Computer Networks lab
BTech V Semester
Assignment III

1. Given a binary stream, WAP to determine the start of a frame (data link layer) and to mark different frame fields like frame delimiter, source and destination ids, error control coding bits, information field etc.
    a. As an illustration, the program should be able to demarcate different fields for HDLC and PPP. A sample of the message stream and other details are given below.

SNRM/UA frame exchange with HDLC parameter negotiation

Sent frame:

7EA00A00020023219318717E

Comments:
7E// HDLC frame opening flag
A00A// frame type and length
0002// destination address (server) upper HDLC address 0x0001
0023// destination address (server) lower HDLC address 0x0011
21// source address (client) 0x01
93// frame type SNRM (Set Normal Response Mode)
// information field with HDLC parameters not present, defaults are proposed
1871// Frame check sequence
7E// HDLC frame closing flag

Received frame:

7EA0232100020023 73F6C58180140502008006020080070400000001080400000001CE6A7E

Comments:
7E// HDLC frame opening flag
A023// frame type and length
21// destination address (client) 0x01
0002// source address (server) upper HDLC address 0x0001
0023// source address (server) lower HDLC address 0x0011
73// frame type UA (Unnumbered acknowledge)
F6C5// Header check sequence
818014//format identifier / group identifier / group length
05020080//parameter identifier/length/value – maximum information field length transmit
06020080//parameter identifier/length /value – maximum information field length receive
070400000001//parameter identifier/length/value – window size transmit

080400000001//parameter identifier/length/value – window size receive
CE6A // Frame check sequence
7E // HDLC frame closing flag

AARQ/AARE exchange

Sent frame: SNRM

7EA00A00020023219318717E

Received frame: UA

7EA023210002002373F6C5818014050200800602008007040000000108040000000 1CE6A7E

Sent frame: AARQ

7EA02E0002002321107ECBE6E600601DA109060760857405080101BE10040E01000000065F1F040000301DFFFFD4C57E

Comments:
7E// opening flag
A02E// frame type and length
0002002321// destination and source addresses
10// frame type I frame
7ECB// Header check sequence
E6E600// LLC bytes
601D// AARQ tag and length
A109060760857405080101//Application context name tag, length and encoded value
// This application association is established with lowest level security, so authentication components are not present
BE10// tag and length for AARQ user field
040E// encoding the choice for user-information (OCTET STRING, universal) and length
01// tag for xDLMS-Initiate request
00// usage field for dedicated-key component – not used
00// usage field for the response allowed component – not used
00// usage field of the proposed-quality-of-service component – not used
06// proposed dlms version number 6
5F1F// tag for conformance block
04// length of the conformance block
00// encoding the number of unused bits in the bit string
00301D//conformance block
FFFF// client-max-receive-pdu-size
D4C5// HDLC frame check sequence
7E// closing flag

Received frame:

7EA03A2100020023309941E6E7006129A109060760857405080101A20302
0100A305A103020100BE10040E0800065F1F040000301D190000070C527E

Comments:
7EA03A2100020023309941//HDLC frame header
E6E700// LLC bytes
6129//AARE tag and length
A109060760857405080101//Application context name tag, length and encoded value
A203// tag and length of the result component
02//encoding the choice for result (INTEGER, universal)
0100// length and value of result (accepted)
A305// tag and length for the result-source-diagnostic component
A103// tag and length of the acse-service-user choice
02// encoding the choice for result-source-diagnostic (INTEGER, universal)
0100// length and value of result-souce-diagnostic
BE10//tag and length for AARE user-field
040E// encoding the choice for user-information (OCTET STRING, universal) and length
08// tag for xDLMS-Initate.response
00// usage field of the negotiated-quality-of-service component
06// negotiated dlms version number 6
5F1F// tag for conformance block
04// length of the conformance block
00// encoding the number of unused bits in the bit string
00301D//negotiated conformance block
1900// server-max-receive-pdu-size
0007// VAA name (0x0007 for LN referencing)
0C52// HDLC frame check sequence
7E// closing flag

Reading and setting the clock

Sent frame:

7EA0199575546835E6E600C001810008000010000FF01000DFD7E

Comments:
7EA0199575546835// HDLC header
E6E600// LLC bytes
C001// GET.request.normal
81// invoke-id and priority
0008// interface class 8, clock
0000010000FF// logical name, OBIS code of the clock

0100// asking for 1st attribute, logical name
0DFD7E// HDLC frame closing

Received frame:

7EA018759574E9E8E6E700C40181000906000001000FFFD497E

Comments:
7EA018759574E9E8E6E700
C401//GET.response.normal
81// invoke-id and priority
00// Get-Data-Result choice data
0906// octet string (6)
0000010000FF// logical name, OBIS code of the clock
FD497E// HDLC closing

Sent frame:

7EA0199575767837E6E600C001810008000001000FF020065D77E

Comments:
7EA0199575767837E6E600
C0018100080000010000FF0200// GET.request normal, 2nd attribute, time
65D77E

Received frame:

7EA01E7595966F67E6E700C4018100090C07D20C04030A060BFF007800F3307E

Comments:
7EA01E7595966F67E6E700// HDLC header
C401// GET.response.normal
8100
090C// octet string (12)
07D2// year 2002
0C// month December
04// day 4th
03// day of the week, Wednesday
0A060B// time 10:06:12
FF// hundredths not specified
0078// deviation 120 minutes
00// status OK
F3307E

Sent frame:

7EA027957598B8DBE6E600C1018100080000010000FF0200090C07D20C04030A060BFF007
80062FB7E

Comments:
7EA027957598B8DBE6E600
C101// SET.normal request
81
0008
0000010000FF
0200// set the 2nd attribute, time
090C07D20C04030A060BFF007800// just SET the value read before
62FB7E

Received frame:

7EA0107595B85101E6E700C501810036CF7E

Comments:
7EA0107595B85101E6E700
C5018100// SET. response normal, success
36CF7E

Sent frame:

7EA0199575BA183BE6E600C0018100080000010000FF0300BDCE7E

Comments:
7EA0199575BA183BE6E600
C0018100080000010000FF0300// GET.request normal, 3d attribute,
//time_zone
BDCE7E

Received frame:

7EA0137595DA8864E6E700C40181001000078563A7E

Comments:
7EA0137595DA8864E6E700
C4018100// GET.response normal
10// integer 16
0078// 120 minutes
563A7E

Sent frame:

7EA01C9575DC7F53E6E600C1018100080000010000FF030010007801177E

Comments:
7EA01C9575DC7F53E6E600
C1018100080000010000FF// SET.request normal
0300// third attribute
100078 //integer 16, 120 minutes
01177E

Received frame:

7EA0107595FC7105E6E700C501810036CF7E

Comments:
7EA0107595FC7105E6E700
C5018100// SET.response normal, success
36CF7E

Sent frame:

7EA0199575FE383FE6E600C0018100080000010000FF0400B5837E

Comments:
7EA0199575FE383FE6E600
C0018100080000010000FF0400// GET.request normal, 4th attribute, status
B5837E

Received frame:

7EA01275951E1BF8E6E700C40181001100AC5B7E

Comments:
7EA01275951E1BF8E6E700
C40181
001100// Data, unsigned8, OK
AC5B7E

Sent frame:

7EA0199575104831E6E600C0018100080000010000FF05006D9A7E

Comments:
7EA0199575104831E6E600C0018100080000010000FF0500// GET.request normal, 5th attribute,
// daylight savings begin6D9A7E

Received frame:

7EA01E75953053A7E6E700C4018100090CFF4F19F30070000000000F090A7E

Comments:
7EA01E75953053A7E6E700
C40181
00090C// data, octet string(12)
FFFFFFFFFFFFFFFFFF800000// insignificant value
090A7E

Sent frame:

7EA0199575325833E6E600C0018100080000010000FF060005B07E

Comments:
7EA0199575325833E6E600C0018100080000010000FF0600// GET.request normal, 6th attribute,
// daylight_savings_end05B07E

Received frame:

7EA01E75955247E7E6E700C4018100090CFF4F19F30070000000000F090A7E

Comments:
7EA01E75955247E7E6E700
C40181
00090C// data, octet string (12)
FFFFFFFFFFFFFFFFFF800000// insignificant value
090A7E

Sent frame:

7EA0199575546835E6E600C0018100080000010000FF0700DDA97E

Comments:
7EA0199575546835E6E600C0018100080000010000FF0700// GET.request normal, 7th attribute, //daylight savings deviationDDA97E

Received frame:

7EA0127595744734E6E700C40181000F002D547E

Comment:
7EA0127595744734E6E700
C40181
000F00//data, integer8, 0
2D547E

Sent frame:

7EA0199575767837E6E600C0018100080000010000FF0800152A7E

Comments:

```
7EA0199575767837E6E600
C0018100080000010000FF0800// GET.request normal, 8th attribute,
//daylight_savings_enabled
152A7E
```

Received frame:

7EA0127595965BF0E6E700C401810003008DFD7E

Comments:

```
7EA0127595965BF0E6E700
C40181
000300// data, boolean, FALSE
8DFD7E
```

Sent frame:

7EA0199575980839E6E600C0018100080000010000FF0900CD337E

Comments:

```
7EA0199575980839E6E600
C0018100080000010000FF0900// GET.request normal, 9th attribute,
//clock_base
CD337E
```

Received frame:

7EA0127595B82738E6E700C40181001601 2D077E

Comments:

```
7EA0127595B82738E6E700
C40181
001601// data, enumerated, 1, internal crystal
2D077E
```

PPP

PPP begins with LCP (Link Control Protocol). The first step, after a communications channel becomes active, is for each side to send a Configure-Request packet. This is a standard type of LCP packet that means the sender is requesting configuration information. Assuming all goes well, each side should respond in turn with a Configure-ACK packet. As soon as both sides have sent a Configure-ACK packet, the link is considered established, and the exchange moves on to the authentication phase. (If one side does not accept the other side's Configure-Request packet, it will respond with a Configure-Reject or a Configure-NAK packet.)

Authentication is actually optional; However, virtually all PPP servers will implement it, since otherwise anybody could log in and start using the system without authorization to do so. If authentication is used, the type of authentication required will actually have already been specified; It is part of the Configure-Request packets that were exchanged earlier. The Configure-Request packet must explicitly describe if (for example) PAP or CHAP authentication is desired. Thus, both systems already know what kind of authentication to use once the authentication phase is reached.

Once the authentication phase is complete, LCP has done its job, and the NCPs (Network Control Protocols) which the client wishes to use may begin. For most dial-up Internet access situations, the main NCP used would be IPCP (IP Control Protocol).

So what exactly goes into an LCP packet? A very basic LCP Configure-Request packet is this:

```
01 01 00 04
```

This is the shortest possible Configure-Request packet. The first byte of an LCP packet indicates what type of message it contains. The code for Configure-Request is 1, so the first byte of a Configure-Request packet must always be 1. The next byte is an identifier. This must be unique for each separate LCP packet sent over the connection, and can be arbitrarily set to anything. In this case, we're using 1 (a logical choice for the first packet). The next two bytes indicate the size of the entire LCP packet, in bytes. In this case, the whole packet is 4 bytes (the smallest possible LCP packet). Thus, 0004 is used, and we are left with a complete LCP packet.

Unfortunately, you cannot just dial in to your ISP and send a raw LCP packet, because they must be encapsulated in PPP format, which is considerably more complicated. The standard for dial-up ISPs is to use "PPP in HDLC-like framing", defined in RFC 1662. Under this system, each byte of actual data to be sent is to be preceded by a 7Dh character (called a "control escape" character), and the actual bytes are to be XORed with 20h before sending. (I know, who's the genius who came up with THAT arrangement? It seems ludicrous, but that's how PPP works.) So, our original LCP byte pattern, under this system, becomes this:

```
7D 21 7D 21 7D 20 7D 24
```

This is almost a complete PPP frame, except for two additions: The beginning and the end. A PPP LCP frame is supposed to begin with the following six bytes: 7E FF 7D 23 C0 21. Also, each frame is to be ended with a 7E character. Adding these to the previous byte string, we now have:

```
7E FF 7D 23 C0 21 7D 21 7D 21 7D 20 7D 24 7E
```

This is a complete PPP frame, which you can transmit to a dial-up ISP as soon as the modem has connected, which will be a valid Configure-Request LCP packet to it. Sending this to the ISP will probably make it begin transmitting Configure-Request packets back to you (if it doesn't automatically do so upon modem connect), and assuming you send it a Configure-ACK, and it sends you a Configure-ACK as well, an LCP link has been established.

Now, of course, you've got to understand what the ISP sends you back. To demonstrate, here's a fairly typical Configure-Request packet that you'll see from a typical ISP. Slight variations are common, but this shows most of the configuration options you'll see out there:

```
7E FF 7D 23 C0 21 7D 21 0F 7D xx xx 7D 22 7D 26 7D xx 7D xx 7D xx 7D xx
7D 23 7D 24 C0 23 7D 25 7D 26 xx xx xx xx 7D 27 7D 22 7D 28 7D 22
(If PPP Multilink Protocol (MP) is used): 7D 31 7D 24 7D xx xx
(If endpoint discriminator option is used): 7D 33 7D xx 7D 21 blah
(Final 3 bytes, consisting of FCS and closing 7E): xx xx 7E
```

The first six bytes are, again, a signal of the beginning of a PPP LCP frame. Next, we have 7D 21, which is an LCP byte 1, meaning this is a Configure-Request message. Immediately after the 21 is a byte 0F, which is used for the identifier. (Different ISPs may choose different identifier numbers, but it doesn't matter, it's just an arbitrary number for reference.) Although technically, according to the RFC, the identifier number *should* be preceded by another 7D character and XORed with 20, it seems to be in fashion for ISPs to do neither. Thus, the identifier comes right after the 21, and is transmitted in its true form, not XORed with 20.

After that comes two more LCP bytes; These are for the size of the entire LCP packet, in bytes. The actual numbers are simply shown as xx here, since they will vary. Although technically, you're supposed to use a 7D character for each new character, it seems to be common practice among ISPs to transmit both size bytes at once, so instead of 7D xx 7D xx, you might just see it without the second 7D, to wit: 7D xx xx.

The first three standard LCP fields (packet code, packet identifier, and packet length) have now been transmitted, and everything following are the configuration options. You can tell which configuration option is being configured by the first byte of it. In this case, the next byte after the size bytes is an LCP byte 2, which is the code for an Async Control Character Map (ACCM). Next is a 6, indicating that this whole option is 6 bytes long. The 2 for the code and the 6 for the length count as part of that, meaning the rest of the code is 4 bytes long. Following that are 4 LCP bytes, which collectively form the actual ACCM. (Don't worry too much about what the ACCM itself means right now.)

Next we have an LCP byte 3, which is the code for indicating authentication protocol. After that is a 4, indicating this field is 4 bytes. The final 2 bytes come immediately without being 7D'd, and they are C0 23, which is the code for PAP (Password Authentication Protocol). If CHAP were desired, C2 23 would have been used instead. (PAP and CHAP were originally documented in RFC 1334, "PPP Authentication Protocols". The CHAP specification was later updated in RFC 1994.)

Next is an LCP byte 5, which is the code for a magic number. After that is a 6, meaning this field is 6 bytes long (including these actual code and length bytes). Right after the 6 come four bytes which together make up the actual magic number itself. (Does it seem odd that the ACCM had each byte preceded by a 7D, while the magic number is transmitted directly and without modification? It's rather silly, but that's how this stuff is implemented.)

Next comes LCP 7 and 2. These two bytes, together, are an indication that the ISP is letting you know it wants to use Protocol Field Compression (PFC). After that is an 8 and a 2, which lets you know it also wants to use Address and Control Field Compression (ACFC).

There are other things which most ISPs nowadays include in the frame as well, most of which are documented in RFC 1990, "The PPP Multilink Protocol (MP)". Most common among these are the MRRU (Maximum Received Reconstructed Unit) option and the endpoint discriminator option. The MRRU begins with an LCP byte 11, which indicates the MRRU LCP option. (RFC 1990 states that it's a 17, but it's talking decimal, and we're talking hexadecimal here. 17 decimal equals 11 hexadecimal.) After that is an LCP byte indicating the size of this option, which is always 4. After that, two bytes in succession (which are not separated by a 7D, although the first one is preceded by a 7D) which indicate the actual MRRU size. If the system actually sends the MRRU option, this indicates that the system sending it implements the PPP Multilink Protocol. If not rejected, the system will continue to use the PPP MP.

The other PPP MP option commonly configured at this stage is the endpoint discriminator option. This starts with an LCP byte 13. (Again, RFC 1990 says this should be a 19, but it's talking decimal. 19 decimal equals 13 hexadecimal.) After that is an LCP byte stating the size of this option, which is always 3 plus the length of the actual address specified in this option (we'll get to the address in a moment). The next LCP byte is usually a 1, indicating a locally-assigned address. Immediately following the 1 is a succession of bytes which are the actual address of the system specified in this option. (These bytes are represented as "blah" in the sample PPP frame above.)

The PPP frame closes with two bytes which you can safely ignore for now (they're the FCS (Frame Check Sequence)). The final byte is 7E, which always closes the PPP frame, just as it opens a new one. So, now you have seen a fairly realistic PPP frame from an ISP, and you know what to expect in the real world (pretty much). So how do we acknowledge this to the ISP? With a Configure-ACK packet. Making a Configure-ACK packet is actually quite simple, because all we need to do is mirror the options that were sent to us. In fact, the complete packet would look almost exactly the same as the Configure-Request packet it is acknowledging, except that instead of beginning with an LCP byte 1, it begins with an LCP byte 2. The identifier byte must be the same, the length will be the same, and all the options must be transmitted back the same way, so the other side of the connection understands that we are acknowledging all of them. Our complete PPP frame that we transmit back, then, would look like this:

```
7E FF 7D 23 C0 21 7D 22 0F 7D xx xx 7D 22 7D 26 7D xx 7D xx 7D xx 7D xx
7D 23 7D 24 C0 23 7D 25 7D 26 xx xx xx xx 7D 27 7D 22 7D 28 7D 22
(If PPP Multilink Protocol (MP) is used): 7D 31 7D 24 7D xx xx
(If endpoint discriminator option is used): 7D 33 7D xx 7D 21 blah
```

```
xx xx 7E
```

Note that the only difference is, the eighth byte has been made 22 instead of 21, making it an LCP byte 2 instead of a 1. As soon as Configure-ACK packets like this have been sent by each side, the LCP link is open and can continue to authentication with PAP or CHAP, or to networking with an NCP if no authentication was specified in the Configure packets.

PPP General Frame Format

All messages sent using PPP can be considered either *data* or *control information*. The word "data" describes the higher-layer datagrams we are trying to transport here at layer two; this is what our "customers" are giving us to send. Control information is used to manage the operation of the various protocols within PPP itself. Even though different protocols in the PPP suite use many types of frames, at the highest level they all fit into a single, *general frame format*.

In the overview of PPP I mentioned that the basic operation of the suite is based on the ISO High-Level Data Link Control (HDLC) protocol. This becomes very apparent when we look at the structure of PPP frames overall—they use the same basic format as HDLC, even to the point of including certain fields that aren't strictly necessary for PPP itself. The only major change is the addition of a new field to specify the protocol of the encapsulated data. The general structure of PPP frames is defined in RFC 1662, a "companion" to the main PPP standard RFC 1661.

The general frame format for PPP, showing how the HDLC framing is applied to PPP, is described in Table 1 and illustrated in Figure 2.

Table 1: PPP General Frame Format

Field Name	Size (bytes)	Description
Flag	1	***Flag:*** Indicates the start of a PPP frame. Always has the value "01111110" binary (0x7E hexadecimal, or 126 decimal).
Address	1	***Address:*** In HDLC this is the address of the destination of the frame. But in PPP we are dealing with a direct link between two devices, so this field has no real meaning. It is thus always set to "11111111" (0xFF or 255 decimal), which is equivalent to a broadcast (it means "all stations").
Control	1	***Control:*** This field is used in HDLC for various control purposes, but in PPP it is set to "00000011" (3 decimal).
Protocol	2	***Protocol:*** Identifies the protocol of the datagram encapsulated in the Information field of the frame. See below for more information on the *Protocol* field.
Information	Variable	***Information:*** Zero or more bytes of payload that contains either data or control information, depending on the frame type. For regular PPP data frames the network-layer datagram is encapsulated here. For control frames, the control

		information fields are placed here instead.
Padding	Variable	***Padding:*** In some cases, additional dummy bytes may be added to pad out the size of the PPP frame.
FCS	2 (or 4)	***Frame Check Sequence (FCS):*** A checksum computed over the frame to provide basic protection against errors in transmission. This is a CRC code similar to the one used for other layer two protocol error protection schemes such as the one used in Ethernet. It can be either 16 bits or 32 bits in size (default is 16 bits). The FCS is calculated over the *Address*, *Control*, *Protocol*, *Information* and *Padding* fields.
Flag	1	***Flag:*** Indicates the end of a PPP frame. Always has the value “01111110” binary (0x7E hexadecimal, or 126 decimal).

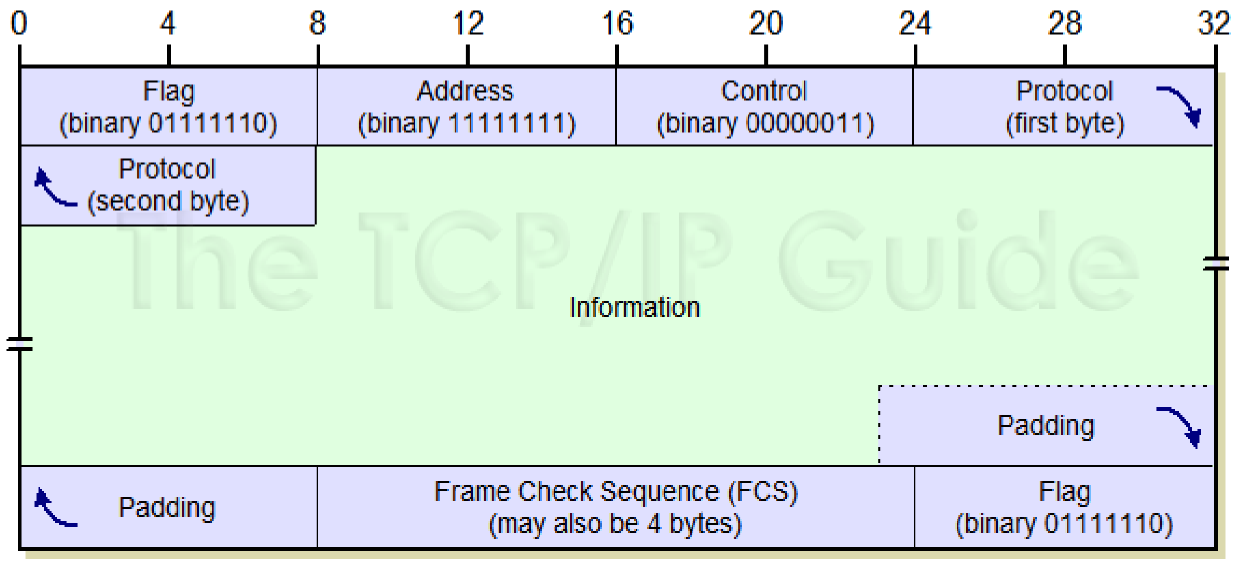
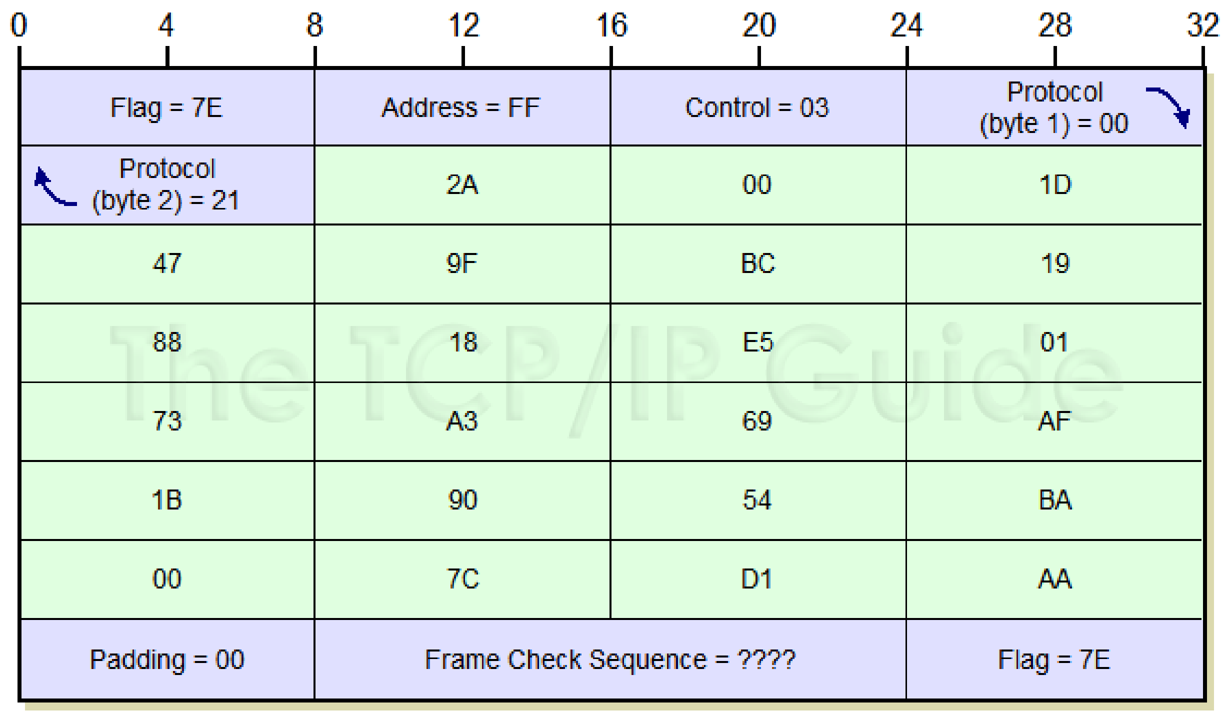


Figure 2: PPP General Frame Format

All PPP frames are built upon the general format shown above. The first three bytes are fixed in value, followed by a two-byte *Protocol* field that indicates the frame type. The variable-length *Information* field is formatted in a variety of ways, depending on the PPP frame type. *Padding* may be applied to the frame, which concludes with an *FCS* field of either 2 or 4 bytes (2 bytes shown here) and a trailing Flag value of 0x7E. See Figure 3 for an example of how this format is applied.

[Figure 3](#) shows one common application of the PPP general frame format: carrying data.The value 0x0021 in the *Protocol* field marks this as an IPv4 datagram. This sample has one byte of *Padding* and a 2-byte *FCS* as well. (Obviously real IP datagrams are longer than the 23 bytes shown here! These bytes are arbitrary and don't represent a real datagram.)


0
4
8
12
16
20
24
28
32

Flag = 7E	Address = FF	Control = 03	Protocol (byte 1) = 00
Protocol (byte 2) = 21	2A	00	1D
47	9F	BC	19
88	18	E5	01
73	A3	69	AF
1B	90	54	BA
00	7C	D1	AA
Padding = 00	Frame Check Sequence = ????		Flag = 7E

Figure 3: Sample PPP Data Frame

This sample PPP data frame has a value of 0x0021 in the *Protocol* field, indicating it is an IP datagram (though the actual data is made up and not a real IP message.)