The Point to Point Protocol in Forth

2 Nov 2000

Howerd Oakford
Inventio Software Ltd
www.inventio.co.uk

The Point to Point Protocol is used by PC programs to connect to an Internet Service Provider (ISP) so that applications can surf the web, or get email.
In this paper I describe an implementation of a PPP Peer in Forth.
English is not a good language to describe complex computer programs, so I have given some examples of Forth source.
Paper is not a good medium to present Forth source - it is better to try out the program interactively.

PPP is complicated. I have done everything I can to simplify it :

1. I captured the actual packets transferred between my PC and ISP when checking my email using a well known email program. This converted the enormous range of possible formats and options to the ones that are actually used.
2 I use a "flat architecture" and avoid the concept of an "IP-stack" or hierarchical layer of protocols. This follows the Forth-like "direct action" approach, where Forth words actually do something, rather than the C-like or operating system approach where functions process and pass data structures to other functions.
3 The current implementation has four tasks (apart from the Operator task). Again the tasks are used in a Forth-like way - each one defines an activity which is inherently asynchronous to the other tasks. The C-like approach to tasks is to use them to isolate functions written by different teams of people, so that each part of the software can operate asynchronously to each other part without interfering with each other.

The Point to Point Protocol Peer has the properties of both a Client (issuing requests) and Server (responding to requests). Each end of a PPP connection is identical to the other.
However, the Client and Server parts of PPP are not independent, as the PPP Peer must step through three levels to achieve an open PPP connection :
LCP Link Control Protocol level sets up the Maximum Transmission Unit (MTU) and other parameters required to pass PPP packets between the two PPP Peers.
The PAP/CHAP level authenticates the other PPP Peer.
The IPCP sets up the Internet Protocol parameters such as IP addresses and compression algorithms.

When all three levels have been achieved by both the Client and Server side, the PPP link is open, and IP packets may be sent in either direction.

IP packets carry a payload of data in one of the higher level formats such as UDP and TCP. A simple example of a UDP packet is given. TCP is more difficult and is my next project!

This is "work in progress", and the source listed below is a only a small part of the current system. I intend to publish a working system as soon as it can get my email…

First the easy bit, PPP Packet Output :

The basic unit of currency on a PPP connection is the packet - a string of characters separated by a PPP Flag character : hex 7E.
The format of a PPP packet is well defined, and includes a PPP protocol field and CRC .

When sending a packet nothing except the payload is stored in an array, just as data on the stack or as literals in the program. For example, executing **IP** actually sends the payload as an IP packet, complete with PPP Flag, PPP header, IP header and checksum and PPP CRC. PPP "escape" characters (hex 7D) are sent before certain characters, these characters are XORd with hex 20 . See the code below for details.

In a conventional IP stack, the payload would be sent to each layer in turn to have another header added to it, before being passed to an output function. Here we calculate the byte stream as we need it and send it immediately. This is what is meant by a "flat architecture".

```
(       720 )

( Output )  HEX                                              \ PPP_ESC  is the PPP "escape" character
7D CONSTANT PPP_ESC   7E CONSTANT PPP_FLAG                   \ PPP_FLAG  is the PPP flag character
                                                             \ ACCM = ASCII Control Character Map
2VARIABLE ACCM   : \ACCM   00000000. ACCM 2! ;  \ACCM        \ACCM  has one bit set for each character from 0 to hex 1F if
CREATE [2**] 1 C, 2 C, 4 C, 8 C, 10 C, 20 C, 40 C, 80 C, 100 C,  \   that character must be escaped.
: ?ACCM ( c - f)   DUP  07 AND  [2**] + C@                   \ ?ACCM  looks up the bit for character c in the ACCM and
   SWAP 8 / ACCM + C@ AND ;                                  \   returns true if it is set ( meaning it must be escaped )
                                                             \ ESC? returns true if c must be escaped.
: ESC? ( c - f)   >R  I PPP_FLAG =  I PPP_ESC = OR           \ The PPP escape character and the PPP flag character must
      I 0 20 WITHIN IF  I ?ACCM OR  THEN  R> DROP ;          \  also be escaped.
                                                             \ HEMIT  sends c as one or two characters. Certain characters
: HEMIT ( c)   DUP  [CRC-OUT] +CRC                           \  must not be sent in a PPP packet - they are XOR'd with 20
   DUP ESC? IF  PPP_ESC REMIT  20 XOR  THEN  REMIT ;         \  and preceded by a PPP escape character = 7D.
                                                             \  HEMIT  also accumulates the CRC of the un-escaped character
: HTYPE ( a n)   ?DUP IF                                     \ HTYPE  sends an "escaped" string
      OVER + SWAP DO  I C@ HEMIT  LOOP  ELSE  DROP  THEN ;   \

(       721 )

   ( Output )  HEX                                           \
: W> ( n)   DUP DUP  >< 0FF AND HEMIT  0FF AND HEMIT  >< +1CS ;  \ W>  sends a 16 bit Word in Big Endian format
                                                             \
: 2C> ( c1 c2)   SWAP >< OR W> ;                             \ 2C>  sends two 8 bit Characters, c1 first then c2.
                                                             \ D>>  sends a 32 bit Word in Big Endian format
: D>> ( d)   W>  W> ;                                        \
                                                             \ @>  sends the 16 bit Word at a in Big Endian format
: @> ( a)   @  W> ;                                          \ 2@>  sends the 32 bit Word at a in Big Endian format
                                                             \
: 2@> ( a)   2@ D>> ;                                        \ PPP< .... >PPP send a formatted PPP packet
                                                             \ PPP<  sends the initial Flag, address and control bytes
: PPP<   SERIAL_OUT GET  PPP_FLAG REMIT  \CRC-OUT            \ >PPP sends the accumulated CRC and terminating Flag
   FF HEMIT  03 HEMIT ;                                      \
                                                             \ Note that since the checksum is 16 bits it is more efficient
: >PPP   [CRC-OUT] @ -1 XOR  >< W>  PPP_FLAG REMIT           \  to send bytes two atat time, hence 2C> and no C>.
   SERIAL_OUT RELEASE ;                                      \  The 1CS word does take care of odd length strings though.

(       722 )

   ( LCP Output )  HEX                                       \
70 USER SO.UR.CE.IP   74 USER SOURCEPRT                      \ SO.UR.CE.IP  the source IP address
76 USER DE.ST.IN.IP   7A USER DESTINPRT                      \ SOURCEPRT   the source port number
                                                             \ The IP address and port define a unique 48 bit socket
: LCP ( a n code seq)   PPP<  C021 W> ( code seq) 2C>        \
   ( length) DUP ( n) 4 + W>  ( a n) HTYPE  >PPP ;           \ DE.ST.IN.IP   the destination IP address
                                                             \ DESTINPRT   the destination port number
: LCP_MTU ( MTU)   PPP<  C021 W>  ( REQ code) 1 ( seq) 1 2C> \
   ( length) 0A W>  ( MTU) W>  >PPP ;                        \ LCP  sends an LCP packet of code and sequence numberseq
```

```
                                                  \
                                                  \ LCP_MTU  sends an LCP request with MTU setup option.
```

```
(        723 )

   ( IPCP Output )  HEX                           \
 : IPCPOPT> ( d opt)   >< 06 OR W>  D>> ;         \IPCPopt send a six byte IPCP option. Most options consist
                                                  \   of an option type, length and 4 data bytes.
 : IPCP ( IPaddr code seq)   PPP<                 \
    8021 W> ( code seq) 2C>                       \  Only one option is sent here : the IP addresses, code 3.
    ( options length + 2 ) 0A W>                  \
    ( IPaddr) 3 IPCPOPT>                          \
 \   ( DNS ) 0. 81 IPCPOPT>  0. 82 IPCPOPT>       \
 \   ( NBNS ) 0. 83 IPCPOPT>  0. 84 IPCPOPT>      \
    >PPP ;                                        \
                                                  \
```

```
(        724 )

   ( IP Output )  HEX                             \
 : PPP<:IP ( prot frag id# )   PPP<  ( PPPprot=IP) 0021 W>    \ PPP<:IP sends an Internet Protocol packet, an IP header plus a
    \1CS  4500 W>  2>R  OVER ( n) 14 + W>  2R>    \   payload.
    ( id#) W> ( frag) W>  ( prot) ( TTL) 40 >< OR W>          \   a n  define the payload string
    ( add in the IP addresses' checksums in advance : )       \  prot  is the payload protocol :
    SO.UR.CE.IP 2@ +1CS +1CS  DE.ST.IN.IP 2@ +1CS +1CS        \      1 is ICMP  Internet Control Message Protocol
    1CS@ -1 XOR  W>                               \      2 is IGMP  Internet Control Message Protocol
    SO.UR.CE.IP 2@ D>>  DE.ST.IN.IP 2@ D>> ;      \      6 is TCP  Transmission Control Protocol
                                                  \      hex 11 is UDP  User Datagra Protocol
 : IP ( a n prot frag id# )   PPP<:IP  ( a n) HTYPE  >PPP ;   \  frag  is the payload's offset in 8 byte units, with
                                                  \     bit 13 zero, bit 14 as "do not fragment" and bit 15 as
 : TEST_IP  PAD 0 11 0 1 IP ;                     \     "more fragments".
                                                  \  id#  is this IP packet's number.
                                                  \    ( calculate the IP header checksum
                                                  \
                                                  \ IP  sends n bytes at a as the payload of an IP packet
```

```
 (        725 )

   ( UDP Output )  HEX                            \
 : UDP ( a n)   ( UDP prot) 11  ( frag) 0  ( id#) 1  PPP<:IP  \ A large payload can be delivered by using many IP packets by
    SOURCEPRT @ W>   DESTINPRT @ W>               \   breaking it up into fragments, the frag  value being set so
    ( n)  28 + W>   2DUP 1CS  1CS@ -1 XOR W>      \   that the receiving computer can reassemble it.
     >PPP ;                                       \
                                                  \ UDP  sends the simplest UDP packet, one IP fragment only.
                                                  \   n must be less than the Maximum Transmission Unit ( MTU ).
                                                  \   The MTU defaults to 1500 bytes, but may be setup by the LCP.
```

A bit more difficult is PPP input :

Again "direct action" is taken, rather than buffering and passing of pointers.

First the start of a PPP packet is detected, then the protocol type is used to calculate the header size, and the header is directed to the Header array and the payload (the rest of the packet) is directed to the Buffer array.
The Header array is a small 64 byte array used to save the current header and any other non-payload data. It is destroyed when the next PPP packet arrives.
The Buffer is a large (16K byte) circular linked list which retains the most recent payloads. Only the oldest payloads are destroyed, and only when the Buffer is full.

The CRC is accumulated during this process, and is checked when the end of the packet is detected.

Detection of the start of packet of a PPP packet is tricky, because there may be one or more PPP Flag characters between any two packets.

The first character of a packet is the first non-Flag character after one or more Flag characters have been received.

```
(        733 )

( PPP input)  HEX                                    \
VARIABLE FLAG-RECVD                                  \ FLAG_RECVD is set true when a PPP Flag character is received
VARIABLE LAST[CRC-IN]                                \
                                                     \ LAST[CRC-IN]  saves the accumulated incoming data CRC _before_
: HKEY ( - c)   RKEY                                 \   the Flag character resets it.
   DUP PPP_FLAG = IF ( the flag character )          \
      [CRC-IN] @ LAST[CRC-IN] !  \CRC-IN  -1 FLAG-RECVD !   \ HKEY  waits for one "un-escaped" char from the serial Port
   ELSE ( any other character, maybe "escaped" )     \  It waits for a second character if an escape character ( 7D )
      DUP PPP_ESC = IF  DROP RKEY 20 XOR  THEN       \  is received.
      DUP [CRC-IN] +CRC                              \  HKEY also accumulates the PPP layer CRC and clears the
   THEN                                              \   CRC if a Flag character is received.
;                                                    \
                                                     \
                                                     \
                                                     \
                                                     \
```

```
(        735 )

   ( PPP input)  HEX                                 \PPPstart waits for a PPP header and returns the Protocol type
: PPPstart ( - n)   0                                \
   BEGIN  DROP                                       \ If FLAG-RECVD is true the end of the last frame's PPP Flag
      HKEY  DUP PPP_FLAG = 0=  ( this one is not a Flag )   \   character was the last one received, and this counts as the
      FLAG-RECVD @  AND  ( but we have had a Flag )  \   first PPP Flag character of the next frame.
   UNTIL  0 FLAG-RECVD !                             \   The start of the PPP packet is when the first non-Flag
   DUP FF = IF  ( an address byte, so remove the control byte )  \   character received after one or more Flag characters.
      DROP  HKEY DROP  ( and get the first protocol byte ) HKEY  \   ( i.e empty frames are discarded )
   THEN  ( otherwise it was the first protocol byte )   \ If first char.is not hex FF it is assumed that Address
   DUP 1 AND 0= IF  ><  HKEY OR  THEN                \   and Code Field Compression is enabled, so it will be
   SAVE-BUF ;                                        \   the first of the Protocol ( one or two ) bytes, the last
                                                     \   byte being marked by bit 0 being true. This always decodes
                                                     \   Protocol Field Compression (pfc ), even if it has not been
                                                     \   selected.
```

Having directed the incoming data stream to its rightful places, the data can now be processed. Four tasks are currently used to handle the Point to Point Protocol.
The tasks allow the functionality of the protocol to be expressed clearly, without getting bogged down in the details of packet parsing and creation.

```
(        773 )

( Peer state)                                        \
: CALC_PPP_PEER_STATE                                \ CALC_PPP_PEER_STATE  changes our PPP peer state only when both
  PPP_CLIENT_STATE @ 1 =  PPP_SERVER_STATE @ 1 = AND \  the client and server tasks have completed the current level.
   IF ( LCP ) 1 PPP_PEER_STATE !  .PPP_STATE  THEN   \
  PPP_CLIENT_STATE @ 2 =  PPP_SERVER_STATE @ 2 = AND \
   IF ( PAP/CHAP ) 2 PPP_PEER_STATE !  .PPP_STATE  THEN  \
  PPP_CLIENT_STATE @ 3 =  PPP_SERVER_STATE @ 3 = AND \
   IF ( IPCP ) 3 PPP_PEER_STATE !  .PPP_STATE  THEN  \
  PPP_CLIENT_STATE @ 4 =  PPP_SERVER_STATE @ 4 = AND \
   IF ( Open ) 4 PPP_PEER_STATE !  .PPP_STATE  THEN  \
;                                                    \
```

```
(        774 )

( Protocols )  HEX                                   \
: PPPpacket   PPPstart  DUP PPP_PROT !               \PPPpacket  receives PPP packets and parses them
   CASE                                              \   The header ( of size determined by the protocol type ) is
      0021 OF  PPP_IP    ENDOF                       \   put into the Header, then the payload into the Buffer.
      8021 OF  PPP_IPCP  ENDOF                       \
      C021 OF  PPP_LCP   ENDOF                       \
      C023 OF  PPP_PAP   ENDOF                       \
      C223 OF  PPP_CHAP  ENDOF                       \
      PPP_UNKNOWN                                    \
   ENDCASE ;                                         \
                                                     \
```

```
( 11 )                                                              \


(        775 )

(  0 ) ( PPP main task)                                             \
(  1 ) : PPP_MAIN   PPP_MAINING ACTIVATE  15 ATT!  >H  20 24 WIN    \ PPP_MAIN  repeatedly receives PPP packets and parses them
(  2 )    BEGIN                                                     \   for the other tasks to pick up from the Header and Buffer
(  3 )       .PPP_STATE  PPPpacket                                  \   It also updates the display of our PPP state
(  4 )    AGAIN ;                                                   \
(  5 )                                                              \

(        776 )

(  0 )    ( PPP client task)  HEX                                   \
(  1 ) : PPP_CLIENT   PPP_CLIENTING ACTIVATE  1F ATT!  >H  1 9 WIN  \ PPP_CLIENT  initiates requests to the remote PPP peer.
(  2 )    BEGIN  PAUSE                                              \
(  3 )    NEW_PPP_PROT @ CASE                                       \
(  4 )       8021 OF  IPCPclient  PAUSE  ENDOF                      \
(  5 )       C021 OF  LCPclient   PAUSE  ENDOF                      \
(  6 )       C023 OF     ENDOF                                      \
(  7 )       C223 OF     ENDOF                                      \
(  8 )    DROP  ENDCASE                                             \
(  9 )    AGAIN ;                                                   \
( 10 )                                                              \


(        777 )

(  0 )    ( PPP server task)  HEX                                   \
(  1 ) : PPP_SERVER   PPP_SERVERING ACTIVATE  5F ATT!  >H  0C 13 WIN  \ PPPpacket  receives PPP packets and parses them
(  2 )    BEGIN  PAUSE                                              \
(  3 )    NEW_PPP_PROT @ CASE                                       \
(  4 )       8021 OF  IPCPserver  PAUSE  ENDOF                      \
(  5 )       C021 OF  LCPserver   PAUSE  ENDOF                      \
(  6 )       C023 OF     ENDOF                                      \
(  7 )       C223 OF     ENDOF                                      \
(  8 )    DROP  ENDCASE                                             \
(  9 )    AGAIN ;                                                   \
( 10 )                                                              \


(        778 )

(  0 )    ( PPP kicking task)  HEX                                  \
(  1 ) : LCP_BACKOFF  400 ?MS ;  : PAP_BACKOFF  400 ?MS ;           \ LCP_BACKOFF  waits for a timeout period
(  2 ) : IPCP_BACKOFF  400 ?MS ;                                    \ PAP_BACKOFF  waits for a timeout period
(  3 )                                                              \ IPCP_BACKOFF  waits for a timeout period
(  4 ) : PPP_KICK   PPP_KICKING ACTIVATE  70 ATT!  >H  0C 13 WIN    \
(  5 )    BEGIN  PAUSE                                              \ PPP_KICK  performs timeout retries
(  6 )    CALC_PPP_PEER_STATE                                       \
(  7 )    PPP_PEER_STATE @ CASE                                     \
(  8 )       -1 OF ( term)   LCP_TERM  LCP_BACKOFF  ENDOF           \
(  9 )       00 OF ( idle)  ENDOF                                   \
( 10 )       01 OF ( LCP)   LCP_KICK  LCP_BACKOFF  ENDOF            \
( 11 )       02 OF ( PAP/CHAP)   PAP_KICK  PAP_BACKOFF  ENDOF       \
( 12 )       03 OF ( LCP and AUTH ok) IPCP_KICK  IPCP_BACKOFF  ENDOF  \
( 13 )       04 OF ( PPP open)  ENDOF                               \
( 14 )    DROP  ENDCASE                                             \
( 15 )    AGAIN ;   ' PPP_KICK  'PPP_KICK !                         \
```

And finally the top level user interface :

```
(       779 )

( Protocols )                                         \
: RUN   COM1  9600 BAUD ( 115200. JBAUD ) HEX  PAGE    \ RUN  starts the PPP peer
   0. MyIPaddr 2!  HisIPaddr 2TALLY                    \
   23 0 TAB  20 24 WIN  \PPP_STATE  .PPP_STATE         \ VIEW  starts the PPP peer in display only mode ( no Client
   \BUF  PPP_MAIN  PPP_CLIENT  PPP_SERVER  PPP_KICK    \  actions )
   24 0 TAB  DECIMAL ;                                 \
                                                      \ GO  opens a PPP link
: VIEW  RUN  \PPP_KICK ;                              \   Note that only one PPP peer needs to issue a GO command, as
                                                      \   the other PPP peer will be started by the receipt of an LCP
: GO   1 PPP_CLIENT_STATE !  1 PPP_SERVER_STATE ! ;   \   configure request.
                                                      \
: \GO   -1 PPP_CLIENT_STATE ! ;                       \ \GO  closes the PPP link
                                                      \
: run  RUN ;  : go  GO ;  : view  VIEW ;  : \tasks  \TASKS ;     \ run  is a lower case alias for RUN etc...
: K1  CR ." F1 = Help, F2 = Go, F3 = Reset, F4 = Stop tasks " ;  \ K1 , K2 , K3 and K4 are the actions of function keys F1 to F4
: K2  GO ;  : K3  RUN ;  : K4  \TASKS ;                          \ Press F1 for Help
```

The Forth program is written in 8086 polyForth. This is a 16 bit LittleEndian Forth with built in multitasking and uses blocks for the Forth source.

The current state of the program is that two PCs connected together with a null modem cable, and both running the program will open a PPP connection.
PAP and CHAP are not supported yet, although they are there in outline form and the MD5 algorithm (for CHAP) is complete and tested.
The LCP and IPCP options are defined at compile time, except for the IPCP IP address option which handles an IPCP configure request with a zero IP address option field by returning a NAK and new IP address.
The Buffer array circular queue does not support linked lists yet. This is only relevant to IP and higher protocols.
The PPP packet parsing and display functions correctly show the packets captured from the PC during fetching of email, and correctly calculate the PPP packet CRC and IP and UDP header checksums.

Howerd Oakford   2 Nov 2000