

WebScreen 遠端桌面提速功能報告(兩階段)

日期:2026-08-08 結果:實測過關,接近 TeamViewer 體感

一、訴求與現象

客戶普遍反映 WebScreen 遠端桌面比 TeamViewer 慢很多(點擊到看到反應約 1 秒,TeamViewer 約 0.15~0.25 秒)。目標:不求追平,但不能差太多。

環節	耗時	說明
輸入上行	10-50ms	常駐 WebSocket, 原本已優
抓屏	70-130ms	fps=15 → 平均等 33-66ms 才有下一幀
編碼+網路	50-100ms	ultrafast/zerolateness/TCP_NODELAY, 原本已優
瀏覽器 MSE 解碼緩衝	300-800ms	★最大瓶頸:mpegts.js 走 MSE 架構的天花板

二、階段一:參數極限化(延遲降到 0.4~0.6 秒)

修改	效果
pusher GOP 寫死 -g 15 改隨 fps 動態(固定 1 秒 keyframe)	低 fps 恢復快、高 fps 不浪費碼率
mpegts.js MaxLatency 0.8→0.5、MinRemain 0.2→0.1	追趕上限收緊,互動延遲約 -0.3 秒
fps 選項加 30/25(原最高 20)	滑鼠回饋流暢度倍增(ddagrab+QS V 硬體鏈 CPU 增量小)
解析度加 1920x1080 原生、碼率加 3000/2500K	1080p 不再縮 720p, 文字清晰(模 糊感也會被感知成慢)

三、階段二:WebCodecs 直解(延遲 0.15~0.25 秒,結案)

核心:MSE 是瀏覽器內部強制緩衝的播放管線,參數擰到極限仍有 0.3 秒以上。WebCodecs 的 VideoDecoder 讓 JS 直接餵 H.264 硬解、逐幀畫 canvas,收到幀就顯示,完全沒有緩衝層。

端	實作
橋接(stk_webscreen.py)	新增 Tsh264Demux:mpegts→PAT/PMT 解析→視訊 PES 重組→每個 PES=一個 H.264 Annex-B 存取單元;新增/wsc 端點,同一條 relay 串流就地拆包逐 AU 直送。pusher/relay/Agent 零改動,純 stdlib 約 100 行
瀏覽器	偵測 VideoDecoder→直解路徑:optimizeForLatency、codec 兩段嘗試(High→Baseline)、JS 掃 NAL 判 keyframe(5/7=key,1=delta)、等第一個含 SPS 的 key 才 configure(annexb 格式免 description);VideoFrame 直繪 canvas
操控零改動	canvas 墊在「轉透明的 video」底下,滑鼠/觸控事件仍由原 video 元素接收,座標換算 map()/unmap() 改吃 canvas 幀尺寸
相容鏈	WebCodecs 任何失敗→自動退 mpegts.js→再退 HLS,三層全保留;凍結 watchdog 改吃解碼幀數當進度;狀態列顯示「已連線(低延遲直解)」辨識路徑

四、踩坑與陷阱

坑	說明
video CSS 黑底蓋住 canvas	canvas 墊底方案必須同時把 video 背景轉 transparent,否則整片黑
mid-stream 加入缺 SPS	configure 必須等第一個含 SPS 的 keyframe;好在每條連線都是新 ffmpeg,串流頭必有 SPS/PPS(x264 另有 repeat-headers=1 每秒重發)
MSE 天花板誤判	參數調到極限(stash 關/追趕 0.5s)仍卡 0.3 秒以上,「再擰參數」是死路,要換架構才有質變
JS 內嵌 Python 三引號	改完必抽出 node --check;本次同時用真 ffmpeg 產 mpegts 餵 demuxer 驗證(44 AU/keyframe 位置/SPS/亂序 chunk)

五、使用技術與驗證

項目	內容
技術	WebCodecs VideoDecoder/EncodedVideoChunk、mpegts TS/PES 解析、Annex-B NAL、canvas 2D、WebSocket binary、x264 zerolatency/GOP 調校
支援面	桌面 Chrome/Edge 全支援;iPhone Safari 16.4+ 支援;更舊裝置自動 fallback
驗證	py_compile+抽頁面 JS node --check+ffmpeg 真流餵 demuxer;實測體感過關
部署	Stk_WebScreen.exe 重建(--clean)→W:\socket\webscreen;客戶派發走 uploader SOCKET 群組