NetworkStream.Write returns immediately - how can I tell when it has finished sending data?

Asked 16 years, 3 months ago Modified 10 years, 8 months ago Viewed 14k times

10

Despite the documentation, NetworkStream.Write does not appear to wait until the data has been sent. Instead, it waits until the data has been copied to a buffer and then returns. That buffer is transmitted in the background.

This is the code I have at the moment. Whether I use ns.Write or ns.BeginWrite doesn't matter - both return immediately. The EndWrite also returns immediately (which makes sense since it is writing to the send buffer, not writing to the network).

```
bool done;
void SendData(TcpClient tcp, byte[] data)
{
    NetworkStream ns = tcp.GetStream();
    done = false;
    ns.BeginWrite(bytWriteBuffer, 0, data.Length, myWriteCallBack, ns);
    while (done == false) Thread.Sleep(10);
}

public void myWriteCallBack(IAsyncResult ar)
{
    NetworkStream ns = (NetworkStream)ar.AsyncState;
    ns.EndWrite(ar);
    done = true;
}
```

How can I tell when the data has actually been sent to the client?

I want to wait for 10 seconds(for example) for a response from the server after sending my data otherwise I'll assume something was wrong. If it takes 15 seconds to send my data, then it will always timeout since I can only start counting from when NetworkStream.Write returns - which is before the data has been sent. I want to start counting 10 seconds from when the data has left my network card.

The amount of data and the time to send it could vary - it could take 1 second to send it, it could take 10 seconds to send it, it could take a minute to send it. The server does send an response when it has received the data (it's a smtp server), but I don't want to wait forever if my data was malformed and the response will never come, which is why I need to know if I'm waiting for the data to be sent, or if I'm waiting for the server to respond.

I might want to show the status to the user - I'd like to show "sending data to server", and "waiting for response from server" - how could I do that?

c# network-programming

Share

Improve this question

Follow

edited Sep 16, 2008 at 0:14

asked Sep 15, 2008 at 22:54

dan gibson

3,655 ● 3 ● 42 ● 57

Have you tried NetworkStream.Write with a 100MB packet? – Calmarius Apr 5, 2011 at 11:57

9 Answers

Sorted by: Highest score (default)

11

I'm not a C# programmer, but the way you've asked this question is slightly misleading. The only way to know when your data has been "received", for any useful definition of "received", is to have a specific acknowledgment message in your protocol which indicates the data has been fully processed.

The data does not "leave" your network card, exactly. The best way to think of your program's relationship to the network is:

> your program -> lots of confusing stuff -> the peer program

A list of things that might be in the "lots of confusing stuff":

- the CLR
- the operating system kernel
- a virtualized network interface
- a switch
- a software firewall
- a hardware firewall
- a router performing network address translation
- a router on the peer's end performing network address translation

So, if you are on a virtual machine, which is hosted under a different operating system, that has a software firewall which is controlling the virtual machine's network behavior - when has the data "really" left your network card? Even in the best case scenario, many of these components may drop a packet, which your network card will need to re-transmit. Has it "left" your network card when the first (unsuccessful)

attempt has been made? Most networking APIs would say no, it hasn't been "sent" until the other end has sent a TCP acknowledgement.

That said, the documentation for NetworkStream.Write seems to indicate that it will not return until it has at least initiated the 'send' operation:

> The Write method blocks until the requested number of bytes is sent or a SocketException is thrown.

Of course, "is sent" is somewhat vague for the reasons I gave above. There's also the possibility that the data will be "really" sent by your program and received by the peer program, but the peer will crash or otherwise not actually process the data. So you should do a `Write` followed by a `Read` of a message that will only be emitted by your peer when it has actually processed the message.

Share Improve this answer Follow

answered Sep 17, 2008 at 3:32

Glyph
31.8k ● 12 ● 90 ● 131

The TCP data package was send always have a underlying TCP ACK from its peer, and this is why we use TCP vs UDP: we always know the status of the connection. Back to the question, if `Write(bytes)` called, later if the TCP ACK was received from other peer, then a notification should pop up, this is the question owner wants. – Shawn Feb 9, 2015 at 7:45

TCP ACK just means that the remote *operating system* (not the remote application) has received some of your data. Or possibly some networking middle-box which is doing TCP buffering to help your remote system. You cannot rely on it for robust delivery of application data. – Glyph Feb 9, 2015 at 19:48

5

TCP is a "reliable" protocol, which means the data will be received at the other end if there are no socket errors. I have seen numerous efforts at second-guessing TCP with a higher level application confirmation, but IMHO this is usually a waste of time and bandwidth.

Typically the problem you describe is handled through normal client/server design, which in its simplest form goes like this...

The client sends a request to the server and does a blocking read on the socket waiting for some kind of response. If there is a problem with the TCP connection then that read will abort. The client should also use a timeout to detect any non-network related issue with the server. If the request fails or times out then the client can retry, report an error, etc.

Once the server has processed the request and sent the response it usually no longer cares what happens - even if the socket goes away during the transaction - because it is up to the client to initiate any further interaction. Personally, I find it very comforting to be the server. :-)

Share Improve this answer Follow

answered Sep 15, 2008 at 23:30

Greg Cottman

658 ● 1 ● 5 ● 7

1

In general, I would recommend sending an acknowledgment from the client anyway. That way you can be 100% sure the data was received, and received correctly.

Share Improve this answer Follow

answered Sep 15, 2008 at 23:04

GEOCHET

21.3k ● 15 ● 77 ● 99

If you have a question, ask a question. Don't leave a question as a comment. – GEOCHET Sep 16, 2020 at 18:14

1

If I had to guess, the NetworkStream considers the data to have been sent once it hands the buffer off to the Windows Socket. So, I'm not sure there's a way to accomplish what you want via TcpClient.

Share

Improve this answer

Follow

edited Sep 15, 2008 at 23:17

answered Sep 15, 2008 at 23:07

Craig Eddy

959 ● 1 ● 7 ● 17

Indeed, a CMD/ACK is the only way to achieve what he wants. – GEOCHET Sep 15, 2008 at 23:19

Then, how should I work with the TcpListener if I want to serve one request only? – Sasuke Uchiha Sep 16, 2020 at 16:46

1

I can not think of a scenario where NetworkStream.Write wouldn't send the data to the server as soon as possible. Barring massive network congestion or disconnection, it should end up on the other end within a reasonable time. Is it possible that you have a protocol issue? For instance, with HTTP the request headers must end with a blank

line, and the server will not send any response until one occurs -- does the protocol in use have a similar end-of-message characteristic?

Here's some cleaner code than your original version, removing the delegate, field, and Thread.Sleep. It preforms the exact same way functionally.

```
void SendData(TcpClient tcp, byte[] data) {
    NetworkStream ns = tcp.GetStream();
    // BUG?: should bytWriteBuffer == data?
    IAsyncResult r = ns.BeginWrite(bytWriteBuffer, 0, data.Length, null, null);
    r.AsyncWaitHandle.WaitOne();
    ns.EndWrite(r);
}
```

Looks like the question was modified while I wrote the above. The .WaitOne() may help your timeout issue. It can be passed a timeout parameter. This is a lazy wait -- the thread will not be scheduled again until the result is finished, or the timeout expires.

Share
Improve this answer
Follow

edited Sep 16, 2008 at 0:28

answered Sep 16, 2008 at 0:15

Alex Lyman
15.9k ● 4 ● 40 ● 42

1

I try to understand the intent of .NET NetworkStream designers, and they must design it this way. After Write, the data to send are no longer handled by .NET. Therefore, it is reasonable that Write returns immediately (and the data will be sent out from NIC some time soon).

So in your application design, you should follow this pattern other than trying to make it working your way. For example, use a longer time out before received any data from the NetworkStream can compensate the time consumed before your command leaving the NIC.

In all, it is bad practice to hard code a timeout value inside source files. If the timeout value is configurable at runtime, everything should work fine.

Share Improve this answer Follow

answered Sep 16, 2008 at 5:55

Lex Li
63k ● 11 ● 123 ● 159

0

How about using the Flush() method.

```
ns.Flush()
```

That should ensure the data is written before continuing.

Share Improve this answer Follow

answered Sep 15, 2008 at 22:57

palehorse

27.4k ● 4 ● 42 ● 49

2 The Flush method implements the Stream.Flush method; however, because NetworkStream is not buffered, it has no affect on network streams. Calling the Flush method does not throw an exception. msdn.microsoft.com/en-us/library/… – GEOCHET Sep 15, 2008 at 23:00

0

Bellow .net is windows sockets which use TCP. TCP uses ACK packets to notify the sender the data has been transferred successfully. So the sender machine knows when data has been transferred but there is no way (that I am aware of) to get that information in .net.

edit: Just an idea, never tried: Write() blocks only if sockets buffer is full. So if we lower that buffers size (SendBufferSize) to a very low value (8? 1? 0?) we may get what we want :)

Share

Improve this answer

Follow

edited Apr 2, 2009 at 0:20

answered Apr 1, 2009 at 22:25

Hex

-1

Perhaps try setting `tcp.NoDelay = true`

Share Improve this answer Follow

answered Sep 15, 2008 at 23:11

sbeskur

2,290 ● 25 ● 24

This tells it not to use Nagal. Nagal is where it waits for ~200ms before sending any data. The idea is that waiting a bit ensures that it is going to send as much data as possible, reducing network usage. You still don't know when the data has been sent. – dan gibson Sep 16, 2008 at 0:01