Two steps forward, one step back..

After our last success, we perhaps celebrated too soon, and lay on our laurels as they say. OK the client works fine against that one TLS1.3 server, but we knew in our hearts that was far from a guarantee that it would work against other TLS1.3 servers.

First thing is to find servers which support TLS1.3. To that end this site is useful -

https://www.cdn77.com/tls-test

So I tried out a few web pages, starting with https://www.tii.ae/ . Surely this one supports TLS1.3. Whoops, no it doesn't(?) Next tried https://www.bbc.com , and yes this one does. Next I tried the site of a company I did some work for, and yes it supported TLS1.3 as well.

Next I pointed our client at this last website – and the handshake crashed. Sort of expected that. Next went to

https://raymii.org/s/tutorials/OpenSSL_test_TLSv1.3_connection_with_s_client.html#toc_1

and directed OpenSSL to do a TLS1.3 handshake with the same website, by executing

```
echo | openssl s_client -debug -tls1_3 -connect www.miracl.com:443
```

Next thing was to pick through the output to figure out what went wrong. And it didn't take too long to find the culprit – *fragmentation*.

Can't say we were not warned. Reading section C.3 "Implementation pitfalls" of the TLS1.3 RFC8446 https://tools.ietf.org/html/rfc8446#appendix-C.3 it asks "Do you correctly handle handshake messages that are fragmented..". And the answer, sadly, is no.

Problem is that the server response may be just too big to fit into a single message. The length encoding in fact sets a limit of 2^{14} bytes, and certificate chains can easily be longer than that. So fragmentation is a necessity.

So now we must take a step back and either reassemble (or defragment) the server response, or deal with the fragments one after the other (which would save space on the client side, but may not be possible). One complication is knowing when the last fragment has been received.

But that's just programming as my mother used to say...