LIV Control Center (Hub) vs LCK Control (Companion App)

Comprehensive Architecture Comparison & Unification Strategy

1. Executive Summary

LIV has two parallel applications for managing game streaming on Quest:

	Hub (`liv-control-center`)	**Control (`lck-control`)**
Stage	Production (Quest Store, low reviews)	Prototype (new architecture)
Stack	Rust + Tauri + Leptos (WASM)	Kotlin + Jetpack Compose + Hilt
Streaming	App captures screen & encodes	Game encodes directly from render pipeline
Communication	Async via backend server	Synchronous IPC via AIDL
Destinations	Single target	Multi-destination
UE5 Plugin	`LCKStreaming` (HTTP/JSON-RPC)	`LCKControl` (AIDL/JNI)

2. High-Level Architecture

2.1 Hub Architecture

```
graph TB
    subgraph Quest Headset
        subgraph "Hub App (Tauri + Leptos WASM)"
            UI_H["Leptos UI
(WASM)"]
            Core_H["Rust Core
(Tauri Backend)"]
            Encoder_H["MediaCodec
H.264 + AAC"]
            RTMP_H["minirtmp
(RTMP Client)"]
            Capture["ScreenCaptureService
(MediaProjection)"]
        end
        subgraph "UE5 Game"
            Plugin_S["LCKStreaming Plugin"]
            API_Client["HTTP/JSON-RPC Client"]
        end
    end

    subgraph "Cloud Server"
        Backend_H["Hub Backend
```

```
(api.obi.gg)"]
    end

    subgraph "Streaming Platforms"
        YT["YouTube Live"]
        TW["Twitch"]
    end

    UI_H <-->|Tauri IPC| Core_H
    Core_H -->|JSON-RPC 2.0
HTTPS + Cert Pinning| Backend_H
    Plugin_S -->|JSON-RPC 2.0
HTTPS| Backend_H
    Backend_H -->|"Device Pairing
(async polling)"| Plugin_S
    Core_H --> Capture
    Capture --> Encoder_H
    Encoder_H --> RTMP_H
    RTMP_H -->|RTMP| YT
    RTMP_H -->|RTMP| TW

    style Backend_H fill:#f96,stroke:#333
    style Capture fill:#ff9,stroke:#333
    style Encoder_H fill:#ff9,stroke:#333
```

Key: The Hub app captures the screen, encodes it, and streams. The game and hub communicate indirectly through the backend server.

2.2 Control App Architecture

```
graph TB
    subgraph Quest Headset
        subgraph "Control App (Kotlin + Compose)"
            UI_C["Compose UI"]
            VM["ViewModels + Repos"]
            Service["LckControlService
(Foreground + AIDL)"]
            DB["Room DB
(Local Cache)"]
        end
        subgraph "UE5 Game"
            Plugin_C["LCKControl Plugin"]
            JNI["JNI Bridge"]
            SDK["lck-control-sdk
(AAR)"]
            Encoder_C["LCK Encoder
(H.264 + AAC)"]
            RTMP_C1["RTMP Sink 1"]
            RTMP_C2["RTMP Sink 2"]
            RTMP_CN["RTMP Sink N"]
        end
```

```
    end

    subgraph "Self-Hosted Server"
        Backend_C["Control Backend
(Node.js + Fastify)"]
        SQLite["SQLite DB"]
    end

    subgraph "Streaming Platforms"
        YT2["YouTube Live"]
        TW2["Twitch"]
        Manual["Custom RTMP"]
    end

    UI_C <--> VM
    VM <-->|REST API
JWT Auth| Backend_C
    VM <--> DB
    VM <--> Service

    Plugin_C --> JNI
    JNI --> SDK
    SDK <-->|"AIDL IPC
(Bound Service)"| Service

    Backend_C <--> SQLite
    Backend_C -->|"OAuth + RTMP
Resolution"| YT2
    Backend_C -->|"OAuth + RTMP
Resolution"| TW2

    Encoder_C --> RTMP_C1
    Encoder_C --> RTMP_C2
    Encoder_C --> RTMP_CN
    RTMP_C1 -->|RTMP| YT2
    RTMP_C2 -->|RTMP| TW2
    RTMP_CN -->|RTMP| Manual

    style Service fill:#9f9,stroke:#333
    style SDK fill:#9f9,stroke:#333
    style Encoder_C fill:#9cf,stroke:#333
```

Key: The game encodes directly from its render pipeline and streams to multiple destinations. The companion app provides stream configuration via direct IPC.

---

3. Communication Model Comparison

3.1 Hub: Server-Mediated Async Communication

```
sequenceDiagram
    participant Game as UE5 Game
(LCKStreaming)
    participant Server as Hub Backend
(api.obi.gg)
    participant Hub as Hub App
(Tauri)
    participant Platform as YouTube/Twitch

    Note over Game,Hub: Device Pairing (one-time)
    Game->>Server: create_device_login_attempt()
    Server-->>Game: 6-digit pairing code
    Game->>Game: Display code to user
    Hub->>Server: pair_device(code)
    Server-->>Hub: Device paired

    Note over Game,Hub: Stream Setup
    Hub->>Server: get_user_profile()
    Server-->>Hub: Streaming target + RTMP URL
    Hub->>Hub: Start screen capture
    Hub->>Hub: Encode H.264 + AAC
    Hub->>Platform: RTMP stream

    Note over Game,Server: Game has no direct
connection to Hub
    Game->>Server: Poll for updates (2.5s)
    Server-->>Game: Current state
```

3.2 Control: Direct IPC Communication

```
sequenceDiagram
    participant Game as UE5 Game
(LCKControl + JNI)
    participant App as Control App
(AIDL Service)
    participant Server as Control Backend
(Fastify)
    participant Platform as YouTube/Twitch

    Note over Game,App: Service Binding (direct)
    Game->>App: bindService() via AIDL
    App-->>Game: ILckControlService binder
    Game->>App: registerAsClient("MyGame", pkg)
    App-->>Game: clientId

    Note over Game,Platform: Stream Lifecycle
    Game->>App: getStreamPlans()
    App-->>Game: List
    Game->>App: prepareStreamPlan(planId)
    App->>Server: POST /streams/plans/{id}/prepare
    Server->>Platform: Create broadcast + get RTMP URLs
```

```
    Platform-->>Server: RTMP URLs + stream keys
    Server-->>App: PrepareResponse
    App-->>Game: StreamPlan (with RTMP data)

    Game->>Game: Encode from render pipeline
    Game->>Platform: RTMP stream (dest 1)
    Game->>Platform: RTMP stream (dest 2)

    Game->>App: startStreamPlan(planId)
    App->>Server: POST /streams/plans/{id}/start
    Server->>Platform: Transition broadcast to LIVE
```

---

4. Technology Stack Comparison

4.1 Application Layer

Component	Hub	Control
Language	Rust (95%) + Kotlin (JNI)	Kotlin (100%)
UI Framework	Leptos 0.8.2 (Rust WASM)	Jetpack Compose (2024.09 BOM)
App Framework	Tauri v2.6.2	Native Android
Styling	TailwindCSS v4	Material Design 3
State Mgmt	Leptos reactive signals	StateFlow + collectAsStateWithLifecycle
DI	None (manual wiring)	Hilt 2.59.2
Navigation	Leptos Router	Compose Navigation 2.8.4
Local Storage	Platform credential store	Room 2.8.4 + EncryptedSharedPreferences
HTTP Client	reqwest (rustls TLS)	Retrofit 2.11.0 + OkHttp 4.12.0
JSON	serde_json	Moshi 1.15.1
Auth SDK	Meta Horizon Platform SDK 77.0.1	Meta Horizon Platform SDK 77.0.1
Crash Reporting	Sentry (Android SDK bridge)	None

4.2 Backend Layer

Component	Hub Backend	Control Backend
Hosting	Cloud (`api.obi.gg`)	Self-hosted (Docker on NAS, port 3100)
Protocol	JSON-RPC 2.0	REST (JSON)
Stack	Unknown (external)	Node.js 20 + Fastify 5 + TypeScript 5.7
Database	Unknown	SQLite (Prisma 6.4 ORM)
Auth	JWT (via JSON-RPC response headers)	JWT HS256 (jose 6.0)

Component	Hub Backend	Control Backend
Token Security	Unknown	AES-256-GCM encryption + SHA256 hashing
OAuth	Server handles YouTube/Twitch	Server handles YouTube/Twitch
Rate Limiting	Unknown	100 req/min (Fastify plugin)
Deployment	Managed cloud	Docker + docker-compose

4.3 UE5 Plugin Layer

Component	LCKStreaming (Hub)	LCKControl (Companion)
Communication	HTTP/JSON-RPC 2.0	AIDL via JNI
Transport	HTTPS (cross-network)	Local IPC (same device)
Auth Flow	Device code (6-digit) + polling	Direct service binding
Token Storage	EncryptedSharedPreferences	None (companion owns tokens)
RTMP Sinks	1 (single destination)	N (multi-destination)
Blocking Model	Async HTTP callbacks	Synchronous JNI calls
Platform Support	Cross-platform capable	Android only
Latency	Network round-trip (100ms+)	IPC (~1ms)
Offline Capable	No (requires server)	Partial (companion has local cache)

5. Streaming Architecture Deep Dive

5.1 Hub: Screen Capture + Re-Encoding

```
graph LR
    subgraph "UE5 Game Process"
        Render["Game Renderer
(GPU)"]
    end

    subgraph "Android OS"
        FB["Framebuffer /
Display Compositor"]
        MP["MediaProjection
(Screen Capture API)"]
    end

    subgraph "Hub App Process"
        VD["VirtualDisplay"]
        MC_V["MediaCodec
(H.264 Encoder)"]
        MC_A["MediaCodec
```

```
(AAC Encoder)"]
        AR["AudioRecord
(System Audio)"]
        MR["minirtmp
(RTMP Client)"]
    end

    subgraph "CDN"
        RTMP["YouTube / Twitch
RTMP Ingest"]
    end

    Render --> FB
    FB --> MP
    MP --> VD
    VD --> MC_V
    AR --> MC_A
    MC_V --> MR
    MC_A --> MR
    MR --> RTMP

    style FB fill:#fbb,stroke:#333
    style MP fill:#fbb,stroke:#333
```

Problems:

- Extra GPU copy through display compositor
- Re-encoding already rendered frames (quality loss)
- Higher latency (capture → encode → send)
- Higher battery/thermal impact (two encoding passes)
- Captures UI overlays, notifications, system bars
- Resolution limited to display resolution

5.2 Control: Direct Render Pipeline Encoding

```
graph LR
    subgraph "UE5 Game Process"
        Render["Game Renderer
(GPU)"]
        SCC["SceneCaptureComponent2D
(Render Target)"]
        ENC["LCK Encoder
(H.264 + AAC)"]
        S1["RTMP Sink 1
(YouTube)"]
        S2["RTMP Sink 2
(Twitch)"]
        S3["RTMP Sink 3
(Custom)"]
    end
```

```
    subgraph "CDN"
        YT["YouTube RTMP"]
        TW["Twitch RTMP"]
        CU["Custom RTMP"]
    end

    Render --> SCC
    SCC --> ENC
    ENC --> S1
    ENC --> S2
    ENC --> S3
    S1 --> YT
    S2 --> TW
    S3 --> CU

    style SCC fill:#bfb,stroke:#333
    style ENC fill:#bfb,stroke:#333
```

Advantages:

- Direct GPU texture access (no compositor overhead)
- Single encode pass (game scene only, no UI clutter)
- Lower latency
- Lower battery/thermal impact
- Configurable resolution independent of display
- Multi-destination from single encode
- Clean game footage (no system overlays)

---

6. Feature Comparison Matrix

Feature	Hub	Control	Notes
Meta/Quest Login	Yes	Yes	Both use Horizon Platform SDK 77.0.1
YouTube OAuth	Yes	Yes	Both server-side token exchange
Twitch OAuth	Yes	Yes	Both server-side token exchange
Multi-Destination Streaming	No (1)	Yes (N)	Major difference
Stream Plans	No	Yes	Control has full lifecycle management
Direct Game Encoding	No	Yes	Control encodes from render pipeline
Screen Capture Streaming	Yes	No	Hub captures and re-encodes

Feature	Hub	Control	Notes
Custom RTMP Targets	Yes	Yes	Both support manual RTMP
Game Client Management	Yes (pairing)	Yes (AIDL)	Different mechanisms
IGDB Game Database	Yes	No	Hub has game cover art
Watermark	Yes	No	Hub has overlay support
Subscription Model	Yes	No	Hub has paid tier
Sentry Crash Reporting	Yes	No	Hub has telemetry
Certificate Pinning	Yes	No	Hub has SPKI pinning
Offline Caching	No	Yes	Control has Room DB
Background Token Refresh	Unknown	Yes	Control backend has scheduler
CI/CD Pipeline	Yes (Jenkins)	Partial (deploy.ps1)	Hub has full CI
Desktop Support	Yes	No	Tauri supports desktop
Cross-Platform	Yes (Desktop + Android)	No (Android only)	Hub has wider reach

7. Pros and Cons

7.1 Hub (liv-control-center)

Pros

- **Cross-platform**: Tauri supports Desktop + Android, one codebase
- **Self-contained streaming**: No dependency on game integration
- **Works with any game**: Screen capture works regardless of game engine support
- **Production infrastructure**: Jenkins CI/CD, Sentry, cloud backend
- **Rich features**: IGDB, watermarks, subscription model
- **Rust performance**: Memory-safe, low-level control over encoding

Cons

- **Screen capture quality**: Re-encoding degrades quality, captures overlays
- **Higher resource usage**: Extra GPU copy + encode pass drains battery faster
- **Single destination**: Can only stream to one platform at a time
- **Complex stack**: Rust + WASM + Tauri + Kotlin JNI is hard to maintain
- **Server dependency**: All communication goes through cloud backend
- **Latency**: Network round-trips for game communication (polling every 2.5s)
- **Low store reviews**: Users experiencing issues (reason for this analysis)

- **Niche UI framework**: Leptos (WASM) has small ecosystem vs Compose
- **No stream plans**: Simple streaming model without plan lifecycle

7.2 Control App (lck-control)

Pros

- **Direct render pipeline**: Game encodes from GPU, best possible quality
- **Multi-destination**: Stream to YouTube + Twitch + custom simultaneously
- **Low latency IPC**: AIDL communication in ~1ms vs 100ms+ network calls
- **Stream plans**: Full lifecycle (DRAFT → READY → LIVE → ENDED)
- **Clean architecture**: Standard Android stack (Compose, Hilt, Room, Retrofit)
- **Own backend**: Full control over API, auth, token management
- **SDK module**: Clean AAR for UE5 consumption via JNI
- **Lower resource usage**: No screen capture or re-encoding overhead
- **Maintainable**: Kotlin + Compose is mainstream Android with large ecosystem
- **Offline caching**: Room DB + encrypted token store

Cons

- **Android only**: No desktop support
- **Requires game integration**: Game must use LCKControl plugin (not universal)
- **Prototype stage**: Not production-ready yet
- **Self-hosted backend**: Requires infrastructure management (Docker on NAS)
- **No CI/CD**: Manual builds via PowerShell script
- **No crash reporting**: No Sentry or equivalent
- **No subscription model**: No monetization built in
- **No IGDB integration**: No game metadata/artwork
- **Blocking IPC**: Synchronous JNI calls could cause ANRs if slow

---

8. UE5 Plugin Comparison

8.1 LCKStreaming Plugin (uses Hub)

```
stateDiagram-v2
    [*] --> Idle
    Idle --> LoggingIn: StartLogin()
    LoggingIn --> WaitingForCode: create_device_login_attempt
    WaitingForCode --> Polling: Display 6-digit code
    Polling --> Authenticated: check_device_login_attempt
(every 2.5s)
    Polling --> Polling: Not yet paired
    Authenticated --> FetchingProfile: get_user_profile
    FetchingProfile --> Ready: Got RTMP target
    Ready --> Streaming: StartStreaming()
    Streaming --> Ready: StopStreaming()
    Ready --> Idle: Logout()
```

```
    note right of Polling
        User must manually enter
        code in Hub app or website
    end note
```

Architecture:

- `ULCKStreamingSubsystem` — GameInstance subsystem, owns API client + RTMP sink
- `FLCKStreamingApiClient` — HTTP client, JSON-RPC 2.0, cert pinning
- `FLCKRtmpSink` / `FLCKRtmpClient` — Single RTMP connection via librtmp
- Auth token stored in platform credential store
- Single streaming target resolved by backend

8.2 LCKControl Plugin (uses Companion App)

```
stateDiagram-v2
    [*] --> Disconnected
    Disconnected --> Connecting: ConnectToCcompanionApp()
    Connecting --> Connected: AIDL service bound
(poll every 1s)
    Connected --> HasPlans: GetStreamPlans()
    HasPlans --> Prepared: PrepareStreamPlan(planId)
→ RTMP URLs resolved
    Prepared --> Streaming: StartStreamPlan(planId)
+ Attach N RTMP sinks
    Streaming --> Prepared: EndStreamPlan(planId)
    Connected --> Disconnected: DisconnectFromCompanionApp()

    note right of Connected
        Direct AIDL binding,
        no pairing code needed
    end note

    note right of Streaming
        Multiple RTMP sinks active
        simultaneously
    end note
```

Architecture:

- `ULCKControlSubsystem` — GameInstance subsystem, owns JNI bridge + multiple RTMP sinks
- `LCKControlAndroid.cpp` — ~700 lines of JNI bindings to `LckControlClient` (AAR)
- Multiple `FLCKRtmpSink` instances — one per stream destination
- No token management — companion app handles all auth
- Full stream plan lifecycle control

8.3 Shared Infrastructure (LCK Base Plugin)

Both plugins share:

- `ILCKStreamingFeature` — Common interface (StartLogin, StartStreaming, StopStreaming, etc.)
- `ILCKEncoderFactory` — Encoder creation
- `ULCKRecorderSubsystem` — Encoder lifecycle management
- `FLCKRtmpSink` / `FLCKRtmpClient` — RTMP transport layer
- H.264 + AAC encoding via platform-specific backends (NVCodec, MediaCodec)

---

9. Backend Comparison

9.1 Hub Backend (`api.obi.gg`)

```
graph TB
    subgraph "Cloud (Managed)"
        API_H["Hub Backend API"]
        DB_H["Database
(Unknown)"]
        IGDB["IGDB API
(Game Metadata)"]
    end

    Hub["Hub App"] -->|"JSON-RPC 2.0
POST /api/rpc"| API_H
    Game_S["LCKStreaming
Plugin"] -->|"JSON-RPC 2.0
POST /api/rpc"| API_H
    API_H --> DB_H
    API_H --> IGDB

    style API_H fill:#f96,stroke:#333
```

Known RPC Methods:

- `LoginUser`, `RefreshUser` — Auth
- `ListMyStreamingTargets`, `CreateStreamingTarget`, `UpdateStreamingTarget`, `DeleteStreamingTarget` — Targets
- `PairDevice`, `UnpairDevice`, `GetConnectedGames` — Device management
- `StartStreaming`, `StopStreaming` — Stream events
- `SearchIgdbGames` — Game metadata
- `CreateOauthConnectIntent`, `GetOauthConnectIntent` — OAuth

9.2 Control Backend (`lck-control-backend`)

```
graph TB
    subgraph "Self-Hosted (Docker on NAS)"
        API_C["Fastify 5 API
(TypeScript)"]
        Prisma["Prisma 6.4 ORM"]
        SQLite["SQLite DB"]
        Scheduler["Token Refresh
```

```
Scheduler (10min)"]
    end

    App["Control App"] -->|"REST API
JWT Bearer Auth"| API_C
    API_C --> Prisma --> SQLite
    Scheduler --> API_C

    API_C -->|OAuth| Google["Google OAuth"]
    API_C -->|OAuth| Twitch_API["Twitch OAuth"]
    API_C -->|Nonce Validate| Meta_Graph["Meta Graph API"]
    API_C -->|Live API| YT_API["YouTube Live API"]
    API_C -->|Helix API| TW_API["Twitch Helix API"]

    style API_C fill:#9cf,stroke:#333
```

REST Endpoints:

Group	Endpoints
Auth	`POST /auth/meta/callback`, `POST /auth/refresh`, `GET /auth/me`, `POST /auth/logout`
Providers	`GET /providers/accounts`, `GET /providers/{yt|tw}/auth-url`, `POST /providers/{yt|tw}/callback`, `DELETE /providers/:serviceId`
Streams	`GET /streams/plans`, `POST /streams/plans`, `GET /streams/plans/:id`, `DELETE /streams/plans/:id`
Lifecycle	`POST /streams/plans/:id/prepare`, `POST /streams/plans/:id/start`, `POST /streams/plans/:id/end`

10. Data Flow Comparison

10.1 Hub: Centralized Server Model

```
graph LR
    subgraph "Data Ownership"
        direction TB
        Server_H["Hub Backend
(owns ALL data)"]
    end

    Hub_App["Hub App
(thin client)"] <-->|"All state via
JSON-RPC"| Server_H
    Game_H["UE5 Game
(paired device)"] <-->|"All state via
JSON-RPC"| Server_H
    YT_H["YouTube API"] <--> Server_H
    TW_H["Twitch API"] <--> Server_H
```

```
    style Server_H fill:#f96,stroke:#333
```

- **Single source of truth**: Backend server
- **No local cache**: App relies on network for all state
- **Game is decoupled**: Only communicates with server, never with app
- **Offline = broken**: Cannot function without server connectivity

10.2 Control: Distributed Ownership Model

```
graph LR
    subgraph "Data Ownership"
        direction TB
        Server_C["Control Backend
(tokens, plans,
OAuth)"]
        App_C["Control App
(local cache,
session tokens)"]
        Game_C["UE5 Game
(RTMP streams)"]
    end

    App_C <-->|REST API| Server_C
    Game_C <-->|"AIDL IPC
(stream plans,
RTMP config)"| App_C
    Server_C <--> YT_C["YouTube API"]
    Server_C <--> TW_C["Twitch API"]
    Game_C -->|"RTMP
(direct)"| CDN["YouTube / Twitch
RTMP Ingest"]

    style App_C fill:#9f9,stroke:#333
    style Game_C fill:#9cf,stroke:#333
```

- **Distributed state**: Backend (tokens, plans), App (cache, session), Game (streams)
- **Local caching**: Room DB provides offline access to plans and accounts
- **Game is tightly coupled**: Direct IPC with companion app
- **Partial offline**: Can view cached plans without network

---

11. Unification Strategy

11.1 Recommended Direction: Evolve Control App into Production

The Control architecture is fundamentally superior for game streaming because:

1. **Direct encode > screen capture** — Quality, performance, and battery life

2. **Multi-destination > single target** — Key user-facing feature
3. **IPC > server polling** — Reliability and responsiveness
4. **Stream plans > ad-hoc streaming** — Better UX for recurring setups
5. **Standard Android stack > Rust/WASM** — Easier maintenance and hiring

11.2 Migration Roadmap

```
gantt
    title Unification Roadmap
    dateFormat YYYY-MM-DD
    axisFormat %b %Y

    section Phase 1: Production Readiness
    CI/CD pipeline (Jenkins/GH Actions)        :p1a, 2026-03-01, 14d
    Sentry crash reporting                      :p1b, 2026-03-01, 7d
    Backend deploy to cloud                     :p1c, 2026-03-08, 7d
    Certificate pinning (OkHttp)                :p1d, 2026-03-08, 3d

    section Phase 2: Feature Parity
    IGDB game metadata integration              :p2a, 2026-03-15, 7d
    Watermark / overlay support in encoder      :p2b, 2026-03-15, 10d
    Subscription model + paywall                :p2c, 2026-03-22, 14d

    section Phase 3: Hub Migration
    Add fallback screen-capture mode            :p3a, 2026-04-05, 14d
    Port device pairing (for non-integrated games) :p3b, 2026-04-05, 10d
    Migrate Hub users to Control                :p3c, 2026-04-19, 14d
    Deprecate Hub app                           :p3d, 2026-05-03, 7d

    section Phase 4: Polish
    Desktop companion (optional)                :p4a, 2026-05-10, 21d
    Advanced stream analytics                   :p4b, 2026-05-10, 14d
    Store listing + marketing                   :p4c, 2026-05-24, 7d
```

11.3 What to Keep from Each

```
graph TB
    subgraph "Unified App"
        direction TB
        A["Control App Architecture
(Kotlin + Compose + Hilt)"]
        B["Control Backend
(Fastify + Prisma + SQLite)"]
        C["LCKControl Plugin
(AIDL + multi-destination)"]
        D["Stream Plan System
(DRAFT → READY → LIVE → ENDED)"]
    end
```

```
    subgraph "Adopt from Hub"
        E["Sentry Crash Reporting"]
        F["IGDB Game Database"]
        G["Certificate Pinning"]
        H["Jenkins CI/CD"]
        I["Watermark Renderer"]
        J["Screen Capture Fallback"]
    end

    subgraph "Discard"
        K["Rust/Tauri/Leptos Stack"]
        L["JSON-RPC 2.0 Protocol"]
        M["minirtmp (Rust RTMP)"]
        N["Device Code Pairing
(replaced by AIDL)"]
        O["Single-Destination Limit"]
    end

    E --> A
    F --> B
    G --> A
    H --> A
    I --> C
    J --> A

    style A fill:#9f9,stroke:#333
    style B fill:#9cf,stroke:#333
    style C fill:#9f9,stroke:#333
    style D fill:#9f9,stroke:#333
    style K fill:#fbb,stroke:#333
    style L fill:#fbb,stroke:#333
    style M fill:#fbb,stroke:#333
    style N fill:#fbb,stroke:#333
    style O fill:#fbb,stroke:#333
```

11.4 Hybrid Mode: Screen Capture Fallback

To maintain the Hub's "works with any game" advantage, add a fallback path:

```
graph TB
    Start["Game Launches"] --> Check{"LCKControl Plugin
integrated?"}
    Check -->|Yes| AIDL["AIDL IPC Path
(direct encode,
multi-destination)"]
    Check -->|No| Capture["Screen Capture Path
(MediaProjection,
single destination)"]
    AIDL --> Stream["Stream to Platforms"]
    Capture --> Stream
```

```
    style AIDL fill:#9f9,stroke:#333
    style Capture fill:#ff9,stroke:#333
```

This gives the unified app both modes:

- **Primary**: Direct encoding via AIDL (high quality, multi-destination)
- **Fallback**: Screen capture for games without plugin integration (compatibility)

---

12. Risk Assessment

Risk	Impact	Mitigation
Hub users lose access during migration	High	Run both apps in parallel during transition, provide migration guide
AIDL only works on Android (no desktop)	Medium	Screen capture fallback for desktop; evaluate PCVR needs later
Self-hosted backend scalability	Medium	Move to managed cloud (Railway, Fly.io) before store launch
Synchronous JNI blocking causes ANR	Medium	Add timeout handling, move to async callback pattern
No subscription model in Control	Low	Implement before store launch using existing Hub billing logic
Losing crash telemetry	Low	Add Sentry SDK early in Phase 1

---

13. Summary Decision Matrix

```
quadrantChart
    title Streaming Quality vs Maintenance Complexity
    x-axis Low Maintenance --> High Maintenance
    y-axis Low Quality --> High Quality
    quadrant-1 Ideal
    quadrant-2 Overengineered
    quadrant-3 Avoid
    quadrant-4 Quick & Dirty

    Control App: [0.35, 0.85]
    Hub App: [0.75, 0.45]
    Unified - Recommended: [0.45, 0.90]
```

Recommendation: The Control App architecture with adopted Hub features provides the best path forward — higher streaming quality with a more maintainable stack. The Hub's Rust/Tauri/Leptos stack adds significant complexity without proportional benefits for an Android-focused product.

Document generated 2026-02-26. Based on analysis of `liv-control-center`, `lck-control`, `lck-control-backend`, and `LCKGame` codebases.