Document Number	**EDCS-821575**
Based on Template	**EDCS-160995 Rev 18**
Created By	**Cisco Systems, Inc.**

VQE-C
System Integration Guide
Release 3.6

This document provides a guide for integrators of the VQE-C (Visual Quality Experience Client).

Modification History

Revision	Date	Originator	Comments
1.0	8/17/2012	Cisco Systems, Inc.	1. Initial version.

Contents

List of Tables

List of Figures

1 Introduction

The VQE Client is the client software of the VQE Solution. This VQE-C client software is shipped as an Open Source BSD licensed Software development kit (SDK). This Document describes the VQE-C software SDK and is intended as a guide to assist 3rd parties in integrating VQE services to their product. This guile contains an overview of the VQE-C software architecture, a description of the SDK components, a section on how to build VQE, Step-by-step integration advice and some information on troubleshooting VQE integrations.

Please refer to following websites for the latest information regarding features and releases:

- Cisco CDA Visual Quality Experience Application User Guide
  http://www.cisco.com/en/US/products/ps7127/products_user_guide_list.html

- Release Notes for Cisco CDA Visual Quality Experience Application
  http://www.cisco.com/en/US/products/ps7127/prod_release_notes_list.html

For information on obtaining documentation, obtaining support, providing documentation feedback, security guidelines, and also recommended aliases and general Cisco documents, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

2 Overview

The VQE-C library provides a set of high-level APIs designed to support easy integration into an existing Set-Top Box (STB) software base. The high-level interface is designed such that there will be minimal, if any changes to the existing threading or socket usage model of the STB application.

The VQE-C library is comprised of following three fundamental components:

VQE-C ifclient API – Maintained programmatic interface to the end user for integration.

VQE-C Control Plane (CP) – System configuration and management

VQE-C Data Plane (DP) – Data path component. This component may be integrated into either *user* or *kernel space* on the target platform.

The VQE-C library also provides capabilities to be easily configures and optimize for given target platform via the use of system configuration parameters and compile time options.

3 Architecture

The diagram below shows the major components of the VQE-C and the associated integration points:

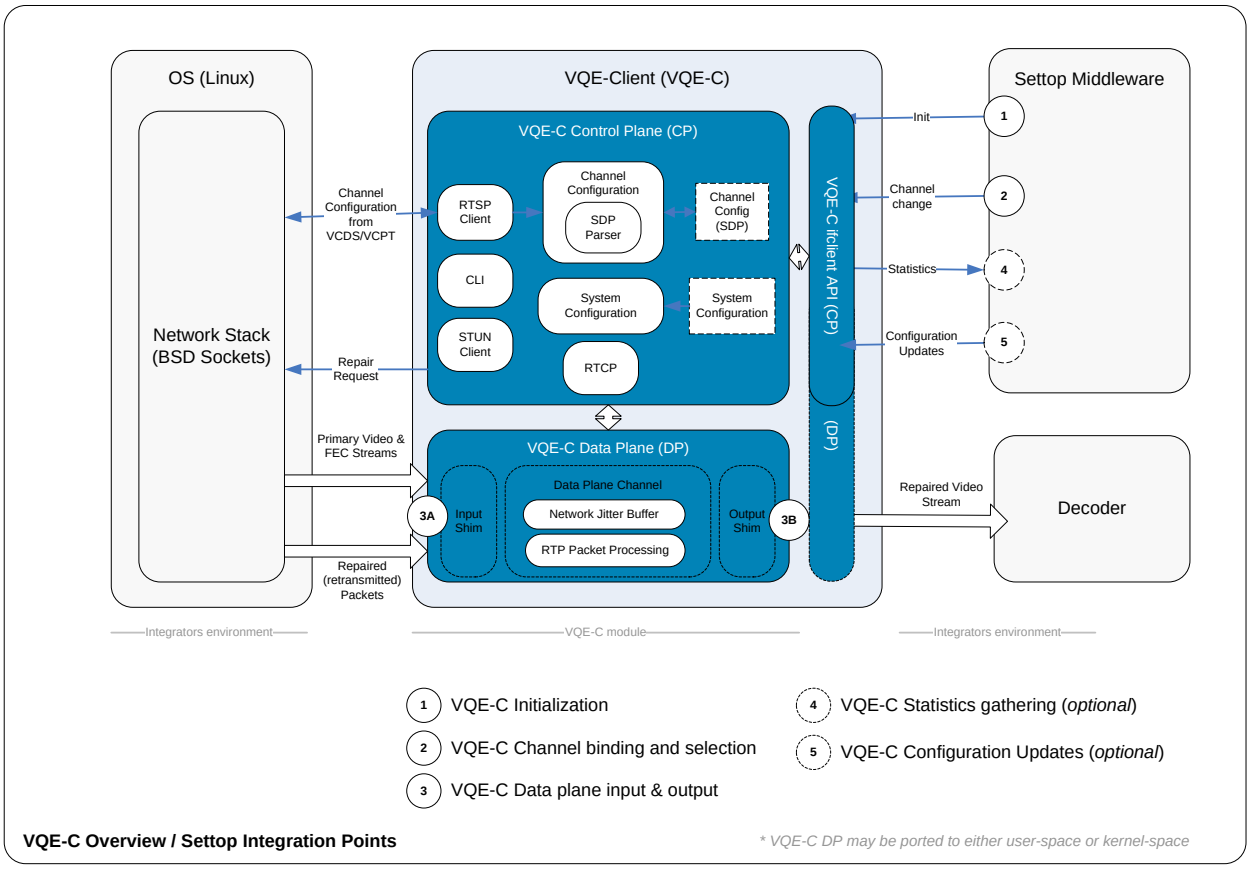


Figure 1 VQE-C Overview / Integration Points

The VQE-C library interfaces are primarily designed to create VQE-C tuner instances, to dynamically bind a tuner instance to a RTP data-stream which has been synthesized for error-repair, and to read the inbound data-stream.

A VQE-C tuner defines a “pipe” between an input “RTP stream” or “channel” and an output sink, which could be a STB decoder thread, or any other client. The VQE-C tuner module is an entity that conceptually “tunes” to a given channel, and provides an output stream of MPEG data. The tuner and its subcomponents provide the client functionality which implements error repair. A VQE-C tuner has an input and output, as well as state information regarding the channel. There is a one-to-one mapping between VQE-C tuners and STB VQE-C tuner clients.

The **vqec_ifclient.h** presents the user level interface to VQE-C. A socket-based interface is provided to replace socket calls for receiving packets in an application integrating the video

services of VQE-C. The primary reason for the tuner client's existence is to permit an application that is written in terms of blocking socket calls to integrate VQE-C without moving to a non-blocking architecture. The tuner client APIs solve this problem in that they allow the caller to block (or not block). VQE-C itself runs multiple tuners as a single, event driven, asynchronous thread. An application cannot block the main event loop of VQE-C without breaking VQE-C.

VQE-C is an event-driven component. One instance of VQE-C can receive events for multiple input video streams simultaneously. All external events (i.e. control packet reception, application timers, etc.) are serialized through the event manager. The event manager role is performed by the 3rd party open source component libevent.

In a typical integration of VQE-C into an STB, the STB code spawns a thread for base VQE-C processing and then uses a separate thread (STB data decode thread) context for replacing existing socket calls with VQE-C's socket replacement API. Note that VQE-C provides APIs for calling from the STB's thread context, but it does not actually perform the thread creation/deletion operations. The tuner client interface (socket replacement API) runs in the execution context of the existing STB application.

VQE-C's memory footprint has a fixed overhead portion (code size + fixed data size) and a portion related to the required input buffering. Input buffering is required for both, de-jittering of the video, error repair and RCC (please refer to section 0). VQE-C's memory footprint is highly dependent on the amount of input buffering that is required.

3.1 VQE –C ifclient API

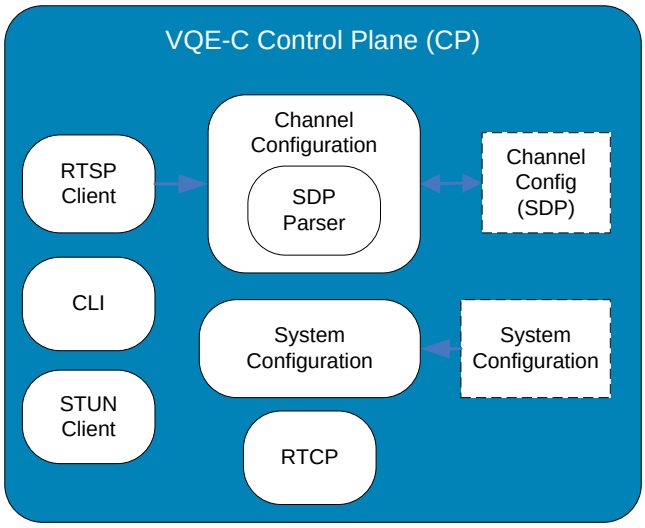
VQE-C provides a programmatic interface to the end user for integration. The programmatic interface provides a "socket replacement" interface, which is used to get packets from a repaired, VQE-Server accelerated video stream.

VQE-C also provides APIs for updating its channel and system configuration data and acquiring statistics on error repairs and channel changes.

Most of the VQE-C ifclient APIs interact with the Control Plane (CP) component, with one expectation being **vqec_ifclient_tuner_recvmsg()** that directly interfaces to the Data Plane (DP) "Output Shim".

3.2 VQE –C Control Plane (CP)

The control plane is responsible for the overall configuration and management of the VQE-C, and contains several major components that are briefly described below.



3.2.1 System Configuration

The system configuration manager reads the system configuration file passed in during the call to the VQE-C initialization function (**vqec_ifclient_init**). The parameters set in this configuration file govern the overall configuration of the VQE-C, including the maximum number of video streams support, source of channel configuration and signaling modes for example. Changes to the system configuration file do not take place until the application integrating VQE-C is initialized again. During VQE-C initialization, VQE-C will also merge configuration changes from a Network Configuration and/or Override Configuration file to the VQE-C running configuration, if such configuration files are present. Please refer to the VQE-C System Configuration Guide (ref 2) for details on configuration updates and the meaning of specific configuration parameters.

3.2.2 Channel Configuration

The configuration manager is a component that communicates with the centralized configuration server (with CDI enabled) to receive updated versions of the 'channel lineup' (set of SDP channel descriptions), parse the SDP and provide access to the configuration data (in internal form) to the rest of VQE-C. Channel configuration updates may also be received via middlewares which support their own configuration delivery mechanism. If no channel configuration updates are received (e.g. a configuration server cannot be reached) VQE-C may use channel configuration that is locally cached via the **channel_lineup** parameter of the System Configuration file.

The configuration manager maintains a 'channel database', an internal representation of the channel lineup, which can be efficiently indexed by address and port.

3.2.3 RTSP Client

The RTSP client is used to obtain updated network configurations and channel configurations from a VQE-C Configuration Delivery Server (VCDS). The DESCRIBE method in the RTSP protocol is used by the client to retrieve updates to these configurations. Refer to section 9.1 for a more detailed description.

3.2.4 STUN Client

STUN (Simple Traversal of UDP through NATs), defined by RFC 3489, is used to open and maintain bindings on a NAT device positioned between the VQE-S and VQE-C. It allows the client application to learn the bindings, mapped ports and public address in a typical residential NAT's case, so that they can be communicated to the server. As of release 3.2, UPnP (Universal Plug and Play) is offered as an alternative to STUN for NAT traversal.

3.2.5 RTCP Client

The RTCP Client provides the Real Time Control Protocol services required to obtain a video repair packet. If a VQE-C notices a missing packet or a group of missing packets, which it detects by a gap in the sequence number in the RTP headers, it sends an RTCP Feedback Generic NACK to the RTCP Feedback Target (unicast). The VQE-C sends the NACK at the appropriate time based on the size of its jitter buffer, the time it will take to receive the repair once the NACK is sent, and a timed offset to minimize the chances of creating a NACK storm if all Receivers perceive the packet as lost. Note that the longer the wait, the more likely it is that the Receiver will be able to batch together multiple packet losses into a single NACK.

On receipt of the RTCP Feedback Generic NACK, the VQE-S looks into its cache, and if it has any of the requested packets, it sends them immediately to the Receiver as RTP retransmission repair packets (also unicast)

3.2.6 Signaling

VQE-C supports two signaling modes: standard mode, which is used when VQE-C is not behind a NAT device; and a NAT mode, which is used when one or more NAT devices are between VQE-C and the VQE-S.

In VQE-C's standard signaling mode a channel's repair ports are configured. The repair port for a given channel is specified by the SDP associated with the channel. The repair port of a given channel for the VQE-S and VQE-C are identical. The repair port for the VQE-C and the VQE-S is specified by the SAME line of SDP configuration.

In NAT mode the repair RTP and RTCP ports in the channel's SDP description are ignored. When VQE-C is behind a NAT device VQE-C learns the repair port bindings of the NAT device using the STUN client.

3.2.7 Command Line Interface

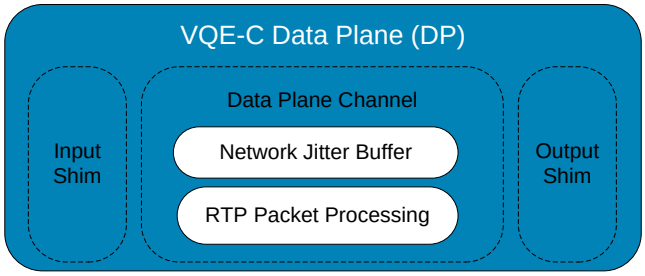
VQE-C provides a telnet based CLI for development, debug and diagnostic purposes. This CLI may also be used internally for scripting purposes. The CLI is intended for diagnostics/testing during troubleshooting or integration and is not intended to be an integral component of the STB running the VQE-C.

Note: The default setting for access to the VQE-C CLI is from the local interface (i.e. loopback). Hence in a typical integration login to the VQE-C CLI would be performed by first logging into the local prompt and then issuing the following (assuming the VQE-C telnet port is set to 8182):

```
telnet 127.0.0.1 8182
```

3.3 VQE –C Data Plane (DP)

The fundamental job of VQE-C is stitching together the primary (multicast) RTP video stream with the repair RTP video stream and delivering the repaired video stream to STB's decoder. The video streams in VQE-C are received encapsulated in RTP (over UDP). The data plane in essence consists of a Network Jitter Buffer and Packet Processing module.



3.3.1 Network Jitter Buffer (PCM)

VQE-C provides de-jittering for RTP streams only if the VQE-C can determine that the incoming stream is RTP from the passed URL, VQE-C can provide de-jittering services for any RTP data stream.

3.3.2 RTP Packet Processing

VQE-C maintains a packet pool for input buffering. VQE-C uses a statically allocated fixed block pool. All packets, control and data, are assumed to be less than the standard Ethernet MTU(i.e. 1508 bytes). There is currently a single buffer pool shared among all tuners.

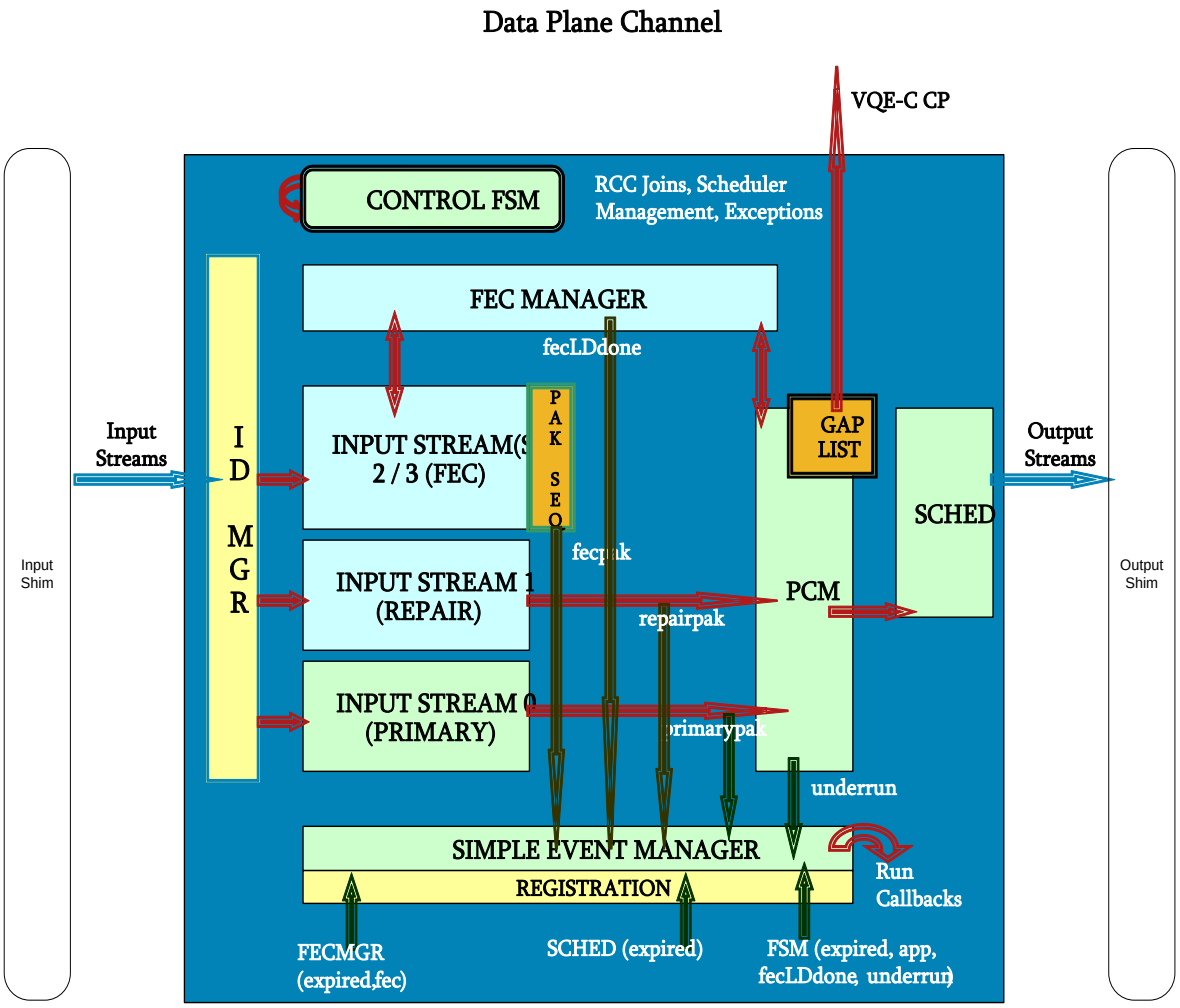
The data packets received are stored and indexed by sequence number. The received packets from the repair stream and primary stream are cached and supplied to the STB decoder client via the VQE-C's socket replacement API. The sequence numbers are within a restricted range, but modified such that a large GAP between repair and primary streams can be tolerated.

The Forward Error Correction (FEC) feature, when enabled, attempts to correct any corrupted RTP packets from primary stream prior to request being made for a repair packet.

A policing function is also provided for RTP repair requests, when this feature is enabled, those error repair requests which exceed the configured traffic profile for a given stream will not be sent by the VQE-C. The policing function is disabled by default.

3.3.3 Data Plane Channel

A dataplane channel provides services to process, reorder & merge the primary, repair and FEC streams and to generate a single repaired stream. In that sense, this object forms the core of the functions provided by the VQEC dataplane.



A dataplane channel encapsulates the following sub-entities:

- RTP input receivers for the processing of primary, repair, FEC column and FEC row streams. The packets will be received through the "input stream" interface from the input shim.
- Packet Cache Manager (PCM). The PCM provides a central cache for reordering and buffering all packets that have been received from the primary & repair streams, and for packets that may have been corrected via FEC.
- FEC Manager. The FEC manager evaluates packet loss on the primary stream, and recovers any packets correctable through available FEC streams.

- Output Scheduler: The output scheduler sends packets to tuners on the output shim, through an "output stream" interface. It schedules packets using RTP timestamps.
- Control FSM: This state machine controls the startup sequence for the channel, and also controls the behavior of RCC.
- Event Manager: The event manager distributes internally generated signal-events to all of the aforementioned sub-entities.

3.3.4 Scheduling and Packet Processing

On initialization the VQE-C DP module creates a main timer event thread, using the **kernel_thread** API (see **/vqecutils/kmod/vqec_event.kmod.c**), this thread is set to wakeup every 20ms. All work to be done by the VQE-C DP module is initiated and completed within the context of this single thread.

The current VQE-C release does not support processing of packets within a hardware interrupt context. Please ensure that all calls to VQE-C library are made outside of an hardware interrupt context.

3.3.5 Input Shim

The input shim receives and delivers network packets for a channel's primary video, repair and FEC streams to the corresponding channel input receiver in dataplane. We expect that some integrators will create their own custom input shim using the existing APIs.

3.3.6 Output Shim

The output shim consists of queues (one per tuner) that act as the receiver or sink of output packets that are sourced from the output scheduler. We expect that some integrators will create their own custom input shim using the existing APIs.

4 Quick Start: Getting Started

As a first goal, it is recommended to build the **test_vqec_ifclient** sample application and get it to run on the target platform. The VQE-C can be used to demonstrate communication with the VQE-S and stream the video out to a decoder/player. Please refer to section 6 for guidance on integrating the VQE-C into a STB.

The following steps will provide guidance in configuring and building, first, the third party libraries used by VQE-C, and then VQE-C itself, along with its test application.

NOTE: Throughout this document, the directory referred to as "<root>" is simply the root directory of the VQE-C SDK, into which the SDK tarball was extracted, and contains the top-level Makefile and README.

Step 1. Configuring & Building VQE-C

First, configure the toolchain and/or its options and flags for the VQE-C component, by running the configure script:

1. After extracting the SDK .tgz file, go to the directory in which it was extracted.
   $ cd <root>
2. Run autoconf and configure.
   $ autoconf
   $ ./configure

This configure script should be run with the appropriate options. See the included file **configure.mips** for an example of how to run configure with appropriate options for a MIPS toolchain. The following command-line is an example of how to point **configure** to a MIPS toolchain in the /opt/VZ/bin/ directory for cross-compilation:

$ ./configure CC=/opt/VZ/bin/mipsel-linux-gcc CPP=/opt/VZ/bin/mipsel-linux-cpp CXX=/opt/VZ/bin/mipsel-linux-c++ AR=/opt/VZ/bin/mipsel-linux-ar LD=/opt/VZ/bin/mipsel-linux-ld --host=mipsel-linux

To build the VQE-C libraries and its test application:

1. From <root>, build VQE-C and its required components.
   $ make

After the build is complete, the VQE-C library will be located at **<root>/lib/vqec.a** and the VQE-C test application can be found at:

<root>/vqec_obj/eva/sample/dev/test_vqec_ifclient

Step 2. Run Test Application and Set Up an Output Stream

To use the VQE-C test application, there will need to be a VQE-S set up and running that is accessible by VQE-C, and there must be a corresponding and up-to-date channel configuration file. This channel configuration file is generated using the VCPT tool, which is accessed by going to the web address **http://<hostname-of-vqes>/vcpt**. Also, a sample channel

configuration file has been included in the VQE-C SDK for reference at **<root>/eva/sample/sample-chan-config.cfg**.

First, the **test_vqec_ifclient** binary and system configuration files need to be copied over to the target platform. Then, when the application is executed, it will scroll some startup messages, and then indicate what telnet port it is listening for CLI telnet connections on. It should be noted that, by default, the CLI telnet port is set to "0" in the configuration, which means that the CLI is disabled.

A sample system configuration file has also been included in the VQE-C SDK for reference at **<root>/eva/sample/sample-vqec-config.conf**. This example configuration file will automatically create one tuner (an instance of the entity managed by VQE-C that receives video stream data packets from one address, performs error repair on this stream, and then sends the repaired video stream packets to one destination, may it be through an API or over a network) with the name "tuner1" that is receiving packets from the stream at rtp://224.1.1.1:50000, and sending the repaired video stream out to the address udp://192.168.1.100:1234 on the network interface "eth1".

Access the VQE-C CLI via telnet via the command (from a system with network access to the host system running the VQE-C test application):

$ telnet *ip-address-of-host-system-running-vqec vqec-cli-port*

Once the telnet session is connected, the following welcome message should appear:

```
Welcome to the VQE-C CLI!

vqec>
```

To verify that VQE-C is receiving the streams and communicating with the VQE-S, at the prompt, type **show counters tuner1**, assuming that the name of the tuner created is "tuner1". Several counters with non-zero values should appear:

```
vqec> show counters tuner1
tuner-name:      tuner1
 inputs:         4942
 drops:          0
 primary:        4942
 repair:         0
 rtcp:           0
 rtp:            4942
 outputs:        0
 output Q drops: 0
```

Now, change to the configure mode by typing **enable**, then **configure terminal.** Assuming that there is a host that can play a UDP video stream with the address 192.168.1.200 and listening on port 1234, and that the target platform running the VQE-C test application has an "eth0" interface with access to the 192.168.1.0 network, use the command **stream-output tuner1 eth0**

udp://192.168.1.200:1234. This will accept incoming RTP video data, and begin streaming UDP video output to that device, which should then be able to be viewed.

One example of a setup to test this would be to use the **stream-output** command to send video to the loopback address at some port (i.e. udp://127.0.0.1:2345), and then configure the set-top box with a static channel lineup to receive and play the video stream.

Another example of this, without the involvement of set-top boxes, would be to have VQE-C running on a host, and have it stream the output to another (or could be the same, using the loopback address) host that is running VLC media player (http://www.videolan.org/vlc), and then have VLC play the video stream on port 1234. The repaired video stream should appear in VLC.

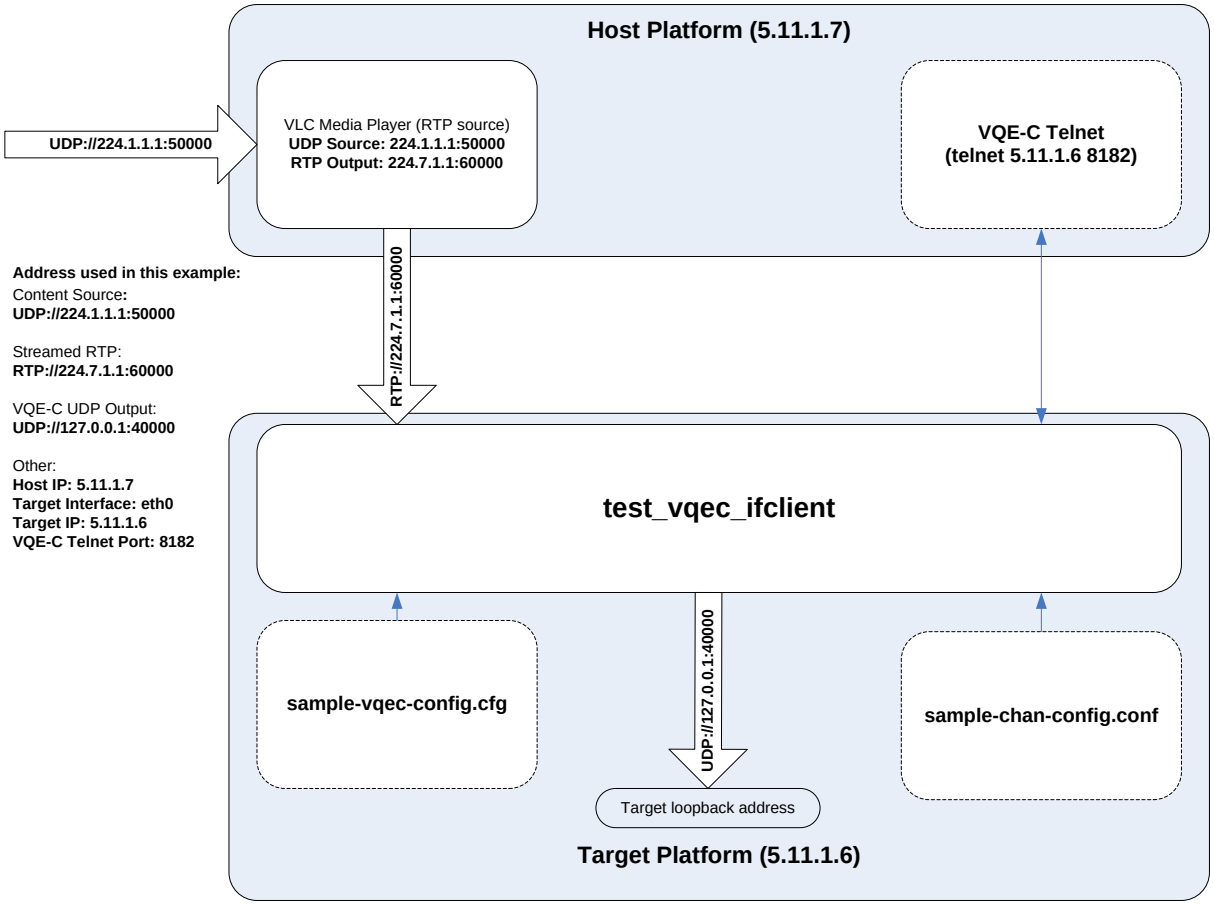
5 Testing without VQE-S

This section describes how some basic testing and verification of the VQE-C can be performed without the use of VQE-S. This test configuration does not support testing of the VQE-C error repair, however does validate the initialization, basic configuration and primary stream reception.

The following equipment and level of integration is assumed:

1. Access to the VLC media player (www.videolan.org) or similar application that capable of streaming RTP. This version (0.8.6c) used in these examples was for the Windows platform and can be obtained from http://www.videolan.org/vlc/download-windows.html
2. Some form of content or video stream that can be played by VLC media player, a UDP stream was used in the following example.
3. Successful compilation, linking and execution of the VQE-C test application **test_vqec_ifclient** on the target platform, see section 4 for details.
4. Access to the following VQE-C system configuration file: **sample-vqec-config.conf**
5. Access to the following VQE-C channel configuration file: **sample-vqec-config.cfg**

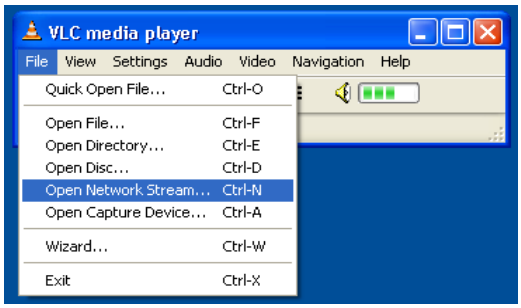
The diagram below shows the test setup:



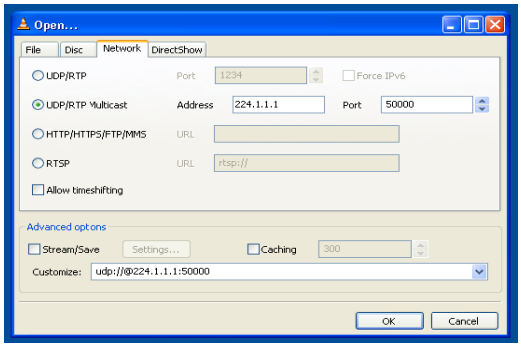
5.1 VLC (RTP Source setup)

This section describes how to setup the VLC Media Player to accept a UDP stream, convert it into RTP and stream it out onto a particular multicast output.

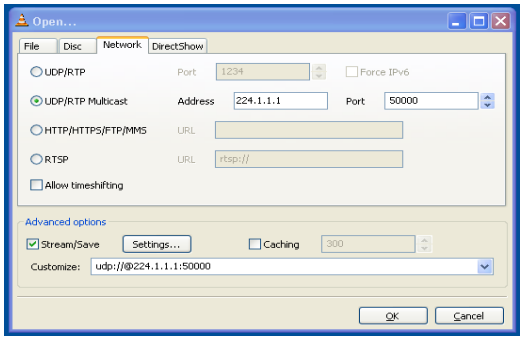
1. Select “Open Network Stream”



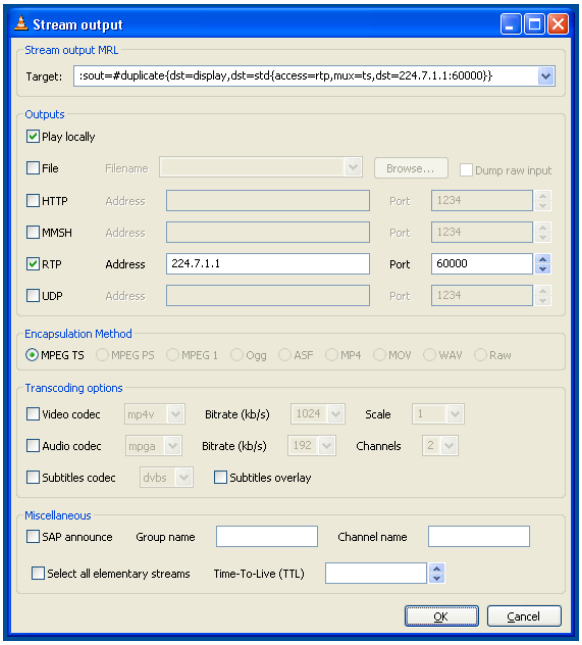
2. Select “Network” (or other depending on the source of your content, i.e. file or disc) tab, and configure the source for **UDP/RTP Multicast** with address **224.1.1.1** and port **50000**.



3. Select “Stream/Save” within the Advanced options and select “Settings.”



4. Now configure the output stream to be **RTP Multicast** with address **224.7.1.1** on port **60000**. Also select “Play locally” for stream playback. Select “OK.”



5. On return to the Network tab select “OK” and VLC should now begin streaming out the source as an RTP multicast on the specified address.

5.2 Modification of sample configuration files

The sample configuration files need some slight modification for the particular target and lab environment, primarily related to the addresses and ports used for receiving and streaming data. This section describes the key parameters that should be modified to support the example outlined in the beginning of section 5**. Note additional changes will be required to test with a VQE-S.**

Below describes the modifications needed to the **sample-vqec-config.conf** file:

Original (sample-vqec-config.conf)

```
/*
 * Copyright (c) 2007 by Cisco Systems, Inc.
 * All rights reserved.
 */
vqec_version = "1.0.0";
channel_lineup = "sample-chan-config.cfg";
sig_mode = "NAT";
input_ifname = "eth1";
max_tuners = 3;
libcli_telnet_port = 0;
tuner_list = (
{
 name = "tuner1";
 url = "rtp://224.1.1.1:50000";
 stream_output_if = "eth1";
 stream_output_url ="udp://192.168.1.100:1234";
}
);
```

Modified (sample-vqec-config.conf)

```
/*
 * Copyright (c) 2007 by Cisco Systems, Inc.
 * All rights reserved.
 */
vqec_version = "1.0.0";
channel_lineup = "sample-chan-config.cfg";
sig_mode = "STD";
input_ifname = "eth0";
max_tuners = 3;
libcli_telnet_port = 8182;
tuner_list = (
{
 name = "tuner1";
 url = "rtp://224.7.1.1:50000";
 stream_output_if = "eth0";
 stream_output_url ="udp://127.0.0.1:40000";
}
);
```

Steps for modification of sample-vqec-config.conf:

1. The signaling mode **sig_mode** should be set to **STD**.
2. The input interface, **input_ifname** may need to be modified accordingly for the target*.
3. To enable the VQE-C CLI, an unused port number needs to be specified for **libcli_telnet_port**.
4. On startup of the test_vqec_ifclient sample application a tuner named "**tuner1**" will be created and will be bound to an RTP source stream specified by the **url** parameter.
5. The output interface, **stream_output_if** should be modified for the target loopback address.

* Most target platforms will use the same interface (usually named "eth0") for input and output.

Below describes the modifications needed to the **sample-chan-config.cfg** file for Channel 1:

Original (sample-chan-config.cfg)	**Modified** (sample-chan-config.cfg)	
# Channel 1 configuration	# Channel 1 configuration	
#	#	
v=0	v=0	
o=- 1179863130 1179863129 IN IP4 titian	o=- 1179863130 1179863129 IN IP4 titian	
s=Channel 230.151.1.1	s=Channel **224.7.1.1.1**	
i=Shrek	i= **TestStream**	
t=0 0	t=0 0	
a=rtcp-unicast:rsi	a=rtcp-unicast:rsi	
a=group:FID 1 3	a=group:FID 1 3	Session Level
m=video 10000 RTP/AVPF 33	m=video **60000** RTP/AVPF 33	Media Levels
i=Original source stream	i=Original source stream	
c=IN IP4 230.151.1.1/255	c=IN IP4 **224.7.1.1**/255	
b=AS:4000	b=AS:4000	
b=RS:50000	b=RS:50000	
b=RR:150000	b=RR:150000	
a=fmtp:33 rtcp-per-rcvr-bw=40	a=fmtp:33 rtcp-per-rcvr-bw=40	
a=recvonly	a=recvonly	
a=source-filter:incl IN IP4 230.151.1.1 5.8.37.2	a=source-filter:incl IN IP4 **224.7.1.1 5.11.1.7**	
a=rtpmap:33 MP2T/90000	a=rtpmap:33 MP2T/90000	
a=rtcp: 10001	a=rtcp: 10001	
a=rtcp-fb:33 nack	a=rtcp-fb:33 nack	
a=mid:1	a=mid:1	
m=video 10000 RTP/AVPF 33	m=video 10000 RTP/AVPF 33	
i=Re-sourced stream	i=Re-sourced stream	
c=IN IP4 230.36.1.1/255	c=IN IP4 230.36.1.1/255	
a=inactive	a=inactive	
a=source-filter: incl IN IP4 230.36.1.1 8.36.1.1	a=source-filter: incl IN IP4 230.36.1.1 8.36.1.1	
a=rtpmap:33 MP2T/90000	a=rtpmap:33 MP2T/90000	
a=rtcp:10001 8.36.1.1	a=rtcp:10001 8.36.1.1	
a=rtcp-fb:33 nack	a=rtcp-fb:33 nack	
a=mid:2	a=mid:2	
m=video 10002 RTP/AVPF 99	m=video 10002 RTP/AVPF 99	
i=Unicast retransmission stream	i=Unicast retransmission stream	
c=IN IP4 8.36.1.1	c=IN IP4 8.36.1.1	
b=RS:40	b=RS:40	
b=RR:40	b=RR:40	
a=recvonly	a=recvonly	
a=rtpmap:99 rtx/90000	a=rtpmap:99 rtx/90000	
a=rtcp:10003	a=rtcp:10003	
a=fmtp:99 apt=33	a=fmtp:99 apt=33	
a=fmtp:99 rtx-time=3000	a=fmtp:99 rtx-time=3000	
a=mid:3	a=mid:3	

Description of modification to **sample-chan-config.cfg**

1. [Session Level]
   Update the session name **`s=Channel 224.7.1.1.1`**

   (s=<session name>)
   A text string name for this session (VQE channel). The length of <session name> must be less than 80 characters. This could be used by customers as a human-readable name for this channel.

2. [Session Level]
   Optionally update the session description **`i= TestStream`**

   (i=<session description>)
   A text string description for this session (VQE channel). The length of <session description> must be less than 80 characters. While this field is optional, omitting it is discouraged.

3. [Media Level]
   There are three Media levels contained in the configuration file, however for this test modification of only the *Original source stream* is required.

   a. Modify the port number **`m=video 60000 RTP/AVPF 33`**

   (m=<media> <port> <proto> <payload-type>)
   Each media description starts with an "m=" field, and is terminated by the next "m=" field or by the end of the session.

   b. Modify the multicast address **`c=IN IP4 224.7.1.1/255`**

   (c=IN IP4 <multicast address>)
   This line identifies the multicast address of the Original Multicast Stream.

   c. Modify the source filter addresses **`a=source-filter:incl IN IP4 224.7.1.1 5.11.1.2`**

   (a=source-filter incl IN IP4 <mcast dest addr> <unicast source addr>)

   VQE network elements will expect this line to be present (mandatory) and should contain:

   - The IP multicast address of this stream (the Original Multicast Stream)
   - The IP source address of this stream

   Note: The source address MUST match the source address of the video source that is in use. In this example it must match the source address applied to the rtp packets as they exit VLC. Note failure to modify this line will result in all primary stream packets being dropped. Note that VQE-C is performing SSM (Single Source Multicast) filtering for these multicast packets.

5.3 Verification

This section describes execution and verification of the test:

1. Execute the test example of the target, ensuring that the configuration file is specified:
   ./test_vqec_ifclient sample-vqec-config.conf

   Typical Output:

```
./test_vqec_ifclient sample-vqec-config.conf
Configuration file is sample-vqec-config.conf
=== pthread (1024) policy (realtime), prio (1) ===
VQE-C CLI Attempting to Listen 8182
VQE-C CLI Listening on port 8182
looking for stream-outputs in config sample-vqec-config.conf..
prot = 1, addr = 10107e0, port = 16540 output_if = 5.11.1.6
=== pthread (4101) policy (realtime), prio (1) ===
```

2. Telnet into the VQE-C CLI and monitor the counters as shown below.

```
> telnet 127.0.0.1 8182

Escape character is '^]'.
Welcome to the VQE-C CLI!

vqec> show counters tuner1
tuner-name:      tuner1
 inputs:         24995
 drops:          0
 primary:        24994
 repair:         0
 rtcp:           0
 rtp:            24994
 outputs:        0
 output Q drops: 0

vqec> show counters tuner1
tuner-name:      tuner1
 inputs:         31618
 drops:          0
 primary:        31617
 repair:         0
 rtcp:           0
 rtp:            31617
 outputs:        0
 output Q drops: 0
```

Note: Counters for inputs, primary, and rtp should all be incrementing if the configuration is correct and the VQE-C is processing the RTP stream from the VLC.

3. The VQE-C output should also be streaming to the target loopback address at port 40000.

6 Software Development Kit (SDK)

The VQE-C SDK includes the source and documentation for VQE-C and its dependent components and libraries. A sample application and the following documentation are also included:

VQE-C System Integration Guide
This document.

VQE-C System Configuration Reference
An introduction and resource for configuring VQE-C.

VQE-C CLI Command Reference
An introduction to the VQE-C CLI and reference for its capabilities.

In addition, documentation created using Doxygen can be found in both HTML and LaTeX formats. All documentation is included in the directory **<root>/eva/docs**.

The root directory of the SDK contains the following subdirectories:
3rd-party-src – contains the source code for the 3rd-party libraries used by VQE-C
add-ons – contains the compiled 3rd-party libraries and header files
eva – contains the main source and documentation for the VQE-C library
include – contains header files that are shared by more than one component
vqec_lib – contains libraries required to build VQE-C, and the VQE-C library itself
vqec_obj – contains the build environment and is the working object directory for builds
cfg, **rtp**, **rtspclient**, **sdp**, **stun, stunclient**, **utils, vqecutils** – contain source for components on which VQE-C depends

The sample application and associated configuration files can be found under **eva/sample**.

6.1 Kernel Port Platform dependent files

As is common with porting software modules to different versions of OS, some modifications may be necessary. The VQE-C release has been structured to minimize the number of places within the code that may be affected by different versions of the operating system. In general the changes required to port the VQE-C release to a different version of operating system version are isolated in the following directories:

/include/utils/kmod
/utils/kmod
/utils/log/kmod
/vqec_obj/kmod
/vqecutils/kmod

/eva/vqec-dp/kmod
/eva/vqec-dp/ouputshim/kmod

The general convention of "kmod" has been applied to both directories and files where it has been necessary to make modifications to the base code for the VQE-C DP to operate in *kernel space*.

6.2 Configure Options

This section will describe some of the options that are available when configuring and building the VQE-C package.

The configure script that comes with VQE-C supports all the usual configure script options, plus some others. A brief explanation of the various options can be seen by typing **configure –help**. Several of these options are described in more detail below. An example of how these are used can be seen in the included file **<root>/configure.mips**.

--host *host-architecture*
Use this option when cross-compiling to define the architecture of the target host.

--prefix *install-directory*
Use this option to specify the path that architecture-specific built VQE-C libraries and header files should be copied by the **vqec_install** build target (discussed later).

--disable-openssl
Use this option if the target system does not have the OpenSSL package installed.

--disable-syslog
Use this option if the target system does not have Syslog installed on it.

--disable-stun
Use this option if you do not wish to include STUN support.

--enable-upnp
Use this option to include UPnP as a STUN alternative. If both STUN and UPnP are included, VQE-C will attempt to use the most suitable method. UPnP alone is not a recommended configuration. Note that UPnP *requires* a default route to be configured in the /proc/net/route file. The Linux "route" command allows such modifications.

--enable-smallvqecstack
Use this option to limit VQE-C thread stack sizes to 32KB instead of the default of 64KB.

--enable-elog
This option is not supported in the current release.

--disable-fcc
Use this option if you do not wish to include RCC support.

--disable-hag
Use this option if you do not wish to include HAG (proxy igmp) support.

--enable-iplm
This option is not supported in the current release.

6.3 General Build Options

This section will describe some of the build targets that are available for building and using the VQE-C package. All build targets should be executed from the top-level **<root>** directory.

all
Build all third-party packages, then the VQE-C library and sample application. This is the default build target. This will produce all core VQE-C and third-party libraries in .a format, as well as .so shared-object libraries for each third-party component. There will also be an SO-linked sample application produced from this target, which uses the .so versions of the third-party libraries.

pic-vqec
Build all third-party packages, then the VQE-C library and sample application. This will produce a single .so VQE-C library (which will encompass all of the core VQE-C libraries) and binary that are in PIC (position independent code) format. The library will be named **libvqec_r.xx.xx.xx.xx.so**, where *xx.xx.xx.xx* represents the version and build number.

clean
Clean all third-party packages, then the VQE-C package, removing all compiled files and libraries.

install
Copy the VQE-C header files and compiled libraries to the architecture-specific install path, as specified by the **–prefix** option from the configuration step earlier. Please refer to the release notes to obtain a complete list of header files and libraries that copied.

6.4 Building the DP Kernel Module

This section explains a typical build procedure for the VQE-C DP module targeted for *kernel space*, the target platform for this example is a MIPS based processor.

6.4.1 Step 1.

The VQE-C CP should first be built; this can be done in the normal manner (this example builds from a MIPs based processor):

Under VQE-C release top-level directory:

```
./configure.mips -t /opt/VZ/smp86xx_toolchain/build_mipsel/staging_dir/bin
./make clean
./make
```

6.4.2 Step 2.

To build the VQE-C DP module for kernel space you will need access to the kernel source applicable to the target platform. Build the kernel for the applicable target, the next step will require access to the kernel source `<KERNEL_SRC_DIR>`.

6.4.3 Step 3.

Now build the VQE-C DP module for kernel space using the following command line:

Under VQE-C release top-level directory:

```
make kmod-vqec KERNELSOURCE=<KERNEL_SRC_DIR> ARCH=mips-linux
```

The VQE-C DP kernel can now be found in **vqec_obj/kmod/<ARCH>/vqec_dp.ko**. This module can be installed in the target system, typically using insmod.

7 Integrating

This section will describe some basics that should be kept in mind to expedite the process of integrating VQE-C with set-top box software.

First, it is important to know that a tuner in VQE-C can be associated with only one input video stream, and only one output thread. If multiple video streams need to be received simultaneously, then there will need to be a separate tuner for each incoming stream.

In general, all VQE-C threads should be run with a higher than normal scheduling priority for best performance, i.e. SCHED_RR or SCHED_FIFO. In addition, VQE-C's libevent must be run in its own thread, separate from all other applications on the system. This is because libevent will block and never exit under normal operating conditions.

For full information on the functions available in the VQE-C API, please see **vqec_ifclient.h** and the Doxygen-created documentation.

7.1 Five Essential Function Calls

There are five APIs that are essential to integrate VQE-C with set-top box software. This section will provide example code and briefly discuss these five APIs, as well as briefly introduce some of the other nonessential APIs provided by VQE-C.

The five essential APIs are as follows:
vqec_ifclient_init() – initializes VQE-C and processes configuration.
vqec_ifclient_start() – starts main VQE-C event loop. This call does not return. The VQE-C event loop runs in the thread context of this call.
vqec_ifclient_tuner_create() – creates a tuner in an unbound state.
vqec_ifclient_tuner_bind_chan() – binds (or "tunes") a tuner so that it starts receiving packets from the specified stream.
vqec_ifclient_tuner_recvmsg() – used to receive a repaired video stream from VQE-C corresponding to a given bound channel.

Four of the five essential APIs are demonstrated in the following example code:

```
/*
 * Initializes VQE-C application, creates one tuner, and then starts
 * the VQE-C event loop.  Upon success, this function will not return.
 *
 * Side effect:  caller must initialize global variable stb_vqec_tuner1_id
 *               to VQEC_TUNERID_INVALID prior to calling this function.
 *               Upon successful creation of a tuner, stb_vqec_tuner_id
 *               will contain a valid tuner ID that is ready for use.
 */
int stb_vqec_init_and_run (void)
{
    vqec_error_t err;

    /* initialize the ifclient library */
    err = vqec_ifclient_init("vqec.cfg");
    if (err != VQEC_OK) {
        printf("Failed to initialize VQE-C - error (%s)\n",
               vqec_err2str(err));
        return -1;
    }

    /* create the tuners (assume just "tuner1") */
    err = vqec_ifclient_tuner_create(&stb_vqec_tuner1_id, "tuner1");
    if (err != VQEC_OK) {
        vqec_ifclient_deinit();
        printf("failed to create tuner! (%s)", vqec_err2str(err));
        return -2;
    }

    /*
     * this function should block indefinitely; it will only return if the
     * client wasn't properly initialized or if an explicit stop is done
     */
    vqec_ifclient_start();

    return 0;
}

/*
 * Binds a (sample) stream to the VQE-C tuner, if VQE-C was successfully
 * initialized and a tuner created.
 */
int stb_vqec_bind_tuner1 (void)
{
    vqec_error_t err;
    vqec_sdp_handle_t sdp_handle;

    if (stb_vqec_tuner1_id == VQEC_TUNERID_INVALID) {
         /* VQE-C not yet initialized or initialization failed */
         return -1;
    }

    sdp_handle = vqec_ifclient_alloc_sdp_handle_from_url(
        "rtp://224.1.1.1:50000");
    if (sdp_handle == NULL) {
        printf("failed to acquire cfg handle from url");
        return -2;
    }

    /* bind tuner1 to the channel rtp://224.1.1.1:50000 */
    err = vqec_ifclient_tuner_bind_chan(stb_vqec_tuner1_id, sdp_handle, NULL);
```

```
    vqec_ifclient_free_sdp_handle(sdp_handle);
    if (err != VQEC_OK) {
        printf("failed to bind channel (%s)\n", vqec_err2str(err));
        return -3;
    }


    return 0;
}
```

Table 1 Code Example

The first function, **stb_vqec_init_and_run()**, utilizes the APIs necessary for initialization of the VQE-C library (**vqec_ifclient_init()**), creation of tuners (**vqec_ifclient_tuner_create()**), and execution of the VQE-C event loop (**vqec_ifclient_start()**) via its calling thread.

Some important information about the **vqec_ifclient_init()** APIs is as follows:

- If the **vqec_ifclient_register_cname()** API is used to customize the CNAME assigned to VQE-C, this API must be called prior to calling **vqec_ifclient_init()**.
- Prior to calling **vqec_ifclient_init()**, the network services of the host system should be initialized, since VQE-C may need to retrieve the MAC address of an interface and request updated configuration files from a server on the network.
- The **vqec_ifclient_init()** API may block while performing network-based updates of VQE-C system or channel configuration.
- If VQE-C configuration dictates that VQE-C services are globally disabled, then VQEC_ERR_DISABLED will be returned and no resources are allocated.

Also note that the **vqec_ifclient_start()** API should block indefinitely under normal circumstances, as it starts the VQE-C event loop.

The second function, **stb_vqec_bind_tuner1()**, utilizes the APIs necessary to bind (or "tune") the created tuner so that it starts receiving packets from a specific stream (**vqec_ifclient_tuner_bind_chan()**). If **stb_vqec_tuner1_id** does not contain a valid tuner ID (either because VQE-C initialization failed, did not yet complete, or the tuner could not be created) then the bind will not succeed.

The fifth API is the **vqec_ifclient_tuner_recvmsg()**, which is used to receive a VQE-C repaired RTP datagram stream, or a raw UDP stream, corresponding to a channel. RTP headers may be stripped, based on the global configuration settings.

7.2 Configuration Updates

VQE-C supports updates to both its Channel Configuration and System Configuration. Integrators must determine whether the STB will receive configuration updates via the Configuration Delivery Infrastructure (i.e. supplied by a VCDS) or via an external configuration delivery mechanism.

7.2.1 CDI-based Updates

If the Configuration Delivery Infrastructure will be used, then the **cdi_enable** configuration parameter must be set to TRUE within the System Configuration file supplied to VQE-C upon initialization. The **network_cfg_pathname** must be assigned if VQE-C is to receive updates to its system configuration from a VCDS, and the **channel_lineup** parameter must be assigned if VQE-C is to persist (to the filesystem) any channel configurations it may receive. (Leaving **channel_lineup** unset will cause VQE-C to store the channel lineup in memory only.) For details on how configuration updates are retrieved via CDI, refer to section 9.1.

7.2.2 Middleware-based Updates

If the STB has an existing configuration delivery mechanism that can be used to deliver VQE-C configuration files, then integrators may supply Channel and System Configuration updates via the following API, after initialization:

```
vqec_ifclient_config_update(
  vqec_ifclient_config_update_params_t *params)
```

Updated channel lineups must be expressed in the form of a block of SDP descriptions of VQE accelerated channels. Note that channel lineups must include the entire new VQE-C channel lineup to be used—the API does not merge any channel descriptions, but rather replaces the channel descriptions with those passed within the **params** structure. Note that tuners already bound to a channel will be unaffected until the *next* time the STB tunes to an updated channel.

System configuration updates are achieved by invoking this API with a Network Configuration file. Network Configuration files contain VQE-C configuration parameter assignments which augment/supersede settings found in the System Configuration file supplied to VQE-C upon initialization. The syntax of a Network Configuration file is identical to that of a System Configuration file, though not all parameters are supported within a Network Configuration file. See the *VQE-C Configuration Guide* for more information about which parameters are supported in a Network Configuration file.

Note that VQE-C stores updated Network Configuration files separately from the System Configuration file, and merges the two after each configuration update. Updates to parameters typically take effect for a channel the next time it is tuned by the STB, as described in the *VQE-C Configuration Guide*.

VQE-C also supports an additional configuration file, called an Override Configuration, for deployments in which per-STB configuration settings are required but not feasible via a Network Configuration file (e.g. if Network Configuration files are sent via multicast). Override Configuration updates are persisted by VQE-C independently of the System Configuration and Network Configuration, and supersede all other forms of configuration. The parameters which

are accepted within a Override Configuration update are identified in the *VQE-C Configuration Guide*. The API which may be used to supply an Override Configuration is **vqec_ifclient_config_override_update()**.

7.2.3 Detecting and Recovering from Corrupted Configuration Files

To ensure the integrity of configuration files used by VQE-C after they have been saved to the filesystem, integrators may configure the **index_cfg_pathname** parameter. When configured, VQE-C will compute and store the MD5 checksums of configuration updates as they arrive in an index file at the configured location. Whenever such configuration files are read from the filesystem, VQE-C recomputes the file's MD5 checksum and verifies that it matches the corresponding MD5 checksum as stored in the index file prior to using the configuration file.

If an MD5 checksum verification fails, VQE-C issues a log message and will not use the configuration file. The configuration delivery mechanism may then supply the current configuration file:

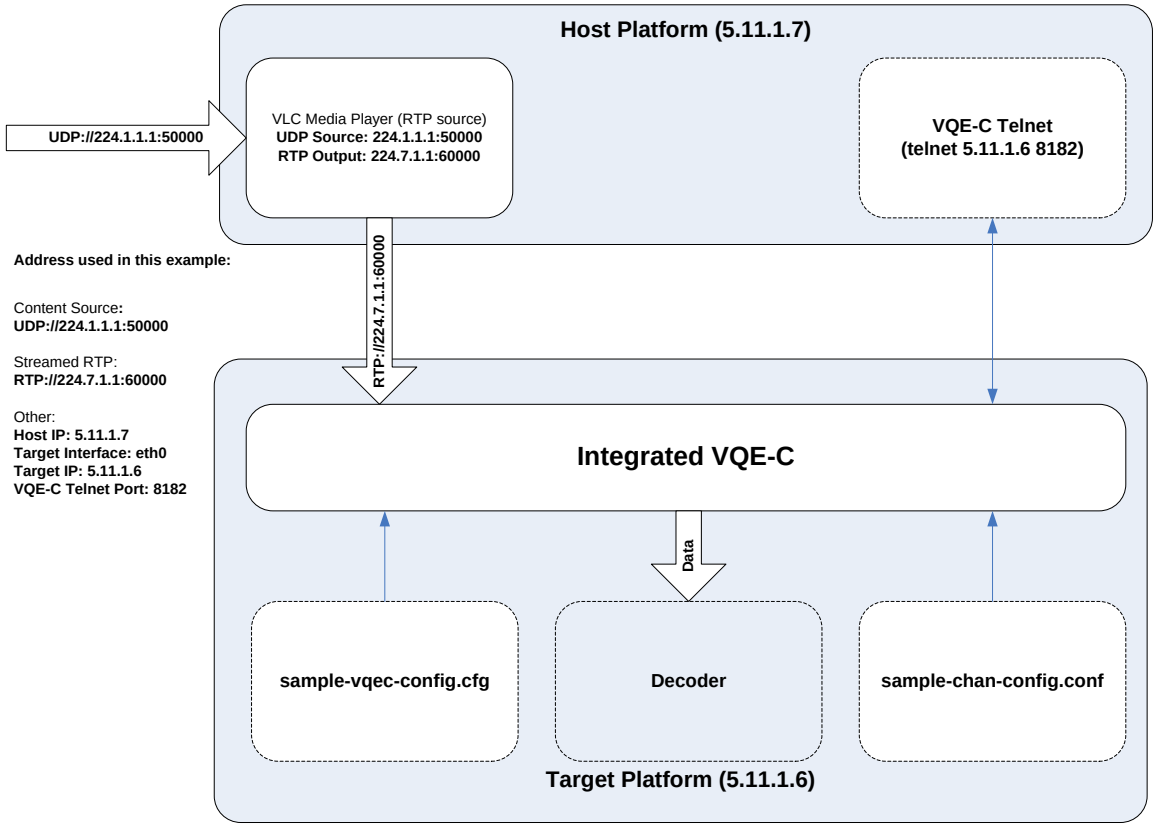
- if CDI is in use, detection of a corrupted configuration file triggers a new copy to be retrieved from the VCDS and saved to the filesystem
- if middleware-based configuration updates are in use, the middleware may register a callback with the **vqec_ifclient_config_register()** API to receive notifications of such events so that a new copy of the configuration can be supplied.

7.3 Nonessential Functions

In addition to the five essential function calls detailed in the previous section, the VQE-C API contains other functions for getting statistics, requesting updates, assigning a CNAME, and completing other tasks as well. More information on these can be found in **<root>/eva/vqec_ifclient.h**.

7.4 Verification of Integration

The VQE-C integration may be verified in similar manner as was described in section 5 for the test application (test_ifclient_vqec). Obviously the expected result is that video would be presented on the final STB output. Again this level of verification does not require a VQE-S, however error repairs will not be performed without a VQE-S present.



It is likely that the tuner creation and configuration (i.e. binding) will have been performed by the STB software (calling the appropriate VQE-C APIs), hence the **sample-vqec-config.conf** file should be modified to remove any unnecessary tuner creation and configuration. A typical configuration file may look like this:

```
vqec_version = "1.0.0";
channel_lineup = "sample-chan-config.cfg";
sig_mode = "STD";
input_ifname = "eth0";
max_tuners = 3;
libcli_telnet_port = 8182;
```

7.5 RTP Fallback

The VQE-C library also provides an API to strip off RTP headers to handle the case when a RTP stream is not present in the VQE-C channel lineup, i.e., not described via SDP in the lineup. This provides:

a. The ability to retrieve the RTP payload for streams that are not handled by VQE-C, since the stream is not present in the VQE-C channel lineup.

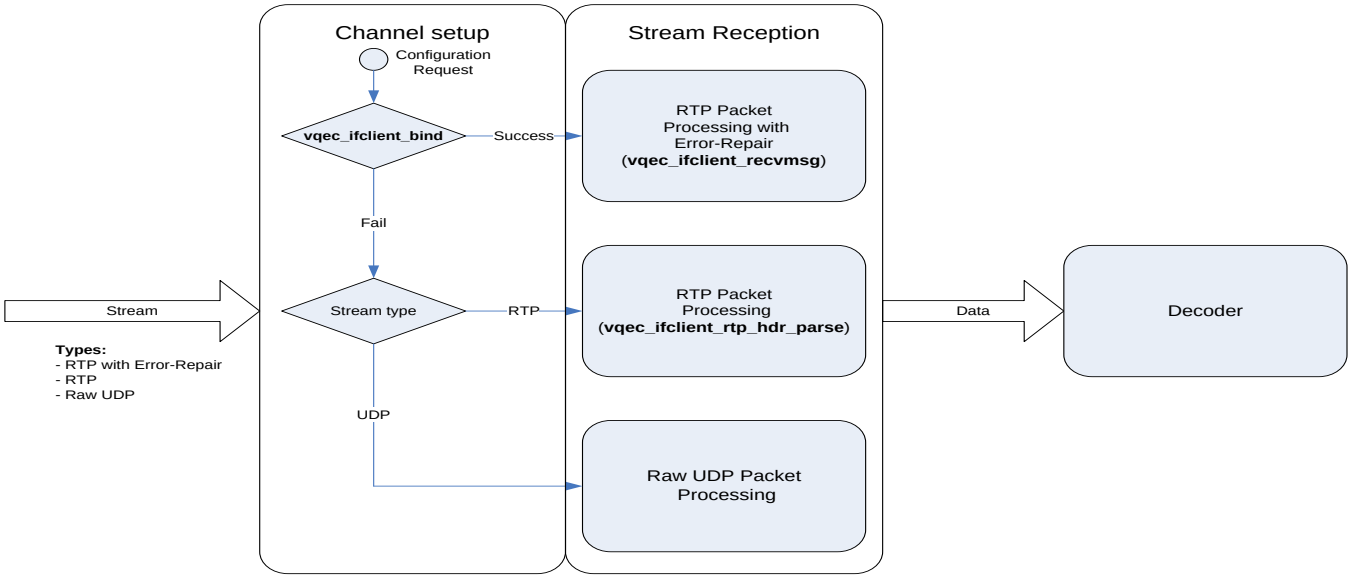
b. A level of fallback in the event of a channel lineup configuration error.

The VQE-C provides a single API that can be used to parse a RTP packet and determine the payload location within it. Note it is *not* necessary to initialize the VQEC library (i.e. call vqec_ifclient_init) to use this API.

.

```
/*-----------------------------------------------------------------------------
 * @param[in]   buf        Pointer to the buffer containing the RTP packet
 * @param[in]   len        Length of the input RTP packet buffer
 * @param[out]  rtp_hdr    On return, contains the RTP header fields of the
 *                         given and validated RTP packet.
 * @param[out]  vqec_err_t Returns VQEC_OK on success.
 *-----------------------------------------------------------------------------*/
VQEC_PUBLIC
vqec_error_t vqec_ifclient_rtp_hdr_parse (char *buf,
                                          uint16_t len,
                                          vqec_rtphdr_t *rtp_hdr);
```

A typical channel setup would consist of a first attempting a **vqec_ifclient_bind** (i.e. RTP with error repair support), if this fails and the stream is determined to be RTP then **vqec_ifclient_rtp_hdr_parse** could be used obtain the location within the packet of the payload. The payload could then be extracted and processed as if it was a raw UDP packet.



7.6 RTP Fallback Verification

Verification of the RTP fallback can be **performed in similar manner as described in section 7.4**, with one exception:

The RTP stream under test should NOT be included in the VQE-C channel configuration file (sample-chan-config.conf), thus forcing the vqec_ifclient_bind API to fail when asked to bind to that channel.

Using the VQE-C CLI the status of session's VQE-C tuner should be verified, none of the counters should be incrementing for the RTP fallback case.

7.7 Rapid Channel Change (RCC)

This section will describe the steps required for a typical integration of the VQE RCC feature.

Although RCC feature will does not required the use of any additional VQE-C APIs the following steps need to be considered for a successful integration:

1. Enabling support for RCC in SDP channel lineup
2. Rapid Channel Change bind
3. Program, Timing and Conditional Access Information
4. Unsuccessful Rapid Channel Change
5. Memory resource (pakpool_size) considerations
6. Testing and Verification

7.7.1 Enabling support for RCC in the SDP channel lineup

Support for RCC is required to be enabled on a per channel basis via the SDP file.

```
v=0
o=- 1213811561377 1213812810714 IN IP4 vqes-demo-1
s=235.1.1.1
i=Channel configuration for 235.1.1.1 ( None )
t=0 0
a=rtcp-unicast:rsi
a=group:FID 1 3
m=video 1025 RTP/AVPF 96
i=Original Source Stream
c=IN IP4 235.1.1.1/255
b=AS:4000
b=RS:53
b=RR:530000
a=fmtp:96 rtcp-per-rcvr-bw=53
a=recvonly
a=source-filter: incl IN IP4 235.1.1.1 53.7.18.20
a=rtpmap:96 MP2T/90000
a=rtcp:1026 IN IP4 5.3.17.200
a=rtcp-fb:96 nack pli
a=mid:1
:
a=fmtp:99 rtx-time=3000
a=mid:3
```

The addition of the **nack pli** to “Original Source Stream” specification enables the channel for RCC. Enabling of the RCC can done during channel provisioning using the Cisco VCPT.

7.7.2 Rapid Channel Change bind API

In order to enable support for the RCC feature the 3rd parameter (`const vqec_bind_params_t *bp`) of **vqec_ifclient_tuner_bind_client** API needs to be populated accordingly. Note the API prototype has remained unchanged and is compatible with the previous releases of the VQE-C source code where this parameter was set to NULL.

```
vqec_error_t vqec_ifclient_tuner_bind_chan(const vqec_tunerid_t id,
                                           const vqec_sdp_handle_t sdp_handle,
                                           const vqec_bind_params_t *bp);
```

The parameter ***bp** is opaque to caller, with the appropriate handle being created using the following API :

```
vqec_bind_params_t* vqec_ifclient_bind_params_create(void)
```

Once the appropriate ***bp** has been created one o fthe following APIs should be used to specify whether the RCC feature is enabled or disabled for the subsequent bind call (i.e. `vqec_ifclient_tuner_bind_chan`):

```
uint8_t vqec_ifclient_bind_params_enable_rcc(vqec_bind_params_t *bp);
```

```
uint8_t vqec_ifclient_bind_params_disable_rcc(vqec_bind_params_t *bp);
```

Note as indicated in section 7.7.1 the corresponding field in SDP file for the channel must also be specified to enable RCC, in summary both the **vqec_ifclient_tuner_bind_client** API parameter setup and SDP file change are required.

Once `vqec_ifclient_tuner_bind_chan` call has been made the ***bp** can be destroyed using the following API:

```
uint8_t vqec_ifclient_bind_params_destroy(vqec_bind_params_t *bp);
```

The following code example highlights the mandatory changes required to enable RCC:

```
/*
 * Binds a (sample) stream to the VQE-C tuner, if VQE-C was successfully
 * initialized and a tuner created.
 */
int stb_vqec_bind_tuner1 (void)
{
    vqec_error_t err;
    vqec_sdp_handle_t sdp_handle;
    vqec_bind_params_t *bp;

    if (stb_vqec_tuner1_id == VQEC_TUNERID_INVALID) {
         /* VQE-C not yet initialized or initialization failed */
         return -1;
    }

    sdp_handle = vqec_ifclient_alloc_sdp_handle_from_url(
        "rtp://224.1.1.1:50000");
    if (sdp_handle == NULL) {
        printf("failed to acquire cfg handle from url");
        return -2;
    }

    /* bind tuner1 to the channel rtp://224.1.1.1:50000 */
    bp = vqec_ifclient_bind_params_create();
    vqec_ifclient_bind_params_enable_rcc(bp);
    err = vqec_ifclient_tuner_bind_chan(stb_vqec_tuner1_id, sdp_handle, bp);
    vqec_ifclient_bind_params_destroy(bp);
    vqec_ifclient_free_sdp_handle(sdp_handle);
    if (err != VQEC_OK) {
        printf("failed to bind channel (%s)\n", vqec_err2str(err));
        return -3;
    }

    return 0;
}
```

It is envisioned that the ability to enable/disable at bind time can be used to control the type of channel change, so that channels destined for the background records do not perform RCC thus reducing the requirements on the size of the VQE-C packet pool (pakpool_size).

7.7.2.1 Enabling support for Fast Fill

The VQE-C Fast Fill feature is used in conjunction with the RCC feature; the guidelines provided in other sections for enabling RCC must be implemented in order to use the Fast Fill feature. In addition the Fast Fill feature is disabled by default, but can be enabled on a per tuner bind basis using the **vqec_ifclient_bind_params_enable_fastfill**. The **vqec_ifclient_bind_params_disable_rcc** is also provided to disable the feature on a per tuner bind basis.

At a minimum the VQE-C also requires the **vqec_ifclient_bind_params_set_fastfill_ops** to be called with 3 appropriate empty callback functions setup via the **vqec_ifclient_fastfill_cb_ops_t** structure. From most integrations these functions will not be required to preform any operations, however they must be present.

The following code example highlights the mandatory changes required to enable Fast Fill:

```
/*
 * Binds a (sample) stream to the VQE-C tuner, if VQE-C was successfully
 * initialized and a tuner created.
 */

/*
 * When STB binds to a new channel with fastfill enabled,
 * it should pass these three function pointers to VQE-C
 * as call back functions.
 */

static void
fastFillStartHandler (int32_t context_id,
                      vqec_ifclient_fastfill_params_t *params)
{
};

static void
fastFillAbortHandler (int32_t context_id,
                      vqec_ifclient_fastfill_status_t *status)
{
};

static void
fastFillDoneHandler (int32_t context_id,
                     vqec_ifclient_fastfill_status_t *status)
{
};

static vqec_ifclient_fastfill_cb_ops_t fastFillOps =
{
    fastFillStartHandler,
    fastFillAbortHandler,
    fastFillDoneHandler
};


int stb_vqec_bind_tuner1 (void)
{
    vqec_error_t err;
    vqec_sdp_handle_t sdp_handle;
    vqec_bind_params_t *bp;

    if (stb_vqec_tuner1_id == VQEC_TUNERID_INVALID) {
         /* VQE-C not yet initialized or initialization failed */
         return -1;
    }

    sdp_handle = vqec_ifclient_alloc_sdp_handle_from_url(
        "rtp://224.1.1.1:50000");
    if (sdp_handle == NULL) {
        printf("failed to acquire cfg handle from url");
        return -2;
    }

    /* bind tuner1 to the channel rtp://224.1.1.1:50000 */
    bp = vqec_ifclient_bind_params_create();
    vqec_ifclient_bind_params_enable_rcc(bp);
    vqec_ifclient_bind_params_enable_fastfill(bp);
    (void)vqec_ifclient_bind_params_set_fastfill_ops(bp, &fastFillOps);
    err = vqec_ifclient_tuner_bind_chan(stb_vqec_tuner1_id, sdp_handle, bp);
    vqec_ifclient_bind_params_destroy(bp);
    vqec_ifclient_free_sdp_handle(sdp_handle);
```

```
    if (err != VQEC_OK) {
        printf("failed to bind channel (%s)\n", vqec_err2str(err));
        return -3;
    }

    return 0;
}
```

7.7.3 Channel Change Instrumentation

In order to help better facilitate debugging and quantifying of channel change behavior within a VQE-C integrated STB, several instruments are provided by the VQE-C SDK that should be integrated. The information collected by these instruments is displayed in the VQE-C CLI, using CLI command "show tuner name <tuner name> rcc". The display is as below:

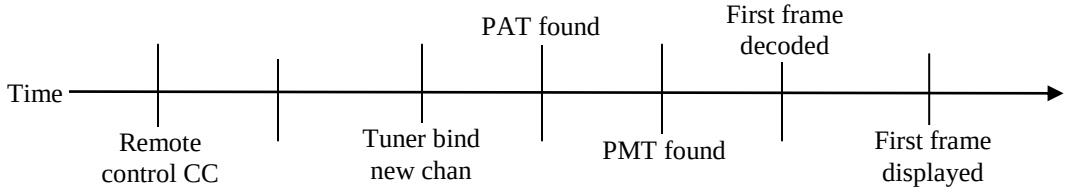
```
STB channel change information (ms):
 remote control CC:          1248985778476

 VQE-C tuner bind:           213
 PAT found time:             259
 PMT found time:             324
 first packet decode time:   527
 total channel change time:  560
Other STB CC information:
 decoded picture snapshot:   16
 last decoded PTS:           1210446092
 first display PTS:          1210407054
```

In the above display, "remote control CC" is the absolute remote control key press time, this is the reference point for all other times displayed. It is given as the number of ***micro***seconds from the Unix timestamp epoch. The "VQE-C tuner bind" is the relative time of tuner bind to a new channel. The "PAT found time" and "PMT found time" are the relative PAT/PMT found times. The "first packet decode time" is the relative time when the first packet was decoded in the STB decoder and the "total channel change time" is the total amount of time of a channel change. The relative times are all provided in milliseconds from "remote control CC". The "decoded picture snapshot" field is the number of pictures decoded by the STB when "first packet decode time" was captured; "last decoded PTS" is the PTS value of the last decoded frame when "first packet decode time" was captured and "first display PTS" is the first display PTS value.

It is assumed that the "decoded picture snapshot", "last decoded PTS", and "first display PTS" can be obtained from the STB subsystem controlling the system decoder.

7.7.3.1 Why channel change instruments are needed.

The main purpose of these instruments is for debugging and quantifying channel change behavior of a STB. In a channel change, usually there are several key points as shown below:



In Fig. 1, several channel change key points are listed in a time axial, these key points can help us debug a channel change behavior and find possible issues.

The “first display PTS” and “last decoded PTS” are used to compare with the I frame PTS value from APP packet in RCC. The I frame PTS is also displayed at the VQE-C CLI. If these three PTSs match, the channel behavior is as expected, no frames are dropped during channel change.

7.7.3.2 How to add these instruments to the STB code.

In vqec_ifclient_defs.h, there is one function which should be registered in the beginning of a channel change from STB, and also one structure to collect this instruments data.

```
/**
* Methods which if registered will allow the client to control and report-on
* on the progress of various CC optimizations.
* We use a structure here for easy extension in the future.
*/
typedef struct vqec_ifclient_get_dcr_info_ops_
{
  int32_t (*get_dcr_stats)(int32_t context_id,
                           vqec_ifclient_dcr_stats_t *stats);
} vqec_ifclient_get_dcr_info_ops_t;


/**
 * Decoder statistics imported by the client.
 */
typedef struct vqec_ifclient_dcr_stats_
{
   uint32_t dec_picture_cnt;          /* decoded picture count */
   uint64_t fst_decode_time;          /* first frame decode time*/
   uint64_t last_decoded_pts;         /* last decoded pts */
   uint64_t ir_time;                  /* remote control CC time- OBSOLETE*/
   uint64_t pat_time;                 /* pat found time */
   uint64_t pmt_time;                 /* pmt found time */
   uint64_t display_time;             /* first frame display time */
   uint64_t display_pts;              /* first displayed PTS value */
} vqec_ifclient_dcr_stats_t;
```

During a channel change, STB collects all this data and records them. These values will be populated when “show tuner name <tuner name> rcc” CLI command is typed.

The instruments “pat_time” and “pmt_time” may be optionally omitted by setting them to zero, obviously this will reduce the information provided on the break down of channel change timing.

Please note, there is a translation between STB side PTS and VQE-C side display. In the vqec_ifclient_dcr_stats_t, all PTS values are in 45KHZ unit, and in the CLI display, all in 90 KHZ. VQE-C already translates them, so in the STB side, use 45 KHZ PCR value.

Care should be taken to ensure that the data used to populate the “vqec_ifclient_dcr_stats” is an accurate snapshot of the channel change, and is not modified until the next channel change. It should be noted that the VQE-C will use the presence of a non-zero value in “display_time” as a indication that the full data set in the “vqec_ifclient_dcr_stats” is available. The data will then be sampled and displayed on request to the VQE-C CLI.

Gathering the IR remote time through the “get_dcr_stats” callback is now obsolete. While existing integrations will continue to function as in previous releases, care must be taken to avoid race conditions gathering channel change time data. The “vqec_ifclient_dcr_stats” structure must contain data relating to only a single channel change at all times. A new tuner bind parameter, “ir_time” accessed by the corresponding bind parameter set function below should now be used to gather the ir_time instead of the “get_dcr_stats” callback.

To take advantage of these channel change instrumentation features the following APIs should be called:

```
uint8_t vqec_ifclient_bind_params_set_context_id(vqec_bind_params_t *bp,
                                                 int32_t context_id);

uint8_t vqec_ifclient_bind_params_set_dcr_info_ops(
                        vqec_bind_params_t *bp,
                        vqec_ifclient_get_dcr_info_ops_t *dcr_info_ops);
uint8_t
vqec_ifclient_bind_params_set_ir_time(vqec_bind_params_t *bp,
                                      uint64_t ir_time);
```

Please see the following example:

```
int dcrRetrieveStats (int32_t context_id, vqec_ifclient_dcr_stats_t *stats)
{
    int dec_picture_cnt;

    /*
     * Integrator specific implementation
     * ----------------------------------
     * Obtain decode picture count from video decoder
     * store in dec_picture_cnt. May use context_id parameter to
     * provide necessary context information etc
     */
    &stats->dec_picture_cnt = dec_picture_cnt;
    &stats->fst_decode_time = fst_decode_time;
    &stats->last_decoded_pts = last_decoded_pts;
    &stats->pat_time = pat_time;
    &stats->pmt_time = pmt_time;
    &stats->display_time = display_time;
    &stats->display_pts = display_pts;

    return (0);
}

static const vqec_ifclient_get_dcr_info_ops_t my_dcr_ops =
{
    &dcrRetrieveStats
};

vqec_ifclient_get_dcr_info_ops_t * get_dcr_ops(void)
{
    return (vqec_ifclient_get_dcr_info_ops_t *)&my_dcr_ops;
}

/*
```

```
 * Binds a (sample) stream to the VQE-C tuner, if VQE-C was successfully
 * initialized and a tuner created.
 */
int stb_vqec_bind_tuner1 (void)
{
    vqec_error_t err;
    vqec_sdp_handle_t sdp_handle;
    vqec_bind_params_t *bp;

    if (stb_vqec_tuner1_id == VQEC_TUNERID_INVALID) {
         /* VQE-C not yet initialized or initialization failed */
         return -1;
    }

    sdp_handle = vqec_ifclient_alloc_sdp_handle_from_url(
        "rtp://224.1.1.1:50000");
    if (sdp_handle == NULL) {
        printf("failed to acquire cfg handle from url");
        return -2;
    }

    /* bind tuner1 to the channel rtp://224.1.1.1:50000 */
    bp = vqec_ifclient_bind_params_create();
    vqec_ifclient_bind_params_enable_rcc(bp);
    vqec_ifclient_bind_params_set_context_id(bp, USER_CONTEXT_ID_TUNER1);
    vqec_ifclient_bind_params_set_dcr_info_ops(bp, get_dcr_ops());
    vqec_ifclient_bind_params_set_ir_time(bp, ir_timestampƒ);
    err = vqec_ifclient_tuner_bind_chan(stb_vqec_tuner1_id, sdp_handle, bp);
    vqec_ifclient_bind_params_destroy(bp);
    vqec_ifclient_free_sdp_handle(sdp_handle);
    if (err != VQEC_OK) {
        printf("failed to bind channel (%s)\n", vqec_err2str(err));
        return -3;
    }

    return 0;
}
```

The **context_id** (specified using **`vqec_ifclient_bind_params_set_context_id`**) would be typically utilized to provide decoder context information; the VQE-C code considers this parameter opaque and will just pass it on unmodified.

7.7.4 Program, Timing and Conditional Access information

Please refer to ITU-T Rec. H.222.0 for reference regarding terminology used in this section.

All transport streams contain the necessary information to allow demux and presentation the Audio/Video content contained within the Packetized Elementary Streams (PES). Typically this information is retrieved by demuxing the incoming transport stream to obtain the necessary program, timing and conditional access information. The multiplexing of this information in the stream is governed by the encoder. Hence the multiple transport stream packets may be required to be demuxed and processed before the complete information set is available for use.

As part of the VQE RCC solution, an essential set of program, timing and conditional access information is provided immediately upon a successful rapid channel change, thus reducing the overall time required to configure relevant subsystems for the new stream.

7.7.4.1 RCC Integration

This section describes the recommended method for integration of the VQE-C RCC feature.

On a successful RCC the first data packet provided by the **vqec_ifclient_tuner_recvmsg()** will contain the following MPEG transport stream packets:

- Program Association Table (PAT) section
- Program Map Table (PMT) section
- Program Counter Reference (PCR) value using packet identifier (PID) as specified in PMT.

In addition the following transport stream packets may also be present:

- Program related access control information using the appropriate PID as specified by the Conditional access descriptor in the PMT.
- Sequence Header

All this "priming" information can then be used to configure and synchronize the appropriate subsystems accordingly, prior to consumption of subsequent data packets from the **vqec_ifclient_tuner_recvmsg()** to ensure successful presentation of the video stream. The system can be configured to provide this "priming" information multiple times, using the ***app_paks_per_rcc*** parameter (please refer to the VQE-C System Configuration Guide).

.

If the VQE-C RCC is disabled then the PSI information is not guaranteed to be present in the first data packets obtained from **vqec_ifclient_tuner_recvmsg()**, i.e. it is dependent on the encoding of the actual stream. The integration code should then fallback to non-RCC method of obtaining the PSI information, typically this would involve parsing the incoming packets until the relevant information has been extracted.

Important System Level Integration Consideration

It is important to consider the complete system behavior and data flow when integrating, in particular the processing of the "priming" information by other subsystems.

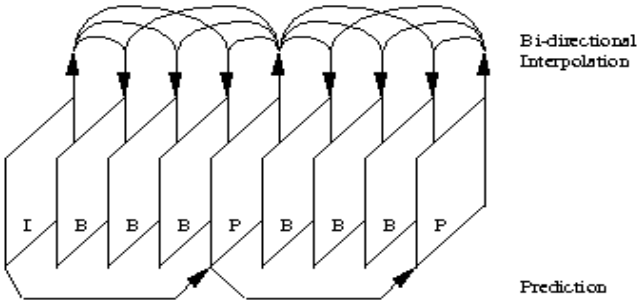
Example:

A typical conditional access system would require that part of "priming" information be provided to a decryption module before video can be successfully decoded. The processing of this information by the decryption module may not be immediate. Hence it is important to ensure that subsequent packets retrieved from the **vqec_ifclient_tuner_recvmsg**() are only feed into the system once the decryption module has completed processing of the "priming" information and is in state ready to decrypt.

As mentioned before the multiple copies of the "priming" information, also referred to as APP packets, may be provided by setting the "app_paks_per_rcc" parameter in the system configuration.

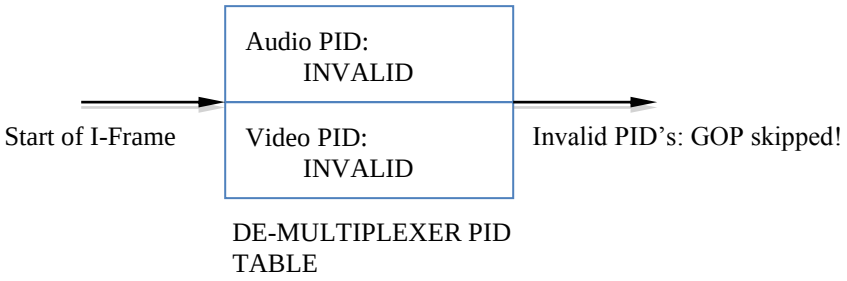
7.7.4.2 "Synchronous" PSI Handling For RCC

The MPEG Program Specific Information for a channel is sent to the de-multiplexer / decoder pipeline in the 1st packet of the Rapid Channel Change (RCC) burst generated by the VQE Client. The PSI data includes the Program Association Table (PAT), Program Map Table (PMT), and if the channel is scrambled, the ECM information. For RCC to cancel the Group-of-Pictures (GOP) related latency on a STB, the first goal is to ensure that the PSI is parsed and the de-multiplexer is programmed with the Audio, Video and CA PID(s) before any payload for these PID(s) is delivered to the de-multiplexer. This particular processing mode is called "Synchronous PSI Handling for RCC". As shown in the figure below, to cancel the channel change time associated with the acquisition of a GOP, the RCC burst must start at an I-frame.



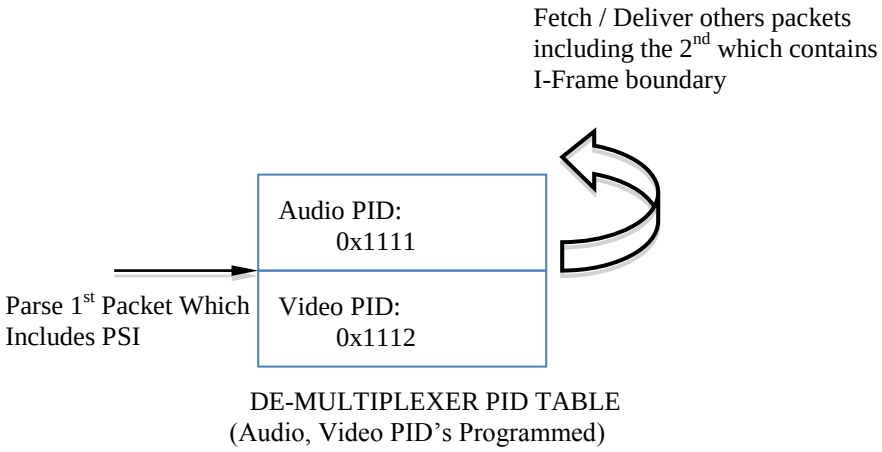
Picture Types (MPEG-2)

However, the de-multiplexer must know at least the Audio and Video PID(s) to collect the incoming audio and video section data, before the packet containing the start of the I-frame is delivered to it. The figure below shows this condition.



Skipped GOP because of invalid PIDs

Hence, the STB middleware must program the de-multiplexer's PID table before any audio or video payload is sent to it. This may be achieved by parsing the 1st packet of the RCC burst that contains the PSI, and programming the de-multiplexer's PID table before any audio or video payload is sent to it. This option is shown in below.



Synchronous PSI handling

7.7.4.2.1 Is STB Middleware Synchronous or Asynchronous for PSI?

If it cannot be pre-determined whether the STB middleware handles PSI synchronously or not, and channel change with RCC appears to be ineffective, then one possibility is to use the 'fallback position' that the middleware is asynchronous. The next section describes a set of debug parameters than may be used to determine if RCC's are ineffective due to asynchronous PSI handling.

7.7.4.2.2 What If STB Middleware Is Asynchronous?

The VQE client provides a couple of debug switches which may help to eliminate or significantly reduce the frequency of skipped GOP's due to asynchronous PSI processing after RCC. However, this may come at a moderate cost: the effective channel change time may be larger than that achievable with synchronous PSI handling.

The basic idea is to replay the packet containing the PSI a user-specified number of times, with a user-specified inter-packet delay, before any audio / video payload is sent to the de-multiplexer / decoder pipeline. The multiple copies of the PSI will work with a middleware that parses PSI sequentially, e.g., the 1st copy is used to parse the PAT and program the PMT PID, while the 2nd copy is used to parse the PMT and program the audio, video PID(s). The inter-packet delay compensates for the time lag that may be inherent in the PSI programming due to asynchronous procedure calls.

The system configuration parameter "app_paks_per_rcc" is used to specify the number of times the PSI data is sent at the beginning of the RCC burst by the VQE client. A reboot of the system is required for the parameter to take effect after a change.

The inter-packet delay for the replayed copies is available as a system configuration parameter in addition to being available via the VQE-C CLI in configuration mode (the parameter will take effect at the take channel change):

```
vqec(config)#
app-delay    Configure app-delay settings
Usage: app-delay <delay>
Where <delay> is the number of msec to delay in between each successive
replicated APP packet
```

The parameter's setting will be lost upon a system reboot if set from the VQE-C CLI.

7.7.4.3 RCC and Conditional Access System Integration

For a 'clear' channel it is important that the PAT and PMT information provided in the first data packet on calling **vqec_ifclient_tuner_recvmsg** is used to configure the system demux with the appropriate PID information before the subsequent packets are feed to the system decoder.

For 'encrypted' channels, in addition to the setup of the demux PID table the conditional access subsystem should also be configured to enable decryption of subsequent packets. Again in a nominal system the ECM information required by the conditional access subsystem will be present in the first packet provided by the VQE-C. Care must be taken to ensure that the information provided by the VQE-C first packet is processed in a manner that ensures that the other subsystems (i.e. demux and conditional access subsystems) are primed and ready to process audio and video payloads in subsequent packets.

Failure to utilize the 'APP' packet information will results channel changes that appear longer than an equivalent channel change for a 'clear' channel using RCC. The following conditions may be observed:

- The VQE-S indicates that a successful RCC has been completed
- The VQE-C indicates that a successful RCC has been completed
- The additional VQE-C RCC statistics indicate that the decoded PTS value does not match the one provided by the VQE-S for the channel change, and as such the GOP cancellation provided by RCC is not utilized by the STB.
- The channel change times have a wide distribution, similar to that seen without RCC
- The conditional access subsystem on the STB reports that it has received packets that it is unable to decrypt, due to not have the appropriate ECM information (i.e. keys etc.)
- Encrypted packets are sent to the system decoder that in turn it is unable to successfully decode then.

It is important to ensure that on reception of the first channel change packet from the VQE-C:

1. The demux PID table is setup with information provided in the first packet
2. The conditional access subsystem is initialized and ready to receive ECM information

3. If conditional access subsystem initialization and setup (i.e. consumption of the ECM) is not synchronous then the appropriate mechanisms should be put in place to ensure that subsequent VQE-C packets are not fed into system until the system is ready to receive them.

7.7.5 Memory Resource (pakpool_size) considerations

The use of the RCC requires the appropriate sizing of the "*pakpool_size*" the system configuration parameter based on the content stream and network characteristics.

The following examples provide guidance on selecting an appropriate *pakpool_size* for RCC enabled systems. These examples make the following assumptions:

- The system contains a maximum of 2 active tuners (i.e. two active channels at a time)
- The maximum aggregate peak bandwidth is15 Mbps
- One channel is enable to support RCC operation
- The output queue depth (*output_pak_limit*) is set to 200
- The IGMP join variance of the network is no more than 200ms

For successful RCCs the "*pakpool_size"* should be sized appropriately. An unsuccessful RCC will result in the system completing the channel bind as if RCC was not enabled.

Regardless of RCC operation the pakpool should always be sized to accommodate the maximum aggregate bit rate of the system..

7.7.5.1 RCC Case 1 & 2 examples

These examples show pakpool_size values for various maximum GOP sizes for both "Conservative" and "Aggressive" RCC modes. The total repair technology induced delay accounts for both jitter buffer and any FEC delay (if applicable).

< need description or reference to explain Conservative and Aggressive modes>

Example A - 200ms delay (Case 1): For example if your system is required to support up to an 8Mbps RCC enabled stream, with an maximum GOP size of 1, and network e-factor of 0.2 then:

- Conservative Mode RCC: 2110 pakpool size
- Aggressive Mode RCC: 1654 pakpool size

Example B - 400ms delay (Case 2): For example if your system is required to support up to an 8Mbps RCC enabled stream, with an maximum GOP size of 1, and network e-factor of 0.2 then:

- Conservative Mode RCC: 2395 pakpool size
- Aggressive Mode RCC: 2034 pakpool size

System Characteristics

*Active Tuners (**max_tuners**)*	2	*tuners*	The maximum number of tuners that can be created in VQE-C.
*Output Queue Per Tuner (**output_pakq_limit**)*	200	*packets*	The limit on the size (in packets) of the output packet queue.
Aggregate Peak Used Bandwidth	15	*Mbps*	The aggregate peak bandwidth of all actively bound streams
IGMP join variance	200	*ms*	IGMP join varanice of the network the system is operating in

Case 1

Repair Technology Delay **200** ms *ms*

Required pakpool for max active tuners and no RCC 685 packets

	Required pakpool_size for max active tuners including 1 RCC enabled channel											
	Conservative						Aggressive					
RCC Enabled Channel Stream Rate (mbps)	GOP = 0.5		GOP = 1		GOP = 1.5		GOP = 0.5		GOP = 1		GOP = 1.5	
	e=0.2	*e=0.4*	*e=0.2*	*e=0.4*	*e=0.2*	*e=0.4*	*e=0.2*	*e=0.4*	*e=0.2*	*e=0.4*	*e=0.2*	*e=0.4*
2	947	845	1042	940	1137	1035	833	826	928	921	1023	1016
4	1208	1004	1398	1194	1588	1384	980	966	1170	1156	1360	1346
8	1730	1322	2110	1702	2490	2082	1274	1246	1654	1626	2034	2006
10	1992	1481	2466	1956	2941	2431	1422	1386	1897	1861	2371	2336

Case 2

Repair Technology Delay **400** ms

Required pakpool_size for max active tuners and no RCC 970 packets

	Required pakpool_size for max active tuners including 1 RCC enabled channel											
	Conservative						Aggressive					
RCC Enabled Channel Stream Rate (mbps)	GOP = 0.5		GOP = 1		GOP = 1.5		GOP = 0.5		GOP = 1		GOP = 1.5	
	e=0.2	*e=0.4*	*e=0.2*	*e=0.4*	*e=0.2*	*e=0.4*	*e=0.2*	*e=0.4*	*e=0.2*	*e=0.4*	*e=0.2*	*e=0.4*
2	1232	1141	1327	1236	1422	1331	1141	1141	1236	1236	1331	1331
4	1493	1312	1683	1502	1873	1692	1312	1312	1502	1502	1692	1692
8	2015	1654	2395	2034	2775	2414	1654	1654	2034	2034	2414	2414
10	2277	1825	2751	2300	3226	2775	1825	1825	2300	2300	2775	2775

7.7.6 Low Memory Optimized RCC

VQE can operate with RCC enabled even in environments where the memory available to the VQE-C application is limited. The idea here is to utilize all or part of the memory which is used by the decoder for buffering, to hold the excess video data that is required to be stored for a successful RCC to occur.This feature is utilized by providing a correct value for **max_fastfill** to be used during a tuner bind. This value can be specified in the VQE-C System Configuration file, or it can be specified directly to the tuner_bind() API. This value should be specified as the maximum number of bytes of video data that can be stored by the decoder buffer. Upon receiving the RCC video data, VQE-C will push out this much data as fast as it is receiving it, in order to offload the excess data from within its own buffers. It should be noted that this feature can only be used with conservative mode RCC.

Another important consideration when using this feature is the size of **output_pakq_limit**, as it can hinder the effectiveness of low memory optimized RCC if set too high. An **output_pakq_limit** that is greater than 20% of the **pakpool_size** may be too large, as the output queue may be using too much of the packet pool, thus depriving the RCC feature of the packet storage that it requires.

The following table shows some good pakpool_size values for the given parameters. This table can serve as a guide when configuring the value of pakpool_size for the memory-optimized RCC case.

System Characteristics

Multicast Lag	20	*ms*	The maximum number of tuners that can be created in VQE-C.
IGMP Join Latency	200	*ms*	IGMP join varanice of the network the system is operating in
max_paksize	1344	*Bytes*	The limit on the size (in packets) of the output packet queue.
max_fastfill	4000000	*Bytes*	The amount of video data that can be pushed into the decoder buffer.

Case 1

Repair Technology Delay **200** ms

RCC Enabled Channel Stream Rate (mbps)	Required pakpool_size for max active tuners including 1 RCC enabled channel											
	GOP = 0.5				GOP = 1.0				GOP = 1.5			
	e=0.1	*e=0.2*	*e=0.3*	*e=0.4*	*e=0.1*	*e=0.2*	*e=0.3*	*e=0.4*	*e=0.1*	*e=0.2*	*e=0.3*	*e=0.4*
2	512	256	171	128	512	256	171	128	512	256	171	128
4	1024	512	342	256	1057	512	342	256	1290	512	342	256
8	2047	1024	683	512	2452	1148	683	512	2918	1613	1033	658
10	2569	1279	853	640	3150	1589	925	640	3731	2171	1506	1088

Case 2

Repair Technology Delay **400** ms

RCC Enabled Channel Stream Rate (mbps)	Required pakpool_size for max active tuners including 1 RCC enabled channel											
	GOP = 0.5				GOP = 1.0				GOP = 1.5			
	e=0.1	*e=0.2*	*e=0.3*	*e=0.4*	*e=0.1*	*e=0.2*	*e=0.3*	*e=0.4*	*e=0.1*	*e=0.2*	*e=0.3*	*e=0.4*
2	512	256	171	128	512	256	171	128	512	256	171	128
4	1024	512	342	256	1057	512	342	256	1290	512	342	256
8	2047	1024	683	512	2452	1148	683	512	2918	1613	1033	658
10	2569	1279	853	640	3150	1589	925	640	3731	2171	1506	1088

7.7.7 Testing and Verification

In addition to actual observations of channel change times by the user, the VQE-C CLI can be used to help verify the operation of RCC. Various VQE-C CLI commands have been updated to include information regarding RCC operations, this section will explain how they may be utilized.

In addition it is often useful to have two systems side by side, one with RCC enabled and one without, both with the same channel lineups. Thus enabling a visual comparison of the RCC and non-RCC systems to be performed.

7.7.7.1 Channel lineup

Ensure that your channel lineup includes RCC enabled channels:

1. First display channels you have available, if no channels are displayed please recheck method you are using to provide the channel lineup (i.e. file, CDI enabled etc.)

```
vqec> show channel
VQE-C channel cfg update:   not in progress
Last update received:       <unknown>
Channel cfg file version:   136a4169327228e2f980f1b03c95e022
Total number of channels:   3

 rtp://229.1.1.9:53202 (STREAM_229_1_1_9)
 rtp://233.131.169.84:14000 (STREAM_233_131_169_84)
 rtp://233.131.169.85:15000 (STREAM_233_131_169_85)
```

   The channel file version is computed based on the MD5 checksum of the file contents, and is used by VQE-C to determine when the file version differs from the version on a VCDS.

2. Display the details of the specific channel that you wish to perform RCC on. The output will indicate if the feature is enabled in the corresponding channel SDP. If RCC for the specific channel is disabled please enabled the feature using the appropriate mechanism (i.e. VCPT, or proprietary channel lineup configuration tool).

```
vqec> show channel rtp://229.1.1.9:53202
Channel name: STREAM_229_1_1_9
Channel sdp_handle: o=- 1201185407773 1213913001575 IN IP4 saturn-iptv
Channel handle: 0x41000008
Channel session identifier: INIP4#-#1201185407773#saturn-iptv
:
Repair stream RTCP sender bandwidth: 53
Repair stream RTCP receiver bandwidth: 53
Error repair: enabled
```

```
Fast channel change: enabled
```

3. Verify RCC is enabled for the specific channel on the VQE-S using the "Application Monitoring Tool"

Note: Enabling/Disabling of RCC on a channel basis is controlled by the channel description contained in the SDP file for both VQE-C and VQE-S.

7.7.7.2 Enable RCC on the VQE-C

The VQE-C system configuration file should include eth following parameter:

```
rcc_enable = true;
```

Use the "show system" VQE-C CLI command to verify the correct setting for "rcc_enable" parameter.

The VQE-C CLI also supports the capability to dynamically enable/disable RCC on a global level, using the following commands:

```
vqec# enable

vqec# configure terminal

vqec(config)# rcc enable

vqec(config)# exit

vqec# show rcc
rcc:                  enabled
```

7.7.7.3 Perform channel change and verify RCC result

Perform the channel change and use the VQE-C CLI command "show tuner name <name> rcc" to verify the RCC result, in particular look at the field "rcc result:"

```
vqec> show tuner name 0 rcc
Tuner name:                 0
--- RCC status ---
 rcc enabled:               true
 rcc result:                success
 cp failure reason:         NONE
 dp failure reason:         NONE
--- Buffer Fill (ms) ---
```

```
 minimum buffer fill:       200
 maximum buffer fill:       3158
 buffer fill from APP:      2175
--- APP expected relative times (ms) ---
 Join      ER       End-Of-Burst
 125       2275     2400
---Actual relative times (ms)---
 CC       Pli      APP      Rep      Join     Prim     ER       Join-lat
 0        7        11       33       149      204      2422     30
--- Pcm snapshots ---
       Head            Tail            Paks
 JOIN  2284            2329            46
 PRIM  2298            2337            40
 EREN  3144            3976            833
--- Output statistics ---
 first primary sequence:    3125
 rcc output loss packets:   0
 rcc output loss holes:     0
 rcc duplicate packets:     26
 repairs in 1st nack:       0
 first packet output time   34
 last packet output time    2363
 first packet decode time   183
```

An intermediate step may be displayed in the “rcc result:” field indicating “ongoing”, the time it takes to transition to “success” will depend on the characteristics of the stream.

Note: The “**`first packet decode time`**” will only contain a valid value if the integration includes the functionality for obtaining the decoder information via “get_dcr_ops()”.

7.7.7.4 Verify packet pool buffer sizing

Use the following VQE-C CLI command to monitor the packet pool buffer following a RCC:

```
vqec# show pak-pool
global input pak pool stats:
 max entries:               1200
 used entries:              76
 high water entries:        1047
 fail pak alloc drops:      0
```

The system configuration parameter “pakpool_size” should be sized appropriately for the streams and associated network characteristics. If it is observed that the “high water entries” equals the “max entries” then the “pakpool_size” should be increased accordingly. It should be noted that the number of entries used for any given channel change will vary based on the timing and characteristics of the stream.

7.8 Error Repair for Unicast VoD Streams

The process of integrating VQE-C for VoD Error Repair differs only slightly from the multicast case described above. Instead of calling **vqec_ifclient_tuner_bind_chan** to perform a channel change, the **vqec_ifclient_tuner_bind_chan_cfg** function is called. The former is used solely for binding a tuner to a pre-existing channel described in VQE-C's channel lineup (as discussed in previous sections of this document), whereas the latter can be used to bind a tuner to a "temporary channel", described by a special API structure: **vqec_chan_cfg_t**. For the purpose of this integration, all VoD streams are considered to be temporary channels.

Underneath the API, VQE-C treats unicast streams differently from multicast channels in the following ways:

1. NAT traversal is not performed on any ports.
2. Temporary channels / sessions do not appear in the channel lineup.

Aside from these small differences, integration is identical to the multicast case. For example, the **vqec_ifclient_tuner_recvmsg** function is used to receive packets from a VoD stream and **vqec_ifclient_tuner_unbind** is used to unbind a tuner whether it is bound to a VoD stream or a multicast channel.

The **vqec_chan_cfg_t** structure is defined in the vqec_ifclient_defs.h header file. Binding to a VoD channel is as simple as filling in this structure and passing it to the bind function:

```
vqec_ifclient_tuner_bind_chan_cfg (const vqec_tunerid_t id,
                                   vqec_chan_cfg_t *cfg,
                                   const vqec_bind_params_t *bp);
```

Most fields in this structure can be zeroed, with the exception of RTCP bandwidth. Header file vqec_ifclient_defs.h provides a number of macros for setting unspecified values. These macros should be used for marking unspecified fields. Fields that contain "_dest_" represent STB addresses and ports. Fields that contain "_source_" represent server-side addresses and ports. For example, if a unicast stream is being sent to IP address 5.3.19.2 on port 44000, then the "primary_dest_addr" field should be set to 5.3.19.2 and the "primary_dest_port" should be set to 44000. Likewise, if the error repair server has IP address 6.14.22.3 then "rtx_source_addr" should be set to that value. At a bare minimum, the "primary_dest_addr", "primary_dest_port", and "bit_rate" fields should be set.

Note that the VQE-S does not currently support unicast error repair.

7.8.1 RTSP Client Integration for VoD

Unicast VoD streams are usually controlled by the Real-time Signaling Protocol (RTSP) or a similar control protocol. This section describes the basic flows steps for setting up a VoD session with VQE-C integrated.

1. Middleware receives a "play" request for a particular VoD content from user.
2. **vqec_ifclient_socket_open** called for reserving primary and retransmission sockets.
3. RTSP SETUP used to exchange transport and content information.
4. RTSP DESCRIBE used to learn content details (if necessary).
5. **vqec_ifclient_chan_cfg_parse_sdp** called to parse SDP description (optional).
6. **vqec_chan_cfg_t** structure populated with session information.
7. **vqec_ifclient_tuner_bind_chan_cfg** called to tune to the VoD stream.
8. NAT traversal between STB and server is initiated (if necessary).
9. RTSP PLAY used to start play out of content from server.
10. **vqec_ifclient_tuner_recvmsg** used for receiving video packets.

…

11. RTSP TEARDOWN used to stop play out and end session.
12. **vqec_ifclient_tuner_unbind** called to unbind tuner.
13. **vqec_ifclient_socket_close** called for releasing VQE-C sockets.

The call to **vqec_ifclient_socket_open** is used for reserving VQE-C port numbers prior to the call to **vqec_ifclient_tuner_bind_chan_cfg**. This function for opening sockets is necessary to guarantee that the ports the RTSP client offers in the RTSP SETUP are actually available on the client. After the session is complete, **vqec_ifclient_socket_close** is used to close the same sockets. Note that if error repair is expected for the session, these functions must be called twice: once for the primary stream and once for the retransmission stream.

The **vqec_ifclient_chan_cfg_parse_sdp** function may be used for parsing a VoD SDP into a **vqec_chan_cfg_t** structure. Alternatively, this function can be used to set the **vqec_chan_cfg_t** structure to its default values by passing in an empty SDP buffer.

Once the RTSP SETUP exchange is complete, the **vqec_chan_cfg_t** structure may be populated with all available transport information. Server side addresses come from the RTSP SETUP response while client side addresses come from the calls to **vqec_ifclient_socket_open**.

Calling **vqec_ifclient_tuner_bind_chan_cfg** should be done before any RTSP PLAY request to avoid missing packets.

7.9 MIB Compatible counters

To support SNMP MIB views similar in spirit to views of counters provided to SNMP methods that populate MIBs in IOS and IOS-XR, CLI and API support is now made available to export a version of the VQE-C statistics counters that are not reset (remain 'free-running') even when the counters are cleared via "clear counters" CLI command or from the ifclient API.

To the VQE-C integrator, this functionality now provides two additional APIs that support retrieval of this cumulative/free-running view of VQE-C counters. Pre-existing (before this feature) APIs for retrieving VQE-C statistics now provide the statistics since last reset.

In order to retrieve the tuner cumulative statistics, **vqec_ifclient_get_stats_cumulative()** API should be called. This will yield the rolled-up/historical counters across all tuners since VQE-C initialization. including events which occurred on tuners/channels which may no longer exist (or whose binding has changed). The API **vqec_ifclient_get_stats()** will return rolled-up/historical counters across all tuners/channels since **vqec_ifclient_clear_stats()** API was called.

To retrieve per-channel cumulative statistics, **vqec_ifclient_get_stats_channel_cumulative()** API should be called. This will return counters for a channel since it became active to 'now' (i.e. the time when this API was called). To retrieve per-channel statistics since the last call to **vqec_ifclient_clear_stats()** API, **vqec_ifclient_get_stats_channel()** API should be called.

7.10 TR-135 Compatible counters

VQE-C offers API and CLI support for counters that could potentially be used by a TR-135 Application as part of populating it's own data model's parameters.

The API's that could be used by a TR-135 application to parameterize/retrieve TR-135 statistics are as follows:

[API #1] **vqec_ifclient_set_channel_params_tr135()**
[API #2] **vqec_ifclient_bind_params_set_stats_config_tr135()**
[API #3] **vqec_ifclient_get_stats_channel()**
[API #4] **vqec_ifclient_get_stats_channel_cumulative()**
[API #5] **vqec_ifclient_get_stats_channel_tr135_sample()**

The first two APIs would be used by the TR-135 Application to set an active channel's TR-135 writable parameters (gmin and severe loss minimum distance) either during the binding of a tuner to a channel (**API #2**), or by updating the TR-135 parameters for an active channel (**API #1**).

Statistics for a VQE-C channel may accumulate from the time the channel is created until the time it is destroyed. Stats may be retrieved for a channel at any time that the channel is active by specifying the channel by URL (containing destination IP and port) via [VQE-C **API #3**, **API #4**, or **API #5**]. Once an active channel is destroyed, its stats may no longer be retrieved.

Note that active channels may be created and destroyed implicitly by VQE-C as a consequence of calls to `vqec_ifclient_tuner_bind_chan()` and `vqec_ifclient_tuner_unbind_chan()` (when the number of tuners bound to the channel transitions from 0 to 1 or 1 to 0, respectively).

TR-135 Applications that wish to "roll-up" a channel's stats can safely assume its stats always start at 0 at the time of binding its initial tuner. Just prior to the unbinding of the last tuner from the channel, the TR-135 application must invoke [VQE-C **API #3** or **API #4**] in order to retrieve the accumulated statistics (i.e. just before the channel is destroyed).[1]

Support is provided by VQE-C within [VQE-C **API #3** or **API #4**] for retrieval of the following TR-135 objects below, listed by group. The sample statistics (for sample intervals defined by the TR-135 Application could be collected by [VQE-C **API #5**].)

- .STBService.{i}.ServiceMonitoring.MainStream.{i}.Total.DejitteringStats.
  - Overruns
  - Underruns
- .STBService.{i}.ServiceMonitoring.MainStream.{i}.Total.RTPStats.
  - PacketsExpected
  - PacketsReceived
  - PacketsLost
  - PacketsLostBeforeEC
  - LossEvents
  - LossEventsBeforeEC
  - SevereLossIndexCount
- .STBService.{i}.ServiceMonitoring.MainStream.{i}.Sample.RTPStats.
  - MinimumLossDistance
  - MaximumLossPeriod
- .STBService.{i}.Components.FrontEnd.{i}.IP.Dejittering.
  - BufferSize

In each of the **API #3**, **API #4**, and **API #5**, the meaning of 'sample interval' is different, or in other words, each of these APIs are designed to report statistics for different time windows. The TR-135 application can make use of these API's are per its requirements. The statistics time windows for each of the above APIs are as follows:

1) **vqec_ifclient_get_stats_channel()**: channel reset to 'now'
2) **vqec_ifclient_get_stats_channel_cumulative()**: channel init to 'now'

[1] Note: VQE-C does not currently support a means to pass back or notify interested parties of a channel's stats upon its deletion (or each upon binding and unbinding).

3) **vqec_ifclient_get_stats_channel_tr135_sample()**: previous call to vqec_ifclient_get_stats_channel_tr135_sample() (or channel initialization) to ‘now’. This can be used by the TR-135 App to get ‘TR-135 sample statistics’.

where ‘now’ means the time-instant at which the API would be called by the TR-135 Application. **API #3** and **API #4** return **vqec_ifclient_stats_channel_t** data structure that carries the relevant TR-135 statistics.

Note that ‘sample interval’ defined by API#5 and the statistics gathered by that API are not disturbed/touched/unpolluted by any other TR-135 related API. This allows the TR-135 application to get a undisturbed sample statistics count as returned by the **vqec_ifclient_stats_channel_tr135_sample_t** data structure.

The basic flow for using these APIs would be first to bind a tuner to a channel and set up the channel to gather TR-135 parameters using **API #1**, or **API #2**

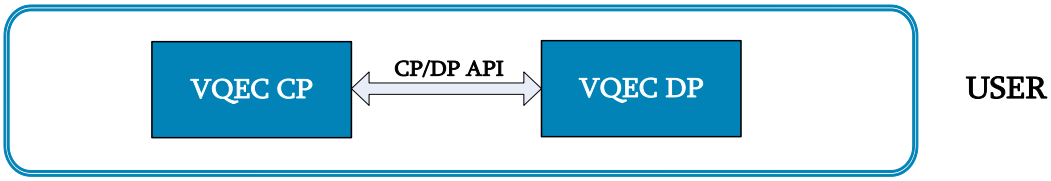
After this, the TR-135 API could use **API #3**, **API #4**, or **API #5**, as per its desired ‘time window’. **API #5** will yield the ‘TR-135 sample statistics’ as defined by the TR-135 specification.

8 Porting VQE-C to the Kernel Space

The VQE-C architecture is essentially divided into two parts “control-plane” (CP) and “data-plane” (DP). The term "control-plane" is defined as the subset of the VQE-C implementation which deals with dataflow setup, statistics collection and signaling to the VQE-S (server). The term "data-plane" is then defined as the subset of the VQE-C implementation that receives video packets and related video repair information and constructs a repair stream of packets. In the case of retransmission based repair the dataplane communicates to the control plane the missing packets and the control plane (RTCP protocol) signals the VQE-S of the missing packets.

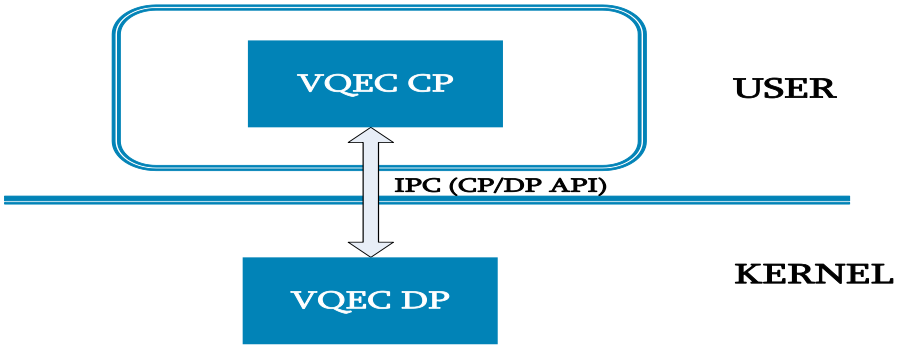
This separation of functionality in VQE-C provides an abstraction for the integrator to support a wide variety of integration models. Prior releases of VQE-C only supported the integration of VQE-C’s CP and DP in the user space as part of a single process (Model 1 below).

MODEL 1 (Single process multithreaded USER space integration)



The VQE-C SDK has been updated to readily support integrations where the VQE-C CP runs in *user-space* and the VQE-C DP runs in the *kernel-space*.

MODEL 2 (DP Kernel space integration)



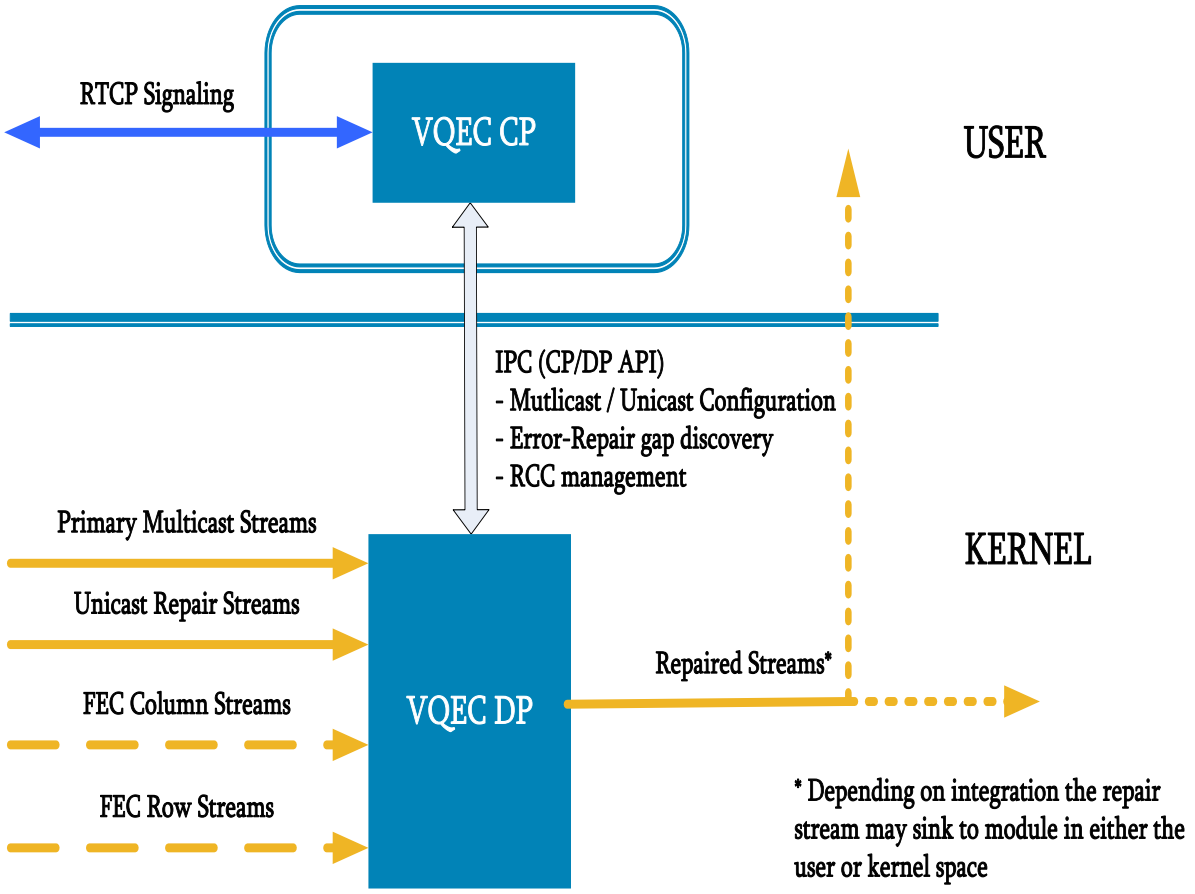
The decision on type of integration performed will depend on the target platform, associated architecture and system requirements. The remainder of the section describes how to approach a VQE-C DP *kernel-space* integration, like the one shown above. The primary motivations for moving the VQE-C DP into the kernel-space is:

- Dataplane processing on most target systems is already done within the context of kernel space.

- Performance – typically for most target systems an improvement in overall system performance will be seen by keeping the data packets flowing in the kernel space context only.

The VQE-C CP has been implemented to exist in the *user-space* only, where typically main control and STB middleware would reside.

The diagram below shows control and data flow for VQE-C DP ported to the *kernel-space*. It should be noted that all of the RTCP signaling from the VQE-C originates in *user-space*, with the VQE-C CP using the target platform's network stack to transmit and receive RTCP protocol.



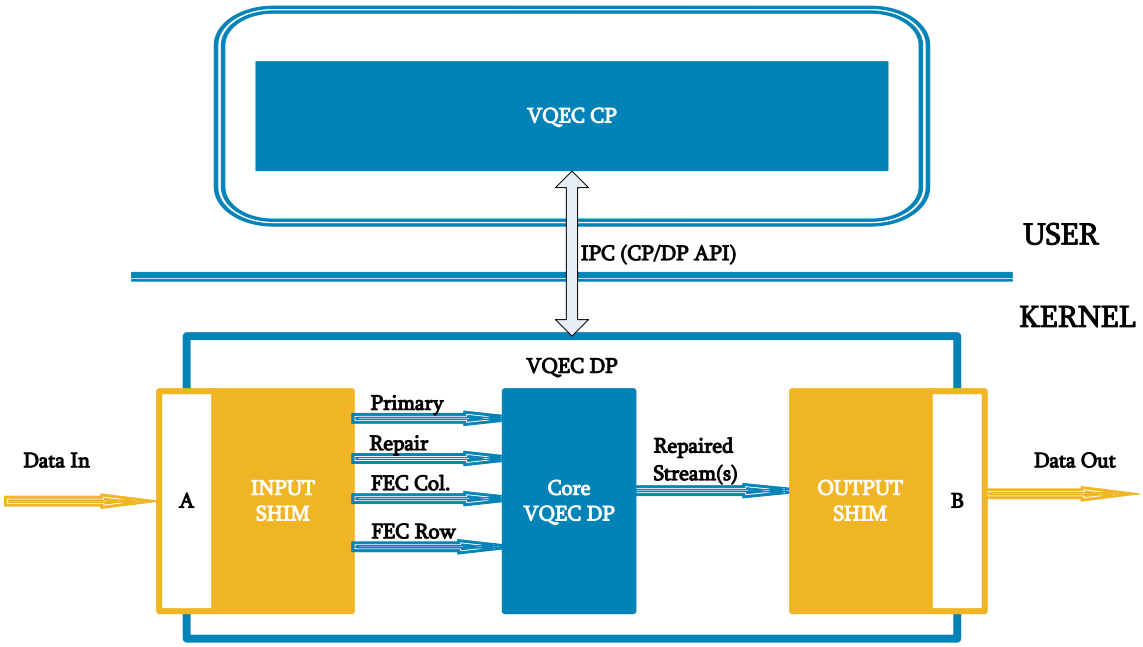
8.1 Tuner creation and Channel binding

The approach to VQE-C tuner creation and channel binding requests is unchanged and should be integrated in that same manner as with a full VQE-C *user-space* integration, using the appropriate ifclient interfaces (see section 7). All of the necessary communication between the VQE-C *user-space* and *kernel-space* modules is handled internally via the use of Remote Procedure Calls (RPC).

8.2 Input & Output Shims

The VQE-C DP has been designed to abstract the input and output packets in system using the "Input Shim" and "Output shim" modules.

The input shim receives and delivers network packets for a channel's primary video, repair and FEC streams to the corresponding channel input receiver. The output shim consists of queues that act as the receiver or sink of output packets that are sourced from the core VQEC DP module.



In addition to the VQE-CP integration in user space, the integrator must consider the "Input Shim" and "Output Shim" integration points A and B respectively. The following sections explain these integration points in more detail. It should be noted that additional custom modifications to "Input Shim" and/or the "Output Shim" may be required for certain integrations, these shims provide the necessary level of abstraction to achieve this.

8.2.1 Input Shim (kernel integration point A)

8.2.1.1 Description

The input shim is a dataplane entity. It's main function is to receive and deliver network packets for a channel's primary video and repair streams to the core VQE-C DP. The dataplane channel will process the incoming packets, detects gaps, and requests repairs.
This main advantage of this approach is that the manner in which packets are received from the network interface is decoupled from their processing in the core VQEC DP code. The expectation is that customers who integrate the VQEC into their existing dataplanes may have proprietary mechanisms for reception and delivery of primary video streams, the "Input Shim" provides the required flexibility to integrate easily into such systems. The input shim interface is designed to enable the VQEC to use the same mechanisms for reception of primary, repair and FEC streams.

The input shim source code can be found within **/eva/vqec-dp/inputshim**.

8.2.1.2 Reference / Recommended Integration Method

A reference implementation of the "Input Shim" is provided as part of the standard VQE-C source code release. This code uses the kernel socket API **`recvmsg()`** that is called by **vqec_recv_sock_read_pak()** in **/vqecutils/kmod/ vqec_recv_socket.kmod.c** to retrieve packets from the appropriate data socket interface.

It should be noted that in the reference implementation the **`recvmsg()`** kernel API allocates a buffer using the common structure **struct sk_buff** from the kernel memory. As a result on final consumption of this data the kernel memory is returned using the kernel API **kfree_skb()** within the **vqec_pak_free()**. If changes are required that eliminate the use of **recvmsg()** then the appropriate changes should be also made to **vqec_pak_free()**.

In summary most integrations will required minimal, if any changes to the "Input Shim" and those that do should only need to modify the appropriate source code contained within **/vqecutils/kmod/**

8.2.2 Output Shim (kernel integration point B)

8.2.2.1 Description

The output shim is a dataplane entity. It's main function will be to facilitate the delivery of packets for a repaired channel directly to either to a audio/video demux driver interface in the kernel, or to a queue for types of data sinks (e.g., a socket), obviously the final consumer of these packets is platform specific (typically a video/audio demux and decoder). To receive packets from the output shim for any repaired channel, a tuner needs to have been created.

Tuner creation, and mapping (binding) of tuners to channels is controlled through the ifclient interface (**`vqec_ifclient_tuner_create`**) within user space. At most one channel can be

mapped or bound to a single tuner at any time. This mapping can be changed at any time through the ifclient interface.

The interfaces provided by the output shim can be divided into two categories: an inbound interface that receives packets for a repaired channel mapped to a particular tuner; an outbound interface which will send these packets to either some device e.g., a demux or to a queue, such as a socket.

The output shim source code can be found within **/eva/vqec-dp/outputshim**.

8.2.2.2 Reference / Recommended Integration Method

The standard VQE-C release provides a reference implementation of the ifclient interface **`vqec_ifclient_tuner_recvmsg()`** for use in the *kernel-space* that consumes data from the "Output Shim". Support of the **`vqec_ifclient_tuner_recvmsg()`** provides an easy method for porting to kernel space, equivalent to a typical user space model integration.

The **`vqec_ifclient_tuner_recvmsg()`** requires that you provide a tuner id as the first parameter, this is the id that was return on creation of the tuner using **`vqec_ifclient_tuner_create()`**. However as the calls to create the tuners where made in user space the returned value will not be directly available in the kernel space, hence the following API should be used to obtain the relevant tuner id:

```
vqec_ifclient_tuner_get_id_by_name(const char *namestr);
```

Using the above API relies on that fact that well known names are given to the relevant tuners on creation.

Note: Functionally for kernel space the **`vqec_ifclient_tuner_recvmsg()`** behaves in the same manner as when used in the user space, with the one exception that is always non-blocking, regardless of the value pass to the timeout parameter.

8.2.2.3 Support for zero copy transfers

The use of **`vqec_ifclient_tuner_recvmsg`** in the *kernel space* introduces a copy of data from to the caller specified buffer(s).

However additional study of the reference implementation of the *Input* and *Output Shims* will highlight that in fact a zero copy transfer mechanism could be implemented by the integrator. This type of integration would involve using the interfaces provided by the *Input* and *Output shims* directly.

8.2.2.4 Interprocess Communication (IPC)

In user-space integrations of the VQE-C DP direct calls between the CP and DP are made, no IPC is used (refer to **eva/vqec-dp/vqec_dp_api.h)**.

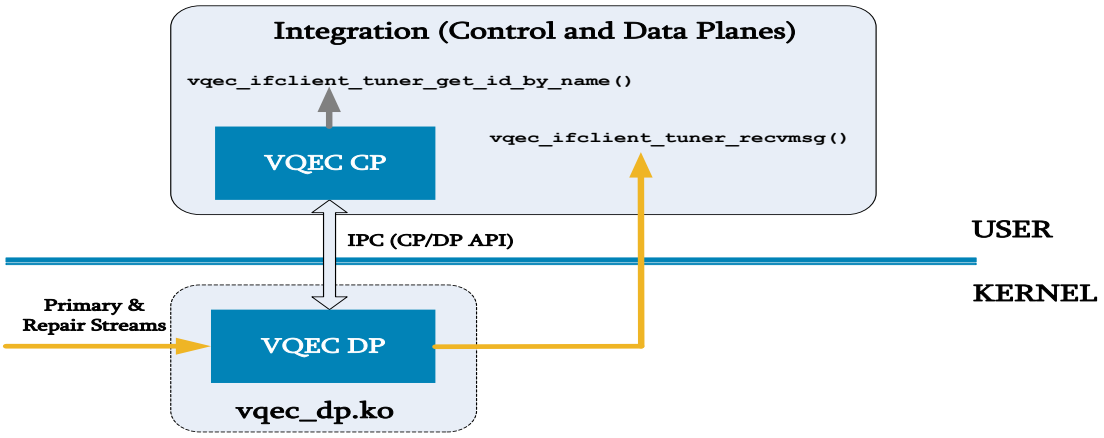
For the kernel implementation the use of an interprocess communication scheme is required, as such the VQE-CP acts as an IPC client and the DP as an IPC server. The APIs defined in **eva/vqec-dp/vqec_dp_api.h** marked with the "RPC" tag, this tag is used by the perl script **eva/vqec_rpcgen.pl** to generate automated clinet / server IPC code. This stub will implement the IPC calls using ioctl's, and invoke the server side in the kernel using the device-driver interface.

8.3 Kernel mode and VQE-C System Configuration flags

The use **vqec_ifclient_tuner_recvmsg()** and **vqec_ifclient_tuner_get_id_by_name()** either in *user* or *kernel space* depends on the setting of the "deliver_paks_to_user" in the system configuration file passed in on VQE-C initialization

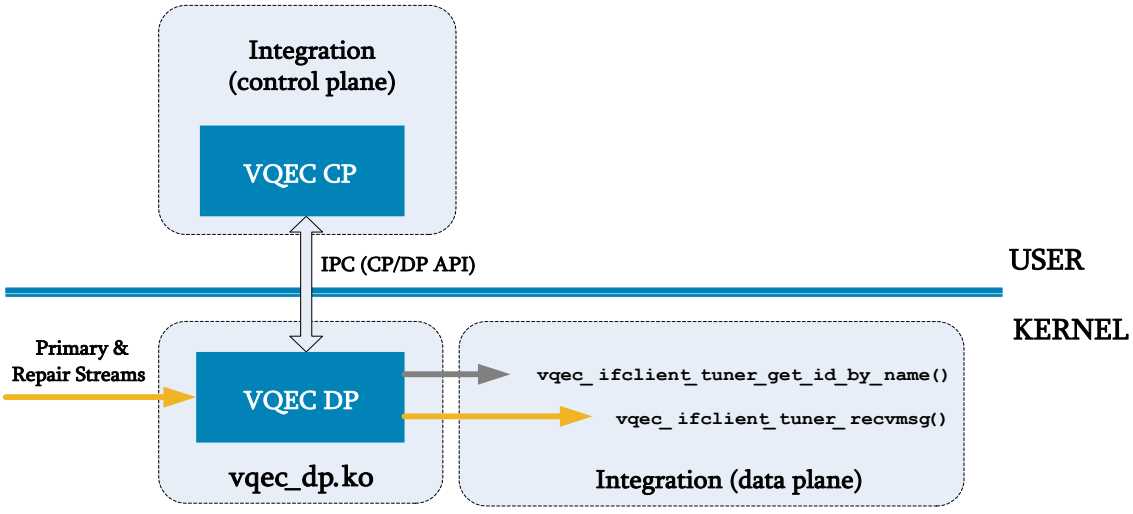
If the parameter is set to "deliver_paks_to_user = true;" than calls to both APIs should be made from the *user space*. In this mode the repaired data stream is transferred by the VQE-C from *kernel* to *user space* via the use of an internal socket.

deliver_paks_to_user = true;



If the parameter is set to "deliver_paks_to_user = false;" then both APIs should be called from the *kernel space* context This can be used for integrations where no video processing takes place in user-space. An all kernel userspace implementation eliminates a data copy between the kernel an userspace which occurs when are read from a BSD style socket in the USER mode integration.

deliver_paks_to_user = false;

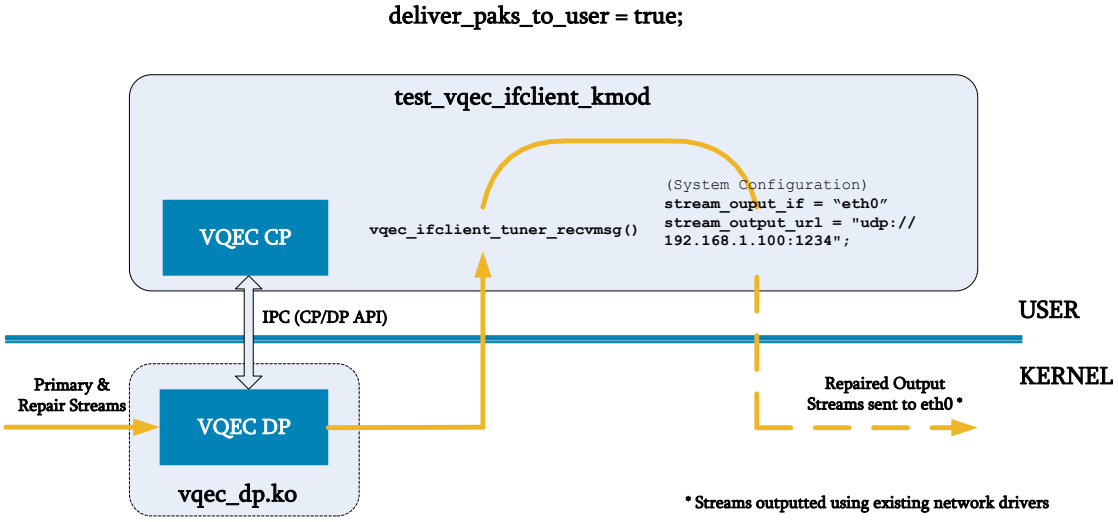


8.4 Example DP Kernel Module and Test Client

As part of the standard VQE-C release an sample application is provided for both *user* and *kernel space* implementations. For purely user space implementations the sample application **test_vqec_ifclient** is available, with **test_vqec_ifclient_kmod** being the equivalent application of use with *kernel space* port of VQE-C DP. The functionality is equivalent for both versions of the sample application; the difference between them is the type of dataplane to which they are linked. Th **test_vqec_ifclient** is linked to the all *user-space* model of the dataplane, while **test_vqec_ifclient_kmod** requires the kernel version of the dataplane.

For **test_vqec_ifclient_kmod** although all primary and repair stream data is received by the VQE-C DP kernel module in the *kernel space*, ultimately this data is passed up into the *user space* with the system configuration parameter "deliver_paks_to_user" being set to true. Based on the system configuration an output stream may be setup to stream out the repaired stream to a specified interface and url. This is useful for validating a port to a new processor or testing the basic functionality without performing a integration. See section 7 for details on doing a full integration.

Support for stream output functionality is currently not provided from the kernel-space (i.e. with "deliver_paks_to_user = false") for the test_vqec_ifclient application. The sample application should only be used as a method to demonstrate basic functionality and aid in the initial integration work.



After following the steps in section 6.4 to build the VQE-C CP the following sample application will have created:

vqec_obj/eva/sample/<ARCH>/test_vqec_ifclient_kmod

The following steps should be taken to execute the **test_vqec_ifclient_kmod**:

1. Load **vqec_dp.ko** into target system, typically using insmod

2. Take the example VQE-C System Configuration file in eva/sample/ sample-vqec-config.conf and modify appropriate for the target system
3. Execute **test_vqec_ifclient_kmod** from user space:
   `./`**`test_vqec_ifclient_kmod`** `sample-vqec-config.conf &`
4. Log into the VQE-C CLI and use the available commands (refer to ..) to verify operation.

Below is typical system configuration file for use with the test_vqec_ifclient_kmod, note the inclusion of the "deliver_paks_to_user = true":

```
/*
 * Copyright (c) 2007-2009 by Cisco Systems, Inc.
 * All rights reserved.
 */
sig_mode = "NAT;
input_ifname = "eth0";
output_ifname = "eth0";
max_tuners = 4;
libcli_telnet_port = 8182;
pakpool_size = 1200;
cdi_enable = true;
deliver_paks_to_user = true;
so_rcvbuf = 1000000;
```

8.5 VQE-C DP Module loading

The VQE-C DP ports to the *kernel-space*, the module must always be loaded before the ifclient interface vqec_ifclient_init() is called in the *user-space*. Typical this would be done at the same time as other modules loaded in the system (i.e. network drivers etc.).

The relevant node should first be created for the VQE-C DP module (vqec0):

```
mknod -m 777 /dev/vqec0 c 248 0
```

The module may then be loaded:

```
insmod /lib/modules/vqec_dp.ko vqec_major_devid=248
```

The following output may be displayed on the standard console interface following successfully loading of the DP module:

```
<vqec-dev>Attempt to register vqec device with major id 248

<vqec-dev>Registered vqec device with major-id 248
```

General information regarding the module may be obtained in the normal manner:

```
uclibc[dev]$ lsmod
Module                  Size  Used by    Tainted: PF
vqec_dp                 479984  3
```

9 Sequence Diagrams

9.1 VQE-C Configuration Updates with CDI Enabled

The diagram below shows how a VQE-C enabled STB gets updated system and channel configuration information from an RTSP server. This scheme depends on a SRV record having been set up in the DNS server which points to one or more equivalent RTSP servers. The SRV record for this service has a service name of **vqe-channel-cfg** rather than **rtsp** to avoid colliding with other RTSP services. The SRV record is placed in the zone file for the domain of the service providers, e.g. example.com

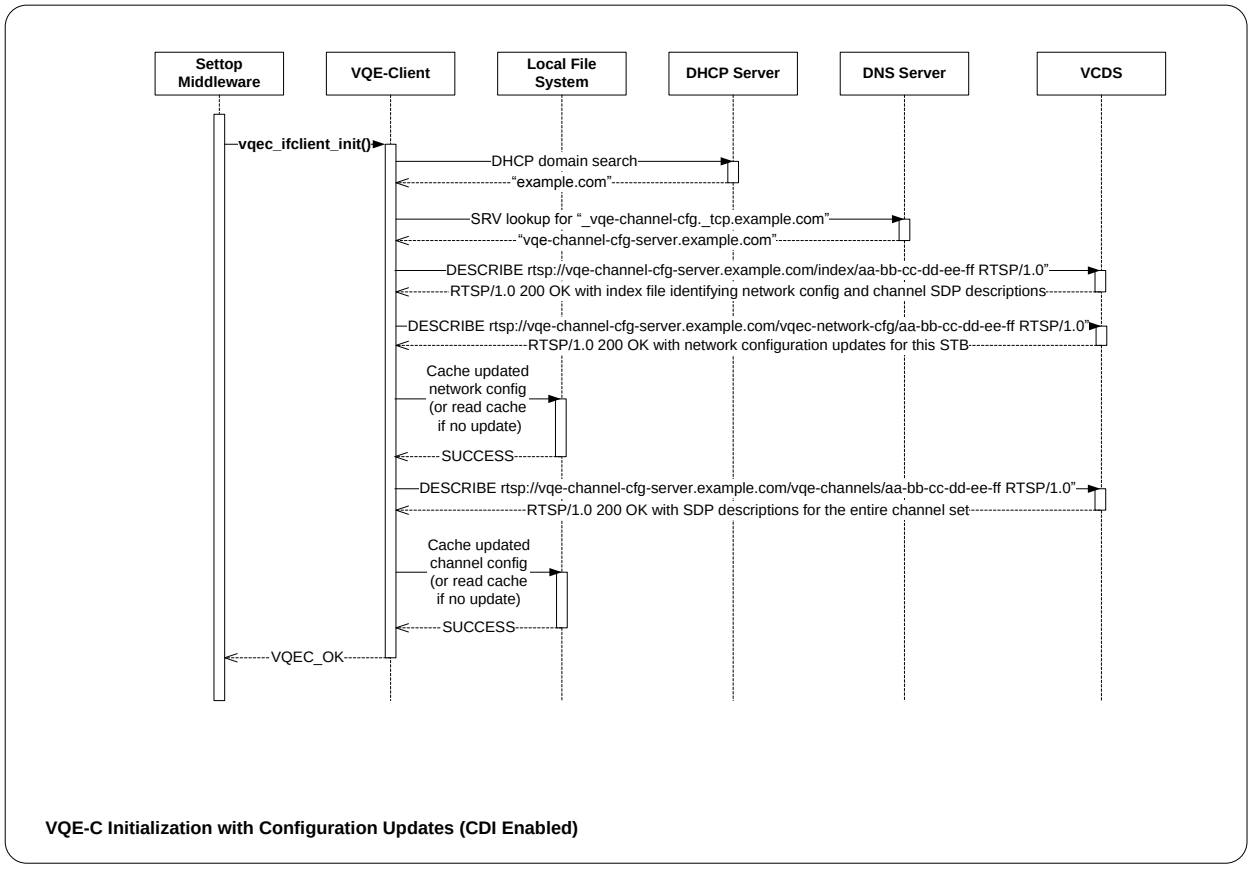


Figure 2 VQE-C Initialization with Configuration Updates (CDI Enabled)

At start-up, the VQE-C first sends out a DNS query for _vam-channel-cfg._tcp. Because the STB may be behind a NAT device, the DNS query will normally go to the service provider's DNS server. If the query succeeds, the STB obtains the RTSP server name defined in SRV records or a list of server names (if multiple are specified). If the query fails, the STB tries a DHCP domain search, which will return "example.com" for the domain which the RTSP server is part of. Then, the STB does another DNS query based on "_vam-channel-cfg._tcp.example.com". If the query is successful, the STB parses out the RTSP server name defined in SRV records. With

knowledge of the RTSP server name, the STB sends out an RTSP DESCRIBE request for an *index* file that includes (embedded in the request) a unique identifier for the STB. The VCDS responds with an *index* file containing the versions of a system configuration update file (called a network configuration file) and channel configuration file available for the requesting STB.

The VQE-C will then request an updated network configuration file and channel configuration file if the version it currently has cached for each differs from the one being advertised:

- The network configuration file contains updates to be applied by VQE-C prior to initialization
- The channel configuration file contains channel configuration data for the entire channel lineup in SDP syntax.

Upon receipt of an updated file, VQE-C will write the received file to its cache based on its configuration.

If the advertised file version matches that in VQE-C's cache, or if a VCDS cannot be reached, then initialization continues using the files in the VQE-C local cache. If appropriate channel configuration cannot be obtained, the VQE-C does not perform error repair.

9.2 VQE-C Initialization and Packet Reception

The diagram below shows a typical sequence involved in getting the VQE-C initialized and configured, and then supplying a repaired video stream to STB client.

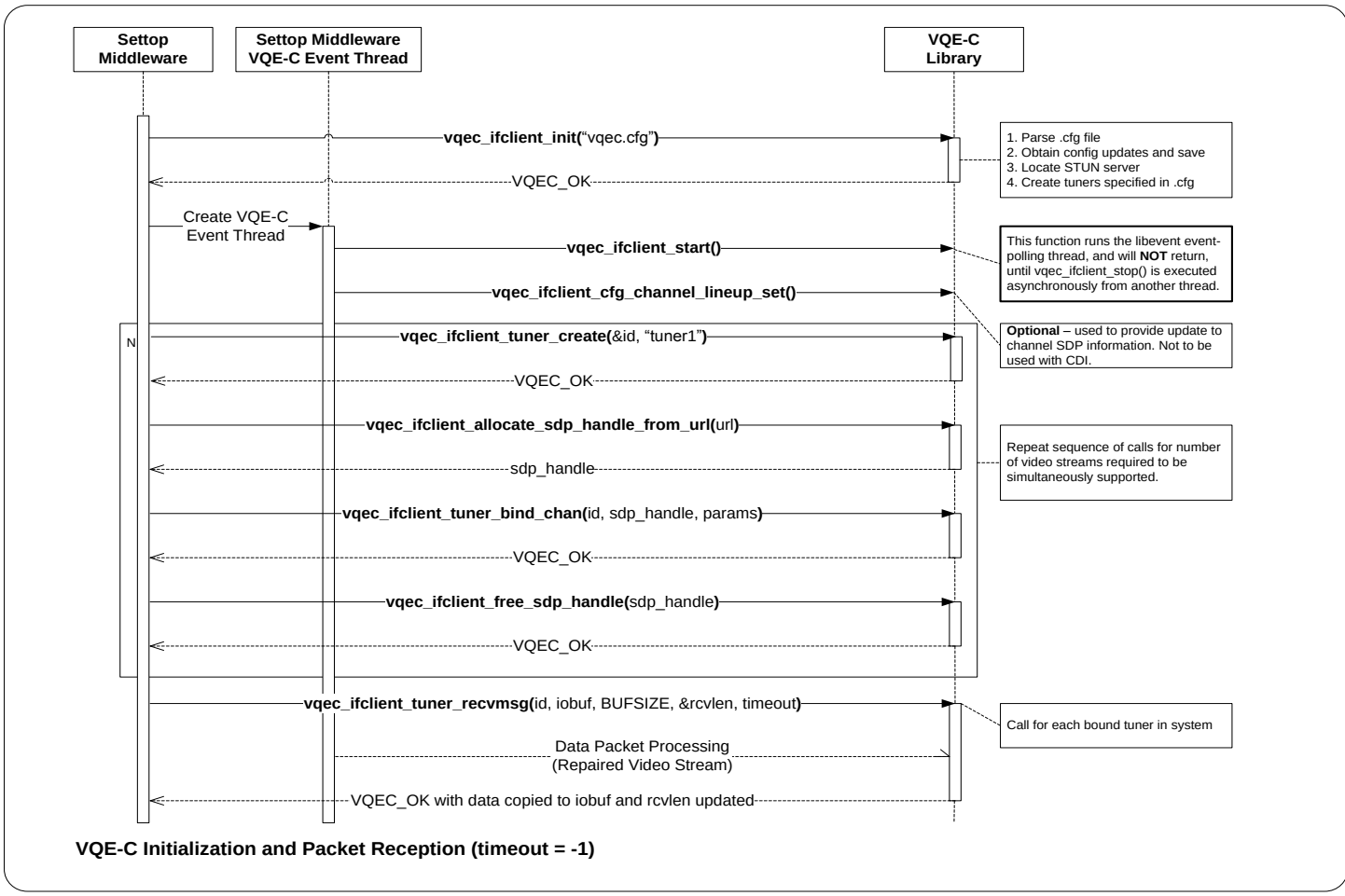


Figure 3 VQE-C Initialization and Packet Reception

During the course of STB initialization the middleware is expected to call **vqec_ifclient_init()**, passing in the System Configuration filename. On completion of this call the VQE-C will have configured its internal modules, attempted to retrieve the channel configuration and entered an initialized state.

In order for the VQE-C to be started the STB middleware needs to create a dedicated thread in which **vqec_ifclient_start()** can be called. This call does not return unless there is some error or the **vqec_ifclient_stop()** function is called from a different thread by the STB middleware.

The channel SDP descriptions may be updated using **vqec_ifclient_cfg_channel_lineup_set()**, this will override any configuration provided by an channel lineup file specified in the system configuration file. This API should not be used when operating in CDI mode.

Unless tuners are created and configured in the System Configuration file, the STB middleware needs to explicitly create and bind the appropriate number of tuners (based on how many simultaneous video streams are required). In order to bind a tuner to a particular channel, an SDP handle must be created for that channel, which is done using the **vqec_ifclient_allocate_sdp_handle_url()** call. On return from the **vqec_ifclient_bind_chan()** call this handle should be freed by the caller.

The VQE-C is now ready to receive data packets and provide a repaired video stream via calls to **vqec_ifclient_recvmsg()**.

9.3 VQE-C Channel Change

The diagram below shows a typical sequence involved in performing a channel change.

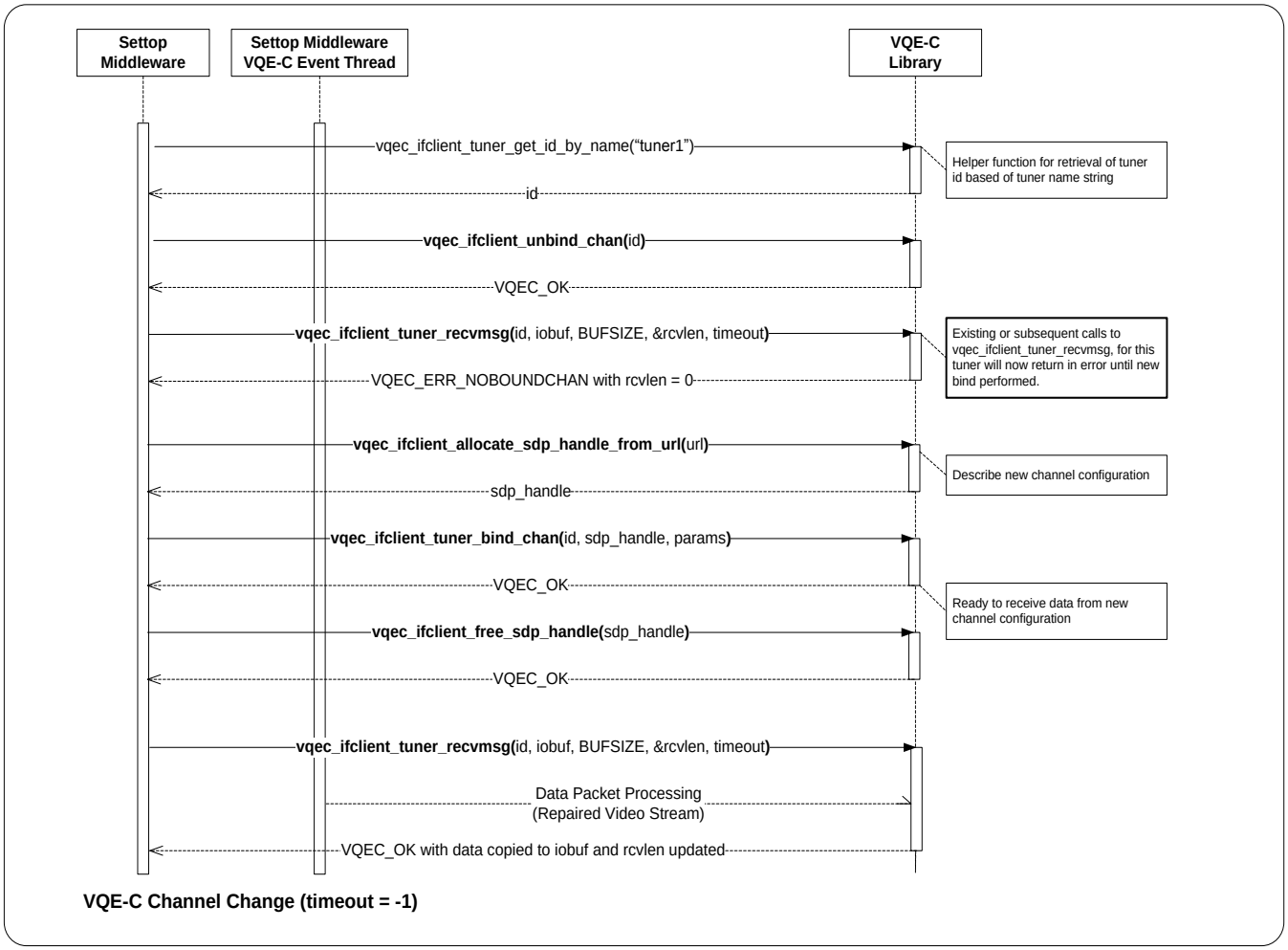


Figure 4 Channel Change

It is assumed that the tuner is already bound and may be receiving data for a particular video stream. On calling **vqec_ifclient_unbind()**, the tuner stops receiving and providing data to the STB client and any further calls to **vqec_ifclient_recvmsg()** for that tuner will return with the error code VQEC_ERR_UNBOUNDCHAN.

Once the new channel configuration has been bound to the tuner, it will again start receiving and providing data to the STB client (again, via **vqec_ifclient_recvmsg()**).

9.4 VQE-C Stop and De-initialization

The diagram below shows the sequence involved in stopping and de-initializing the VQE-C. On completion of this sequence, all resources allocated previously by the VQE-C will have been freed. Once a call to **vqec_ifclient_deinit()** has returned, all further calls to **vqec_ifclient_recvmsg()** will return with an error code.

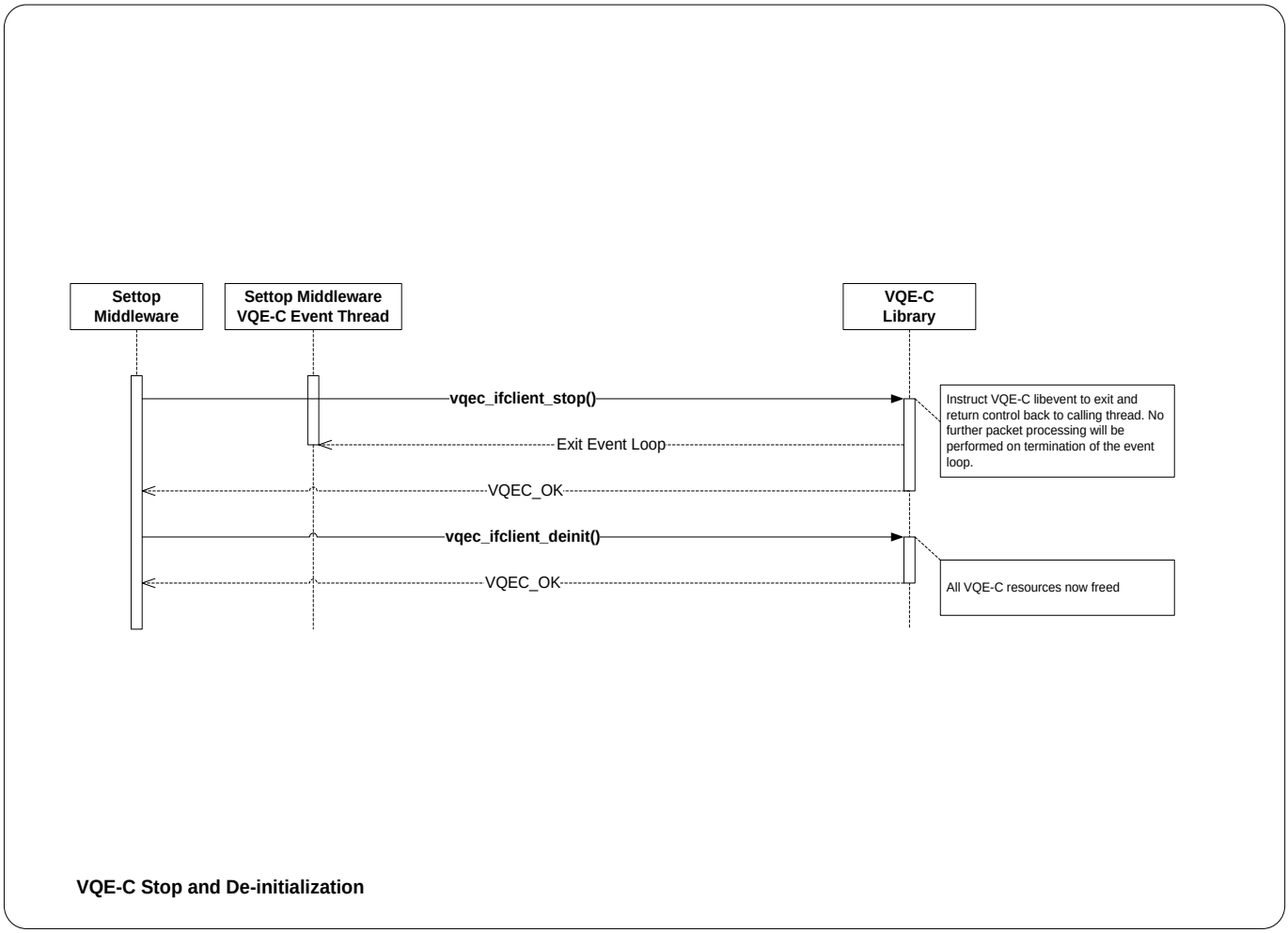
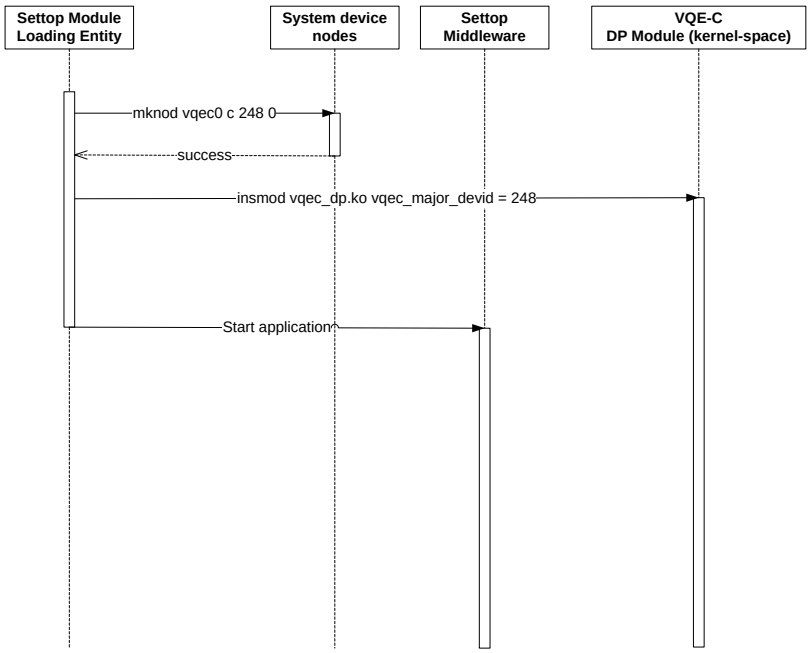


Figure 5 Stop and De-initialization

9.5 VQE-C Initialization for DP kernel ports

The diagram below shows a typical sequence involved in getting the VQE-C initialized for integrations that use the kernel port of the VQE-C DP prior to initialization of the VQE-C CP in the *user-space* (refer to 9.2):



VQE-C Initialization (DP kernel-space model)

10 Frequently Asked Questions

How is the code distributed?

VQE-C consists of a software development kit.
The code is distributed as source code to the customer or STB vendor.
The code resembles a standard Linux software component.

What OS and versions are supported?

Source code is distributed in the development kit.
The source code is currently supported only for the Linux operating system.

What hardware architectures are supported?

The code has been run on the following HW processor types:
MIPS, Intel
The code is designed to be portable other general purpose processor types are not expected to be an issue. The code is written entirely in 'C'. No assembly language.

What are the memory requirements for the VQEC-?

Memory requirements for a given integration of VQE-C are dependent on the stream rate and number of streams.
VQE-C can be run in some configurations using less than 3MB of memory.

Does the VQE-C run in user space or kernel space?

The VQE-C can be ported to either user-space or kernel-space.

What is the licensing model on the distributed code?

VQE-C source code is covered by the Cisco BSD license. The text for this license is contained in LICENSE.txt. In addition VQE-C has dependencies on several other 3rd party libraries with separate licenses that are bundled with the VQE-C distribution. See the <root>/3rd-party-src directory for bundled components.

Configuration parameters required to compile the code?

Configuration parameters required to compile the code are learned from a standard linux configure script. The configure script ships out of the box for Intel processors, but can be rebuilt for other architectures.

What are the options and the default settings of the system configuration parameters?

Note that system configuration settings have defaults. The VQE-C CLI command‘show system-configuration’ can be used to view the current settings of a running system.

What is level of integration needed for middleware? How is channel-change expected to be notified?

VQE-C's required integration with the middleware is very small.
VQE-C sits between the application and the Linux Socket layer for the video data sockets. VQE-C requires no more information than is required to open a multicast socket.

How to do scale testing?

For scale testing many instances of VQE-C can be run on a single processor. We have run 50-100 instances of VQE-C on a single multi-processor linux server.

Does VQE-C support UDP?

No VQE-C offers no services for UDP streams. It is currently illegal to describe a UDP stream to VQE-C.

Does my RTP source have to be standards compliant?

Yes. The VQE-C relies on information contained in the RTP header to be accurate.

11 Troubleshooting

11.1 General

1. **Bad source address for video stream**.
   All primary streams perform SSM filtering that is packets MUST come from the source address or they WILL be dropped.

2. **Excessive drop rate with high drop rate seen on input, higher than test stream rate.**
   Ensure that there is adequate buffering in pak_pool. Pakpool_size large enough.

3. **Some packets not repaired (output GAPs increasing)**
   a. validate that GAPS are being reported
   b. make sure repair packets are coming in late (check late packet counter… lookup exact names of parameters, show output)
   c. debug error repair enable

4. **No configuration received in CDI**
   a. turn on 'debug updater enable'
   b. turn on debug xxx'
   c. test connectivity to CDS server
   d. make sure VCDS is seeing requests

5. **System configuration changes ignored**
   a. validate settings via show system
   b. on mismatch look for reasons why invalid (parse errors not matched), show sys shows proper command names, comments watch out for multiline.

11.2 RCC Troubleshooting

This section provides some tips on potential troubleshooting areas related to RCC integration and testing.

1. **The "show system" indicates RCC is enabled, however "show tuner name <name> rcc" shows that RCC is disabled**

```
vqec> show tuner name 0 rcc
Tuner name:                0
--- RCC status ---
```

```
rcc enabled:              false
rcc result:               failure
cp failure reason:        RCC_DISABLED
dp failure reason:        RCC_DISABLED
```

a) Verify that RCC has not been disabled at run time, use "show rcc" to display the current state. Use "rcc enable" to re-enable if necessary.
b) Is RCC enabled for the specific channel, use "show channel <rtp://address:port> to verify. Update the channel SDP to enable RCC if needed.
c) Is the "do_rcc" parameter set to 1 during the requested to bind to channel using the "vqec_ifclient_Tuner_bind" API?

2. **RCC enabled, however all RCC attempts fail**
   a) Verify that the VQE-C and VQE-S have the same channel configuration
   b) Make sure the system has a route to the "feedback" target IP addresses for the RCC channels.

3. **Macroblocking seen intermittently on channel changes for RCC enabled channels**
   a) Verify that the VQE-C packet pool has been sized correctly, in conjunction with the VQE-S e-factor. Increasing the e-factor will reduce the demand on the VQE-C packet pool during RCCs.
   b) Verify that the actual experienced join time is less than the join time expected by the VQE-S. The join time can be obtained using the "show tuner name <tuner_name> rcc" command on the VQE-C CLI (see section 7.7.7.3

```
---Actual relative times (ms)---
CC       Pli      APP      Rep      Join     Prim     ER       Join-lat
0        7        11       33       149      204      2422     30
```

4. **The RCC enabled system appears to be display video out of sync (behind) when compared to a non-RCC system, even when they are tuned to the same channel.**
   This is normal; the RCC channel may be "behind" by up to a full GOP length of data.

5. **The VQE-C CLI indicates that the RCC was successful; however there seems to be no perceived difference in channel change time, even with large**

GOP size streams.

Ensure that "priming" information, see section **Error! Reference source not found.**, provided by the VQE-C is being used by the integration to configure the decoder path. VQE-C has the ability to replicate this information, please refer to the "*app_paks_per_rcc"* parameter in the VQE-C System Configuration Guide.

6. **The overall channel change time appears to vary with different streams.**

   The RCC eliminates the GOP and repair buffer delay for streams however does not have any effect on the VBV (Video Buffer Verifier) buffer delay of the stream. Establish if the dominant factor in the channel change time is related to the VBV buffering incorporated into the stream.

7. **Bad sequence range packets seen on VQE-C with high bitrate and low e-factor, as well as artifacts in the output video.**

   The VQE-C has a limitation where it cannot store and process more than 8,192 packets at a given time. When this upper limit is hit, the VQE-C will flush all packets in its buffer and reset itself, then continue on as if it had just joined the stream again.

12 References

The following documents are provided in the VQE-C tarball under eva/docs:

1. VQE-C System Integration Guided (EDCS-596270) – This document
2. VQE-C System Configuration Reference (EDCS-590664)
3. VQE-C CLI Command Reference (EDCS-590631)
4. VQE-C Release Notes

End of Document