TCP 連線異常處理研究報告

1. 背景與目標

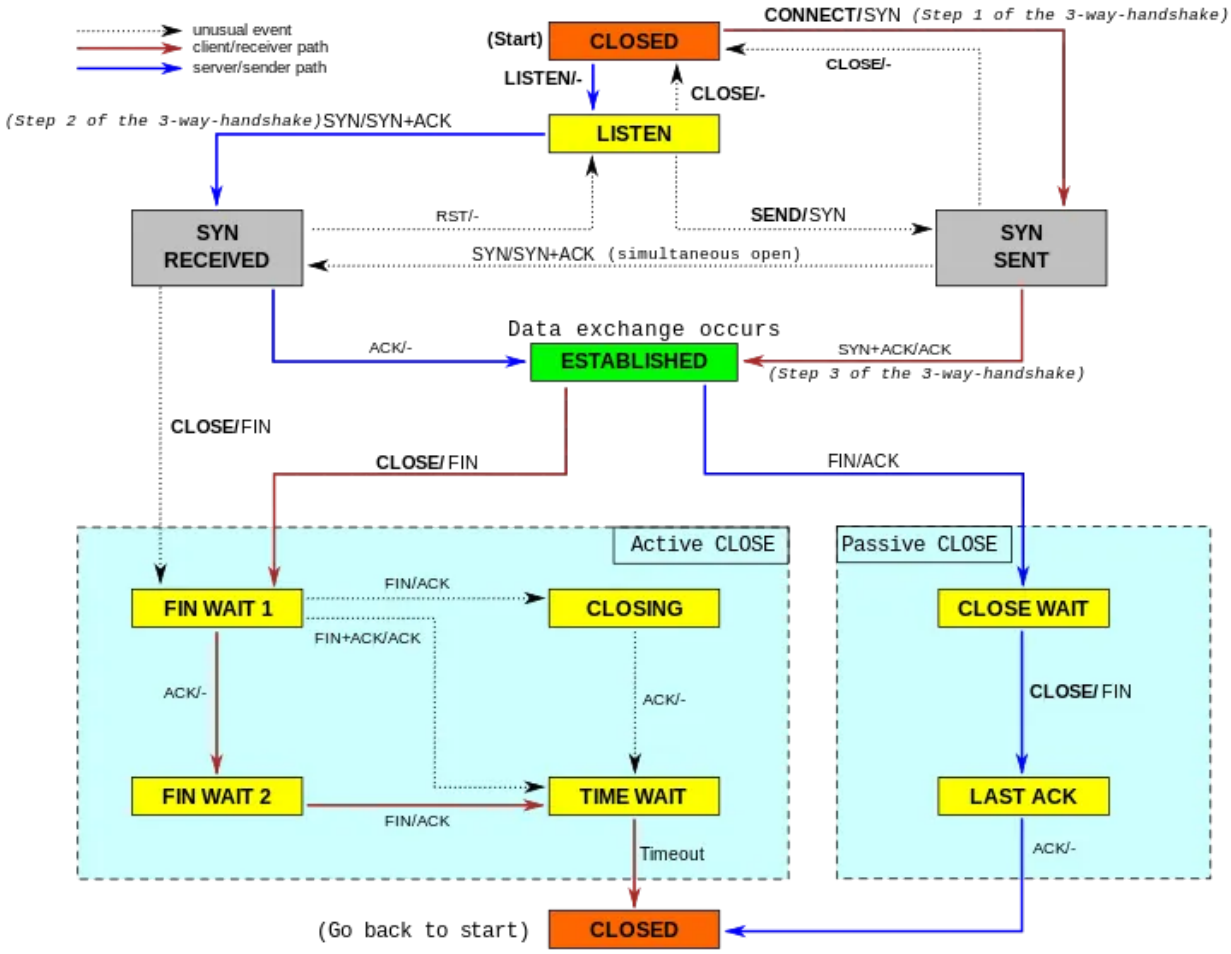
在實際部署系統（如 IoT、工業自動化、客戶端應用）中，TCP 連線異常經常導致資料遺失或設備無法控制。此報告旨在分析常見異常情況，並提出有效的解決方案以提高系統穩定性與可靠性。

---

2. 問題描述

為什麼 TCP 無法即時知道對方斷線？

- TCP 是 **無錯誤即沉默**（silent failure）
- 如果對方拔線、重啟，不會主動送 FIN 或 RST → 你本地的 TCP 還以為連著(參考TCP狀態圖)
- 所以需要 **KeepAlive** + **探測機制** 來「**主動發現異常**」



---

3. 現況系統設計觀察

- 無明確斷線偵測機制（被動等待資料）
- 連線異常常需等到傳送失敗才被發現
- 缺乏自動重連與狀態監控機制

4. 解決方案比較

方法	原理	優點	缺點
TCP KeepAlive	OS 層定時探測是否還有連線	簡單、OS 支援	可調參數有限、不即時
自訂 Heartbeat	應用層定時發 Ping 封包	可控性高、偵測快	實作複雜、額外負擔
`System.Net.Sockets.Socket` 類別中的方法	定時輪詢可讀狀態	輕量、不需心跳	仍需輪詢、有偵測延遲

使用 Socket 檢查 TCP 連線狀態的 4 種方法

整理四種常見的 TCP Socket 健康檢查方法，適用於 .NET 中的 `System.Net.Sockets.Socket` 類別。

- `Socket.Connected`
  - 檢查目的： - 確認此 socket 是否仍然「標記為連線中」
  - 缺點： - 不準確！ 只在連線剛建立時準確，無法偵測對方是否中斷或關閉連線，不會即時反映。
- `Poll(SelectMode.SelectRead) + Available == 0`
  - 檢查目的：
    - 偵測對方是否已「優雅關閉」連線（發送了 FIN）
  - 缺點：
    - Available == 0 在某些時候（如網路 delay）可能誤判
    - 偶爾會錯誤觸發 false positive
    - 搭配其他檢查（如 Peek）更穩定
- `Poll(SelectMode.SelectError)`
  - 檢查目的：
    - 偵測 socket 是否進入錯誤狀態，例如網路中斷或被重置（RST）
  - 缺點：
    - 偵測極端狀況有效，但無法判斷正常的關閉流程
    - 應作為輔助機制使用
- `Receive(..., SocketFlags.Peek)`
  - 檢查目的：
    - 透過偷看封包，不影響 buffer，準確判斷 socket 是否已被關閉
  - 優點：
    - 極為準確，對方 Close() 會立即反映
    - 不會消耗資料（使用 Peek）
  - 缺點：
    - 成本略高（會觸發一次 I/O）
    - 頻繁使用時略有效能影響

方法比較表

檢查方式	準確性	成本	推薦用途

檢查方式	準確性	成本	推薦用途
`socket.Connected`	低	無	不推薦
`Poll(Read) + Available == 0`	高	快	偵測優雅關閉
`Poll(Error)`	中	快	極端錯誤時可補充使用
`Receive(Peek)`	高	中	精準判斷是否已斷線

實戰建議

- 若要快速檢查：用 Poll + Available
- 若要精準偵測：加上 Receive(..., Peek)
- 若用於心跳或長連線監控：建議兩者搭配使用

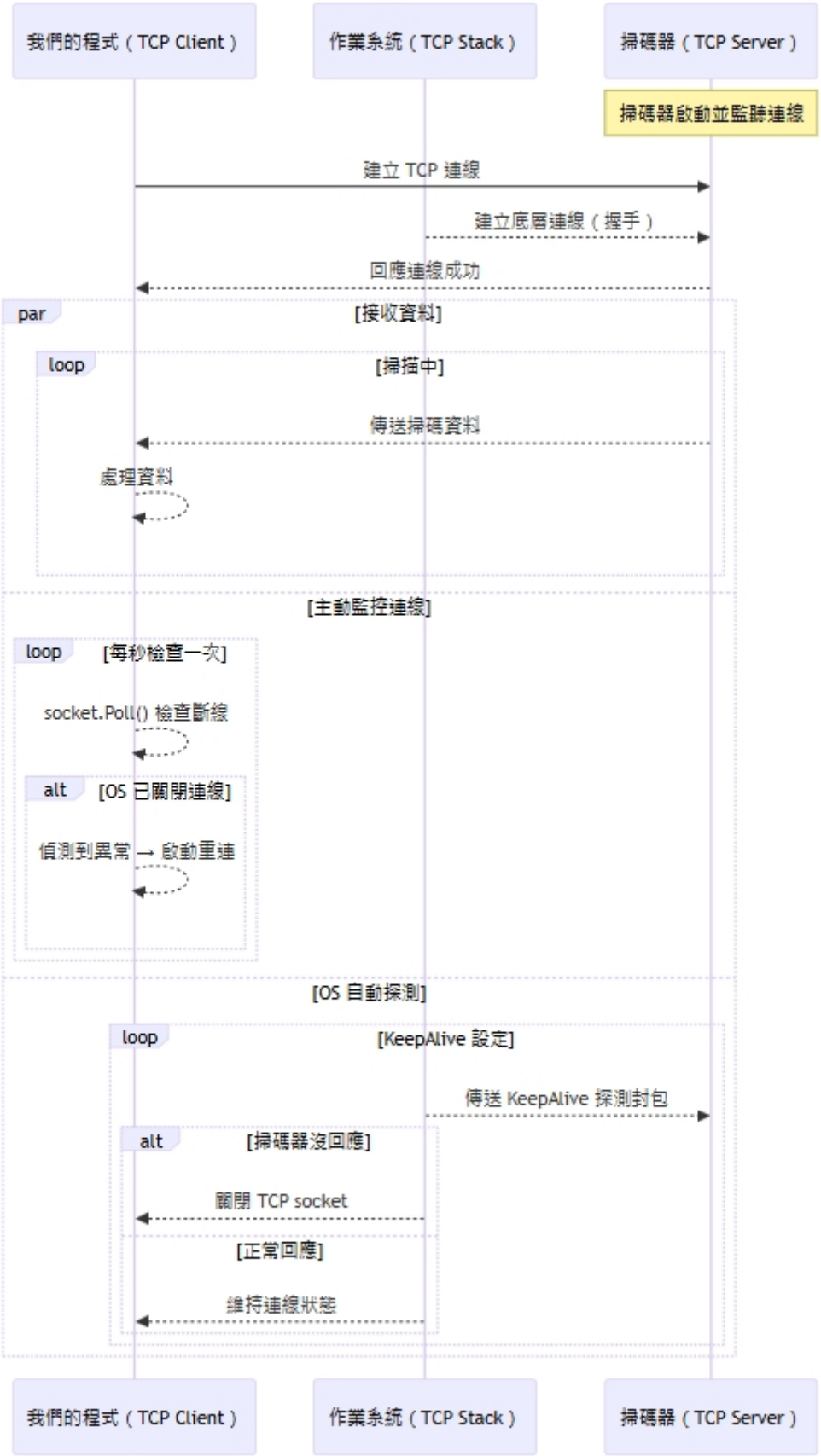
---

5. 實驗方法

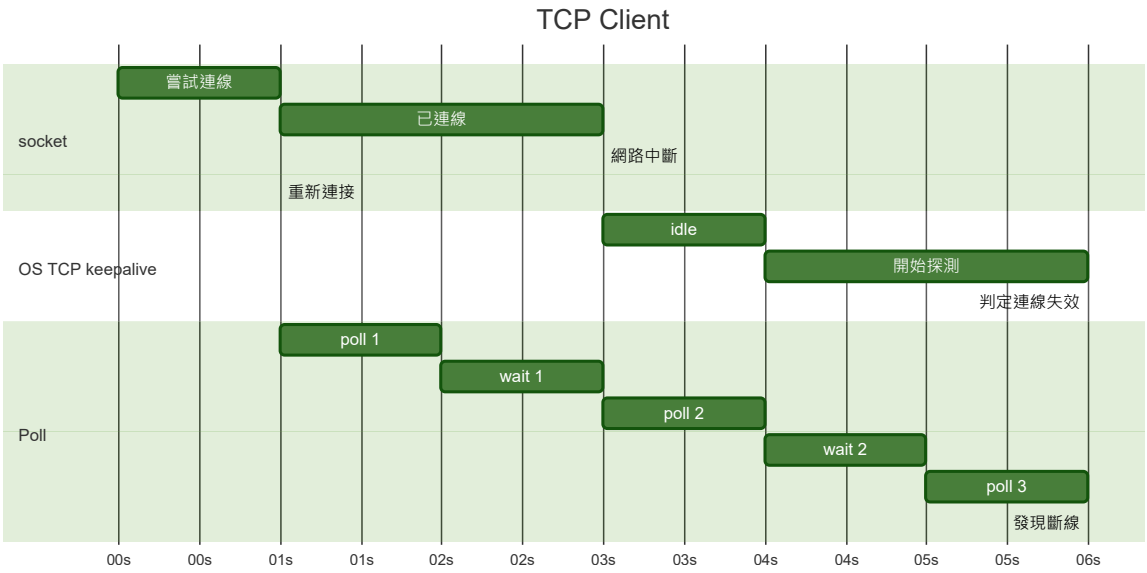
掃碼器情境應用說明

使用 `TCP keepalive` 、 `poll` 、 `heartbeat` 是最簡單且直觀的方式，但以掃碼器的案例來說，與該設備的通訊流程沒有辦法透過實作應用層 `heartbeat` 的方式去處理連線異常問題的，只能使用 `TCP keepalive` 、 `poll` 來處理。實際流程如下所示：

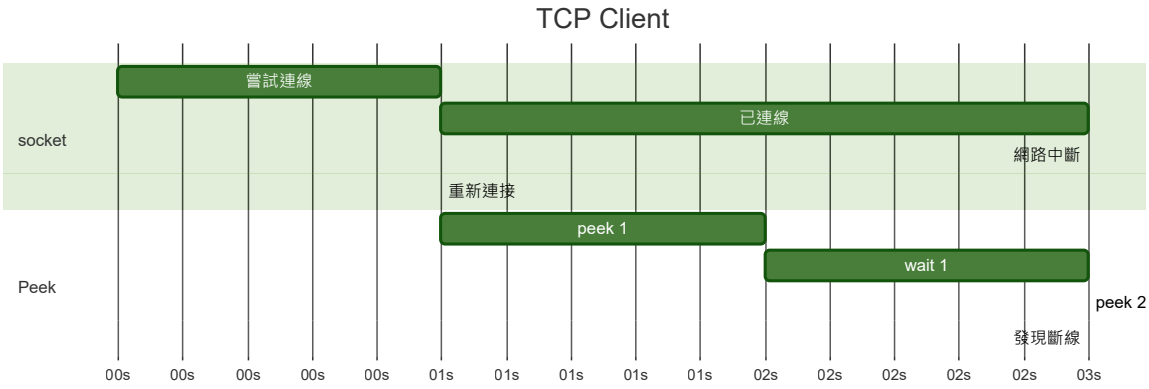
> 下圖為實作一個讀取掃碼器的程式的循序圖，其內容表達了，我們的程式(TCP Client)如何利用連線檢查來改善和掃碼器溝通時遇到的TCP連線問題，以及OS在其中扮演的角色。



下圖是一個有連線偵錯功能的 `Tcp client` 實際運作時各個角色的甘特圖，分別是使用 `socket.Poll()` 、 `socket.Peek()`來進行診斷的狀況



連線診斷甘特圖 Poll



連線診斷甘特圖 Peek

實驗過程

---

6. 其他建議

監測工具與數據分析

- 使用 Wireshark 分析封包異常情況
- 使用 netstat 或 socket 統計觀察狀態
- 導入 NLog / Serilog 寫入結構化日誌方便後續分析

---

7. 結論與建議

單靠 OS 的 TCP KeepAlive 不足以即時偵測異常，建議搭配 應用層輪詢 / heartbeat，以及自動重連來達到完整的TCP/IP異常連線處理機制。

---

8. 附錄

Sample Code - 設定 KeepAlive (C#)

```
        byte[] option = new byte[12];
        BitConverter.GetBytes(1).CopyTo(option, 0);            // 開啟
        BitConverter.GetBytes(timeMs).CopyTo(option, 4);       // 空閒多久後開始探測
        BitConverter.GetBytes(intervalMs).CopyTo(option, 8);   // 每次探測間隔
        socket.IOControl(IOControlCode.KeepAliveValues, option, null);
```

方法 1 Sample Code：socket.Connected

```
if (!socket.Connected)
{
        // 處理斷線
}
```

方法 2 Sample Code：socketPoll(SelectMode.SelectRead) + Available == 0

```
if (socket.Poll(chkConnIntervalMs, SelectMode.SelectRead) && socket.Available ==
0)
{
        // 對方可能已優雅關閉連線（FIN）
}
```

方法 3 Sample Code：socketPoll(SelectMode.SelectError)

```
if (socket.Poll(chkConnIntervalMs, SelectMode.SelectError))
{
        // socket 發生錯誤，如網路異常或強制中斷
}
```

方法 4 Sample Code：socketReceive(..., SocketFlags.Peek)

```
try
{
        byte[] buffer = new byte[1];
        int result = socket.Receive(buffer, 0, 1, SocketFlags.Peek);

        if (result == 0)
        {
                // 對方已關閉連線（graceful shutdown）
        }
```

```
}
catch (SocketException ex)
{
        if (ex.SocketErrorCode != SocketError.WouldBlock)
        {
                // 非阻塞模式下的真正錯誤，可視為斷線
        }
}
catch
{
        // 其他例外，視為已斷線
}
```

參考資料

- https://learn.microsoft.com/zh-tw/dotnet/api/system.net.sockets.socket.iocontrol?view=net-9.0
- https://learn.microsoft.com/zh-tw/dotnet/api/system.net.sockets.socket.poll?view=net-9.0
- https://learn.microsoft.com/zh-tw/dotnet/api/system.net.sockets.socket.receive?view=net-8.0
- https://zh.wikipedia.org/zh-tw/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE
- https://tech.youzan.com/you-zan-tcpwang-luo-bian-cheng-zui-jia-shi-jian/
- https://zhuanlan.zhihu.com/p/496790111
- https://blog.darkthread.net/blog/detected-tcpclient-connection-status/