Single Page

Top Document: UUCP Internals Frequently Asked Questions
Previous Document: UUCP Protocol
Next Document: UUCP `f' Protocol

[Usenet FAQs | Search | Web FAQs | Documents | RFC Index]

UUCP `g' Protocol

```
UUCP `g' Protocol
=================

The `g' protocol is a packet based flow controlled error correcting
protocol that requires an eight bit clear connection.  It is the
original UUCP protocol, and is supported by all UUCP implementations.
Many implementations of it are only able to support small window and
packet sizes, specifically a window size of 3 and a packet size of 64
bytes, but the protocol itself can support up to a window size of 7 and
a packet size of 4096 bytes.  Complaints about the inefficiency of the
`g' protocol generally refer to specific implementations, rather than
to the correctly implemented protocol.

The `g' protocol was originally designed for general packet drivers,
and thus contains some features that are not used by UUCP, including an
alternate data channel and the ability to renegotiate packet and window
sizes during the communication session.

The `g' protocol is spoofed by many Telebit modems.  When spoofing is
in effect, each Telebit modem uses the `g' protocol to communicate with
the attached computer, but the data between the modems is sent using a
Telebit proprietary error correcting protocol.  This allows for very
high throughput over the Telebit connection, which, because it is
half-duplex, would not normally be able to handle the `g' protocol very
well at all.  When a Telebit is spoofing the `g' protocol, it forces
the packet size to be 64 bytes and the window size to be 3.

This discussion of the `g' protocol explains how it works, but does not
discuss useful error handling techniques.  Some discussion of this can
be found in Jamie E. Hanrahan's paper, cited above.

All `g' protocol communication is done with packets.  Each packet
begins with a six byte header.  Control packets consist only of the
header.  Data packets contain additional data.

The header is as follows:

`\020'
     Every packet begins with a `^P'.

K (1 <= K <= 9)
     The K value is always 9 for a control packet.  For a data packet,
     the K value indicates how much data follows the six byte header.
     The amount of data is 2 ** (K + 4), where ** indicates
```

exponentiation. Thus a K value of 1 means 32 data bytes and a K value of 8 means 4096 data bytes. The K value for a data packet must be between 1 and 8 inclusive.

checksum low byte
checksum high byte
The checksum value is described below.

control byte
The control byte indicates the type of packet, and is described below.

xor byte
This byte is the xor of K, the checksum low byte, the checksum high byte and the control byte (i.e., the second, third, fourth and fifth header bytes). It is used to ensure that the header data is valid.

The control byte in the header is composed of three bit fields, referred to here as TT (two bits), XXX (three bits) and YYY (three bits). The control is TTXXXYYY, or `(TT << 6) + (XXX << 3) + YYY'.

The TT field takes on the following values:

`0'
This is a control packet. In this case the K byte in the header must be 9. The XXX field indicates the type of control packet; these types are described below.

`1'
This is an alternate data channel packet. This is not used by UUCP.

`2'
This is a data packet, and the entire contents of the attached data field (whose length is given by the K byte in the header) are valid. The XXX and YYY fields are described below.

`3'
This is a short data packet. Let the length of the data field (as given by the K byte in the header) be L. Let the first byte in the data field be B1. If B1 is less than 128 (if the most significant bit of B1 is 0), then there are `L - B1' valid bytes of data in the data field, beginning with the second byte. If `B1 >= 128', let B2 be the second byte in the data field. Then there are `L - ((B1 & 0x7f) + (B2 << 7))' valid bytes of data in the data field, beginning with the third byte. In all cases L bytes of data are sent (and all data bytes participate in the checksum calculation) but some of the trailing bytes may be dropped by the receiver. The XXX and YYY fields are described below.

In a data packet (short or not) the XXX field gives the sequence number of the packet. Thus sequence numbers can range from 0 to 7, inclusive. The YYY field gives the sequence number of the last correctly received packet.

Each communication direction uses a window which indicates how many unacknowledged packets may be transmitted before waiting for an acknowledgement. The window may range from 1 to 7, and may be different in each direction. For example, if the window is 3 and the last packet

acknowledged was packet number 6, packet numbers 7, 0 and 1 may be sent but the sender must wait for an acknowledgement before sending packet number 2. This acknowledgement could come as the YYY field of a data packet, or as the YYY field of a `RJ' or `RR' control packet (described below).

Each packet must be transmitted in order (the sender may not skip sequence numbers). Each packet must be acknowledged, and each packet must be acknowledged in order.

In a control packet, the XXX field takes on the following values:

1 `CLOSE'
The connection should be closed immediately. This is typically sent when one side has seen too many errors and wants to give up. It is also sent when shutting down the protocol. If an unexpected `CLOSE' packet is received, a `CLOSE' packet should be sent in reply and the `g' protocol should halt, causing UUCP to enter the final handshake.

2 `RJ' or `NAK'
The last packet was not received correctly. The YYY field contains the sequence number of the last correctly received packet.

3 `SRJ'
Selective reject. The YYY field contains the sequence number of a packet that was not received correctly, and should be retransmitted. This is not used by UUCP, and most implementations will not recognize it.

4 `RR' or `ACK'
Packet acknowledgement. The YYY field contains the sequence number of the last correctly received packet.

5 `INITC'
Third initialization packet. The YYY field contains the maximum window size to use.

6 `INITB'
Second initialization packet. The YYY field contains the packet size to use. It requests a size of 2 ** (YYY + 5). Note that this is not the same coding used for the K byte in the packet header (it is 1 less). Most UUCP implementations that request a packet size larger than 64 bytes can handle any packet size up to that specified.

7 `INITA'
First initialization packet. The YYY field contains the maximum window size to use.

To compute the checksum, call the control byte (the fifth byte in the header) C.

The checksum of a control packet is simply `0xaaaa - C'.

The checksum of a data packet is `0xaaaa - (CHECK ^ C)', where `^' denotes exclusive or, and CHECK is the result of the following routine as run on the contents of the data field (every byte in the data field participates in the checksum, even for a short data packet). Below is the routine used by an early version of Taylor UUCP; it is a slightly

modified version of a routine which John Gilmore patched from G.L. Chesson's original paper. The `z' argument points to the data and the `c' argument indicates how much data there is.

```
     int
     igchecksum (z, c)
          register const char *z;
          register int c;
     {
       register unsigned int ichk1, ichk2;

       ichk1 = 0xffff;
       ichk2 = 0;

       do
         {
           register unsigned int b;

           /* Rotate ichk1 left.  */
           if ((ichk1 & 0x8000) == 0)
             ichk1 <<= 1;
           else
             {
               ichk1 <<= 1;
               ++ichk1;
             }

           /* Add the next character to ichk1.  */
           b = *z++ & 0xff;
           ichk1 += b;

           /* Add ichk1 xor the character position in the buffer counting from
              the back to ichk2.  */
           ichk2 += ichk1 ^ c;

           /* If the character was zero, or adding it to ichk1 caused an
              overflow, xor ichk2 to ichk1.  */
           if (b == 0 || (ichk1 & 0xffff) < b)
             ichk1 ^= ichk2;
         }
       while (--c > 0);

       return ichk1 & 0xffff;
     }
```

When the `g' protocol is started, the calling UUCP sends an `INITA' control packet with the window size it wishes the called UUCP to use. The called UUCP responds with an `INITA' packet with the window size it wishes the calling UUCP to use. Pairs of `INITB' and `INITC' packets are then similarly exchanged. When these exchanges are completed, the protocol is considered to have been started.

Note that the window and packet sizes are not a negotiation. Each system announces the window and packet size which the other system should use. It is possible that different window and packet sizes will be used in each direction. The protocol works this way on the theory that each system knows how much data it can accept without getting overrun. Therefore, each system tells the other how much data to send before waiting for an acknowledgement.

When a UUCP package transmits a command, it sends one or more data packets. All the data packets will normally be complete, although some UUCP packages may send the last one as a short packet. The command string is sent with a trailing null byte, to let the receiving package know when the command is finished. Some UUCP packages require the last byte of the last packet sent to be null, even if the command ends earlier in the packet. Some packages may require all the trailing bytes in the last packet to be null, but I have not confirmed this.

When a UUCP package sends a file, it will send a sequence of data packets. The end of the file is signalled by a short data packet containing zero valid bytes (it will normally be preceeded by a short data packet containing the last few bytes in the file).

Note that the sequence numbers cover the entire communication session, including both command and file data.

When the protocol is shut down, each UUCP package sends a `CLOSE' control packet.

Top Document: UUCP Internals Frequently Asked Questions
Previous Document: UUCP Protocol
Next Document: UUCP `f' Protocol

Single Page

[Usenet FAQs | Search | Web FAQs | Documents | RFC Index]

Send corrections/additions to the FAQ Maintainer:
ian@airs.com (Ian Lance Taylor)

Last Update June 06 2007 @ 01:06 AM