Thanks a bunch for the reply Tony

"Also, have you tried disposing of the ticker instance only when this error happens?"

Couldn't get this part

If you are suggesting using/calling dispose on **Ticker** instance Then it does not have *Dispose* method to call.

There is a Dispose method on **_ws.**

WebSocket.cs

```
/// <summary>
/// A wrapper for .Net's ClientWebSocket with callbacks
/// </summary>
internal class WebSocket
{
  // Instance of built in ClientWebSocket
     ClientWebSocket _ws;
```

Ticker.cs

```
// Instance of WebSocket class that wraps .Net version
   private WebSocket _ws;
```

Which is private and not exposed through Ticker instance.

WebSocket.cs

```
// Instance of built in ClientWebSocket
   ClientWebSocket _ws;
```

If you are suggesting to close Ticker when error happens no I don't do that I let Kite handle it to Reconnect or otherwise

"You can add it to the Ticker class and use it as the last resort.
But if it is a normal connection close make sure you **close** it without **Abort** because this will inform our servers that it is intentional.
Otherwise, it might count that your connection is still alive and next time you might get **rate limited** *for opening more connections*"

That is **scary** now!!.

Ticker.cs

```
/// <summary>
/// Reconnect WebSocket connection in case of failures
/// </summary>
private void Reconnect()
{
  if (IsConnected)
     _ws.Close(true);

  if (_retryCount > _retries)
  {
     _ws.Close(true);
      DisableReconnect();
      OnNoReconnect?.Invoke();
  }
  else
  {
      OnReconnect?.Invoke();
      _retryCount += 1;
     _ws.Close(true);
      Connect();
      _timerTick = (int)Math.Min(Math.Pow(2, _retryCount) * _interval, 60);
      if (_debug) Console.WriteLine("New interval " + _timerTick);
          _timer.Start();
  }
}
```

You are yourself calling _ws.Close(true)

What this means is connection if connected would be **aborted** anyway on reconnect. And your *server* has no way of knowing if it was intended or not and Connect() any way creates a brand-new ClientWebSocket. So is it a graceful close or abrupt Abort does not matter at this point.

WebSocket.cs

```
/// <summary>
/// Connect to WebSocket
/// </summary>
public void Connect(Dictionary<string, string> headers = null)
{
    try
    {
        // Initialize ClientWebSocket instance and connect with Url
        _ws = new ClientWebSocket();
```

Ticker.cs

```
/// <summary>
/// Start a WebSocket connection
/// </summary>
public void Connect()
{
    _timerTick = _interval;
    _timer.Start();
    if (!IsConnected)
    {
        _ws.Connect(new Dictionary<string, string>() { ["X-Kite-Version"] = "3" });
    }
}
```

I think it is not the right approach. Because it is only good until it is good and if something goes wrong then it is pure luck if we get **rate limited** or not!. I think until it is figured out. I might use the approach below.

WebSocket.cs

```
/// <summary>
/// Tries to close WebSocket connection. Default timeout is
/// 1 Minute or 60,000 milliseconds.
/// </summary>
public bool TryClose( double wait_period = 60,000 )
{

    if(_ws.State == WebSocketState.Open)
    {
        try
        {

          _ws.CloseAsync(WebSocketCloseStatus.NormalClosure, "", CancellationToken.None).Wait( TimeSpan.FromMilliseconds( wait_period ) );

        }
        catch (Exception e)
        {
            OnError?.Invoke("Error while trying to close connection. Message: " + e.Message);
        }
    }

    return IsConnected() ;
}
```

And then
`if ( TryClose( ) == false ) Close ( Abort: true ) ;`

Thanks Regards.