

[Does tokio TcpStream require special cancellation handling?](#)

[help](#)

[jcreswell-eos](#) July 15, 2025, 9:52pm 1

I've got a tokio::net::TcpStream working inside the future given to a tokio::time::timeout() as follows:

```
async fn stream_comm(url: String, data: &Vec<u8>) -> Result<[u8; 512]> {
    let mut stream = TcpStream::connect(url).await?;
    stream.write_all(data).await?;
    let buf: &mut [u8; 512] = &mut [0; 512];
    stream.read(buf).await?;
    ...
    Ok(*buf)
}

async fn monitored_comms(url: String, data: &Vec<u8>) {
    match tokio::time::timeout(Duration::from_secs(5), stream_comm(url, data)).await {
        Ok(_) => {dbg!("Completed on time");},
        Err(elapsed) => {dbg!("Timed out! Elapsed says {:?}", elapsed);},
    }
    // TODO: cleanup?
}
```

The timeout docs say that when a future times out it is cancelled for you, but I'm worried about the AsyncWriteExt shutdown function -- do I need to call it explicitly somewhere to fully release the stream resources (for a cancelled future or otherwise)? Would I need to do explicit cleanup work if I cancelled the future myself?

[kpreid](#) July 15, 2025, 11:01pm 2

You only need to do something extra if you are concerned with exactly how much data was written to the socket. If stopping any time is okay, then since **the connection is owned by the cancelled future**, there is no problem. All necessary cleanup to avoid resource leaks happens via implicit drops.

You would have a big problem if you wanted to cancel a `write_all()` and then continue using the connection, because you would not know how much of a partial message had been written.

1 Like

[alice](#) July 16, 2025, 7:08am 3

Since you ask about the async shutdown function, let me add a bit about that. Overall, what [@kpreid](#) tells you is correct; you don't have to call it explicitly. In the case of `TcpStream` it doesn't give you anything additional over just dropping it.

The case where `TcpStream::shutdown` is useful is for gracefully closing a connection. This is because it only closes it in one direction. I.e., the other end of the connection will get an EOF when they read data, indicating that you will never send any more data. It doesn't prevent the other peer from writing data, and you can still read (but not write) after shutdown. A fully graceful shutdown happens when both peers have called shutdown.

In short, dropping the `TcpStream` does everything that `shutdown` does (closing read direction), and in fact it does *more* than what `shutdown` does (it closes the write direction too).

4 Likes

[system](#) Closed October 14, 2025, 7:09am 4

This topic was automatically closed 90 days after the last reply. We invite you to open a new topic if you have further questions or comments.

Related topics

Topic	Replies	Views	Activity
Cannot `shutdown` TcpStream after upgrading Tokio from 0.x to 1.x	6	840	August 2, 2021
help			
How to close tcp connection when using tokio help	2	5616	January 26, 2022
Calling shutdown on resolution of Tokio Copy future	3	570	January 20, 2020
Cancellation in Tonic/Tokio: How does it work? help	3	2115	July 11, 2024

- [Home](#)
- [Categories](#)
- [Guidelines](#)
- [Terms of Service](#)