ChatGPT

Ursachenanalyse

Bei der Durchsicht des Repos `hi-its-lukas/HASS-Dashboard` ergeben sich mehrere potenzielle Problemstellen rund um die UniFi Protect Livestreams:

1. Initialisierung der UniFi-Protect-Verbindung

Im Backend wird für UniFi Protect eine **direkte API-Anbindung** genutzt. Zwei getrennte Authentifizierungswege werden erwartet: - **API-Key** für die REST-Integration (Snapshots, Gerätesuche usw.) und - **Benutzer/Passwort** für den **Livestream**.

Stellen Sie sicher, dass beide korrekt im **Einstellungs-Frontend** hinterlegt sind. Der Code prüft dies explizit: Fehlt z.B. Benutzer oder Passwort, gilt UniFi Protect als „nicht konfiguriert“ und der WebSocket schließt sofort mit entsprechender Fehlermeldung [1] [2]. In diesem Fall würde im Frontend **„UniFi Protect nicht konfiguriert“** erscheinen. Da Sie stattdessen „Stream disconnected“ sehen, waren die Grunddaten vermutlich gesetzt. Dennoch sollte geprüft werden:

- **Controller-URL & API-Key:** Die API-Verbindung wird über `ProtectClient` aufgebaut, der per API-Key auf `…/proxy/protect/integration/v1` zugreift [3]. Nutzen Sie die *Testverbindung*-Funktion im UniFi-Einstellungsdialog: Sie sollte **„Verbunden! X Kameras gefunden“** melden. Andernfalls stimmen URL oder API-Key nicht oder die UniFi-Protect-Version unterstützt die Integrations-API nicht (das Backend wirft z.B. bei HTTP 404 einen Fehler „Integration API nicht verfügbar. Protect Version 5.3+ erforderlich“ [4]). In diesem Fall ist ein UniFi-Protect-Update nötig oder es muss auf reines RTSP umgestellt werden.

- **Lokaler Benutzer vs. Ubiquiti-Cloud:** Das Livestream-Modul `ProtectLivestreamManager` loggt sich mit Benutzer/Passwort ein [5]. Wichtig: Verwenden Sie *lokale* UniFi-OS-Zugangsdaten (unter „Lokaler Zugriff“ im UniFi OS aktiviert). Cloud-Only-Anmeldedaten funktionieren hier evtl. nicht. Im Log sollte nach Aufruf eines Streams **„[ProtectLivestream] Login successful“** erscheinen [6]. Falls **„Login failed“** [7] geloggt wird, sind die Credentials falsch oder haben keine ausreichenden Rechte.

2. Handling von Authentifizierung, Tokens und Session

Nach erfolgreichem Login holt das Livestream-Modul einen **Bootstrap** mit der Kameraliste [8]. Prüfen Sie, ob **Kameras gefunden** wurden (Log: „Found N cameras“ [9]). Wenn hier 0 steht, könnte ein Berechtigungs- oder API-Problem vorliegen. Ein weiterer Punkt: In älteren Protect-Versionen musste RTSP pro Kamera manuell aktiviert werden. Das verwendete NPM-Paket `unifi-protect` abstrahiert dies zwar; dennoch lohnt es sich zu kontrollieren, ob im Protect-UI für die genutzte Qualitätsstufe (High/Med/Low) **„RTSP Enabled“** ist. Der Code verwendet standardmäßig `rtspChannel: 1` (Medium-Stream) [10] – falls dieser in Protect deaktiviert ist, könnte `livestream.start` fehlschlagen. In den Logs würde dann **„Failed to start stream for: [Kamera]“** erscheinen [11] und der WebSocket würde mit *Stream konnte nicht gestartet werden* geschlossen werden [12]. Da letzteres anscheinend nicht sichtbar war, hat der Start zumindest **kein sofortiges Fehlerereignis** geliefert.

3. WebSocket-Handling und Stream-Übertragung

Die Kommunikation Browser ↔ Backend für den Livestream läuft über einen WebSocket auf `/ws/livestream/{cameraId}`. Aus dem Frontend-Code ist ersichtlich, dass die Meldung **„Stream disconnected“** genau dann gesetzt wird, wenn die WebSocket-Verbindung *geschlossen* wird, ohne dass bereits ein Fehlerzustand vorlag [13]. Das deutet darauf hin, dass der Socket vom Server (oder durch Netzzeitout) beendet wurde, **ohne** zuvor eine Fehlermeldung zu schicken. Mögliche Ursachen:

- **Backend schließt die Verbindung**: Im aktuellen Code passiert das nur bei bekannten Fehlern *vor* oder *beim Start* des Streams. Z.B. wird bei misslungener Initialisierung `clientWs.close(4503, 'UniFi Protect not configured')` gesendet [2], oder wenn `startStream` `false` liefert, `close(4500, 'Stream start failed')` [14]. In **diesen** Fällen hätte der Browser aber eine spezifische Fehlermeldung im JSON (`msg.type === 'error'`) erhalten und angezeigt (z.B. „Stream konnte nicht gestartet werden“). Da das hier nicht geschah, liegt der Fehler zeitlich **nach dem Start**. Der Server sendet ab dann Videodaten über den Socket – oder sollte es zumindest. Ein Abbruch ohne Nachricht kann durch einen **internen Abbruch der Stream-Session** entstehen.

- **Livestream-Session bricht ab**: Das `ProtectLivestream`-Objekt feuert Events `'close'` und `'error'`, die vom Manager abgefangen werden [15]. Der Code versucht dann einen **Reconnect**: bis zu 3 Versuche mit 2s,4s,6s Abstand [16] [17]. Wichtig: Diese Logik informiert den Client **nicht** direkt – sie hält den WebSocket offen in der Annahme, den Stream wiederherstellen zu können. Sollte jedoch nach 3 Versuchen keine Verbindung zustande kommen, gibt der Manager auf, ohne dem Frontend Bescheid zu geben (“Max reconnect attempts reached” [18]). In diesem Moment existiert kein `LivestreamSession` mehr für die Kamera, aber der WebSocket blieb offen. Möglicherweise kappt erst ein **Timeout** auf Proxy-/Browser-Seite schließlich die Verbindung, was beim Client dann als „disconnected“ erscheint. Mit anderen Worten: Der Code behandelt dauerhafte Stream-Abbrüche (z.B. Kamera offline, Netzwerkprobleme) nicht sauber, da kein Fehler zurück an den Client geleitet wird. Hier könnte man ansetzen und im Falle abgebrochener Reconnects eine `error`-Message an den WebSocket senden, um dem Nutzer z.B. „Stream beendet“ anzuzeigen.

Schauen Sie deshalb in die **Backend-Logs**, z.B. in der Docker-Ausgabe oder Konsole: Sind dort Meldungen wie `Stream closed for [Kameraname]` oder `Stream error for [Kameraname]` zu sehen? Folgt darauf `Reconnecting stream for ... attempt 1/2/3` [19]? Falls ja, deutet das auf Verbindungsabbrüche hin. Ursachen könnten ein instabiles Netzwerk zum NVR oder Inkompatibilitäten (siehe nächster Punkt) sein.

4. API-Inkompatibilitäten & Versionsthemen

Das verwendete `unifi-protect`-NPM-Paket (v4.27.6) implementiert die inoffizielle Protect-API. Wenn UniFi Protect kürzlich ein Update erhalten hat, könnten **Änderungen im Streaming-Protokoll** Probleme machen. Beispielsweise nutzen neuere Protect-Versionen für „Enhanced Video“ H.265-Streams, die im Browser nicht decodiert werden können. Zwar würde der Client beim Erhalt eines unbekannten Codecs eine Fehlermeldung werfen („Codec not supported“) – was hier offenbar nicht geschah – dennoch sollten Sie sicherstellen, dass **alle Kameras H.264 ausgeben** (in Protect-Web-UI unter Kamera->Einstellungen->Video den Codec auf *Standard/H.264* setzen). Andernfalls startet der Stream ggf. nur scheinbar, und bricht intern ab. Ähnlich könnten *Timing*-Probleme auftreten, wenn der Codec-Info-Frame später als erwartet kommt. Der Client versucht zunächst mit einem Standardcodec zu initialisieren, falls die *codec*-Message nicht sofort eintrifft [20] [21]. In Ihren Logs sollten Sie sehen, ob

`[LivestreamPlayer] Received codec info: ...` gemeldet wurde. Falls **nicht**, wurde eventuell gar keine `codec`-Info vom Server gesendet, was auf ein frühes Versagen der Streaming-Session hindeutet.

5. Frontend-Player und sonstige Hinweise

Der React-Player (`WebRTCPlayer.tsx`) ist grundsätzlich korrekt aufgebaut: Er nutzt die **Media Source Extensions (MSE)**, um H.264-Videosegmente vom WebSocket einzuspeisen. Ein paar Punkte sollten Sie überprüfen:

- **Browser-Kompatibilität:** Testen Sie den Livestream in verschiedenen Browsern. Chrome und Firefox unterstützen MSE/H.264 gut. Safari (insbesondere iOS) hatte historisch Einschränkungen bei MSE. „Keine Fehler in der Konsole“ lässt vermuten, dass zumindest keine generelle API-Untersützung fehlt, aber ein Wechsel des Browsers kann Aufschluss geben.

- **Parallelstreams:** Wenn mehrere Kameras gleichzeitig im Dashboard angezeigt werden, öffnen die Backend-Module parallel Streams (der Manager verwaltet pro Kamera eine Session [22] [23]). UniFi Protect könnte begrenzen, wie viele gleichzeitige Livestreams ein einzelner User oder NVR verarbeiten kann. Versuchen Sie testweise, nur **eine Kamera** in der UniFi-Einstellungsliste zu belassen, um zu sehen, ob dann zumindest ein Stream stabil läuft. Wenn ja, deutet das auf Ressourcen- oder Limit-Probleme hin. In diesem Fall könnten Sie die Qualität (`rtspChannel`) reduzieren (0 = High, 1 = Medium, 2 = Low) oder die Zahl paralleler Streams beschränken.

- **HTTPS/WSS-Konfiguration:** Stellen Sie sicher, dass das Dashboard unter derselben Origin auf den WebSocket zugreift. Im Code wird `window.location.host` genutzt [24] – bei Docker wird das via `gateway.js` wahrscheinlich auf Port 80 gebündelt. Verbindungsabbrüche könnten auftreten, wenn z.B. ein Proxy (Nginx, Cloudflare) eine Idle-Timeout hat. Sollte Ihr Setup hinter einem Proxy laufen, prüfen Sie die WebSocket-Timeouteinstellungen dort. Gegebenenfalls kann ein regelmäßiges WebSocket-Ping implementiert werden (das `ws`-Modul unterstützt Ping/Pong).

6. Debug-Tipps und Verbesserungsvorschläge

- **Backend-Logging aktivieren:** Erhöhen Sie die Sichtbarkeit, indem Sie das Logging im Livestream-Manager erweitern. Etwa können Sie nach **jedem** Reconnect-Versuch eine Ausgabe machen, oder im `tryReconnect` nach endgültigem Abbruch ein `console.error` mitteilten. Auch ein Log im WebSocket-Proxy nach einer gewissen Zeit ohne Daten könnte helfen, das „lautlose“ Ende zu erkennen.

- **Client-Feedback verbessern:** Derzeit erfährt der Benutzer nichts von Reconnect-Versuchen. Eine Verbesserung wäre, dem Client bei **Reconnect-Start** einen Hinweis zu schicken (z.B. ein `type: 'warning', message: 'Verbindung unterbrochen, versuche neu zu verbinden...'`). Falls nach max. Versuchen kein Erfolg: `type: 'error', message: 'Livestream abgebrochen'` und WebSocket schließen. So würde statt „Stream disconnected“ ein klarerer Fehler erscheinen. Dazu müsste der `ProtectLivestreamManager` dem `ws-proxy` Bescheid geben – etwa via Events auf `ProtectLivestreamManager` (diese erbt von EventEmitter, kann also eigene Events emitten).

- **Aktualisierung des `unifi-protect`-Pakets:** Prüfen Sie, ob es eine neuere Version gibt. Das Projekt von hjdhjd wird regelmäßig gepflegt; möglicherweise wurden Streaming-Probleme in neueren Releases behoben. Ein Update könnte Inkompatibilitäten ausräumen.

- **Server-Ressourcen:** Ein hoher Durchsatz mehrerer HD-Streams kann die Server-CPU belasten. Da hier kein Hardware-Decoder genutzt wird (die Videodaten werden direkt durchgeleitet), sollte CPU zwar nicht hoch sein, aber checken Sie im Container die Auslastung, um auszuschließen, dass z.B. Memory-Leaks oder der Buffer (`bufferQueue`) unkontrolliert wachsen. Im Code werden die H.264-Daten stückweise per MSE appended; bei Fehlverhalten könnten im Browser **Media Source-Fehler** geloggt sein (Codec-Buffer voll etc.). Solche Fehler würden in der Konsole als MediaError auftauchen – laut Ihrer Aussage war da aber nichts.

Zusammengefasst: **Wichtigste Ansätze** sind das **Lesen der Backend-Logs** (um den genauen Abbruchgrund zu erkennen) und das Überprüfen der **Zugangsdaten und Protect-Version**. Achten Sie auf die genannten Code-Stellen, die im Fehlerfall ansetzen. Mit diesen Informationen können Sie gezielt nachbessern – sei es durch Konfigurationskorrekturen (Credentials, RTSP-Optionen), Updates oder Code-Erweiterungen, um den „Stream disconnected"-Fall besser zu behandeln.

Quellen:

- Ausschnitte aus `ws-proxy.ts` (WebSocket-Handling) [2] [12]
- Ausschnitte aus `ProtectLivestreamManager` (Start, Events, Reconnect) [5] [15] [16]
- Ausschnitte aus `WebRTCPlayer.tsx` (Client-WebSocket und MSE-Player) [13] [20]

---

[1] [2] [10] [12] [14] ws-proxy.ts
https://github.com/hi-its-lukas/HASS-Dashboard/blob/1d146020a93fbe7963253ef4e6b8f761bfcfd967/server/ws-proxy.ts

[3] [4] protect-client.ts
https://github.com/hi-its-lukas/HASS-Dashboard/blob/1d146020a93fbe7963253ef4e6b8f761bfcfd967/lib/unifi/protect-client.ts

[5] [6] [7] [8] [9] [11] [15] [16] [17] [18] [19] [22] [23] protect-livestream.ts
https://github.com/hi-its-lukas/HASS-Dashboard/blob/10e0007ac41e214cb5cb467b98bd7985b610f0d3/lib/streaming/protect-livestream.ts

[13] [20] [21] [24] WebRTCPlayer.tsx
https://github.com/hi-its-lukas/HASS-Dashboard/blob/1d146020a93fbe7963253ef4e6b8f761bfcfd967/components/streaming/WebRTCPlayer.tsx