Fire TV Connection Issues - Root Cause Analysis

Date: October 27, 2025
Analyzed By: DeepAgent
Project: Sports Bar TV Controller

Executive Summary

The Fire TV devices are losing connection due to several architectural issues in the connection management system. The primary problem is that ADB client instances and their keep-alive timers are not persisted properly across API calls, leading to connection drops.

Current Architecture Problems

1. Non-Persistent Connection State (CRITICAL)

Location: `src/app/api/firetv-devices/test-connection/route.ts`

Issue:
- Each test connection creates a new ADB client instance
- The keep-alive timer is started but not persisted
- After the API response is sent, Node.js garbage collects the client instance
- The keep-alive timer is destroyed, leaving no mechanism to maintain the connection

Code Evidence:

```
// Line 36-39 in test-connection/route.ts
adbClient = new ADBClient(ip, devicePort, {
  keepAliveInterval: 30000,
  connectionTimeout: 5000
})

// Line 66-68 - Comment says we don't cleanup, but no storage either
// Note: We don't call cleanup() here to keep the connection alive
// The keep-alive mechanism will maintain the connection
```

Impact: Fire TV boxes lose connection shortly after testing because the keep-alive timer doesn’t persist.

---

2. Aggressive Connection Timeout (HIGH)

Location: `src/app/api/firetv-devices/send-command/route.ts`

Issue:
- Connections are automatically cleaned up after 5 minutes of inactivity
- In a sports bar, 5 minutes between commands is very common (during game play)
- This causes frequent reconnections during busy times

Code Evidence:

```
// Line 10 in send-command/route.ts
const CONNECTION_TIMEOUT = 5 * 60 * 1000  // 5 minutes
```

Impact: Connections drop during normal operation when no commands are sent for 5 minutes.

---

3. No Centralized Connection Manager (CRITICAL)

Issue:
- Each API route manages connections independently
- Test connection route doesn't share state with send-command route
- No single source of truth for connection status
- No background service to proactively maintain connections

Current State:
- `test-connection/route.ts`: Creates clients but doesn't persist them
- `send-command/route.ts`: Has a Map but only populates on command send
- No shared connection pool or manager

Impact: Inconsistent connection state across the application.

---

4. Insufficient Reconnection Logic (MEDIUM)

Location: `src/lib/firecube/adb-client.ts`

Issue:
- Keep-alive ping failures trigger immediate reconnection
- No exponential backoff or retry limits
- Could cause connection storms if device is temporarily offline
- No circuit breaker pattern

Code Evidence:

```
// Lines 84-98 in adb-client.ts
this.keepAliveTimer = setInterval(async () => {
  try {
    await this.executeShellCommand('echo keepalive')
  } catch (error) {
    // Immediate reconnection attempt
    try {
      await this.connect()
    } catch (reconnectError) {
      // No backoff, just logs error
    }
  }
}, this.options.keepAliveInterval)
```

Impact: Potential for rapid reconnection attempts that could overwhelm devices or network.

---

5. No Connection Health Monitoring (HIGH)

Issue:
- No background service checking device health
- Connection status only updated on user action (test or command)
- `isOnline` flag in devices.json is never automatically updated
- No alerts or notifications when devices go offline

Evidence:

```
// All devices show isOnline: false in firetv-devices.json
{
  "name": "Amazon 1",
  "isOnline": false,  // Never updated automatically
  "ipAddress": "192.168.10.131"
}
```

Impact: No proactive detection of connection issues.

---

6. Device IP Address Mismatch (HIGH)

Location: `data/firetv-devices.json`

Issue:
- Amazon 1 has IP `192.168.10.131` (different subnet)
- Amazon 2-4 have IPs `192.168.1.132-134` (same subnet)
- Inconsistent network addressing could indicate network configuration issues

Evidence:

```
"Amazon 1": "192.168.10.131",  // .10 subnet
"Amazon 2": "192.168.1.132",   // .1 subnet
"Amazon 3": "192.168.1.133",
"Amazon 4": "192.168.1.134"
```

Impact: Amazon 1 might be unreachable if routing isn't configured properly.

---

7. Missing Error Recovery Features (MEDIUM)

Issues:
- No retry logic with backoff in connection attempts
- No connection pooling with health checks
- No automatic device discovery/recovery
- No notification system for connection failures

Recommended Solutions

Solution 1: Create Global Connection Manager Service (CRITICAL)

Priority: HIGH
Effort: Medium

Implementation:
1. Create a singleton `FireTVConnectionManager` service
2. Maintain a global connection pool for all devices
3. Implement proper lifecycle management
4. Share connections across all API routes

Benefits:
- Single source of truth for all connections
- Persistent keep-alive timers
- Consistent connection state

Solution 2: Implement Background Health Monitor (HIGH)

Priority: HIGH
Effort: Medium

Implementation:
1. Create a background service that runs on server startup
2. Periodically check connection health for all registered devices (every 30-60 seconds)
3. Automatically reconnect failed connections with exponential backoff
4. Update `isOnline` status in real-time
5. Log connection events for debugging

Benefits:
- Proactive connection management
- Automatic recovery from failures
- Real-time device status

Solution 3: Increase Connection Timeout (QUICK WIN)

Priority: HIGH
Effort: Low

Implementation:

```
// Change from 5 minutes to 30 minutes or longer
const CONNECTION_TIMEOUT = 30 * 60 * 1000  // 30 minutes
```

Benefits:
- Immediate improvement in connection stability
- Reduces unnecessary reconnections

Solution 4: Add Exponential Backoff to Reconnection (MEDIUM)

Priority: MEDIUM
Effort: Medium

Implementation:
1. Add retry counter to ADB client
2. Implement exponential backoff (1s, 2s, 4s, 8s, 16s, max 60s)
3. Add maximum retry limit before marking device as offline
4. Reset retry counter on successful connection

Benefits:
- Prevents connection storms
- More resilient to temporary network issues

Solution 5: Fix Network Configuration (CRITICAL)

Priority: HIGH
Effort: Low (if just config) to High (if network redesign needed)

Action Items:
1. Verify network topology and routing rules
2. Ensure all Fire TV devices are on the same subnet or have proper routing
3. Fix Amazon 1's IP address to match subnet (192.168.1.x) or ensure routing works
4. Verify firewall rules allow ADB port (5555)

Benefits:
- Resolves potential connectivity issues for Amazon 1
- Standardizes network configuration

Solution 6: Add Connection Event Logging (QUICK WIN)

Priority: MEDIUM
Effort: Low

Implementation:
1. Create a connection events log
2. Log all connection/disconnection events with timestamps
3. Add API endpoint to view connection history
4. Include in admin dashboard

Benefits:
- Better visibility into connection issues
- Easier debugging and diagnostics

Implementation Priority

Phase 1 (Immediate - Can be done today)

1. ✅ Create Global Connection Manager
2. ✅ Implement Background Health Monitor
3. ✅ Increase connection timeout to 30 minutes
4. ✅ Add exponential backoff

Phase 2 (Follow-up)

1. Add connection event logging and history
2. Create admin dashboard for connection monitoring
3. Implement notifications for connection failures

Phase 3 (Network Fix - Requires Site Visit)

1. Verify and fix network configuration
2. Standardize IP addressing
3. Check physical network infrastructure

Technical Implementation Plan

New Files to Create:

1. `src/services/firetv-connection-manager.ts` - Global connection manager
2. `src/services/firetv-health-monitor.ts` - Background health monitoring
3. `src/app/api/firetv-devices/connection-status/route.ts` - Real-time status API

Files to Modify:

1. `src/lib/firecube/adb-client.ts` - Add exponential backoff
2. `src/app/api/firetv-devices/test-connection/route.ts` - Use connection manager
3. `src/app/api/firetv-devices/send-command/route.ts` - Use connection manager, increase timeout
4. `src/components/FireTVController.tsx` - Show real-time connection status

Expected Outcomes

After implementing these fixes:
1. ✅ Fire TV devices will maintain persistent connections
2. ✅ Automatic reconnection on failure with smart retry logic
3. ✅ Real-time connection status updates in UI
4. ✅ Reduced connection drops during normal operation
5. ✅ Better visibility into connection health
6. ✅ More resilient to network issues

Risk Assessment

Low Risk:
- Connection manager implementation (isolated service)
- Timeout increase (simple config change)
- Adding logging (non-breaking change)

Medium Risk:
- Background health monitor (needs proper testing)
- Exponential backoff (could delay reconnections if not tuned properly)

High Risk:
- Network reconfiguration (requires site visit, could affect other systems)

Testing Strategy

1. **Unit Tests**: Test connection manager and health monitor logic
2. **Integration Tests**: Test with actual Fire TV devices
3. **Load Tests**: Verify multiple concurrent connections
4. **Failure Tests**: Test reconnection logic with simulated failures
5. **Long-Running Tests**: Verify connections stay alive for 24+ hours

---

Conclusion

The root cause of Fire TV connection issues is the lack of persistent connection management. The current architecture creates temporary client instances that are garbage collected, destroying their keep-alive timers. The proposed Global Connection Manager and Background Health Monitor will solve these issues by maintaining persistent connections and proactively monitoring device health.

Next Steps:
1. Implement Global Connection Manager
2. Implement Background Health Monitor
3. Deploy to remote server
4. Monitor connection stability for 24 hours
5. Address network configuration issues if Amazon 1 still has problems