

## Re: pushing the envelope with sockets

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2006-05/msg04363.html>

---

- *From:* Dan Ritchie <[DanRitchie@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:DanRitchie@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Wed, 24 May 2006 18:31:02 -0700
- 

Sorry Helge, ignore my previous post (no additional content). I wrote a rather lengthy reply, during which time my authentication expired and apparently the wrong data was posted.

Anyway, I am using .Net 1.1, and from what I can see, UDP packets are not buffered in 1.1. The `Socket.ReceiveBufferSize` property you mention is new to 2.0, so it would seem 2.0 does indeed buffer UDP. I have been thinking about trying 2.0 to solve the problem, and even if the same problem persists, it might then be possible to fix it with synchronous I/O.

"Helge Jensen" wrote:

Dan Ritchie wrote:

Thanks for the reply (comments below):

It wouldn't, but CPU is not the ONLY issue. With UDP, if an I/O has not been issued when a packet is received the packet is dropped. Using async I/O and issuing subsequent reads immediately in the handler is the only way I was able to keep up with the amount of data coming in (~30 Mbps in ~1Kb packets).

Then you must be doing some handling before issuing the next read?

I still don't understand why you expect `async-io` to be able to *\*receive\** faster than `sync-io`.

The OS *\*does\** buffer UDP-pacakges, they are not discarded if no-one is receiving on the socket they are received (upto the buffer size), you can even change what happens if the buffer runs full. If

## Re: pushing the envelope with sockets

your data comes in "too fast", probably the simplest way to increase performance is to make the buffer larger via `Socket.ReceiveBufferSize`.

I also don't understand how `async-io` should be able to resume listening faster than `sync io`.

The following code is cleaned from `non-IO`. Can you suggest to me where you dispatch to processing the input, and why using `async-IO` would be better than doing the `*processing*` of the input asynchronously.

(`new_buffer` may be freshly allocated, or reused from a pool or whatnot.

`async:`

```
void continueRead(IAsyncResult r
```