

Re: pushing the envelope with sockets

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2006-05/msg04282.html>

- *From:* Helge Jensen <helge.jensen@xxxxxxx>
 - *Date:* Wed, 24 May 2006 19:56:37 +0200
-

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 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(IAAsyncResult r)
{
```

Re: pushing the envelope with sockets

```
int read = S.EndReceive(r);
if ( read != 0 )
S.BeginReceive(new_buffer, 0, new_buffer.Length,
SocketFlags.None, continueRead, new_buffer);
// process(read)
}
```

sync:

```
int read;
do
{
read = s.Receive(new_buffer);
// AsyncProcess(read)
} while (read != 0);
```

30% or 60% of available CPU (ie, out of 100% for a given time slice in perfmon). CPU usage holds steady at under 30% on average for, say about 30 seconds, then jumps to 60% on average and stays that way. Again, the point is there is nothing that changes in the load to explain the additional usage.

Okay, so during *one* run, the cpu spent increases when input-data is expected to be considered equivalent wrt. processing.

the majority of the increase in CPU time is spent in the BeginReceiveFrom call. This is surprising to me for a couple of reasons. First, this is a non-blocking call, so I would expect it to return quickly whether data is received or not. I also don't see any corresponding increase/decrease in

Are you using async-IO on non-blocking sockets? that is effectively a busy loop, which will consume all CPU if no input is available. non-blocking sockets and IO-completion-ports isn't a good combination.

That hypothesis could fit with your observations. After your code has been able to catch-up the OS-buffered data it starts burning CPU by busy-waiting when no data is available. How much CPU is spent when *no* data is received? just to confirm you are not having that problem.

BeginReceiveFrom may complete synchronously, or it may need to queue in an IO-completion-port for the next receive. The latter will probably spend more or less CPU (although I haven't measured it).

Either way the call itself should not block. They always complete

Re: pushing the envelope with sockets

I will assume you are using blocking sockets here, since async-io and non-blocking sockets is not a good combination.

If data is available synchronously the call to BeginReceive may fill the buffer using the invoking thread. You can check `IAAsyncResult.CompletedSynchronously`.

If no data is available in the OS-buffer BeginReceive will need to begin a new IO-completion port, update some data-structures... This *may* require additional processing, I don't know for sure.

An async-IO operation *may* be able to receive the data directly into it's allocated buffer. I don't know for sure.

However, I suspect that the increased cpu-usage is due to context-switching a lot more when the OS-buffer is empty when you issue your receive.

Have you tried reading synchronously with a very large buffer, just to see how that measures up to your async-IO?

Yes. Async I/O does better. It's the only way I can keep up with the send rate.

That's pretty weird, 30Mbps should be easily received on a modestly modern machine by either sync or async IO.

If you re-issue BeginReceive immediately you must be allocating a new receive-buffer for each receive. Perhaps a less CPU-intensive approach would be to only have one buffer and process that before re-issuing receive?

See above, I must re-issue I/Os quickly in order to avoid UDP packet loss.

You could try increasing the buffer-size.

And again, the CPU usage is not consistent throughout the run (in fact it changes suddenly after remaining steady) despite the fact that the I/O load is consistent. Through measurement, I've determined that the vast majority of the additional CPU time is spent inside `BeginReceiveFrom`, not in allocating buffers, etc. And if you're wondering, the amount of time it

That's exactly why I suspect that BeginReceive is more expensive when it cannot complete from the buffer.

Re: pushing the envelope with sockets

Re: pushing the envelope with sockets

You may be seeing switch-overhead. The faster you BeginReceive, the fewer packages the OS will have queued up for you and the more calls to Begin/End-Receive will be executed.

There's no queueing here beyond issuing the next I/O. That's because the next I/O isn't issued until the previous one completes (the handler issues the next I/O).

But there will still (possibly) be additional context-switching if no data is ready. If it were me I would try to prove that context-switching isn't the problem is a simple test application that does no processing of the received data at all.

I have attached a test-program which explores async vs. sync IO. It's not to be trusted 100% as it all runs locally, but I think it does support my views enough to justify further arguments for using async IO.

The output of a run of the the program on my machine is also attached.

--

Helge

```
recv_buf=1024, mbps=1000
SyncReader[10240] read: 100Mb in 5,24s = 152,74Mbps, 0 lost of 102400
AsyncReader[10240,True] read: 96,59Mb in 6,47s = 119,45Mbps, 3489 lost of 102400
AsyncReader[10240,False] read: 99,19Mb in 6,45s = 123,04Mbps, 829 lost of 102400
recv_buf=10240, mbps=1000
SyncReader[10240] read: 98,49Mb in 5,36s = 147,07Mbps, 1544 lost of 102400
AsyncReader[10240,True] read: 98,26Mb in 6,46s = 121,69Mbps, 1785 lost of 102400
AsyncReader[10240,False] read: 98,93Mb in 6,45s = 122,71Mbps, 1099 lost of 102400
recv_buf=102400, mbps=1000
SyncReader[10240] read: 99,51Mb in 5,28s = 150,84Mbps, 502 lost of 102400
AsyncReader[10240,True] read: 96,89Mb in 6,55s = 118,35Mbps, 3183 lost of 102400
AsyncReader[10240,False] read: 99,62Mb in 6,53s = 122,06Mbps, 390 lost of 102400
recv_buf=1024, mbps=50
SyncReader[10240] read: 99,08Mb in 16s = 49,53Mbps, 947 lost of 102400
AsyncReader[10240,True] read: 98,38Mb in 16s = 49,18Mbps, 1660 lost of 102400
AsyncReader[10240,False] read: 98,75Mb in 16s = 49,37Mbps, 1276 lost of 102400
recv_buf=10240, mbps=50
SyncReader[10240] read: 97,94Mb in 16s = 48,96Mbps, 2111 lost of 102400
AsyncReader[10240,True] read: 97,34Mb in 16s = 48,66Mbps, 2724 lost of 102400
AsyncReader[10240,False] read: 98,72Mb in 16s = 49,35Mbps, 1315 lost of 102400
recv_buf=102400, mbps=50
SyncReader[10240] read: 99,01Mb in 16s = 49,5Mbps, 1013 lost of 102400
AsyncReader[10240,True] read: 98,26Mb in 16s = 49,12Mbps, 1778 lost of 102400
AsyncReader[10240,False] read: 98,98Mb in 16s = 49,48Mbps, 1044 lost of 102400
recv_buf=1024, mbps=30
SyncReader[10240] read: 99,36Mb in 26,67s = 29,81Mbps, 651 lost of 102400
```

Re: pushing the envelope with sockets

Re: pushing the envelope with sockets

AsyncReader[10240,True] read: 98,96Mb in 26,67s = 29,68Mbps, 1069 lost of 102400
AsyncReader[10240,False] read: 99,03Mb in 26,68s = 29,7Mbps, 992 lost of 102400
recv_buf=10240, mbps=30
SyncReader[10240] read: 99,42Mb in 26,67s = 29,82Mbps, 593 lost of 102400
AsyncReader[10240,True] read: 99,29Mb in 26,67s = 29,79Mbps, 724 lost of 102400
AsyncReader[10240,False] read: 99,26Mb in 26,67s = 29,78Mbps, 753 lost of 102400
recv_buf=102400, mbps=30
SyncReader[10240] read: 99,57Mb in 26,67s = 29,87Mbps, 439 lost of 102400
AsyncReader[10240,True] read: 98,84Mb in 26,67s = 29,65Mbps, 1184 lost of 102400
AsyncReader[10240,False] read: 99,45Mb in 26,67s = 29,83Mbps, 568 lost of 102400
recv_buf=1024, mbps=10
SyncReader[10240] read: 99,88Mb in 80,01s = 9,99Mbps, 127 lost of 102400
AsyncReader[10240,True] read: 99,47Mb in 80,01s = 9,95Mbps, 546 lost of 102400
AsyncReader[10240,False] read: 99,58Mb in 80,01s = 9,96Mbps, 426 lost of 102400
recv_buf=10240, mbps=10
SyncReader[10240] read: 99,88Mb in 80,01s = 9,99Mbps, 119 lost of 102400
AsyncReader[10240,True] read: 99,58Mb in 80,01s = 9,96Mbps, 432 lost of 102400
AsyncReader[10240,False] read: 99,76Mb in 80,01s = 9,98Mbps, 243 lost of 102400
recv_buf=102400, mbps=10
SyncReader[10240] read: 100Mb in 80,01s = 10Mbps, 0 lost of 102400
AsyncReader[10240,True] read: 99,96Mb in 80,01s = 10Mbps, 40 lost of 102400
AsyncReader[10240,False] read: 99,93Mb in 80,01s = 9,99Mbps, 68 lost of 102400
using System;
using System.Threading;
using System.Net;
using System.IO;
using System.Net.Sockets;

```
class IOtest
{
    public abstract class Reader {
        public Socket Socket;
        public byte[] Buffer;
        public volatile int TotalRead;
        public volatile int TotalRecvs;
        public volatile bool Running;
        public DateTime LastRead;
        public virtual void Reset(Socket s, byte[] buffer)
        {
            if (Running)
                throw new InvalidOperationException("Cannot reset Running");
            Socket = s;
            Buffer = buffer;
            TotalRead = 0;
            TotalRecvs = 0;
            LastRead = DateTime.MinValue;
        }
        public virtual void Close()
        {
            Socket.Close();
            lock (this)
```

Re: pushing the envelope with sockets

```
while (Running)
Monitor.Wait(this);
}
public abstract void Start();
}
class AsyncReader: Reader
{
public bool BeginBeforeUpdate;
public AsyncReader(bool beginBeforeUpdate): base() {
this.BeginBeforeUpdate = beginBeforeUpdate;
}
void continueRead(IAsyncResult r)
{
int read;
try
{
read = Socket.EndReceive(r);
}
catch (ObjectDisposedException)
{
read = 0; // closed
}
if (read == 0)
lock ( this ) {
Running = false;
Monitor.PulseAll(this);
}
else
{
if (!BeginBeforeUpdate)
{
TotalRead += read;
LastRead = DateTime.Now;
}
Socket.BeginReceive(Buffer, 0, Buffer.Length, SocketFlags.None, continueRead, null);
if ( BeginBeforeUpdate )
lock (this)
{
TotalRead += read;
DateTime now = DateTime.Now;
if ( now > LastRead )
LastRead = DateTime.Now;
}
}
}
public override void Start() {
lock (this)
{
if ( Running )
throw new InvalidOperationException("Cannot Start() while Running");
Socket.BeginReceive(Buffer, 0, Buffer.Length, SocketFlags.None, continueRead, null);
}
```

Re: pushing the envelope with sockets

```
Running = true;
Monitor.PulseAll(this);
}
}
public override string ToString()
{
return string.Format("{0} [{1}, {2}]",
GetType().Name, Buffer.Length, BeginBeforeUpdate);
}
}
class SyncReader: Reader
{
public SyncReader(): base() {}
void run()
{
lock (this)
{
Running = true;
Monitor.PulseAll(this);
}
try
{
int read;
do
{
read = Socket.Receive(Buffer, 0, Buffer.Length, SocketFlags.None);
// no lock, we are the only updaters
TotalRead += read;
LastRead = DateTime.Now;
} while ( read > 0 );
} catch ( SocketException ) {
// closed
}
finally
{
lock (this)
{
Running = false;
Monitor.PulseAll(this);
}
}
}
public override void Start()
{
ThreadStart ts = new ThreadStart(run);
Thread t = new Thread(ts);
t.Start();
}
public override string ToString()
{ return string.Format("{0} [{1}]", GetType().Name, Buffer.Length); }
}
```

Re: pushing the envelope with sockets

```
static void SendLimited(Socket s, byte[] Message, int count, EndPoint to, double packages_pr_second)
{
    s.SendBufferSize = 1;
    DateTime start = DateTime.Now;
    for (int i = 0; i < count; ++i)
    {
        s.SendTo(Message, to);
        // To limit rate, wait untill sending the next message.
        // to ensure a steady-flow Thread.Sleep(0) is used
        // which prevents the thread for sleeping for 10ms,
        // which would yield bust...quiet...burst...quiet behaviour
        while ( true ) {
            DateTime now = DateTime.Now;
            double used_seconds = (now - start).TotalSeconds;
            double expected_seconds = i/packages_pr_second;
            if (expected_seconds < used_seconds)
                break;
        }
    }
}

static Socket MakeUDPSocket(EndPoint ep, int recv_buf_size)
{
    Socket s = new Socket(ep.AddressFamily, SocketType.Dgram, ProtocolType.Udp);
    s.ExclusiveAddressUse = true;
    s.ReceiveBufferSize = recv_buf_size;
    s.Bind(ep);
    return s;
}

public static void testReader(
    Socket sender, Reader reader, int msg_size, int count, double packages_pr_second, TextWriter trace)
{
    byte[] msg = new byte[msg_size];
    reader.Start();
    DateTime start = DateTime.Now;
    SendLimited(sender, msg, count, reader.Socket.LocalEndPoint, packages_pr_second);
    TimeSpan awhile = TimeSpan.FromSeconds(2.0);
    Thread.Sleep(awhile); // allow time for processing late-received packages
    reader.Close();
    TimeSpan spent = reader.LastRead - start;
    trace.WriteLine(
        "{0} read: {1:###.##}Mb in {2:###.##}s = {3:###.##}Mbps, {4} lost of {5}",
        reader,
        reader.TotalRead / (1024.0*1024.0),
        spent.TotalSeconds,
        (reader.TotalRead*8) / (spent.TotalSeconds * 1024 * 1024),
        count - ((double)reader.TotalRead) / msg_size,
        count);
}

public static void Main()
{
    TextWriter trace = Console.Out;
```

Re: pushing the envelope with sockets

```
int msg_size = 1024;
int count = 100*1024;
IPAddress host = Dns.GetHostEntry("localhost").AddressList[0];
int port = 16000;
Socket sender_socket = MakeUDPSocket(new IPEndPoint(host, port + 0), 1000);

Reader[] readers = {
    new SyncReader(),
    new AsyncReader(true),
    new AsyncReader(false)
};
double[] mbps_tries = { 1000, 50, 30, 10 };
int[] rcv_buf_sizes = { 1*msg_size, 10*msg_size, 100*msg_size };
foreach (double mbps in mbps_tries)
    foreach ( int rbs in rcv_buf_sizes )
    {
        trace.WriteLine("rcv_buf={0}, mbps={1}", rbs, mbps);
        byte[] read_buf = new byte[msg_size * 10];
        foreach (Reader reader in readers)
        {
            reader.Reset(MakeUDPSocket(new IPEndPoint(host, port + 1), rbs), read_buf);
            testReader(sender_socket, reader, msg_size, count, (mbps*1024*1024)/((double)msg_size*8), trace);
        }
    }
if (!System.Diagnostics.Debugger.IsAttached)
{
    Console.WriteLine("Waiting for ENTER");
    Console.ReadKey();
}
}
```