基於終端模擬器圖形協議的遠端桌面串流系統

A Remote Desktop Streaming System Based on Terminal Emulator Graphics Protocols

專題簡介

- 隨著 Wayland 顯示伺服器協議逐漸取代 X11 成為 Linux 桌面環境的主流標準，傳統的遠端桌面方案面臨相容性挑戰。X11 提供原生的 display forwarding 機制（ssh -X），允許使用者透過 SSH 直接轉發圖形介面；然而 Wayland 的設計哲學強調安全隔離，不提供等效的原生轉發機制，使得遠端操作 Wayland 桌面必須依賴 VNC、RDP 等需要在本地端安裝專用客戶端的工具。
- 本專題提出並實作一個以 C 語言撰寫的 Wayland 遠端桌面串流系統，利用現代終端模擬器所支援的 Kitty Graphics Protocol（KGP）作為畫面傳輸通道，搭配 SSH 連線，使使用者在本地端無需安裝任何額外程式，僅需一個支援 KGP 的終端模擬器（如 Kitty、Ghostty、WezTerm）即可操作遠端桌面。

背景與動機

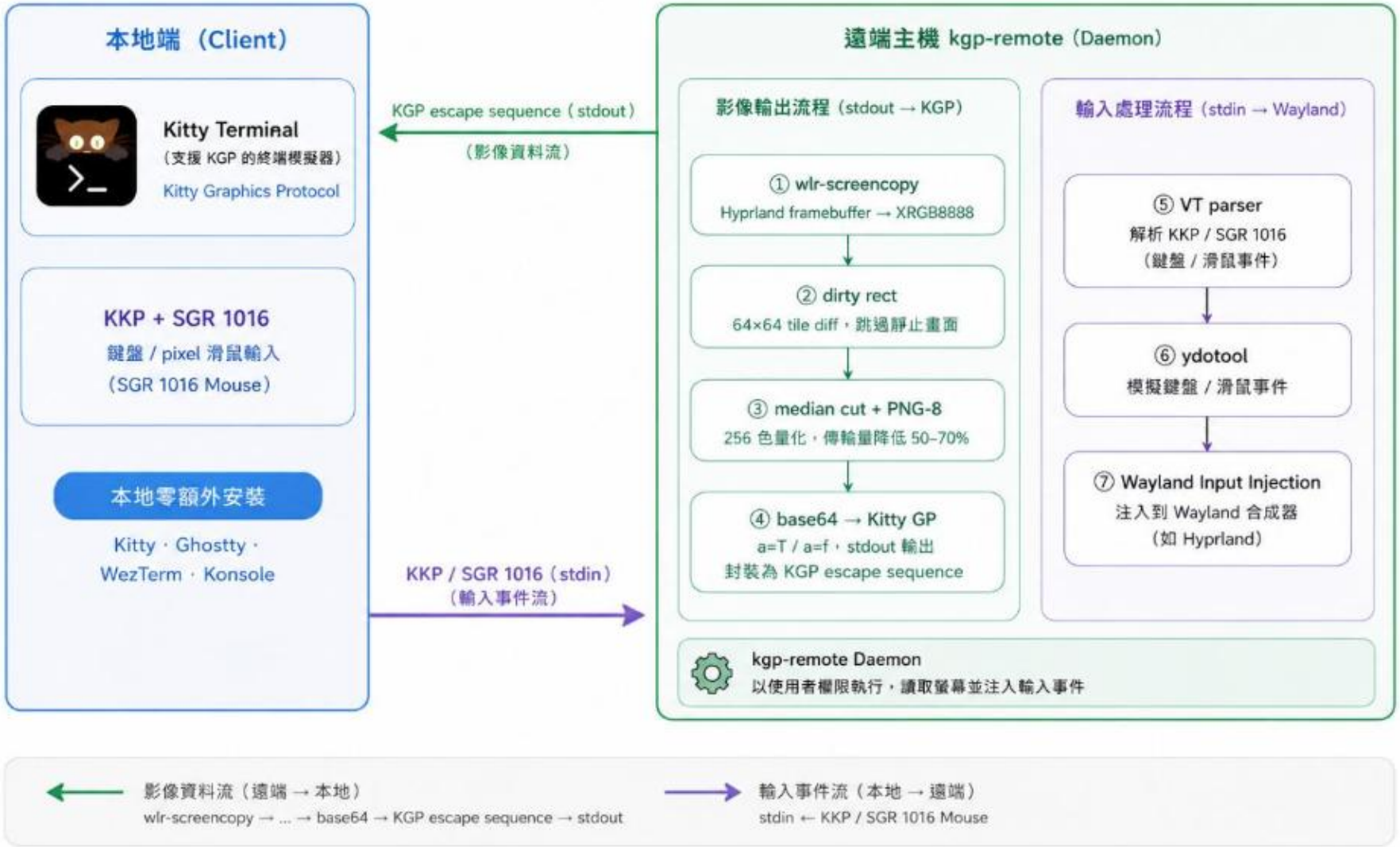
- 現有遠端桌面方案（VNC、RDP）皆需要在本地端安裝專用客戶端軟體。
- Kitty Graphics Protocol 是由 Kitty terminal 開發、現已被多個主流終端模擬器採納的圖形顯示標準，其設計允許在終端機視窗內渲染任意解析度的點陣圖圖片。本系統利用此特性，將遠端 Wayland 桌面的畫面透過 SSH 通道以 KGP 格式傳輸至本地終端機渲染，實現零客戶端安裝的遠端桌面體驗。

核心技術與系統架構

- 系統由遠端單一執行檔組成，分為以下四個部分

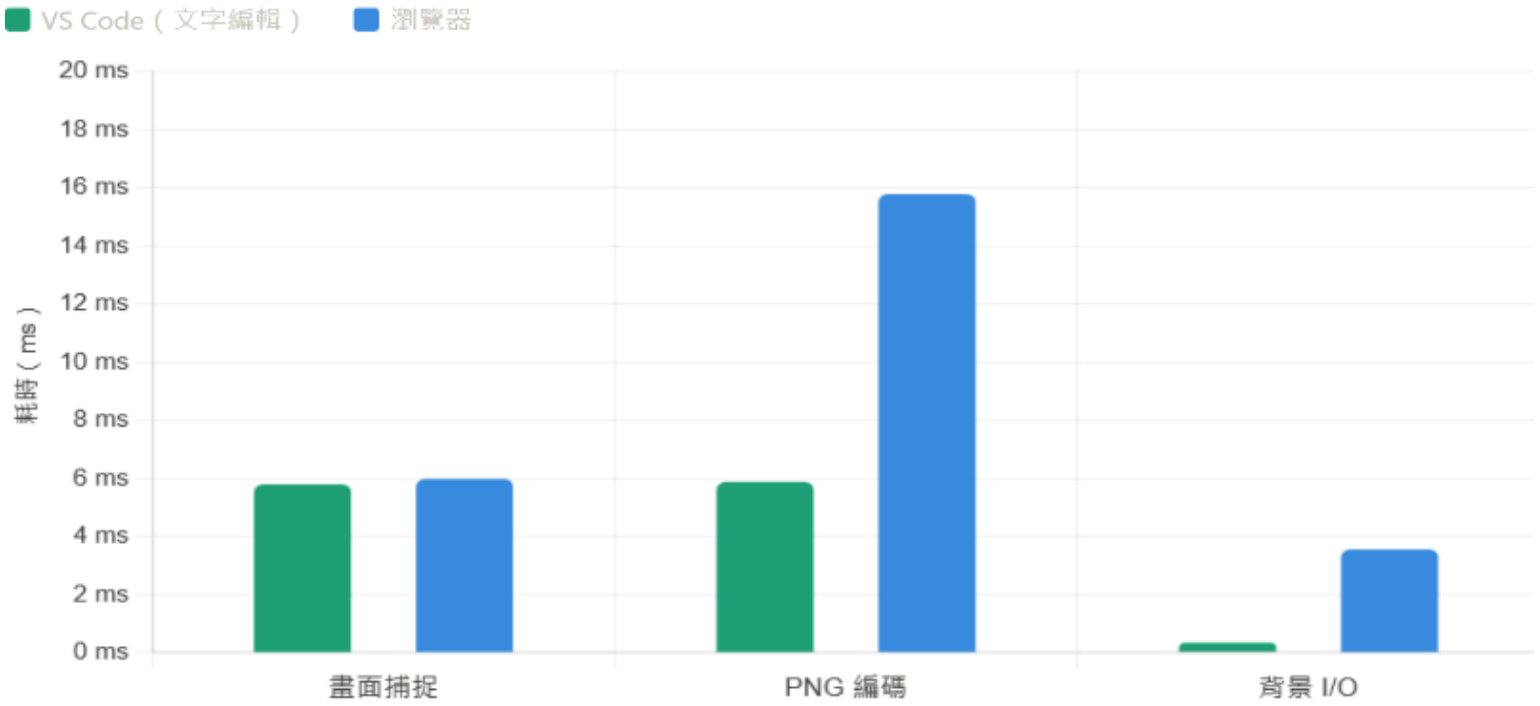
1. 畫面捕捉：透過 wlr-screencopy Wayland protocol 捕捉 Hyprland compositor 的 framebuffer，取得 XRGB8888 格式的原始像素資料。
2. 差異偵測（Dirty Rect）：將畫面切割為 64×64 像素的 tile 進行逐行比對，取出所有變動 tile 的 bounding box，僅傳輸有變動的畫面區域，大幅降低靜態桌面的傳輸量。
3. 圖像編碼：透過 libpng 編碼為 PNG 格式可以搭配使用 Median Cut 演算法將 RGB24 色彩空間量化為 256 色調色盤（PNG-8）；相較 PNG-24，傳輸資料量降低約50–70%。
4. 輸入注入：解析本地端 Kitty Keyboard Protocol（KKP）的 escape sequence，將按鍵及滑鼠事件透過 ydotool 注入遠端 Wayland 環境，支援 SGR 1016 pixel-level 滑鼠座標。

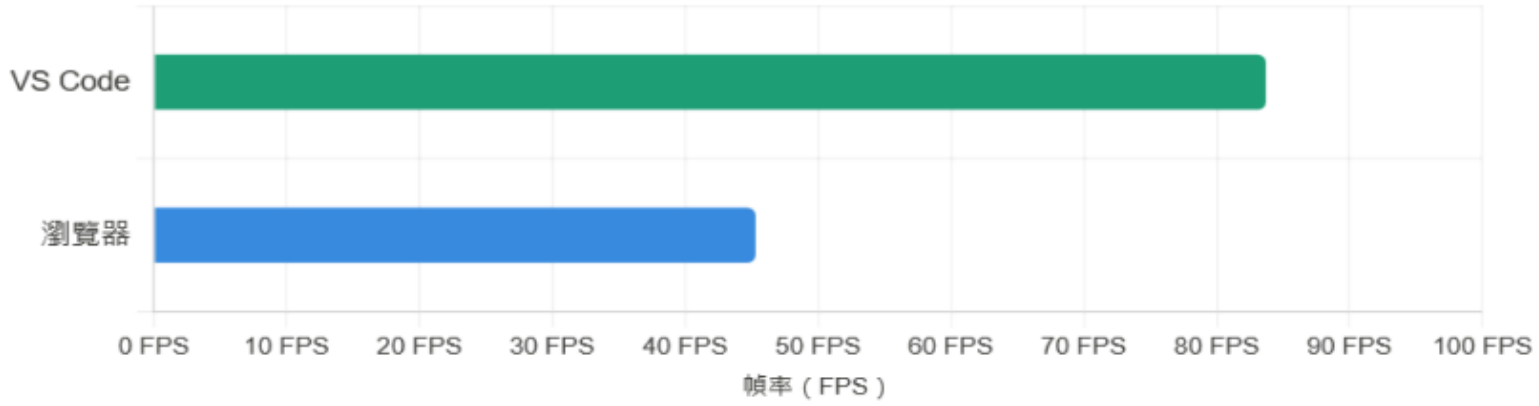
系統架構圖


本地端（Client）
Kitty Terminal
（支援 KGP 的終端模擬器）
Kitty Graphics Protocol
KKP + SGR 1016
鍵盤 / pixel 滑鼠輸入
（SGR 1016 Mouse）
本地零額外安裝
Kitty · Ghostty ·
WezTerm · Konsole
KGP escape sequence（stdout）
（影像資料流）
KKP / SGR 1016（stdin）
（輸入事件流）
遠端主機 kgp-remote（Daemon）
影像輸出流程（stdout → KGP）
① wlr-screencopy
Hyprland framebuffer → XRGB8888
② dirty rect
64×64 tile diff，跳過靜止畫面
③ median cut + PNG-8
256 色量化，傳輸量降低 50–70%
④ base64 → Kitty GP
a=T / a=f，stdout 輸出
封裝為 KGP escape sequence
輸入處理流程（stdin → Wayland）
⑤ VT parser
解析 KKP / SGR 1016
（鍵盤 / 滑鼠事件）
⑥ ydotool
模擬鍵盤 / 滑鼠事件
⑦ Wayland Input Injection
注入到 Wayland 合成器
（如 Hyprland）
kgp-remote Daemon
以使用者權限執行，讀取螢幕並注入輸入事件
影像資料流（遠端 → 本地）
wlr-screencopy → … → base64 → KGP escape sequence → stdout
輸入事件流（本地 → 遠端）
stdin ← KKP / SGR 1016 Mouse

結果與效能表現

- 系統在兩種情境下進行效能量測。 VS Code 文字編輯場景畫面變動極小，平均每幀處理時間降至約 12 ms，達到 83.7 FPS 且無影格丟失。瀏覽器操作場景因動態內容較多，dirty rect 面積較大，平均影格處理時間約 22 ms，影格損失率約 5%，達到 45.3 FPS。


VS Code（文字編輯）
瀏覽器
20 ms
18 ms
16 ms
14 ms
12 ms
10 ms
8 ms
6 ms
4 ms
2 ms
0 ms
耗時（ms）
畫面捕捉
PNG 編碼
背景 I/O


VS Code
瀏覽器
0 FPS
10 FPS
20 FPS
30 FPS
40 FPS
50 FPS
60 FPS
70 FPS
80 FPS
90 FPS
100 FPS
幀率（FPS）

結論與未來展望

- 系統驗證了以終端圖形協議實現零客戶端遠端桌面的可行性。靜態內容場景（VS Code 文字編輯）FPS 達 83.7，充分展示 Dirty Rect 差異壓縮的效益。未來方向：改進動態場景壓縮效率、探索硬體加速編碼路徑（VAAPI / AMF）。

參考資料及程式原始碼

- https://sw.kovidgoyal.net/kitty/graphics-protocol
- https://sw.kovidgoyal.net/kitty/keyboard-protocol/
- https://invisible-island.net/xterm/ctlseqs/ctlseqs.html
- https://github.com/ReimuNotMoe/ydotool
- https://github.com/PinkNekoFist/kitty-desk