above the session layer interface. The functions of the multiplexor are

(i) Connection set-up.

(ii) Creation of $LNTP_s$, $LNTP_r$ and the demultiplexor on successful connection set-up.

The function of the demultiplexor is to route incoming packets to $LNTP_s$ or $LNTP_r$ based on the packet type.

In UNIX 4.2 BSD [1], the session level interface is provided in the form of abstract objects called *sockets* that are supported by type-specific or client-specified protocols. Three types of sockets are supported:

(i) *Stream sockets*

Supported by TCP/IP. LNTP support has been added to this type of sockets.

(ii) *Datagram sockets*

UDP (User Datagram Protocol) provides the transport service above IP. Both are datagram protocols.

(iii) *Raw sockets*

Supported by IP.

Figure 6.2 gives the protocol implementation environment provided in UNIX 4.2 BSD together with the interactions between the protocol and the various system modules. All vendor supplied protocols are implemented within the kernel for performance reasons. The performance tradeoffs involved in alternative implementation structures are discussed elsewhere [3]. LNTP too is implemented within the kernel at the same functional level as TCP/IP (Figure 6.3); and the implementation makes extensive use of the timer facilities, the queue disciplines and the memory management features provided by the kernel [2].

6.2 Protocol performance

A preliminary implementation of LNTP in the 4.2 BSD UNIX kernel running on a SUN workstation has been made. The protocol was tested in a software loopback mode, and its performance compared to TCP/IP running under identical conditions [11]. In this mode of operation, the client and the server processes reside on the same machine. A steady stream of data flows from the client to the server. UNIX 4.2 BSD supports a software loopback interface at the network interface level which mimicks a physical network and echoes back the data to the receiver. Figure 6.4 compares the maximum throughput rate (as a function of packet sizes) for TCP/IP and LNTP. All system and workload parameters were identical. The results show LNTP to be better by about 30%. This improvement is basically due to the simplicity in the control structure resulting in lower protocol overhead for LNTP. However, since flow control cannot be adequately tested in the software loopback mode, the full impact of the new design cannot be

determined until a machine-to-machine transfer is effected. This requires Ethernet support for LNTP. The implementation is under way.

7.0 Future plans

Ethernet support is being provided by assigning a unique identifier for LNTP in the Ethernet type field, and providing an appropriate protocol scheduler. The performance of the protocol for both stream and interactive applications will be studied. The contributions to performance improvement due to simplicity, modified flow control and control traffic will be studied. The deterministic model for flow control developed in [11] will be extended to compute the optimal threshold values.

Our long term plan is to create a separate address domain exclusively for local network communications. Currently, UNIX 4.2 BSD defines many types of address domains in which sockets reside [1]. The most important of them are

(i) *UNIX sockets*

Sockets created in this domain are for local interprocess communication (IPC), and are implicitly stream type sockets. UNIX internal protocols are used for the operation of these sockets.

(ii) *INTERNET sockets*

Sockets created in this domain are for remote IPC (though in principle, this can be used for local communications as well). They are supported by TCP/IP and UDP.

In the current implementation, LNTP is used as another protocol support for the internet sockets. This means that internet addresses are generated by the socket layer, but are not fully used by LNTP since the packets are assumed to be local. To remove the anamoly between the semantics of internet sockets and the actual support provided by LNTP, the new address domain will provide address

generation for hosts on the local network only. The sockets in this address domain will be supported by LNTP while internet sockets will continue to be supported by TCP/IP and UDP. The proposed structure is shown in Figure 7.1.

8.0 Conclusions

We have described LNTP, a new transport protocol for LANs. This protocol was designed with the characteristics of LANs in mind. Consequently, it is far more efficient than protocols originally designed for LHNs but adopted to be used on LANs. The design philosophy of LNTP was outlined, and the protocol itself was compared with some other popular protocols. A preliminary version of LNTP was implemented in 4.2 BSD UNIX, and its performance under the software loop-back mode was measured and compared to that of TCP/IP running under the same environment. Initial results showed about 30% improvement in maximum throughput rate.

Since LNTP is a protocol supporting communications within the local network only, packets intended for other networks must use an internet protocol such as the IP [5] at the gateway. Since the majority of the packets in a LAN are for local consumption, this scheme improves the overall network throughput rate.

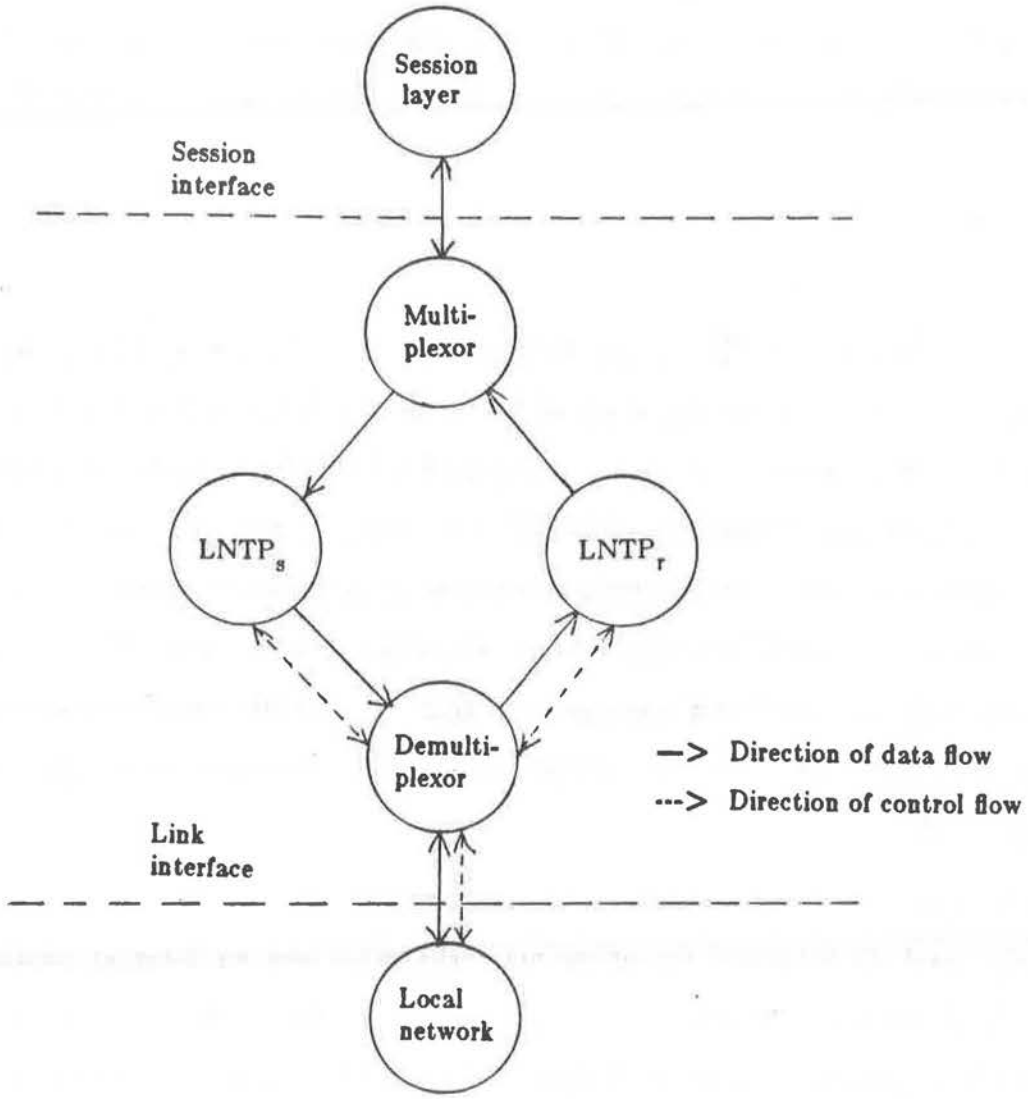


Figure 6.1 A generalised implementation structure.

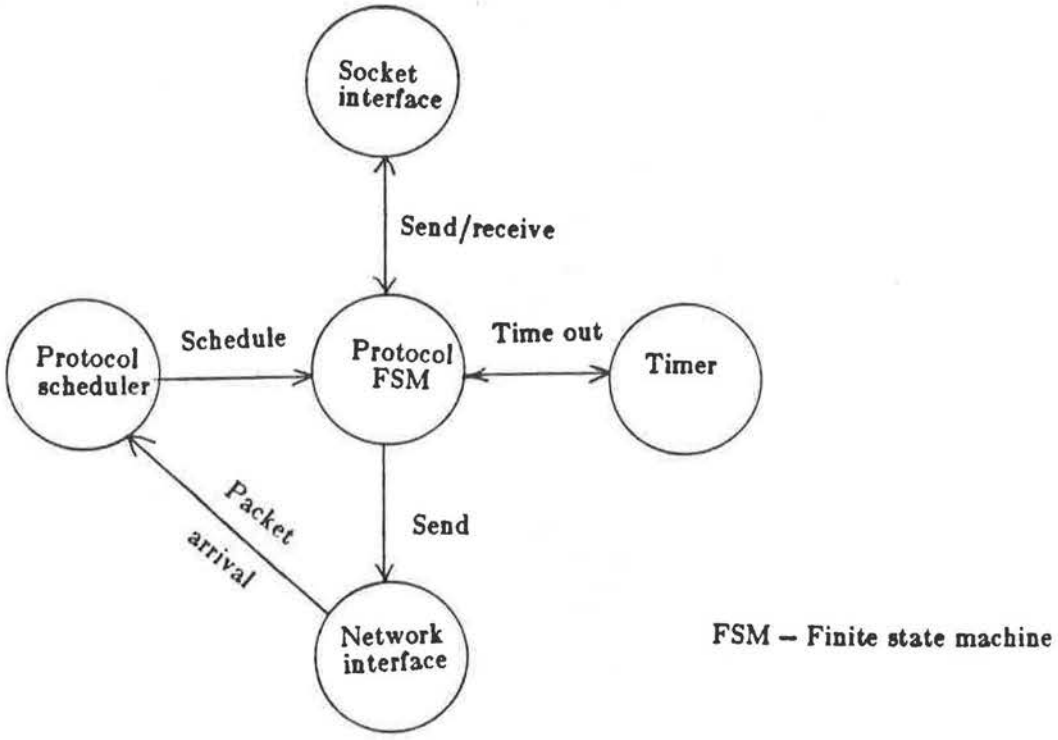


Figure 6.2 Protocol implementation environment provided by UNIX 4.2 BSD

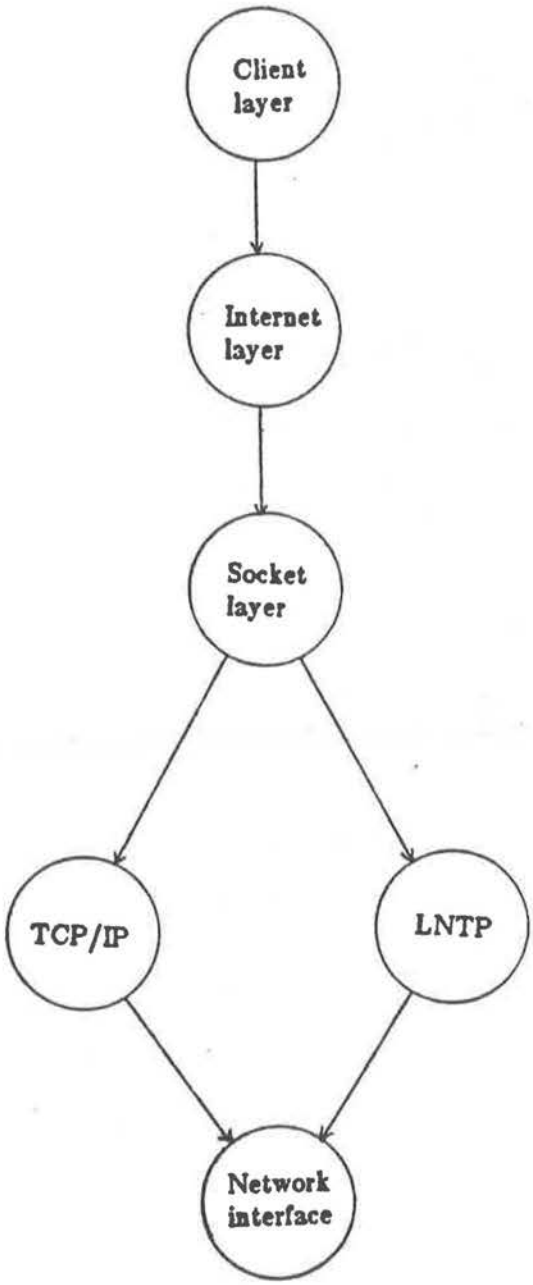


Figure 6.3 Implementation structure of LNTP in UNIX 4.2 BSD

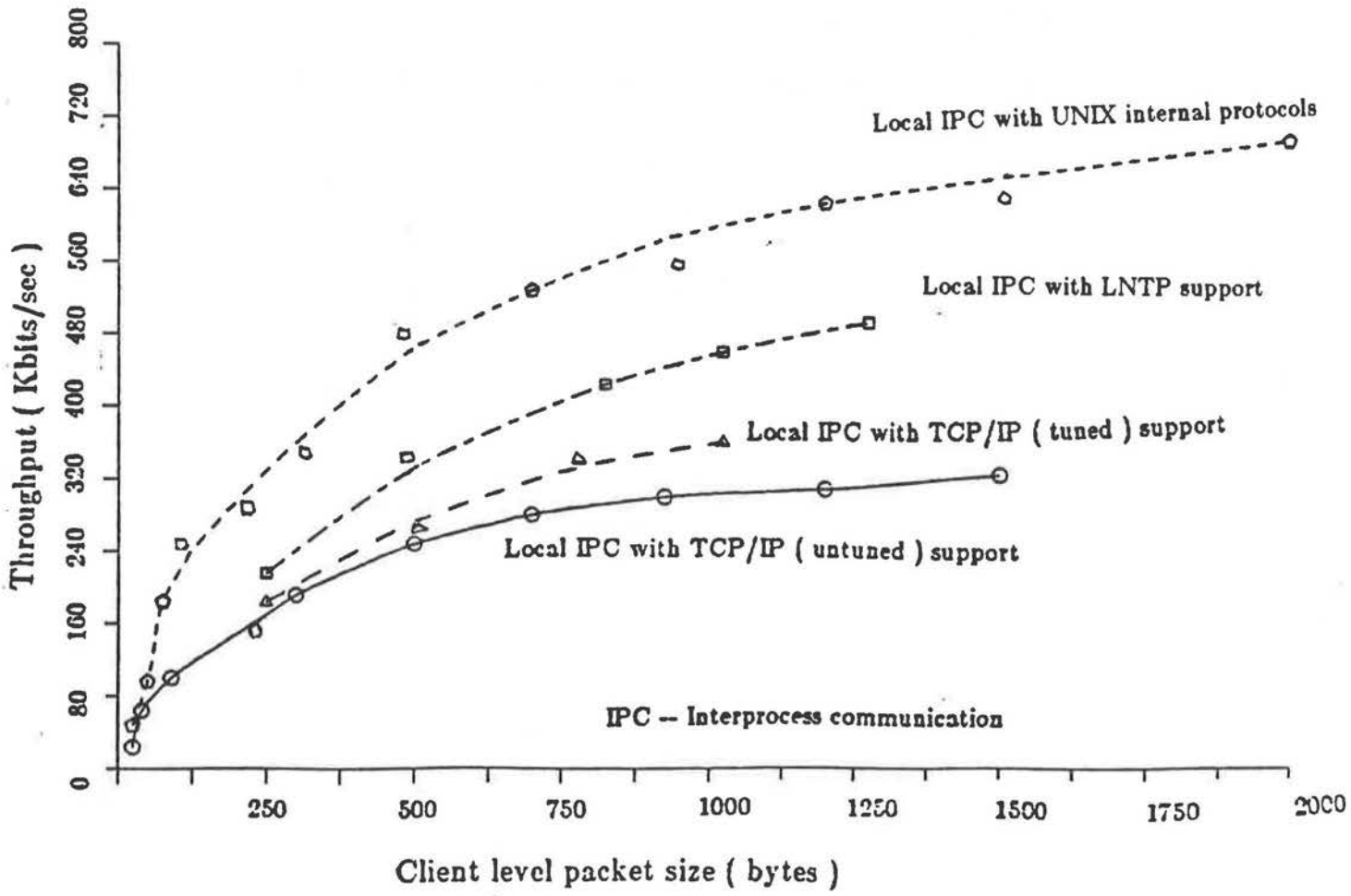


Figure 6.4 Performance comparison between LNTP,
TCP/IP and UNIX internal protocols

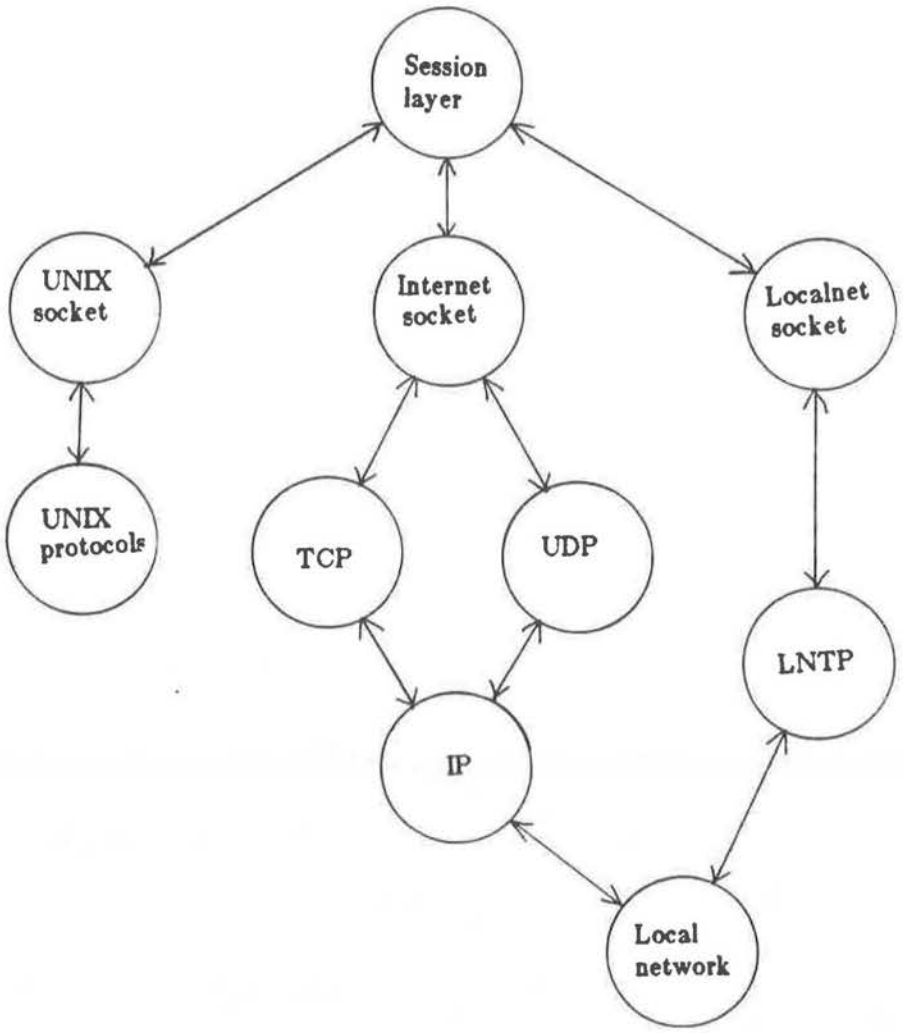


Figure 7.1 Proposed address domain for LNTP

References

1. S.J.Leffler, R.S.Fabry, and W.N.Joy, *A 4.2 BSD Inter process communication primer,* Comp. Syst. Res. group, Univ. of California, Berkeley, Draft of Mar. '83.

2. S.Leffler, W.Joy, and R.Fabry, *4.2 BSD Networking implementation notes,* Comp. Syst. Res. group, Univ. of California, Berkeley, Draft of Sept. '83.

3. D.D.Clarke, *Modularity and efficiency in protocol implementation,* RFC 817, MIT Lab. for Computer Science, July '82.

4. DARPA Internet program protocol specifications, *Transmission control protocol,* RFC 793, Information Sciences Institute, USC, CA, Sept. '81.

5. DARPA Internet program protocol specifications, *Internet protocol,* RFC 791, Information Sciences Institute, USC, CA, Sept. '81.

6. Intl. Telephony and Telegraphy Consultative Committee, *Recommendations X.25/interface between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) operating in packet mode on Public Data Networks,* CCITT Orange Book, Vol.7, '80.

7. J.B.Postel, *An informal comparison of three protocols,* Computer Networks, North-Holland Pub. Co., vol.3, '79, pp.67-76.

8. R.Metcalfe, and D.Boggs, *Ethernet: distributed packet switching for local computer networks,* CACM, vol.19, no.7, July '76, pp.395-404.

9. D.Clark, K.Pogran, and D.Reed, *An introduction to Local Area Networks,* IEEE Proc., vol.66(11), Nov.'78, pp.1497-1517.

10. I.W.Cotton, *Technologies for Local Area Computer Networks,* Computer Networks, North-Holland Pub. co., vol.4, '80, pp.197-208.

11. S.T.Chanson, K.Ravindran, and S.Atkins, *A performance evaluation of the ARPANET TCP in a LAN environment,* accepted for publication in INFOR journal.

12. Intl. Organization for Standardization, *Reference model of Open Systems Interconnection,* ISO/TC97/SC16/ Draft Intl. Standard, ISO/DIS/7498, '82.

13. D.R.Cheriton, *Local networking and internetworking in the V-system,* Proc. of the 8th Data Commn. Symp., ACM SIGCOMM, Oct.'83, pp.9-16.

14. J.A.Stankovic, *A perspective on Distributed Computer Systems,* IEEE Trans. on Computers, vol.C-33, no.12, Dec.'84, pp.1102-1115.

15. A.Z.Spector, *Performing remote operations efficiently on a Local computer network,* CACM, vol.25, no.4, April 1982, pp.246-260.

16. J.F.Shoch, and J.A.Hupp, *Measured performance of an Ethernet Local computer network,* CACM, vol.23, no.12, Dec.1980, pp.711-721.

17. M.Marathe, and B.Hawe, *Predicted capacity of Ethernet in a university environment,* Introduction to Local area Networks, DEC publication, 1982, pp.145-159.

18. A.S.Tanenbaum, *Computer Networks,* Prentice-Hall Inc., 1981.

Appendix-A

Description of LNTP

A brief description of LNTP is given below in terms of the packet structure and the functional components of the protocol. The finite state machine representation of the protocol during the different phases of a connection is given in the diagrams in Figures A.1 to A.5.

A.1 Packet structure

The structure of LNTP packets is denoted by a 'C'-like structure

```
pkt = struct
      {
      SRCADRS   16 bits ;
      DSTNADRS  16 bits ;
      PKTID      4 bits ;
      CHKOPT     1 bit  ;
      LEN       11 bits ;
      SEQNO      4 bits ;
      RETXCNT    4 bits ;
      DATA      <LEN> bytes ;
      CHKSUM    16 bits ;
      }
```

pkt.SRCADRS –> Port address associated with the sending peer

pkt.DSTNADRS –> Port address associated with the receiving peer

Usually these address fields decompose into a host identifier and a locally unique port identifier.

pkt.PKTID –> Packet identifier. The different types of packets are DATA, DATA + PRMPT, PRMPT, RESUME, HOLD, RETXMT, CREQ, CCONF, DREQ and DCONF.

pkt.CHKOPT --> Indicates whether the packet includes a checksum.

pkt.LEN --> Data length in bytes

pkt.SEQNO --> Sequence number of the data packet from sender. Wrap around occurs after every MAX_SEQNO packets. For a control packet sent by the receiver, the value of this field is one plus the sequence number of the in-sequence packet last received correctly.

pkt.RETXCNT --> Retransmission count of a data packet (zero initially).

pkt.DATA --> Data field of length pkt.LEN bytes

pkt.CKSUM --> 1's complement checksum of the entire packet, including header, if checksum option is exercised; else 0.

The variables describing the protocol machine are :

MAX_SEQNO --> Maximum sequence number space (16)

Sender:

SND.NXT --> Sequence number of the next packet to be sent

SND.UNA --> Sequence number of the first packet in the sender waiting to be acknowledged

N_{tx} --> Sender's threshhold in the protocol window

Two timers are defined (it is the same timer used differently at two different times):

(i) A timer is associated with the packets in the flow controlled region at the sender

(ii) Each of the outgoing packets at the sender also resets and restarts a timer (which expires after T_s) in the non flow-controlled region. The timer is reset (but not restarted) by an acknowledgement packet. Similarly there is a timer at the receiver which is reset and restarted by each incoming packet (the receiver timer expires after T_r). The expiry of T_s provokes a DATA + PRMPT from the sender. The expiry of T_r causes the receiver to send an acknowledgement. T_s and T_r must be chosen to satisfy the inequality

$$T_s > E_{tt(dat)} + T_r + E_{tt(ack)}$$

where E_{tt} is the packet end-to-end transfer time.

Receiver:

RCV.NXT -->	Sequence number of the next packet to be received
RCV.CNS -->	Sequence number of the first packet in the receiver waiting to be consumed
N_{tr1} -->	Threshold value when the window decreases
N_{tr2} -->	Threshold value when the window increases

$$(N_{tr2} < N_{tr1})$$

A.2 Functional components

The protocol is divided into three phases:

(i) Connection set-up

(ii) Execution

(iii) Close-down

A.2.1 Connection set up phase

The connection set up phase is fairly standard. A connect request may come from one of the two sources:

(i) local client as indicated by event CCR

(ii) Remote peer in a CREQ packet

The connection set up dialogue is best illustrated by the state diagram in Figure A.1. The timer CTMOUT handles loss of CREQ and CCONF packets. The peer that accepts the connection in response to CREQ sends a CCONF, confirming the connection.

A.2.2 Execution phase

This is concerned with the orderly transfer of data from the sender to the receiver. The figures in A.2 and A.3 give a detailed state diagram representation of this phase. Figure A.2 is the state diagram of the normal flow control in the execution phase, assuming no error or lost packets. Figure A.3 is the error control component superimposed on the flow control, and is initiated by the receiver when it receives an out-of-order packet.

The sending peer can be in one of three states :

(i) NORMAL state
(ii) PRMPT_SENT state
(ii) HOLD state

In the NORMAL state, only data packets flow to the receiving peer. The transition to PRMPT_SENT when the window size becomes less than N_{tx} provokes a PRMPT control packet. In this state the protocol can still accept packets from the client layer and send DATA + PRMPT packets. In the HOLD state, no packets are sent except those provoked by RETXMT(n) control packet.

The type of packets the sending peer may send are:

(i) DATA packets
(ii) DATA + PRMPT packet
(iii) PRMPT control packet

The function of the PRMPT packet is to force an acknowledgement packet from the receiving peer.

The receiving peer can exist in one of three states:

(i) NORMAL state
(ii) RETXMT_SENT state

(iii) HOLD_SENT state

In the NORMAL state, packets are accepted without generating any acknowledgement unless an out-of-order packet arrives which provokes an immediate RETXMT(n) control packet. The receiver continues to accept packets in the HOLD_SENT state until its window closes.

Three types of packets may be sent by the receiving peer:

(i) RETXMT(n) packet
(ii) RESUME(n) packet
(iii) HOLD(n) packet

where 'n - 1' is the sequence number of the last packet that has been received correctly. Their functions are:

RETXMT(n) --> Advises sender to retransmit packet of sequence number 'n'. This is sent when an out-of-order packet arrives at this layer. The out-of-order packet is buffered.

RESUME(n) --> Advises sender to resume transmission starting from the packet with sequence number 'n'. This is used when sufficient window space is available to reenable the sending peer. This is also sent in response to a PRMPT control packet when packets up to 'n - 1' have been received correctly and the receiver is expecting packet 'n'.

HOLD(n) --> Instructs the sending peer to hold transmission until instructed to resume. This is sent when the window size falls below a threshhold.

A.2.3 Close down phase

Before both peers can enter the DISCONNECTING state for the actual disconnect sequence, they must be brought to a consistent state. The protocol defines a FLUSHING state which is an intermediate state between the current state when the disconnect request arrives and the DISCONNECTING state. It is illustrated in Figure A.4. The disconnect request itself might originate from the local process (DRC) or from the remote peer (DREQ packet). There are three situations that the protocol might be in when a disconnect request arrives:

(i) Packets pending to be sent -
These are sent with PRMPT piggybacked on the last data packet.

(ii) No data pending but packets are being held waiting to be acknowledged.
PRMPT packets are sent to elicit acknowledgement.

(iii) No packet waiting to be sent or acknowledged.
In this case, the protocol immediately moves to the DISCONNECTING state, in which the sender sends a DREQ to the receiver and moves to the CLOSED state.

The protocol enters the FLUSHING state in cases (i) and (ii). The rest of the disconnect phase is standard, and handled in a way similar to the connection set-up phase. It is illustrated in Figure A.5. The peer which accepts the disconnect request sends a DCONF packet confirming disconnection. Timeout (DTMOUT) is provided to guard against loss of control packets.

A.3 Error recovery:

Recovery from various error conditions is performed as indicated :

Out-of-order packets -->RETXMT control packets are used to initiate selective retransmissions (missing packets are detected using sequence numbers assigned to packets).

Damaged packets -->	RETXMT packets (error is detected by checksum validation).
Duplicates -->	Discard packets (error is detected by RETX_CNT assigned to packets).
Lost packets -->	RETXMT packets (error is detected by timeouts provided in the different phases).

A.4 Flow control:

This is achieved by means of PRMPT packets from the sender, and HOLD and RESUME packets from the receiver.

The sender's window is defined as :

Sender's window size (SWS) = MAX_SEQNO - (SND.NXT - SND.UNA)

(N.B., modulo arithmetic is used in the computation of sequence numbers and window sizes with MAX_SEQNO as the modulus).

Two regions are defined :

$SWS < N_{tx}$ ... Region 1

$SWS >= N_{tx}$ ... Region 2

No flow control is initiated from the sender in Region 1. Once the sender enters Region 2 it initiates flow control measures while maintaining the data flow at the same time until the protocol stops when the window is filled.

The receiver's window is defined as :

RCV.NXT <= sequence number < RCV.CNS

Receiver's window size (RWS) = MAX_SEQNO - (RCV.NXT - RCV.CNS)

Two regions are defined for RWS:

$< N_{trl}$ ... Region 1

$>= N_{trl}$ ... Region 2

when in the NORMAL state, and

$< N_{tr2}$ ... Region 1

$>= N_{tr2}$ ... Region 2

when in the HOLD_SENT state

(In all cases, $N_{tr2} < N_{tr1} <$ MAX_SEQNO)

The point at which flow control takes effect

= [Time at which RWS crosses N_{tr1}]

or

[Time at which SWS crosses N_{tx}]

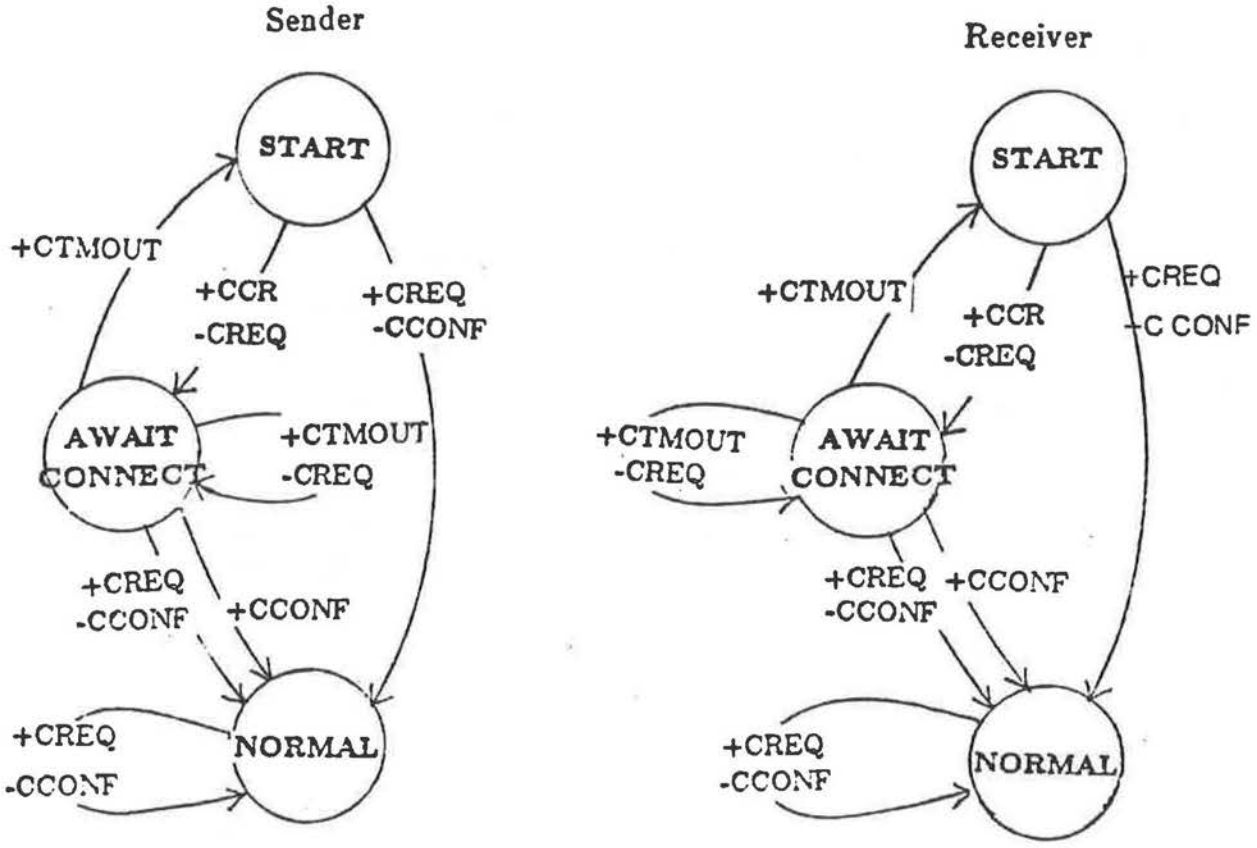
whichever occurs first

The hysteresis ($N_{tr1} - N_{tr2}$) is needed to absorb any transient surge in packet arrivals when the sender moves from the HOLD state to the NORMAL state, thereby avoiding any ping-pong effect.

A.5 Connection surveillance

This is provided by means of a simple asynchronous protocol running on top of LNTP at both the sender and receiver. The protocol periodically exchanges AYT (synonym for "Are You There?") control packets in a symmetric fashion to ascertain the connection is alive. Absence of AYT from the other end after repeated tries results in a forcible shutdown of the connection.

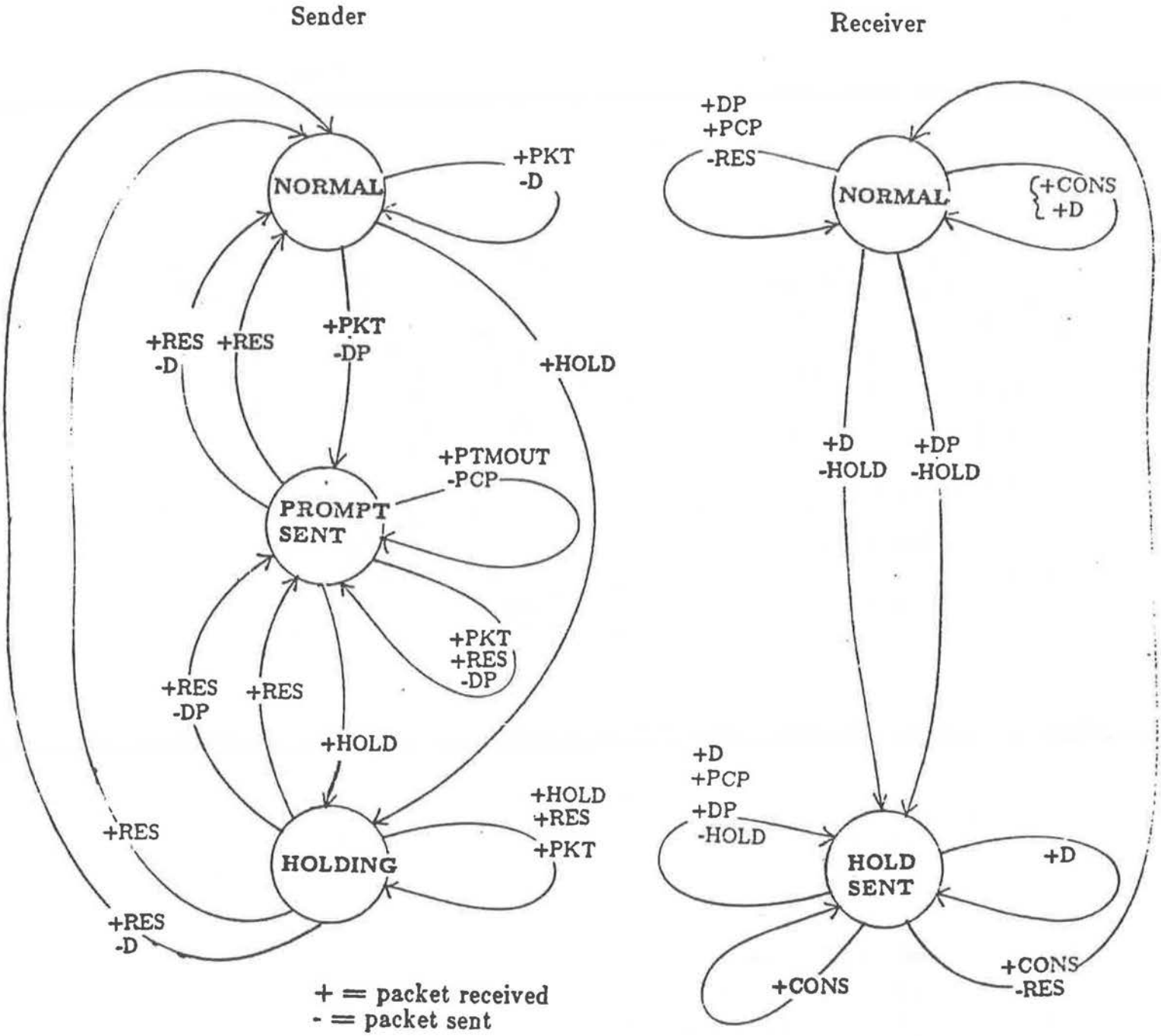
Figure A.1 Connection set up phase



+ = packet received
- = packet sent

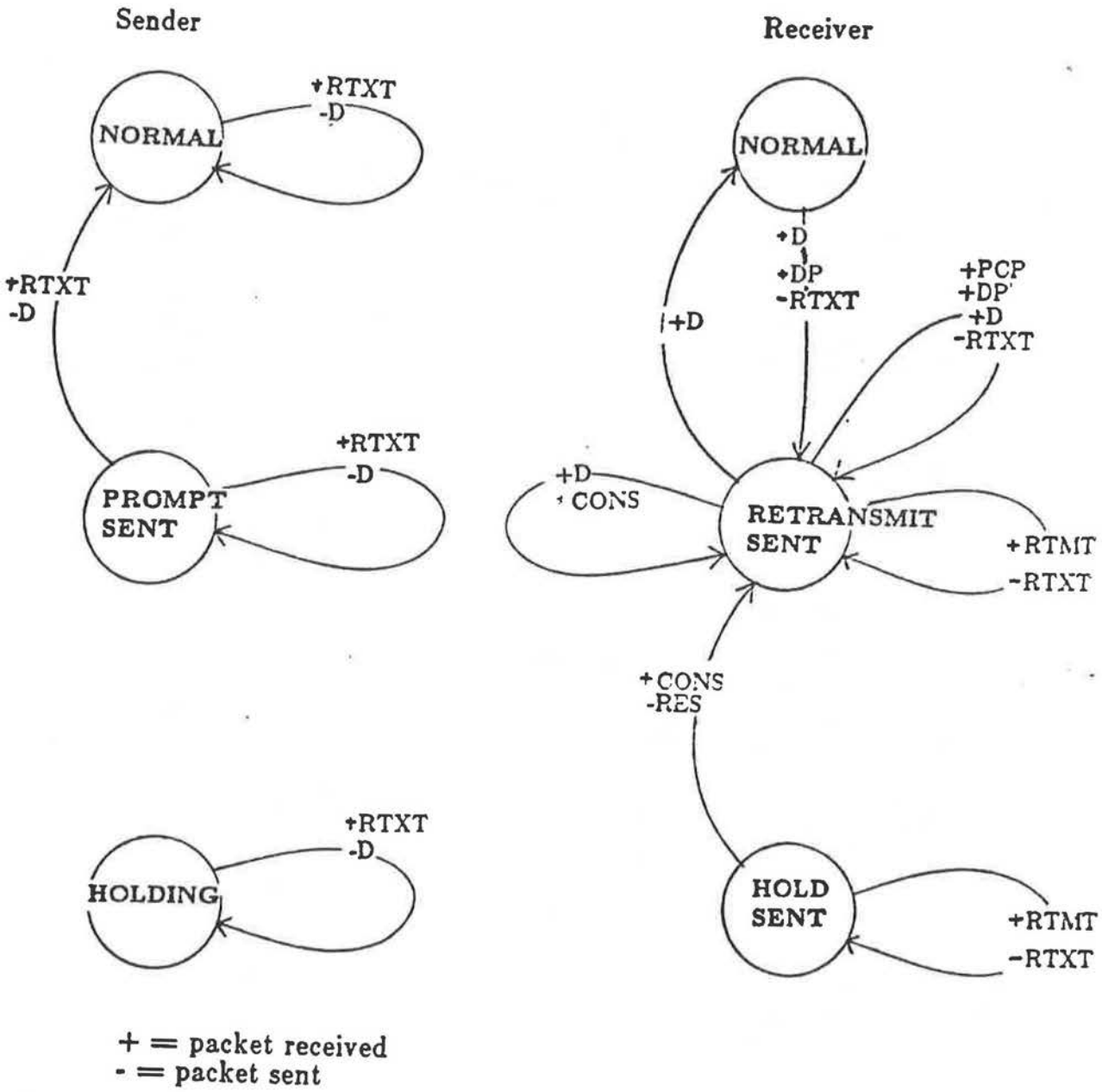
CCONF	connect confirm packet
CCR	connect request from local client
CREQ	connect request from remote peer
CTMOUT	timer to handle loss of CREQ and CCONF packets.

Figure A.2 Execution phase with flow control



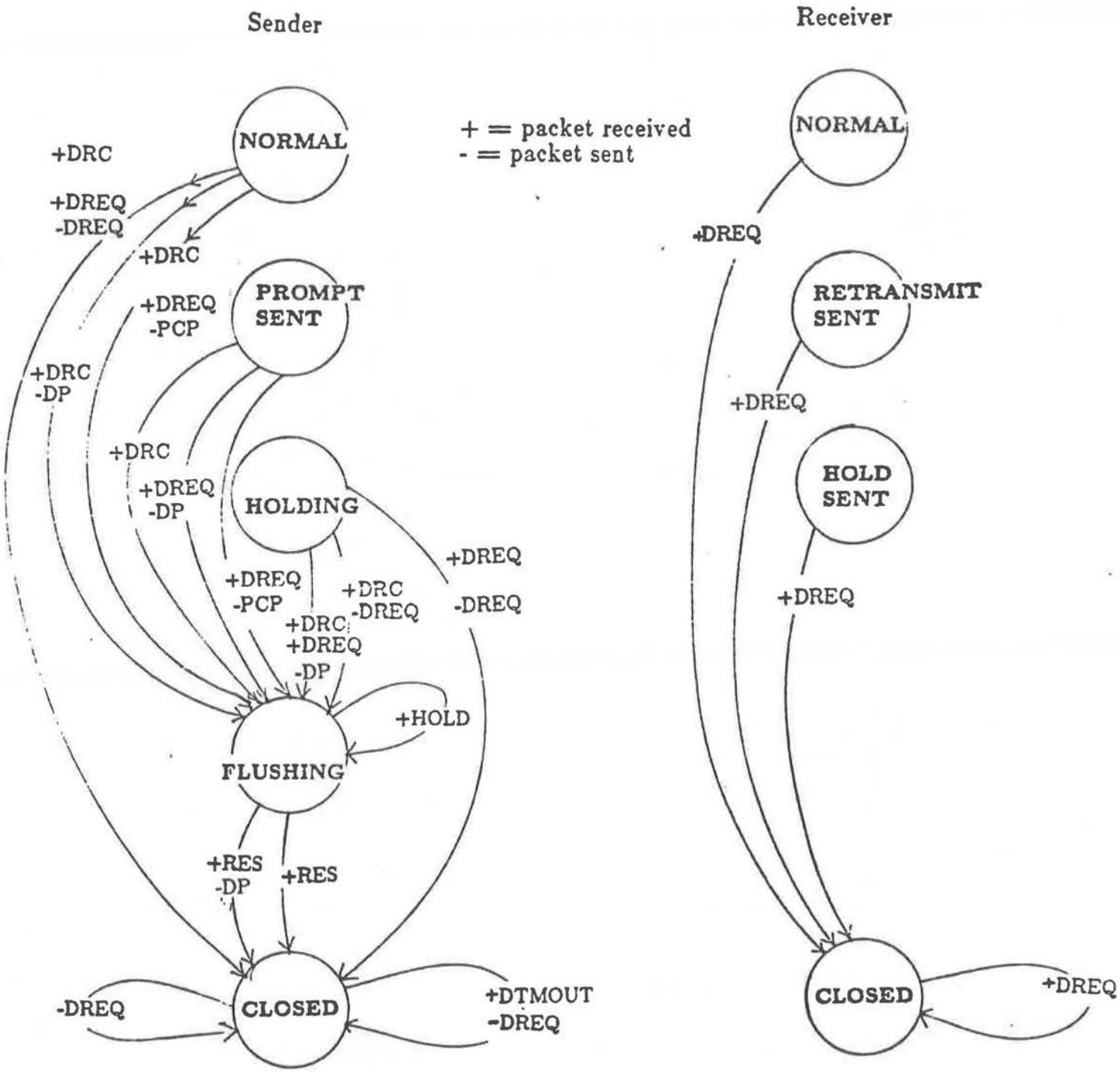
CONS	consume request from receiver's client
D	data packet
DP	data + prompt packet from sender to receiver, to force an acknowledgement.
HOLD	hold packet from receiver to sender, to prevent further transmissions.
PCP	prompt control packet from sender to receiver, to force an acknowledgement.
PKT	data packet from sender's client.
PTMOUT	prompt timer for sender.
RES	resume packet from receiver to sender, to acknowledge last correct packet.

Figure A.3 Execution phase with error recovery (but no flow control)



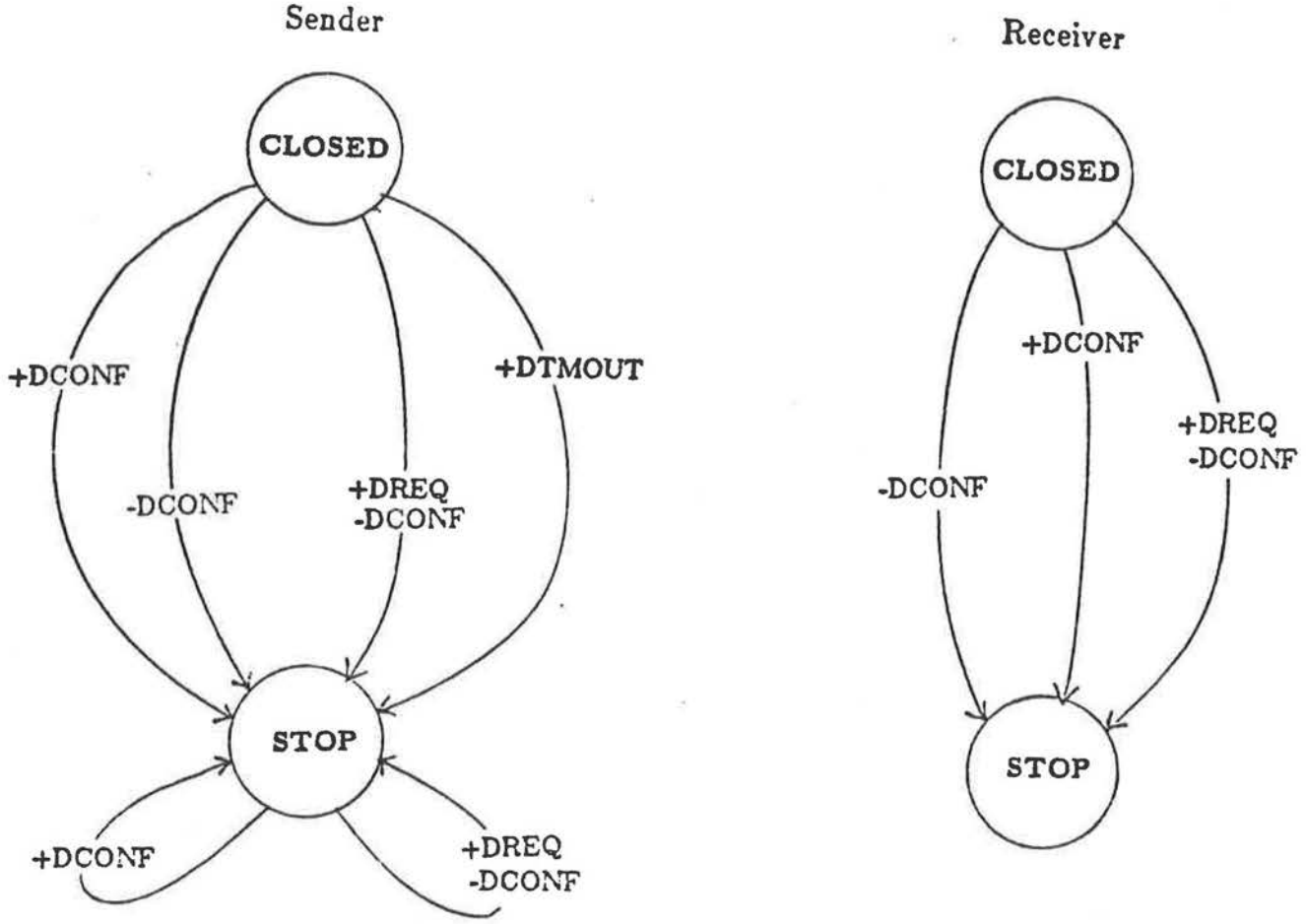
CONS	consume request from receiver's client
D	data packet
DP	data + prompt packet from sender to receiver, to force an acknowledgement.
HOLD	hold packet from receiver to sender, to prevent further transmissions.
PCP	prompt control packet from sender to receiver, to force an acknowledgement.
RES	resume packet from receiver to sender, to acknowledge last correct packet.
RTMT	retransmit timer, one for each lost packet at the receiver.
RTXT	retransmit packet, from receiver to sender.

Figure A.4 Flushing phase



DRC	disconnect request from local client
DREQ	disconnect request from remote peer
DTMOUT	timer to handle loss of DREQ and DCONF packets
D	data packet
DP	data + prompt packet from sender to receiver, to force an acknowledgement.
HOLD	hold packet from receiver to sender, to prevent further transmissions.
PCP	prompt control packet from sender to receiver, to force an acknowledgement.
RES	resume packet from receiver to sender, to acknowledge last correct packet.

Figure A.5 Disconnect phase



+ = packet received
- = packet sent

DCONF	disconnect confirm packet
DRC	disconnect request from local client
DREQ	disconnect request from remote peer
DTMOUT	timer to handle loss of DREQ and DCONF packets