Internet Draft

C. Burri
Synecta Informatik
Expires Jan 2002

Handling IRC continuation message lines
draft-burri-irc-continuation-message-lines-00.txt

Status of this Memo

Abstract

Due to the way the IRC protocol is implemented, it may occur that a server sends incomplete messages to a client, so called continuation message lines.

There seems to exist confusion about how to handle continuation message lines; many implementations are broken and do not respect them at all. Others rely on timers to complete continuation lines, which is not recommended due to the asyncronous nature of IRC communications.

This Memo proposes an algorithm to handle continuation lines received from an IRC connection in such way that no timers are needed, and is intended as a supplement to the existing RFC 1459 which describes the Internet Relay Chat protocol.

Copyright Notice

Table of Contents

1. Introduction

Current IRC implementations utilize input and output buffers for async network IO, whereas the input buffers are always processed first. All output gets stacked in the send queue, and is not sent to the client until processing of the input buffer has completed. This process helps TCP build larger packets, as possibily multiple messages are bundled into one network transmission (TCP segment). For more information consult RFC 1459, Sections 8.2, 8.3

The same process can however lead to incomplete messages, which are cropped due to TCP limitations, namely the TCP window size. Such lines appear incomplete to the client, which does not normally cause any problems by itself. However, the line that follows the truncated line will be incomplete too, only containing data that did not fit within the last TCP segment. This line is special in such way that, if it is treatened like a normal line, then this might lead to arbitrary data being parsed as a complete message from the server.

This circumstance has been observed to cause problems in various IRC clients. Most of them seem to completely ignore the existence of the problem, which may possibly result in severe brain damage, or even loss of chanop status incase the broken implementation is being used in a bot that maintains an IRC channel, since it is not clearly defined what happens when a continuation message line is received. This undefined behaviour could possibly be exploited, by trying to make the implementation believe that it received a message from somewhere, where infact the true origin is spoofed.

The algorithm proposed in this Memo has been designed to reassemble continuation message lines before processing them in the message parser. It does this without the use of any timers or delays, which could lead to loss of data, incase the reassembly timeout has been set to a low value; or to undesirable high delays in reading from the network, incase the reassembly timeout has been choosen too high.

The proposed algorithm relies on the facts that:

- for any incomplete message line, there will be a resulting completion message line received in the next TCP segment that is read from the network.

- the transmission of the completion message line following the incomplete line will always occur before any other network transmission occurs, or in other words, the completion message line will always be the first line of the next delivered TCP segment.

The proposed algorithm does not depend on any particular programming language. Instead, it is designed to work with every programming language that has provides buffers (variables) and comparision tests.

It can be implemented as a preprocessor that is located infront of the IRC message parser, in the data stream.

It seems further notable to the author that the proposed algorithm is Public Domain property and may be freely used and implemented without paying any fee for whatsoever to anyone.

2. Handling continuation message lines

In order to reassemble continuation message lines, they must be detected in a reliable way. After detecting, they need to be marked as incomplete, and stored in a temporary buffer, for later reassembly.

2.1. IRC Message Format

IRC RFC 1459 defines the message format as follows:

```
<message>  ::= [':' <prefix> <SPACE> ] <command> <params> <crlf>
<prefix>   ::= <servername> | <nick> [ '!' <user> ] [ '@' <host> ]
<command>  ::= <letter> { <letter> } | <number> <number> <number>
<SPACE>    ::= ' ' { ' ' }
<params>   ::= <SPACE> [ ':' <trailing> | <middle> <params> ]
<middle>   ::= <Any *non-empty* sequence of octets not including
                SPACE or NUL or CR or LF, the first of which may
                not be ':'>
<trailing> ::= <Any, possibly *empty*, sequence of octets not
                including NUL or CR or LF>

<crlf>     ::= CR LF
```

As we can see from the above representation, every complete IRC

message must end in the sequence <crlf>. Section 8 of RFC 1459 also reports the usage of either CR *or* LF as message delimiter.

It might be a good idea for any implementation to accept all three variants; for the sake of simplicity we will however refer to CRLF as the message delimiter in this document.

2.2. Discovery

Once we know how IRC messages are delimited, we can check any IRC message line for completeness. Any complete line must end in the line delimiter sequence. If a given IRC message line does not end in that sequence, then it must have been truncated.

To speed up performance, the proposed discovery algorithm performs the end delimiter test on each received TCP segment, instead of each received line, since each received TCP segment must also end in a CRLF sequence if the last contained line has not been truncated by the sending TCP.

Notice that we are not looking for continuation lines, since the algorithm cannot recognize a continuation line by itself. That is impossible to do because of the arbitrary structure of that line. The line might infact be composed of data that has been sent to either a channel or to the client via PRIVMSG or other means.

The solution to this problem is to recognize the lines that have been truncated, and store them for reassembly, instead of parsing them. The discovery of a truncated line is also to be used to recognize the following line as the continuation message line.

Thus, the discovery algorithm sets a flag, whenever a TCP segment, that containis a truncated line, is received.

2.3. Reassembly

After recieving a TCP segment and discovering incomplete lines, the preprocessor checks a buffer (which holds data if the previous segment did contain a truncated line; described below) for the presence of any data. If there is data in the buffer, then the preprocessor does append the received data to the data in the buffer (this will effectively reassemble the truncated- and the continuation line), cycles the buffer (so it is empty after reassembly took place) and passes the resulting data to the next step.

The next step parses the received data segment into IRC messages by splitting the data on each occurence of the delimiter sequence CRLF.

If the recieved, tested, possibly reassembled and then splitted message did not contain an incomplete last line (the discovery flag

was not set), then the preprocessor calls the message parser normally for each received line.

If the received and splitted message has been marked as incomplete (discovery flag set), then the preprocessor calls the IRC message parser for each received message (line), but not for the last message (which will lack the message delimiter, because it is incomplete). The preprocessor does not call the message parser with the incomplete line, instead it resets the incomplete flag from the end-delimiter test and temporarily stores the incomplete line until arrival of the next TCP segment.

3. Expiration Notice

This document expires in Jan 2002.

4. Credits and Authors' Address

Christian Burri
jun. System & Network Engineer
Synecta Informatik AG
Zwinglistrasse 3
9000 St. Gallen
SWITZERLAND

Email: christian.burri@synecta.ch

Special credits to Jarkko Oikarinen and all other contributors for creating such a cool thing as IRC :)