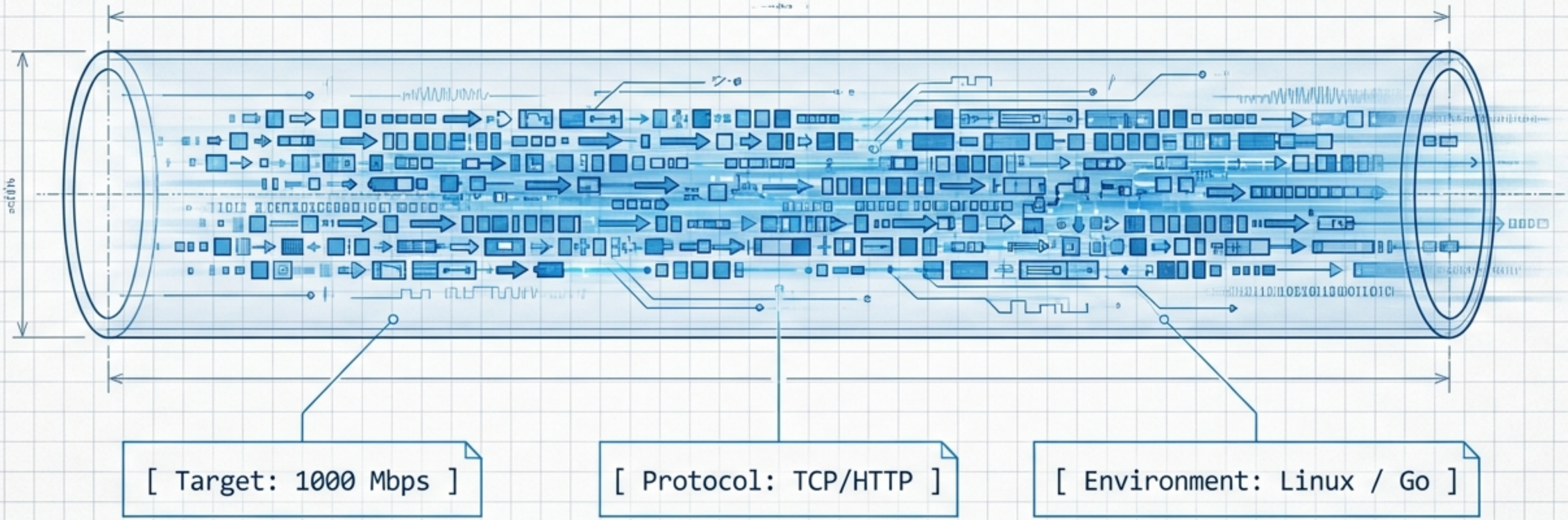
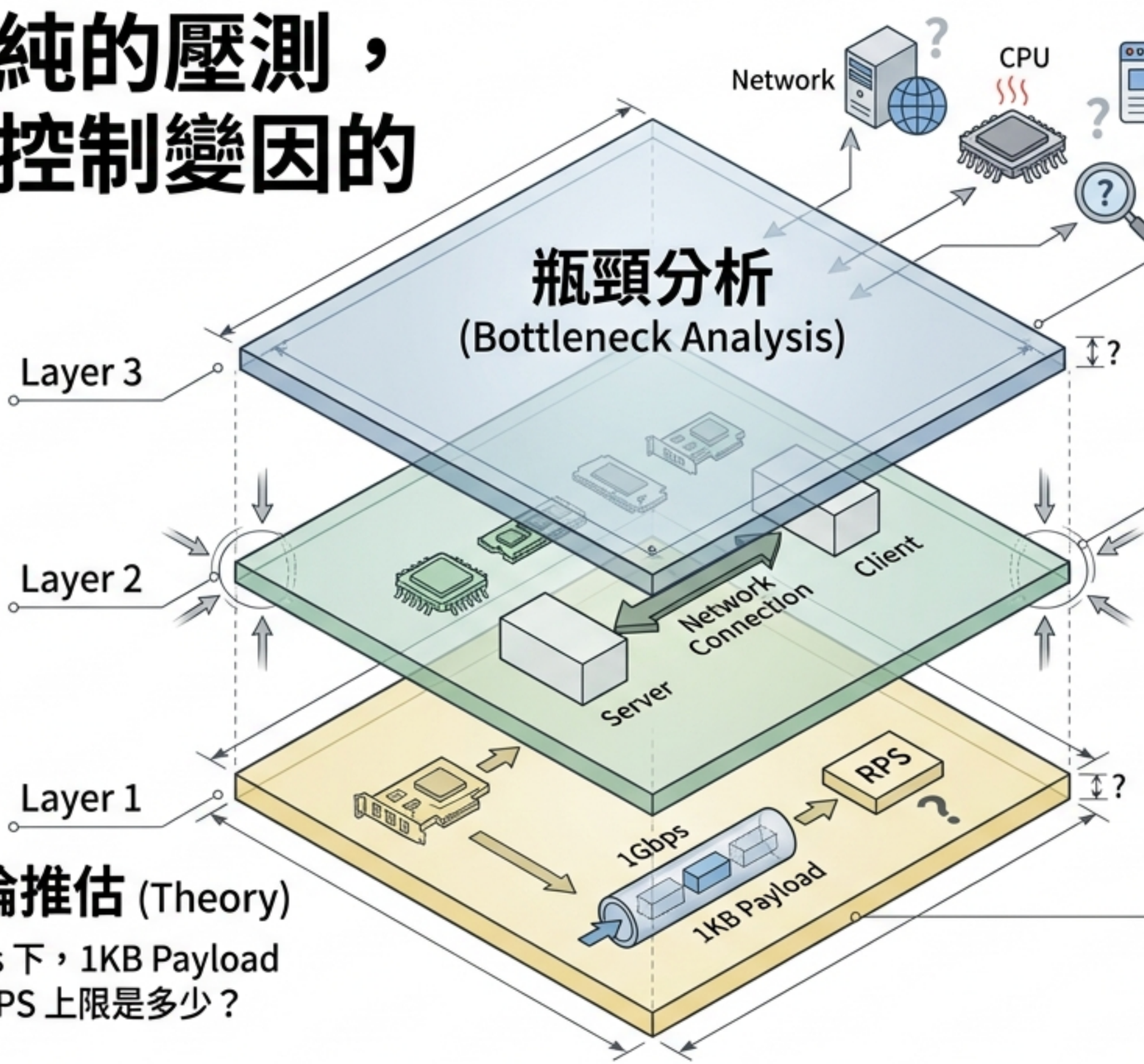
1Gbps 網路效能極限探測

從基礎網路到 C10K 的 HTTP 效能工程與瓶頸診斷實錄



這不是單純的壓測，而是一場控制變因的工程實作



瓶頸分析
是網路、CPU、OS
還是測試工具先崩潰？

工程實作
(Implementation)
極簡 Server / Client
能否逼出硬體與網路極限？

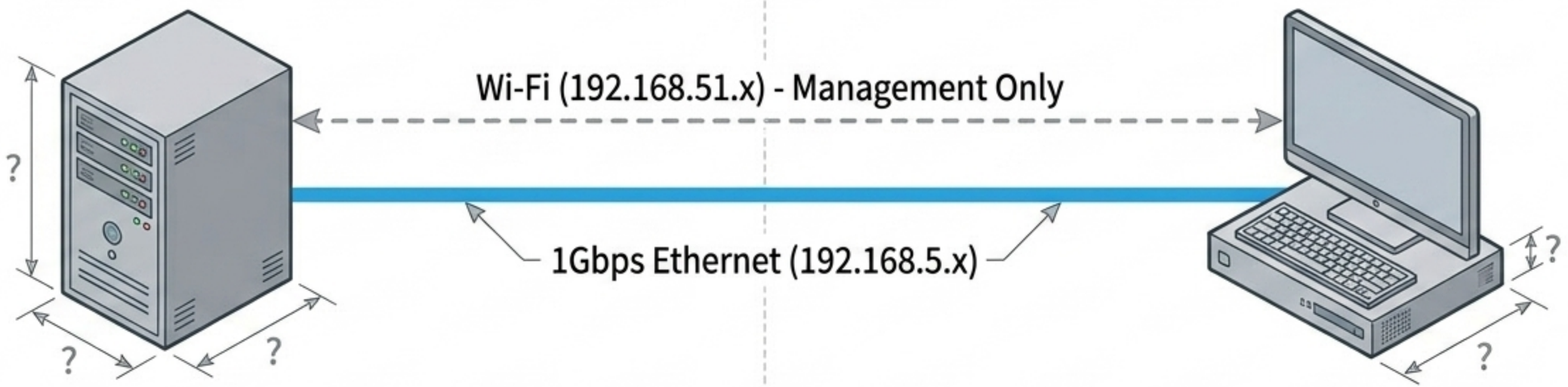
理論推估 (Theory)
1Gbps 下，1KB Payload
理論RPS 上限是多少？

核心提問：「用兩台 Mini PC 能否打滿 1Gbps 頻寬？如果不行，隱形牆在哪裡？」

隔離變因：硬體拓撲與專用網路對接

Server (gtinfra02)

Client (gtinfra03)

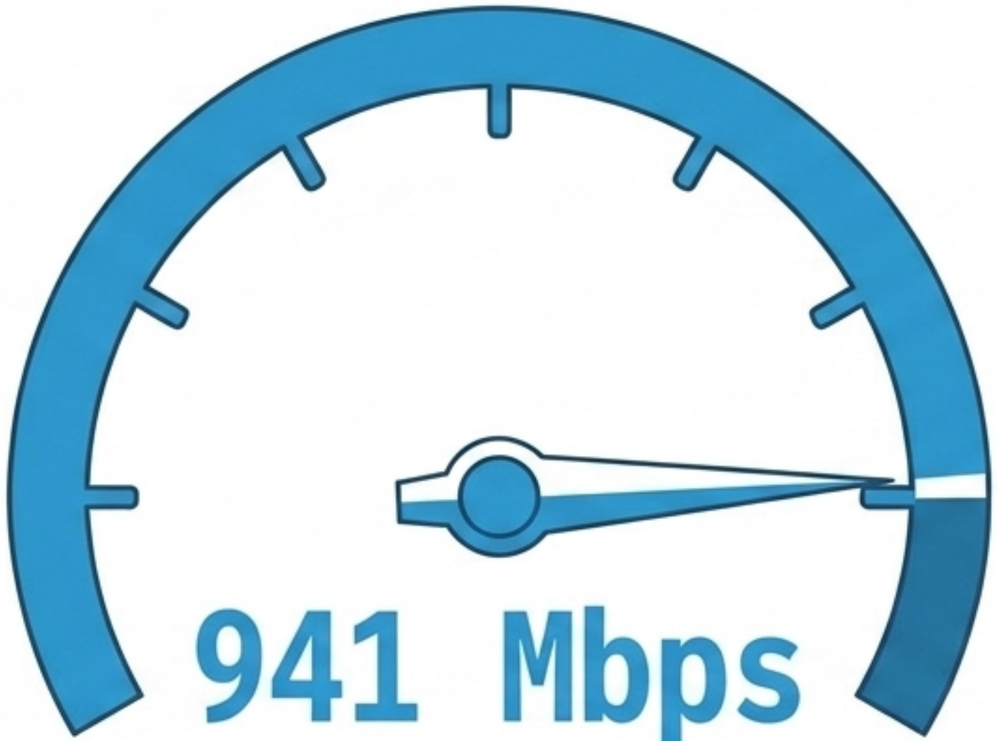


CPU: AMD Ryzen 7 5825U (16 Cores)
RAM: 62GB
OS: Ubuntu 24.04

CPU: Intel Core i7-1360P (16 Cores)
RAM: 62GB
OS: Ubuntu 24.04

壓測流量強制綁定實體 Ethernet，管理指令走 Wi-Fi，徹底排除 SSH 操作造成的雜訊干擾。

Phase 0：排除軟體雜訊，確立物理極限 (TCP Baseline)

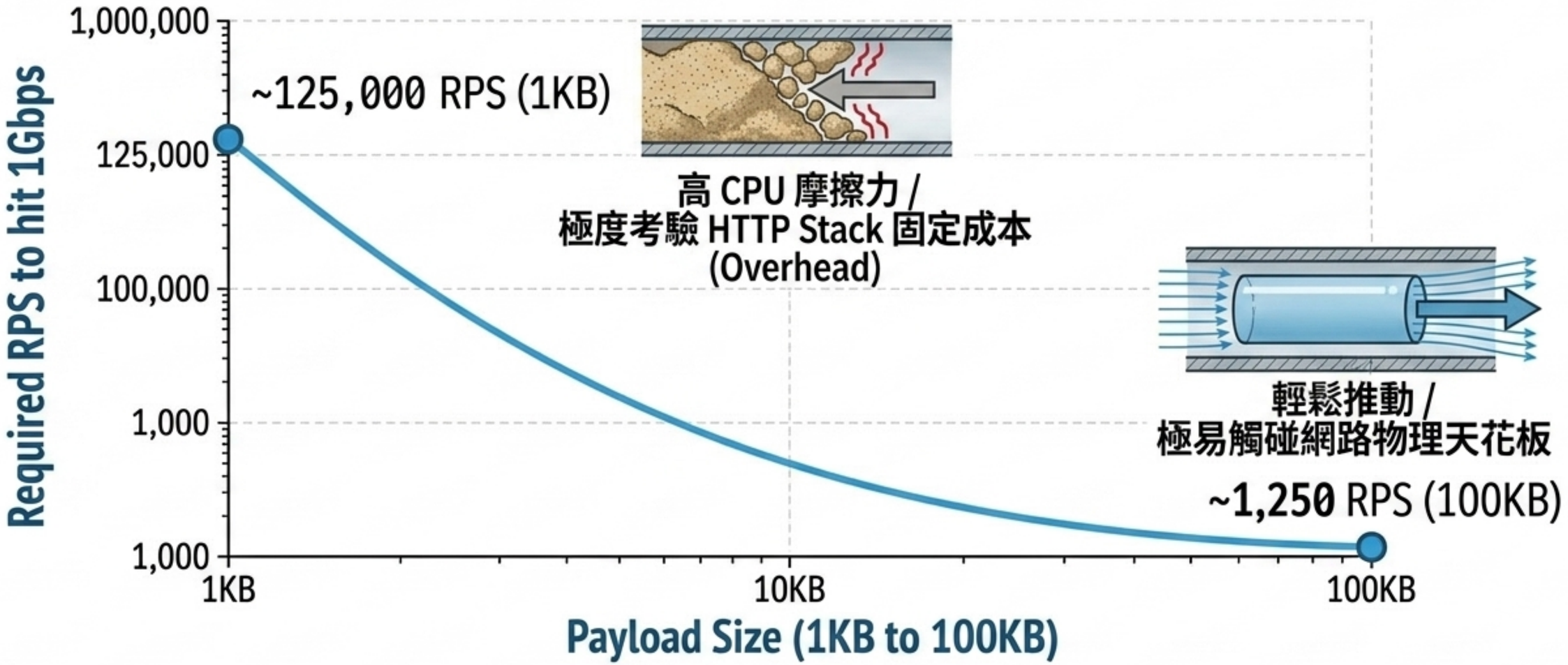


iperf3 Bare-metal TCP Throughput

- TCP 重傳率 (Retransmits): 0
- CPU 負載 (System-wide): < 20%
- 雙向傳輸 (Reverse Mode): 941 Mbps

網路通道極度健康。後續 HTTP 若沒打滿 1Gbps，
優先懷疑軟體堆疊為軟體堆疊 (Software Stack) 或 OS 限制，而非實體網路。

假說：塞滿 1Gbps 管道需要多大的「推力」？



1 Gbps ≈ 125 MB/s = (RPS × Payload Size) + Protocol Overhead

Phase 1：極簡 HTTP Server 成功逼近可用極限

1KB Payload

RPS: **101,431**

Throughput:

110.47 MB/s

Server CPU: **382%**

高耗能，接近 CPU
與框架處理上限

10KB Payload

RPS: **11,281**

Throughput:

111.45 MB/s

Server CPU: **73%**

中等耗能，平穩打滿網路

100KB Payload

RPS: **1,147**

Throughput:

112.20 MB/s

Server CPU: **13%**

極低耗能，完美滿載 1Gbps

伺服器成功輸出 ~112 MB/s 應用層吞吐量，對應 941 Mbps TCP 基線。第一階段工程實作過關！

Phase 2 併發矩陣：吞吐量的「物理天花板」

Heatmap Matrix (Throughput (MB/s)

	c16	c64	c128	c256	c512	c1024
1KB	47.5	103.8	103.8	103.8	103.8	103.6
10KB	115.5	115.4	115.4	115.4	115.3	115.2
100KB	117.5	117.4	117.4	117.2	117.0	116.5

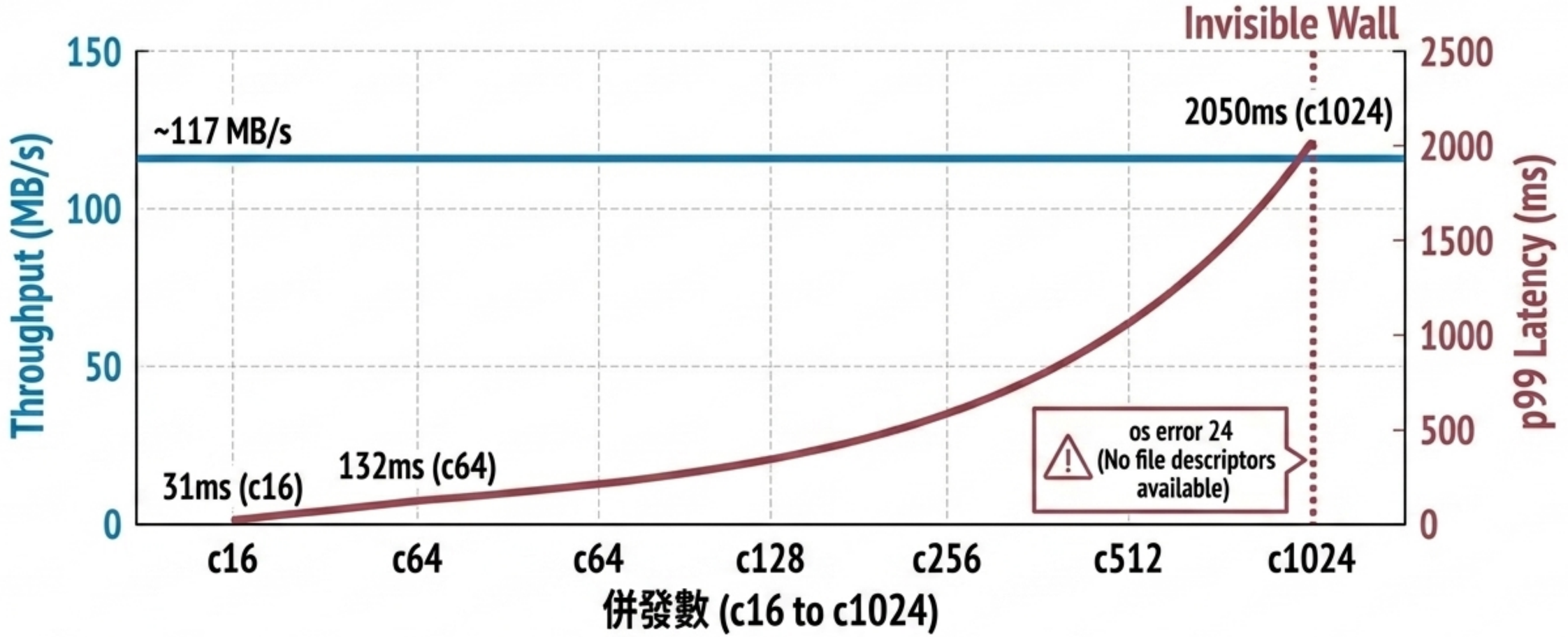
最佳穩定點 (Best Stable Cases)

1KB 最佳點為 c64；10KB/100KB 最佳點為 c16。

邊際效應遞減法則

一旦抵達網路傳輸極限 (Network-Bound)，繼續增加連線線數完全無法增加吞吐量。

盲目增加併發的代價：佇列與延遲崩潰 (Latency Cliff)



吞吐量不變，多出的 Request 全部塞在佇列 (Queueing)。
最終壓測工具 (Client) 本身先因 OS 限制發出哀嚎。

效能工程心法：如何科學定位系統瓶頸？

Network Bound (網路瓶頸)

- Symptoms: CPU 未滿，網卡 TX/RX 接近物理極限。吞吐量平坦。
- Action: 最佳結果。若需突破，只能升級實體網路或壓縮 Payload。

CPU Bound (運算瓶頸)

- Symptoms: 網卡未滿，但 Server 總 CPU 或單核 CPU 達 100%。
- Action: 需優化 HTTP 框架、業務邏輯或開啟多執行緒。

Queueing Bound (佇列瓶頸)

- Symptoms: 吞吐量持平，但 p99 延遲異常飆升。
- Action: 併發設定過高，應調降 Client worker 數以符合系統最佳處理點。

Client Limit (發壓端瓶頸)

- Symptoms: 伺服器未滿載，但 Client 報出 OS limit 或 Socket error。
- Action: 測試失真。需調優發壓端 ulimit -n 或擴展 Ephemeral Ports。

16:9 DIAGNOSTIC MATRIX

Phase 3：拆解 C10K 迷思

一萬個連線不等於一萬 RPS。我們將 C10K 拆分為三個現實情境：

Idle (閒置)

測資源容量

建立 10,000 條 Keep-Alive 連線並保持。

檢視 Memory, File Descriptors 消耗。

Low-Rate (低頻)

測真實長連線

每條連線每 10 秒發送一次請求 (總計 1k QPS)。

模擬 IoT, WebSockets 等真實場景。

Active (活躍)

測極限排隊

10,000 條連線全力猛打。

測試極限佇列延遲與系統瓶頸。

閒置與低頻 C10K：系統餘裕超乎想像

10,000 Idle Connections

Server RAM:
~209 MB

Server CPU:
0.40%

20,000 Idle Connections
(Beyond C10K)

Server RAM:
~407 MB (Linear Scaling)

10,000 Low-Rate (1k QPS)

Success Rate:
100%

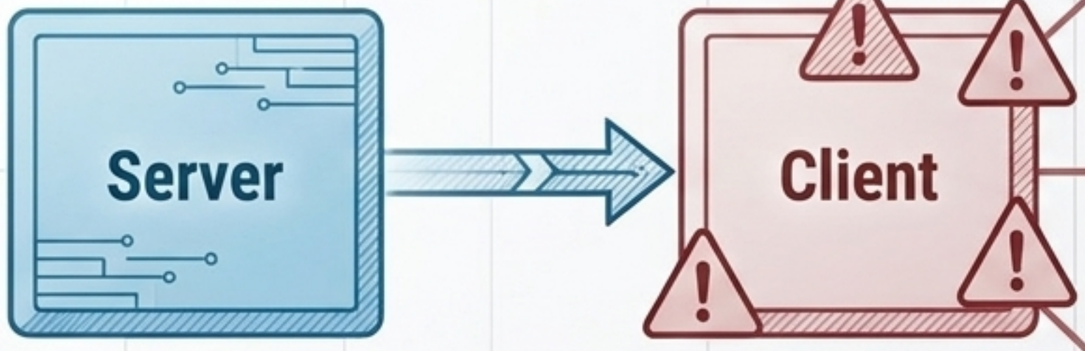
Server CPU:
13.09%

p99 Latency:
0.99 ms

現代硬體與 OS 維持上萬條 TCP Keep-Alive 狀態的成本極低。真正的挑戰不在於「維持連線」。

活躍 C10K：當發壓端 (Client) 成為最大瓶頸

10k 併發全力猛打時，吞吐量卡在 1Gbps 物理定律，但系統底層發生了什麼事？



File Descriptor (FD) 枯竭

預設 ulimit -n (1024) 首先被耗盡。
必須提權解鎖至 200,000 才能乾淨測試。

Ephemeral Ports (臨時埠) 限制

單一 Client IP 對單一 Server IP:Port，可用埠受限於 ~28,000。超越 20k 後極易撞牆。

TIME_WAIT 殘留

高頻壓測後，大量 Socket 滯留 TIME_WAIT。
若無 Cooldown 機制，會導致後續連線失敗。

在極限壓測下，壓測機 (Client) 往往比伺服器更早崩潰。

從實驗淬鍊的「效能驗證框架」

Step 1: 建立物理基線 (Establish Baselines)

別急著測 API。先用 iperf3 測量裸網路，確保底層沒有偷吃步。確立絕對的「物理天花板」。

Step 2: 隔離業務邏輯 (Isolate Variables)

使用極簡 Payload 排除 JSON/DB 等運算耗時，分離出純 HTTP 框架與作業系統的極限。

Step 3: 觀測全域指標 (Holistic Metrics)

不要只看 RPS。必須將 Throughput、p99 Latency、TX/RX 與 Server/Client 雙端端的 CPU/FD Limits 放上同一個天平檢視。

任務達成：1Gbps 網路頻寬極限探索

- ☑ 成功透過工程實作將兩台 Mini PC 的 1Gbps 網路頻寬榨乾 (~117 MB/s)。
- ☑ 驗證了 Payload 大小、RPS 與 CPU 負載的動態反比關係。
- ☑ 證明了 C10K 的難點已不在伺服器運算力，而在網路頻寬上限、佇列排隊管理，與作業系統資源調度。

> 「真正的效能優化，不是盲目追求數字，
> 而是精準看透每一個位元在系統中的旅行軌跡。」