

## Re: problem reading TCP packets on socket

---

*Source:* <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2005-12/msg00970.html>

---

- *From:* "John C. Bollinger" <[jobollin@xxxxxxxxxxx](mailto:jobollin@xxxxxxxxxxx)>
  - *Date:* Tue, 06 Dec 2005 21:18:59 -0500
- 

antoine wrote:

I'm having a pretty annoying issue with a TCP connection in a client/server environment.

my client connects to a server that sends it messages of variable lengths.

on reception of each message, the client strips down the message in the following manner:

1. the first two characters of the message are first read to indicate the length of the message (as in the "decodeLength" method in the code below)
2. knowing the length of the message, I read the appropriate number of characters on my bufferedReader, and I start over.

So far, so good, though you didn't actually include the implementation of decodeLength(). Your approach is a standard one.

this method has proven effective for a very long time for me, but recently I installed a new server on a slightly different environment (stable though), and started experiencing strange things. after tracking down the errors, I realized that problems arised when TCP packets were not received properly, and the TCP stack asked for a resend.

I haven't exactly figured out what's happening at the TCP layer yet, but I "trust" the TCP stack, and think my client should be able to handle such situation, however it does not.

[...]

## Re: problem reading TCP packets on socket

Your client /could/ easily handle the situation, but it is buggy.

```
public char[] receive() throws IOException {
    char[] bufLen = new char[2];
    try {
        _bufferedReader.read(bufLen);
```

The above is broken, though in practice it is not likely to fail often.  
(See below.)

```
    }
    catch (SocketException se) {
        _exit = true;
        se.printStackTrace();
        return null;
    }
```

```
    int len = decodeLength(bufLen);
```

```
    try {
        char[] _rawMsg = new char[len];
        _bufferedReader.read(_rawMsg);
```

This bit has the same bug as the earlier read, but it is more likely to be affected. (See explanation below.)

```
        return _rawMsg;
    }
    catch (SocketException se) {
        se.printStackTrace();
        _exit = true ;
    }
    return null;
}
```

You should read the docs of `Reader.read(char[])`. You will then see that the method returns an `int`, which indicates the number of bytes actually read. This is by no means a formality: the method promises to block until it receives at least one character or sees the end of the stream, but it

## Re: problem reading TCP packets on socket

is by no means guaranteed to fill the array, regardless of the number of characters the sender is sending. There are many reasons why one logical message might be split up over multiple reads, and the TCP behavior you observed is just one of them. Using a `BufferedReader` as you have done improves the chances of getting a whole array in one read, but still does not ensure it.

The usual idiom for reading a fixed number of chars from a character stream is this:

```
Reader reader;
char[] buf;
int numberToRead;
```

[Assign suitable values to the variables]

```
// Read repeatedly until the expected number of chars has been read:
for (int totalCharsRead = 0; totalCharsRead < numberToRead; ) {
    int numberLeft = numberToRead - totalCharsRead;
    int numberRead = reader.read(buf, totalCharsRead, numberLeft);

    if (numberRead < 0) {
        // premature end of data
        break;
    } else {
        totalCharsRead += numberRead;
    }
}
```

You can pop that into a handy method so that you don't need to repeat it every time, but you do need to use something like it for *both* of the reads in your method. Also, when you do it that way you are effectively buffering the input manually, so the `BufferedReader` doesn't provide much advantage over an unbuffered stream.

By the way, the appropriate technique for reading a fixed number of bytes from a byte stream is completely parallel to the above.

```
--
John Bollinger
jobollin@xxxxxxxxxxxxx
.
```