Inter-Office Memorandum

To	Communication Protocols	Date	July 3, 1978
From	Ed Taft	Location	Palo Alto
Subject	Synchronous Line Transport Protocol	Organization	PARC/CSL

XEROX

Filed on: [Maxc1]<Pup>SyncTransport.press

Low-speed synchronous lines used to transport Pups are controlled by means of the Synchronous Line Transport Protocol (sometimes referred to as the SLA protocol because synchronous lines were first interfaced to Novas using a Synchronous Line Adapter). In addition to Pup encapsulation, this protocol entails some network-specific, non-Pup communication that enables hosts to maintain dynamic information about line status and sub-network connectivity.

The Synchronous Line Transport Protocol is used by Nova and Alto Gateways. We presently consider it to be a private’’ protocol among Gateways, and it is subject to change at any time.

All numbers are decimal unless suffixed with B’, in which case they are octal.

Packet Framing

All packets (whether or not they are Pups) are transmitted as transparent BiSync data frames. We are using only a subset of the full BiSync line control protocol, namely the part dealing with transparent transmission of data frames.

Frame Format

A frame consists of the following sequence of 8-bit bytes:

SYN SYN ... SYN DLE STX ... transparent data ... DLE ETX CRC1 CRC2

The frame begins with at least two SYN bytes, which the receiver searches for in order to establish byte synchronization. The sequence DLE STX signals the start of the data portion of the frame.

Within the data portion, all bytes are treated as literal data except DLE, which is an escape character. A literal data byte whose code corresponds to DLE is transmitted by doubling it, i.e., by sending DLE DLE. If the sequence DLE SYN appears, both bytes are ignored. Some synchronous interfaces automatically transmit DLE SYN when a transmit data-late condition occurs.

The end of the transparent data is indicated by the sequence DLE ETX. Following this are two bytes containing the 16-bit CRC (Cyclical Redundancy Check), transmitted low-order byte first. The CRC algorithm is given below. The CRC is computed over the transparent data bytes and the ETX. When the sequence DLE DLE appears, only one of the DLEs contributes to the CRC. When DLE SYN appears, neither byte is included.

Byte synchronization is not necessarily maintained from one frame to the next, so after each frame the receiver should restart its bit-at-a-time search for a SYN byte. The data transmitted between frames is not specified, but it should not be a pattern that could be mistaken for SYN or DLE. All-zeroes or all-ones are good choices for inter-frame data.

All bytes are transmitted low-order bit first. When the transparent data is treated as 16-bit words, the order of bytes in each word is high-order, then low-order.

Character Codes

We use the ASCII codes for the special characters. These are:

SYN	026B
DLE	020B
STX	002B
ETX	203B

CRC *Algorithm*

The CRC algorithm used is the industry-standard CRC-16. The CRC is a 16-bit number which is the remainder from dividing the data bit stream by the polynomial $x^{16}+x^{15}+x^2+x^0$. Hardware implementations of CRC-16 are available in the form of integrated circuits such as the Fairchild 9401.

The manner in which the CRC is generated and checked is as follows. The transmitter initializes the CRC to zero. Then, for every bit in the data stream, it updates the CRC using a simple XOR-feedback technique. At the end of the data, the transmitter appends the CRC, low-order bit first.

The receiver computes the CRC in a similar manner except that the 16 bits of CRC at the end of the data are *included* in the CRC computation. The CRC algorithm has the property that if no errors have occurred, the result of applying the algorithm to the data *and* the received CRC will be zero.

The CRC may be computed in software or microcode by any of several methods. One that is easy to understand works a single bit at a time, but this is relatively expensive since it must be executed 8 times for every data byte. The following BCPL procedure, given a partial CRC and a new data byte (right-justified), returns the updated CRC:

```
let UpdateCRC(crc, data) = valof
    [
    for i = 1 to 8 do
        [
        let xorFeedback = (crc xor data) & 1
        crc = crc rshift 1
        data = data rshift 1
        if xorFeedback ne 0 then crc = crc xor 120001B
        ]
    resultis crc
    ]
```

The idea is to right-shift the partial CRC and the data byte by one bit and to compare the bits that are shifted out. If they are the same, the CRC is not modified further; if they are different, the (shifted) CRC is XORed with the constant 120001B.

A faster technique updates the CRC a full byte at a time, using a 256-word table, CRCTAB, which is initialized as follows:

```
for i = 0 to 255 do
    [
    let crc = 0
    let val = i
    for power = 0 to 7 do
        test (val & 1) eq 0
            ifso val = val rshift 1
            ifnot
                [
                crc = crc xor (120001B rshift (7 - power))
                val = (val rshift 1) xor 120001B
                ]
    CRCTAB ! i = crc
    ]
```

The procedure for updating the CRC is then simply:

```
let UpdateCRC(crc, data) =
    (crc rshift 8) xor CRCTAB ! ((crc xor data) & 377B)
```

Packet Formats

Two types of packets are presently transported over synchronous lines: *encapsulated Pups* and *sub-network routing tables*. Refer to Figure 1. The two types are distinguished by the first word of the packet. An encapsulated Pup is type 512 and a routing table is type 513. 16 bits are reserved for the Type word because we contemplate subdividing it to include other information such as that required for sub-network fragmentation of large packets.

Note that what is shown in the figure is the *transparent data* only; all packets are carried within frames as described in the preceding section. This should be assumed for the remainder of this document.

Pup Encapsulation

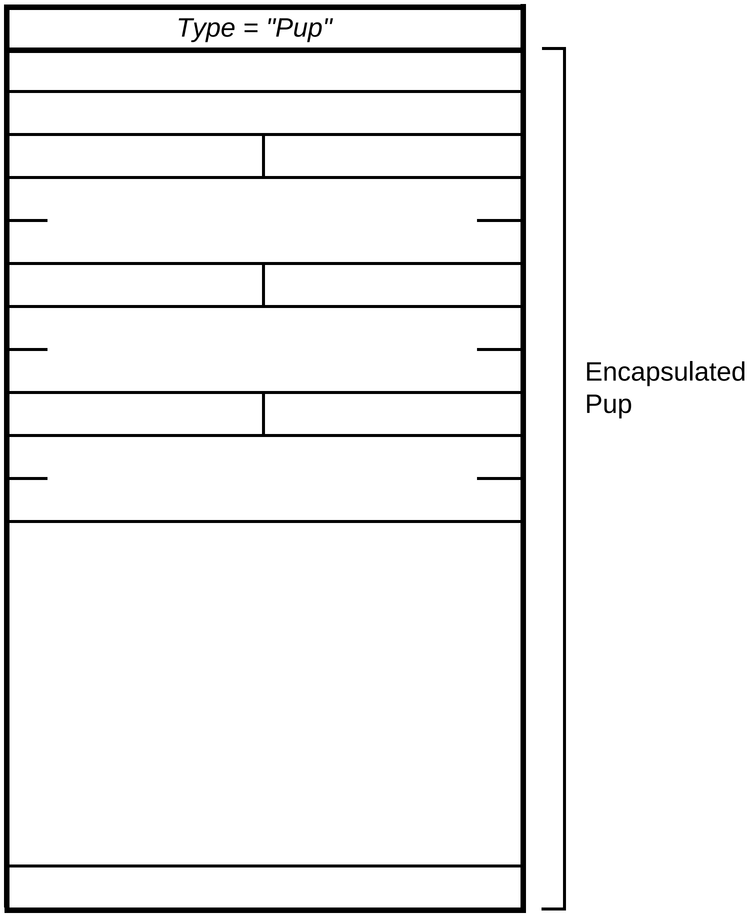
A Pup is encapsulated simply by prefixing the appropriate Type word. Note that the packet does *not* carry immediate source and destination host numbers, though the immediate destination is derived and used by the sub-network routing algorithm to determine on which outgoing line to transmit the packet. A consequence of the packet's not carrying the immediate destination host number is that the Pup must undergo inter-network Pup routing at every sub-network node through which it passes. This in turn implies that all sub-network nodes must be Pup Gateways.

Sub-Network Routing Table

A Routing Table is a *non-Pup* packet carrying sub-network routing information. This should not be confused with a Gateway Information Pup, which is network-independent and carries inter-network routing information.

The packet consists of an identifying Type word, the sub-network host number of the packet's sender, the number of routing entries, and the routing entries themselves. The first routing entry refers to sub-network host 1, the second to host 2, etc. This arrangement implies that there can't be very many hosts in the sub-network, and the host number space must be fairly dense.

Each routing table entry contains a hop count and a line number. The hop count is the estimated minimum number of point-to-point lines between the host generating the routing table and the host to which the routing table entry refers. The entry corresponding to the sending host itself must



Pup Encapsulation

Type = "Routing Table"	
Source Host	Number of Entries
Host 1 Hop Count	Line Number
Host 2 Hop Count	Line Number
Host n Hop Count	Line Number

Sub-Network Routing Table

Figure 1.
Synchronous Line Transmission Protocol Packet Formats

contain a hop count of zero. Entries corresponding to hosts that are believed to be inaccessible contain a hop count of 255.

The line number identifies the line over which the sending host believes it can achieve the minimum path claimed by the hop count. This information is useless to a host receiving the routing table. It is included so that the body of a Routing Table packet may be generated simply by copying the internal representation of the sender's routing table.

Sub-Network Organization and Algorithms

A collection of hosts interconnected by synchronous lines is treated as a single network within the Pup inter-network. This network is assigned a unique network number, and the hosts are assigned unique host numbers within that network. To avoid any confusion between the Pup inter-network and a specific network composed of nodes interconnected by synchronous lines, we refer to the latter as the *sub-network*.

Each host in the sub-network maintains a local routing table containing an entry for every other host. Each entry contains a hop count and a line number, as described in the preceding section. Additionally, for each synchronous line connected to the host there is a *line state* with values *down*, *up*, and *looped back*, and a timer used to time out dead lines.

Every 5 seconds, the host sends a Routing Table packet on every line.

Upon receiving a routing table from some line *l*, the host updates its local state as follows:

> The timer for line *l* is reset.
>
> The local routing table is enumerated. For each entry whose line number field is equal to *l*, the hop count is set to 255. The effect of this is to purge obsolete information about line *l*, since new information is about to be incorporated.
>
> If the *Source Host* is equal to the local host number, the line state for line *l* is set to *looped back* and no further processing is performed on the Routing Table packet.
>
> The line state for line *l* is set to *up*.
>
> The local routing table and the entries in the packet are enumerated in parallel. For each host, if the hop count in the local routing table is greater than the corresponding hop count in the packet, the local routing table entry's hop count is replaced by the packets hop count plus one and the line number is replaced by *l*. If the resulting hop count is greater than the constant *MaxHops* (presently 15), it is replaced by 255 (thereby marking the host inaccessible).

If no routing tables are received over some line *l* during a period of 20 seconds, the line's state is changed to *down*. Then all entries in the local routing table are enumerated. For each entry whose line number field is equal to *l*, the hop count is set to 255.

Outgoing Pups are routed to lines by computing the *immediate destination host* (an operation performed at the Pup inter-network level) and simply using it as an index into the local routing table. If the routing table entry's hop count is 255, the host is inaccessible and the packet is discarded.

This is an extremely simplified version of the routing algorithm used in the Arpanet. Its major defects are that (a) it uses an imperfect metric (hop count) for making routing decisions, (b) it is unable to make effective use of multiple paths to a given destination, and (c) it does not extend well to large sub-networks.