Video Streaming mit freier Software

http://etherpad.servus.at/p/streamdoku

---

1 Streaming Workshop with Christian Pointner

1.1 Christian Pointner

- Funkfeuer
- mur.at
- Radio Helsinki

-> live video streaming Elevate Festival

1.2 Teilnehmer

- Peter Wagenhuber
- Ushi Reiter
- Markus Decker
- Ufuk Serbest
- Christoph Haag
- Stefan Hageneder

2 Grundlagen

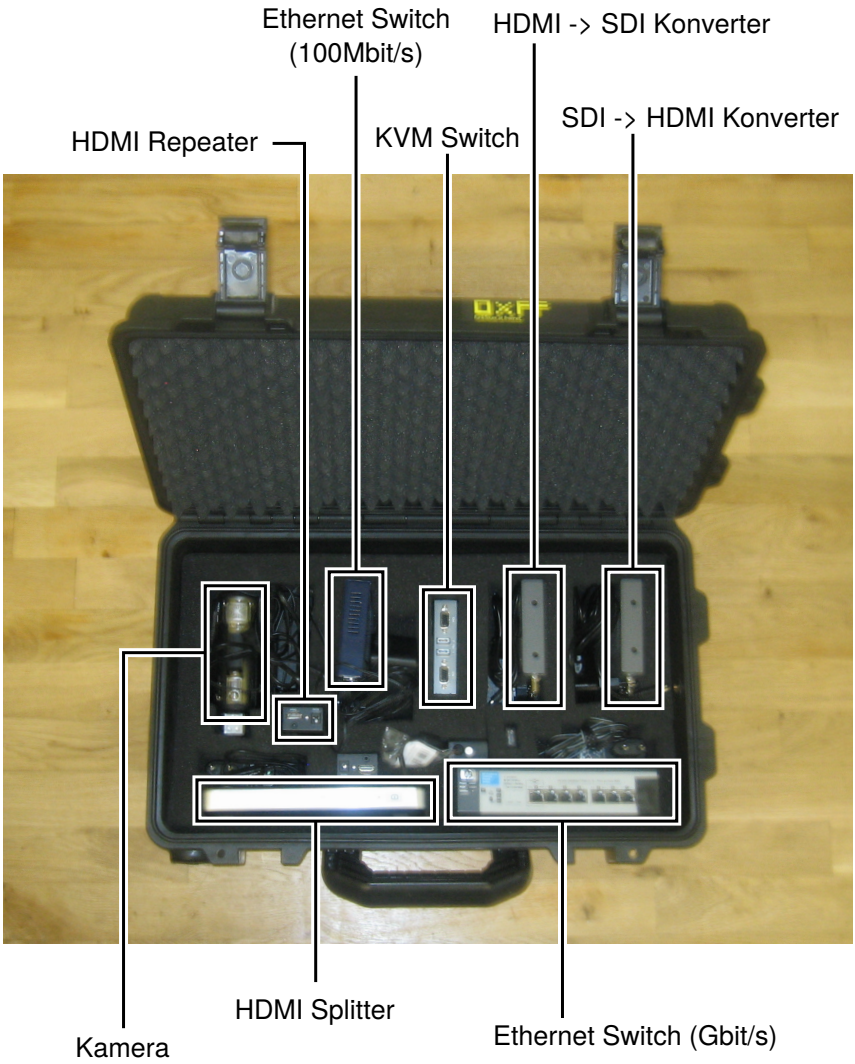
2.1 Live vs. On-Demand Streaming

On-Demand

Gewisse Header und ein Index sind beim *On-Demand Streaming*, im Gegensatz zum Live Streaming, vorhanden, weil bestimmte Informationen wie z.B. die Länge schon vorliegen. Oft wird die Downloadrate von der Serversoftware allerdings begrenzt, daher sieht es für den Client einem “echten” Livestream sehr ähnlich. Praktische Beispiele für *On-Demand Streaming*: Youtube, Vimeo, diverse Mediatheken. Achtung bei der Auswahl der Streamingsoftware: Manchmal wird *On-Demand Streaming* fälschlicherweise als “Live” Streaming bezeichnet.

Live

Da beim Livestreaming kein Index vorhanden ist und damit keine Informationen z.b. über die Länge der Datei vorliegen, macht das bei manchen Clients wie z.b. Internet Explorer Probleme. Hier ist leider immer noch ein Rückgriff auf einen Flashplayer notwendig. Das gleiche “Problem” stellt sich beim direkten Dump des Streams für Archivzwecke dar. D.h. Files ohne Information über die Länge des Files können nicht ohne weiteres von Software-Mediaplayer abgespielt werden. Es ist daher notwendig die Archive-files mit der dafür notwendigen Zeitinformation auszustatten > zb. klassische Aufnahme via Videosoftware usw. anstatt des direkten Dump. Manche Streamingformate wie zb HLS haben dieses Problem aufgrund der spezifischen Medien(File) Distributions Technik nicht, s.u.


Ethernet Switch
(100Mbit/s)
HDMI -> SDI Konverter
HDMI Repeater
KVM Switch
SDI -> HDMI Konverter
Kamera
HDMI Splitter
Ethernet Switch (Gbit/s)

2.2 Kameras und Hardware als Grundlage für Qualität

Kameras

Die Streaming-Qualität ist abhängig von Kamera und Lichtsituation. Ein Live-Stream mit einer Konsumerkamera im Innenbereich liefert meistens unbefriedigende Ergebnisse Warum: Das Videobild ist meistens körning (Noiseratio) und produziert mehr Rauschen. Das hat eine unmittelbare Auswirkung auf den Encoder und das heisst mehr CPU-Zeit verbauch und auch mehr Bandbreite da sich Rauschen nicht komprimieren lässt. Für den Innenbereich muss man diesen Fehler durch zusätzliches Licht, Licht-Regie und bessere Kameraobjektive ausgleichen. Im Aussenbereich und am Tag funktionieren Konsumerkameras meist gut bzw. ausreichend.

Kabel

- Standard für Video-Streaming: SDI (75 Ohm Koax Kabel), Übertragung über lange Wege möglich (rund max. 200m)
- Mit Vorsicht zu geniessen: HDMI Kabeln, Übertragung bis zu maximal 10m!
- Achtung Kabelführung: Beim Verlegen von Video-Kabeln ist die Nähe von Stromkabeln (Lichtverkabelung, Netzwerkkabeln, etc) zu vermeiden! Sie bereitet Probleme und kann beim Live-Stream zu Bild-Aussetzern führen und den Stream unterbrechen . Der Grund ist die meist schlechte Abschiermung von den für den Zweck verwendeten SDI (75 Ohm Koax Kabel) Kabeln. Generell ist die Sinalqualität bei langen Kabeln immer zu prüfen.

Videomischer vs. Video-Hub

Zwischen Empfangen und Verarbeiten von Video-Signalen zu einem Stream ist ein Videomischer von Vorteil. Er liefert Kontinuität, weil die gesamte Information Frame per Frame transportiert wird. Unterbrechungen des Streams können so verhindert werden, da der Videomischer auch wenn kein Frame anliegt das Signal aufrecht erhält (Schwarzbild) Bei einem Video-Hub kann es beim Umschalten zu kurzzeitigen Signalabbruch kommen.

Best Practise Hardware: Gute Erfahrungen wurden mit Hardware (Videomischer) von der Firma “Black Magic Design” gemacht. **Empfehlung**: Decklink SDI, Intensity Pro, Mini Recorder

- Vorteile: Preis, Linux-Support
- Nachteile: Proprietäre Treiber

Wichtige Tips:

- Schnittregie trennen vom Streaming. Sie ist verantworlich auch diverse Eingangs-Signale für die Postproduktion aufzuzeichnen.
- Mehr an verschiedenen Stellen - Kamera, Stream,.., zu Recorden ist nie ein Fehler.
- Bei der Planung muss man div Standards einführen. Es ist ratsam, gleiche Bild- und Audio- Standards einzuführen und auf allen Geräten einzuhalten!

2.3 Formate/Codec/Protokolle

- H264
  - proprietär (x264 Opensource Implementierung)
  - Clients: iPad, iPhone (h264 im Baseline Profil)
- VP8
  - royality free
  - Clients: Android, Firefox, Chromium/Chrome, Opera, diverse Player wie VLC, mplayer, ...
  - Bitraten-Einstellungen von VP8 sind etwas trickreich. Nie unter 50% der Bitrate von GOP für Keyframe

Keyframes (Frames die das Gesamtbild beschreiben) sind eine wichitige Komponente im Stream, die Buffergrösse wird dadurch mit definiert und je nach Bewungsintensität im Bild muss die Keyframerate angepasst werden. In etwa kann man sagen eine Keyframerate von 50 Frames (2sec bei 25FpS) sind ausreichend. (== 2sec minimal Buffer, 2 Keyframes sollten im Buffer enthalten sein dh 4 sec Bufferzeit ist ideal).
Ein gutes Streamingsetup sollte zumindest die 2 Formate VP8 und H264 aufgrund der guten Client unterstützung bieten.

funktionierende Containerformate

Containerformate kümmern sich um das Synchrone abspielen der unterschiedlichen Streams, d.h. Audio, Video, Subtext.
Unter http://caniuse.com/#feat=video gibt es eine Auflistung welche Browser welche Formate seit welcher Version unterstützen. Im Normalfall braucht man aktuell nur noch 2 Formate: H264 + AAC in MP4, VP8 + Vorbis in WebM. Alle gängigen Browser können zumindest eines dieser Formate abspielen. Bei den Live Streams siehts da leider etwas anders aus. Da Mpeg-4 zwar theoretisch Live Streams muxen kann aber es defacto keine Implementierung gibt die damit umgehen kann, müssen bei echten Live Streaming andere Formate verwendet werden, wie zb.:

- flv (http://en.wikipedia.org/wiki/Flash_Video) Flash Video ist zu vernachlaessigen und eine alte Technologie, einzig der IE benoetigt dieses Format noch, da der IE kein anderes Live Streamingformat abspielen kann. Man benötigt dafür Flash (zb.: flowplayer).
- WebM (http://en.wikipedia.org/wiki/WebM) HTML5
- HLS (http://en.wikipedia.org/wiki/HTTP_Live_Streaming) .ts files, .m3u8 (Playlist file) HTML5, fuer alle mobilen Geraete. Aufgrund der Fragmentierung in einzelne zeitlich begrentzte Packte eignet sich dieses Format sehr gut fuer die Archivierung, da in diesen Fragmenten der Timecode inkludiert ist. Bei allen anderen Streamingformaten ist das nicht der Fall. Adaptive Bitrating (http://en.wikipedia.org/wiki/Adaptive_bitrate_streaming) Nachteil: HLS schwierig zu monitoren.

-burst on connect-, ist ein Feature im Streamingserver. Die Clients beginnen das Video quasi augenblicklich abzuspielen. > server push vom Videobuffer. Streaming-Server: Icecast und flumotion unterstützen burst on connect (Einstellung am Server ca 6 Sekunden)

2.4 Archiv Formate

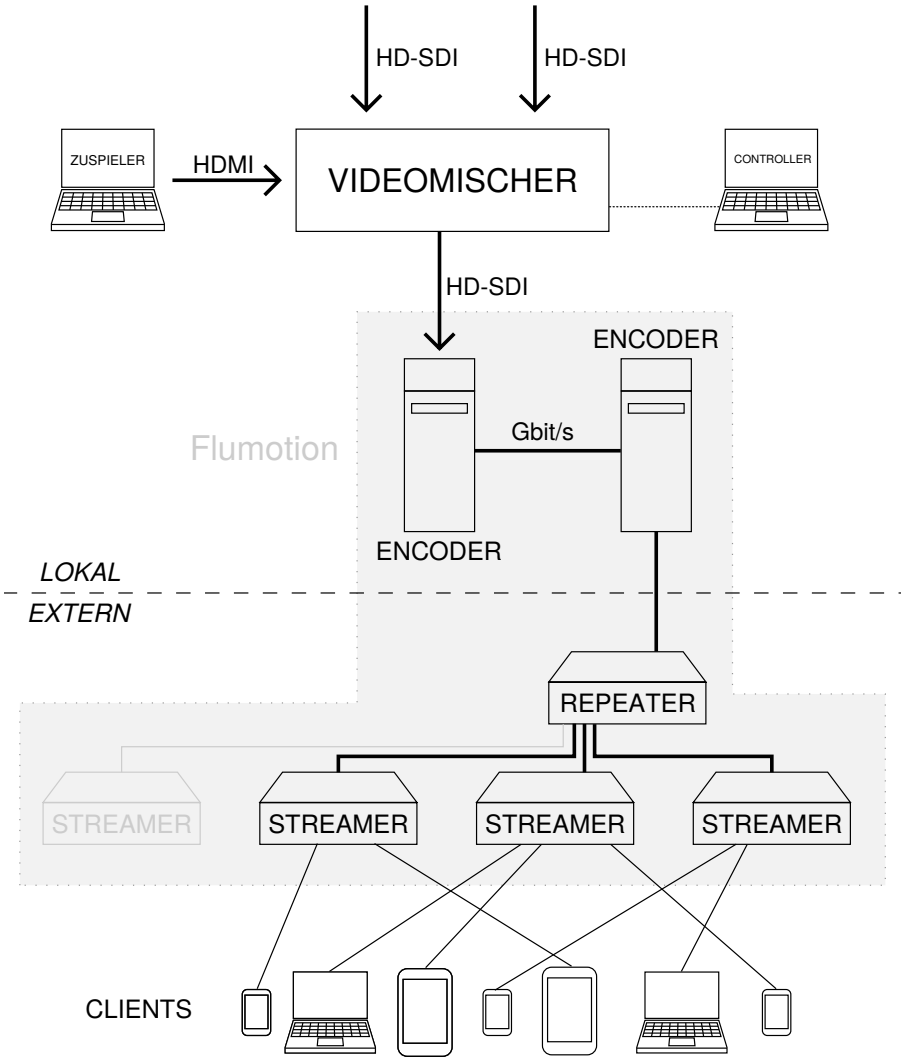
Eine grosse Abdeckung der derzeitigen Browserlandschaft findet sich mit: h264 + aac in FLV (für Flash -> IE) und mpegts (für HLS) (alternativ funktioniert mp4 auch in allen Browsern die h264 abspielen können und in Flash) vp8 + vorbis in WebM
Eine generelle Archivierung der Videofiles wird hier nicht im Detail behandelt, einzig das sich jemand darum kümmern muss, und am besten mit einer Videosoftware aufgezeichnet wird die den Archivefile mit einem Timecode versieht. Gleichzeitig ist ein Streamdump als Backup ratsam.

2.5 Netzwerk Infrastruktur

Bei Videostream Projekten kann eine grosse Bandbreite benötigt werden um die Clients zu versorgen. Eine ungefähr genaue Einschätzung der Benutzerinnenanzahl ist von Vorteil. Um grosse Bandbreiten kurzfristig und für einen kurzen Zeitraum billig nutzen zu können, bieten sich Mietserver und Leitungen diverser Serverfarmen an. Zwei Projekte können als ungefähre Aulastungs und Konfigurations Beispiele dienen: Das Elevate Festival nutz 2 Distributionsserver mit einer Bandbreite von ca. 100 MBit bei einem Userpeak von ca 100 Streams in den verschiednen Auflösungen (240p25, 360p25, 480p25, 720p25). Der CCC hingegen mit unbekannter Benutzerinnenzahl, hatte 2013 wesentlich mehr Bandbreite zur Verfügung und div. Distributionsserver extern verteilt.
Die Erfahrung hat gezeigt das die Encodierung der einzelnen Streams besser vor Ort stattfinden soll und die gemieteten Server ausschliesslich zur Distribution verwendet werden. Der Unterschied zwischen den beiden Distributions-techniken ist das man vor Ort das unkomprimierte Videosignal abnimmt und in die einzelnen Streams konvertiert, wohingegen am Server, der bereits komprimierte hochauflösende Stream rekodiert wird und sich daraus, eine schlechtere Videoqualitaet und eine hoehere Bandbreite der rekodierten Streams ergibt. Beide Artefakte will man im Normalfall vermeiden. Das Abwegen zwischen den beiden Distributionsformen, findet vor allem mit der vorhandenen Bandbreite vor Ort zum Server statt. Wählt man die zweite Variante der Transkodierung am Server muss man vor allem darauf achten eine schnelle Cpu mitzubuchen.

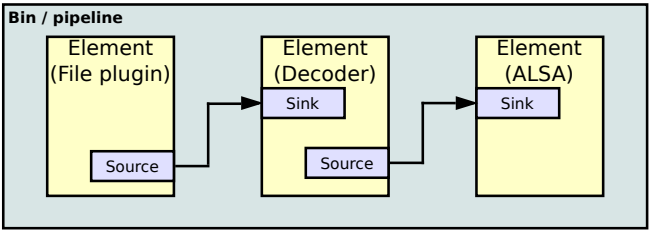
Protokolle:

- Webstreaming über HTTP (1.1)
- RT*P (Sessionmanagement von Daten getrennt) (RTSP für android < 3, RTMP für flash)
- Dash.js wird die Zukunft


HD-SDI
HD-SDI
ZUSPIELER
HDMI
VIDEOMISCHER
CONTROLLER
HD-SDI
ENCODER
Flumotion
Gbit/s
ENCODER
LOKAL
EXTERN
REPEATER
STREAMER
STREAMER
STREAMER
STREAMER
CLIENTS

3 Gstreamer

Bei Gstreamer handelt es sich um ein modernes sehr modulares Multimedia Framework, das wie ein klassiches Signalverarbeitungssystem arbeitet: Quelle -> Konverter -> Filter -> Senke. Der Flumotion Server baut auf diesem Framework auf und bei diversen Modifikationen ist es notwendig die jeweiligen Gstreamer-module manuel ein bzw umzubauen.

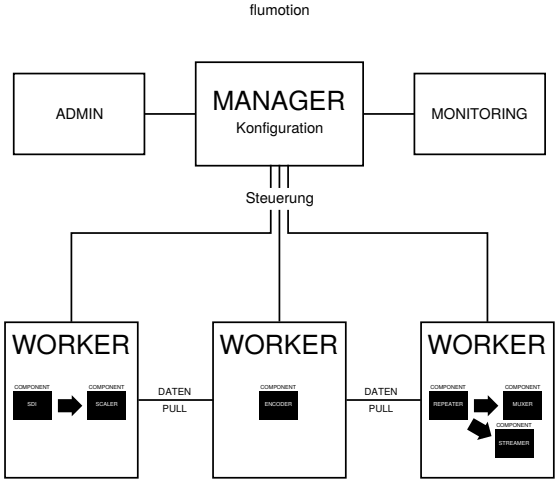


Gstreamer *bin* ist sowas ähnliches wie ein subpatch http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/section-bin-create.html

3.1 gstreamer-tools

- gst-inspect: gibt alles moegliche aus was so da ist (alle plugins)
  - gst-inspect (optionen anzeigen)
  - e.g. gst-inspect alsasrc
- gst-launch
  - ! = pipe operator
  - gst-launch alsasrc ! audioconvert ! vorbisenc ! oggmux ! filesink location=tmp.ogg
  - gst-launch audiotestsrc freq=1000 ! autoaudiosink
  - gst-launch audiotestsrc freq=1000 ! vorbisenc ! mux. videotestsrc ! vp8enc ! mux. matroskamux name=mux ! filesink location=tmp.mkv
  - gst-launch uridecodebin uri=http://live.helsink.at:8088/live160.ogg ! autoaudiosink
  - http://linux.die.net/man/1/gst-launch-0.10
  - GST_DEBUG=3 gst-launch audiotestsrc ! fakesink 2> tmp.log
  - gst-launch filesrc location=in.ogg ! decodebin2 name=decoder decoder. ! audioconvert ! lame ! muxer. decoder. ! ffmpegcolorspace ! x264enc ! muxer. matroskamux name=muxer ! filesink location=tmp.mkv
- gst-launch audiotestsrc is-live=true freq=1000 ! audiotestsrc ! audio/x-raw-int,channels=2,rate=44100,depth=16 ! vorbisenc ! queue ! mux. videotestsrc is-live=true pattern=ball ! video/x-raw-yuv,height=720,width=1280 ! vp8enc ! queue ! mux. matroskamux name=mux ! filesink location=tmp.mkv

3.2 Flumotion (http://www.flumotion.net)

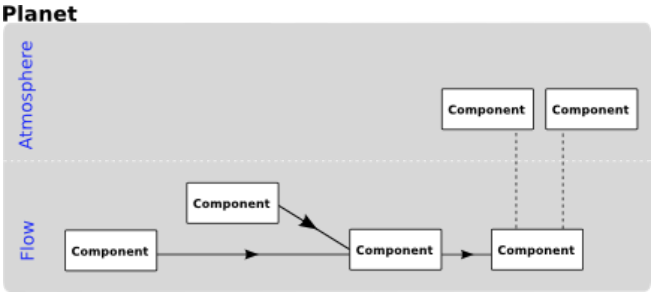


Ist ein Open-Source Multimedia Streaming Server. Die Software ist in Python geschrieben und benutzt die Python Bindings von Gstreamer, darum ist dieser ähnlich modular aufgebaut. Unterschiedliche Aufgaben wie z.B. das Encodieren von Video werden von verschiedenen Komponenten erledigt und können auch auf mehrere Rechner verteilt werden. Damit ist es auch möglich unterschiedliche Formate und Auflösungen als Live-Stream anzubieten.

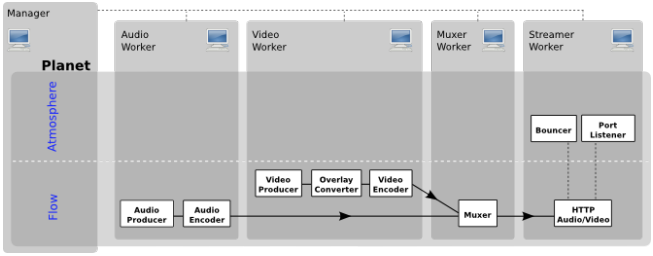
Die Konfiguration zerfällt in die Konfiguration des *Managers* und des/der *Worker*.

3.3 Manager

Verteilt und Monitored die Konfiguration und steuert die Worker. Die Konfiguration erfolgt im planet.xml File im Verzeichnis /etc/flumotion/managers. Das planet.xml besteht aus 2 Hauptteilen der *atmosphere* und dem *flow*, die beide *components* enthalten.



atmosphere: Beherbergt Komponenten die nicht direkt an der Signalerzeugung oder Verarbeitung beteiligt sind. (Wer darf mit Wem Wie kommunizieren) flow: Beschreibt die Komponenten und den Datenfluss dazwischen: Wesentliche Arten von Komponenten: * Producer: Streamquellen (SDI, Soundkarte, . . .) * Converter: Bearbeiten Stream Daten. (konvertieren, muxen,. . .) * Consumer: Sind lediglich Stream “Konsumenten”. (stellen z.B. einen Webstream zur Verfügung)



3.4 Worker

Sind die eigentlichen “Arbeitstiere”. Die einzelnen Worker kommunizieren über TCP-Sockets. Verschiedene Komponenten können auf unterschiedliche Worker verteilt werden. (z.B. Encodieren und Multiplexen von unterschiedlichen Streams auf verschiedenen Rechnern)
Manager und alle Worker sollten eine public IP haben. Aber, Vorsicht mit beim Routing, NAT, und der Server kann kein IPv6!

3.5 Komponenten

Der Datenfluss durch die Komponenten ist so organisiert, dass eine Komponente einen sog. feed zur Verfügung stellt von dem sich nachfolgende Komponenten die Daten abholen. (eater) Nachfolgende Komponente “holt” sich die Daten. Wenn mal ein “feeder” ausfällt dann wird die nachfolgenden Komponente die von diesem “feeder” isst (“eater”) “hungrig” (Status in der flumotion Administrations-GUI)
Die Flumotion Administrations-GUI wird mit dem Befehl flumotion-admin gestartet und bietet Zugriff auf den Manager und die Worker. Die GUI ist in erster Linie zum monitoren der Komponenten nützlich. Es gibt auch die Möglichkeit über einen Wizard

eine neue Konfiguration zu erstellen, diese bietet aber nur sehr eingeschränkte Funktionalität. Als Startpunkt zum ersten Experimentieren ist dies aber trotzdem hilfreich. Für Komplexere Setups führt kein Weg am Editieren (oder mittels selbsgeschriebener Tools generieren) der XML Konfigurationsdateien vorbei. Ein hilfreiches Tool dabei ist flumotion-inspect. flumotion-inspect listet alle vorhandenen Komponenten auf wenn es ohne weitere Parameter aufgerufen wird. Wenn beim Aufruf von flumotion-inspect der Name einer Komponente als Parameter übergeben wird liefert das Tool nähere Informationen zu der jeweiligen Komponente. (ähnlich wie gst-inspect - siehe oben)
Um SDI Karten als Input (Producer) zu verwenden gibts ein flumotion decklink plugin: http://cs.brown.edu/~jsb/flumotion/.

3.6 flufigut

flufigut ist ein Python Script das die Erstellung, der doch sehr umfangreichen und schlecht zu editierenden planet.xml des Flumotion Servers, anhand von kleinen sogenannten .json config Files erleichtert. flufigut ist eine Entwicklung von Christian Pointner, das Repository ist hier http://git.spreadspace.org/?p=flufigut.git;a=summary zu finden.
Im obigen Repository sind auch zusätzliche Flumotion Komponenten (im Verzeichnis contrib/flumotion-components) zu finden. Diese Dateien müssen in das Verzeichnis /usr/lib/flumotion/python/flumotion/component/ kopiert werden. Damit und dem pipeline-conveter ist es nun auch möglich mit der Open Source Version von flumotion Flash Streams mit h264 und AAC zu erstellen und den flv Stream per rtmp and einen RTMP Server (z.b. Nginx rtmp-module) zu schicken.
Eine Beispielkonfiguration ist hier zu finden:
http://git.spreadspace.org/?p=flufigut.git;a=tree;f=contrib/flumotion-sample-conf

3.7 alternative Streamingserver/Encoder

Für den nginx Webserver http://nginx.org/, existiert ein https://github.com/arut/nginx-rtmp-module nginx rtmp Streaming Modul. Dieses Setup unterstützt auch IPv6. Der Nachteil bei diesem Server ist die Reduzierung der Streamingformate auf MPEG-4. Unterschützte Container und Protokolle: HLS, RTMP, Mpeg Dash
RTMPD http://www.rtmpd.com/
Anderer Ansatz: Als Encoder dazu bietet sich der OpenBroadcastEncoder an http://www.obe.tv/. SDI-IN -> OB Encoder -> Multicast (dieser Multicast kann mit ffmpeg dann an beliebige Streamingserver verteilt werden) Nachteil: Retranscoding Vorteil: unterbrechungsfreier Output Stream.
Mögliche alternative für CG overlay (zu BM Videomischer) html5 editor für MLT Framework. https://github.com/inaes-tic/mbc-etiquette

3.8 -load balancing- an den Servern:

Mit roundrobin-dns kann man die Anfragen auf die Verschiedenen Server verteilen, das reicht in den meisten Fällen aus. Bei HLS Streaming ist da allerdings Vorsicht geboten da man die laufend generierten Video-File-Fragmente auch zeitgleich auf die Rechner verteilen muss. Das kann man mit verschiedenen Filesystemen lösen, zb glusterfs. Siehe auch Flumotion Repeater weiter oben.

Unterstützt von: