My initial assumption about TLS was that it is a protocol which establishes an encrypted tunnel between client and server, and then it has done its job, and exits, passing control over to a higher level application like HTTP. Turns out that it is not like that at all. A clue is provided by the fact that a client can in some circumstances provide "early" application data in its very first communication. Turns out that there is no clear demarcation between when TLS ends and application data communication begins.

When is TLS finished successfully?

The last stage of TLS1.3 appears to be when the client sends its "Finished" message. Then the server-client fall into the exchange of two-way encrypted application data, which is implementing whatever communications protocol is entered into after TLS has done its job of negotiating suitable session keys to encrypt this secure communication in both directions.

But what if the server is unhappy with that last Client-Finished message? Then the first "application" communication that the client will hear from the Server will in fact be an alert record (not the expected application-data record), indicating that TLS has in fact failed.

I guess the conclusion must be that TLS does not just do its job, hand over to the application, and then disappear. The reality is that TLS does not go away, in fact it oversees the entire application communication, while sitting in the background. As well as application records, alert records and handshake records (for key updates, or resumption ticket provision) may occur at any time, and TLS needs to be ready to take action to process these. And due to disruption by an attacker (like deletion of a record) TLS might have to step in to drop the connection.

But since the application involves two-way communication of application data, this raises another question...

Full duplex or half duplex?

That is should server-client communications proceed on a your-turn-my-turn basis, or should both parties be able to communicate at will? Is it a walkie-talky experience, or a telephone experience?

The TLS 1.3 protocol, like most (all?) cryptographic protocols is intrinsically half-duplex. Each party knows what is is supposed to communicate, and when it is supposed to communicate it. Computing and communicating a handshake response ahead of hearing everything that the other party has to say, would probably lead to insecurities.

However once TLS1.3 has "completed", and both parties are set up with their symmetric cryptographic keys for communication in both directions, then it may make sense to switch to full-duplex. Indeed common communications protocols, like HTTP, can be naturally full duplex.

Switching to full duplex means transitioning from blocking TCP/IP sockets, to non-blocking sockets. The difference being that a non-blocking sockets does not hang around waiting for expected input. If there is none it goes away and does something else, maybe providing some output of its own. A blocking socket on the other hand allows a multi-tasking operating system (if there is one) to go off and do something else with its precious clock cycles, while waiting for some data to turn up.

Half-duplex is simpler, and since a single I/O buffer can be shared between input and output, it uses less resources (good for IoT). However full-duplex covers all cases, and takes us seamlessly into what may be a full-duplex application protocol.

For the moment I think we will stick with half-duplex. In the IOT world the protocols of immediate interest to us are likely to be half-duplex anyway. But it is an issue we need to think some more about.