The IL protocol

Dave Presotto
Phil Winterbottom

presotto,philw@plan9.bell–labs.com

ABSTRACT

To transport the remote procedure call messages of the Plan 9 file system protocol 9P, we have implemented a new network protocol, called IL. It is a connection–based, lightweight transport protocol that carries datagrams encapsulated by IP. IL provides retransmission of lost messages and in–sequence delivery, but has no flow control and no blind retransmission.

Introduction

Plan 9 uses a file system protocol, called 9P [PPTTW93], that assumes in–sequence guaranteed delivery of delimited messages holding remote procedure call (RPC) requests and responses. None of the standard IP protocols [RFC791] is suitable for transmission of 9P messages over an Ethernet or the Internet. TCP [RFC793] has a high overhead and does not preserve delimiters. UDP [RFC768], while cheap and preserving message delimiters, does not provide reliable sequenced delivery. When we were implementing IP, TCP, and UDP in our system we tried to choose a protocol suitable for carrying 9P. The properties we desired were:

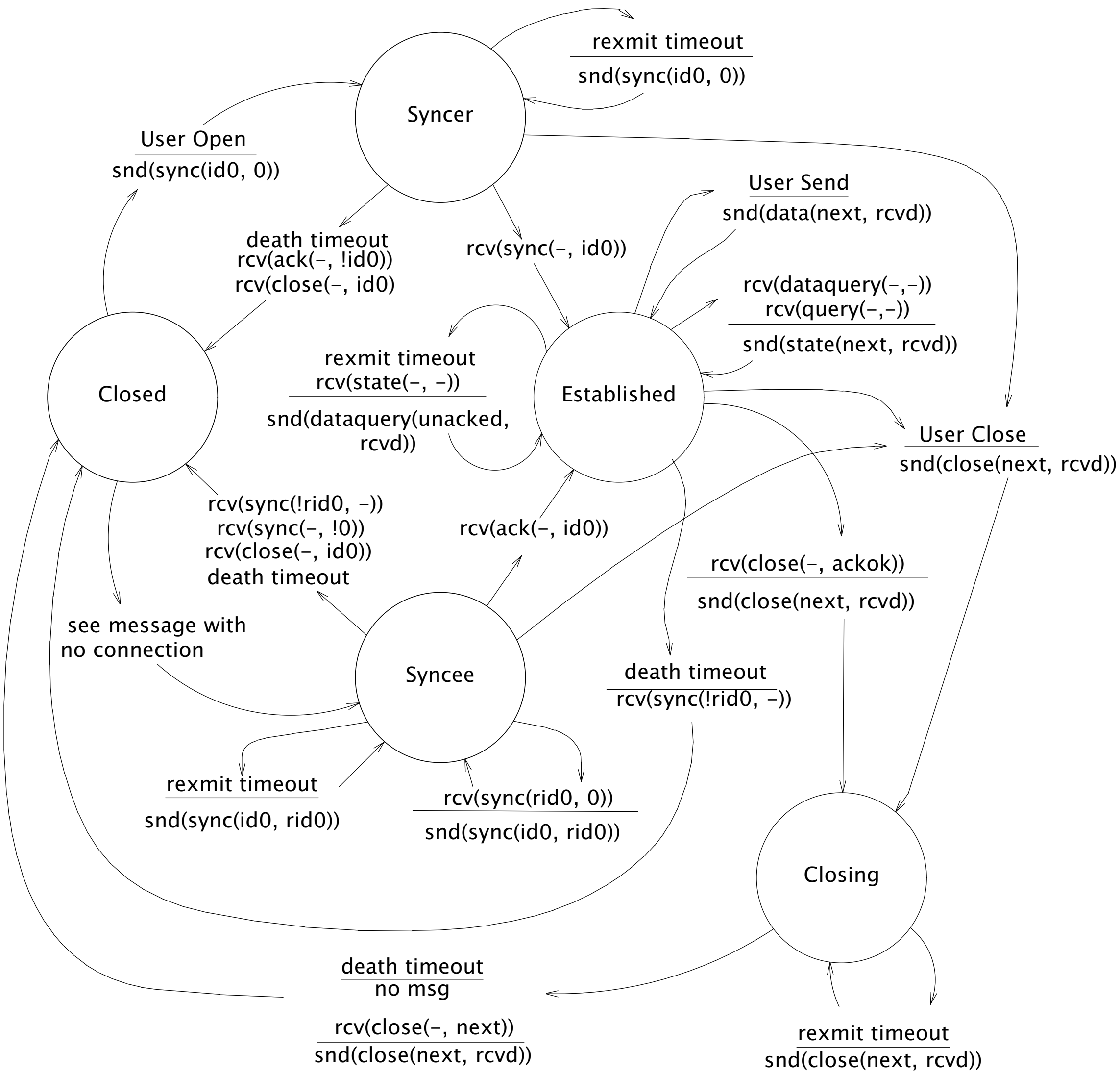
- Reliable datagram service
- In–sequence delivery
- Internetworking using IP
- Low complexity, high performance
- Adaptive timeouts

No standard protocol met our needs so we designed a new one, called IL (Internet Link).

IL is a lightweight protocol encapsulated by IP. It is connection–based and provides reliable transmission of sequenced messages. No provision is made for flow control since the protocol is designed to transport RPC messages between client and server, a structure with inherent flow limitations. A small window for outstanding messages prevents too many incoming messages from being buffered; messages outside the window are discarded and must be retransmitted. Connection setup uses a two–way handshake to generate initial sequence numbers at each end of the connection; subsequent data messages increment the sequence numbers to allow the receiver to resequence out of order messages. In contrast to other protocols, IL avoids blind retransmission. This helps performance in congested networks, where blind retransmission could cause further congestion. Like TCP, IL has adaptive timeouts, so the protocol performs well both on the Internet and on local Ethernets. A round–trip timer is used to calculate acknowledge and retransmission times that match the network speed.

Connections

An IL connection carries a stream of data between two end points. While the connection persists, data entering one side is sent to the other side in the same sequence. The functioning of a connection is described by the state machine in Figure 1, which shows the states (circles) and transitions between them (arcs). Each transition is labeled with the list of events that can cause the transition and, separated by a horizontal line, the messages sent or received on that transition. The remainder of this paper is a discussion of this state machine.



ackok	any sequence number between id0 and next inclusive
!x	any value except x
–	any value

Figure 1 – IL State Transitions

The IL state machine has five states: *Closed*, *Syncer*, *Syncee*, *Established*, and *Closing*. The connection is identified by the IP address and port number used at each end. The addresses ride in the IP protocol header, while the ports are part of the 18-byte IL header. The local variables identifying the state of a connection are:

state	one of the states
laddr	32-bit local IP address
lport	16-bit local IL port
raddr	32-bit remote IP address
rport	16-bit remote IL port
id0	32-bit starting sequence number of the local side
rid0	32-bit starting sequence number of the remote side
next	sequence number of the next message to be sent from the local side
rcvd	the last in-sequence message received from the remote side
unacked	sequence number of the first unacked message

Unused connections are in the *Closed* state with no assigned addresses or ports. Two events open a connection: the reception of a message whose addresses and ports match no open connection or a user explicitly opening a connection. In the first case, the message's source address and port become the connection's remote address and port and the message's destination address and port become the local address and port. The connection state is set to *Syncee* and the message is processed. In the second case, the user specifies both local and remote addresses and ports. The connection's state is set to *Syncer* and a `sync` message is sent to the remote side. The legal values for the local address are constrained by the IP implementation.

Sequence Numbers

IL carries data messages. Each message corresponds to a single write from the operating system and is identified by a 32-bit sequence number. The starting sequence number for each direction in a connection is picked at random and transmitted in the initial `sync` message. The number is incremented for each subsequent data message. A retransmitted message contains its original sequence number.

Transmission/Retransmission

Each message contains two sequence numbers: an identifier (ID) and an acknowledgement. The acknowledgement is the last in-sequence data message received by the transmitter of the message. For `data` and `dataquery` messages, the ID is its sequence number. For the control messages `sync`, `ack`, `query`, `state`, and `close`, the ID is one greater than the sequence number of the highest sent data message.

The sender transmits data messages with type `data`. Any messages traveling in the opposite direction carry acknowledgements. An `ack` message will be sent within 200 milliseconds of receiving the data message unless a returning message has already piggy-backed an acknowledgement to the sender.

In IP, messages may be delivered out of order or may be lost due to congestion or faults. To overcome this, IL uses a modified ''go back n'' protocol that also attempts to avoid aggravating network congestion. An average round trip time is maintained by measuring the delay between the transmission of a message and the receipt of its acknowledgement. Until the first acknowledge is received, the average round trip time is assumed to be 100ms. If an acknowledgement is not received within four round trip times of the first unacknowledged message (*rexmit timeout* in Figure 1), IL assumes the message or the acknowledgement has been lost. The sender then resends only the first unacknowledged message, setting the type to `dataquery`. When the receiver receives a `dataquery`, it responds with a `state` message acknowledging the highest received in-sequence data message. This may be the retransmitted message or, if the receiver has been saving up out-of-sequence messages, some higher numbered message. Implementations of the receiver are free to choose whether to save out-of-sequence messages. Our implementation saves up to 10 packets ahead. When the sender receives the `state` message, it will immediately resend the next unacknowledged message with type `dataquery`. This continues until all messages are acknowledged.

If no acknowledgement is received after the first `dataquery`, the transmitter continues to timeout and resend the `dataquery` message. The intervals between retransmissions increase exponentially. After 300 times the round trip time (*death timeout* in Figure 1), the sender gives up and assumes the connection is dead.

Retransmission also occurs in the states *Syncer*, *Syncee*, and *Close*. The retransmission intervals are the same as for data messages.

Keep Alive

Connections to dead systems must be discovered and torn down lest they consume resources. If the surviving system does not need to send any data and all data it has sent has been acknowledged, the protocol described so far will not discover these connections. Therefore, in the *Established* state, if no other messages are sent for a 6 second period, a `query` is sent. The receiver always replies to a `query` with a `state` message. If no messages are received for 30 seconds, the connection is torn down. This is not shown in Figure 1.

Byte Ordering

All 32- and 16-bit quantities are transmitted high-order byte first, as is the custom in IP.

Formats

The following is a C language description of an IP+IL header, assuming no IP options:

```
typedef unsigned char byte;
struct IPIL
{
        byte    vihl;        /* Version and header length */
        byte    tos;         /* Type of service */
        byte    length[2];   /* packet length */
        byte    id[2];       /* Identification */
        byte    frag[2];     /* Fragment information */
        byte    ttl;         /* Time to live */
        byte    proto;       /* Protocol */
        byte    cksum[2];    /* Header checksum */
        byte    src[4];      /* Ip source */
        byte    dst[4];      /* Ip destination */
        byte    ilsum[2];    /* Checksum including header */
        byte    illen[2];    /* Packet length */
        byte    iltype;      /* Packet type */
        byte    ilspec;      /* Special */
        byte    ilsrc[2];    /* Src port */
        byte    ildst[2];    /* Dst port */
        byte    ilid[4];     /* Sequence id */
        byte    ilack[4];    /* Acked sequence */
};
```

Data is assumed to immediately follow the header in the message. `Ilspec` is an extension reserved for future protocol changes.

The checksum is calculated with `ilsum` and `ilspec` set to zero. It is the standard IP checksum, that is, the 16-bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a message contains an odd number of header and text bytes to be checksummed, the last byte is padded on the right with zeros to form a 16-bit word for the checksum. The checksum covers from `cksum` to the end of the data.

The possible *iltype* values are:

```
enum {
        sync=           0,
        data=           1,
        dataquery=      2,
        ack=            3,
        query=          4,
        state=          5,
        close=          6,
};
```

The `illen` field is the size in bytes of the IL header (18 bytes) plus the size of the data.

Numbers

The IP protocol number for IL is 40.

The assigned IL port numbers are:

7	echo all input to output
9	discard input
19	send a standard pattern to output
565	send IP addresses of caller and callee to output
566	Plan 9 authentication protocol
17005	Plan 9 CPU service, data
17006	Plan 9 CPU service, notes
17007	Plan 9 exported file systems
17008	Plan 9 file service
17009	Plan 9 remote execution
17030	Alef Name Server

References

[PPTTW93] Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey, and Phil Winterbottom, ''The Use of Name Spaces in Plan 9'', *Op. Sys. Rev.,* Vol. 27, No. 2, April 1993, pp. 72-76, reprinted in this volume.
[RFC791] RFC791, *Internet Protocol, DARPA Internet Program Protocol Specification,* September 1981.
[RFC793] RFC793, *Transmission Control Protocol, DARPA Internet Program Protocol Specification,* September 1981.
[RFC768] J. Postel, RFC768, *User Datagram Protocol, DARPA Internet Program Protocol Specification,* August 1980.