Perfect Streamer

Release 1.13.2.444

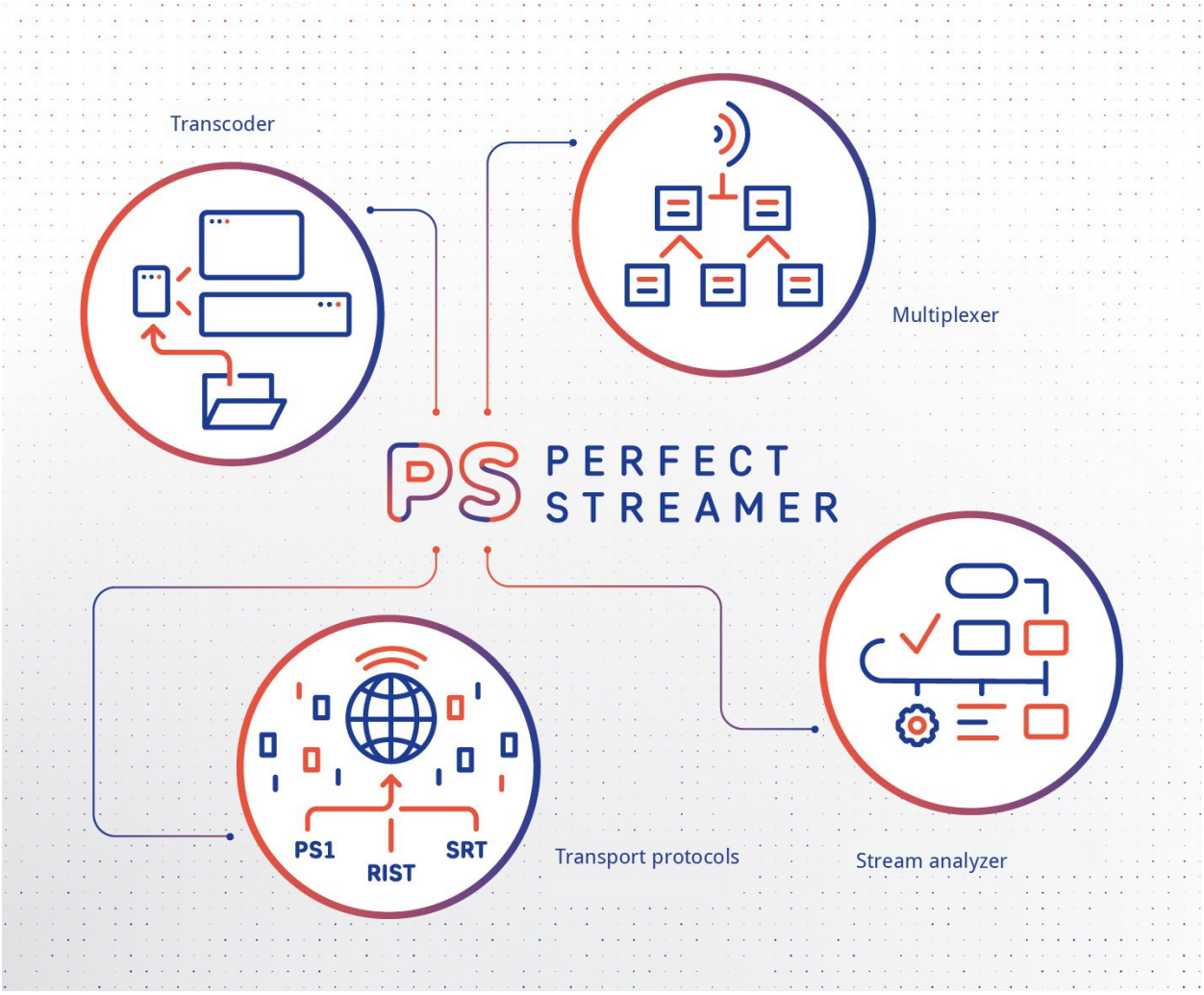
Perfect Soft

01.06.2026

Inhaltsverzeichnis

KAPITEL 1

Zweck



Das Programm **Perfect Streamer** ist für die Übertragung von Streams im Format MPEG-TS über das öffentliche Internet mit Paketverlusten und Verzögerungen vorgesehen. Verwendet wird das eigene UDP-basierte Protokoll Perfect Stream (**PS1**). Zusätzlich werden die Standardprotokolle **Pro-MPEG / RTP+FEC** (auch bekannt als SMPTE 2022-1/2) und **SRT** unterstützt, sodass Kanäle sowohl zwischen **Perfect Streamer**-Instanzen als auch mit anderen Programmen oder Geräten aufgebaut werden können, die diese Protokolle unterstützen.

- Das Protokoll **PS1** arbeitet nach Automatic Repeat reQuest (ARQ). Es ist ressourcenschonend und überträgt Streams mit hoher Bitrate.
- **Pro-MPEG / RTP+FEC** (Pro-MPEG COP3, auch bekannt als SMPTE 2022-1/2) — RTP mit Vorwärtsfehlerkorrektur (FEC). Beschrieben in der IEEE-Norm (https://ieeexplore.ieee.org/document/6738329) und von zahlreichen Geräten unterstützt. Vorteil: niedrige Latenz. Nachteil: hoher zusätzlicher Datenverkehr; bei hohen Paketverlusten arbeitet das Protokoll schlecht.
- **SRT** — offenes Protokoll von Haivision auf UDT-Basis, weit verbreitet und mit guter Paketverlust-Kompensation.
- **RIST** — offenes Protokoll auf Basis von RTP/RTCP. Arbeitet nach Automatic Repeat reQuest (ARQ) ohne ACK, nur NACK; das gewährleistet hohe Effizienz.

Unterstützt werden Standard-Transportprotokolle HLS, HLS SSL, UDP, RTP, HTTP usw.

Transkoder mit Unterstützung für Nvidia Encoder und Software CPU.

Das Programm bietet Funktionen zur Stream-Redundanz, einen EPG-Server, einen Multiplexer und Demultiplexer, einen EIT-Generator, die Arbeit mit DVB-Karten, einen professionellen Analysator (TR 101 290 und erweiterter), Diagramme, AES-Verschlüsselung, Mosaik, die Modifikation von Metadaten in MPEG-TS u. a.

Integration mit Monitoring-Systemen wie Zabbix, Grafana usw. wird unterstützt.

KAPITEL 2

Einrichtung

2.1 Systemanforderungen

- **Perfect Streamer** läuft unter Linux. Hauptanforderung: GLIBC >= 2.17.
- Die vom Streamer-Dienst genutzten Netzwerkschnittstellen müssen statisch konfiguriert sein.
- Die Installer-Skripte nutzen das Werkzeug **sudo** — es muss daher im System installiert sein.

Für die Red-Hat- und Debian-Familien stehen Installationspakete und Repositories bereit. Unterstützt werden RHEL 7 und höher (CentOS usw.) sowie RPM-kompatible Distributionen — zum Beispiel openSUSE Leap (siehe die Anmerkung am Ende des RHEL-Abschnitts). Debian-basierte Systeme (Ubuntu usw.) müssen über den systemd-Dienst verfügen.

Hardware-Richtwerte: 1 Kern @ 2,4 GHz und 1 GB RAM pro 200 Mbit Datenverkehr. Die Schätzung ist näherungsweise und hängt von den eingesetzten Protokollen und Einstellungen ab.

Eigenschaften der kostenlosen Demo-Version:

- Auf 10 Streams begrenzt
- Transkoder läuft nur im CPU-Software-Modus
- Ohne Einschränkungen im Funktionsumfang
- Ohne Zeitbeschränkung

Die Distributionen der Voll- und Demo-Version unterscheiden sich. Verwenden Sie für die Installation der Demo-Version das entsprechende Paket pstreamer-demo. Beim Wechsel auf die Vollversion muss zuerst die Demo-Version deinstalliert und dann die Vollversion installiert werden. Die Konfigurationsdatei der Demo-Version ist mit der Vollversion des Pakets kompatibel, aber die Konfigurationsdatei der vollen pstreamer-Version kann mit pstreamer-demo inkompatibel sein — der Dienst startet möglicherweise nicht und die manuelle Entfernung der Datei pss.json kann erforderlich sein.

2.2 Installation auf RHEL-Systemen

Repository für RHEL 7 installieren:

```
$ sudo yum install yum-utils
$ sudo yum-config-manager --add-repo=http://repo.pstreamer.tv/pub/pstreamer/pstreamer.
↪repo
```

Oder für RHEL 8+:

```
$ sudo yum config-manager --add-repo=http://repo.pstreamer.tv/pub/pstreamer/pstreamer.
↪repo
```

Paket installieren:

```
$ sudo yum -y install pstreamer

or

$ sudo yum -y install pstreamer-demo
```

Paket aktualisieren:

```
$ sudo yum -y update pstreamer

or

$ sudo yum -y update pstreamer-demo
```

Alle Pakete entfernen:

```
$ sudo yum -y remove pstreamer aksusbd

or

$ sudo yum -y remove pstreamer-demo
```

Bemerkung: **openSUSE** (Leap, die neueste stabile Version) ist ein RPM-basiertes System. Es wird dasselbe Repository wie für RHEL verwendet, die Vorgänge werden jedoch mit dem Paketmanager **zypper** ausgeführt:

```
$ sudo zypper addrepo http://repo.pstreamer.tv/pub/pstreamer/pstreamer.repo
$ sudo zypper --gpg-auto-import-keys refresh
$ sudo zypper install pstreamer       # or pstreamer-demo
```

Aktualisierung — `sudo zypper update pstreamer`, Entfernung — `sudo zypper remove pstreamer aksusbd`.

2.3 Installation auf Debian-Systemen

Repository installieren:

```
$ sudo wget http://repo.pstreamer.tv/pub/deb/dists/pstreamer/pstreamer.list -O /etc/
↪apt/sources.list.d/pstreamer.list
$ sudo apt-get update
```

Paket installieren:

```
$ sudo apt-get install pstreamer

or

$ sudo apt-get install pstreamer-demo
```

Paket aktualisieren:

```
$ sudo apt install pstreamer

or

$ sudo apt install pstreamer-demo
```

Alle Pakete entfernen:

```
$ sudo apt-get remove pstreamer aksusbd

or

$ sudo apt-get remove pstreamer-demo
```

2.4 Dateien und Dienste

/usr/local/bin/pss
: Ausführbare Datei.

/opt/pss/config/pss.properties
: Globale Einstellungen, Logs, Pfade usw. Nach Änderungen den Dienst neu starten.

/opt/pss/config/pss.json
: Hauptkonfigurationsdatei. Wird automatisch erstellt und aktualisiert. Beim Start versucht der Dienst, genau diese Datei zu laden.

/opt/pss/config/pss_back.json
: Sicherungskopie der vorherigen funktionierenden Konfiguration. Wird automatisch beim Auslösen der Aktion **Einstellungen wiederherstellen** in der Weboberfläche gespeichert und als erste Fallback-Variante verwendet, wenn die Haupt-*pss.json* nicht verarbeitet werden kann.

/opt/pss/config/pss_default.json
: Standardkonfigurationsdatei. Wird mit dem Paket ausgeliefert und als letzte Fallback-Variante verwendet, wenn weder die Haupt-*pss.json* noch *pss_back.json* geladen werden kann.

/opt/pss/config/pss_back.json
: Archiv beschädigter *pss.json*-Dateien. Wenn die Hauptkonfigurationsdatei beim Start nicht verarbeitet werden kann, wird sie unter einem Namen der Form *pss_YYYYMMDD_HHMMSS.json* hierher verschoben. Das Verzeichnis wird automatisch angelegt. Näheres siehe Abschnitt *Verhalten beim Start und bei Konfigurationsfehlern*.

/opt/pss/data
: Datenverzeichnis. Wird automatisch erstellt und aktualisiert. Kann in der globalen Konfigurationsdatei geändert werden.

/usr/lib/systemd/system/pss.service
: systemd-Service-Datei.

/var/log/pss
: Log-Verzeichnis. Kann in der globalen Konfigurationsdatei geändert werden.

Dienstname **pss**. Wird unter dem Benutzer **pss** ausgeführt.

Während der Installation wird das Begleitpaket **aksusbd** des Schutzsystems installiert; es enthält die Dienste **hasplmd** und **aksusbd**.

2.5 Nach der Installation

Nach der Installation von **Perfect Streamer** *Aktivierung und Erstkonfiguration des Dienstes durchführen*.

2.6 Transkoder

Installation der Transkoder für Perfect Streamer.

Verfügbare Pakete:

- pstreamer-tcsw: Transkodierung auf CPU (Software).
- pstreamer-tcnv: Transkodierung auf Nvidia-GPU. Nur für das Paket *pstreamer* (Vollversion mit Schutz).
- pstreamer-tcivpl: Transkodierung auf Intel-GPU. Nur für das Paket *pstreamer* (Vollversion mit Schutz).

Haupterfordernis: GLIBC >= 2.28.

Funktioniert jetzt unter AlmaLinux 8.9.

Der Intel-VPL-Transkoder funktioniert auf AlmaLinux-10-Systemen (RHEL 10).

2.6.1 RHEL 8+

1. pstreamer oder pstreamer-demo installieren.
2. Für Nvidia die Repositories installieren und das System aktualisieren (falls noch nicht geschehen):

```
sudo dnf config-manager --add-repo https://developer.download.nvidia.com/compute/cuda/
↪repos/rhel9/x86_64/cuda-rhel9.repo
sudo dnf clean all
sudo dnf update -y
reboot
```

3. NVidia Encoder.

- CUDA installieren:

```
sudo dnf -y install cuda-toolkit-12-5
```

- Treiber installieren (Variante wählen):

Legacy

```
sudo dnf -y module install nvidia-driver:latest-dkms
```

New

```
sudo dnf -y module install nvidia-driver:open-dkms
```

Nach der Installation den Rechner unbedingt neu starten:

```
reboot
```

Nach dem Neustart die Treiberfunktion prüfen:

```
nvidia-smi
modprobe nvidia
sudo lsmod | grep nvidia
```

oder

```
modprobe nouveau
sudo lsmod | grep nouveau
```

4. Intel VPL Encoder.

- Intel-Treiber und Intel VPL installieren (RHEL 10):

```
dnf install -y intel-gpu-firmware
dnf install -y https://mirrors.rpmfusion.org/free/el/rpmfusion-free-release-$(rpm -E
↪%rhel).noarch.rpm https://mirrors.rpmfusion.org/nonfree/el/rpmfusion-nonfree-
↪release-$(rpm -E %rhel).noarch.rpm
dnf install -y intel-media-driver
dnf install -y intel-vpl-gpu-rt
```

5. Transkoder-Pakete installieren.

- CPU-Software-Methode:

```
sudo dnf install -y pstreamer-tcsw
```

- Nvidia GPU:

```
sudo dnf install -y pstreamer-tcnv
```

- Intel VPL GPU:

```
sudo dnf install -y pstreamer-tcivpl
```

2.6.2 Ubuntu 22/24

1. pstreamer oder pstreamer-demo installieren.
2. NVidia Encoder.

- CUDA Toolkit Version 12.5 installieren:

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-
↪keyring_1.1-1_all.deb
sudo dpkg -i cuda-keyring_1.1-1_all.deb
sudo apt-get update
sudo apt-get -y install cuda-toolkit-12-5
```

- Treiber installieren (Variante wählen):

legacy kernel module flavor:

```
sudo apt-get install -y cuda-drivers
```

oder

open kernel module flavor:

```
sudo apt-get install -y nvidia-driver-555-open
sudo apt-get install -y cuda-drivers-555
```

Nach der Installation den Rechner unbedingt neu starten:

```
reboot
```

Nach dem Neustart die Treiberfunktion prüfen:

```
nvidia-smi
```

Für den Betrieb des Transkoders ist CUDA 12.5 erforderlich. Im System kann bereits eine andere CUDA-Version installiert sein. Auch beim System-Update ist ein Upgrade auf eine neuere CUDA-Version möglich. Das beeinträchtigt den Betrieb nicht — die alten Versionen müssen nicht entfernt werden, da unterschiedliche CUDA-Versionen in separaten Verzeichnissen installiert werden.

3. Transkoder-Pakete installieren.

- CPU-Software-Methode:

```
sudo apt-get install -y pstreamer-tcsw
```

- Nvidia GPU:

```
sudo apt-get install -y pstreamer-tcnv
```

Die Transkoder-Pakete installieren folgende Dateien:

- /usr/local/bin/tcsw — ausführbare Datei des SW(CPU)-Transkoders.
- /usr/local/bin/tcnv — ausführbare Datei des Nvidia-(GPU-)Transkoders.
- /opt/pss/config/pss_tc_sw.properties — Startkonfigurationsdatei des SW-(CPU-)Transkoders.
- /opt/pss/config/pss_tc_nv.properties — Startkonfigurationsdatei des Nvidia-(GPU-)Transkoders.

4. Transkoder-Installation prüfen.

Die Installation der Transkoder-Pakete startet den pss-Dienst neu. Die Installation lässt sich im Bereich About prüfen — Version oder Fehler werden angezeigt.

2.6.3 Weitere auf Debian und RHEL basierende Systeme

Zur Installation des pstreamer-tcnv-Transkoders auf anderen OS als Ubuntu 22/24 und RHEL 8+ verwenden Sie den Konfigurator auf der Nvidia-Website, um passende CUDA- und Treiberversionen zu wählen:

https://developer.nvidia.com/cuda-toolkit-archive

Bei der Auswahl jeder CUDA-Version ist die Information verfügbar, für welche OS-Version sie geeignet ist. Unterstützte Architektur — x86_64. Wenn eine ältere OS-Version benötigt wird, prüfen Sie deren Unterstützung in älteren CUDA-Versionen. Die Hauptanforderung an das OS — Unterstützung von GLIBC >= 2.28.

Der Nvidia-Treiber muss aus dem Repository installiert werden, das auf der Seite der gewählten CUDA-Version für Ihre OS-Version angeboten wird.

Die Unterstützung von Nvidia-Grafikkarten für die Transkoder-Funktionalität pstreamer-tcnv lässt sich auf der Nvidia-Website unter Video Encode and Decode Support Matrix prüfen. Auf dieser Seite stehen die Matrix der unterstützten Videoformate für Decoder und Encoder sowie weitere Eigenschaften zur Verfügung.

2.7 Entfernen der alten CUDA- und Treiber-Version

Bei einer falschen CUDA-/Treiberversion ist ggf. ein Wechsel auf eine andere Version nötig — zuerst die bestehende Installation entfernen.

https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html?highlight=uninstall#removing-cuda-toolkit

2.7.1 CUDA entfernen

RHEL:

```
dnf remove "cuda*" "*cublas*" "*cufft*" "*cufile*" "*curand*" "*cusolver*" "*cusparse*
↪" "*gds-tools*" "*npp*" "*nvjpeg*" "nsight*" "*nvvm*" "*nvptx*"
```

Debian/Ubuntu:

```
apt remove --autoremove --purge "*cuda*" "*cublas*" "*cufft*" "*cufile*" "*curand*"
↪"*cusolver*" "*cusparse*" "*gds-tools*" "*npp*" "*nvjpeg*" "nsight*" "*nvvm*"
↪"*nvptx*"
```

2.7.2 Treiber entfernen

Ubuntu 22/24:

```
apt remove --autoremove --purge -V \
        cuda-compat\* \
        cuda-drivers\*  \
        libnvidia-cfg1\* \
        libnvidia-compute\* \
        libnvidia-decode\* \
        libnvidia-encode\* \
        libnvidia-extra\* \
        libnvidia-fbc1\* \
        libnvidia-gl\* \
        libnvidia-gpucomp\* \
        libnvidia-nscq\* \
        libnvsdm\* \
        libxnvctrl\* \
        nvidia-dkms\* \
        nvidia-driver\* \
        nvidia-fabricmanager\* \
        nvidia-firmware\* \
        nvidia-headless\* \
        nvidia-imex\* \
        nvidia-kernel\* \
        nvidia-modprobe\* \
        nvidia-open\* \
        nvidia-persistenced\* \
        nvidia-settings\* \
        nvidia-xconfig\* \
        xserver-xorg-video-nvidia\*
```

RHEL 9:

```
dnf module remove --all nvidia-driver
dnf module reset nvidia-driver
```

RHEL 10:

```
dnf remove nvidia-driver\*
```

KAPITEL 3

Einrichtung und Aktivierung

3.1 Eigenschaften der kostenlosen Demo-Version

- Auf 10 Streams begrenzt
- Transkoder läuft nur im Software-CPU-Modus
- Ohne Einschränkungen im Funktionsumfang
- Ohne Zeitbeschränkung

3.2 Temporäre Aktivierung und Start

Für die zeitlich begrenzte Version ohne Stream-Anzahl-Limit. Nach der Installation ist der Dienst **pss** inaktiv und benötigt eine Aktivierung. Für die temporäre Aktivierung das Skript ausführen:

```
$ /opt/pss/tools/activate.sh
```

Der Dienst **pss** wird gestartet und für den Autostart eingerichtet. Erfolgreichen Start prüfen:

```
$ systemctl status pss
```

Bei Problemen siehe Logs:

```
$ tail -n 20 /var/log/pss/main.log
$ journalctl -u pss -n 20
```

3.3 Anfangseinstellungen

Im Browser eine Verbindung zur Weboberfläche herstellen:

http://host:8808

Anmeldung: *admin*

Passwort: *admin*

Den Login der Weboberfläche unbedingt ändern — Bereich *Configuration/Administration/Administrators List*.

Bei Bedarf den Port der Weboberfläche im Bereich *Configuration/HTTP Server* ändern; der Dienst wird dabei neu gestartet.

Die Aktivierungsdaten lassen sich über die Weboberfläche im Bereich *Configuration/About* prüfen.

3.4 Permanente Aktivierung

Das Schutzsystem unterstützt Software- (SL) und USB- (HL) Lizenzen. Für die permanente Aktivierung:

1. In der Weboberfläche unter *Configuration/About* die C2V-Daten für die Aktivierungsanfrage abrufen und an den Anbieter senden.
2. Die vom Anbieter erhaltenen V2C-Daten in derselben Sektion der Weboberfläche aus Datei oder Zwischenablage einfügen.
3. Im selben Abschnitt der Weboberfläche die Aktivierungsdaten prüfen.

Damit Perfect Streamer den USB-Schlüssel während der aktiven Testlizenz erkennt, muss der PSS-Dienst neu gestartet werden. Das ist über das Web-Portal möglich: Configuration — Maintenance — Reboot. Alternativ wechselt der Dienst nach Ablauf der Testlizenz selbsttätig auf den dauerhaften USB-Schlüssel.

3.5 Bei Ablauf der Jahreslizenz oder der Trial

Wenn der PSS-Dienst nicht startet, führen Sie folgenden Befehl aus:

```
$ journalctl -u pss -n 20
```

Folgender Fehler bedeutet, dass die Perfect-Streamer-Lizenz abgelaufen ist:

```
LDK Protection System: Feature has expired (H0041)
```

KAPITEL 4

Benutzerdokumentation

4.1 Planung und Datenübertragungsprotokolle

Die Software **Perfect Streamer** dient der Übertragung von MPEG-TS-Streams über das öffentliche Internet mit Paketverlusten und Latenzen auf UDP-Basis.

Für jeden MPEG-TS-Stream (**Stream**) werden ein Sender-Server (**Sender**) und ein oder mehrere Empfänger (**Receiver**) konfiguriert; diese Verbindung wird im Folgenden als **Peer** bezeichnet.

Die Konfiguration von Sender und Empfänger reduziert sich auf das Einrichten einer Stream-Liste und der Einstellungen für **input** und **output** jedes Streams. Mehrere **input** in der Liste sorgen für Quellen-Redundanz. Mehrere **output** in der Liste ermöglichen die gleichzeitige Übertragung von Streams an verschiedene Empfänger.

Beim Sender bezeichnet **input** die Quellen der MPEG-TS-Streams und **output** die Übertragung der Streams an die Empfänger. Bei Empfängern bezeichnet **input** den Empfang der Streams vom Sender.

Es stehen vier **Peer**-Protokolle für die Übertragung von Streams zwischen Sender und Empfänger zur Verfügung:

- Perfect-Stream-Protokoll (PS1).
- SRT.
- Pro-MPEG / RTP+FEC.
- RIST.

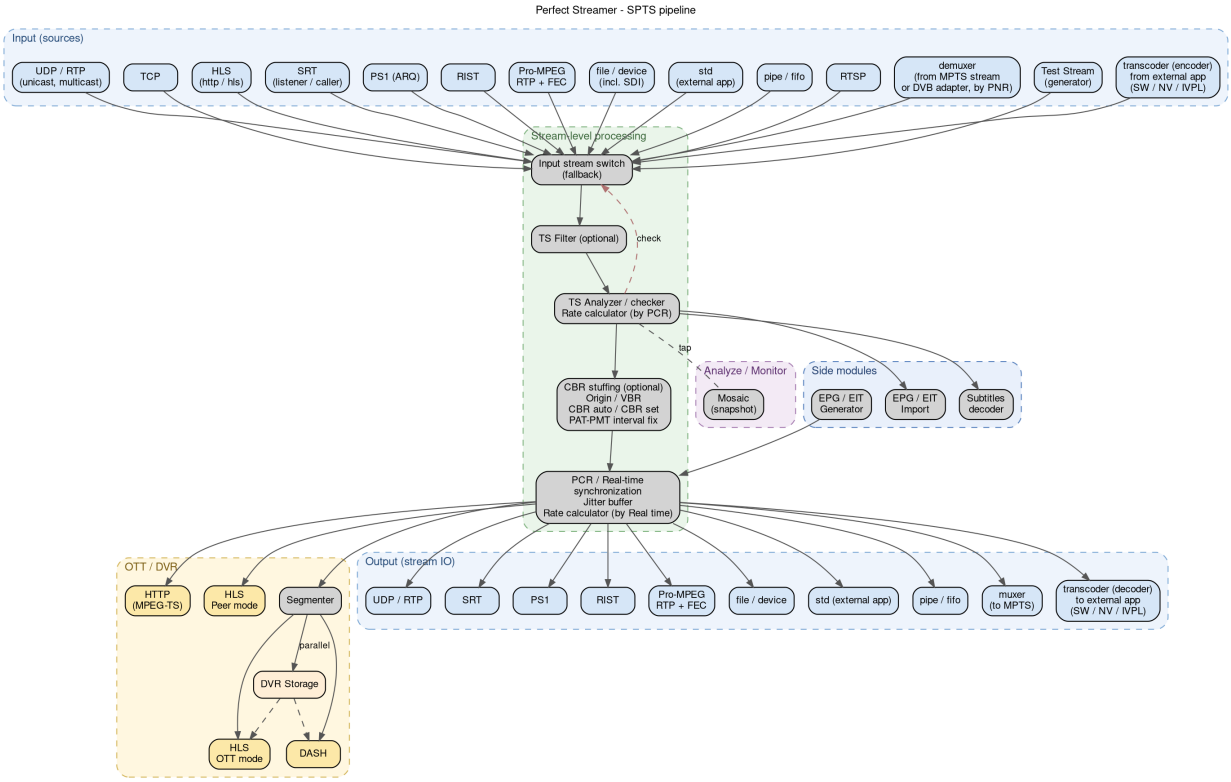


Abb. 1: Gesamtarchitektur des SPTS-Streams: Eingänge, Verarbeitung auf Stream-Ebene (Input-Switch, TS-Filter, Analyzer, Synchronisierer), OTT / DVR und Ausgänge. Details zu den einzelnen Blöcken siehe folgende Abschnitte.

4.1.1 PS1-Protokoll

Das PS1-Protokoll basiert auf Automatic Repeat reQuest (ARQ). Es ist ressourcenschonend und erlaubt die Übertragung von Streams mit hoher Bitrate.

Am Sender — wird im output konfiguriert. Pro Stream ist nur eine Instanz verfügbar. In **Peer** müssen die Logins der Empfänger eingetragen werden. Es wird die UDP listen port festgelegt — sie muss für jeden Stream eindeutig sein.

Am Empfänger — wird im Input konfiguriert. Host und Port des Senders sowie Login und Passwort werden angegeben.

Stream-Verschlüsselung (Crypto protection) ist verfügbar, AES-128 wird verwendet. Zum Aktivieren auf beiden Seiten **Crypt Passphrase** als gemeinsamen Schlüssel eintragen.

Während des Betriebs sendet der Empfänger (Client) seine Statistiken an den Sender (Server). Diese sind im Bereich *Peers* nach Auswahl des Clients sichtbar.

Stream-Latenz und Verlustkompensation hängen von den Empfängereinstellungen ab:

Round Trip Time — RTT in ms, Standard 300. Geschätzte Kanal-Latenz (Ping). Nach Stream-Start ist der reale RTT in der Statistik (PS1 recovery delay) sichtbar.

Client Latency (RTT multiplexor) — RTT-Multiplikator (Standard 10), der die Stream-Latenz im Sender-Puffer bestimmt. Bei Standard ergibt sich also eine Pufferverzögerung von 3000 ms.

Senderseitig gibt es die Latenz-Einstellung (Pufferlänge) **Latency (ms)**. Sie muss größer als die clientseitigen Latenzen sein.

Die Fähigkeit des Protokolls, Verluste zu kompensieren, hängt von der Anzahl der Retransmissions-Anfragen ab und ist von **Client Latency (RTT multiplexor)** bestimmt. Große Verluste führen zu zusätzlichem Netzwerkverkehr. Zur Verringerung der Latenz sollten diese Parameter feiner eingestellt werden.

Korrekte Protokollarbeit zeigt sich in der Client-Statistik. Bei **PS1 recovery**: Not found → Sender-Buffer vergrößern; Duplicates → RTT erhöhen.

Beim Umschalten des Stream-Eingangs (Wechsel der Quelle) am Sender kann die Sendewarteschlange kurzzeitig anwachsen. PS1 bewältigt dies reibungslos: Die ältesten Pakete werden stillschweigend verworfen, während Nummerierung (*seqID*) und Zeitstempel bei den Empfängern durchgängig bleiben — die Empfänger holen die Verluste über den regulären Wiederholungsmechanismus (retr) auf, ohne die Verbindung neu zu initialisieren. Die Anzahl der auf diese Weise verworfenen Pakete ist in der erweiterten Statistik des PS1-Ausgangs sichtbar.

Since the connection is initiated from the side of the receiver, the transmitter requires authentication, the receivers are registered in the peers section. Login and password required.

4.1.2 SRT-Protokoll

Offenes Protokoll von Haivision. Basiert auf UDT, ist weit verbreitet und bietet gute Eigenschaften zur Kompensation von Paketverlusten.

Anwendungsfälle:

- Peer zwischen **Perfect Streamer**-Instanzen. **Am Sender** — wird im Output konfiguriert, Listen-Modus (Standard). Für einen Stream kann nur ein solcher Output gesetzt werden. In diesem Modus können mehrere Empfänger verbunden werden. Zur Autorisierung müssen in **Peer** Logins für die Empfänger eingetragen werden. **Am Empfänger** — wird im Input konfiguriert. Host und Port des Senders werden angegeben, ebenso Login und Passwort. Zur Übergabe von Login und Passwort verwendet der SRT-Streamer die streamid im Format `login|password`.
- Peer zwischen **Perfect Streamer** und beliebigen SRT-Streamern von Drittanbietern. **Am Sender** lässt sich der SRT-Client-Modus einrichten, indem listen deaktiviert wird. Die SRT-streamid wird bei Bedarf im Feld login eingetragen. Für den listen-Modus ist die Autorisierung per IP-Adresse verfügbar — sie wird im Feld login bei **Peer** eingetragen. **Am Empfänger** lässt sich der listen-Modus aktivieren, die SRT-streamid im Feld login angeben sowie der Host festlegen, von dem der Empfang erlaubt ist.

Betrieb im Listener-Modus: Empfang und Weitergabe eines TV-Streams mit Angabe des Empfangsports.

Die Ports im Listen-Modus müssen eindeutig sein.

Stream-Verschlüsselung (Crypto protection) ist verfügbar, AES-128 wird verwendet. Zum Aktivieren auf beiden Seiten **Crypt Passphrase** als gemeinsamen Schlüssel eintragen.

Verwendet der Sender den listen-Modus (Standard), wird die Verbindung vom Empfänger initiiert; der Sender erfordert Authentifizierung, und die Empfänger werden in peer registriert. Login und Passwort sind erforderlich.

Die SRT-Protokolloptionen entsprechen der Beschreibung unter API-socket-options.md

Reorder (SRTO_LOSSMAXTTL) — Der Wert, bis zu dem die Umordnungstoleranz ansteigen kann. Die Umordnungstoleranz ist die Anzahl der Pakete, die auf eine erkannte „Lücke“ in den Sequenznummern der eingehenden Pakete folgen müssen, bevor ein Verlustbericht gesendet wird (in der Hoffnung, dass die Lücke durch Paketumordnung und nicht durch Verlust verursacht wird). Der Wert der Umordnungstoleranz beginnt bei 0 und erhöht sich, wenn eine Paketumordnung erkannt wird. Dies geschieht, wenn ein „verspätetes“ Paket mit einer höheren Sequenznummer als das zuletzt empfangene, aber ohne Wiederholungs-Flag empfangen wird. Bei einer solchen Erkennung wird die Umordnungstoleranz auf den Wert des Intervalls zwischen der letzten Nummer und der Sequenznummer dieses Pakets gesetzt, jedoch nicht höher als der durch den Parameter SRTO_LOSSMAXTTL festgelegte Wert. Standardmäßig beträgt dieser Wert 0, was bedeutet, dass dieser Mechanismus deaktiviert ist. SRTO_LOSSMAXTTL

Overhead (SRTO_OHEADBW, %) — Overhead zur Bandbreitenwiederherstellung über die Eingangsrate hinaus (siehe SRTO_INPUTBW), in Prozent der Eingangsrate. Wirksam nur, wenn SRTO_MAXBW auf 0 gesetzt ist. Sender: vom Benutzer konfigurierbar; Standard: 25%.

Empfehlungen: Der Overhead dient dazu, zusätzliche Bandbreite bereitzustellen, falls ein Paket einen Teil der Bandbreite verbraucht hat, dann aber verloren ging und neu übertragen werden muss. Die effektive maximale Bandbreite sollte daher hinreichend höher als der Bitrate Ihres Streams sein, um Platz für Retransmissionen zu lassen, aber begrenzt, damit retransmittierte Pakete bei Verlust großer Paketgruppen nicht zu einem starken Anstieg der Bandbreitenauslastung führen. Setzen Sie keinen zu niedrigen Wert und vermeiden Sie 0,

wenn SRTO_INPUTBW auf 0 gesetzt ist (automatisch). Andernfalls wird Ihr Stream bei jedem Anstieg der Paketverluste rasch unterbrochen. SRTO_OHEADBW

Max Band (SRTO_MAXBW, bps) — Diese Option ist nur wirksam, wenn SRTO_MAXBW gleich 0 ist (relativ). Sie steuert die maximale Bandbreite zusammen mit SRTO_OHEADBW nach der Formel: MAXBW = INPUTBW * (100 + OHEADBW) / 100. Ist diese Option auf 0 gesetzt (automatisch), wird der tatsächliche INPUTBW-Wert anhand der Rate des Eingangsstreams (in Fällen, in denen die Anwendung srt_send* aufruft) während der Übertragung geschätzt. Der Mindestwert der Schätzung ist durch SRTO_MININPUTBW begrenzt, d. h. INPUTBW = MAX(INPUTBW_ESTIMATE; MININPUTBW).

Empfehlungen: setzen Sie für diesen Parameter den erwarteten Bitrate Ihrer Übertragung und behalten Sie den Standardwert 25% für SRTO_OHEADBW bei. SRTO_INPUTBW

Timeout (SRTO_CONNTIMEO, ms) — Wert des Verbindungs-Timeouts in Millisekunden. Es ist die Zeit, während der das verbindende Objekt versucht, eine Verbindung herzustellen, und auf eine Antwort des entfernten Endpunkts wartet, bevor die Verbindung mit Fehlercode abgebrochen wird. SRTO_CONNTIMEO

4.1.3 Pro-MPEG / RTP+FEC Protokoll (COP3 / SMPTE 2022-1/2)

Lieferung von MPEG-TS über RTP mit vorwärts gerichteter Fehlerkorrektur (FEC, Forward Error Correction). Dasselbe Protokoll erscheint in Literatur und Geräten unter unterschiedlichen Namen:

- **Pro-MPEG / Pro-MPEG COP3** — Code of Practice #3 des Pro-MPEG-Forums, beschrieben im IEEE-Standard (https://ieeexplore.ieee.org/document/6738329);
- **RTP + FEC** — funktionaler Name (RTP-Stream plus FEC-Kanäle);
- **SMPTE 2022-1** — Column FEC (dasselbe Schema, als SMPTE-Standard veröffentlicht);
- **SMPTE 2022-2** — Row + Column FEC (zweidimensionale Matrix, in PSS implementiert).

Vorteil: niedrige Latenz. Nachteil: hoher zusätzlicher Datenverkehr (Overhead); zudem arbeitet es bei großen Paketverlusten (über 0,2 %) schlecht.

Dieses Protokoll basiert auf RTP mit zwei zusätzlichen FEC-Kanälen (Fehlerkorrekturcode). Die beiden FEC-Kanäle nutzen die Ports port+2 und port+4 — das ist beim Hinzufügen mehrerer Streams auf einen Host oder eine Multicast-Gruppe zu beachten.

Beim Sender werden RTP-Pakete in einer Matrix mit **Cols** Spalten und **Rows** Zeilen gruppiert. Beispiel für cols=8 und rows=4 (Standard):

RTP01	RTP02	RTP03	RTP04	RTP05	RTP06	RTP07	RTP08	R1
RTP11	RTP12	RTP13	RTP14	RTP15	RTP16	RTP17	RTP18	R2
RTP21	RTP22	RTP23	RTP24	RTP25	RTP26	RTP27	RTP28	R3
RTP31	RTP32	RTP33	RTP34	RTP35	RTP36	RTP37	RTP38	R4
C1	C2	C3	C4	C5	C6	C7	C8	

Die Pakete Rx und Cx bilden FEC-Daten zeilen- und spaltenweise. Je kleiner die Matrix, desto besser die Korrekturfähigkeit, aber desto höher der zusätzliche Verkehr. In diesem Beispiel kommen auf 32 RTP-Pakete des Streams 12 FEC-Pakete.

Stream-Verschlüsselung (Crypto protection) ist mit AES-128 verfügbar, jedoch nicht im Standard enthalten — die Kompatibilität mit Drittsoftware oder -hardware ist daher nicht garantiert.

Es gibt nicht-standardisierte Protokollerweiterungen:

Multiplexing — Multiplexen von RTP-Kanälen über einen einzigen UDP-Port. Kann die Netzwerkkonfiguration vereinfachen.

4.1.4 RIST-Protokoll

Neues offenes Protokoll auf Basis von RTP/RTCP. Arbeitet nach Automatic Repeat reQuest (ARQ) ohne ACK, nur NACK — das garantiert hohe Effizienz.

Verwendet Unicast und Multicast.

Es sind die Profile Simple und Main implementiert. Simple nutzt zwei aufeinanderfolgende UDP-Ports — der konfigurierte Port muss gerade sein. Main verwendet einen einzigen RTP-Port mit Datenmultiplexing.

On transmitter — wird im output konfiguriert. Für Unicast werden Adresse und Port des Empfängers festgelegt. Für Multicast muss die Netzwerkschnittstelle angegeben werden, über die die Datenübertragung erfolgt. Für Multicast lässt sich außerdem eine Autorisierung der Empfänger über die IP-Adresse einrichten, indem Logins in **Peer** eingetragen werden.

Am Empfänger — in input konfigurieren. Für Unicast werden Empfangsport (listen) und zwingend die Netzwerkschnittstelle festgelegt. Für Multicast wird nur die Multicast-Gruppe samt Port angegeben.

RIST unterstützt mehrere separate Peers (Adressen). Mit einem Gewicht > 1 lässt sich Lastverteilung zwischen den Peers entsprechend dem Gewicht aktivieren.

Verwendet der Sender Multicast, kann es viele Empfänger geben. In diesem Fall ist eine Authentifizierung der Empfänger anhand der IP-Adresse möglich. Aktivieren Sie dazu die Authentifizierung in den Sendereinstellungen (standardmäßig deaktiviert) und fügen Sie den Client zur Peer-Liste hinzu; tragen Sie im Login-Feld die IP-Adresse ein.

4.1.5 Andere Protokolle

Neben den **peer**-Protokollen stehen für Empfang und Senden von Streams weitere zur Verfügung:

Protocol	Input	Output
UDP	Yes	Yes
RTP	Yes	Yes
TCP	Yes	No
HLS	Yes	Yes
RTSP	Yes	No
pipe	Yes	Yes

UDP (Unicast oder Multicast) — Empfangen und Senden von MPEG-TS im UDP-Paket, bis zu 7 TS-Pakete pro UDP-Paket.

RTP (Unicast oder Multicast) — RFC-basiertes Standardprotokoll. Reordering-Wiederherstellung wird unterstützt.

TCP — Empfang von MPEG-TS über eine TCP-Verbindung im TCP-Client-Modus.

HLS — Empfang und Auslieferung von MPEG-TS über HTTP oder das Apple-Standard-HLS-Protokoll. Beim Empfang wird aus einer adaptiven Playlist die Variante mit der höchsten

Bitrate ausgewählt. **HLS input** steht nur für **SPTS**-Streams zur Verfügung; für MPTS UDP / RTP / TCP / file oder einen DVB-Adapter verwenden.

RTSP — Empfang eines Videostreams aus RTSP-Quellen (IP-Kameras, RTSP-Server) auf Basis von FFmpeg avformat. PSS öffnet eine RTSP-Sitzung (`DESCRIBE` → `SETUP` → `PLAY`), liest die ES-Pakete, remuxt sie in einen internen MPEG-TS und speist die Stream-Pipeline. Der Annex-B-BSF (`h264_mp4toannexb` / `hevc_mp4toannexb`) wird automatisch eingebunden. Enthält die Quelle keine Audiospuren, wird eine stille MPEG-1-Layer-2-Spur (48 kHz, Stereo, 128 kbps) mit am Video gekoppeltem PTS hinzugefügt — dies ist für die Kompatibilität mit der Stream-Pipeline erforderlich, die mindestens eine Audiospur voraussetzt.

Einstellungen des **RTSP**-Inputs:

- **URI** — die vollständige URL der RTSP-Sitzung, z. B. `rtsp://cam.local/play1.sdp`.
- **Login**, **Password** — RTSP-Anmeldedaten, falls Basic-/Digest-Authentifizierung erforderlich ist.
- **User-Agent** — benutzerdefinierter User-Agent (optional).
- **Cookies** — `Cookie`-Header (optional, für Server mit Cookie-basierter Authentifizierung).
- **Transport** — RTP-Transport innerhalb von RTSP:
  - `UDP` — RTP über UDP (geringe Latenz, paketverlustanfällig);
  - `TCP` (Standard) — interleaved RTP innerhalb der RTSP-TCP-Sitzung; durchquert NAT und Firewalls;
  - `HTTP` — RTSP-over-HTTP-Tunnel zur Durchleitung durch reine HTTP-Proxys.
- **Timeout** — Öffnungs- und Lese-Timeout in Sekunden. Standardwert: 10.
- **Trace** — ausführliches Protokoll für Diagnosezwecke.

Der RTSP-Input läuft im selben Prozess wie PSS und benötigt keine externe Binärdatei. Der Workaround über **std** + ffmpeg / gstreamer (siehe FAQ) bleibt verfügbar und ist nützlich für Protokolle, die der eingebaute RTSP nicht abdeckt (z. B. RTMP).

Der RTSP-Input ist eine Single-Program-Quelle und steht daher **nur für SPTS-Streams** zur Verfügung.

pipe — Lesen aus und Schreiben in eine bestehende benannte Pipe (FIFO). Im Gegensatz zu **std** startet PSS keinen Kindprozess — der externe Producer / Consumer muss unabhängig gestartet werden und die Pipe auf seiner Seite offen halten. In den Einstellungen gibt **File Path** den Pfad zu einer bereits existierenden FIFO an (erzeugt z. B. mit dem Befehl `mkfifo`). PSS öffnet die FIFO im Read-Modus (für **input**) bzw. Write-Modus (für **output**). Verfügbar für SPTS und MPTS.

4.1.6 Streams mit Dateien und Geräten

Für **input** und **output** steht das Protokoll **file/device** für Dateien und Geräte zur Verfügung.

output file/device — Aufzeichnung in eine Datei oder Ausgabe an ein Gerät. Eine Datei-Aufzeichnung kann zur ts-Datei-Aufnahme und nachfolgenden Diagnose mit anderen Analyzern erforderlich sein. Geräte-Ausgabe — beliebiges Gerät (auch SDI), das in **/dev** registriert ist.

input file/device — Loop-Wiedergabe von Video aus einer TS-Datei.

Bei Arbeit mit Dateien wird der vollständige Dateipfad im Feld *File Path* angegeben:
/catalog/stream.ts.

Bei Geräten wird zusätzlich das Flag *Is Device* aktiviert.

4.1.7 Liste erlaubter Streams und Peer-Beschränkung

Zur Einschränkung des Zugriffs auf TV-Streams seitens des Clients (**Peer**) in den Modi SRT Listen, PS1, HLS und HTTP ist im Programm die Funktionalität einer Liste erlaubter Streams implementiert. In den **Peer**-Einstellungen am Sender wird die Liste der verfügbaren Sender festgelegt. Dazu im Feld *Stream Access* aus der allgemeinen Senderliste des Servers nur diejenigen Sender hinzufügen, die diesem **Peer** ausgegeben werden sollen. Standardmäßig sind bei leerer Liste alle Sender verfügbar.

Für **Peer** lassen sich Zeit- und Verbindungslimits pro Transportprotokoll setzen.

Anonymous peer

Standardmäßig hat das Peer-Login den Wert *anonymous*. Ein anonymer Peer erlaubt die Stream-Verteilung ohne Bindung an IP oder Login/Passwort. Es gelten Einschränkungen hinsichtlich der Anzahl ausgegebener Streams je Transportprotokoll, des Ablaufdatums und der Liste erlaubter Streams.

Es ist möglich, einen individuellen Peer per Login (Name) und Passwort anzulegen.

Für die Peer-Autorisierung per IP muss die Option „Login Is IP“ aktiviert werden.

Autorisierungsoptionen:

- Nach einzelner IP
- Nach IP-Bereich, z. B. „192.168.1.10-192.168.1.20“
- Kombinierte Variante, Syntax der IP-Listen: ip[-ip2][,...]

4.1.8 Anbindung von Drittanwendungen

Zur Unterstützung anderer Protokolle, die mit den eingebauten Mitteln nicht abgedeckt sind, lassen sich Streams über externe Konsolenanwendungen empfangen und übertragen. Dafür gibt es ein eigenes **std**-Protokoll für **input** und **output**. Der MPEG-TS-Stream wird über die Standardein-/-ausgabe des Betriebssystems empfangen und übertragen.

In den Einstellungen werden die Konsolenanwendung (absoluter Pfad) und die Kommandozeile angegeben; Umgebungsvariablen sind ebenfalls möglich.

Bei **input** mit externer Anwendung müssen Meldungen auf stdout vermieden werden — Ausgabe nur auf stderr.

Für den **output** lässt sich Paketierung von bis zu 7 MPEG-TS-Paketen einstellen.

4.1.9 Anforderungen an den Eingangsstream

Konformität mit ISO 13818-1, Single Program (SPTS) oder Multi Program Transport Stream (MPTS). MPTS-Besonderheiten unten; die folgenden Einstellungen beziehen sich auf SPTS.

Mindestens eine Tonspur ist erforderlich.

Streams ohne Video werden unterstützt, aktiviert über den **Radio**-Modus.

Verschlüsselte Streams werden unterstützt; dafür muss **Scrambled Stream** aktiviert werden.

Für die **Synchronisation** muss der Stream gültige PCR-Marken enthalten.

4.1.10 Stream-Einstellungen

Einen eindeutigen Stream-Namen vergeben (lateinische Buchstaben, Ziffern, „_“, „-“). Zusätzlich kann ein Anzeigename gesetzt werden, der Kyrillisch und andere Sprachen unterstützt.

Stream Timeout — globales Stream-Timeout. Liefert kein gültiger Input innerhalb dieser Zeit, erfolgt ein vollständiger Neustart.

Pause — versetzt **stream** sowie alle **input** und **output** in den inaktiven Zustand. Neu hinzugefügte **stream**s sowie **input**s/**output**s sind standardmäßig pausiert und inaktiv.

Das Programm prüft den Eingangsstream am **input** auf Gültigkeit. Schlägt die Prüfung fehl, gilt der **input** als fehlerhaft.

Check Interval — Intervall für erneute Stream-Prüfung.

MPEG-TS-Filtereinstellungen:

Remove All Unnecessary Data — entfernt alle nicht notwendigen Daten außer PAT/PMT, Video und Audio, mit Ausnahme der in den separaten Filtern festgelegten Elemente (siehe unten)

Remove SDT — SDT-Daten entfernen (Kanalname, Anbieter usw.).

Remove EIT/EPG — EPG-Daten entfernen.

Remove Teletext — Teletext entfernen.

Remove Subtitles — Untertitel entfernen.

Bitratensteuerung des Streams:

Bitrate mode — Bitratensteuerungsmodus.

1. Origin (default) — der Stream wird unverändert weitergegeben.
2. VBR — entfernt NULL-Pakete und minimiert so den Bitrate. Nur für OTT-Distribution aktivieren.
3. CBR auto — aktiviert Bitratenausgleich durch NULL-Pakete (Stuffing). Die Ziel-Bitrate richtet sich nach der maximalen Eingangsbitrate.
4. CBR set stuffing bitrate — die gewünschte Bitrate explizit setzen. Liegt sie unter der Eingangsbitrate, wird wie im CBR-Auto-Modus angeglichen.

Bei aktivem CBR entspricht die PCR Accuracy der TR 101 290; das PCR-Intervall bleibt wie im Originalstream.

Stream-Korrekturen:

Fix PAT/PMT interval — passt den Intervall an TR 101 290 an, indem zusätzliche PAT/PMT eingefügt werden.

4.1.11 Quellenredundanz

Mehrere **input**s können als Liste angegeben werden, aber nur einer ist aktiv. Fällt der aktive **input** aus, wird der nächste der Liste versucht — zyklisch.

Wird im **stream Fallback Check** aktiviert, erfolgt während des Betriebs eines Backup-**input** (nicht des ersten in der Liste) eine erneute Prüfung der höher gelisteten Einträge im Intervall **Check Interval**. Ist der Stream bei der Wiederholungsprüfung gültig, schaltet **stream** auf ihn um.

Da die Reihenfolge der **input**s relevant ist, kann sie geändert werden. Ein pausierter **input** wird im Betrieb nicht berücksichtigt.

4.1.12 Filtern und Modifizieren von MPEG-TS

Standardmäßig wird der MPEG-TS-Stream unverändert weitergegeben.

Für jeden **input** stehen folgende Filteroptionen für den MPEG-TS-Stream zur Verfügung:

PID Accept — Liste erlaubter PIDs. Bei leerer Liste ist alles erlaubt außer **PID Reject**.

PID Reject — Liste verbotener PIDs. Hat Vorrang vor **PID Accept**.

PIDs können geändert werden. Dafür werden die Listen **PID Old** und **PID New** befüllt.

Mapping PID and Languages — Neuzuordnung der Sprache von Audiospuren.

Default Language — Standardsprache festlegen, falls die Tonspur keine Sprache hat.

Für den **stream** können neue MPEG-TS-Daten (SDT-Tabelle) gesetzt werden:

- MPEG-TS Network ID
- Service Name
- Provider Name
- Language

4.2 MPTS-Streams

MPTS-Stream — ein MPEG-TS-Stream mit mehreren Diensten; jeder Dienst besitzt eine eindeutige Programmnummer (PNR). Wird für die DVB-Ausstrahlung verwendet.

Für MPTS-Streams stehen keine Filteroptionen zur Verfügung. Die Streams werden unverändert weitergegeben.

RTSP kann keine Quelle eines MPTS-Streams sein — es ist ein single-program-Protokoll, das nur als SPTS-Quelle anwendbar ist.

Die Mosaic-Funktion ist standardmäßig aus. Auf schwachen CPUs nicht aktivieren — kann zusätzlichen Jitter erzeugen.

In der Stream-Diagnose werden Daten für jedes Programm separat sowie eine Gesamtstatistik angezeigt.

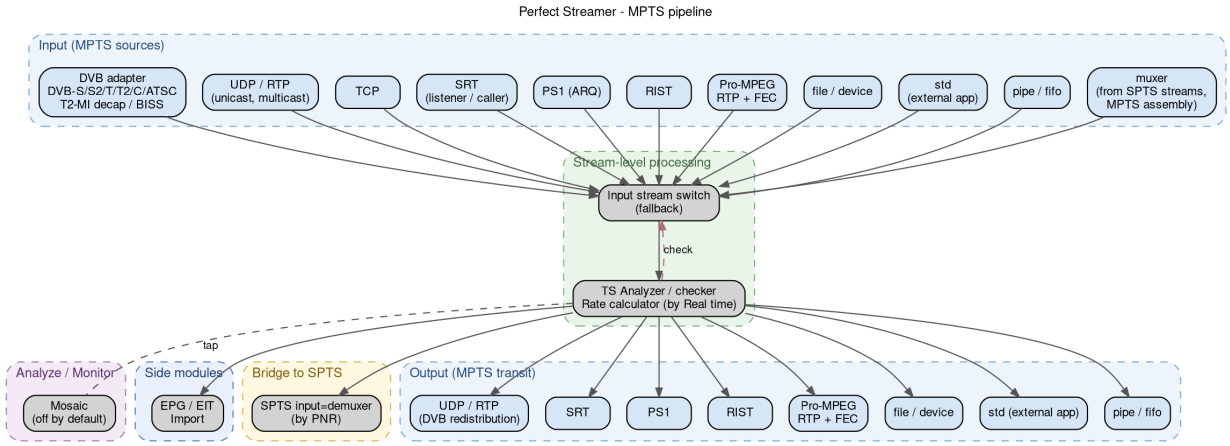


Abb. 2: Architektur des MPTS-Streams: Durchleitung ohne dienstebezogene Filterung; für eine dienstebezogene Verarbeitung (OTT, DVR, Filterung) wird der Demultiplexer-Input eines SPTS-Streams verwendet.

4.2.1 Demultiplexer

Extrahiert einzelne Streams aus einem MPTS-Stream. Hierzu im SPTS-Stream einen Input vom Typ demux hinzufügen, die Quelle wählen und den Dienst per PNR auswählen. Ist die Quell-MPTS aktiv, steht bei der PNR-Auswahl eine Liste zur Verfügung; andernfalls ist die PNR manuell einzugeben.

4.2.2 Multiplexer

Stellt aus einzelnen SPTS-Streams einen MPTS-Stream zusammen. So konfigurieren Sie ihn:

- MPTS-Stream erstellen.
- **Einen Input vom Typ muxer hinzufügen. Bitrate für CBR-Stream-Alignment setzen (Stuffing, TR-101-290- und T-STD-Konformität).**
  Wird 0 (Standard) gesetzt, findet kein Alignment statt — die Bitrate entspricht den Eingangsstreams. Dort können auch einige MPEG-TS-Parameter angegeben werden; für die meisten Anwendungen sind die Standardwerte ausreichend.
- Im Quell-Stream einen Output vom Typ muxer hinzufügen. Servicename und ggf. Providername eintragen. Bei Nicht-Latein-Schrift in den MPEG-TS-Stream-Einstellungen die Sprache wählen.
- Für alle Quellen wiederholen.

Der Multiplexer erzeugt für den Stream SDT, NIT und TDT/TOT (Zeitmarken). EIT (EPG) stammt aus den Quellströmen. PIDs werden neu vergeben.

4.3 Teststreams

Test Stream generator - test stream (test card). It allows to create generated video streams as plugs for broadcasting or failures on main streams. It is possible to set the type of image, sound, overlay text and time.

Die Konfiguration erfolgt durch Aktivieren des passenden Input-Typs am Stream. Videoformat, Auflösung, Bitrate, Lautstärke und Audiofrequenz u. a. lassen sich einstellen.

Die Liste der verfügbaren Test Streams findet sich im linken Seitenmenü der Anwendung.

4.4 OTT-Dienst

Liefert Streams über HTTP-basierte Protokolle — **HLS** (über MPEG-TS), **MPEG-DASH** und **Low-Latency HLS** (über CMAF — fragmentiertes MP4, seit Version 1.13) sowie **MPEG-TS over HTTP**. Unterstützt werden HTTPS (SSL) und HTTP/3 (QUIC). Die Auslieferung wird auf der Registerkarte **OTT** der **Stream**-Einstellungen aktiviert.

Die Verbindungs-URLs haben das Format:

- http://host:port/http/stream/login/password — Autorisierung per Login und Passwort
- http://host:port/http/stream/login — Autorisierung per Login (Token)
- http://host:port/http/stream/ — Autorisierung per IP

host und **port** werden in den **http server**-Einstellungen festgelegt.

stream — **ID** des Streams. Nicht zu verwechseln mit der Reihenfolge in der Stream-Liste. Die **ID** wird im Kopf der Stream-Statistikseite und in der Spalte *ID* der Stream-Liste angezeigt; sie wird bei der Stream-Erstellung festgelegt und ändert sich nie.

Analog für **HLS**, **DASH** und **Low-Latency HLS** (die letzten beiden — nur im *OTT/HLS/LL-HLS/LL-Dash*, siehe unten):

- http://host:port/hls/stream/login/password
- http://host:port/hls/stream/login
- http://host:port/hls/stream/
- http://host:port/dash/stream/login/password
- http://host:port/dash/stream/login
- http://host:port/dash/stream/
- http://host:port/llhls/stream/login/password
- http://host:port/llhls/stream/login
- http://host:port/llhls/stream/

Auf der Stream-Statistik-Seite werden die URLs der verbundenen Protokolle (als Vorlage) und ihr aktueller Status angezeigt. Nicht autorisierter Zugriff ist verboten — Clients müssen in **Peers** eingetragen sein.

Für **HLS** und **DASH** sind in der URL zusätzliche Parameter verfügbar (optional):

[URL]?a=1&s=40&m=40&v=5&h3=1

- **a**: 1 — absoluter Pfad in der Playlist, 0 — relativer Pfad (Standard).

- **s**: Länge der dynamischen Playlist (Sekunden), Standard 40 s.
- **m**: Mindestlänge der dynamischen Playlist (s); Standard 40 s. Maximale Länge der dynamischen Playlist 60 s. Liegt die aktuelle Chunk-Buffer-Größe unter der im Request geforderten Mindestgröße, wird HTTP 404 zurückgegeben. So startet HLS stets mit einem gefüllten Chunk-Buffer auf dem Server.
- **v**: die in der Playlist ausgegebene HLS-Protokollversion. Standardmäßig hängt der Wert vom HLS-Modus ab (siehe unten): *OTT/HLS* und *OTT/HLS/LL-HLS/LL-Dash — 6*, *Peering/HLS — 3*. Ein expliziter Wert in der URL überschreibt den Standard. Ein Wechsel der Version kann für die Kompatibilität mit einem bestimmten HLS-Client erforderlich sein.
- **h3**: Opt-in für HTTP/3 (QUIC) für diese OTT-Sitzung. Veranlasst den Server, im Response einen `Alt-Svc`-Header auszugeben; ein kompatibler Player / Browser wechselt daraufhin auf QUIC. Siehe Abschnitt HTTP/3 (QUIC) weiter unten.

Für die Kompatibilität mit einigen HLS-Clients kann der Dateiname index.m3u8 an die URL angehängt werden, z. B. http://host:port/hls/stream/login/password/index.m3u8.

Der Auslieferungsmodus wird über die Stream-Einstellung *OTT HLS* festgelegt (Registerkarte **OTT**): *Peering/HLS*, *OTT/HLS* oder *OTT/HLS/LL-HLS/LL-Dash*.

Peering/HLS — ein Modus mit einfacher Aufteilung in Segmente (Chunks). Empfohlen für das Peering (die Distribution) von Streams. Es wird nur **HLS** über MPEG-TS (`/hls`) ausgeliefert. Standardmäßig wird die Playlist als *EXT-X-VERSION:3* ausgegeben, um mit Peer-Clients kompatibel zu sein.

OTT/HLS — ein Modus mit einer Segmentaufteilung, die auf einen schnellen Player-Start beim OTT-Broadcasting optimiert ist. In diesem Modus ist die CPU-Last höher; er wird für das Broadcasting empfohlen. Es wird **HLS** über MPEG-TS (`/hls`) ausgeliefert. Standardmäßig wird die Playlist als *EXT-X-VERSION:6* mit *EXT-X-INDEPENDENT-SEGMENTS* und dem Attribut *CHARACTERISTICS* in *EXT-X-MEDIA TYPE=SUBTITLES* ausgegeben (Apple HLS, hls.js, Safari, dash.js/Shaka). Benötigt ein bestimmter Client einen früheren Wert, setzen Sie ihn über den Query-Parameter `?v=` (siehe die Parameterliste oben).

OTT/HLS/LL-HLS/LL-Dash — ein Auslieferungsmodus über **CMAF** (fragmentiertes MP4, *fMP4*; seit Version 1.13). Der Stream erzeugt fMP4-Segmente (`.m4s` + gemeinsame `init.mp4`, `mimeType="video/mp4"`), auf deren Basis ausgeliefert werden:

- **MPEG-DASH** unter `/dash` — jetzt über CMAF und **nicht** über MPEG-TS;
- **Low-Latency HLS** unter `/llhls` (siehe Low-Latency HLS unten);
- bei aktivierter Stream-Einstellung *Enable TS Chunk* (Standard *true*) — zusätzlich Legacy-**HLS** über MPEG-TS (`/hls`), wie im *OTT/HLS*; bei *false* werden nur fMP4-Segmente ausgeliefert, was Festplatte und CPU schont.

Die **HLS**-Playlist im *OTT/HLS/LL-HLS/LL-Dash* ist standardmäßig ebenfalls *EXT-X-VERSION:6*.

Bemerkung: **MPEG-DASH** und **Low-Latency HLS** sind nur im *OTT/HLS/LL-HLS/LL-Dash* verfügbar. Im *OTT/HLS* und *Peering/HLS* wird ausschließlich **HLS** über MPEG-TS ausgeliefert.

Für den HTTP-Server kann SSL (HTTPS) aktiviert werden — in den Server-Einstellungen.

Chunk Min Interval und Chunk Max Interval

Im OTT-Modus wird der Stream auf PAT/PMT/SPS/PPS/IFrame analysiert, und die Chunks werden nach dem Kriterium des schnellen Player-Starts geschnitten. Die Analyse beginnt bei *min interval*, und falls die Daten aus irgendeinem Grund nicht gefunden werden, wird der Chunk bei *max interval* zwangsweise geschnitten.

GOP-aligned segments

Im *OTT/HLS* richtet der Segmenter die Chunk-Grenzen an Random-Access-Punkten aus und unterscheidet zwischen *IDR* und einem gewöhnlichen *I-frame*. Sendet die Quelle mit *closed-GOP* (IDR vorhanden), beginnt jedes *.ts*-TS-Segment garantiert mit *SPS / PPS / IDR* (bei HEVC — zusätzlich *VPS*) — einem echten Einstiegspunkt, an dem der Player den Stream öffnet. Ist die Quelle *open-GOP* / ohne IDR, dient der nächstgelegene I-frame als Grenze (das bisherige Verhalten). Der Segmenter schneidet den „Schwanz“ des vorherigen GOP — die *P / B*-Slices — aus dem Fenster zwischen dem führenden PAT und dem ersten SPS des Chunks heraus. Das:

- beseitigt das anfängliche Schwarzbild beim Start einer VOD-Sitzung (`?t=` / `?epg=`) in *hls.js*, *Safari* und *VLC*: der MSE-Decoder erhält das korrekte Header-Set der NAL-Einheiten bereits im ersten Segment und startet sofort;
- bringt die Ausgabe in Übereinstimmung mit *HLS RFC 8216 §3* („Each Media Segment MUST contain a SPS and a PPS that decode its first Access Unit“);
- macht die *EXT-X-INDEPENDENT-SEGMENTS*-Erklärung in einer *EXT-X-VERSION:6*-Playlist korrekt.

Steuerung über die Stream-Einstellung *gop-aligned-segment* (Standard *true*). Wirkt nur bei *HLS = OTT/HLS*: im *Peering/HLS* und bei MPTS-Streams bleibt das Verhalten unverändert. PSI (PAT / PMT) und Audio bleiben vollständig erhalten; einziger Nebeneffekt ist ein einbildiger *continuity_counter*-Sprung auf der Video-PID an der Grenze zweier benachbarter Chunks, was für HLS / DASH MSE eine legitime Decode-Boundary ist.

Bei der Ausrichtung an Einstiegspunkten kann die tatsächliche Chunk-Dauer über *Chunk Min Interval* hinaus aufgerundet werden. Daher spiegelt der Wert *EXT-X-TARGETDURATION* in der Live-Playlist die **maximale tatsächliche** Segmentdauer wider (Abschnitt 4.3.3.1 von *RFC 8216*) statt des konfigurierten Minimums. Das hält das Manifest innerhalb des Standards: *hls.js* und kompatible Player verkürzen das Aktualisierungsintervall der Playlist nicht und lösen keine falschen *bufferStalledError* aus.

Low-Latency HLS (/llhls)

Im *OTT/HLS/LL-HLS/LL-Dash* steht neben `/dash` ein separater **Low-Latency HLS**-Endpoint zur Verfügung — Auslieferung mit reduzierter Latenz auf denselben CMAF-fMP4-Segmenten:

- http://host:port/llhls/stream/login/password
- http://host:port/llhls/stream/login
- http://host:port/llhls/stream/

Die Medien-Playlist wird in *partielle Segmente* (*parts*, `#EXT-X-PART`-Direktiven) aufgeteilt: Der Player beginnt die Wiedergabe, ohne auf das fertige vollständige Segment zu warten. Verwendet werden ein blockierendes Neuladen der Playlist (der Server hält die Anfrage zurück, bis der nächste part bereit ist) und der Preload-Hinweis `#EXT-X-PRELOAD-HINT`.

Die Ziel-Dauer eines part wird über die Stream-Einstellung *Part Target Duration* festgelegt (ms; Standard 500, Bereich 100–5000). Der Wert wird im laufenden Betrieb übernommen, ohne den Stream neu zu starten, und muss kleiner als *Chunk Min Interval* sein.

HLS / DASH / LL-HLS Adaptive Multistream

HLS Adaptive Multistream wird seit Version 1.10 unterstützt, DASH Adaptive Multistream seit Version 1.12 (seit Version 1.13 — über CMAF), Low-Latency HLS Adaptive seit Version 1.13.

Für adaptive Streams wird eine separate Playlist konfiguriert. Dazu ist Folgendes nötig:

- OTT-Auslieferung bei den Streams aktivieren, die in die adaptive Playlist aufgenommen werden: *OTT/HLS* — für adaptives **HLS** über MPEG-TS; *OTT/HLS/LL-HLS/LL-Dash* — für adaptives **DASH / Low-Latency HLS** über CMAF.
- Im Hauptmenü erscheint ein Bereich für adaptive Streams. Dort muss ein Stream hinzugefügt und alle Streams angegeben werden, die in diese Playlist aufgenommen werden sollen.
- Für Streams kann ein Bitrate-Parameter gesetzt werden. Standardmäßig ist er 0 — die Bitrate wird aus dem gemessenen Wert übernommen; andernfalls kann sie explizit gesetzt werden.

Für adaptive Playlists gilt eine andere URL:

- http://host:port/hls/adaptive/stream/login/password
- http://host:port/hls/adaptive/stream/login
- http://host:port/hls/adaptive/stream/
- http://host:port/dash/adaptive/stream/login/password
- http://host:port/dash/adaptive/stream/login
- http://host:port/dash/adaptive/stream/
- http://host:port/llhls/adaptive/stream/login/password
- http://host:port/llhls/adaptive/stream/login
- http://host:port/llhls/adaptive/stream/

Peers (Clients) können — wie bei normalen Streams — Zugriffsbeschränkungen auf adaptive Streams haben. Eine Berechtigung für einen adaptiven Stream schließt die Berechtigung für alle enthaltenen Substreams ein.

4.5 HTTP/3 (QUIC)

Seit Version 1.13.1.438 enthält **Perfect Streamer** einen integrierten **HTTP/3**-Server für die OTT-Auslieferung von HLS, MPEG-DASH und Low-Latency HLS über **QUIC** (RFC 9000 + RFC 9114). Der Stack ist **ngtcp2** (QUIC v1) + **nghttp3** (HTTP/3 frame layer); die TLS-Infrastruktur verwendet dasselbe Zertifikat wie der HTTPS-Listener.

QUIC bedient ausschließlich OTT-Routen:

- `/` — Root-Redirect,
- `/hls/...`, `/dash/...` und `/llhls/...` — Master-Playlists / MPD,
- `/h<sessID>/...` — Per-Session-URL (Media-Playlists, Segmente, VTT),
- `/http/<stream>/...` — Raw-MPEG-TS über HTTP.

Low-Latency HLS und DASH werden über QUIC inkrementell (chunked) ausgeliefert: *parts* gehen an den Client, sobald sie bereit sind, ohne auf das vollständige Segment zu warten.

Administrative Pfade (`/data`, `/config`, `/xmltv`, `/db/`, `/login`, `/logout`, `/restart`) liefern über QUIC **404** — die Admin-API verbleibt auf HTTPS / HTTP-TCP. Dies ist eine bewusste Einschränkung zur Reduzierung der Angriffsfläche des QUIC-Listeners.

4.5.1 Aktivierung

Die QUIC-Einstellungen befinden sich im Abschnitt **Configuration / HTTP server** (Knoten `/config/http-server`):

- **HTTP/3 Enable** (`http3-enable`) — globales Flag zur Aktivierung des QUIC-Listeners. Standardwert: **off**. Die Aktivierung öffnet einen UDP-Socket an `http3-port`; die Deaktivierung schließt ihn.
- **HTTP/3 Port** (`http3-port`) — UDP-Port des QUIC-Listeners. Standardwert: **43984**. QUIC läuft auf UDP, HTTPS auf TCP; die Ports können übereinstimmen oder abweichen — ein Konflikt zwischen ihnen besteht nicht.
- **HTTP/3 0-RTT Enable** (`http3-zero-rtt-enable`) — erlaubt den 0-RTT-Handshake (RFC 9001 §4.6.1) bei wiederaufgenommenen Verbindungen desselben Clients. Reduziert die Start-Latenz; das Replay-Risiko ist bei nicht idempotenten Anfragen zu berücksichtigen (für rein lesende OTT-Last unbedenklich). Standardwert: **on**.

Konfigurationsänderungen werden im laufenden Betrieb angewendet, ohne Neustart des Dienstes.

Zertifikat und Schlüssel werden vom HTTPS-Listener übernommen — es gibt keine separate Zertifikatskonfiguration für HTTP/3. Ist HTTPS nicht konfiguriert (`ssl-enable=false`), liefert die Aktivierung von HTTP/3 nichts — der Handshake scheitert.

4.5.2 Der Parameter `?h3` — Opt-in pro Sitzung

Ein Browser verbindet sich nicht direkt mit einem HTTP/3-Endpunkt — er geht zunächst über HTTPS/TCP, empfängt im Response den Header `Alt-Svc: h3=":<port>"; ma=86400` (RFC 7838) und schaltet erst nachfolgende Anfragen an diesen Origin auf QUIC um.

In Perfect Streamer ist die Ausgabe von `Alt-Svc` ein **Opt-in pro OTT-Sitzung**. Der Server kündigt QUIC nicht jedem Client unterschiedslos an: Der Client muss HTTP/3 explizit über den Query-Parameter `?h3` auf der HLS-/DASH-Master-URL anfordern.

Wert in der URL	Opt-in-Wert	`Alt-Svc` im Response
Parameter fehlt	off (default)	nein
`?h3` (ohne Wert)	on	ja
`?h3=1`, `?h3=on`, `?h3=yes`, `?h3=true`	on	ja
`?h3=0`, `?h3=off`, `?h3=no`, `?h3=false`	explicit off	nein (wie bei fehlend)

Sobald der Client die Session-URL `/h<sess>/...` erhalten hat, wird das `wantH3`-Flag in der Sitzung gespeichert — alle nachfolgenden Media-Playlist-Refreshs, Segment-GETs und VTT-Chunk-GETs unter dieser Session-URL erhalten **ebenfalls** `Alt-Svc`, auch wenn `?h3` in der jeweiligen Anfrage nicht vorhanden ist. Ohne Opt-in auf dem Master entsteht kein Sticky-Verhalten.

`Alt-Svc`-Gating:

1. Der QUIC-Listener ist global aktiviert (`http3-enable=true`); andernfalls wird die Ankündigung selbst bei gesetztem `?h3` unterdrückt.
2. Die Anfrage kam **nicht** über QUIC — bei Anfragen, die bereits über H3 laufen, ist `Alt-Svc` sinnlos und wird nicht ausgegeben.
3. Per-Session- oder Per-URL-`wantH3=true` (siehe Tabelle oben).
4. Der Admin- und der EPG-Server geben niemals `Alt-Svc` aus — sie verfügen über keinen QUIC-Listener.

Dieses Verhalten ist mit RFC 7838 §3 konform — der Server entscheidet selbst, bei welchen Antworten `Alt-Svc` ausgegeben wird.

4.5.3 Szenario der QUIC-Umschaltung im Browser

Typischer Ablauf (Chrome / Firefox / Safari):

1. Der Player öffnet die Master-URL mit explizitem Opt-in:

```
https://stream.example.com:41982/hls/test1/login/password/index.m3u8?h3=1
```

2. Die erste Anfrage erfolgt über HTTPS/TCP. Im Response gibt der Server `Alt-Svc: h3=":43984"; ma=86400` aus.
3. Der Browser merkt sich die Zuordnung `stream.example.com:41982 → h3=":43984"`. In Chrome lässt sie sich auf der Seite `chrome://net-internals/#alt-svc` einsehen; in Firefox — `about:networking#http3`.
4. Bei nachfolgenden Anfragen an denselben Origin (Playlist-Refresh, Segment-GETs, VTT) öffnet der Browser eine QUIC-Verbindung auf `udp/43984` und kommuniziert weiter über HTTP/3. In den DevTools → Network wird die Spalte **Protocol** für diese Anfragen `h3`.

Lässt sich die QUIC-Verbindung nicht aufbauen (UDP-Port am Firewall blockiert, Zertifikat im System nicht vertrauenswürdig), bleibt der Browser transparent auf HTTPS/TCP — der Stream läuft funktional ohne Unterbrechung weiter.

4.5.4 Client-Accounting und Monitoring

Die reale IP-Adresse eines QUIC-Clients im Accounting aktiver Peers (`/data/http-clients`, Limits konkurrierender Verbindungen, IP-basierte ACLs) ist die **tatsächliche Peer-Adresse** der QUIC-Verbindung und nicht der Loopback der internen Backend-Brücke.

Das Attribut `ott-type` in `/data/http-clients` ist zusammengesetzt, im Format `<PROTO>/<scheme>`:

- `PROTO` — OTT-Protokoll: `HLS`, `DASH` oder `HTTP`.
- `scheme` — der tatsächliche Netzwerktransport: `http`, `https` oder `quic`.

Beispiele: `HLS/quic`, `DASH/https`, `HTTP/http`. Das Admin-UI zeigt sowohl das OTT-Protokoll als auch den Netzwerktransport jedes Clients in einer einzigen Spalte.

Das OTT-Sitzungs-Timeout beträgt **60 Sekunden**, unabhängig vom Transport. Wechselt ein Client zwischen Transporten, wird das Schema ausschließlich entlang der Prioritätskette `http → https → quic` aufgewertet. Ein paralleler Rückfall auf eine weniger gesicherte Verbindung überschreibt den aktuellen Typ nicht — im Accounting verbleibt der sicherste beobachtete Transport.

4.5.5 Player-Kompatibilität

HTTP/3 für OTT unterstützen:

- **iOS AVPlayer / Safari** — nativ, über `Alt-Svc`; 0-RTT wird unterstützt.
- **Chrome, Firefox, Edge, Brave** — über `Alt-Svc`; HLS wird mit hls.js, DASH mit dash.js / Shaka wiedergegeben; der Player-Traffic erbt HTTP/3 vom Browser-fetch-API.
- **Android ExoPlayer** — über die Cronet-QUIC-Engine (nicht der Standardtransport; erfordert clientseitige Konfiguration).

VLC (libVLC) unterstützt HTTP/3 nicht — auf diesen Clients funktioniert der Stream weiter über HTTPS/TCP, ohne den QUIC-Vorteil.

Der praktische Vorteil von QUIC gegenüber HTTPS/TCP zeigt sich am deutlichsten in Mobilfunknetzen / Wi-Fi / verlustbehafteten Netzen:

- kein Head-of-Line-Blocking auf TCP-Ebene — ein Verlust in einem HTTP-Stream blockiert die übrigen nicht;
- 0-RTT-Handshake bei wiederaufgenommenen Verbindungen desselben Clients;
- stabilere ABR-Bitrate bei Paketverlusten von 1–5 %.

In verwalteten Netzen / Unternehmens-Wi-Fi fällt der Gewinn meist bescheiden aus — 5–10 % bei der Startup-Latenz und nahezu null im Steady-State. Die CPU-Last auf dem Server ist bei QUIC höher als bei Kernel-TCP — der Preis für TLS und Transport im User Space.

4.6 Caching-Modell für OTT HLS und DASH

Der Server liefert Antworten in drei Kategorien, die sich in Inhaltslebensdauer und Cache-Eignung in Zwischenknoten (Reverse Proxy, CDN, Client-Cache) unterscheiden.

4.6.1 1. Caching-Modell

1.1. Ressourcen und HTTP-Header

Ressource	URL	Content-Type	Cache-Control
TS-Segment (HLS)	`/h<sess>/<keyHex>.ts`, `/h<sess>/sub/<pid>/<keyHex>.vtt`	`video/mp2t`	`public, max-age=60, immutable`
fMP4-Segment (DASH / LL-HLS)	`/h<sess>/init.mp4`, `/h<sess>/<key|N>.m4s`	`video/mp4`	`public, max-age=60, immutable`
DASH MPD	`/h<sess>/index.mpd`	`application/dash+xml; charset=utf-8`	`public, max-age=1`
HLS master	`/hls/<stream>/<login>/<pass>/index.m3u8`	`application/vnd.apple.mpegurl`	`public, max-age=1`
HLS media	`/h<sess>/index.m3u8`, `/h<sess>/sub/<pid>/index.m3u8`	`application/vnd.apple.mpegurl`	`public, max-age=1`
302 Redirect	`/dash/<stream>/<login>/<pass>/index.mpd`	—	`no-cache, no-store`
Raw TS	`/http/<stream>/<login>/<pass>`	`video/mp2t`	nicht gesetzt; nicht gecacht

1.2. Segmenteigenschaften

Der hexadezimale Segment-Identifier in der URL (`<keyHex>` in den Pfaden `/h<sess>/<keyHex>.ts`) wird als CRC64 über die Segment-Startzeit und die Stream-ID gebildet und ist global eindeutig. Die Segment-URL adressiert unveränderliche Inhalte — bei wiederholten Anfragen an dieselbe URL wird ein identischer Bytestrom zurückgegeben (solange das Segment innerhalb des Schiebefensters bleibt).

Die Direktive `immutable` unterdrückt die bedingte Revalidierung durch den Client (`If-None-Match`, `If-Modified-Since`). `max-age=60` ist mit dem typischen `timeShiftBufferDepth=40s` kompatibel.

CMAF-fMP4-Segmente (`.m4s`) und die gemeinsame `init.mp4` für **DASH / Low-Latency HLS** werden analog adressiert und nach demselben Modell zwischengespeichert (`immutable`, `max-age=60`). Partielle Segmente (*parts*) von LL-HLS sind Byte-Range-Anfragen in dieselbe `.m4s`, sodass sie keinen separaten Cache-Eintrag bilden.

1.3. Eigenschaften der Manifeste

`max-age=1` begrenzt die obere Schranke der Inhalts-Veralterung im Cache auf eine Sekunde. Zusammen mit `proxy_cache_lock on` (nginx) werden Anfragenspitzen am Manifest zu einer einzigen Origin-Anfrage pro Sekunde zusammengefasst.

1.4. Inhaltsvariabilität

Bei `absPath=0` (Standard; ohne URL-Parameter `a`) enthalten HLS-media- und DASH-MPD-Manifeste keinen Sitzungs-Identifier im Body. Der Manifest-Inhalt ist zwischen Sitzungen identisch, die zur selben (stream, param)-Kombination gehören. Dadurch kann ein Reverse-Proxy-Cache bei normalisiertem Cache-Key einen einzigen Eintrag sitzungsübergreifend wiederverwenden.

Bei `absPath=1` (URL-Parameter `a=1`) enthält der Manifest-Body absolute URLs einschließlich Schema, Host und Sitzungs-Identifier. Der Inhalt wird sitzungsspezifisch; eine sitzungsübergreifende Cache-Wiederverwendung ist nicht möglich.

4.6.2 2. Client-Verhalten

Klient	Manifest-Refresh-URL	Auswirkung auf die Sitzungszahl
VLC 3.x HLS	`/h<sess>/index.m3u8`	Eine Sitzung pro Wiedergabe
VLC 3.x DASH	`/dash/<stream>/.../index.mpd`	Wird per Session-Reuse behandelt (siehe 3.3)
ffmpeg 5.x HLS	`/h<sess>/index.m3u8`	Eine Sitzung pro Wiedergabe
ffmpeg 5.x DASH	`/dash/<stream>/.../index.mpd` (Wiederholungsschleife)	Wird per Session-Reuse behandelt (siehe 3.3)
dash.js, hls.js	`/h<sess>/...` über `<Location>` / Session-URL	Eine Sitzung pro Wiedergabe

4.6.3 3. Spezielle Mechanismen

3.1. HTTP 302 Redirect für DASH

Eine Anfrage der Form `/dash/<stream>/<login>/<pass>/index.mpd` liefert die Antwort `302 Found` mit dem Header `Location: /h<sess>/index.mpd`. Der Antwort-Body ist leer. Authentifizierung und Sitzungs-Allokation finden in der Phase der Redirect-Verarbeitung statt.

Clients, die Redirect-Caching unterstützen, greifen in nachfolgenden Anfragen direkt auf die Sitzungs-URL zu. Clients ohne Unterstützung wiederholen die Redirect-Anfrage. Die Kosten der Redirect-Wiederverarbeitung beschränken sich auf Authentifizierungsprüfung und Session-Reuse-Operationen.

3.2. Session reuse für DASH

Bei der Verarbeitung einer `/dash/.../index.mpd`-Anfrage desselben Logins an denselben Stream (mit demselben *adaptive*-Merkmal) findet der Server eine bereits bestehende DASH-Sitzung und gibt deren Kennung erneut zurück. Es wird keine neue Sitzung erstellt; ein Platz im Limit gleichzeitiger Verbindungen wird nicht verbraucht.

Gilt nur für DASH. Für HLS ist kein eigener Reuse-Mechanismus erforderlich: HLS-Clients aktualisieren die Media-Playlist über die Session-URL und erzeugen bei jedem Refresh keine neue Sitzung.

3.3. Segmentwiederverwendung zwischen Sitzungen

Der Pfad `/h<sess>/<keyHex>.ts` ist unabhängig von `<sess>`, wenn `<keyHex>` auf Inhalt aufgelöst wird: `<keyHex>` identifiziert ein TS-Segment innerhalb eines Streams global eindeutig. Nginx mit einem normalisierten Cache-Key (Entfernen des Präfixes `/h<sess>/`) bedient jede Anfrage nach demselben `<keyHex>` aus einem einzigen Cache-Eintrag, unabhängig davon, welche Clients sie gestellt haben.

Dasselbe gilt für CMAF-fMP4-Segmente (`.m4s`) und `init.mp4`: Ihr Inhalt ist unveränderlich und wird innerhalb des Streams adressiert, sodass die Normalisierung des Cache-Keys dieselbe sitzungsübergreifende Deduplizierung im Cache ergibt.

4.6.4 4. Anfrageparameter

Parameter	Standardwert	Auswirkung
`a`	`0`	`1` — absolute URLs in den Manifesten; `0` — relative
`s`	`40`	`timeShiftBufferDepth` in Sekunden
`m`	`40`	Mindestfensterlänge für die Manifest-Ausgabe
`v`	`6` für *OTT/HLS* und *OTT/HLS/LL-HLS/LL-Dash*, `3` für *Peering/HLS*	`#EXT-X-VERSION` in HLS (von DASH ignoriert); ein expliziter Wert in der URL überschreibt den Default
`h3`	fehlt (off)	Opt-in für HTTP/3 (QUIC) — veranlasst den Server, `Alt-Svc` im Response auszugeben. Erkannte Werte: presence ⇒ on, `1`/`on`/`yes`/`true` ⇒ on, `0`/`off`/`no`/`false` ⇒ explicit off. Sticky auf der Session-URL `/h<sess>/...`. Siehe Abschnitt HTTP/3 (QUIC).

Das Ändern eines Parameters per Query-String aktualisiert die in der Sitzung gespeicherten Werte beim nächsten erneuten Öffnen der Sitzung.

4.6.5 5. Lasteigenschaften

Die Origin-Last skaliert mit der Anzahl gleichzeitig beobachteter unterschiedlicher Streams. Die Erhöhung der Anzahl gleichzeitig denselben Stream beobachtender Clients erhöht die Anzahl der Origin-Anfragen nicht, sofern ein Reverse-Proxy-Cache mit normalisiertem Cache-Key vorhanden ist.

Szenario	Origin-Request-Rate (Ref.)
1 Client pro Stream X	MPD: 0.4 req/s, segment: 0.2 req/s
N Clients auf einem Stream X (Cache aktiv)	MPD: 1 req/s, segment: 0.2 req/s
N ffmpeg-Clients im Replay-Modus auf einem Stream	MPD: 1 req/s (mit `proxy_cache_lock`)
N Clients auf N verschiedene Streams	MPD: 0.4·N req/s, segment: 0.2·N req/s

4.6.6 6. Nginx als zwischenspeichernder Reverse Proxy

6.1. Basiskonfiguration

```
proxy_cache_path /var/cache/nginx/pss_segments
    levels=1:2 keys_zone=pss_segments:100m
    max_size=20g inactive=30m use_temp_path=off;

proxy_cache_path /var/cache/nginx/pss_manifests
    levels=1:2 keys_zone=pss_manifests:10m
    max_size=256m inactive=5m use_temp_path=off;

upstream pss_backend {
    server 127.0.0.1:41972;
    keepalive 64;
}

map $uri $pss_cache_key {
    ~^/h[0-9a-f]{16}(?<tail>/.+\.(ts|m3u8))$  "stream:$tail";
    default                                    $uri;
}

server {
    listen 80;
    server_name stream.example.com;

    location ~* "^/h[0-9a-f]{16}(/[0-9]+)?/([0-9a-f]+\.(ts|m4s)|init\.mp4)$" {
        proxy_cache              pss_segments;
        proxy_cache_key          $pss_cache_key;
        proxy_cache_valid        200 60s;
        proxy_cache_valid        404 403 0s;
        proxy_cache_lock         on;
        proxy_cache_use_stale    updating error timeout;
        proxy_cache_revalidate   on;
        add_header               X-Cache-Status $upstream_cache_status;

        proxy_pass               http://pss_backend;
        proxy_http_version       1.1;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        proxy_set_header          Connection "";
        proxy_buffering           on;
    }

    location ~* "(^/h[0-9a-f]{16}(/[0-9]+)?/index\.(m3u8|mpd)$|^/(hls|dash)/.*\.
↪(m3u8|mpd)$)" {
        proxy_cache               pss_manifests;
        proxy_cache_key           $pss_cache_key;
        proxy_cache_valid         200 1s;
        proxy_cache_valid         404 403 0s;
        proxy_cache_lock          on;
        proxy_cache_lock_timeout  2s;
        proxy_cache_use_stale     updating;
        add_header                X-Cache-Status $upstream_cache_status;

        proxy_pass                http://pss_backend;
        proxy_http_version        1.1;
        proxy_set_header          Connection "";
    }

    location / {
        proxy_pass                http://pss_backend;
        proxy_http_version        1.1;
        proxy_set_header          Connection "";
        proxy_set_header          X-Forwarded-Proto $scheme;
        proxy_set_header          X-Forwarded-Host  $host;
        proxy_buffering           off;
        proxy_read_timeout        3600s;
    }
}
```

6.2. Zweck der Direktiven

Direktive	Zweck
`proxy_cache_lock on`	Serialisiert Upstream-Anfragen bei parallelen Cache-Misses auf denselben Schlüssel
`proxy_cache_use_stale updating`	Liefert die veraltete Kopie an parallele Anfragen während der Cache-Aktualisierung
`proxy_cache_revalidate on`	Nutzt `If-Modified-Since` bei Cache Miss mit vorhandener Kopie
`proxy_cache_valid 404 403 0s`	Verbietet das Caching von Authorization-Fehlern und 404
`keepalive 64` im Upstream	Verwaltet einen Pool persistenter Verbindungen zum Origin
`proxy_buffering on`	Für Segmente; aktiviert die Antwortpufferung in nginx
`proxy_buffering off`	Für den Bereich `/`; deaktiviert Buffering (Raw Streaming)

6.3. Berechnung von `max_size` für den Segment-Cache

Richtwert: `bitrate × timeShiftBufferDepth × distinct_streams × 2`

Beispiel: 10 Streams × 8 Mbps × 40 s × 2 ≈ 800 MB. Es wird empfohlen, einen 10-fachen Sicherheitsfaktor für Bitratenschwankungen einzuplanen.

6.4. TLS-Terminierung

Der Perfect-Streamer-Server akzeptiert Verbindungen auf HTTP- und HTTPS-Ports. Bei TLS-Terminierung am nginx verwendet das Upstream den HTTP-Port. Die Weiterleitung der Header `X-Forwarded-Proto` und `X-Forwarded-Host` ist erforderlich für die korrekte Bildung absoluter URLs bei `absPath=1`.

```
server {
    listen 443 ssl http2;
    server_name stream.example.com;

    ssl_certificate           /etc/letsencrypt/live/stream.example.com/fullchain.pem;
    ssl_certificate_key       /etc/letsencrypt/live/stream.example.com/privkey.pem;
    ssl_protocols             TLSv1.2 TLSv1.3;
    ssl_session_cache         shared:SSL:10m;
    ssl_session_timeout       1d;

    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

    location ... {
        proxy_pass                http://pss_backend;
        proxy_set_header          X-Forwarded-Proto https;
        proxy_set_header          X-Forwarded-Host  $host;
        proxy_set_header          Host              $host;
        # + caching directives from 6.1
    }
}

server {
    listen 80;
    server_name stream.example.com;
    return 301 https://$host$request_uri;
}
```

Bei HTTPS zwischen nginx und Origin gelten `proxy_ssl_verify` und `proxy_ssl_trusted_certificate`. Bei Loopback-Verbindungen ist Verschlüsselung überflüssig.

6.5. Multi-host

Wenn ein nginx-Prozess mehrere `server_name` bedient, wird `$host` zum Cache-Schlüssel hinzugefügt, um Inhalte zu isolieren:

```
map $uri $pss_cache_key {
    ~^/h[0-9a-f]{16}(?<tail>/.+\.(ts|m3u8))$  "$host:stream:$tail";
    default                                    "$host:$uri";
}
```

Die `keys_zone`-Größe wird mit 8000 Keys/MB berechnet. Für Multi-Host-Installationen mit Tausenden von Streams werden `keys_zone=...:300m` oder mehr empfohlen.

4.6.7 7. Client-seitiges Caching

`Cache-Control: immutable` wird von den Browsern Chrome/Firefox/Safari berücksichtigt. Der Client-Cache liefert das Segment bei wiederholtem Zugriff ohne bedingte Anfrage zurück (auch bei Rückwärts-Seek innerhalb des Player-Buffers).

Service Workers können basierend auf dem Cache-Control-Inhalt eine `cache-first`-Strategie anwenden. DASH-Player (dash.js, Shaka) nutzen MSE über `SourceBuffer`; ein in den Buffer eingelegtes Segment bleibt ohne erneute HTTP-Anfrage verfügbar, bis es das Schiebe-Fenster verlässt.

Für Cross-Domain-Anfragen erlaubt der Header `Access-Control-Allow-Origin: *` Caching in shared caches ohne `Vary: Origin`. Beim Wechsel des ACAO-Werts auf einen konkreten Origin wird `Vary: Origin` erforderlich, was die Effizienz des shared cache reduziert.

4.6.8 8. Bereitstellung über CDN

Perfect Streamer ist mit Pull-from-Origin-CDNs kompatibel (Cloudflare, Akamai, Fastly, BunnyCDN, Amazon CloudFront).

Origin shield. Empfohlen werden ein oder mehrere Shield-Knoten zwischen CDN-Edge und Origin, um die Origin-Anfragerate bei global verteilten Clients zu senken.

Purge. Content-addressed Segmente erfordern keinen Purge. Bei Stream-Metadaten-Änderungen (Codec, Auflösung) aktualisieren sich Manifeste innerhalb von `max-age=1` ohne expliziten Purge.

Cache Warming. Bei erwartetem Lastanstieg auf einen Stream darf das CDN von mehreren Standorten aus vor Sendebeginn vorgeheizt werden.

Geo-Verteilung. Segmente (`max-age=60`) eignen sich gut für geografisch verteiltes Caching. Manifeste (`max-age=1`) tolerieren bis zu eine Sekunde Lieferverzögerung — akzeptabel für non-low-latency live.

4.6.9 9. Überwachung

9.1. X-Cache-Status

`add_header X-Cache-Status $upstream_cache_status;` in jeder gecachten Location ergänzen. Werte:

Wert	Beschreibung
`HIT`	Antwort aus Cache
`MISS`	War nicht im Cache; vom Origin geholt und gespeichert
`EXPIRED`	Abgelaufen, erneuert
`UPDATING`	Stale-Kopie an parallele Anfrage während der Aktualisierung ausgeliefert
`STALE`	`use_stale` lieferte die abgelaufene Kopie (Origin nicht erreichbar)
`REVALIDATED`	Origin lieferte 304 Not Modified
`BYPASS`	`proxy_cache_bypass` wurde ausgelöst

9.2. Format des access-log

```
log_format pss_cache '$remote_addr $status $request_method "$request" '
                     '$body_bytes_sent rt=$request_time ut=$upstream_response_time '
                     'cache=$upstream_cache_status key=$pss_cache_key';

server {
    access_log /var/log/nginx/pss.log pss_cache;
}
```

9.3. Metriken

Das Modul `nginx-vts` exportiert Per-Zone-Metriken im Prometheus-Format:

```
GET /status/format/prometheus
```

Empfohlene Schwellwerte für Alerts:

Metrik	Schwelle	Mögliche Ursache
Segment HIT rate	< 90 % über 5 Minuten	Cache-Key-Normalisierung defekt; `max_size` zu klein
Manifest MISS rate	> 50 % über 1 Minute	`proxy_cache_lock` serialisiert die Anfragen nicht
Upstream response time p95	> 500 ms über 1 Minute	Origin-Überlastung
Cache zone fill	> 90 % über 10 Minuten	Annäherung an `max_size`; LRU-Eviction wird geplant

4.6.10 10. Diagnose

Symptom	Wahrscheinliche Ursache	Lösung
Niedrige Segment-HIT-Rate	`Vary: Origin` mit hoher Origin-Variabilität; defekte Normalisierung in `map`	Header und Regex in der `map`-Direktive prüfen
404 bei Segmenten nach Verlassen des Fensters	Gecachter 404 bei Segment, das aus dem Sliding Window gefallen ist	`proxy_cache_valid 404 0s` in der Segments-Location ergänzen
Playback-Start-Verzögerung 2–5 s	`proxy_cache_lock_timeout` überschreitet die Ziel-Latenz	Auf 1–2 s senken; `proxy_cache_use_stale updating` aktivieren
Manifest aktualisiert nicht	`proxy_cache_valid` überschreibt `max-age`	`proxy_cache_valid 200 1s` explizit setzen
Wachsendes TIME_WAIT am Upstream	`keepalive` fehlt im Upstream-Block	`keepalive 64`, `proxy_http_version 1.1` und `proxy_set_header Connection ""` hinzufügen
403 bei `/dash/.../<segment>.m4s` von ffmpeg	Der Client löst relative URLs gegen die Pre-Redirect-URL auf	Der Server gibt `<BaseURL>/h<sess>/</BaseURL>` aus (absolute Pfade); im aktuellen Build kompatibel
Lags, häufiges Rebuffering bei entfernten Clients	Niedriger effektiver TCP-Durchsatz aufgrund von Slow Start und Idle Restart bei großen RTTs (300 ms und mehr)	Tuning des Linux-Netzwerk-Stacks auf dem Origin: siehe 10.1

10.1. TCP-Tuning des Origin für High-RTT-Clients

Das Problem tritt bei Clients mit großem RTT zum Origin (z. B. 300 ms und mehr) auf, wenn die Stream-Bitrate nahe an der Kanalkapazität liegt. Symptome im Player (VLC, ffmpeg, dash.js) — häufiges Rebuffering, Warnungen wie `ES_OUT_SET_PCR called too late` (mit Anstieg von `pts_delay`), `buffer deadlock prevented`, Stream-Abbrüche. Auf dem Server wirkt der Client dabei normal, Fehler gibt es keine, und der Durchsatz auf `/data/stream/...` entspricht dem Eingangsstrom.

Ursache:

- **TCP slow start.** Jede neue TCP-Verbindung beginnt mit einem Congestion Window von etwa 14 KB und vergrößert es über mehrere RTTs. Bei einem RTT von 300 ms dauert das Erreichen des vollen Fensters 2-3 Sekunden. In dieser Zeit wird ein HLS/DASH-Segment von 5 s Dauer (4-6 MB) merklich langsamer als in Echtzeit heruntergeladen.
- **TCP idle restart.** Zwischen Segmentanfragen pausiert ein HLS-Pull-Modell-Client 4-5 s. Standardmäßig setzt der Linux-Kernel nach einer solchen Pause das Congestion Window der Verbindung auf das Initial cwnd zurück (Verhalten `net.ipv4.tcp_slow_start_after_idle=1`). Folglich beginnt die Keep-Alive-Verbindung beim nächsten GET die Übertragung erneut aus dem Slow Start — auch bei einer bereits aufgewärmten Sitzung.

Eine zusätzliche Verschärfung — das standardmäßige CUBIC-Congestion-Control kommt mit langen RTTs und Paketverlusten auf zwischengeschalteten Netzabschnitten schlecht zurecht.

Lösung — zwei sysctl-Parameter auf dem Origin:

```
# Keep congestion window across idle pauses inside keep-alive sessions.
sysctl -w net.ipv4.tcp_slow_start_after_idle=0

# Use BBR instead of CUBIC: better behaviour on long-RTT paths
# with mild packet loss; paces sending instead of bursting.
modprobe tcp_bbr
sysctl -w net.ipv4.tcp_congestion_control=bbr
```

Für eine dauerhafte Anwendung:

```
cat > /etc/sysctl.d/99-pss-net.conf <<EOF
net.ipv4.tcp_slow_start_after_idle = 0
net.ipv4.tcp_congestion_control    = bbr
EOF
sysctl --system
```

Den Haupteffekt liefert der erste Parameter (`tcp_slow_start_after_idle=0`). Er beseitigt direkt den erneuten Slow Start zwischen Segment-Anfragen innerhalb einer Keep-Alive-Verbindung. Der zweite (BBR) bietet zusätzliche Robustheit und gilt für alle neuen Verbindungen.

Das Tuning erfordert keinen Neustart von Perfect Streamer und gilt unmittelbar nach `sysctl -w` für alle neuen TCP-Verbindungen. Bestehende Verbindungen behalten das Congestion Control, mit dem sie aufgebaut wurden.

4.6.11 11. Sicherheit

11.1. Session URL

Eine URL der Form `/h<sess>/...` erfüllt die Funktion eines Sitzungs-Tokens — eine erneute Authentifizierung ist nicht nötig. Die Lebensdauer ist durch das idle timeout begrenzt (Wert 30 s). Bei Inaktivität wird die Sitzung von der cleaner-Aufgabe entfernt.

Anforderungen:

- HTTPS für alle OTT-Pfade (`/hls/`, `/dash/`, `/h<sess>/`) in Produktion
- Die Session-ID im `Location`-Header der 302 wird nicht gecacht (`no-cache, no-store`)

11.2. Rate limiting

```
limit_req_zone $binary_remote_addr zone=dash_top:10m  rate=5r/s;
limit_req_zone $binary_remote_addr zone=hls_top:10m   rate=5r/s;
limit_req_zone $binary_remote_addr zone=llhls_top:10m rate=5r/s;

server {
    location /dash/ {
        limit_req zone=dash_top burst=20 nodelay;
        proxy_pass http://pss_backend;
    }
    location /hls/ {
        limit_req zone=hls_top burst=20 nodelay;
        proxy_pass http://pss_backend;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
    location /llhls/ {
        limit_req zone=llhls_top burst=20 nodelay;
        proxy_pass http://pss_backend;
    }
}
```

Session-URLs (`/h<sess>/`) benötigen kein Rate Limiting — die Verarbeitung ist günstig, Antworten werden gecacht.

11.3. Caching von Fehlerantworten

```
proxy_cache_valid 200 60s;
proxy_cache_valid 301 302 0s;
proxy_cache_valid 404 403 0s;
proxy_cache_valid any 1s;
```

Verbietet das Caching von Redirects (eindeutiges sess in Location) und von Antworten mit Auth- oder Not-Found-Fehlern.

11.4. Einschränkung des Netzwerkzugriffs auf den Origin

Port 41972 (41982 für HTTPS) muss für externen Verkehr geschlossen sein. Zulässige Konfigurationen:

1. Perfect Streamer an `127.0.0.1` binden (bei lokalem nginx)
2. Firewall-Regel:

```
iptables -A INPUT -p tcp --dport 41972 ! -s 10.0.0.0/8 -j DROP
```

4.6.12 12. Integration mit Middleware

12.1. Das prefix-login-Modell

Perfect Streamer kann die Nutzeridentifikation über Prefix-Login an Middleware/Billing-Systeme delegieren. Ein externer Connector zum Billing-System ist im aktuellen Release nicht enthalten.

Konfiguration des Embedded-Benutzers:

```
{
  "id": 9,
  "login": "sub",
  "password": "xxx",
  "is-prefix": true,
  "max-conn-http-hls": 1,
  "accept-stream": [ ... ]
}
```

Mit `is-prefix: true` akzeptiert der Server URLs mit Logins der Form `<prefix><billing_user_id>`:

```
/dash/test1/sub42/xxx/index.mpd
/hls/test1/sub43/xxx/index.m3u8
```

12.2. Statistikformat

```
<clients>
  <client login-id="-1974387287" login="sub" match-login="sub42"
          sess-id="11331..." ott-type="dash" stream-id="10000" .../>
  <client login-id="-2147031294" login="sub" match-login="sub43"
          sess-id="11132..." ott-type="dash" stream-id="10000" .../>
</clients>
```

Das Feld `login-id` enthält den Hash des URL-Logins. `login` ist der konfigurierte Wert. `match-login` ist der vom Client genutzte URL-Login.

12.3. Einschränkungen von prefix-login

- **Gemeinsames Passwort.** Alle Teilnehmer des Prefix-Pools nutzen einen einzigen Passwortwert. Eine Kompromittierung des Passworts gewährt Zugriff auf jedes `<prefix><string>`.
- **ACL-Granularität.** `accept-stream` gilt für den gesamten Prefix-Pool; eine subscriber-spezifische ACL gibt es ohne externes Billing nicht.
- **Passwort-Rotation.** Eine Passwortänderung trennt alle aktiven Teilnehmer. Für eine schrittweise Umstellung sind vorübergehend zwei Prefix-Logins erforderlich.

4.6.13 13. WebVTT-Untertitel

Die Untertitelquelle ist DVB Teletext / DVB Subtitling aus dem Eingangs-MPEG-TS. In den Abschnitten **Media Information** oder **Original Media Information** müssen Teletext-Untertitel-Spuren vorhanden sein. Im Abschnitt **Analyzer** kann zusätzlich überprüft werden, dass Pakete der entsprechenden PIDs aktiv sind.

Für OTT HLS/DASH muss der OTT-Modus aktiviert sein (im *Peering/HLS* sind WebVTT-Untertitel nicht verfügbar). Im Abschnitt **Output # OTT** muss der Chunk-Zähler **OTT WebVTT buffer chunk count** einen Wert ungleich null aufweisen.

Zur Diagnose der Untertitel **Analyze** und **Trace** am Stream aktivieren. Beim Stream-Start sollte im Stream-Log Folgendes erscheinen:

```
Start Teletext subtitle decoder
[ttxsubdec] ttx: pid=331 magazine=8 page=0x88 lang=***
```

Im Folgenden wird der dekodierte Untertiteltext in das Log geschrieben.

13.1. URL der VTT-Segmente

Schema	URL	Inhalt
HLS master	`/hls/.../index.m3u8`	`#EXT-X-MEDIA:TYPE=SUBTITLES, GROUP-ID="subs",...,URI="/h<sess>/sub/<pid>/index.m3u8"`
HLS subtitle playlist	`/h<sess>/sub/<pid>/index.m3u8`	Liste `<keyHex>.vtt` mit `#EXTINF`
HLS-VTT-Segment	`/h<sess>/sub/<pid>/<keyHex>.vtt`	VTT mit HLS-spezifischer X-TIMESTAMP-MAP
DASH MPD AdaptationSet	in `index.mpd`	`contentType="text" mimeType="text/vtt"` + `<SegmentTemplate media="$Number$.vtt">`
DASH-VTT-Segment	`/h<sess>/sub/<pid>/<seq>.vtt`	VTT mit DASH-spezifischer X-TIMESTAMP-MAP

`<keyHex>` ist ein 16-stelliger Hex-CRC64 aus Segment-Startzeit, Stream-ID und Untertitelspur-PID. `<seq>` ist die dezimale laufende Nummer eines Untertitel-Stream-Chunks (die Untertitel-Nummerierung ist von der TS-Chunk-Nummerierung unabhängig).

4.7 DVR / Archiv

Ab Version 1.13 bietet **Perfect Streamer** einen integrierten DVR — ein persistentes Stream-Archiv auf der Platte, das parallel zur normalen OTT-Ausgabe (HLS / DASH) läuft. Das Archiv wird automatisch geschrieben, ohne separaten Prozess, und über dieselben OTT-URLs wie Live wiedergegeben — Unterschied ist nur der Query-Parameter.

Funktionen:

- Aufzeichnung jedes OTT-Streams ins Archiv auf dem gewählten Speicher.
- HLS-, DASH- und Low-Latency-HLS-Wiedergabe des Archivs (VOD) über dieselben URLs wie live (DASH und LL-HLS — über CMAF, im *OTT/HLS/LL-HLS/LL-Dash*).
- Untertitel-Unterstützung (WebVTT) — neben TS-Chunks geschrieben.
- Mehrere Speicher — ein Stream wird an einen gebunden; verschiedene Streams können auf verschiedene Platten schreiben.
- Automatische Bereinigung nach Aufbewahrungszeit und Speicherauslastung.
- EPG-aligned VOD — Archivausgabe per EPG-Ereignis-Bezug.
- Adaptives VOD — wird für adaptive Gruppen unterstützt.

DVR erfordert keine separate Lizenz. Aktivierung pro Stream durch Hinzufügen einer Speicherbindung.

DVR ersetzt das Live-Broadcasting nicht. Hat ein Stream ein Archiv, erhält der Client eine Live-Playlist mit identischem Verhalten wie ohne DVR. Das Archiv beginnt erst dann zu spielen, wenn der Client den VOD-Modus explizit über einen URL-Query-Parameter anfordert (siehe unten).

4.7.1 Speicher-Konfiguration

Ein Speicher ist ein Eintrag im Abschnitt **Configuration / DVR Storage**. Jeder Eintrag beschreibt ein Verzeichnis auf der Platte, in das PSS Archivdateien schreibt. Ein Stream nutzt einen Speicher.

Beim Hinzufügen eines Speichers werden konfiguriert:

Name — Anzeigename.

Dir Path — Pfad zum Verzeichnis auf der Platte. Nach Erstellung des Eintrags ist der Pfad **nicht änderbar** — um das Archiv zu verschieben, Eintrag löschen und neuen mit neuem Pfad anlegen. Bestehende Dateien werden beim Löschen **auf der Platte nicht angerührt**.

Max Usage, % — Schwellenwert der Auslastung (Standard 90 %). Bei Überschreitung startet die size-based Bereinigung (siehe unten). Minimum 1 %, Maximum 100 %.

Cleanup Interval, Sek — Periode der Bereinigungsaufgabe (Standard 10 Sek). Bei jedem Tick wird zunächst alles älter als die Aufbewahrungstiefe entfernt; danach — bei **Max Usage**-Überschreitung — alte Chunks.

Disk Pressure Grace, Sek — wie lange **Used % Max Usage** durchgehend überschreiten muss, bevor **Size-based cleanup** startet (Standard 60 Sek). Filtert kurze Spitzen.

Disk Pressure Cut, Sek — Obergrenze pro Bereinigungs-Tick: wie viele Videosekunden pro Stream auf einmal gelöscht werden dürfen (Standard 300 Sek). Der Rest geht in den nächsten Tick.

Disk Emergency Bytes — Schwellenwert für freien Speicher, unter dem der Speicher in *Error* übergeht und Aufzeichnung stoppt (Standard 2 GiB). Auto-Wiederherstellung bei freiem Platz ≥ 2 × diesem Wert.

Alarm Disk-Full Hysteresis, % — der Abstand unterhalb von **Max Usage**, bis zu dem die größenbasierte Bereinigung den Füllstand absenkt, damit **Used %** nicht direkt an der Schwelle schwankt (standardmäßig 2 %).

Die meisten Standardwerte passen für typische Installationen; Anpassungen sind meist nur für **Max Usage** und **Dir Path** nötig.

Pro Platte ist es sinnvoll, einen Speicher anzulegen. Werden auf derselben Platte mehrere Einträge mit verschiedenen Unterordnern angegeben, konkurrieren sie um freien Platz — die Platte ist gemeinsam, die Bereinigung läuft jedoch je Speicher.

4.7.2 Stream an Speicher binden

In den **Stream / OTT**-Einstellungen erscheint ein **DVR**-Abschnitt:

Storage — Dropdown verfügbarer Speicher; *0* bedeutet „Archiv für diesen Stream deaktiviert“.

Storage Hours — Archivtiefe für diesen Stream, in Stunden, von 1 bis 2160 (bis zu 90 Tage). Chunks, die älter als dieser Wert sind, werden bei jedem Tick der Bereinigungsaufgabe (**Rolling cleanup**) gelöscht.

Storage Min Hour — untere Schutzschwelle (Stunden). Die Bereinigung löscht nie Chunks jünger als dieser Wert, auch nicht unter **Max Usage**-Druck. Verwenden, wenn die Geschäftslogik eine garantierte frische Aufzeichnung verlangt, z. B. „letzte 2 Stunden immer vorhanden“.

Storage zur Laufzeit ändern:

- *0* setzen — deaktiviert das Archiv; vorhandene Chunks bleiben **erhalten**, neue werden nicht geschrieben. VOD-Sitzungen liefern fortan 404;
- Auswahl eines anderen Speichers — der Stream wird vom alten gelöst und schreibt in den neuen. Dateien vom alten Speicher werden nicht migriert.

Änderungen an **Storage Hours** und **Storage Min Hour** wirken sofort — beim nächsten Tick verwendet die Bereinigung den neuen Wert.

Nach Konfiguration schreibt der Stream automatisch das Archiv, sobald er in den Zustand *Running* wechselt.

4.7.3 VOD: Archivwiedergabe

Das Archiv wird über **dieselben URLs** wie Live HLS / DASH wiedergegeben (siehe Abschnitt *OTT-Service*) — nur der Query-Parameter unterscheidet sich.

URLs und Parameter

URLs für HLS, DASH und Low-Latency HLS (DASH und LL-HLS — über CMAF, erfordern den Modus *OTT/HLS/LL-HLS/LL-Dash*):

- http://host:port/hls/stream/login/password/index.m3u8
- http://host:port/dash/stream/login/password/index.mpd
- http://host:port/llhls/stream/login/password/index.m3u8

Ohne Query-Parameter — normales Live mit **Sliding Window**. Mit *t*-Parameter — VOD-Modus.

Parameter	Zweck
`t=<epoch>`	VOD-Startzeit (Unix epoch, Sek). *t=0* — ab Archivanfang. Das Vorhandensein von *t* (auch *t=0*) aktiviert den VOD-Modus.
`d=<sec>`	VOD-Fensterdauer, Sek. *d=0* oder Parameter fehlt — „bis zum aktuellen Moment“. Nur sinnvoll zusammen mit *t*.
`epg=<epoch>`	EPG-aligned VOD: Der Server findet selbst das zum angegebenen Moment aktive EPG-Ereignis und nimmt dessen *start* und *duration* als Fenstergrenzen. Inkompatibel mit *t* und *d* (server-seitige Ersetzung). Siehe unten.
`a`, `s`, `m`, `v`	Standardparameter für Live (siehe *OTT-Service*); `s` und `m` werden im VOD-Modus ignoriert.

Verhalten nach *t* und *d*:

t	d	Fenster	404-Bedingung
nein	—	live (Sliding Window)	—
0	keiner / 0	[Archivanfang, jetzt]	DVR nicht gebunden
0	> 0	[Archivanfang, +d]	DVR nicht gebunden
> 0	keiner / 0	[t, jetzt]	DVR nicht gebunden
> 0	> 0	[t, t+d]	DVR nicht gebunden

Grenzen-Normalisierung:

- *t* vor Archivanfang — start wird automatisch auf den ersten verfügbaren Chunk gezogen (Cleanup könnte schon getrimmt haben). Das ist **kein 404** — der Rest wird ausgeliefert.
- *t* in der Zukunft oder Fenster komplett vor dem Archiv — leere, aber gültige Playlist (HLS: nur Header + `EXT-X-ENDLIST`; DASH: `@type="static"`, `mediaPresentationDuration="PT0S"`).
- Chunks werden strikt anhand des Chunk-Startzeitpunkts ausgewählt, der in das halboffene Intervall *[t, t+d)* fällt — es gibt keine Teilsegmente.

VOD auf einem Stream ohne Archiv (oder dessen Speicher in *Error* ist) — **404** sofort auf Master-Playlist / MPD. Keine Sitzung wird angelegt.

Fehlertexte im 404-Body (in Server-Logs und HTTP-Body sichtbar):

- `VOD: stream not running` — Stream ist im Konfig, aber nicht in *Running*.
- `VOD: no DVR archive` — kein Speicher am Stream eingestellt oder Speicher in *Error*.
- `VOD: DVR detached` — Stream wurde zwischen Anfragen vom Speicher gelöst.
- `VOD: EPG event not found` — kein Ereignis für `?epg=` gefunden.

VOD-HLS-Playlist — **geschlossen** (Player sieht Dauer und kann spulen):

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-TARGETDURATION:6
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:5.000,
...
#EXT-X-ENDLIST
```

In der Live-Playlist fehlen diese Marker — das ist der einzige Unterschied.

VOD-DASH-MPD — **statisch**: `@type="static"`, festes `mediaPresentationDuration`, explizite `<SegmentURL>`. Live-DASH bleibt `@type="dynamic"`.

Hat das gewählte Intervall **Lücken** im Archiv (z. B. durch Aufzeichnungs-Neustart oder mittendrin getrimmten Cleanup), wird das DASH-MPD automatisch in mehrere `<Period>` aufgeteilt — eines pro durchgehende Spur. Player (VLC, dashjs, Shaka) spulen über die Periodengrenze ohne Sonderkonfiguration.

EPG-aligned VOD

Ist der Stream mit einer EPG-Quelle verbunden (Felder **EPG Source** und **EPG Channel** in den Stream-Einstellungen), kann der Client das Archiv **anhand eines Zeitpunkts innerhalb eines EPG-Ereignisses** anfordern:

- http://host:port/hls/stream/login/password/index.m3u8?epg=1778500000
- http://host:port/dash/stream/login/password/index.mpd?epg=1778500000

Der Server findet das beim angegebenen *epoch* aktive EPG-Ereignis und setzt dessen *start* und *duration* als VOD-Fenstergrenzen ein. Nützlich für Programmkataloge: Die UI kennt die Ereigniszeit, muss aber keine genauen Sek-/ms-Grenzen berechnen.

Ist der Stream nicht an EPG gebunden oder gibt es zu diesem Moment kein Ereignis — **404 ``VOD: EPG event not found``**. *t* und *d* werden bei vorhandenem *epg* ignoriert.

Adaptive Streams

Adaptive Gruppen (siehe HLS Adaptive Multistream) unterstützen dieselben VOD-Parameter:

- http://host:port/hls/adaptive/group/login/password/index.m3u8?t=0
- http://host:port/dash/adaptive/group/login/password/index.mpd?t=0

In die Master-Playlist (HLS) gelangen nur Varianten mit konfiguriertem DVR-Speicher. Varianten ohne DVR werden übersprungen (im Log `VOD: variant N has no DVR`); der Aufbau aus den verbleibenden Varianten läuft weiter.

In der DASH-Variante wird jede Qualität zu einem eigenen `<Representation>` innerhalb gemeinsamer `<Period>`; der Player kann zwischen Qualitäten ohne Manifest-Reopen wechseln.

Player-Verhalten

VOD-HLS / DASH aus dem Archiv läuft in Standard-Playern: VLC, hls.js, dashjs, Shaka, ffmpeg. Timeline-Spulen funktioniert.

Fordert der Player ein Segment an, das der Cleanup bereits entfernt hat, gibt der Server **404 für dieses Segment** zurück (nicht 500). VLC, hls.js, dashjs überspringen das Segment und spielen vom nächsten weiter.

Zusätzliche Möglichkeiten:

- **Schutz aktiver Sitzungen**: Solange eine VOD-Sitzung offen ist, löscht der Cleanup keine Chunks im Fenster (siehe unten).
- **Live-edge bridge**: Fordert ein Client in einer VOD-Sitzung ein Segment jenseits der rechten Grenze des VOD-Fensters an — z. B. nach Erreichen des Archivendes vorwärts in der Timeline — liefert der Server das Segment automatisch aus dem Live-Speicher. Keine Weiterleitungen und keine erneute Autorisierung.
- **Playlist-Cache**: Auf wiederholte Anfragen derselben VOD `index.m3u8` / `index.mpd` liefert der Server eine byteidentische Antwort — ohne Neuaufbau. Geeignet für CDN vor PSS.

4.7.4 Untertitel im Archiv

Trägt der Stream Untertitel (DVB Subtitling, Teletext oder WebVTT) und ist die Option **OTT WebVTT** aktiviert, werden Untertitel parallel zu den TS-Chunks archiviert — als `.vtt`-Dateien neben `.ts`.

Im Live-Modus enthält die Master-Playlist `EXT-X-MEDIA:TYPE=SUBTITLES`; im VOD-Modus liefert der Server eine **VOD-Untertitel-Playlist** (mit `ENDLIST`) und `.vtt`-Segmente unter denselben URLs.

Hinweise:

- **HLS**: Der `X-TIMESTAMP-MAP`-Header bleibt am Anfang jeder `.vtt` (per HLS-Spezifikation erforderlich).
- **DASH**: Der `X-TIMESTAMP-MAP`-Header wird im Flug entfernt (an absolute PCR gebunden, kollidiert mit dem DASH-Anker; sonst zeigt VLC leere Untertitel).
- In adaptiver Gruppe: Schreibt der aktive Sub-Stream keine Untertitel, liefern VTT-Segmente 404. Auf der nächsten Variante kann der Player wieder Untertitel erhalten.

Untertitel lassen sich entweder per **OTT WebVTT**-Option am Stream deaktivieren oder indem HLS im Modus *OTT/HLS* gar nicht aktiviert wird.

4.7.5 Bereinigung und Retention

PSS verwendet zwei Bereinigungsstrategien, die bei jedem Tick der Bereinigung laufen (Standard alle 10 Sek):

1. **Rolling cleanup** (pro Stream): Pro Stream werden Chunks älter als **Storage Hours** gelöscht. Läuft immer, auch bei halbleerer Platte.
2. **Size-based cleanup** (speicherweit): Wenn **Used %** durchgehend für **Disk Pressure Grace** Sek **Max Usage** überschreitet, werden die ältesten Chunks **proportional** über alle Streams des Speichers getrimmt. Pro Tick **Disk Pressure Cut** Sek Video je Stream. Nie Chunks jünger als **Storage Min Hour**.
3. **Emergency disk-full cut**: Fällt der freie Speicher unter **Disk Emergency Bytes**, läuft die Bereinigung aggressiv und kann auch sitzungsgeschützte Chunks löschen. Aufzeichnung stoppt, bis der freie Platz mit Hysterese × 2 wiederhergestellt ist.
4. **Orphan scan**: Auf der Platte können Dateien verbleiben, die nicht im Index erfasst sind (nach einem plötzlichen Stopp von PSS). Der Collector durchläuft die Stream-Unterverzeichnisse und löscht solche „vergessenen“ Dateien — der erste Lauf kurz nach dem Start des Dienstes, danach einmal pro Stunde und bei Auslösen von Disk Pressure. Als Schutz vor einem Race mit dem Schreiben werden Dateien jünger als 60 Sek. übersprungen.

Hinweis. Bereinigungsaufgaben laufen im Hintergrund; in der Regel ist kein Eingreifen nötig. Wächst das Archiv schneller als es getrimmt wird, **Storage Hours** auf den Streams senken oder **Disk Pressure Cut** erhöhen.

4.7.6 Schutz aktiver VOD-Sitzungen

Wenn ein Client eine VOD-Sitzung öffnet, wird auf jedem beteiligten Speicher ein „Schutzslot“ mit Fenster-Startzeit registriert. **Rolling**- und **size-based**-Cleanup berühren Chunks im Fenster offener Sitzung nicht. Der Slot wird beim Schließen (FIN, Timeout) automatisch freigegeben.

Das heißt:

- Hält ein Client eine VOD-Sitzung lange, kann er garantiert zu jedem Moment im Fenster spulen — Chunks „verschwinden nicht unter ihm“.
- Die Bereinigung per **Max Usage** kann **Used %** vorübergehend nicht auf die Schwelle drücken, solange eine Sitzung aktiv ist; verlässt der Client den Stream, holt die Bereinigung auf.
- **Emergency disk-full cut** und **Storage Min Hour** umgehen den Schutz: Ist die Platte fast leer, werden Chunks gelöscht und der Client erhält 404 für die betroffenen Segmente (der Player überspringt).
- Nach PSS-Neustart verschwinden Schutzslots — Bereinigung läuft sofort weiter.

4.7.7 Mehrere Speicher

Beliebig viele **DVR Storage**-Einträge sind möglich — je Verzeichnis / Platte einer. Streams werden unabhängig an verschiedene Speicher gebunden. Bereinigung und Schwellen (**Max Usage**, **Disk Pressure**) wirken per-storage.

Anwendungsfälle:

- **Tiering nach Wert**: SSD-Speicher für Premium-Kanäle mit großer Tiefe, HDD-Speicher hoher Kapazität für den Rest.
- **Eigene Platte für das Archiv**: damit die DVR-Aufzeichnung nicht mit Systemplatte oder Temp-Dateien konkurriert.
- **Projekttrennung**: eine Platte für Kanalpaket Nr. 1, eine andere für Nr. 2 — vereinfacht Migration und Audit.

Das Ändern der Stream-Bindung (Feld **Storage** in **Stream / OTT**) schaltet die Aufzeichnung im Betrieb um. Alte Dateien bleiben auf der bisherigen Platte unangetastet — sie lassen sich manuell löschen oder durch Rückumstellung wieder anbinden.

4.7.8 Speicherzustand und Monitoring

Im Abschnitt **Data / DVR Storage List** (sowie über API `GET /data/dvr-storage-list`) werden pro Speicher angezeigt:

- **State** — *Ready* / *Error* (plus den Zwischenzustand *Not Ready* bei einem gerade hinzugefügten Storage).
- **Total / Free / Used Bytes** — freier und belegter Speicherplatz auf der Platte.
- **Used %** — aktueller Auslastungsanteil.
- **Archived Bytes** — Gesamtgröße der im Index aller gebundenen Streams erfassten Chunks (ohne Orphan-Dateien).
- **Pressure Since Sec** — Moment (epoch), zu dem **Used %** im aktuellen Druckepisode erstmals **Max Usage** überschritten hat; *0* heißt „kein Druck jetzt“.
- **Active Task** — die gerade laufende Wartungs-Hintergrundoperation: *gc-orphans* (Entfernen verwaister Dateien), *disk-pressure-trim* (Kürzen unter Plattendruck) oder *none*, sowie wie viele Sekunden sie bereits läuft. Kurzlebige (< 2 s) Operationen flackern in der UI nicht.
- **Last cleanup** — eine Zusammenfassung der letzten Bereinigungsläufe: wie lange der vorherige Einsammelvorgang verwaister Dateien zurückliegt (und wie viele davon gelöscht wurden) sowie das letzte Kürzen unter Speicherplatzdruck (wie viel Platz freigegeben wurde).
- **Attached streams** — die Liste der angebundenen Streams; für jeden werden der Name, der Indikator einer laufenden Aufzeichnung (*active*), die konfigurierte Archivtiefe (*retention-hours*) und der tatsächliche Füllstand (*archived-sec* und *archived-bytes*) angezeigt. Die Summe der *archived-bytes* über alle Streams entspricht den **Archived Bytes** des gesamten Speichers.

Zustände:

- *Ready* — das Verzeichnis ist verfügbar, Aufzeichnung und Bereinigung laufen normal.

- *Error* — das Speicherverzeichnis ist nicht verfügbar (ausgehängt, fehlende Rechte, Dateisystemfehler) oder es ist kritisch wenig freier Platz auf der Platte; die Aufzeichnung ist gestoppt. Die Wiederherstellung erfolgt automatisch — sobald das Verzeichnis wieder verfügbar ist und genügend freier Platz vorhanden ist.

Ist der Speicher in *Error* gegangen, prüfen Sie, ob das Archivverzeichnis eingehängt ist und ob auf der Platte freier Speicher vorhanden ist — PSS verlässt diesen Zustand nicht von selbst, solange das Problem nicht physisch behoben ist.

Archivfüllstand im Zeitverlauf:

- Auf Stream-Ebene (**Data / Stream**) steht die Metrik **storage-gap-percent** zur Verfügung — der Anteil zeitlicher Lücken im angesammelten Archiv: *0 %* bedeutet durchgehende Aufzeichnung, höhere Werte bedeuten, dass es Lücken im Archiv gibt (Neustarts der Quelle, durch die Bereinigung gekürzte Abschnitte).
- Der Endpoint `GET /data/dvrstat` liefert ein Histogramm der Archivfüllung nach Zeitintervallen mit Markierung von Aufnahmeereignissen (Start/Stopp der Quelle, **PMT**-Wechsel, Scrambling-Umschaltungen, Index-Neuaufbau, Bereinigungslauf) und Untertitel-Aktivität — zur Darstellung der DVR-Archiv-Zeitleiste in der Admin-Oberfläche.

4.7.9 Schutz vor versehentlichem Datenverlust

Mehrere Operationen der Admin-Oberfläche führen zum **Verlust des aufgezeichneten Archivs** oder **Ende der VOD-Auslieferung**. Vor der Ausführung zeigt die Admin-UI eine modale Bestätigung:

- **DVR-Storage löschen** — alle gebundenen Streams verlieren VOD-Zugriff; Dateien bleiben auf der Platte, sind aber ohne Eintrag über PSS nicht erreichbar.
- **Stream auf anderen Speicher umstellen** — VOD über das alte Archiv funktioniert nicht mehr.
- **Stream vom Speicher lösen** (Storage = 0) — gleicher Effekt.
- **Max Usage senken** — kann size-based Bereinigung auslösen und alte Chunks löschen.
- **Storage Hours / Storage Min Hour senken** — kann Teile des Archivs aus dem Rolling-Fenster drängen.

Dies ist eine bewusste Produktentscheidung: Die Operation ist möglich, aber mit Bestätigung, und gelöschte Dateien bleiben auf der Platte (Restore aus Backup möglich). Das Archiv auf separatem Fileserver / RAID senkt das Risiko irreversiblen Verlusts deutlich.

4.7.10 Einschränkungen der aktuellen Version

- Eine vom Client geöffnete VOD-Sitzung **verfolgt** die Live-Edge **nicht** — wenn während der Sitzung neue Chunks ins Archiv kamen, muss der Client die Playlist neu anfordern (Standard aller HLS / DASH Player).
- Pro Platte ist ein Speicher sinnvoll — mehrere Einträge mit verschiedenen Unterordnern konkurrieren um freien Platz.
- Segmente werden per Hash adressiert (HLS über MPEG-TS) oder über die Vorlage `$Number$.m4s` mit gemeinsamer `init.mp4` (live DASH über CMAF) / über explizite `<SegmentURL>` auf `.m4s` (VOD DASH). Eine Änderung der Chunk-Größe zwischen live und VOD erfordert kein erneutes Öffnen der URL.

- Der Orphan-Scanner läuft stündlich; zur Beschleunigung PSS neu starten.

4.8 Stream-Operationen

Löschen. Zum Entfernen eines Streams in seine Einstellungen gehen und auf *Delete stream* klicken.

Klonen. Zum Klonen eines Streams in seine Einstellungen gehen und auf *Clone stream* klicken.

Sortierung. Klicken Sie auf die Schaltfläche *Sort* im Stream-Listen-Fenster, um die Stream-Sortierung zu konfigurieren. Geben Sie anschließend die gewünschte Reihenfolge an, indem Sie die Streams in der Liste nach oben oder unten ziehen. Klicken Sie auf *Save order*, um die festgelegte Sortierreihenfolge zu speichern, oder auf *Cancel*, um die Änderungen zu verwerfen.

Listenfilterung. Klicken Sie auf das Such-Symbol und geben Sie den Filter ein, um die Stream-Liste zu filtern. Mit dem Zurück-Pfeil heben Sie den Filter auf.

Gruppen-Operationen. Im Filter-Modus der Stream-Liste lassen sich mehrere Streams durch Aktivieren der Kontrollkästchen in der linken Spalte auswählen. Sind Streams ausgewählt, werden die Schaltflächen *Löschen* und *Klonen* für die ausgewählten Streams verfügbar.

4.8.1 Export und Import von Streams per Python-Skript

Konfigurations-Export/-Import läuft über eine .m3u-Playlist und ein Python-Skript.

Python 3 muss unter /usr/bin/python3 verfügbar sein.

4.8.2 Stream-Export und -Import über die Weboberfläche

Beim Klick auf die Schaltfläche *Playlist* in der Stream-Liste öffnet sich der Dialog zum Export der Streams in eine m3u8-Playlist. Exportiert werden nur die unter Berücksichtigung der angewandten Filter aktuell angezeigten Streams.

Einstellungen:

- **Host / IP** — Servername oder -adresse, die zum Aufbau der Stream-URLs verwendet wird.
- **Protocols** — Auswahl der Protokolltypen für den Export.
- **Login** — Auswahl des Kontos, das zur Bildung der Stream-URLs verwendet wird.
- **Use Display Names** — *Display name* des Streams anstelle des *Stream name* in der m3u8-Datei verwenden.

Die Playlist wird beim Klick auf *Download* als m3u8-Datei heruntergeladen.

Beim Klick auf die Schaltfläche **Import Playlist** wechselt der Dialog in den Modus zum Importieren von Streams aus einer m3u8-Playlist. Beim Import werden bestehende Streams nicht entfernt. Bei allen importierten Streams wird die Importzeit im Feld *Note* vermerkt.

Einstellungen:

- **Playlist** — Auswahl der Playlist-Datei für den Import.
- **Create Outputs** — Protokollauswahl für die automatische Erstellung von Ausgängen für importierte Streams.
- **Output Ports From** — Startportnummer für die Generierung der Ausgänge.
- **Output IP** — Auswahl der Schnittstelle, an die die Ausgangsstreams gebunden werden.
- **Tags** — Tags, mit denen importierte Streams markiert werden. Nützlich für Gruppenoperationen, z. B. das Löschen aller importierten Streams.

Ist im Feld **Playlist** eine zu importierende Datei angegeben, wird die Schaltfläche **Load Playlist** aktiv. Beim Klick auf **Load Playlist** wird die Playlist vorab geladen und das Parsing-Ergebnis in einer Tabelle angezeigt. Nach dem Parsing wird die Schaltfläche **Import Streams** aktiv — bei deren Klick werden die Streams importiert und Outputs für sie generiert.

4.9 Berichte und Diagnose

Im Bereich **Streams** werden alle **stream**s als Tabelle angezeigt. Die **Pause**-Aktion steht den Rollen **admin** und **restricted admin** zur Verfügung.

Für jeden **stream** stehen detaillierte Statistiken und Berichte zur Verfügung.

Peers — Liste aktiver Empfänger (Clients). Für jeden gibt es separate Statistiken.

4.9.1 Stream-Analyse

Der Streamer-Analyzer misst und analysiert verschiedene Parameter des MPEG-TS-Streams, was eine Qualitätsbewertung ermöglicht.

Input speed — Stream-Rate (Bitrate) in `kbps`. Ändert sich nach der Filterung anhand der PCR-Marken. Wird als Diagramm dargestellt, das auch die Ausgangsbitrate nach dem Synchronisierer zeigt.

Raw data speed — die Empfangsdatenrate über das gewählte Protokoll. **Overhead** — der prozentuale Aufschlag für den Protokolloverhead.

CC errors — Paketverluste (CC, Discontinuity). Es werden Intervallzähler sowie ein kumulativer Zähler über die Stream-Uptime angezeigt. Zusätzlich wird eine grafische Historie dargestellt.

Scrambled — Zähler verschlüsselter MPEG-TS-ES-Pakete. Werte ≠ 0 bedeuten Ausfälle beim Entschlüsseln verschlüsselter Kanäle.

Sync by — Synchronisationsquelle — PCR.

PCR interval — Intervall zwischen PCR-Marken. Empfohlen: nicht mehr als 50 ms.

PCR jitter — Maß für die Synchronisationsgenauigkeit des Ausgangsstreams; gemessen als Differenz zwischen PCR und Echtzeit.

Analyze PCR PMT Gap — wird über eine eigene Einstellung aktiviert. Analysiert wird die Abweichung zwischen PCR und PTS/DTS für jeden ES. Die Historie wird grafisch dargestellt. Eine zu große Abweichung kann auf Playern mit kleinem Synchronisationspuffer zu Problemen führen. Die Analyse wird über die Einstellung **Analyze PAT/PMT/KF** aktiviert.

PAT interval und **PMT interval** — Intervall zwischen PAT-(PMT-)Tabellen. Empfohlen: nicht mehr als 500 ms. Aktivierung der Analyse über **Analyze PAT/PMT/KF**.

Key Frame interval (GOP-Intervall) — das Intervall zwischen Keyframes (GOP-Anfängen). Für Player, die an beliebiger Stelle in den Stream einsteigen, werden maximal 1 s empfohlen. Die Analyse wird über die Einstellung **Analyze PAT/PMT/KF** aktiviert.

IDR interval — das Intervall zwischen IDR-Frames, die echte Random-Access-Punkte sind (*SPS / PPS / IDR*). Es wird nur angezeigt, wenn die Quelle mit IDR sendet. Stimmt **IDR interval** ungefähr mit **Key Frame interval** überein, hat der Stream *closed-GOP*: Jeder GOP beginnt mit einem vollständigen Einstiegspunkt, und der Player kann ab jedem Segment starten. Fehlt die Metrik, ist der Stream *open-GOP* ohne IDR — es gibt Keyframes, sie sind aber keine vollständigen Einstiegspunkte. Aus diesem Verhältnis ist der GOP-Strukturtyp der Quelle sofort ersichtlich. Die Analyse wird über die Einstellung **Analyze PAT/PMT/KF** aktiviert.

PAT/KF interval — misst das mittlere Intervall zwischen Anfang und Ende der PAT/PMT/SPS/PPS/KF-Sequenz. Davon hängt die Startzeit der Wiedergabe für Player ab, die an beliebiger Stelle in den Stream einsteigen. Gemessen wird ab dem Beginn des KF im Stream; die tatsächliche Startzeit des Players ist daher länger. Die Analyse wird über die Einstellung **Analyze PAT/PMT/KF** aktiviert.

Aktivieren von **Analyze PAT/PMT/KF** schaltet den Stream-Analyzer dauerhaft ein, was die CPU-Last erhöht.

4.9.2 Jitter-Steuerung

Für die Jitter-Steuerung gibt es im Stream mehrere Einstellungen:

- **Jitter Compensation Delay (ms)** — Funktion zur Kompensation des Netzwerk-Jitters; legt die Buffergröße fest. Ausführliche Beschreibung: https://forum.pstreamer.tv/viewtopic.php?t=25
- **Jitter Auto sync** — setzt 2000 ms für TCP-basierte Protokolle (HTTP, HLS); für UDP-Protokolle bleibt der Wert bei 500 ms.
- **Limit PCR gap (ms)** — prüft, wie weit der PCR springen darf; bei Überschreitung erfolgt eine Resynchronisation.

Treten nach dem Transkoder *PCR Accuracy*-Fehler auf, müssen Sie für den codierten Stream über *stuffing* eine gleichmäßige Bitrate auf folgendem Pfad festlegen: „**input — transcoder — Align Total Bitrate**“. Die Geschwindigkeit muss garantiert höher als die Bitrate von Video und Audio gewählt werden.

4.9.3 PCR drift

Die Metrik *PCR drift* misst die systematische Abweichung der Quelltaktfrequenz gegenüber der Echtzeit und wird in *ppm* (parts-per-million) angegeben. Im Gegensatz zum momentanen *PCR jitter*, der das Zittern einzelner Zeitstempel erfasst, charakterisiert *PCR drift* gerade den Langzeit-Drift des Encoder-Quarzes — eine stetige Abweichung, die sich über Minuten und Stunden akkumuliert.

Die Messung erfolgt durch lineare Regression über Paare (kumulierte *PCR*-Zeit in Sekunden, Empfangszeit in Sekunden). Das Messrauschen sinkt mit wachsendem Fenster wie `1/T^1.5`, sodass ein belastbares Urteil nur über lange Intervalle möglich ist. Das Fenster wird automatisch bis auf 300 *s* (**TR 101 297**) erweitert, danach wird eine gleitende Neuverankerung mit einem Limit von 6 *Stunden* angewendet — dadurch wird ein Präzisionsverlust durch Überlauf der `float64`-Mantisse ausgeschlossen.

In der **XML**-Statistik des Streams werden exportiert:

- `pcr-drift-ppm` — aktuelles Regressionsurteil;
- `pcr-drift-window-s` — Länge des Fensters, über das das Urteil berechnet wurde;
- `pcr-drift-samples` — Anzahl der Punkte in der Regression.

Die in **ISO/IEC 13818-1** §2.4.2.1 festgelegte Toleranz beträgt ±30 *ppm*. Bei anhaltender Überschreitung (siehe Abschnitt zu Alarmen unten) wird ein separates Attribut `pcr-drift-alert` gesetzt, und nach 300 *s* ununterbrochener Verletzung — `pcr-drift-alert-acceptance` (Abnahmestufe gemäß **TR 101 297**).

Die Auslagerung von *PCR drift* in ein eigenes Attribut (statt in das gemeinsame `tr101290-alert`) ist beabsichtigt: Quarzdrift ist ein Problem des Upstream-Encoders, nicht des Empfängers, und der Betreiber muss es getrennt sehen, nicht mit Transportintegritätsfehlern vermischt.

4.9.4 PCR accuracy

Die Metrik *PCR accuracy* (Abschnitt P2.3 in **TR 101 290**) misst die Genauigkeit der Einfügung von *PCR*-Zeitstempeln in den Transportstrom. Laut Spezifikation muss der Wert jedes *PCR* mit dem tatsächlichen Zeitpunkt seines Durchgangs durch den Multiplexer mit einer Abweichung von höchstens ±500 *ns* übereinstimmen.

Die Messung wird als *Residual* einer linearen Regression auf einem gleitenden Fenster von 30 *s* ermittelt. Die Fenstergröße ist so gewählt, dass der gesamte *PCR*-Wiederholzyklus (≤40 *ms* laut Spezifikation) mit großem Reservepuffer erfasst wird, ohne die schnelle Reaktion auf eine Verschlechterung der Einfügequalität zu opfern.

In der **XML**-Statistik des Streams wird `pcr-accuracy-max-ns` exportiert — der Spitzenwert des Residuals über das Fenster. Bei Überschreitung von ±500 *ns* wird dem gemeinsamen Attribut `tr101290-alert` das Token `pcr-acc` hinzugefügt.

Die Messung ist nur für **CBR**-Multiplexe sinnvoll: bei **VBR** können die Intervalle zwischen *PCR* vom errechneten linearen Trend abweichen, ohne den Standard zu verletzen. Der Analysator deaktiviert das Token `pcr-acc` automatisch für Streams, die als *vbr* klassifiziert sind (siehe Abschnitt zum Modus-Detektor).

4.9.5 PCR-Drift-Kompensator

Liegt der *PCR drift* der Quelle innerhalb der **ISO/IEC 13818-1**-Toleranz, ist er jedoch nicht null, so „läuft“ der Ausgangsstrom langsam gegenüber der Netzwerkzeit des Empfängers weg. Bei einem Player mit kleinem Puffer äußert sich dies als akkumulierender Synchronisationsverlust oder periodische Wiedergaberuckler.

Der Kompensator beseitigt den Drift ohne harte Resynchronisation: ausgehende Pakete erhalten einen Mikroversatz von ~11.1 *µs* (Dauer eines einzelnen 188-Byte-*TS*-Pakets bei 100 *Mbps*), und die Entscheidung „ist ein Versatz nötig“ wird anhand der tatsächlichen Differenz zwischen Netzwerkzeit und *PCR*-Takt getroffen. Harte Resyncs (`Limit PCR gap`) bleiben als Fallback für Stromabbrüche bestehen — der Kompensator arbeitet auf einer feineren Skala und erhält die Kontinuität.

Einstellungen unter **stream**:

- *Sync Drift Compensation* — aktiviert/deaktiviert den Kompensator. Standardmäßig aktiviert.

- *Sync Drift Soft Window* — weiches Fenster, innerhalb dessen Korrekturen einzeln angewendet werden (1–60000 *ms*, Standard 500). Die obere Grenze ist durch `8 × Sync Disc Window` begrenzt, ab dem ein harter Resync als fällig betrachtet wird.

In der **XML**-Statistik wird `slew-adjust-count` exportiert — der Zähler der Korrekturen seit `reset-stat`. Ein abrupter Anstieg dieses Zählers bedeutet, dass die Quelle entweder ihren Toleranzbereich verlassen hat oder einen instabilen Quarz besitzt.

4.9.6 T-STD Video-Pufferanalysator

Das **T-STD**-Modell (*Target Decoder*) ist in **ISO/IEC 13818-1** §2.4.2 beschrieben. Es ist das Referenzmodell des Empfänger-Puffers: wird es eingehalten, ist garantiert, dass der Strom auf jedem konformen Hardware-Decoder wiedergegeben werden kann.

Der Analysator modelliert den `MBn`-Puffer (Multiplex-Buffer für Video) und prüft, dass:

- der Puffer nicht überläuft (overflow → der Encoder muss Frames verwerfen);
- der Puffer nicht leerläuft (underflow → der Decoder zeigt eine Pause oder Artefakte).

Wird durch die Einstellung *Analyze T-STD video* aktiviert (standardmäßig deaktiviert — verursacht CPU-Last im heißen Verarbeitungspfad).

Unterstützte Video-ES-Typen (`stream_type`):

- `0x01` / `0x02` — MPEG-2 video, Kapazität 750 *KB*;
- `0x10` — MPEG-4 part 2, Kapazität 750 *KB*;
- `0x1B` — H.264 / AVC, Kapazität 3 *MB*;
- `0x24` / `0x25` — HEVC, Kapazität 3.75 *MB*.

Die Abflussrate (*drain rate*) wird adaptiv an die tatsächliche Video-Bitrate angepasst: verwendet wird ein exponentiell gewichteter gleitender Mittelwert (EMA) mit `α=0.2` über ein 5-*s*-Fenster. Ohne Adaption liefert das Modell auf jedem *VBR*-Video schnell falsche Underflows.

Der Puffer wird nach **PCR**-Takten (90 *kHz*) entleert, nicht nach der Hostsystemzeit — dies macht die Analyse robust gegenüber Betriebssystem-Pausen und CPU-Lastschwankungen. Die Host-Echtzeit wird ausschließlich zur Verifikation des Drift-Kompensators verwendet, nicht für T-STD.

In der **XML**-Statistik werden exportiert:

- `tstd-video-cap` — Modellkapazität in Bytes;
- `tstd-video-drain-bps` — aktuelle adaptierte Abflussrate, *bytes/s*;
- `tstd-video-overflows` — Überlaufzähler seit `reset-stat`;
- `tstd-video-underflows` — Unterlaufzähler;
- `tstd-video-max-fill` — Spitzenfüllung in Bytes.

Bei jedem von Null verschiedenen Zähler (`overflows > 0` oder `underflows > 0`) wird dem gemeinsamen `tr101290-alert` das Token `tstd-video` hinzugefügt. Wie bei `pcr-acc` gilt das Token nur für **CBR**-Streams — bei einem **VBR**-Multiplex sind transiente Puffereinbrüche zulässig.

4.9.7 Multiplex-Bitratenmodus-Detektor

Ein Teil der **TR 101 290**-Tests ist nur im **CBR**-Modus (konstante Multiplex-Bitrate) sinnvoll. Um keine falschen Alarme an **VBR**-Kanälen auszulösen, bestimmt der Analysator automatisch den Quellmodus und exportiert das Urteil in das Attribut `bitrate-mode-detected`.

Algorithmus: Vergleich der mittleren Bitraten über zwei Fenster — 5 *s* und 60 *s*. Übersteigt die Abweichung 20 %, wird der Strom als `vbr` markiert; bleibt sie darunter, als `cbr`. Bis genügend Statistik akkumuliert ist (~70 *s* ab Start oder `reset-stat`), ist das Urteil `unknown` und die **CBR**-only Tests (`pcr-acc`, `tstd-video`) sind vorübergehend unterdrückt.

Attributwerte:

- `cbr` — konstante Bitrate, alle Tests aktiv;
- `vbr` — variable Bitrate, `pcr-acc` und `tstd-video` sind deaktiviert;
- `unknown` — unzureichende Daten, **CBR**-only Tests sind ausgesetzt.

4.9.8 TR 101 290 Alarme

Das Sammelattribut `tr101290-alert` vereint mehrere Konformitätsdetektoren nach **TR 101 290**. Spricht zum aktuellen Zeitpunkt mindestens einer an, enthält das Attribut eine durch Leerzeichen getrennte Token-Liste; spricht keiner an, fehlt das Attribut im **XML**.

Mögliche Token und ihre Bedeutung:

- `pcr-int` — Überschreitung des Intervalls zwischen *PCR* (Abschnitt 5.2.4 von **TR 101 290**, Grenzwert 40 *ms*, Empfehlung ≤25 *ms*);
- `pcr-acc` — Überschreitung der *PCR*-Einfügegenauigkeit (Abschnitt 5.2.5, ±500 *ns*, **CBR**-only);
- `pcr-disc` — ein harter Resync über *PCR* wurde erkannt (Abschnitt 5.2.4, `PCR_discontinuity_indicator_error`);
- `pat-int` — Überschreitung des *PAT*-Intervalls (Abschnitt 5.1.3, Grenzwert 500 *ms*, erfordert *Analyze PAT/PMT/KF*);
- `pmt-int` — Überschreitung des *PMT*-Intervalls (Abschnitt 5.1.5, Grenzwert 500 *ms*, erfordert *Analyze PAT/PMT/KF*);
- `tstd-video` — ein Über- oder Unterlauf des **T-STD**-Modells wurde erkannt (Abschnitt 5.3.16, erfordert *Analyze T-STD video*, **CBR**-only).

Um ein Flackern der Anzeige bei kurzen Spitzen zu vermeiden, wird eine Debounce-Logik angewendet: ein Token gelangt nur dann in `tr101290-alert`, wenn es mindestens 30 der letzten 60 Sekunden lang ausgelöst hat. Dies gilt einheitlich für alle Token.

Zusätzlich wird `tr101290-alert-acceptance` exportiert — eine Kopie des Attributs, in die ein Token erst nach 300 *s* ununterbrochener Verletzung gelangt (Abnahmestufe gemäß **TR 101 297**). Es ist die „endgültige“ Sammelmetrik für die technische Abnahme des Kanals.

Die `pcr-drift`-Anzeige wird absichtlich in ein eigenes `pcr-drift-alert` (+ `pcr-drift-alert-acceptance`) ausgelagert. Quarzdrift ist ein Problem des Upstream-Encoders, sie ist unabhängig von der Transportintegrität und muss separat klassifiziert werden.

4.9.9 KI-Reklamationsassistent

Falls bei einem Kanal *TR 101 290*- oder *PCR drift*-Alarme ausgelöst werden, kann der Betreiber mit einem einzigen Befehl einen fertigen englischsprachigen Prompt für einen KI-Chat abrufen, der ein formelles Reklamationsschreiben an den Upstream-Stream-Anbieter formuliert.

Endpoint: `GET /data/stream/<id>/ai-complaint-prompt`. Verfügbar mit der Rolle *Viewer* (wie alle GET-Anfragen unter `/data/*`). Liefert `text/plain; charset=utf-8`. Für **MPTS** wird 404 zurückgegeben — die Metriken *T-STD* / *PCR drift* sind nur auf **SPTS** anwendbar.

Der Prompt ist mit jedem modernen KI-Chat kompatibel: „ChatGPT“, „Claude“, „DeepSeek“, „Qwen“, „Doubao“. Der Ton des Schreibens ist sachlich, ohne Anschuldigungen; am Ende des Prompts wird die Sprache des fertigen Dokuments wählbar gemacht: Englisch, Russisch, vereinfachtes Chinesisch oder jede andere nach Wahl des Betreibers.

Inhalt des Prompts:

- Name und gemessene Parameter des Streams;
- die Liste aller aktiven *TR 101 290*- und *PCR drift*-Token mit den zugehörigen Standardverweisen;
- Zahlenwerte und Schwellwerte;
- die Auswirkung jedes Fehlers auf der Decoder-Seite.

Was **nicht** in den Prompt aufgenommen wird: Streamname, ID, Quell-URI. Diese Felder werden durch den Platzhalter `<Stream Designation>` ersetzt — der Betreiber trägt sie vor dem Versand selbst ein, damit sie nicht an einen Drittanbieter-KI-Dienst gelangen.

4.9.10 System Monitor

Überwachung der wichtigsten Systemparameter.

4.9.11 Mosaic

Funktion zur Screenshot-Aufnahme aus dem Eingangsstream. Wird pro **stream** einzeln aktiviert.

Falls nicht benötigt, kann sie zur Ressourcenschonung im Bereich **Settings/Server Settings** vollständig deaktiviert werden.

4.10 Verwaltung

Im Bereich **Configuration/Administration/Administrators List** werden Benutzer für den Zugriff auf die Weboberfläche (eingebauter HTTP-Server) angelegt.

Benutzern werden Rollen zugewiesen:

admin — Vollzugriff.

restricted admin — Einstellungen sind nicht verfügbar; es kann nur **pause** geändert werden.

viewer — Zugriff nur im Lesemodus.

4.10.1 Sicherung der Einstellungen

Für ein Backup der Einstellungen den Inhalt des Ordners */opt/pss/config* sichern.

Zur Wiederherstellung den Dienst stoppen und den Inhalt von */opt/pss/config* ersetzen.

4.10.2 Verhalten beim Start und bei Konfigurationsfehlern

Beim Start versucht der Dienst nacheinander, die Konfigurationsdateien aus dem Verzeichnis */opt/pss/config* zu laden:

1. *pss.json* — Hauptkonfigurationsdatei.
2. *pss_back.json* — Sicherungskopie der vorherigen funktionierenden Konfiguration.
3. *pss_default.json* — Standardkonfiguration, die mit dem Paket ausgeliefert wird.

Es wird die erste erfolgreich geladene Datei verwendet. Sind alle drei Dateien nicht vorhanden oder beschädigt, startet der Dienst mit leeren Einstellungen; in diesem Fall sind ein manuelles Setzen des Administratorpassworts und ein Neustart erforderlich.

Strukturell beschädigte Konfigurationsdatei. Wenn *pss.json* einen JSON-Syntaxfehler, unbekannte Schlüssel, einen falschen Werttyp oder eine Eindeutigkeitsverletzung enthält (zum Beispiel zwei Streams mit derselben *id*), verschiebt der Dienst die beschädigte Datei in das Archiv */opt/pss/config/bad/* unter einem Namen der Form *pss_YYYYMMDD_HHMMSS.json* (Datum und Uhrzeit des Starts). Anschließend setzt der Dienst die Ladeversuche in gewohnter Reihenfolge fort und speichert die funktionierende Konfiguration aus *pss_back.json* oder *pss_default.json* erneut in *pss.json*. Details (Schlüsselname, Fehlerbeschreibung, Dateiname im Archiv) werden in das Dienstprotokoll geschrieben.

In das Archiv gelangt nur die Haupt-*pss.json*. Die Dateien *pss_back.json* und *pss_default.json* werden bei Beschädigung nicht archiviert — die Logeinträge reichen zur Diagnose aus, und die Dateien selbst bleiben an Ort und Stelle und können manuell korrigiert werden.

Existiert beim nächsten Laden im Verzeichnis */opt/pss/config/bad/* bereits eine Datei mit demselben Zeitstempel (zum Beispiel bei schnellen Neustarts innerhalb derselben Sekunde), wird sie überschrieben.

Numerische Werte außerhalb des zulässigen Bereichs. Enthält die Konfigurationsdatei einen numerischen Wert, der kleiner als das zulässige Minimum oder größer als das zulässige Maximum für diesen Parameter ist, verwirft der Dienst die Datei nicht vollständig. Stattdessen wird eine Warnung in das Protokoll geschrieben, die den Parameternamen, den gelesenen Wert und die angewandte Grenze nennt; der Wert selbst wird auf die nächstgelegene zulässige Bereichsgrenze (Minimum oder Maximum) gesetzt. Nach Abschluss des Ladens speichert der Dienst *pss.json* automatisch mit den korrigierten Werten erneut, sodass diese Warnungen bei einem erneuten Start nicht mehr erscheinen.

Dieses Verhalten gilt nur beim erstmaligen Laden der Konfigurationsdatei. Werden die Einstellungen über die Weboberfläche oder eine externe API geändert, werden Werte außerhalb des zulässigen Bereichs weiterhin mit einem Fehler abgelehnt — ohne automatische Korrektur.

4.10.3 Anbindung externer Monitoring-Systeme

pss-metrics — universeller Metriken-Exporter für Perfect Streamer

Ein einzelnes Python-3-CLI-Skript, das Statistiken über die HTTP-API des PSS-Webservers abruft und die Ausgabe für die gängigsten Monitoring-Systeme formatiert:

- Zabbix (UserParameter, Low-Level Discovery, zabbix_sender trapper)
- Prometheus (Text-Expositionsformat für den textfile collector)
- InfluxDB / Telegraf (Line Protocol oder JSON für den exec-Input)
- Universelles JSON für beliebige Skripte und Nagios-artige Statusprüfungen

Der Exporter ist eine einzelne, in sich geschlossene Datei ohne Drittanbieter-Abhängigkeiten und nutzt ausschließlich die Standardbibliothek von Python 3.6+ (*urllib*, *xml.etree*, *json*, *argparse*).

Dateien

```
pss-metrics.py                    main CLI (executable)
userparameter_pss.conf.example    UserParameter template for Zabbix
```

Einrichtung

Der Exporter wird unter */opt/pss/monitoring/pss-metrics.py* ausgeliefert. Stellen Sie sicher, dass Python 3.6 oder neuer installiert ist:

```
# RHEL / Rocky / AlmaLinux
yum install -y python3

# Debian / Ubuntu
apt-get install -y python3
```

Es werden keine zusätzlichen Pakete benötigt.

Konfiguration

Standardmäßig verbindet sich *pss-metrics* mit *http://127.0.0.1:43971* und ermittelt den Port automatisch aus */opt/pss/config/pss.json* (oder */opt/pss/config/pss_default.json*). Parameter können über Umgebungsvariablen, die Datei */etc/pss-metrics.conf* (Format *Schlüssel=Wert*) oder Kommandozeilen-Flags überschrieben werden. Reihenfolge: CLI > env > Datei > Standardwerte.

Unterstützte Variablen:

```
PSS_URL          full URL, e.g. http://10.0.0.1:43971         (auto by default)
PSS_USER         web server login (if authorization is enabled)
PSS_PASS         web server password
PSS_TIMEOUT      HTTP timeout, in seconds                      (default 5)
PSS_CACHE_DIR    cache directory                               (default /run/pss-
↪metrics)
PSS_CACHE_TTL    cache TTL, in seconds                         (default 10)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
PSS_CA_BUNDLE    path to CA bundle for HTTPS
PSS_INSECURE     1 – disable TLS certificate verification
PSS_VERBOSE      1 – log requests to stderr
```

Cache: jeder Lauf führt höchstens einen HTTP-GET pro Endpoint innerhalb des TTL-Fensters aus. Bei TTL=10 s erzeugen selbst hunderte UserParameter-Abfragen pro Minute nur ~6 HTTP-Anfragen pro Minute an PSS.

Schnellstart

Statusprüfung (Nagios-Exit-Codes):

```
pss-metrics.py health
# OK: total=42 running=15 stopped=27 unhealthy=0 version=1.12.2.430d
```

Zabbix-LLD-Erkennung:

```
pss-metrics.py discover streams
pss-metrics.py discover inputs --running-only
pss-metrics.py discover outputs
```

Eine einzelne Metrik abrufen (für Zabbix UserParameter):

```
pss-metrics.py get summary.running
pss-metrics.py get stream.10031.bitrate
pss-metrics.py get input.10031.1.speed1
pss-metrics.py get output.10031.1.speed
pss-metrics.py get sysmon.cpu.self-usage
pss-metrics.py get server.server-version
```

Vollständiger Export:

```
pss-metrics.py dump --format=json
pss-metrics.py dump --format=prometheus
pss-metrics.py dump --format=influx
pss-metrics.py dump --format=zabbix-trapper --zabbix-host=streamer-01
```

Metrik-Pfade

pss-metrics get akzeptiert einen mit Punkten getrennten Pfad. Leere Ausgabe bedeutet „Wert fehlt“ (zum Beispiel existiert die Metrik nur für laufende Streams).

```
server.<attr>                 e.g. server.server-version, server.uptime
summary.<key>                 total | running | stopped | unhealthy |
                              input_bitrate_kbps | output_bitrate_kbps
sysmon.cpu.<attr>             self-usage | total-usage | cores
sysmon.memory.<attr>          self-usage-kb | available-kb | total-kb
sysmon.netbw.<iface>.<attr>   rx-bw | tx-bw  (interface name as in XML)
stream.<id>.<attr>            any <stream> attribute
input.<sid>.<iid>.<attr>      any <input> attribute
output.<sid>.<oid>.<attr>     any <output> attribute
```

Nützliche Stream-Attribute (aus /data/stream/detail):

```
stream:  state, state-str, bitrate, thread-usage, mpts
input:   speed1, recv-bytes, recv-packets, recv-err,
         stat-disc, stat-disc1, stat-scrambled, stat-scrambled1,
         health-state-good, health-status, check-status
output:  speed, sent-bytes, sent-packets, sent-err, uri, type
```

Integration mit Zabbix

Zwei Szenarien werden unterstützt — wählen Sie das für Ihre Umgebung passende.

1) Statischer UserParameter + LLD (Zabbix-Agent v1 / v2)

   Kopieren Sie *userparameter_pss.conf.example* nach */etc/zabbix/zabbix_agentd.d/pss.conf,* starten Sie *zabbix-agent* neu und importieren Sie auf dem Server ein Template mit LLD-Prototypen, die die *pss.discover*-Schlüssel verwenden. Beispielbindungen:

   ```
   UserParameter=pss.discover[*],/opt/pss/monitoring/pss-metrics.py discover $1
   UserParameter=pss.get[*],/opt/pss/monitoring/pss-metrics.py get $1
   UserParameter=pss.health,/opt/pss/monitoring/pss-metrics.py health
   ```

   Auf dem Zabbix-Server:

   ```
   Discovery rule key:        pss.discover[streams]
   Item prototype keys:       pss.get[input.{#STREAM_ID}.1.speed1]
                              pss.get[stream.{#STREAM_ID}.bitrate]
                              pss.get[summary.unhealthy]
   ```

2) Trapper (Push) über zabbix_sender

   Per Timer (cron / systemd) ausführen und die Ausgabe in eine Pipe leiten:

   ```
   /opt/pss/monitoring/pss-metrics.py dump --format=zabbix-trapper \
       --zabbix-host="$(hostname)" \
     | zabbix_sender -z zabbix.example.com -i -
   ```

Integration mit Prometheus

Zwei Varianten.

a) Textfile collector (empfohlen für One-Shot-Umgebungen).

   Periodischen Export per systemd-timer oder cron ausführen:

   ```
   */1 * * * * /opt/pss/monitoring/pss-metrics.py dump --format=prometheus \
       > /var/lib/node_exporter/textfile_collector/pss.prom.$$ \
       && mv /var/lib/node_exporter/textfile_collector/pss.prom.$$ \
             /var/lib/node_exporter/textfile_collector/pss.prom
   ```

   node_exporter stellt die Datei über *–collector.textfile.directory* bereit.

b) Direkter Scrape über einen kleinen Wrapper (z. B. *socat* + *pss-metrics dump*) oder einen beliebigen HTTP-Proxy eines Drittanbieters.

Integration mit Telegraf / InfluxDB

Telegraf *inputs.exec*:

```
[[inputs.exec]]
  commands = ["/opt/pss/monitoring/pss-metrics.py dump --format=influx"]
  interval = "10s"
  timeout  = "5s"
  data_format = "influx"
```

Für den JSON-Parser verwenden Sie *–format=json* und konfigurieren *data_format = „json“* mit Feldpfaden.

HTTPS und Authentifizierung

Wenn der PSS-Webserver hinter HTTPS läuft oder durch ein Passwort geschützt ist:

```
PSS_URL=https://streamer.example.com:8443 \
PSS_USER=monitor PSS_PASS=secret \
pss-metrics.py health
```

Selbstsignierte Zertifikate: setzen Sie *PSS_INSECURE=1* (nicht empfohlen) oder geben Sie *PSS_CA_BUNDLE=/path/to/ca.pem* an.

Rückgabewerte

pss-metrics folgt der Nagios-Konvention:

```
0  OK
1  WARNING       (e.g. running streams report unhealthy)
2  CRITICAL      (PSS is unreachable)
3  UNKNOWN       (invalid arguments / internal error)
```

get und *dump* geben eine leere Zeile aus und beenden mit Code 0, wenn die angeforderte Entität fehlt — dies entspricht der Erwartung von Zabbix, dass ein leerer Wert als „NOT_SUPPORTED“ und nicht als Agent-Fehler interpretiert wird.

Diagnose

```
pss-metrics.py -v health        # log every HTTP request to stderr
pss-metrics.py --cache-ttl=0 …  # bypass cache while debugging
rm -rf /run/pss-metrics         # purge cache
```

4.10.4 Let's Encrypt und certbot für HTTPS

Ab Version 1.9.2.340 unterstützt Perfect Streamer die automatische Erneuerung von Let's-Encrypt-Zertifikaten für HTTPS in Web Server, HTTP Server und EPG Server.

4.10.5 certbot für RHEL einrichten.

Einschränkung:

- Der TCP-Port 80 muss frei sein und es muss eine öffentliche IP mit Public-Domain-Name vergeben sein.
- Alle HTTPS-Server nutzen denselben Hostnamen (Web Server, HTTP Server, EPG Server).

Installation von certbot (https://certbot.eff.org/instructions?ws=other&os=snap):

```
sudo yum install snapd
sudo systemctl enable --now snapd.socket
sudo ln -s /var/lib/snapd/snap /snap
sudo snap install certbot --classic
```

certbot konfigurieren:

```
sudo ln -s /snap/bin/certbot /usr/bin/certbot
sudo certbot certonly --standalone
```

certbot-Prüfung:

```
sudo certbot renew --dry-run
```

certbot-Timer prüfen:

```
systemctl list-timers | grep certbot
```

In der Perfect-Streamer-Adminoberfläche HTTPS für Web Server, HTTP Server und EPG Server aktivieren.

Hook-Skript erstellen (https://pstreamer.tv/distrib/scripts/cert_update.zip) und es unter folgendem Pfad ablegen:

```
/opt/pss/scripts/cert_update.sh
```

Dort kann bei Bedarf der Hostname eingetragen werden; standardmäßig wird er aus /etc/hostname übernommen.

Datei ausführbar machen.

```
chmod +x /opt/pss/scripts/cert_update.sh
```

Skriptausführung prüfen, es sollten keine Fehler auftreten:

```
/opt/pss/scripts/cert_update.sh
```

Die HTTPS-Einstellungen sollten greifen; die Statusänderung wird in den Logs angezeigt.

Hook-Datei in certbot hinzufügen:

```
certbot renew --deploy-hook "/opt/pss/scripts/cert_update.sh"
```

certbot erneut prüfen:

```
sudo certbot renew --dry-run
```

4.10.6 certbot für Debian/Ubuntu einrichten.

Die Einrichtung unter Debian läuft analog zu RHEL; kurze Beschreibung am Beispiel Ubuntu 24.04.2 LTS.

certbot installieren:

```
apt install certbot
certbot certonly
```

Skript ausführbar machen:

```
chmod +x /opt/pss/scripts/cert_update.sh
```

Skript ausführen:

```
/opt/pss/scripts/cert_update.sh
```

Ausgabe:

```
Select domain name (your domain name)
```

Prüfen, ob die Zertifikate erneuert wurden:

```
ls -lat /opt/pss/config/cert/
total 44
-rw------- 1 root root  241 May 26 07:52 epgserver.key
-rw------- 1 root root  241 May 26 07:52 httpserver.key
-rw------- 1 root root  241 May 26 07:52 webserver.key
-rw-r--r-- 1 root root 1338 May 26 07:52 epgserver.crt
-rw-r--r-- 1 root root 1338 May 26 07:52 httpserver.crt
-rw-r--r-- 1 root root 1338 May 26 07:52 webserver.crt
```

Das Datum muss aktuell sein.

4.11 DVB-Adapter

Perfect Streamer unterstützt jeden im System installierten DVB-Adapter. Unterstützte Standards: DVB-S, DVB-S2, DVB-T, DVB-T2, DVB-C, ATSC. Zusätzlich implementiert: T2-MI-Dekapselung (ETSI TS 102 773) und BISS-1 / BISS-E-Entschlüsselung.

Die Hauptvoraussetzung ist ein korrekt installierter und funktionierender Adaptertreiber im System.

Der Bereich DVB erscheint nur, wenn im System gültige DVB-Adapter vorhanden sind. Eine Rekonfiguration zur Laufzeit wird nicht unterstützt; ein Neustart des Streamers ist erforderlich.

4.11.1 Adapteranbindung

Um einen neuen DVB-Adapter hinzuzufügen, wechseln Sie in den entsprechenden Abschnitt und fügen Sie den Adapter hinzu:

- Adaptername festlegen.
- Einen Adapter aus der Liste der im System verfügbaren auswählen.
- Modus **Stream** wählen.
- Den Typ des Übertragungssystems angeben: **DVB-S**, **DVB-S2**, **DVB-T**, **DVB-T2**, **DVB-C**.
- Empfangsparameter angeben: **Trägerfrequenz**, **Polarisation**, **Symbolrate**, **FEC**, **Modulation**, **DVB-Stream-ID**, **Oszillatorfrequenz**, **High-Band-Oszillator**, **High-Band-Grenze**, **DiSEqC-1.0-Modus** sowie weitere typabhängige Parameter.

Optional kann die Aufzeichnung von EIT in die EPG-Datenbank aktiviert werden (**EIT in EPG-DB schreiben**).

Das Hinzufügen für jeden im System vorhandenen Adapter wiederholen.

4.11.2 DVB-Scan

Damit die Empfangsparameter (Frequenz, Polarisation, Symbolrate, FEC, Modulation) nicht manuell eingegeben werden müssen, enthält Perfect Streamer einen integrierten Transponder-Scanner. Der Scanner durchläuft die Transponder des ausgewählten Satelliten (DVB-S/S2) oder regionalen Bandes (DVB-C, DVB-T/T2), führt das Tuning und den Empfang jedes einzelnen durch, sammelt die PSI/SI-Tabellen (PAT, PMT, SDT) und erstellt eine abschließende Liste der Multiplexe mit ihren Programmen. Jeder gefundene Multiplex kann mit einer Schaltfläche zur DVB-Adapterliste hinzugefügt werden.

Der Scanner belegt den physischen Adapter vollständig; für den Start wird ein Adapter benötigt, der weder vom Betriebssystem-Kernel noch von einem aktiven DVB-Adaptereintrag in Perfect Streamer verwendet wird.

Verfügbare Adapter

Beim Öffnen des Scan-Bildschirms in der Administrationsoberfläche wird eine Liste der physischen DVB-Adapter des Systems mit dem jeweiligen Belegungsstatus angezeigt:

- **free** — der Adapter steht für das Scannen zur Verfügung.
- **kernel** — das Gerät wird von einem anderen Betriebssystemprozess belegt.
- **pss-id-N** — der Adapter wird bereits durch einen DVB-Adaptereintrag in Perfect Streamer mit der angegebenen Kennung verwendet. Solange dieser Eintrag aktiv ist, kann der Scanner darauf nicht gestartet werden. Um den Adapter vorübergehend freizugeben, muss der vorhandene DVB-Adaptereintrag pausiert werden (das Kennzeichen **Pause** in seinen Einstellungen).

Transponderlisten

Der Scanner orientiert sich an Referenzlisten im Enigma2-Format: der Satellitenliste `satellites.xml` und den regionalen Listen `cables.xml` / `terrestrial.xml`. Jede Datei enthält eine Reihe von Transpondern für eine bekannte Orbitalposition oder ein regionales DVB-T/C-Band (näheres auf der Projektseite oe-alliance-tuxbox-common).

Die Dateien befinden sich im Verzeichnis `sat/` relativ zu `pss.json` (standardmäßig `/etc/pss/sat/`). Sie sind im Lieferumfang von Perfect Streamer enthalten und werden beim Öffnen des Scan-Bildschirms geladen. Bei Bedarf können sie durch Ersetzen der jeweiligen XML-Datei aktualisiert werden.

In der Administrationsoberfläche sind die Referenzlisten als drei Listen dargestellt:

- **Satellit** — Orbitalpositionen (zum Beispiel *Hot Bird 13.0°E*, *Astra 19.2°E*).
- **Kabelregion** — Land oder DVB-C-Anbieter.
- **Terrestrische Region** — DVB-T/T2-Region.

Ist der benötigte Satellit oder die Region nicht in den Referenzlisten enthalten, kann das XML aktualisiert oder stattdessen der **Blindscan** verwendet werden (siehe unten).

Start

Im Scan-Dialog werden nacheinander folgende Angaben festgelegt:

- Ein freier physischer Adapter.
- Delivery-System-Typ: **DVB-S**, **DVB-S2**, **DVB-T**, **DVB-T2** oder **DVB-C**.
- Quelle:
  - für DVB-S/S2 — eine Orbitalposition aus der Satellitenliste und LNB-Parameter (Lokaloszillator-Frequenzen LO1 und LO2, Grenzfrequenz des oberen Bandes, DiSEqC-Port);
  - für DVB-C — eine Kabelregion;
  - für DVB-T/T2 — eine terrestrische Region.

Nach Drücken von **Start** wird der Scan im Hintergrund ausgeführt. Der Fortschritt wird in der Administrationsoberfläche angezeigt:

- der Fortschritt in Prozent, bezogen auf die Anzahl der verarbeiteten Transponder;
- die aktuelle Frequenz und Polarisation;
- die Zähler *Multiplexe gefunden* und *Programme gefunden*;
- ein Baum der bereits gefundenen Multiplexe, der bis zu den Programmen aufgeklappt werden kann.

Der Scanner ist eine streamerweite gemeinsame Ressource: Es läuft jeweils höchstens ein Scan gleichzeitig. Wird während eines laufenden Scans ein neuer gestartet, wird der vorherige automatisch abgebrochen. Die Schaltfläche **Abbrechen** bricht den Scan ab und leert die angesammelte Liste.

Die Scan-Dauer hängt von der Anzahl der Transponder in den ausgewählten Referenzlisten ab (typischerweise bis zu 5 Sekunden pro Transponder: bis zu 2 Sekunden für den Signal-Lock und bis zu 5 Sekunden für die Erfassung der PSI). Typische Werte:

- DVB-S Hot Bird 13.0°E — etwa 2 Minuten (44 Transponder).

- DVB-S Astra 19.2°E — etwa 1,5 Minuten.
- DVB-T europäische Region — weniger als eine Minute.

Ergebnis

Das Scan-Ergebnis wird als Baum **Multiplex → Programme** angezeigt.

Multiplex-Parameter:

- Frequenz, Polarisation, Symbolrate;
- FEC, Modulation, Delivery System;
- Transportstrom-Kennung (**TSID**);
- die Frontend-Messwerte zum Zeitpunkt des Endes der PSI-Erfassung — **SNR**, **Signal**, **BER**;
- die Zähler *pmt-total* / *pmt-recv* — wie viele PMT-Tabellen von der PAT angekündigt wurden und wie viele innerhalb des Timeouts tatsächlich erfasst wurden.

Programmparameter:

- **PNR** (program_number) — die Service-Kennung innerhalb des Multiplexes;
- **Name** und **Anbieter** — aus der SDT-Tabelle, in UTF-8;
- **scrambled** — der Verschlüsselungs-Indikator. Seine Quelle wird in der folgenden Reihenfolge bestimmt: das Bit `free_CA_mode` aus der SDT (Deklaration des Broadcasters) → die Transport-Scrambling-Flags aus der PMT. Treffen weder SDT noch PMT innerhalb des Timeouts ein, ist der Standardwert 0 („unverschlüsselt“); der tatsächliche Zustand wird beim Empfangsversuch festgestellt;
- **video-pid**, **audio-pid**, **pcr-pid** — die wichtigsten Elementarströme des Dienstes.

Anwendung des Ergebnisses

Der im Baum ausgewählte Multiplex wird mit einer einzigen Schaltfläche zur DVB-Adapterliste hinzugefügt. Es wird ein neuer Eintrag mit den Parametern aus dem Scan-Ergebnis (Frequenz, Polarisation, Symbolrate, FEC, Modulation, Delivery System) sowie den zu Beginn des Scans angegebenen LNB-Parametern und dem Paar *adapter/device* erstellt. Der Adaptername wird vom Benutzer festgelegt; zusätzliche Parameter (BISS-Schlüssel für verschlüsselte Kanäle, T2-MI, gemeinsam genutztes LNB usw.) werden nach dem Anlegen des Eintrags in dessen Einstellungen ergänzt.

Programme aus den gefundenen Multiplexen werden separat angewandt — durch Anlegen eines Streams (**Stream**) mit einer **demuxer**-Eingangsquelle, adressiert über die PNR (siehe Abschnitt *Anbindung eines SPTS-Streams an einen DVB-Multiplex-Dienst*).

Blindscan

Der Blindmodus wird verwendet, wenn:

- der benötigte Satellit nicht in den Referenzlisten enthalten ist (nicht standardisierte Orbitalposition, lokaler Uplink);
- die regionalen DVB-T/C-Referenzdaten unzureichend oder für einen bestimmten Standort veraltet sind;
- ein Bandabschnitt unabhängig von der Liste bekannter Transponder erneut überprüft werden soll.

In diesem Modus greift der Scanner nicht auf die Referenzlisten zu, sondern erzeugt eine Liste der Transponder aus einem Frequenzraster. Standardmäßig werden die folgenden typischen Bereiche verwendet:

- DVB-S/S2 Ku — 10700..12750 MHz, Schrittweite 4 MHz, beide Polarisationen (H und V), typische Symbolraten 22000 / 27500 / 30000 kSym/s.
- DVB-C — 47000..862000 kHz, Schrittweite 8000 kHz, QAM-64 und QAM-256, typische Symbolraten 6875 / 6900 / 6952 kSym/s.
- DVB-T/T2 — 174000..862000 kHz, Schrittweite 8000 kHz.

Ein vollständiger Blinddurchlauf des Ku-Bands dauert in der Größenordnung von 100 Minuten (mehrere tausend Abstimmpunkte). In der Praxis wird der Bereich in der Administrationsoberfläche manuell eingeschränkt — minimale/maximale Frequenz und Rasterschritt. Ein Durchlauf von 11700..11800 MHz mit 4 MHz Schritt für ein einzelnes LNB-Band dauert beispielsweise etwa 5 Minuten.

Das Ergebnisformat eines Blindscans ist mit dem eines normalen Scans identisch. Besonderheiten:

- die Felder **FEC** und **Modulation** der gefundenen Multiplexe werden auf dem Wert **AUTO** festgesetzt — der Scanner ermittelt ihre genauen Werte nicht;
- Das Delivery System entspricht dem angeforderten (DVB-S, DVB-S2, …). Für gemischte Netze wird empfohlen, zwei Durchläufe durchzuführen — DVB-S und DVB-S2 getrennt.

Das Anwenden eines Multiplexes aus einem Blindscan erfolgt auf dieselbe Weise wie aus einem normalen Scan — über die Schaltfläche zum Hinzufügen zur DVB-Adapterliste. Die Felder **FEC** und **Modulation** werden in der Regel auf AUTO belassen und bei Bedarf nach einem stabilen Signal-Lock auf dem jeweiligen Transponder verfeinert.

4.11.3 Größe des Kernel-Empfangspuffers

Der Parameter **buffer-size** (Ganzzahl, Standard 512) legt die Größe des Kernel-DVB-Demux-Ringpuffers in Blöcken zu 65536 Byte fest.

- `512` (32 MB) — empfohlener Standardwert. Deckt Volltransponder-Szenarien DVB-S/S2 (>= 33 Mbit/s) mit einem oder mehreren MPTS-Verbrauchern ab. Ausgewählt anhand von Stand-Tests mit einem TBS-Adapter unter voller Transponderlast.
- `8...64` (512 KB … 4 MB) — zulässig für eingebettete Systeme mit begrenztem RAM oder für Adapter im Scanner-/Femon-Modus mit geringem Datenverkehr.
- `0` — Treiber-Standard beibehalten (in der Regel 8…32 KB). Geeignet nur für sehr gering belastete Szenarien. Bei Streams über 10 Mbit/s treten Verluste auf.

Wenn eine Meldung folgender Form im Protokoll erscheint:

```
DVB adapter X/Y dvr buffer overflow (NN so far, KK pids);
raise 'buffer-size' or reduce pid filter
```

buffer-size erhöhen oder die Anzahl der den Filter passierenden PIDs verringern (z. B. den MPTS-Ausgang weglassen, falls nicht benötigt).

Speicherbedarf: Wert N verbraucht N × 64 KB Kernel-Speicher pro Adapter. Bei vielen Adaptern (8 und mehr) ist dies zu berücksichtigen (8 × 32 MB = 256 MB).

4.11.4 SPTS-Stream an einen DVB-Multiplex-Dienst anschließen

Beim Hinzufügen eines neuen SPTS-Kanals zum `input` des Streams wählen Sie:

- Typ: **demuxer**.
- Quelle: **DVB-Adapter**.
- Auf dem DVB-Adapter erstellter Multiplex, nach Adaptername.
- PNR — wird aus der Popup-Liste der im Multiplex erkannten Dienste ausgewählt oder manuell eingegeben.

4.11.5 Zugriffsrechte für DVB-Geräte

Wenn DVB-Adapter in Perfect Streamer nicht angezeigt werden, gehen Sie wie folgt vor:

```
sudo nano /etc/udev/rules.d/99-dvb-permissions.rules
SUBSYSTEM=="dvb", GROUP="video", MODE="0660"

sudo usermod -aG video pss
sudo chown -R root:video /dev/dvb/*
sudo reboot
```

4.11.6 T2-MI-Dekapselung

T2-MI (T2-Modulator Interface, ETSI TS 102 773) ist ein Format für den Transport von DVB-T2-Streams über DVB-S2 multistream. Der äußere DVB-S/S2-Transponder trägt eine oder mehrere T2-MI-carrier-PIDs, von denen jede BBFRAMEs mit einem oder mehreren PLPs (Physical Layer Pipe) kapselt. Nach der Dekapselung wird aus dem BBFRAME der innere MPEG-TS mit Programmen und PSI/SI-Tabellen extrahiert.

Die Perfect-Streamer-Implementierung arbeitet im **Multi-Carrier-Modus**: ein einzelner physischer Adapter liefert gleichzeitig den äußeren DVB-S/S2-Multiplex **und** alle dekapselten T2-MI-Carrier (einen pro carrier-PID, der im PMT des äußeren Streams gefunden wurde).

Konfigurationsparameter

t2mi-mode (Int, 0..2, Standard 0) — Dekapselungsmodus:

- `0` — Deaktiviert. Der äußere MPEG-TS wird unverändert weitergeleitet. Wenn ein T2-MI-Deskriptor (tag 0x51) im PMT erkannt wird, wird ein einmaliger Hinweis protokolliert.
- `1` — Manuell. Die Dekapselung ist dauerhaft aktiviert. Wenn `t2mi-pid` ungleich 0 ist, wird beim Start ein carrier auf dieser PID vorab erstellt. Weitere carrier werden weiterhin automatisch über das PMT erkannt.
- `2` — Auto. Carrier werden automatisch aus dem PMT des äußeren Multiplexes für alle ES erkannt, die wie T2-MI aussehen (Deskriptor 0x51 oder einzelnes ES mit stream_type=0x06 auf einem Dienst ohne andere A/V-ES). Werden keine carrier gefunden, arbeitet der Adapter als regulärer DVB-Multiplex.

t2mi-pid (Int, 0..8191, Standard 0) — PID für das Vorab-Erstellen eines carrier beim Start, vor Eintreffen des PMT:

- `0` — keine Vorab-Erstellung. Carrier werden aus dem PMT erkannt (empfohlen für Auto-Modus 2).
- `1..8191` — carrier auf dieser PID vorab erstellen. Zusätzliche im PMT gefundene T2-MI-ES erhalten dennoch eigene carrier.

Im Multi-Carrier-Modus ist der Parameter `t2mi-pid` **kein** Selektor für einen einzelnen carrier — jedes erkannte T2-MI-ES erhält einen eigenen carrier mit eigenem Dekapselator. Der Parameter ermöglicht eine frühe Initialisierung für eine bekannte PID.

t2mi-plp (Int, 0..255, Standard 0) — Kennung des PLP, das aus jedem T2-MI-carrier auf dem Adapter extrahiert wird. Wird auf **alle** carrier angewendet — eine Überschreibung pro carrier wird in der aktuellen Version nicht unterstützt. Wenn im Betrieb verschiedene carrier unterschiedliche PLPs tragen, sollte man:

- ein für alle carrier gemeinsames PLP angeben, oder
- separate Adapter für verschiedene PLPs mittels `lnb-sharing` konfigurieren.

Dies ist die Kennung des BBFRAME-Feldes `plp_id`, **nicht** der DVB-S2-multistream-ISI (der über den Parameter `dvb-stream-id` festgelegt wird). Es handelt sich um verschiedene Kennungen auf verschiedenen Ebenen.

Diagnose der PLP-Auswahl:

- Fünf Sekunden nach dem Start eines carrier wird, falls für das konfigurierte PLP kein BBFrame empfangen wurde, aber andere PLPs zu sehen sind, eine Warnung mit der Liste der beobachteten `plp_id` protokolliert.

t2mi-tsid (Int, -1..255, Standard -1) — für zukünftige Nutzung reserviert. Selektor der T2-MI-Stream-Kennung, wenn mehrere T2-MI-Streams eine carrier-PID teilen. In der aktuellen Version ignoriert.

Zusammengesetzte PNR — SPTS-Anbindung aus T2-MI

Ein Adapter kann mehrere logische Multiplexe bereitstellen:

- `carrier-id = 0` — äußerer DVB-S/S2-Multiplex (reguläre A/V-Dienste).
- `carrier-id = 1..N` — dekapselte T2-MI-carrier (einer pro äußerem T2-MI-ES).

4.11.7 BISS-Entschlüsselung

Die Entschlüsselung verschlüsselter DVB-Streams nach BISS-1 (Modus E1) und BISS-E (Modus E2) wird unterstützt. Anwendbar auf die Übertragungssysteme DVB-S, DVB-S2, DVB-T, DVB-T2.

Die Implementierung erlaubt es, mehrere Entschlüsseler gleichzeitig auf einem Adapter aktiv zu halten:

- Nach **PNR** im äußeren Multiplex (regulärer Dienst).
- Nach `plp_id` zur Entschlüsselung der T2-MI carrier-PID **vor** der Dekapselung (erforderlich für verschlüsselte `multistream`-Streams — andernfalls verwirft der Dekapselator jedes verschlüsselte Paket, siehe Zähler `<t2mi scrambledDropped>`).

4.12 EPG

4.12.1 EPG/XMLTV-Import

EPG-Daten werden aus verschiedenen Quellen in der EPG-Datenbank gesammelt:

- EIT aus empfangenen Streams (SPTS und MPTS). Aktivierbar über Stream: Extract EIT to EPG Database.
- Import in XMLTV format from different external sources. Set in Configuration/EPG/EPG Sources. Supported sources EPG in the form of a link to a web resource and a local file, with the full path specified.

Die EPG-Event-Aufbewahrungsdauer wird über den Parameter EPG storage period (days) gesetzt.

Auto-Clean-Datenbank — Programme ohne Events werden entfernt.

Im EPG-Bereich werden EPG-Quellen und zugehörige Daten angezeigt. Für jeden Kanal (EPG Channels List) lassen sich festlegen:

- Channel Name — Name, der beim Export auf dem XMLTV-Server verwendet wird.
- Time Zone — die Zeitzone kann korrigiert werden, falls beim Import keine Bindung an UTC erfolgte.
- EPG Channel Sets — Kanal an ein Channel Set binden (siehe unten).
- Icon — URL des Kanal-Icons (*http://example.com/mychannel/myicon.png*).

4.12.2 EIT-Generator

EIT-Daten aus der EPG Database können in einem SPTS Stream generiert werden. Setzen Sie dazu in den Stream-Einstellungen **EPG Source ID** und wählen Sie **EPG Channel ID**. Dabei wird die SDT zwingend generiert, auch wenn sie in der Quelle nicht vorhanden ist. Setzen Sie **SDT Data** korrekt.

Wird dieser Stream im Multiplexer eingesetzt, kann der Service Name separat in der output/muxer-Einstellung überschrieben werden.

4.13 EPG-Server (XMLTV)

Ein separater integrierter HTTP-Server in Perfect Streamer liefert das vollständige XMLTV für einen vorgegebenen Kanalsatz. Der Endpunkt ist für Middleware und Player gedacht, die das Programm lokal vorhalten und es selten aktualisieren — als ganze Datei, typischerweise ein- bis zweimal täglich.

Der Server und seine Clients werden unter **Configuration/EPG/EPG Server** konfiguriert.

4.13.1 URL und Authentifizierung

Der Dienst wird von einem **separaten** HTTP-Server `epg-server` bereitgestellt (nicht von dem für `/data/*`). Standardmäßig lauscht er auf den Ports `10444` (HTTP) und `10445` (HTTPS); Ports und SSL werden unter `/config/epg-server` konfiguriert.

Routen:

URL	Content-Type	Verhalten
`GET /xmltv`	`text/xml`	Bei `Accept-Encoding: gzip` wird die Antwort spontan komprimiert (`Content-Encoding: gzip`), sonst als unkomprimiertes XML ausgeliefert.
`GET /xmltv.gz`	`application/octet-stream`	Liefert stets einen Gzip-Stream mit `Content-Disposition: inline; filename="xmltv.xml.gz"` — praktisch zum Speichern als Datei.

Drei Authentifizierungsmethoden werden unterstützt:

- **HTTP Digest** (bevorzugt) — Konto aus `/config/epg-server/login`.
- **URL-Parameter** — `?l=<login>&p=<password>` (Synonyme: `login=…`, `password=…`).
- **Loopback** — eine Anfrage von `127.0.0.1` wird anonym behandelt. Praktisch für Skripte, die auf derselben Maschine laufen.

> **Warnung:** Login und Passwort in der URL landen in den Access-Logs des Reverse-Proxys und im Browserverlauf. Für die öffentliche XMLTV-Verteilung verwenden Sie HTTP Digest oder lassen Sie Anfragen nur von privaten Adressen zu.

4.13.2 Zugriff per channel-set

Jedes `epg-server/login`-Konto ist an **ein** `channel-set` aus `/config/epg-channel-set` gebunden. Das EPG in der Antwort enthält nur die Kanäle des angegebenen Sets. So kann eine einzige PSS-Installation unterschiedlichen Betreibern/Middleware unterschiedliches XMLTV ausliefern.

Grundeinrichtung in der UI:

1. Erstellen Sie unter **Configuration/EPG/EPG Channel Sets** eine Kanalgruppe und weisen Sie die gewünschten Kanäle bei den EPG-Quellen zu.
2. Legen Sie unter **Configuration/EPG/EPG Server Clients** ein Konto an und binden Sie die erstellte Gruppe daran. Ohne Channel-Set-Bindung erhält der Client ein leeres XMLTV.

Zusätzliche Einschränkungen für einen Login:

- `ip-addr` — wenn gesetzt und kein Platzhalter, erhält eine Anfrage von einer anderen IP `403 Forbidden`.
- `limit-day` — Unix-Epoche-Sek., nach der das Konto nicht mehr bedient wird (`403 Forbidden`). Praktisch für ein Abonnementmodell.
- `pause` — einen Login vorübergehend deaktivieren, ohne ihn zu löschen.

4.13.3 Antwortformat

Der Antwortkörper ist ein XMLTV-Dokument mit der Wurzel `<tv>`. Die Struktur folgt dem gängigen XMLTV DTD-Schema:

```
<?xml version="1.0" encoding="utf-8"?>
<tv source-info-name="..." source-info-url="...">
  <channel id="channel.one">
    <display-name lang="en">Channel One</display-name>
    <icon src="https://.../channel-one.png"/>
  </channel>
  ...
  <programme start="20260504060000 +0300"
             stop ="20260504070000 +0300"
             channel="channel.one">
    <title lang="en">Morning News</title>
    <desc  lang="en">Overview of the day's events</desc>
    <rating system="MPAA"><value>PG</value></rating>
  </programme>
  ...
</tv>
```

Hinweise zu Feldern:

- Die Attribute `source-info-name` und `source-info-url` der `<tv>`-Wurzel werden aus den Feldern **EPG Source Name** und **EPG Source URL** unter **Configuration/EPG/EPG Server** gefüllt.
- Die Attribute `start` und `stop` werden im Format `YYYYMMDDhhmmss ±zone` angegeben (die Zeitzone stammt aus dem Feld `time_zone` des Kanals).

- Ein <programme> darf mehrere <title>/<desc> für verschiedene Sprachen enthalten. Das Attribut lang ist leer, wenn die Sprach-ID aus den EPG-Quelldaten nicht im Wörterbuch zugeordnet werden konnte (der Eintrag erscheint dennoch in der Ausgabe).
- Kanäle mit einem konfliktbehafteten channel_id (wenn dieselbe ID aus mehreren Quellen kommt) werden einmal angezeigt, die übrigen Quellen werden mit einer Warnung im Serverlog übersprungen.
- Nur Ereignisse mit stop_time >= now werden in die Ausgabe aufgenommen.

4.13.4 HTTP-Header

Der Server sendet immer:

```
Cache-Control: no-cache, no-store, must-revalidate
Pragma:        no-cache
Expires:       0
Connection:    close
```

Für /xmltv zusätzlich — Content-Encoding: gzip bei vorhandenem Accept-Encoding: gzip in der Anfrage. Für /xmltv.gz — Content-Disposition: inline; filename="xmltv.xml.gz".

Das Verbot des clientseitigen Cachens ist beabsichtigt: XMLTV ändert sich bei jedem EIT-Import oder Refresh externer XMLTV-Quellen, und der Player darf veraltete Daten nicht unbegrenzt vorhalten. Ein Edge-Cache (nginx) ist hingegen voll akzeptabel — siehe Abschnitt zur Leistung weiter unten.

4.13.5 Server-Cache und sein Zurücksetzen

Das fertige XMLTV wird im Speicher des PSS-Prozesses gecacht:

- Ein Eintrag pro channel-set; intern werden beide Körpervarianten (raw und gzip) vorgehalten — wiederholte Anfragen mit Accept-Encoding: gzip oder /xmltv.gz komprimieren die Daten nicht erneut.
- Jeder Eintrag erhält einen update-time-Zähler. Jede EPG-Aktualisierung (EIT-Import, XMLTV-Quellen-Refresh) erhöht den Zähler, und der Cache wird bei der nächsten Anfrage neu aufgebaut.

Cache erzwungen leeren:

```
POST /xmltv/reset-cache
```

Die Route wird vom **Admin-Server** (Port 43971/43981) bedient, nicht vom epg-server. Der Anfragekörper ist leer; die Antwort ist 200 OK mit einem JSON-Umschlag.

4.13.6 HTTP-Antwortcodes

Code	Bedingung
`200 OK`	Anfrage verarbeitet. Der Körper ist ein XMLTV-Dokument (eventuell ein leeres `<tv></tv>` bei einem vorübergehenden DB-Fehler).
`401 Unauthorized`	Weder Digest noch `l/p`-Parameter haben die Prüfung bestanden (für nicht-Loopback-Anfragen).
`403 Forbidden`	Der Login existiert, aber die Anfrage kommt nicht von einer erlaubten IP, oder `limit-day` ist abgelaufen.
`404 Not Found`	Jede URL außer `/xmltv` und `/xmltv.gz`.
`405 Method Not Allowed`	Methode ist nicht GET.

Der Fehlerkörper ist ein JSON-Umschlag mit festem Format:

```
{"status": 401, "message": "Unauthorized"}
```

4.13.7 Leistung und Skalierung

Server-Cache

Der Server-Cache bedient wiederholte Anfragen an dasselbe `channel-set` ohne SQLite-Zugriff — durch Kopieren des fertigen Körpers.

Der XMLTV-Aufbau „von Grund auf“ (Cache-Miss) ist teurer: pro Kanal ein eigenes SELECT über `channel_name`, pro Ereignis eines über `event_text` und `event_rating`. Anhaltspunkte zur Bauzeit:

Ausgabegröße	Cache hit	Cache miss (build)
100 Kanäle / Tag	einige Dutzend ms	~0,5–1 s
500 Kanäle / Tag	~50 ms	2–5 s
1000+ Kanäle / Woche	~100–300 ms	5–15 s

Für die meiste Middleware ist es akzeptabel, XMLTV alle paar Stunden oder einmal täglich abzufragen.

Wann ein externer Reverse-Proxy (nginx) nötig ist

Im Gegensatz zu `/data/epg/channel` (kurze JSON-Antworten) ist XMLTV ein einziges großes Dokument pro `channel-set` und damit ideal für einen Edge-Cache:

- **Zehn bis Hunderte Clients pro channel-set** — der interne PSS-Cache reicht meist aus, wenn sie XMLTV stündlich bis täglich abfragen.
- **Tausende gleichzeitige Clients** — ein cachender Reverse-Proxy wird empfohlen. Das Ausliefern einer XMLTV-Datei an Hunderte/Tausende Anfragen ist in erster Linie eine Netzwerklast (Hunderte KB – einige MB pro Antwort), die man besser von PSS fernhält.
- **Geografisch verteilte Auslieferung** — ohne CDN/Edge-Cache geht es nicht, unabhängig von der Clientzahl.

PSS sendet `Cache-Control: no-cache`, daher muss nginx explizit angewiesen werden, den Upstream-Header zu ignorieren und einen eigenen TTL zu führen.

Beispielkonfiguration für nginx

```
# /etc/nginx/conf.d/pss-xmltv.conf

proxy_cache_path /var/cache/nginx/pss-xmltv
                 levels=1:2
                 keys_zone=pss_xmltv:8m
                 max_size=4g
                 inactive=2h
                 use_temp_path=off;

upstream pss_epg {
    server 127.0.0.1:10444;
    keepalive 16;
}

server {
    listen 80;
    # listen 443 ssl http2;     # SSL termination makes sense here
    server_name epg-files.example.com;

    # Do not enable gzip on: /xmltv.gz is already compressed, and /xmltv
    # arrives gzip-encoded from PSS — re-compressing is pointless.

    location ~ ^/xmltv(\.gz)?$ {
        proxy_pass http://pss_epg;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        # PSS returns no-cache; force-cache it at the edge.
        proxy_ignore_headers Cache-Control Expires Set-Cookie;
        proxy_hide_header   Cache-Control;
        proxy_hide_header   Pragma;
        proxy_hide_header   Expires;

        # Cache key = full URL including query (login/password in /xmltv?l=...&p=...)
        # produce distinct keys for different accounts with different channel-sets.
        proxy_cache_key     "$scheme$host$request_uri";

        proxy_cache         pss_xmltv;
        proxy_cache_valid   200      30m;       # XMLTV changes infrequently
        proxy_cache_valid   401 403  1m;
        proxy_cache_lock           on;          # coalesce cache misses
        proxy_cache_lock_timeout   60s;         # XMLTV build may take seconds
        proxy_cache_use_stale      error timeout updating
                                   http_500 http_502 http_503;

        # Large buffer: XMLTV for a big channel-set can reach
        # several megabytes.
        proxy_buffering      on;
        proxy_buffers        16 256k;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        proxy_buffer_size    256k;
        proxy_busy_buffers_size 1m;

        # Let the client cache for a short period.
        add_header Cache-Control "public, max-age=600";
        add_header X-Cache-Status $upstream_cache_status always;
    }
}
```

Erläuterungen und Empfehlungen

- **TTL ``proxy_cache_valid 200 30m``** — XMLTV ändert sich selten häufiger als alle halbe Stunde. Erfolgt die Synchronisation mit den Quellen stündlich oder seltener, kann der Wert auf 1 Stunde und mehr angehoben werden; ist Frische nach `POST /xmltv/reset-cache` wichtig, senken Sie ihn.
- **``proxy_cache_lock_timeout 60s``** — gegenüber `/data/epg/channel` erhöht (dort üblich sind 5 Sekunden), weil der Aufbau eines XMLTV für ein großes channel-set länger dauert.
- **Eine dedizierte ``keys_zone``** — selbst bei einer großen Installation sind eindeutige XMLTV-Schlüssel rar (Anzahl Logins × `channel-set`), 8 MB reichen mit Reserve. `max_size` wird nach dem XMLTV-Volumen, nicht nach der Schlüsselanzahl bemessen.
- **gzip auf nginx-Seite ist unnötig**: Für `/xmltv.gz` ist die Antwort bereits komprimiert, für `/xmltv` antwortet PSS selbst gzip-encoded bei vorhandenem `Accept-Encoding: gzip`.
- **HTTPS-Terminierung** auf nginx liefert bessere Leistung bei vielen gleichzeitigen TLS-Handshakes.

4.13.8 Verwandte Endpunkte

- `POST /xmltv/reset-cache` — erzwungenes Leeren des serverseitigen XMLTV-Caches (auf dem Admin-Server 43971/43981).
- `POST /data/epg/update?s=<src_id>` — erzwungene Aktualisierung einer externen XMLTV-Quelle; nach einer erfolgreichen Aktualisierung wird der serverseitige XMLTV-Cache automatisch geleert.
- `GET /data/epg/channel?…` — JSON-EPG-Ausgabe für einen Kanal pro Tag; siehe separaten Abschnitt.

Die vollständige Liste und ausführliche Beschreibung der HTTP-API finden Sie in `manual/http_data_api.txt`.

4.14 EPG für OTT-Middleware

Der Server liefert Programmdaten (EPG) für den gewählten Tag und einen einzigen Kanal im JSON-Format. Der Endpunkt ist für OTT-Middleware-Backends gedacht, die den Programmplan aus Perfect Streamer aggregieren, um das Programm beim Endkunden aufzubauen.

4.14.1 URL und Authentifizierung

Der Endpunkt wird vom integrierten Admin-Server bereitgestellt. Standardmäßig ist er unter den Ports `43971` (HTTP) und `43981` (HTTPS) erreichbar; die Ports werden unter **Settings/Server Settings** konfiguriert.

```
GET /data/epg/channel?src=<src_id>&ch=<channel_id>&lang=<lang>&t=<time>
```

Die Authentifizierung erfolgt per HTTP Digest, wie bei den übrigen `/data/*`. Für Middleware reicht ein Konto mit der Rolle `viewer` (nur lesen).

Bemerkung: Anfragen von der Loopback-Adresse (`127.0.0.1`) umgehen die HTTP-Digest-Prüfung — der Server behandelt sie als anonym. Das ist praktisch für lokale Skripte und Health-Checks von Middleware, die auf derselben Maschine wie Perfect Streamer läuft; für entfernte Zugriffe sind Zugangsdaten zwingend.

4.14.2 Anfrageparameter

Parameter	Typ	Pflicht	Standard	Beschreibung
`src`	vorzeichenlose Ganzzahl	nein	`0`	EPG-Quelle. `0` — aus MPEG-TS-EIT der Eingangsströme importierte Daten. `1`, `2`, … — Eintrags-ID aus `/config/epg/epg-source` (externe XMLTV-Quelle).
`ch`	Zeichenkette	**ja**	—	Kanal-Kennung aus der EPG-Datenbank: Wert des Feldes `channel_id` in der Tabelle `channel`. Die Liste der verfügbaren Kennungen lässt sich per SQL-Abfrage über `POST /data/epg/sql` ermitteln.
`lang`	Ganzzahl oder ISO 639	nein	Systemstandardsprache	`0` — Systemstandardsprache; eine Ganzzahl `> 0` — interne Sprach-ID (siehe `GET /schema/lang`); eine Zeichenkette — zwei- oder dreibuchstabiger ISO-639-Code, z. B. `eng`, `rus`, `fra`.
`t`	vorzeichenlose Ganzzahl (Unix-Epoche, Sek.)	nein	aktuelle Serverzeit	Beliebiger Zeitpunkt innerhalb des interessierenden Tages. Der Server liefert Ereignisse für den **UTC-Tag**, zu dem `t` gehört: Intervall `[t / 86400 · 86400, (t / 86400 + 1) · 86400)`. Für Daten des «nächsten Tages» reicht es, `86400` zur aktuellen Zeit zu addieren.

Bemerkung: Der Tag wird in UTC gerechnet, nicht in der lokalen Zeitzone. Baut die Middleware den Zeitplan nach dem lokalen Kalendertag auf, fällt die UTC-Tagesgrenze nicht zwingend auf die lokale Mitternacht; in diesem Fall sind zwei Anfragen (für zwei benachbarte UTC-Tage) nötig, deren Ergebnisse über das Feld `start` zusammengeführt werden.

4.14.3 Antwortformat

- `Content-Type`: `application/json`.
- Die HTTP-Header verbieten das Caching auf der Client-Seite und auf zwischengeschalteten Proxys:

```
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: 0
```

- Der Antwortkörper ist ein JSON mit der Ereignisliste für den Tag:

```
{
  "event": [
    {"start": 1715000000, "end": 1715003400, "title": "Morning News", "desc":
↪"Overview of the day's events"},
    {"start": 1715003400, "end": 1715007000, "title": "Weather Forecast", "desc": ""}
  ]
}
```

Ereignisfelder:

- `start`, `end` — Anfang und Ende der Sendung in Unix-Epoche (Sek.), UTC.
- `title` — Sendungstitel in der gewählten Sprache. Fehlt ein Titel in der angeforderten Sprache, greift der Server auf den Eintrag in der Systemstandardsprache zurück und danach auf jede andere verfügbare Sprache.
- `desc` — erweiterte Beschreibung. Kann ein leerer String sein, wenn die Datenbank keine separate Beschreibung für das Ereignis enthielt.

Besonderheiten der Ausgabe:

- Ereignisse mit leerem Titel und ohne Beschreibung sowie Ereignisse mit ungültigen Zeitstempeln werden aus der Ausgabe ausgeschlossen.
- Duplikate nach `start` werden ausgesondert: Für denselben Startzeitpunkt wird ein Datensatz mit der besten Sprachpriorität zurückgegeben (angefordert → System-Default → andere).
- Die Reihenfolge der Ereignisse im Array ist nicht garantiert — bei Bedarf sortiert die Middleware die Liste selbst nach dem Feld `start`.
- Wenn der Kanal nicht gefunden wird, die Quelle nicht existiert oder am gewählten Tag keine Ereignisse vorliegen, gibt der Server `200 OK` mit dem leeren Array `{"event":[]}` zurück — oder, im seltenen Fall einer fehlenden Quelle, mit leerem Körper. Die Middleware muss beide Varianten korrekt behandeln.

4.14.4 Caching auf dem Server

Das fertige JSON wird vom Server unter dem Schlüssel `(channel_id, UTC-Datum, Sprache)` gecacht; wiederholte Anfragen für denselben Tag und Kanal werden daher ohne Datenbankzugriff bedient. Die Middleware muss am Cache nichts steuern.

Der Cache wird automatisch zurückgesetzt:

- beim Eintreffen neuer Ereignisse aus MPEG-TS-EIT der Eingangsströme (`src=0`);
- bei einer erfolgreichen Aktualisierung einer externen XMLTV-Quelle (`src > 0` — geplant oder erzwungen über `POST /data/epg/update?s=<src_id>`);
- wenn ein Tag aus dem EPG-Aufbewahrungsfenster fällt.

Ein separater Endpunkt zum erzwungenen Leeren des Caches ist weder vorgesehen noch erforderlich.

4.14.5 HTTP-Antwortcodes

Code	Bedingung
`200 OK`	Anfrage verarbeitet. Der Körper ist ein JSON mit der Ereignisliste (eventuell leer).
`400 Bad Request`	Der Parameter `ch` fehlt oder ist leer; oder `src`, `t` lassen sich nicht als vorzeichenlose Ganzzahl parsen; oder ein numerischer `lang` verweist auf eine nicht existierende Sprach-ID.
`401 Unauthorized`	Die HTTP-Digest-Authentifizierung fehlt oder ist ungültig (für Anfragen nicht von einer Loopback-Adresse).

Sonstige Situationen (unbekanntes `src`, fehlender Kanal, keine Ereignisse am Tag) führen nicht zu 4xx — die Middleware erhält `200 OK` mit dem leeren Array `{"event":[]}`.

Bei einem Fehler ist der Antwortkörper ein JSON-Umschlag mit festem Format:

```
{"status": 400, "message": "Bad Request"}
```

Das Feld `status` dupliziert den HTTP-Code; das Feld `message` enthält eine konkretere Fehlerursache auf Englisch (oder den Standard-HTTP-Statustext, falls keine weiteren Informationen vorliegen). Der `Content-Type` der Fehlerantwort ist `application/json`.

4.14.6 Beispiel

```
curl -u middleware:secret --digest \
  'http://pss.example.com:43971/data/epg/channel?src=0&ch=12.0.1&lang=rus&t=1715000000
↪'
```

Beispielantwort:

```
{
  "event": [
    {"start": 1715000000, "end": 1715003400, "title": "Morning News", "desc":
↪"Overview of the day's events"},
    {"start": 1715003400, "end": 1715007000, "title": "Weather Forecast", "desc": ""}
  ]
}
```

4.14.7 Leistung und Skalierung

Perfect-Streamer-Server-Cache

Im PSS-Prozess gibt es einen In-Memory-LRU-Cache der Antworten mit dem Schlüssel `(channel_id, UTC-Datum, Sprache)` und einem Hard-Cap von 1024 Einträgen je EPG-Quelle. Bei typischer Last (zig bis hunderte Kanäle × 1–3 Sprachen × `keep-day` Tage) passen alle aktuellen Einträge vollständig in den Cache; wiederholte Anfragen werden ohne SQLite-Zugriff bedient.

Größenordnung (Debug-Build, lokales Loopback, ohne HTTPS):

Szenario	Latenz (Einzelanfrage)	Durchsatz (P=8)
Cache hit	~0,3 ms	~1100 Anfragen/s
Cache miss (SQL + JSON)	~1,0–1,5 ms	~1000 Anfragen/s

In einem Release-Build und ohne Debug-Logging sind die Werte etwa 2- bis 3-mal besser. Bandbreite — etwa 14 KB pro Antwort für einen typischen Kanal pro Tag.

Wann ein externer Reverse-Proxy (nginx) nötig ist

Der Server-Cache beschleunigt wiederholte Anfragen für dasselbe (`channel, Tag, Sprache`), aber jede Anfrage geht weiterhin durch den eingebauten PSS-HTTP-Server und belegt einen Thread aus dessen Pool. Bei vielen Clients ist es sinnvoll, das Caching an den Rand zu verlagern:

- **bis ~1.000 Online-Clients** — der interne Cache reicht meist aus, ein Reverse-Proxy ist nicht erforderlich.
- **Zehntausende und mehr** — ein cachender Reverse-Proxy (z. B. nginx) wird empfohlen. Ein Edge-Cache wickelt 99 % der Anfragen ohne PSS ab, dämpft Spitzen (Middleware-Start, massenhafte Player-Aktualisierung) und erlaubt es, die SSL-Terminierung auf einen separaten Knoten zu verlagern.
- **geografisch verteilte Auslieferung** — ein externes CDN/Proxy ist nötig, noch bevor die Clientzahl gezählt wird.

PSS sendet einen eigenen Header `Cache-Control: no-cache, no-store, must-revalidate`, damit Endclients das EPG nicht lange zwischenspeichern. Der Reverse-Proxy darf (und sollte) die Antwort selbst cachen — unten wird gezeigt, wie man nginx explizit anweist, den Upstream-`Cache-Control` zu ignorieren und einen eigenen TTL zu führen.

Beispielkonfiguration für nginx

Eine minimale Konfiguration für einen EPG-Edge-Cache, ausgelegt für Zehntausende Clients mit Polling-Intervall 1–5 Minuten:

```
# /etc/nginx/conf.d/pss-epg.conf

proxy_cache_path /var/cache/nginx/pss-epg
                 levels=1:2
                 keys_zone=pss_epg:32m
                 max_size=2g
                 inactive=30m
                 use_temp_path=off;

upstream pss_admin {
    server 127.0.0.1:43971;
    keepalive 64;
}

server {
    listen 80;
    # listen 443 ssl http2;  # recommended: SSL termination here
    server_name epg.example.com;
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    # gzip helps: typical EPG JSON compresses ~5–8x.
    gzip on;
    gzip_types application/json;
    gzip_min_length 512;
    gzip_proxied any;

    location = /data/epg/channel {
        proxy_pass http://pss_admin;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        # PSS returns no-cache; force-cache it at the edge.
        proxy_ignore_headers Cache-Control Expires Set-Cookie;
        proxy_hide_header   Cache-Control;
        proxy_hide_header   Pragma;
        proxy_hide_header   Expires;

        # Cache key = full URL with query string. The src/ch/lang/t
        # parameters already determine response uniqueness.
        proxy_cache_key     "$scheme$host$request_uri";

        proxy_cache         pss_epg;
        proxy_cache_valid   200 60s;          # staleness TTL
        proxy_cache_valid   400 404 10s;
        proxy_cache_lock           on;        # coalesce cache misses
        proxy_cache_lock_timeout   5s;
        proxy_cache_use_stale      error timeout updating
                                   http_500 http_502 http_503;

        # Serve the JSON to the client with its own TTL
        # (the player won't recheck EPG before that).
        add_header Cache-Control "public, max-age=60";
        add_header X-Cache-Status $upstream_cache_status always;
    }
}
```

Erläuterungen und Empfehlungen

- **TTL ``proxy_cache_valid 200 60s``** — Kompromiss zwischen EPG-Frische und Last auf dem Upstream. Das Programm ändert sich nicht in Echtzeit, daher sind 30–300 Sekunden angemessen. Nach dem Import neuer Ereignisse leert PSS seinen Cache sofort, der Edge-Cache zieht beim nächsten TTL nach.
- **``proxy_cache_lock on``** ist bei vielen Clients zwingend: Bei einem Cache-Miss bündelt es parallele Anfragen auf denselben Schlüssel zu einer einzigen Upstream-Anfrage und schützt SQLite vor Spitzen-`BUSY` unter Last.
- **``keys_zone`` und ``max_size``** werden anhand der Anzahl (`Kanal × Tag × Sprache`) dimensioniert: 32 MB `keys_zone` reichen für Hunderttausende Schlüssel; 2 GB `max_size` decken einen Monat Verlauf für Hunderte Kanäle mit Reserve ab.
- **gzip** reduziert den Verkehr deutlich: Antworten lassen sich gut komprimieren (sich wiederholende JSON-Schlüssel, Kyrillisch in UTF-8).

- ``**X-Cache-Status**`` in der Antwort erlaubt der Middleware, `HIT/MISS/EXPIRED` zu sehen und die Effektivität des Caches zu beurteilen.
- Liegen nginx und PSS auf derselben Maschine, verlangt der Admin-Server für Loopback keine HTTP-Digest-Authentifizierung, daher kann der Upstream-Block ohne `proxy_set_header Authorization …` bleiben. Bei einer Trennung über Netze legen Sie ein dediziertes `viewer`-Konto an und ergänzen Sie die Digest-Authentifizierung in `proxy_pass`.
- **HTTPS** wird sinnvollerweise an nginx terminiert: PSS unterstützt HTTPS direkt, aber ein Edge-Server ist meist effizienter bei der Abwicklung von TLS-Handshakes für Tausende gleichzeitige Clients.

4.14.8 Verwandte Endpunkte

- `POST /data/epg/sql?s=<src_id>` — beliebige SQL-Abfrage an die EPG-Datenbank (insbesondere, um eine Liste der `channel_id` zu erhalten).
- `POST /data/epg/update?s=<src_id>` — erzwungene Aktualisierung einer externen XMLTV-Quelle.

Die vollständige Liste und ausführliche Beschreibung der HTTP-API finden Sie in `manual/http_data_api.txt`.

4.15 Programmoptimierung

Treten bei vielen **stream**s Probleme mit hoher CPU-Last oder Speichermangel auf, können die Einstellungen optimiert werden.

You can disable the MPEG-TS filtering and processing features if you don’t need to.By default, the **stream** has the **Clean All Unnecessary Data** function enabled, disable it if there is no unwanted data in the stream. Disabling these features completely will remove the **Original Media Information** section of the Report.

Vollständige Deaktivierung oder Änderung der **Mosaic**-Einstellungen. Eine vollständige Deaktivierung erfolgt in den Server-Einstellungen. Sie können sie individuell für jeden **stream** deaktivieren oder das Aktualisierungsintervall mit der Einstellung **Check Interval** ändern.

Anstieg des Speicherverbrauchs bei einer großen Anzahl von Streams. PSS gibt nicht genutzten Speicher periodisch an das Betriebssystem zurück, und die mitgelieferte *systemd*-Unit begrenzt standardmäßig die Anzahl paralleler Speicherzuteilungs-Pools (Umgebungsvariable **MALLOC_ARENA_MAX**). Das bremst den allmählichen Anstieg des **RSS** beim Betrieb mit Dutzenden von Streams und ist keine Folge eines Lecks. Den Wert manuell zu ändern ist in der Regel nicht erforderlich.

4.15.1 Queue-Overload-Fehler für die Datenbanken DBStat und DBEPG

Treten auf, wenn die Datenbankleistung nicht ausreicht — langsame Festplatte oder überlastetes System.

Den Speicherort der Datenbanken legt der Parameter data-dir in der Konfigurationsdatei pss.properties fest

Mögliche Lösungen:

1. Verschieben der Datenbankdateien nach /tmp. Es wird der Systemspeicher verwendet — eine Abschätzung des verfügbaren Speichers und Anpassung der Statistik-Aufbewahrungsdauer ist erforderlich (siehe Server-Einstellungen). Bei einem System-Neustart gehen die Daten verloren.
2. Detaillierungsgrad der Statistik reduzieren — siehe Parameter dbstat-detail. Standard 5 s, lässt sich bis 20 erhöhen.
3. Die EPG-Datenbank im Arbeitsspeicher halten — Parameter dbepg-memory=true setzen.

4.16 Transkoder

Die Transkoder sind als eigenständige Binärdateien implementiert, die von pstreamer als separate Prozesse gestartet werden.

1-zu-N-Konfigurationen werden unterstützt — von einem Decoder können mehrere Streams mit unterschiedlichen Encoder-Einstellungen erzeugt werden.

Der Quellstream muss Video und Audio enthalten; Varianten ohne Video oder ohne Ton werden nicht unterstützt.

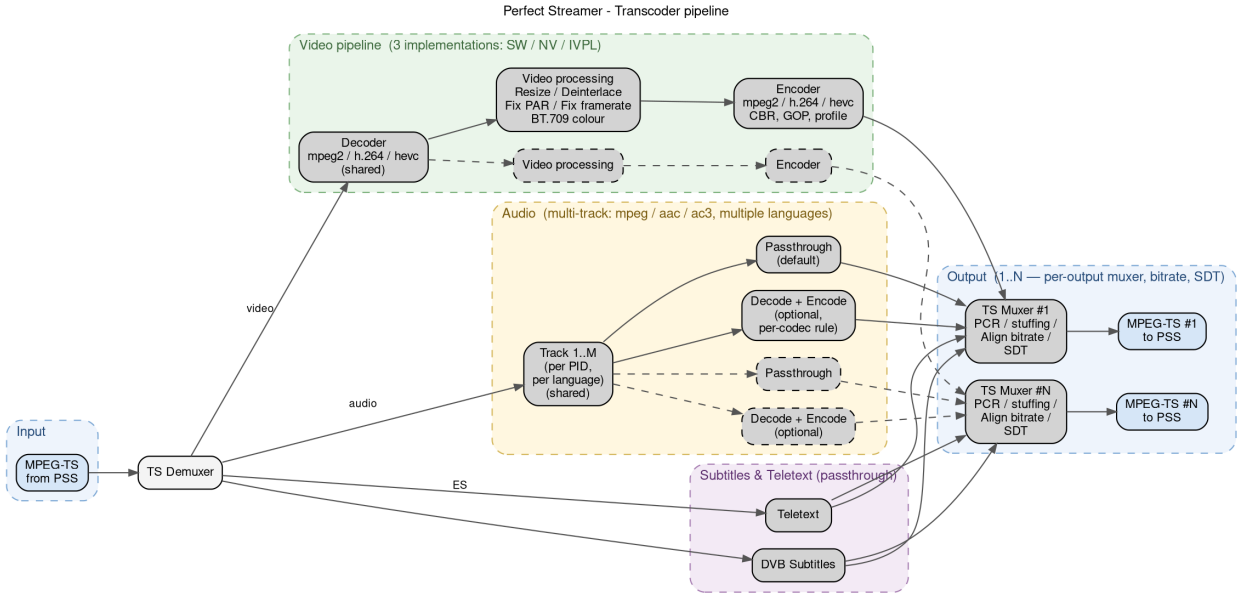


Abb. 3: Transcoder-Architektur: ein gemeinsamer **decoder → N** unabhängige Zweige *VPP + encoder* (1:N); Audio ist mehrspurig mit optionaler Transcodierung pro Output; Untertitel und Teletext werden durchgereicht.

Implementierte Codecs:

- Video SW decoder: mpeg2, h.264, hevc (h.265)
- Video NW decoder: mpeg2, h.264, hevc (h.265)
- Video SW encoder: mpeg2, h.264, hevc (h.265)
- Video NW encoder: h.264, hevc (h.265)

Interlaced Streams werden am Eingang und Ausgang unterstützt.

Für H.264- und HEVC-Decoder wird Interlace Alternate (zwei getrennte Halbbilder im Stream) unterstützt; es wird in Interlace Interleaved umgewandelt.

Der HEVC-Decoder unterstützt das Profil Main10 mit bt.709 (SDR) und bt.2020 (HDR). Der HEVC-Encoder nutzt stets das Profil Main mit bt.709.

Für die H.264- und HEVC-Decoder wird VBR (Variable Frame Rate) unterstützt; es wird in eine konstante Framerate umgewandelt.

- Audio decoder — mpeg (layer 1,2,3), aac, ac3
- Audio encoder — mpeg (layer 2), aac

Es gibt den Transkodierungsmodus **Video Passthrough** — Video wird nicht transkodiert, nur der Ton; verwendet wird der *SW*-Transkoder.

Bemerkung: Für die Transkodierung müssen zwei oder mehr Streams konfiguriert werden — mit output (Decoder) und input (Encoder).

Zur Konfiguration einer Transkoder-Instanz ist nötig:

- Quelle — im Stream-Output-Transkoder (Decoder) hinzufügen. In den Einstellungen Typ wählen: SW, NV oder Video Passthrough.
- Ausgangsstream — im Stream-Input-Transkoder (Encoder) hinzufügen; in den Einstellungen die Quelle (Decoder) wählen.
- Wiederholen, wenn mehrere Ausgangsstreams für einen Decoder benötigt werden.

4.16.1 Einstellungen des Output-Transkoders (Decoder)

- **Convert colors to BT.709** — Konvertierung von SD BT.470-2 (PAL) und SMPTE 170M (NTSC) nach BT.709
- **Trace** — für die Diagnose das detaillierte Transkoder-Log aktivieren.

Für den korrekten Betrieb des Transkoders muss der Quellstream bestimmte Anforderungen erfüllen; in einigen Fällen lässt sich das korrigieren. Diese Einstellungen konvertieren den Stream nicht — sie wirken als Hinweise für den korrekten Betrieb des Transkoders.

Für die Korrektur der Eingangsdaten gibt es folgende Einstellungen:

- **Fix PAR** — Pixel Aspect Ratio korrigieren. Wird als Bruchzahl im Format N/D angegeben; für Wide SD z. B. 16/9.
- **Fix Framerate** — Framerate explizit angeben. In manchen Streams kann die Framerate im SPS fehlen, im Transkoder-Log erscheint dann der entsprechende Fehler. In diesen Fällen ist die Framerate manuell anzugeben. Wird als Bruch im Format N/D angegeben.

Beispiele für Framerate-Werte:

- PAL — 25/1
- NTSC — 30/1 oder 30000/1001
- Cinema — 24/1 oder 24000/1001

4.16.2 Einstellungen des Input-Transkoders (Encoder)

- **Encoder Type** — Video-Codec.
- **Align Total Bitrate** — Stuffing-Bitrate des Streams (Auffüllen mit null-Paketen). Wichtig, wenn der Stream für DVB-Übertragung genutzt wird. Die Bitrate muss garantiert höher sein als die Bitrate von Video und allen Audio-Spuren.
- **Video Profile** — für H.264 lässt sich das Kodierungsprofil wählen.
- **Video Bitrate** — Bitrate des Videostreams in kbps. Die Kodierung läuft stets in CBR; die Gesamtbitrate ist durch die Tonspuren höher.
- **Speed Preset** — Voreinstellungen für die Kodierung, Werte 1–7. Niedriger = höhere Qualität und höherer Ressourcenbedarf. Standard 4.
- **GOP Interval** — Intervall in Frames für die GOP (entspricht dem Key Frame Interval). Standard 25 (1 Sekunde bei 25p); empfohlen, wenn Player zufällig einsteigen.
- **BFrame** — für höhere Qualität aktivieren. Empfohlener Wert: 3.
- **Lookahead** — für höhere Qualität aktivieren. Empfohlen: 20–50 Frames.
- **Resize** — Bildgrößenänderung.
- **Deinterlace** — wandelt Interlace in Progressive um.

Einfügen eines *crop* (leere Ränder am Bild) wird nicht unterstützt. Beliebige Bildgrößen werden nicht zugelassen, da Seitenverhältnisse verzerrt würden.

Für **resize** stehen folgende Optionen zur Verfügung:

- Größe proportional um Faktor 2 und 4 verkleinern.
- Format Wide SD 16:9 setzen, der passende Aspect Ratio wird gewählt.
- Upscale SD→HD. Wird auf SD-PAL/NTSC-Quellen angewandt. Interlace wird nicht unterstützt — bei Bedarf vorher deinterlacen.
- Breite festlegen. Die Höhe wird proportional berechnet.
- Höhe festlegen. Die Breite wird proportional berechnet.

Einige Parameter können mit dem gewählten Transkoder unverträglich sein; Fehler erscheinen in dessen Log.

4.16.3 Audioverarbeitung

Standardmäßig werden alle Tonspuren ohne Verarbeitung von Input zu Output durchgereicht. Unnötige Spuren lassen sich über PID-Filter im Stream entfernen.

Wenn der Ton transkodiert werden soll, lässt sich das über Regeln pro Audio-Codec konfigurieren. Option *skip* — Tonspur mit diesem Codec entfernen.

Fehlen im Ausgangsstream Tonspuren, kommt es zu einem Fehler — siehe Transkoder-Logs.

4.16.4 PCR-Erzeugung und TR 101 290.

Der MPEG-TS-Multiplexer erzeugt einen neuen PCR. Bei korrekt gesetztem **Align Total Bitrate** (über der Summe aus Video- und Audiobitrate) sollte der PCR die TR-101-290-Prüfung bestehen.

4.16.5 Status der Transkoder

Bei Problemen im Transkoder-Betrieb (kein Stream vom Encoder) sind die Logs im Bereich **Transcoders** zu prüfen — dort ist die Liste der Instanzen zu sehen (jede Zeile ist eine separate Instanz, decoder + N encoders); beim Klick auf eine Instanz öffnet sich der Log-Status-Dialog. Angezeigt werden das aktuelle Log und das Log des vorherigen Starts. Für ein detailliertes Log aktivieren Sie *trace* in den output (decoder)-Einstellungen.

KAPITEL 5

FAQ

5.1 SRT und Login/Passwort-Authentifizierung in Drittsoftware

Die Autorisierung mit Login und Passwort in SRT zwischen Perfect-Streamer-Instanzen wird nativ unterstützt und über die entsprechenden Felder in output und input konfiguriert. Mit anderer Software ist die Funktion nicht garantiert — sie ist nicht standardisiert und in unterschiedlicher Software unterschiedlich implementiert.

Für universelle Interoperabilität zwischen Perfect Streamer und anderer Software gibt es eine Autorisierung über einen Peer, dessen Name dem Stream-ID-Wert entspricht. Beispiel-Link:

```
srt://Stream_IP:port?streamid=!#::u=1234567890,password=1234567890
```

Peer-Name:

```
!#::u=1234567890,password=1234567890
```

Die Syntax der Stream-ID ist für Perfect Streamer unerheblich; beliebige Werte bis 511 Zeichen werden unterstützt.

Unterschiedliche SRT-empfangende Software kann die Stream-ID auch in ihrem eigenen Format übergeben. Funktioniert der Empfang über einen bestimmten Typ der Stream-ID-Verknüpfung nicht, können Sie am SRT-Ausgang von Perfect Streamer die Trace-Option aktivieren und im Empfangslog des Streams den Fehler sowie die tatsächlich von der Fremdsoftware ausgegebene Stream-ID einsehen. Auf dieser Grundlage lässt sich die Stream-ID korrigieren — etwa durch Entfernen überflüssiger Zeichen am Anfang der Stream-ID-Zeichenkette, die die Übergabe des Stream-ID-Werts durch die Fremdsoftware behindern.

5.2 Arbeiten mit RTSP und RTMP im Perfect Streamer mittels FFmpeg

Für **RTMP** und andere Transportprotokolle, für die Perfect Streamer keinen integrierten Input besitzt, lässt sich der STD-Input-Typ des Streams (stdin) verwenden. Dieselbe Methode eignet sich als Behelfslösung für nicht standardkonforme **RTSP**-Quellen — für reguläre RTSP gibt es in Perfect Streamer einen integrierten RTSP-Input. Es können FFmpeg, GStreamer und beliebige andere Anwendungen verwendet werden, die stdout unterstützen.

Beispielhafte Konfiguration mit FFmpeg:

1. Eingabetyp wählen — std.
2. Im Feld Cmd den FFmpeg-Pfad angeben — /usr/bin/ffmpeg.
3. Im Feld Args den Befehl zum Empfang des Streams eingeben:

```
-loglevel error -i rtmp://192.168.1.29/channelTV/chanelSD -c copy -f mpegts -
```

oder

```
-loglevel error -i rtsp://viewer:viewer200@172.31.91.197:554/play1.sdp -c copy -f␣
↪mpegts -
```

Hat das Kamerasignal keinen Ton, erscheinen Fehler auf der Stream-Seite. Um sie zu vermeiden, in den Stream-Typ-Einstellungen „Only Video“ wählen.

Beschreibung in der *Dokumentation* zur Anbindung von Drittanwendungen.

5.3 RTSP- und RTMP-Arbeit im Perfect Streamer mittels GStreamer

Für Transportprotokolle ohne integrierte Unterstützung in Perfect Streamer (zum Beispiel **RTMP**) sowie als Umgehungslösung für nicht standardkonforme **RTSP**-Quellen kann der STD-Input-Typ (stdin) für einen Stream verwendet werden. Es können FFmpeg, GStreamer und alle anderen Anwendungen verwendet werden, die stdout unterstützen.

Beispielhafte Konfiguration mit GStreamer.

1. Eingabetyp wählen — std.
2. Im Feld Cmd den Pfad zum Kommandointerpreter angeben — /bin/bash.
3. Im Feld Args den Pfad zum Stream-Generator-Skript und weitere Argumente eingeben:

```
/opt/conf/pss/scripts/rtmp.sh rtmp://192.168.1.2/live/mmtv2025air
```

oder

```
/opt/conf/pss/scripts/rtsp.sh rtsp://viewer1:viewer300@172.34.95.198:553/play1.sdp
```

Die vollständige Anleitung ist im Forum verfügbar.

Skripte: scripts_gstreamer.zip.

5.4 Empfehlungen zur Arbeit mit UDP-Multicast

Aufgabe:

UDP-Multicast mit mehreren Hundert Mbit/s (1 Gbit/s und mehr) auf einem Server stabil empfangen.

Problem:

Empfang auf Netzwerkkarten mit RJ-45-Anschluss — bei Verkehr über mehrere hundert Megabit beginnen zunehmende Multicast-Verluste. Das Tuning der Netzwerkkarten-Einstellungen hilft nicht (war in alten Betriebssystemversionen relevant; in modernen werden bereits optimale Einstellungen verwendet). Auf Netzwerkkarten mit besseren Chips (Intel, Broadcom u. a.) tritt das Problem ebenfalls auf, besonders über 500 Mbit/s. Bonding zweier Netzwerkkarten löst es ebenfalls nicht.

Lösung:

Für Empfang und Übertragung von UDP-Multicast empfehlen wir Netzwerkkarten mit SFP+-Anschluss, da sie leistungsfähigere Chips verwenden. Eine günstige Karte der Klasse Intel X520-DA1/2 (Intel 82599ES) reicht aus — ihr Einsatz beseitigt das Problem von UDP-Multicast-Verlusten oberhalb 1 Gbit/s vollständig.

Als Nebeneffekt sinkt die CPU- und GPU-Last bei Verwendung des Transkoders deutlich.

5.5 Empfehlungen zur Netzwerkkonfiguration für Multicast

Netzwerkparameter in `/etc/sysctl.conf` setzen:

```
net.core.rmem_default=8388608
net.core.rmem_max=16777216
```

Anwenden:

```
sudo sysctl --system
```

5.6 Flussonic und SRT

Beispiel einer SRT-URL für den Empfang auf „Flussonic“:

`srt://Stream_IP:port?streamid=flussonic`

Hier ist **streamid** der Login des Clients in „Flussonic“, der unter „Konfiguration – Pair-Einstellungen“ vergeben wird.

Beim Hinzufügen eines neuen Peers genügt die Angabe des Logins — im Test-Link steht „flussonic“. Die «Flussonic»-Software erzeugt *streamID* automatisch bei SRT, und wird im empfangenden Link das Login *streamID* nicht angegeben, wird der Stream nicht empfangen, und der «Perfect Streamer» kann ihn nicht ausliefern.

Analog lässt sich *streamID* als Login/Passwort setzen, indem in „Perfect Streamer“ ein Peer mit dem Wert im Feld **Login or IP** angelegt wird: `!#::u=1234567890,password=1234567890`

und in „Flussonic“ den Link eintragen: `srt://Stream_IP:port?streamid=!#::u=1234567890,password=1234567890`

Es ist möglich, *streamid/login* im IP-Adressformat in „Perfect Streamer“ anzugeben:

- 192.168.1.1 (einzelne IP)
- 192.168.1.1-192.168.1.254 (IP-Bereich; im Peer muss die Option **Login is IP** aktiviert sein)

In beiden Fällen sieht der IP-basierte Empfangs-Link in „Flussonic“ so aus: `srt://Stream_IP:port?streamid=*`

Es erfolgt eine Bindung an die Client-IP; der Empfang über diesen Link ist nur von dieser IP (oder dem IP-Bereich) aus möglich.

KAPITEL 6

Werkzeuge

6.1 TS Analyze Perfect Streamer Toolkit v2.2 — TR 101 290

Teil des **Perfect Streamer Toolkit** — https://pstreamer.tv

Konsolen-**Analyzer für MPEG-TS-Transportströme** mit Konformitätsprüfung nach **ETSI TR 101 290 V1.4.1** und Validierung des T-STD-Puffermodells aus **ISO/IEC 13818-1**.

Liest **UDP-Multicast/-Unicast** oder **TS-Dateien**, erkennt PCR-PIDs automatisch über PAT/PMT und gibt einen ausführlichen oder kurzen Bericht auf stdout aus.

6.1.1 Was geprüft wird

Der Analysator durchläuft jedes TS-Paket und meldet Verstöße:

- **Priority 1** (TS-Dekodierbarkeit): TS-Sync, Sync-Verlust, Vorhandensein und CRC von PAT/PMT, Continuity Counter, PID-Vorhandensein
- **Priority 2** (empfohlenes Monitoring): TEI-Indikator, CRC-Fehler, PCR-Wiederholung/-Genauigkeit/-Diskontinuität, PTS-Intervall, CAT-Vorhandensein
- **Priority 3** (erweitertes Monitoring): Intervalle NIT/SDT/EIT/TDT, unreferenzierte PIDs, Über-/Unterlauf der T-STD-Buffer

Zusätzlich werden ausgegeben:

- **PCR-Genauigkeit** bis ±500 ns — Regression über Byte-Positionen
- **PCR-Drift** in ppm (nur Live-Modus)
- **T-STD-Pufferspeichermodell** für jeden Elementarstream (Live-Modus)
- Validierung der **SI-Sektionsgrößen** gegen ISO/EN-Grenzwerte mit Warnungen zur EIT-on-STB-Kompatibilität (>1024 B)
- **UDP IAT** (Inter-Arrival Time) — Jitter-Statistik auf Datagramm-Ebene (nur Live-Modus)

6.1.2 Nutzung

```
ts_analyze [options] <input>
```

Eingänge

Formular	Beschreibung
`udp://239.1.1.1:1234`	UDP multicast
`udp://eth0@239.1.1.1:1234`	UDP-Multicast auf der angegebenen Schnittstelle
`udp://192.168.1.100:1234`	UDP unicast
`udp://lo@127.0.0.1:12655`	UDP-Unicast auf Loopback (Testquelle)
`/path/to/file.ts`	Lokale TS-Datei

RTP-gekapseltes UDP wird automatisch erkannt und entpackt.

Optionen

Option	Beschreibung	Standard
`-t, --time <sec>`	Analysedauer in Sekunden	`30`
`-s, --short`	Zusammenfassender Kurzbericht	–
`-f, --full`	Ausführlicher Bericht	ja
`-b, --bitrate <Mbps>`	TS-Bitrate-Hinweis für den Dateimodus	`38.8`
`-p, --pcr-pid <pid>`	Nur die angegebene PCR-PID analysieren (dezimal oder `0xHHHH`)	automatisch
`-l, --pcr-limit <ms>`	PCR-Wiederholungsfehler-Limit in ms	`40`
`--no-eit`	EIT-Analyse überspringen — P3.7..P3.10 werden als `N/A` ausgewiesen; EIT-Beitrag zu P2.2 und Sektionsgrößen-Summe entfällt	EIT aktiviert
`--no-nit`	NIT-Analyse überspringen — P3.1, P3.2 werden als `N/A` ausgewiesen; NIT-Beitrag zu P2.2 und Sektionsgrößen-Summe entfällt	NIT aktiviert
`--no-color`	ANSI-Farben in der Ausgabe deaktivieren	Farben aktiviert
`--xml`	Strukturiertes XML auf stdout (ohne stderr)	Text
`-h, --help`	Hilfe anzeigen	–

Beispiele

```
# 30-second TR 101 290 check on a multicast stream
ts_analyze -t 30 udp://239.10.10.1:1234

# short summary, save to log (no color)
ts_analyze -s --no-color -t 60 udp://239.10.10.1:1234 > report.txt

# analyze a TS file
ts_analyze -t 30 recording.ts

# analyze only a single PCR PID
ts_analyze -p 0x0100 -t 30 udp://239.10.10.1:1234

# machine-readable XML for CI / monitoring
ts_analyze --xml -t 30 udp://239.10.10.1:1234 > result.xml

# skip EIT/NIT analysis (e.g. for streams where they are intentionally absent)
ts_analyze --no-eit --no-nit -t 30 udp://239.10.10.1:1234
```

EIT- oder NIT-Analyse deaktivieren

In manchen Streams werden EIT oder NIT bewusst weggelassen (geschlossene Netze, Lab-Quellen, OTT-only contribution etc.). Die entsprechenden P3-Prüfungen aus TR 101 290 ergeben dann immer FAIL beim OVERALL-Gate — das ist nur Rauschen.

Verwenden Sie `--no-eit` und/oder `--no-nit`, um diese Prüfungen zu deaktivieren:

Flag	Übersprungene Prüfungen	Nebenwirkungen
`--no-eit`	P3.7 (EIT actual P/F), P3.8 (EIT other P/F), P3.9 (EIT actual schedule), P3.10 (EIT other schedule)	EIT-Sektionen werden nicht zusammengesetzt, ihr Beitrag zu P2.2 (CRC) und zur SI-Sektionsgrößen-Summe ist null. Die Warnung zur EIT-on-STB-Kompatibilität wird unterdrückt.
`--no-nit`	P3.1 (NIT actual), P3.2 (NIT other)	NIT-Sektionen werden nicht zusammengesetzt, ihr Beitrag zu P2.2 (CRC) und zur Sektionsgrößen-Summe ist daher null.

Übersprungene Prüfungen erscheinen im Bericht als `N/A` mit der Markierung `disabled (--no-eit)` / `disabled (--no-nit)`, im XML als `applicable="false" result="N/A"`. Im Kurzbericht wird statt eines Fehlerzählers `NIT=off` / `EIT=off` angezeigt.

Die Flags betreffen nur die Verarbeitung von EIT (PID `0x0012`) und NIT (PID `0x0010`) — alle anderen TR-101-290-Prüfungen (P1.x, P2.x, SDT, TDT, CAT, T-STD, PCR-Drift, IAT) laufen wie gewohnt.

XML-Ausgabemodus (`--xml`)

`--xml` veranlasst den Analyzer, ein einziges, eigenständiges UTF-8-XML-Dokument auf **stdout** auszugeben. Sämtliche begleitende Information (Banner, «Stream locked», «PCR PIDs discovered», sekündlicher Fortschritt, Capture-Zusammenfassung, Warnung bei kurzer Laufzeit) wird unterdrückt; **stderr bleibt leer**, sofern kein echter Fehler auftritt (Eingang nicht öffenbar, keine PCR-Daten, Stream zu kurz, Unterbrechung durch Signal). ANSI-Farben werden zwingend deaktiviert.

Der Exit-Code entspricht dem im Textmodus: `0` bei OVERALL=PASS, `65` bei OVERALL=FAIL, dazu die Standardfehler-/Signalcodes (1, 2, 3, 130, 143).

XML-Struktur der obersten Ebene:

```
<?xml version="1.0" encoding="UTF-8"?>
<ts_analyze version="2.2">
  <source>udp://239.1.1.1:5000</source>
  <timestamp>2026-04-30T12:00:00+0300</timestamp>
  <duration_s>30.00</duration_s>
  <packets total="..." null="..."/>
  <ts_bitrate_mbps>20.012</ts_bitrate_mbps>

  <programs>
    <program number="1" pmt_pid="0x0100" pcr_pid="0x0101">
      <es pid="0x0101" stream_type="0x1b" name="H.264/AVC"/>
      <es pid="0x0102" stream_type="0x03" name="MPEG-2 Audio"/>
    </program>
  </programs>

  <tr101290>
    <check id="1.1" name="TS Sync Byte Error" applicable="true" errors="0" result=
↪"PASS"/>
    ...
    <check id="2.3" name="PCR Repetition Error"
           applicable="true" errors="0" result="PASS" soft_violations="3"/>
    ...
    <check id="3.4" name="Unreferenced PIDs" applicable="true" count="2" result="INFO
↪"/>
    ...
  </tr101290>

  <si_section_size oversize_total="0" eit_stb_warn_total="12">
    <table name="EIT_actual_pf" max_bytes="1380" std_limit="4096"
           oversize="0" eit_stb_warn="12" result="WARN_STB"/>
    ...
  </si_section_size>

  <iat datagrams="33252" intervals="33251" min_ms="0.002" max_ms="5.009"
       avg_ms="0.150" stddev_ms="0.295" p95_ms="0.872" p99_ms="1.076"
       max_jitter_ms="4.272" gap_threshold_ms="100.0" gaps_over_threshold="0"/>

  <pcr_pids>
    <pcr_pid value="0x0101" sid="1" pcr_count="1500" discontinuities="0"
             estimated_bitrate_mbps="18.750">
      <interval samples="1499" min_ms="19.812" max_ms="20.195"
                avg_ms="20.001" p95_ms="20.102"
                iso_hard_violations="0" tr_soft_violations="0"
                rec_violations="0" result="PASS"/>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      <accuracy samples="1499" min_ns="-148.3" max_ns="201.7" abs_max_ns="201.7"
                avg_ns="2.1" stddev_ns="45.6" p95_ns="102.3"
                violations="0" result="PASS"/>
      <drift measured="true" ppm="0.618" limit_ppm="30"
             verdict_mode="informational" result="PASS"/>
      <tstd overflows="0" underflows="0" max_fill_bytes="34218" result="PASS">
        <es_buffer es_pid="0x0101" stream_type="0x1b"
                   capacity_bytes="3000000" measuring="true"
                   es_bitrate_mbps="15.200" max_fill_bytes="34218"
                   overflows="0" underflows="0"/>
      </tstd>
    </pcr_pid>
  </pcr_pids>

  <unreferenced_pids count="2">
    <pid value="0x01ff"/>
    <pid value="0x0200"/>
  </unreferenced_pids>

  <overall result="PASS"/>
</ts_analyze>
```

Wichtige Konventionen:

- Alle PIDs werden als `0xHHHH` (4-stelliger Hex mit Präfix `0x`) formatiert.
- Das Attribut `result` nimmt die Werte `PASS` / `FAIL` / `WARN` / `N/A` / `INFO` / `SKIP` / `ok` / `WARN_STB` an.
- Für nicht anwendbare Prüfungen (Datei-Modus, kein Scrambling etc.) erscheint das Element dennoch mit `applicable="false"` und `result="N/A"`, damit Schema-Konsumenten eine stabile Form sehen.
- `<drift>` trägt `verdict_mode="informational"` für $30\ \text{s} \le T < 300\ \text{s}$ und `verdict_mode="hard"` für $T \ge 300\ \text{s}$; `result="SKIP"` für Läufe unter 30 s.
- `<iat>` fehlt bei Läufen im Dateimodus.
- `<overall>` spiegelt dasselbe Gate wie die OVERALL-Zeile im Textbericht wider und entspricht dem Prozess-Exit-Code.

Die Ausgabe ist wohlgeformtes XML (validierbar mit `xmllint --noout`); kann ohne Parser-Anpassung direkt in XSLT, Python `lxml` usw. eingespeist werden.

6.1.3 Bericht lesen

Statusfarben

Status	Farbe	Wert
PASS / ok	grün	Prüfung entspricht dem Standard
WARN / WARNING / WARN(STB)	gelb	Weiche Verletzung, beeinflusst OVERALL nicht
FAIL / ERROR	rot	Standardverletzung, beeinflusst OVERALL
INFO / NOTE / N/A	standardmäßig	Nur informativ

Verwenden Sie `--no-color` beim Umleiten in Log-Dateien oder Nicht-ANSI-Terminals.

Mindestanalysedauer

Dauer	Abdeckung	Exit-Code
< 2 s	Zu wenig — Analyse wird abgelehnt	**3** (Fehler)
2–10 s	P1 + P2 sind verlässlich; einigen P3-Prüfungen können Daten fehlen	0 + WARN
10–30 s	P1 + P2 + die meisten P3; TDT (30 s) reicht ggf. nicht	0 + NOTE
≥ 30 s	Vollständige Abdeckung aller TR-101-290-Prüfungen	0

Die Standardlaufzeit beträgt **30 Sekunden** — ausreichend für vollständige TR-101-290-Abdeckung. Verwenden Sie `-t <sec>` zur Verlängerung (z. B. für Abnahmetests bei PCR-Drift) oder Verkürzung (schnelle smoke-Prüfungen).

Während der Analyse aktualisiert der Analyzer jede Sekunde eine einzeilige Fortschrittsanzeige auf stderr:

```
Progress: 47.3%  (14.2s / 30.0s, 330614 packets)
```

Die Zeile nutzt Carriage-Return (`\r`), um in einem Terminal als eine Zeile zu bleiben; mit `2>/dev/null` lässt sich stderr unterdrücken.

PCR-Drift-Verdikt — zweistufiges Fenster

Die PCR-Taktfrequenztoleranz nach **ISO/IEC 13818-1 §2.4.2.1** und **ETSI TR 101 290** beträgt **±30 ppm**. Der Drift-Wert im Bericht entsteht durch lineare Regression der kumulativen PCR-Sekunden gegenüber der Wanduhr-Eintreffzeit; der statistische Fehler nimmt wie 1 / T^(3/2) ab — daher muss das Analysefenster lang genug sein, dass das Messrauschen deutlich unterhalb der ±30-ppm-Grenze liegt.

Daher koppelt der Analysator das Drift-Verdikt an die Analysedauer:

Fenster	Verdikt bei Drift außerhalb der Toleranz	Auswirkung auf OVERALL
`T < 30 s`	**skipped** (Rauschen dominiert die Grenze von ±30 ppm)	keine
`30 s ≤ T < 300 s`	**WARN** — nur informativ	keine
`T ≥ 300 s`	**FAIL**	OVERALL = FAIL, exit 65

`300 s` ist das Abnahmetest-Fenster nach **DVB TR 101 297** — lang genug, damit selbst ein bursty/loopback-Lieferpfad unter 1 ppm Messrauschen mittelt; ein Out-of-Spec-Ergebnis spiegelt dann den Encoder-Takt wider, nicht das Netz. Der vollständige Bericht zeigt die aktuelle Stufe in der Zeile `Verdict mode` des PCR-DRIFT-Blocks.

Für ein hartes PASS/FAIL-Verdikt zur Drift mit `-t 300` oder länger ausführen.

Empfehlungen zur Quellenqualität (informativ; Verdiktstufen ändern sich nicht):

Quelle	Minimales Fenster für ±5 ppm	Für ±2 ppm	Abnahme
Broadcast-Multicast (CBR-Netz, Jitter < 100 µs)	**30 s**	60 s	5 min
Stabiles IP-Netz (Jitter < 200 µs)	**30 s**	2 min	5–10 min
Loopback / bursty Sender (UDP-Unicast auf `lo`)	**5 min**	15 min	30 min
Kalibrierung / Labormessung	—	30 min	1+ Stunde

Beispiele:

```
# Quick PCR drift check on a real broadcast multicast (30 s)
ts_analyze -t 30 -s udp://239.1.1.1:5000

# Reliable check on a loopback source (5 min)
ts_analyze -t 300 -s udp://lo@127.0.0.1:12655

# Lab acceptance (30 min, full report to file)
ts_analyze -t 1800 -f --no-color udp://239.1.1.1:5000 > acceptance.txt
```

Wenn derselbe Stream in mehreren kurzen Fenstern analysiert wird und der Drift-Wert zwischen den Fenstern um mehr als wenige ppm variiert, liegt der Engpass im Delivery-Jitter (Pacing des Senders oder Netz), nicht am Encoder-Takt — Fenster vergrößern.

Exit-Codes

Code	Wert
`0`	Analyse abgeschlossen, **OVERALL = PASS**
`1`	Argument- oder Eingabefehler
`2`	Stream enthält keine PCR-Daten
`3`	Streamdauer unter dem Minimum von 2 s
`65`	Analyse abgeschlossen, **OVERALL = FAIL** — Verstoß gegen TR 101 290 / ISO 13818-1
`130`	Abgebrochen durch **SIGINT** (Ctrl+C) — Analyse abgebrochen, kein Bericht
`143`	Abgebrochen durch **SIGTERM** — Analyse abgebrochen, kein Bericht

`65` ist `EX_DATAERR` aus POSIX `<sysexits.h>` — „input data was incorrect“. In CI/Monitoring nutzbar als Gate für die Stream-Konformität:

```
ts_analyze -s -t 60 udp://239.1.1.1:5000 || {
    case $? in
        65)  echo "stream does not conform — see report" >&2 ;;
        130) echo "interrupted by user" >&2 ;;
        *)   echo "tool error" >&2 ;;
    esac
}
```

Die Codes 130/143 folgen der POSIX-Shell-Konvention `128 + signal_number`, sodass `$?` nach `Ctrl+C` mit dem übereinstimmt, was bash für jeden durch SIGINT/SIGTERM beendeten Prozess zurückgibt. Bei Unterbrechung gibt der Analyzer eine Zeile auf stderr aus (`Analysis`

`interrupted by signal N — no report produced.`) und überspringt die Berichtserzeugung vollständig.

6.1.4 Beispielausgabe

Vollständiger Bericht (Auszug)

```
================================================================================
  MPEG-TS ANALYZER v2.2 — TR 101 290 FULL REPORT
================================================================================
  Source     : udp://239.1.1.1:5000
  Duration   : 30.00 s
  Packets    : 398936 total, 12045 null
  TS bitrate : 20.012 Mbit/s
--------------------------------------------------------------------------------

================================================================================
  TR 101 290 — PRIORITY 1 (TS decodability)
================================================================================
  |  1.1   TS Sync Byte Error            :        0 errors  PASS
  |  1.4   Continuity Count Error        :        0 errors  PASS
  |  1.6   PID Error (5s absence)        :        0 errors  PASS
  ...

================================================================================
  TR 101 290 — PRIORITY 2 (recommended monitoring)
================================================================================
  |  2.3   PCR Repetition Error          :        0 errors  PASS
  |  2.5   PCR Accuracy Error            :        0 errors  PASS
  ...

================================================================================
  OVERALL COMPLIANCE: PASS  — stream is TR 101 290 compliant
================================================================================
```

Kurzbericht

```
MPEG-TS Analyzer v2.2
TR 101 290 Summary | udp://239.1.1.1:5000 | 30.0s
--------------------------------------------------------------------------------
↪--------------
  P1: sync=0 CC=0 PAT=0 PMT=0 PID=0  P2: TEI=0 CRC=0 PTS=0  P3: NIT=0 SDT=0 EIT=0␣
↪TDT=0 unref=2
  IAT: dgrams=33252 avg=0.150 ms max=5.009 ms p99=1.076 ms gaps>100ms=0
--------------------------------------------------------------------------------
↪--------------
PCR PID       SID     Count     Intv max      Jitter max    Drift     Interval ␣
↪Accuracy  T-STD
--------------------------------------------------------------------------------
↪--------------
0x0101        1       1500      20.195 ms     76.4 ns       0ppm      PASS      PASS ␣
↪    PASS
--------------------------------------------------------------------------------
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
↪--------------
OVERALL: PASS
```

6.1.5 Anmerkungen

- **Dateimodus**: PCR-Drift, T-STD-Puffermodell und UDP IAT werden nicht gemessen — sie erfordern eine Echtzeitreferenz. Alle anderen Prüfungen funktionieren in beiden Modi.
- **Ein Transportstream**: pro Lauf wird ein MPTS oder SPTS analysiert.

6.2 MPTS Migrate Perfect Streamer Toolkit v1.0 — MPTS-Identitätsmigration

Teil des **Perfect Streamer Toolkit** — https://pstreamer.tv

Erfasst die DVB-SI/PSI-Identität eines laufenden Multi-Programm-MPEG-TS-Streams (MPTS) und reproduziert sie auf einer Perfect-Streamer-Instanz (PSS) auf demselben Host. Ergebnis: Endkunden-Empfänger (STB / TV) arbeiten **ohne erneuten Sender-Suchlauf** nach der Migration oder dem Failover weiter.

6.2.1 Voraussetzungen

Stellen Sie vor dem Start des Tools sicher, dass:

- **PSS läuft** auf demselben Host (oder auf einem über `--pss-base` erreichbaren Host). Das Tool sucht `pss` in `/proc` und liest `pss.json`, um den Admin-Port (Standard 43971) zu ermitteln.
- **MPTS-Quelle erreichbar**, falls eine Erfassung geplant ist (Modi 1, 2, save+apply): die als Positionsargument `<input>` übergebene URL muss einen MPEG-TS-Stream liefern. Für UDP multicast — IGMP / Firewall müssen den Empfang erlauben; bei Dateien — der Pfad muss existieren.
- **Ziel-MPTS bereits im PSS konfiguriert**: das Tool legt keine neuen Streams an. Ein MPTS-Objekt sowie mindestens so viele SPTS-Feeder, wie das Inventar Services hat, müssen vorab existieren. Services ohne verfügbaren Feeder erscheinen im Dialog und können übersprungen werden.
- **Die HTTP-Admin-API ist auf localhost offen** — fürs Lesen von `/data/stream` (verify) und Schreiben in `/config/stream` (apply).

6.2.2 Was wird migriert

Alle vom Empfänger sichtbaren Identifikatoren auf Transportstream- und Service-Ebene:

- **Transport-Stream**: TSID, ONID, network ID, network name, **provider name** (wird mux-weit als `sdt-provider-name` angewendet, wenn alle Quell-Services denselben Wert haben), Delivery-Descriptor (Parameter terrestrischer/Kabel-/Satelliten-Übertragung), PAT/SDT/NIT-Versionen
- **Pro Service**: `service_id`, `pmt_pid`, `pcr_pid`, `service_type`, Servicename, logische Kanalnummer (LCN), free-CA-Flag, EIT-present-/EIT-schedule-Flags
- **Elementarstreams**: PIDs (Identity-Remap angewandt — siehe *Einschränkungen*), Streamtypen, Sprachtags
- **Conditional Access**: CA-Deskriptoren auf Programm- und ES-Ebene

Service- und Provider-Namen in nicht-ASCII-DVB-Kodierungen (z. B. ISO-8859-5 für Kyrillisch) werden automatisch nach UTF-8 dekodiert.

Per-Service-ES-PID-Remap wird als Identity-Paare (`mpegts-pid-old ≡ mpegts-pid-new`) für jedes PCR/video/audio/teletext/data PID gebildet, sodass die resultierende multiplexte Ausgabe die Quell-PIDs **byte-exact** beibehält. Legacy-Empfänger, die das PMT nach dem ersten Suchlauf cachen, arbeiten ohne Neukonfiguration weiter.

6.2.3 Anwendungsfälle

- **Failover**: Umschalten der Decoder vom primären auf den Backup-MPTS bei gleichbleibenden Kanälen am Empfänger
- **Hardware-Migration**: Verlagerung eines laufenden Multiplexes von einem PSS-Host auf einen anderen ohne Instruktionen für Zuschauer
- **Snapshot vor/nach Update**: Multiplex vor dem PSS-Upgrade erfassen und danach erneut anwenden, um bit-identische SI/PSI zu garantieren
- **Manuelle Bearbeitung und erneutes Anwenden**: Erfassen, `migrate.json` bearbeiten (Services umbenennen, LCN ändern, `service_type` korrigieren) und erneut anwenden
- **Dry-Run-Review**: jeder HTTP-POST, der gesendet *würde*, wird ausgegeben, ohne PSS zu beeinflussen

6.2.4 Schnellstart

```
# Capture from a live stream and apply to local PSS in one run
mpts_migrate udp://239.1.1.1:1234

# Capture, save to migrate.json and apply
mpts_migrate -s udp://239.1.1.1:1234

# Capture only — write to file, do not apply
mpts_migrate -o backup.json udp://239.1.1.1:1234

# Apply a previously saved JSON
mpts_migrate -i backup.json
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
# No arguments — load ./migrate.json and apply
mpts_migrate

# Preview what apply would do, without changes
mpts_migrate -i backup.json --dry-run
```

6.2.5 Arbeitsablauf

1. **Erfassung** — das Tool öffnet den Stream, parst PAT / PMT / SDT / NIT / EIT für `-t` Sekunden (Standard 30) und erstellt das Inventar; die Gesamtbitrate wird gemessen.
2. **(Optional) Speichern** — mit `-s` oder `-o` wird das Inventar zur späteren Wiederverwendung als JSON gespeichert.
3. **PSS-Suche** — findet ein laufendes PSS per `/proc`-Scan und liest `pss.json` zur Ermittlung des Admin-Ports (Standard 43971); `--pss-base http://host:port` umgeht die Auto-Discovery.
4. **Mapping-Bestätigung** — ein interaktiver Dialog fragt, wie jeder erfasste Service einem vorhandenen SPTS-Feeder zuzuordnen ist; `--non-interactive` akzeptiert die Vorschläge und bricht bei Konflikt ab; `--target-mpts <id>` überspringt die MPTS-Auswahl.
5. **Auto-Unpause der Feeder** — für jeden zugeordneten SPTS/muxer-output sendet das Tool `{"pause":false}`, falls der Feeder pausiert war, damit der Multiplexer nach dem Apply tatsächlich Daten empfängt.
6. **Auto-Bitrate-Anpassung** — wenn `captured_bitrate × (1 + headroom%)` `mpegts-output-bitrate` des Ziel-MPTS übersteigt, hebt das Tool diese Grenze auf dem PSS per einzelnem POST an (aufgerundet auf das nächste Vielfache von 1000 kbps). Deaktivierbar über `--no-bitrate-adjust`.
7. **Planung** — vergleicht das Inventar mit dem aktuellen Baum `/config/stream` auf dem PSS und bereitet HTTP-POST-Requests nur für die abweichenden Felder vor. Das ES-PID-Remap wird als Identity-Paare (`mpegts-pid-old` ≡ `mpegts-pid-new`) erzeugt, sodass die multiplexte Ausgabe jedes Quell-PID unverändert beibehält — einschließlich PCR, video, audio, teletext, SCTE-35, DSM-CC.
8. **Apply** — sendet die geplanten POST-Anfragen und schaltet anschließend MPTS pause/unpause um, damit PSS die Konfiguration neu liest; bei `--dry-run` wird der Plan nur gedruckt.
9. **Verify** (standardmäßig aktiv, auch wenn der Plan leer ist) — erfasst das resultierende MPTS über einen der UDP-Outputs des PSS und vergleicht mit dem Ziel; kritische Abweichungen (TSID, ONID, service_id, name, type, LCN) → exit 5.

Eine erneute Ausführung der gesamten Pipeline ist **idempotent**: Beim zweiten Lauf wird `no changes needed` gemeldet, wenn PSS dem Inventar bereits entspricht; verify bestätigt das durch erneute Erfassung.

6.2.6 CLI-Optionen

Modusauswahl

Option	Beschreibung	Standard
`-f, --file <path>`	Pfad zur Migrations-JSON (Standardimport ohne `-i` und Speicherziel mit `-s`)	`./migrate.json`
`-s, --save`	Erfasstes Inventar in eine Migrations-Datei speichern (kombinierbar mit Stream-Input; Apply wird trotzdem ausgeführt)	—
`-o, --output <file>`	Nur Erfassung: in Datei schreiben, ohne Apply	—
`-i, --input <file>`	Nur Apply: Datei laden, ohne Erfassung	—

Erfassung

Option	Beschreibung	Standard
`-t, --time <sec>`	Maximale Erfassungsdauer	`30`
`-b, --bitrate <Mbps>`	TS-Bitrate-Hinweis für das Pacing im Dateimodus	`38.8`

Apply / Verify

Option	Beschreibung	Standard
`--target-mpts <id>`	MPTS-Auswahl überspringen; auf diese Stream-ID am PSS anwenden	—
`--non-interactive`	Dialogvorschläge automatisch akzeptieren; bei Konflikt abbrechen	—
`--dry-run`	POST-Plan ausgeben, nichts senden	—
`--pss-base <url>`	PSS-Auto-Discovery überschreiben (z. B. `http://host:43971`)	automatisch
`--no-verify`	Post-Apply-Erfassung und Diff überspringen	verify aktiviert
`--verify-time <s>`	Erfassungsfenster für die Verifikation	`10`
`--no-bitrate-adjust`	`mpegts-output-bitrate` am Ziel-MPTS nicht anheben	Anhebung aktiviert
`--bitrate-headroom <pct>`	Bitrate-Sicherheitsreserve über dem gemessenen Wert beim Anpassen	`15`

Sonstiges

Option	Beschreibung
`-v, --verbose`	Detailliertes Log (jeder HTTP-POST und Dialogzweig)
`-h, --help`	Hilfe anzeigen und beenden

6.2.7 Migrations-Datei (`migrate.json`)

Menschenlesbares JSON mit `format_version: 1`. Standardpfad `./migrate.json`; mit `-f` überschreibbar. Beispielstruktur:

```
{
  "format_version": 1,
  "tool": "mpts_migrate",
  "capture": {
    "source": "udp://239.1.1.1:1234",
    "captured_at_utc": "2026-04-30T08:15:00Z",
    "duration_s": 8.2,
    "packets": 109344
  },
  "transport_stream": {
    "transport_stream_id": 1234,
    "original_network_id": 8442,
    "network_name": "Operator",
    "delivery": { "type": "terrestrial", "frequency_khz": 522000 }
  },
  "services": [
    {
      "service_id": 1, "pmt_pid": 256, "pcr_pid": 256,
      "service_type": 1, "service_name": "Channel 1",
      "provider_name": "MyProvider", "logical_channel_number": 101,
      "free_ca_mode": false,
      "elementary_streams": [
        { "pid": 256, "stream_type": 27, "language": "rus" }
      ]
    }
  ]
}
```

Die Datei lässt sich vor einem erneuten Apply bearbeiten: Services umbenennen, LCN ändern, `service_type` umschalten, `network_name` anpassen — `mpts_migrate -i migrate.json` sendet nur die geänderten Felder.

6.2.8 Verbindung zu PSS

- **Auto-Discovery**: `/proc/<pid>/comm` nach `pss` durchsuchen, dessen `--config`-Datei lesen und `web-server.bind-port` (Standard 43971) übernehmen.
- **Manuell**: `--pss-base http://host:port` umgeht die Auto-Discovery vollständig. Nützlich für ein entferntes PSS oder wenn `--dry-run` ohne lebendes PSS einen Plan erstellen soll.
- Die Admin-REST-API erfordert keine Authentifizierung auf `localhost`.

6.2.9 Verifikation

Wenn `--no-verify` **nicht** gesetzt ist (Standard), führt die Utility nach Anwendung des Plans Folgendes aus:

1. Sucht einen UDP-Localhost-Ausgang am Ziel-MPTS oder fügt vorübergehend einen unter `127.0.0.1:<auto>` mit Marker `added by mpts_migrate for verification` hinzu.
2. Erfasst den Live-MPTS über diesen Ausgang für `--verify-time` Sekunden (Standard 10).
3. Vergleicht das erfasste Inventar mit dem Ziel: - **Kritische** Abweichung (TSID, ONID, service_id, name, type, LCN) → Exit-Code **5**. - **Weiche** Abweichung (PMT PID, PCR PID, ES PID) → wird als Warning ausgegeben, Exit-Code 0.
4. Wenn der Ziel-MPTS überlastet ist (Output-Bitrate ≥ konfigurierter `mpegts-output-bitrate`), wird `WARNING: target MPTS is overloaded` ausgegeben — siehe *Bitrate adjust* unten.

6.2.10 Dry-run

`--dry-run` gibt jede HTTP-Anfrage aus, die das Tool *senden würde* (Pfad, JSON-Body), schickt aber keine. Nützlich für:

- Plan-Review mit dem Bediener vor dem Commit.
- Erzeugung reproduzierbarer Änderungsmengen in CI / Change-Management.
- Arbeit bei nicht erreichbarem PSS (mit `--pss-base http://host:port` kombinieren).

Dry-Run führt keine Verifikation aus.

6.2.11 Bitrate adjust

Wenn `captured_bitrate × (1 + headroom%)` `mpegts-output-bitrate` des Ziel-MPTS übersteigt, hebt das Tool standardmäßig diese Grenze auf dem PSS an, bevor SI/PSI-Änderungen angewendet werden. Ohne ausreichende Reserve verliert ein überlasteter MPTS Daten niedriger Priorität — typische Symptome: Audio-Verlust bei Radio-Services, gelegentliche EIT-CRC-Fehler. Deaktivierbar über `--no-bitrate-adjust`; die Reserve wird über `--bitrate-headroom <pct>` gesteuert (Standard 15).

6.2.12 Exit-Codes

Code	Wert
`0`	Erfolg — Apply und (falls aktiviert) Verifikation liefen sauber durch. Auch ein erfolgreicher `--dry-run` liefert diesen Code.
1	Argument-, Datei- oder Erkennungsfehler
2	In der Quelle wurde keine PAT gefunden — Eingabe ist kein gültiger MPEG-TS oder das Erfassungsfenster ist zu kurz
3	Ein oder mehrere HTTP-POSTs des Apply schlugen fehl
4	Apply lief, aber der Ziel-MPTS ist nicht in den Zustand *Running* zurückgekehrt
5	Verifikation fehlgeschlagen — der erfasste Stream weicht in kritischen Feldern vom Ziel ab

6.2.13 Einschränkungen und Stolperfallen

- **Das PSS muss das Ziel-MPTS und die Feeder bereits halten**: das Tool legt keine neuen Streams an. Das Ziel-MPTS sowie mindestens so viele SPTS-Feeder, wie das Inventar Services hat, müssen vorab existieren; Services ohne verfügbaren Feeder werden im Dialog übersprungen.
- **Die MPTS-Quelle muss bei der Erfassung erreichbar sein** (Modi 1, 2, save+apply): die als Positionsargument `<input>` übergebene URL muss einen MPEG-TS-Stream liefern; bei UDP multicast müssen IGMP / Firewall den Empfang erlauben.
- **Das ES-PID-Remap wird automatisch angewendet**: das Per-Service-Identity-Remap (`mpegts-pid-old ≡ mpegts-pid-new`) wird für jedes PCR/video/audio/teletext/data PID erzeugt, damit die multiplexte Ausgabe die Quell-PIDs byte-exact beibehält. Das PMT-Layout bleibt über Migrationen hinweg stabil — selbst Legacy-Empfänger (STBs vor 2015, Samsung vor H-Serie, LG vor WebOS 3.0), die PIDs cachen, brauchen keinen erneuten Suchlauf.
- **Der Delivery-Descriptor muss zur tatsächlichen Trägerstrecke passen** für Empfänger, die sich über NIT (DVB-T/T2/C/S) neu abstimmen. Ein nicht passender `delivery`-Block kann den Empfänger auf eine falsche Frequenz leiten.
- **LCN-Konsistenz** zwischen Haupt- und Backup-MPTS ist beim Failover entscheidend — bei Abweichungen verschieben sich Kanalpositionen in der Empfängerliste nach dem Umschalten.
- **Provider name am PSS — mux-weit gemeinsam** (ein einziger `sdt-provider-name` am muxer-input MPTS). Das Tool wendet ihn automatisch an, wenn alle erfassten Services denselben Wert haben; haben verschiedene Services unterschiedliche `provider_name`-Werte, wird ein Warning ausgegeben und das Feld bleibt unverändert — die Entscheidung trifft der Operator.
- **Kein Konfigurationstool für PSS selbst**: `mpts_migrate` berührt nur die SI/PSI-Identitätsfelder, das `pause`-Flag pro Feeder und (optional) `mpegts-output-bitrate`. Encoder, Inputs, Verschlüsselung, Scheduling etc. konfiguriert es nicht.

6.2.14 Diagnose

Symptom	Wahrscheinliche Ursache / Lösung
`Error: no PAT seen in stream`	Quelle ist kein MPEG-TS, IGMP/Firewall blockiert Multicast oder `-t` ist zu kurz
`Error: cannot reach PSS`	`--pss-base http://host:port` verwenden, um die Auto-Discovery zu umgehen
Apply lief, aber das MPTS bleibt pausiert	PSS-Log prüfen; mit `-v` neu starten, um den vollständigen POST-Plan zu sehen
Verifikation meldet kritische Abweichung	goal mit capture im JSON vergleichen; meist ist ein Feeder im Dialog versehentlich dem falschen Service zugeordnet
`WARNING: target MPTS is overloaded`	`mpegts-output-bitrate` am PSS erhöhen oder `--bitrate-headroom <higher %>` setzen; ohne Reserve werden Radio-Audio und PSI-Tabellen beschädigt

KAPITEL 7

Versionshistorie

7.1 Version 1.13.2.444 Beta

31.05.2026

- **OTT (Low-Latency HLS / DASH über CMAF)**: ein neuer *OTT/HLS/LL-HLS/LL-Dash*-Auslieferungsmodus (*ott-hls = 3*) — der integrierte Multiplexer erzeugt fragmentiertes **MP4** / **CMAF** (*fMP4*), auf dessen Basis **MPEG-DASH** (jetzt über CMAF statt MPEG-TS) und Low-Latency **HLS** über einen neuen Endpoint (Pfad *…/llhls/…*) ausgeliefert werden. Der Player beginnt die Wiedergabe, ohne auf das vollständige Segment zu warten: Die **LL-HLS**-Medien-Playlist wird in *partielle Segmente* («parts») aufgeteilt, und es werden ein blockierendes Neuladen der Playlist (der Server hält die Anfrage zurück, bis der nächste part bereit ist) sowie der Preload-Hinweis *EXT-X-PRELOAD-HINT* verwendet.
- **OTT (Low-Latency: Einstellungen und Synchronisation)**: Die Ziel-Dauer eines part wird über die Einstellung *Part Target Duration* festgelegt (ms, im laufenden Betrieb ohne Neustart des Streams übernommen); die Option *Enable TS Chunk* legt fest, ob parallel Legacy-MPEG-TS-**HLS** (Playlist *…/hls/…*) ausgegeben wird — bei Deaktivierung gehen nur fMP4-Segmente auf Festplatte und CPU. Für eine präzise niedrige Latenz wurden den Manifesten *Producer Reference Time* (*prft*) und *UTCTiming* hinzugefügt, die die Medienzeit an **UTC** binden.
- **DVR (Start des Subsystems)**: Ein persistentes Archiv auf der Festplatte wird parallel zum integrierten Live-Segmenter für **HLS** / **MPEG-DASH OTT** geschrieben und nutzt dieselbe Segmentierung sowie dieselben URLs der OTT-Sitzung — der Wiedergabemodus wird über einen Query-Parameter umgeschaltet. Im *OTT/HLS/LL-HLS/LL-Dash*-Modus führt das Archiv zwei unabhängige Indizes — einen für MPEG-TS-Chunks und einen für fMP4-/CMAF-Segmente —, sodass VOD im selben Container wie live ausgeliefert wird. *Vollständige DVR-Beschreibung*.
- **DVR (Wiedergabe)**: VOD über **HLS** und **MPEG-DASH** mittels der Query-Parameter *t=<epoch>* (Startzeitpunkt, *t=0* — ab Beginn des Archivs) und *d=<sec>* (Fensterdauer, leer oder *0* — „bis zum aktuellen Zeitpunkt“) sowie Bindung an **EPG** über *epg=<epoch>* (der Server setzt selbst *start* und *duration* des aktiven Ereignisses als

Fenstergrenzen ein). Eine geschlossene **HLS**-VOD-Playlist mit den Markern *EXT-X-PLAYLIST-TYPE:VOD* und *EXT-X-ENDLIST*; eine statische **DASH MPD** (*@type="static"*, festes *mediaPresentationDuration*) mit automatischer Aufteilung in mehrere *Period*-Elemente bei Aufnahmeunterbrechungen. Anfragen außerhalb des Archivs werden ohne Fehler auf die verfügbaren Grenzen normalisiert.

- **DVR (adaptives VOD)**: Bei adaptiven **HLS**- und **DASH**-Gruppen erscheinen nur Varianten mit einer DVR-Speicherbindung im Manifest, jede Qualität ist eine eigene *Representation* innerhalb gemeinsamer *Period*-Elemente, und Qualitätsumschaltung funktioniert ohne erneutes Öffnen des Manifests.
- **DVR (Schutz und Auslieferung)**: solange eine VOD-Session offen ist, tasten die size-based und die Sliding-Window-Bereinigung die Chunks in ihrem Fenster nicht an (der Schutz wird per Timeout oder *FIN* aufgehoben); ein transparenter Übergang VOD → Live-Edge, wenn der Player die rechte Fenstergrenze erreicht — das Segment wird aus dem Live-Speicher ohne Weiterleitungen ausgeliefert; der VOD-Playlist-Cache liefert wiederholte Anfragen derselben *index.m3u8* / *index.mpd* Byte für Byte identisch (praktisch für **CDN**).
- **DVR Storage (Speichereinstellungen)**: mehrere gleichzeitige Speicher, jeder mit einem *Max Usage*-Schwellenwert, einem *Cleanup Interval*-Intervall, einer *Disk Pressure Grace*-Entprellung, einer *Disk Pressure Cut*-Obergrenze für die Löschung pro Zyklus, einem *Disk Emergency Bytes*-Notfallschwellenwert sowie den Zuständen *Ready* / *Error*.
- **DVR (Stream-Einstellungen)**: *Storage Hours* — Archivtiefe in Stunden mit Sliding-Window-Bereinigung (die Obergrenze ist auf 90 Tage festgelegt), und *Storage Min Hour* — eine geschützte Untergrenze, die die size-based Bereinigung auch unter Disk Pressure nicht entfernt.
- **DVR (Untertitel)**: **WebVTT** wird parallel zu den TS-Chunks ins Archiv geschrieben, mit einem Index pro PID; die VOD-Untertitel-Playlist wird unter denselben URLs ausgeliefert (bei **DASH** wird der Header *X-TIMESTAMP-MAP* on the fly entfernt). Untertitel-Chunks ohne Cue werden nicht auf die Festplatte geschrieben — ein Chunk mit Nullgröße wird beim Lesen synthetisiert, was die Dateisystem-Last bei Kanälen mit sporadischen Untertiteln reduziert.
- **DVR (Wartung)**: ein Hintergrund-Collector „verwaister" Dateien (der erste Lauf etwa eine Minute nach dem Start, danach stündlich und bei *disk pressure*; Schutz vor einem Race mit dem Writer anhand der mtime), monotone Chunk-Nummerierung über Dienst-Neustarts hinweg; behoben wurde ein Modus, in dem die hintergründige volumenbasierte Bereinigung und der Collector möglicherweise nicht starteten.
- **DVR (Beobachtbarkeit und Monitoring)**: *GET /data/dvr-storage-list* liefert für jeden Speicher *State*, *Total / Free / Used Bytes*, *Used %*, *Archived Bytes*, *Pressure Since Sec*, das Kennzeichen einer aktiven Hintergrundoperation (*active-task*: *gc-orphans* / *disk-pressure-trim* / *none*) mit ihrer Dauer und Angaben zu den letzten Bereinigungsläufen sowie eine Liste der gebundenen Streams mit den Attributen *retention-hours*, *archived-sec*, *archived-bytes* und *active*; die Archivgröße wird zusätzlich nach Container (TS / MP4) aufgeschlüsselt. Auf Stream-Ebene gibt *GET /data/stream/<id>* die Metrik *storage-gap-percent* aus (Prozentsatz der zeitlichen Lücken im Archiv), und ihr Histogramm nach Zeit-Buckets wird vom neuen Endpoint *GET /data/dvrstat* geliefert — zum Zeichnen der DVR-Archiv-Skala in der Admin-UI mit Markierung von Aufnahmeereignissen und Untertitelaktivität.
- **OTT (IDR-basierte Segmentierung)**: Die Segmentierung von **HLS** und **MPEG-DASH** unterscheidet ein *IDR* von einem gewöhnlichen *I-Frame* in **H.264**- / **HEVC**-Streams. Bei Closed-GOP-Inhalten werden die Segmentgrenzen an IDR ausgerichtet — jeder Chunk beginnt mit einem echten wahlfreien Zugriffspunkt (*SPS+PPS+IDR*, in **HEVC** zusätzlich

unter Berücksichtigung der separaten *VPS*-NAL), und der Player kann den Stream garantiert ab jedem beliebigen Segment öffnen; bei Open-GOP- / IDR-losen Quellen dient das nächstgelegene I-Frame als Grenze.

- **OTT (Analysator-Metriken)**: neue Metriken am Videostream — *idr-int-max / avg* (IDR-Intervall) und *kf-int-max / avg* (GOP-Intervall). Aus ihrem Verhältnis erkennt der Administrator sofort den Typ der GOP-Struktur: closed-GOP (*idr-int* ≈ *kf-int*) oder open-GOP (*idr-int* fehlt). Die XML/JSON-Schlüsselnamen bleiben zur Abwärtskompatibilität unverändert.
- **OTT HLS (Playlist)**: *EXT-X-VERSION* wird standardmäßig nach dem HLS-Modus gewählt — *OTT/HLS* und *OTT/HLS/LL-HLS/LL-Dash* ergeben *EXT-X-VERSION:6* mit *EXT-X-INDEPENDENT-SEGMENTS* und dem Attribut *CHARACTERISTICS* in *EXT-X-MEDIA TYPE=SUBTITLES* (im *OTT/HLS/LL-HLS/LL-Dash* gibt auch der Legacy-Master *…/hls/…* ein Untertitel-*EXT-X-MEDIA* aus), *Peering/HLS — EXT-X-VERSION:3* zur Kompatibilität mit älteren Clients (der Query-Parameter `?v=` überschreibt den Standard). Der Wert *EXT-X-TARGETDURATION* spiegelt nun die maximale tatsächliche Segmentdauer wider (Abschnitt 4.3.3.1 von **RFC 8216**) und nicht die Einstellung *chunk-min-interval* — bei GOP-ausgerichteter Segmentierung verletzt das Manifest den Standard nicht, und *hls.js* halbiert das Aktualisierungsintervall der Playlist nicht und löst keine falschen *bufferStalledError* aus.
- **HTTP/3 (QUIC)**: Ein integrierter Server auf Basis von **ngtcp2** + **nghttp3** liefert **HLS** und **MPEG-DASH** über **QUIC** — aktiviert durch die Einstellung *HTTP/3 Enable* des Webservers (Port *HTTP/3 Port*, UDP, standardmäßig identisch mit dem HTTPS-Port), unterstützt *0-RTT*. Low-Latency **HLS** / **DASH** werden über **QUIC** inkrementell (chunked) ausgeliefert — *parts* gehen an den Client, sobald sie bereit sind, ohne auf das vollständige Segment zu warten. Auf dem QUIC-Transport werden nur OTT-Routen angenommen; administrative Pfade bleiben auf HTTPS/HTTP. Die echte IP des Clients wird über den internen Header `x-pss-peer-addr` übergeben und in die Zählung aktiver Peers einbezogen, ohne durch die Loopback-Adresse ersetzt zu werden. Das Umschalten von **HLS** / **DASH** auf **HTTP/3** wird auch über den Query-Parameter *?h3* aktiviert — zum testweisen Umschalten einer einzelnen Sitzung ohne Neukonfiguration des Clients.
- **Aktive Peers**: einheitliches OTT-Session-Timeout von 60 Sekunden unabhängig vom Transport; Aktualisierungen des Client-Datensatzes bei Schemawechsel erfolgen nur „aufwärts“ nach Priorität (`http` → `https` → `quic`). Das Attribut *ott-type* in *http-clients* enthält jetzt einen zusammengesetzten Wert der Form `<PROTO>/<scheme>` (*PROTO* = *HLS* / *DASH* / *HTTP*; *scheme* = *http* / *https* / *quic*) — das Admin-UI sieht sowohl das OTT-Protokoll als auch den tatsächlichen Netzwerktransport jedes Clients.
- **PS1 output**: am *PS1*-Ausgang wurde eine reibungslose Verarbeitung von Stream-Eingangswechseln umgesetzt. Bei einem Warteschlangen-Spike während des Quellwechsels werden die ältesten Pakete still verworfen, während die seqID / TS der Clients durchgehend bleiben — empfangende Peers begnügen sich mit dem regulären retr-Mechanismus statt einer Neuinitialisierung der Verbindung mit *StateError*. Der Zähler verworfener Pakete ist in der erweiterten PS1-Ausgangsstatistik sichtbar.
- **SPTS / TR 101 290**: an Eingangsströmen ist ein PCR-Drift-Kompensator aktiviert — ein langsames Abwandern des Quellenreferenzoszillators gegenüber der lokalen Uhr wird durch einen sanften *sync DT*-Versatz im Hintergrund aufgefangen, ohne sichtbare Ruckler am Ausgang. Gesteuert über die Stream-Einstellungen *Sync Drift Compensation* (standardmäßig aktiviert) und *Sync Drift Soft Window* (ms).
- **SPTS / TR 101 290**: eine lineare PCR-Regression über ein breites Fenster misst *drift* (ppm) und *PCR accuracy* (ns gemäß Abschnitt *P2.3*) gegenüber dem Referenztakt. Die Metriken *pcr-drift-max / avg*, *pcr-acc-max-ns* sowie die Intervalle *pcr-int*, *pat-int*, *pmt-int*

werden in *GET /data/stream/<id>* ausgegeben und in die Datenbank der historischen Statistik geschrieben (die neuen Tabellen sind für *Resetting Stat* sichtbar).

- **SPTS / T-STD**: ein Video-Pufferanalysator des Referenzdecoders (**T-STD**, **ISO/IEC 13818-1** §2.4.2). Die *MBn*-Kapazität wird anhand des *stream type* der Video-PID gewählt; die Drain-Rate stabilisiert sich innerhalb einer 1-sekündigen „Einlaufphase“ nach der PCR-Uhr (nicht nach der Systemuhr des Hosts — so reagiert der Analysator nicht auf Pausen des CPU-Schedulers). Die Zähler *tstd-video-overflows / underflows / max-fill / drain-bps* werden in *GET /data/stream/<id>* ausgegeben und fließen in *tr101290-alert* ein.
- **SPTS**: Runtime-Detektor für den Multiplex-Bitratenmodus — das Attribut *bitrate-mode-detected* (*cbr* / *vbr* / *unknown*) auf Basis eines Vergleichs der 5- und 60-Sekunden-Bitraten. Die Prüfungen *pcr-acc* und *tstd-video* in *tr101290-alert* werden bei erkanntem *VBR* automatisch unterdrückt — dort würden sie sonst Fehlalarme erzeugen.
- **Analysator für die Werbeeinblendung (ad-insertion)**: Auf dem eingehenden **SPTS**-Stream wird ein Codec-«Passport» erstellt — ein Video-Passport (vollständiges *SPS*, **H.264**- / **HEVC**-Profil und -Level) und ein Audio-Passport (**MPEG Audio**, **AC-3**, **AAC** in den Formaten *ADTS* und *LATM*) — und es werden **SCTE-35**-Sektionen (*splice_info_section*) mit der Auszeichnung von Splice-Punkten ausgewertet. In *GET /data/stream/<id>* (bei aktivierter kontinuierlicher **SPTS**-Analyse) werden Signale für Zugriffs- und Splice-Grenzen ausgegeben — *GOP*, *RAI*, *splice-point*, **SCTE-35**-Ereignisse; die Einstellung *Splice Point Notify At* legt den Vorlauf der Benachrichtigung über den Einfügepunkt fest. Die Daten sind für die serverseitige Werbeeinblendung aufbereitet.
- **KI-Reklamationsassistent**: ein neuer Endpoint *GET /data/stream/<id>/ai-complaint-prompt* liefert einen fertigen englischsprachigen Prompt für jedes Chat-Modell, der das Modell anweist, ein offizielles Reklamationsschreiben an den Provider mit Auflistung der erkannten Verstöße gegen **TR 101 290** / **ISO/IEC 13818-1** zu verfassen. Der Prompt enthält exakt dieselben Token und gemessenen Werte wie *tr101290-alert*; Streamname und Quell-URI gelangen nicht in den Prompt — es wird der Platzhalter *<Stream Designation>* verwendet, den der Betreiber manuell ausfüllt. Die Sprache des Schreibens wird in der Antwort auf den Prompt gewählt.
- **Webportal (Rollen)**: Servereinstellungen, EPG und die Verwaltung der Liste der Administratorkonten sind nur der Rolle *Admin* erlaubt; die Rolle *RestrictAdmin* kann Streams und Kanäle pausieren, aber die übrigen Einstellungen nicht ändern; die Rolle *Viewer* ist nur zum Ansehen. *POST*-Routen sind standardmäßig gesperrt, und jede neue HTTP-Operation erfordert eine explizite Berechtigung für eine herabgestufte Rolle — der Zugriff wird nicht stillschweigend erweitert.
- **Server (Speicher)**: periodische Rückgabe freien Speichers aus den *glibc*-Arenen an das System (*malloc_trim* alle 30 s) und Begrenzung der Anzahl der Arenen über die Umgebungsvariable *MALLOC_ARENA_MAX* in der *systemd*-Unit — beseitigt das allmähliche Anwachsen des *RSS* im Dauerbetrieb mit Dutzenden Streams, ohne logische Lecks.
- **MPEG-TS-Filter**: die Einstellung *Filter Teletext* verwirft wieder beide Typen von Teletext-PES-Streams (klassisch und subtitles) nach der internen Reklassifizierung im Analysator.
- **MPTS input**: der Transport **RTSP** wurde aus der Liste der für MPTS zulässigen entfernt — **RTSP** ist single-program und nur als SPTS-Quelle anwendbar.
- Weitere Verbesserungen und Fehlerbehebungen.

7.2 Version 1.12.3.433

09.05.2026

- DVB-Scanner für **DVB-S/S2**, **DVB-C** und **DVB-T/T2**: Transponder-Suche und Erstellung einer Programmliste, mit der Möglichkeit, die gefundenen Parameter direkt in den DVB-Adapter-Einstellungen anzuwenden.
- DVB-Scanner: Transponder-Referenzdaten werden aus Dateien im *Enigma2*-Format (*satellites.xml*, *cables.xml*, *terrestrial.xml*) im Einstellungsverzeichnis geladen.
- DVB-Scanner: *blind scan*-Modus für **DVB-S/S2** und **DVB-C/T/T2** — Durchlauf von Frequenzen, Polarisationen und Symbolraten ohne Transponder-Referenz.
- DVB-Scanner: Für jedes gefundene Programm werden *PNR*, Dienstname, Provider, das *scrambled*-Flag (abgeleitet aus *free_CA_mode* in der **SDT** mit Rückfall über **PMT**) sowie die wichtigsten *PID*-Werte (Video, Audio, *PCR*) ausgewiesen.
- Hardware-**BISS-1**- und **BISS-E**-Descrambler für den Empfang verschlüsselter Kanäle von DVB-Karten. Schlüssel werden pro Programm oder pro einzelnem *PLP* im **T2-MI**-Modus vergeben; beide Schlüsselformate werden unterstützt (12 oder 16 Hex-Zeichen mit automatischer Prüfung der **BISS-1**-Kontrollbytes).
- Multi-Stream-**T2-MI**-Unterstützung (*ETSI TS 102 773*): mehrere *T2-MI carrier* auf einem Transponder, PLP-Auswahl pro Dienst, automatische und manuelle *carrier PID*-Auswahlmodi, Filterung nach *TSID*.
- **MPEG-DASH**-Unterstützung am **HLS OTT**-Ausgang: Erzeugung eines *MPD*-Manifests im Profil *mp2t-simple* mit denselben Segmenten wie **HLS**.
- **WebVTT**-Untertitel-Unterstützung in **HLS OTT**: automatische Dekodierung von Teletext-Untertiteln, Segmentierung der Untertitelspur an den **HLS**-Segmentgrenzen und Veröffentlichung in der Playlist. Gesteuert über die Stream-Option *ott-webvtt*.
- Teletext-basierter Untertitel-Decoder (**ETSI EN 300 706**): vollständige Tabellen nationaler Alphabete, korrekte Zusammensetzung der Seitenzeilen und Auslieferung der Untertitel an den Player.
- **MPTS**-Multiplexer: automatische Erkennung des *Service Type* aus der *PMT* (HD/SD H.264, HEVC, MPEG-2, digitales Radio u. a.) mit der Möglichkeit eines manuellen Overrides über die Einstellung *Service Type*.
- **MPTS**-Multiplexer: manuelles PID-Remapping (*mpegts-pid-old* / *mpegts-pid-new*) mit Kollisionsschutz bei der automatischen PID-Auswahl benachbarter Elementarströme.
- **MPTS**-Multiplexer: Durchleitung von Dienst-Elementarströmen (*DSM-CC*, *AIT*, **SCTE-35**), die durch entsprechende Deskriptoren in der *PMT* gekennzeichnet sind — bislang wurden solche Ströme bedingungslos herausgefiltert.
- **MPTS**-Multiplexer: die Obergrenze der Gesamtbitrate wurde von 64 auf 128 Mbit/s angehoben.
- Einstellungsbereich *DVR Storage*: Anbindung von DVR-Speichern und deren Bindung an **SPTS**-Streams (Parameter *dvr-storage*) — Vorbereitung der Aufzeichnungsfunktionalität.
- Unterstützung für ASI-Geräte.
- Transkoder: Unterstützung von Streams ohne IDR-Frames.
- Transkoder: 5.1-Audio-Encoder-Profil mit Lautheitskorrektur. Lautheitskorrektur beim Transkodieren von 5.1 nach Stereo/Mono.

- Server-Cache von Perfect Streamer und externer Reverse-Proxy (nginx) für hochbelastete Systeme.
- Integration mit Prometheus, Telegraf / InfluxDB.
- Werkzeuge: *TS Analyze Perfect Streamer Toolkit v2.2 — TR 101 290*.
- Werkzeuge: *MPTS Migrate Perfect Streamer Toolkit v1.0 — MPTS-Identitätsmigration*.
- Fehlerbehebungen und weitere Verbesserungen.
- Version 1.2.0.95 der Transkoder *pstreamer-tcsw* und *pstreamer-tcnv* veröffentlicht.
- Version 1.0.0.28 des Transkoders *pstreamer-ivplv* (Intel VPL) veröffentlicht.

7.3 Version 1.11.1.420

07.04.2026

- MPTS-Muxer überarbeitet. Bitrate wird im *input muxer* gesetzt. Konformität mit **TR 101 290** und **T-STD**.
- RTSP input.

7.4 Version 1.11.1.417

31.03.2026

- SPTS Stream / MPEG-TS: Einstellung *Bitrate Mode* hinzugefügt.
- SPTS-Stream: Restamp PCR für **TR 101 290**-Konformität hinzugefügt.
- SRT: Deadlock-Fixes bei hoher Last.
- Fehlerbehebungen und weitere Verbesserungen.

7.5 Version 1.11.1.407

13.03.2026

- Transkoder: Unterstützung für Variable Frame Rate (VFR) hinzugefügt.
- Transkoder: Unterstützung für HEVC Main10 mit bt.709 (SDR) und bt.2020 (HDR) hinzugefügt.
- Transkoder: Option zur Konvertierung von SD BT.470-2 (PAL) und SMPTE 170M (NTSC) nach BT.709 hinzugefügt.
- Transkoder: Resize-Preset „Upscale SD→HD“ hinzugefügt. Wird auf SD-PAL/NTSC-Quellen angewandt; Interlace wird nicht unterstützt, ggf. vorher deinterlacen.
- Transkoder: kritischer Fehler beim Hängenbleiben des Prozesses beim Entladen des Nvidia-Encoders behoben. Er beeinträchtigte den Transkoder und erforderte einen manuellen Stream-Neustart.
- Streamer: kritischer Fehler im Videoanalysator (H.264 und HEVC) behoben, der zu anomal hoher CPU-Last führte und den Streamer blockieren konnte.

- Im TCNV-Transkoder wurde Unterstützung für Interlace/Alternate 8 Bit/10 Bit hinzugefügt.
- Bildqualität von TCNV verbessert; Post-Processing auf Nvidia CUDA überarbeitet.
- Output-Transkoder: erweiterte Statistik.
- Unterstützung für IGMP v3 SSM hinzugefügt.
- Möglichkeit, in der HLS/HTTP-URL einen benutzerdefinierten Stream-Namen statt der ID anzugeben.
- SRT input/output: Parameter AES Type.
- Komfortables Kopieren der Output-Stream-Links.
- Such-/Filterformular bei aktiven Peers.
- Fehlerbehebungen und weitere Verbesserungen.
- Version 1.2.0.86 der Transkoder *pstreamer-tcsw* und *pstreamer-tcnv* veröffentlicht.

7.6 Version 1.11.1.384

21.12.2025

- Transkoder: Unterstützung für Interlace Alternate (zwei Halbbilder getrennt im Stream) hinzugefügt.
- Deutlich geringere CPU-Last beim Empfang von SRT-Streams (*SRT input Caller mode* → *Disable TSBPD*) durch den eigenen Synchronizer von Perfect Streamer.
- Korrektur der Eingangsstream-Daten: *Fix PAR* (Korrektur des Pixel Aspect Ratio) und *Fix Framerate* (wird konfiguriert, wenn die Framerate-Daten im SPS des Streams fehlen — notwendig für die nachfolgende Transkodierung).
- Neue HLS/HTTP-Modus-Einstellung: *Auto* — Modus-Erkennung anhand *Content-Type*.
- Verbesserungen rund um Untertitel und Teletext.
- Verbesserung des UDP-Playlist-Imports.
- Fehlerbehebungen und weitere Verbesserungen.
- Version 1.0.0.70 der Transkoder *pstreamer-tcsw* und *pstreamer-tcnv* veröffentlicht.

7.7 Version 1.11.1

19.10.2025

- Unterstützung für Debian 13/Ubuntu 25 und RHEL 10/AlmaLinux 10.
- Für die Transkoder *Nvidia enc* und *Software CPU* wurde die GLIBC-Anforderung von 2.34 auf 2.28 gesenkt: Unterstützung für Debian 10 und AlmaLinux 8.
- Für H.264-Transkoder wurde die Wahl der Profile *Main* und *High* hinzugefügt.
- Neues Feature *output file* — Aufzeichnung des Streams in eine TS-Datei oder Ausgabe an ein beliebiges Gerät (auch SDI), das unter /dev erscheint.
- Neue Möglichkeit *input file* — zyklische Videowiedergabe aus einer TS-Datei.

- Verbesserung der Transkoder-Arbeit.
- Verarbeitung von Conditional Access MPEG-TS (CA): ECM und EMM hinzugefügt.
- HLS-OTT-Buffer-Entladen beim Deaktivieren des Streams behoben.
- Neues Feature *Jitter Auto sync*.
- Bessere Kompatibilität beim Empfang nicht-standardisierter HLS-URLs.
- Bessere EPG-Server-Kompatibilität mit XMLTV-Quellen.
- Weitere Verbesserungen und Fehlerbehebungen.

7.8 Version 1.10.1.364

20.08.2025

- Test-Stream-Generator — Testsignale (Testbilder).
- Funktion des Peer-Logins anonymous: Streams ohne Authentifizierung empfangen.
- Peer-Autorisierung über IP-Adressbereich.
- Peer-Option *Login is ip* — Autorisierung per IP (oder IP-Bereich) anstelle von Login.
- Verbesserungen der adaptiven HLS-Funktionalität.
- Bildqualitätsverbesserung für den Nvidia-Transkoder.
- CBR-Korrektur für H.264 beim Software-CPU-Transkoder.
- Aktualisierung der OpenSSL-Bibliothek auf Version 3.0.9.
- Das Scrollen der Stream-Tabelle in der Stream-Liste wurde überarbeitet.
- Weitere Verbesserungen und Fehlerbehebungen.
- Version 1.0.0.57 der Transkoder *pstreamer-tcsw* und *pstreamer-tcnv* veröffentlicht.

7.8.1 Hinweise zur Migration von früheren Versionen:

Aufgrund von Änderungen in den Autorisierungsmechanismen über IP und IP-Adressbereich für den Empfang auf «Flussonic»-Software müssen für im «Perfect Streamer» erstellte Peers mit IP-Autorisierung Links im Format `srt://Stream_IP:port?streamid=*` verwendet werden.

Früher wurde im Link statt * die IP-Adresse des Empfangsservers mit „Flussonic“ verwendet, z. B. `srt://Stream_IP:port?streamid=Your_IP`

Ab Version 1.10.1.364 funktioniert der Empfang eines Streams in diesem Format nicht mehr.

Mehr Details zum SRT-Empfang von „Perfect Streamer“ in „Flussonic“ siehe *FAQ*.

Aufgrund von Änderungen in den Mechanismen zur Identifikation der Grafikkarten ist eine erneute Bindung der Grafikkarten im Transkoder erforderlich. Öffnen Sie dazu die transcoder-output-Einstellungen, prüfen Sie, dass das richtige Gerät (Device ID) ausgewählt ist, und speichern Sie die Einstellungen — unabhängig davon, ob das Gerät geändert wurde oder nicht.

7.9 Version 1.10.1

30.06.2025

- Erzeugung von adaptivem HLS. Beschreibung in der Dokumentation.
- Automatische Erneuerung von Let's-Encrypt-SSL-Zertifikaten über certbot.
- Unterstützung für LCN (Logical Channel Number) hinzugefügt.
- Anzeige und Analyse von SCTE-35-Markern im Stream hinzugefügt.
- Verbesserter Software-Transkoder. Höhere Bildqualität und korrigiertes CBR für MPEG-2.
- GStreamer und Codecs sind bereits in den Paketen der Distributionen tcsw und tcnv enthalten (die Installation von GStreamer ist nicht mehr zwingend erforderlich — sie kann lediglich für die RTSP-, RTMP-Funktionalität und die Test-Stream-Tabelle nötig sein).
- Eingebauter GStreamer auf Version 1.26 aktualisiert.
- Der Nvidia-Transkoder (tcnv) arbeitet mit jeder CUDA-Version; es gibt keine harte Bindung an Version 12.5.
- Die Deinterlace-Einstellung des Nvidia-Transkoders wurde aus der allgemeinen GPU-Konfiguration in den Input jedes enkodierten Streams verlagert — analog zur Software-Methode.
- Verbesserter EPG-Server und SSL-Modi für EPG, HTTP.
- Fehlerbehebung.

7.10 Version 1.9.2.340

07.05.2025

- Unterstützung für *Video Passthrough* im Transkoder-Modus hinzugefügt. In diesem Modus wird das Video unverändert durchgereicht, nur Audioformat und -bitrate werden geändert.
- Einstellungen *NV lookahead* und *bframe* für den Nvidia-Transkoder hinzugefügt.
- Audio-Eingangsunterstützung für MPEG-1 Layer 1, 2, 3 (mp3) hinzugefügt.
- Der Abschnitt *Transkoder* im linken Seitenmenü wurde überarbeitet und detaillierter gestaltet.
- Verbesserte Stabilität und Kompatibilität des Transkoders mit verschiedenen TV-Streams.
- Verbesserungen am EPG-Server.
- Verbesserungen an HTTPS-Server, EPG SSL und HLS SSL.
- Unterstützung für HLS-Links, bei denen die Playlist auf eine Playlist mit neuer Sitzung verweist, hinzugefügt.
- Weitere Verbesserungen und Fehlerbehebungen.
- Version 0.9.6.34 der Transkoder *pstreamer-tcsw* und *pstreamer-tcnv* veröffentlicht.

7.11 Version 1.9.2

31.03.2025

- (Beta) Transkoder-Funktion auf Basis von Nvidia Encoder und Software CPU hinzugefügt. Unterstützt HEVC (H.265), H.264 und MPEG-2 in allen Auflösungen von 4K bis SD.
- Der Bereich „Systemmonitor“ wurde überarbeitet — Nvidia-GPUs werden mit gpu, memory, encoder und decoder angezeigt.
- Neuer Bereich „Transkoder“. Zeigt zusammenfassende Informationen über aktive Streams in der Transkodierung (Decoder und Encoder), Quellen, Laufzeit und Status.
- Im Bereich „Transkoder“ ist für jeden Stream in Transkodierung ein Log mit detaillierter Statusbeschreibung sowie möglichen Fehlern und deren Ursachen verfügbar.
- Bereich „DVB-Adapter“ wieder verfügbar. Empfang von TV-Kanälen über DVB-S/S2-, DVB-C-, DVB-T2-Karten; Analyse des Eingangssignals und der Streams.
- Verbesserungen am Transportprotokoll RIST.
- Anpassungen und Verbesserungen am EPG-Server.
- Verbesserter integrierter Stream-Analysator für TV-Kanäle.
- Verbesserungen und Fehlerbehebungen am Webportal.
- PID-Ersetzung bei SPTS-Streams hinzugefügt.
- Anzeige von TS ID und TS Net ID im Stream-Info-Block der MPTS-Seite hinzugefügt.
- Verbesserte PID-Behandlung bei Streams.
- Weitere Verbesserungen und Fehlerbehebungen.

7.12 Version 1.9.1

10.02.2025

- Verbesserungen und Anpassungen am Multiplexer.
- Stuffing-Modus: PCR und Realtime (System Clock) für SPTS und MPTS.
- Fehlerbehebung.

7.13 Version 1.8.1.315

02.01.2025

- Zugriffslisten für Streams bei Peers.
- Login- und Passwort-Optionen für HLS/HTTP-Input hinzugefügt.
- Bessere Kompatibilität der Login/Passwort-Authentifizierung über SRT mit Drittsoftware.
- Verbesserter Betrieb und Leistungsoptimierung.

- Fehlerbehebung.

7.14 Version 1.8.1

28.11.2024

- Leistungsverbesserung des HLS-OTT-Modus.
- Verbesserung der Benutzerfreundlichkeit.
- Verbesserung des Playlist-Exports.
- Fehlerbehebung.

7.15 Version 1.7.1.300

04.09.2024

- Leistungsverbesserung im SRT-Betrieb.
- Verbesserung der Benutzerfreundlichkeit.
- Verbesserte Kompatibilität bei der Arbeit mit HLS.
- Verbesserung der Stream-Gruppenoperationen.
- Verbesserter Kanalimport aus Playlists, Unterstützung der Transportprotokolle UDP und RTP am Ausgang bei automatischer Erzeugung von Ausgängen.
- Bitratenanzeige pro PID.
- Fehlerbehebung.

7.16 Version 1.7.1

08.02.2024

- Optimierung und Refactoring des Codes, deutlich geringere CPU-Last.
- HLS-Betriebsmodi: Peering und OTT.
- Export der TV-Kanäle in verschiedenen Transportprotokollen in eine .m3u8-Playlist.
- Import von TV-Kanälen aus einer Playlist in verschiedenen Transportprotokollen mit anschließender Konfiguration des Ausgangs der Streams im gewählten Protokoll und Portbereich.
- Stream-Klonen.
- Stream-Gruppenoperationen — Klonen und Löschen.
- Usability-Verbesserungen am Programm.
- Verschiedene Verbesserungen und Fehlerbehebungen.

7.17 Version 1.6.1

15.10.2023

- XMLTV-Import aus externen Quellen.
- XMLTV-Server.
- EIT-Generator für SPTS-Stream und Multiplexer.

7.18 Version 1.6

15.08.2023

- MPEG-TS-Multiplexer.

7.19 Version 1.5.1

18.04.2023

- Peer-Einschränkungen — Pause, Datumslimit, Limits der Session-Anzahl pro Protokoll.
- Stream-Name-Funktion und Unterstützung für kyrillische Zeichen hinzugefügt.
- Sortierung nach deaktivierten und aktivierten Kanälen.
- SRT-Bibliothek aktualisiert.
- Analysator-Funktion korrigiert.
- Weitere Verbesserungen und Korrekturen.

7.20 Version 1.5

28.12.2022

- OTT http/hls output.
- HTTPS-Unterstützung für Web- und HTTP-Server.
- Erweiterter Stream-Analysator.
- Fehlerbehebungen.

7.21 Version 1.4.3

12.09.2022

- Programmoptimierung: geringere CPU-Last.
- Bitraten-Einstellung am Stream entfernt.
- HTTP-Input wurde entfernt; dieses Protokoll wird jetzt vom HLS-Input unterstützt.
- Für HLS wurde Unterstützung für https:// und Redirects hinzugefügt.

7.22 Version 1.4.2

27.05.2022

- Unterstützung des RIST-Transportprotokolls.
- Korrektur fehlerhafter PCR-Marken (PCR Fix).
- Empfang und Senden von SRT-Streams im Listener-Modus.
- Fehlerbehebung.

7.23 Version 1.4

16.12.2021

- MPEG-TS-Analysator für CAT/ECM/EMM.
- Filteroptionen für CAT/ECM/EMM.
- Diagramm der Eingangsstream-Verluste.
- Verbesserungen am Web-Interface.
- Fehlerbehebungen.

7.24 Version 1.3

14.11.2021

- DVB-Geräte — Empfang und Analyse der Streams. Qualitätskontrolle.
- MPTS-Demultiplexing für DVB- und MPTS-Streams.
- Kontrastreiches Theme der Weboberfläche.
- Lokale Einstellungen der Weboberfläche: Theme, Zeitzone.
- Fehlerbehebungen.

7.25 Version 1.2

01.09.2021

- Arbeit mit EPG.
- XMLTV-Export.
- Fehlerbehebungen.

7.26 Version 1.1

26.08.2021

- Empfangen und Senden von MPTS-Streams. Inhaltsanalyse.
- Verschlüsselte Streams.
- Anzeige zusätzlicher MPEG-TS-Stream-Parameter — EPG, Teletext, Untertitel.
- Zusätzliche Filteroptionen für MPEG-TS-Streams — EPG, Teletext, Untertitel.

7.27 Version 1.0

11.07.2021

Erstes öffentliches Release.