SCTE CABLE-TEC EXPO 22
SEPTEMBER 19-22 • PHILADELPHIA, PA

2022 Fall Technical Forum
SCTE • CABLELABS • NCTA

Scaling a SCTE-224 Policy Decision System to Accommodate Burst Loads Driven by Marquee Events

A Technical Paper prepared for SCTE by

Madhuvanth Gopalan
Principal Engineer 1
Comcast India Engineering Center
Chennai, India
+91 9677189018
Madhuvanth_Gopalan@comcast.com

Timothy Wilson
Software Development Engineer 5
Comcast Technology Solutions
Chicago, IL
720-502-3789
Timothy_Wilson3@cable.comcast.com

Stuart Kurkowski, PhD, Comcast Technology Solutions

Table of Contents

Title **Page Number**

List of Figures

Title **Page Number**

List of Tables

Title **Page Number**

1. Introduction

Internet linear delivery has introduced the need for out-of-band schedule and entitlements information (SCTE 224) alongside event signaling (SCTE 35). The popularity and adoption of SCTE 224, Event Scheduling and Notification Interface (ESNI) is opening new use cases for which the protocol is a great fit. SCTE 224 has proven itself as an efficient and effective means for machine-to-machine communication of out-of-band linear rights management. This combination of SCTE 224 and SCTE 35 to trigger the in-band signaling allows precision execution of linear rights for content substitution and addressable advertising management. But the data maintained by the system in the SCTE 224 format is also useful for other entitlement use cases. The main use case we look at in this paper is around sports entitlement and using the SCTE 224 objects to determine whether a user has the rights to see a game or not – before even seeing the video – based on their location.

With its increase in popularity, content providers and operators are now delivering internet linear television to millions of subscribers simultaneously. As a result, SCTE 224 decision systems can receive millions of simultaneous decision requests from individual playback devices. Scaling these systems is paramount to effective delivery and subscriber satisfaction. One scaling strategy is the deployment of distributed decision systems as "serverless" functions. However, merely moving the decision logic to serverless functions did not achieve performant scale for loads expected for marquee events like major sports championships. Even though these functions exhibited acceptable elasticity in launch, the decision logic was not performant at runtime, because SCTE 224 objects do not lend themselves to expedient searches for applicable viewing policies. The Comcast Technology Solutions (CTS) team optimized the design by indexing the SCTE 224 objects and took advantage of other cloud features to create an optimal and performant distributed decision engine that scales in a ready state, making it instantly responsive across the video subscriber footprint. Our highly performant infrastructure decision system expands automatically to accommodate peak loads. As a result, this distributed decision system delivers 50 milliseconds responses under a load of nearly 20K requests per second. Furthermore, the system scales back under times of lower viewership, reducing our cloud processing cost.

2. Linear Rights Background

The ESNI protocol is perfectly suited to communicate rules and policies at an audience-based level, thereby providing a substrate to implement linear rights. ESNI is the key mechanism to communicate linear rights around web or over-the-top (OTT) embargoes, regional blackouts, and even dynamic advertising. ESNI also provides policies for specific rules at the precise audience level. ESNI therefore facilitates appropriate rights decisioning at scale from machine-to-machine. The ESNI objects are perfect for determining such things as channels and events a user has the rights to watch in real-time.

ESNI objects are XML (Extensible Markup Language) messages with relevant fields for rights such as ViewingPolicy actions for content switching, blackouts, and playout restrictions. These ESNI messages are managed with a REST (representational state transfer) interface for exchange between the providers and the distributors. The out-of-band ESNI execution components then link these markers with the ESNI instructions. The hierarchy of ESNI Policy to ViewingPolicy and to Audiences allows support of various ways to evaluate linear rights against these audiences.

3. SCTE 224 Context

SCTE 224, Event Scheduling and Notification Interface (ESNI) is an XML based standard that provides a defined protocol for carrying machine-to-machine metadata for video. There are five basic constructs within SCTE 224, as shown in Figure 1. These constructs work together to provide a programmer with a

means to convey video rights for content replacement as well as advertising instructions on the distributor or operator side of the workflow. These five constructs are Media, MediaPoints, Policy, ViewingPolicy, and Audiences. We describe each of these here, and then tie them all together with the various use cases around sports entitlement.

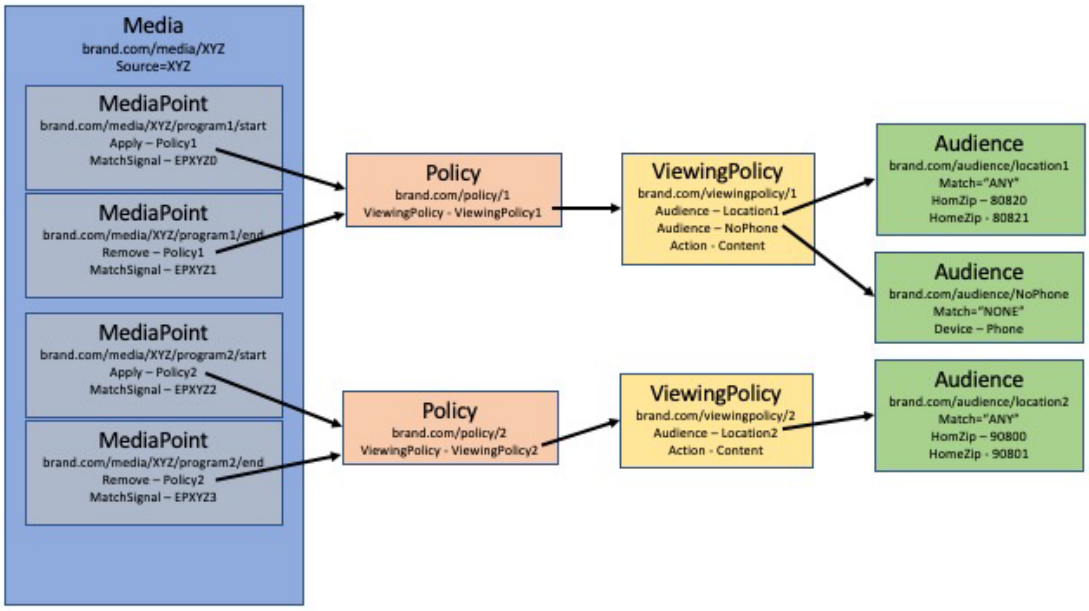

Figure 1 – SCTE 224 Constructs

3.1. SCTE 224 Media

The Media object is a top-level container representing a linear channel whose primary function is carrying all the MediaPoints, so it contains an ordered list of MediaPoint elements as shown in Figure 1. The Media also contains a few key elements like a description and a source for the linear channel it represents.

3.2. SCTE 224 MediaPoint

The MediaPoint object describes a point in the Media when a decision needs to be made or an action needs to be taken. These points can either be time-based (i.e., the presence of a @matchTime attribute in the MediaPoint) or an SCTE 35 in-band signal based for frame accuracy. The signal-based MediaPoints contain a MatchSignal element with XPath matching logic to link the MediaPoint to the presence of the in-band signal. Signals can be reused, because MediaPoints also have an effective/expires window constraining when the MediaPoint can be evaluated.

When a MediaPoint is triggered, based on time or signal, it can either "Apply" or "Remove" one or more Policy objects which affect the state of the linear playout. See the Policy object description below for more details. MediaPoints that "Apply" a Policy do so until another MediaPoint explicitly "Remove" that Policy or they time out based on the duration indicated in the "Apply" statement.

3.3. SCTE 224 Policy

A Policy object is nothing more than a container for defining a set of ViewingPolicy elements that should be acted upon based on this Policy being "Apply" or "Removed" from the Policy stack. The "Apply" or application of a Policy means putting that Policy on that Medias stack via first-in-last-out queue, so multiple Policy objects can be affecting the state at one time. The removal of a Policy then takes it off that stack and out of the state of that Media. SCTE 224 has explicit rules about how to manage the Policy queue in a SCTE 224 execution engine.

3.4. SCTE 224 ViewingPolicy

The ViewingPolicy object is the key SCTE 224 object that associates one or more actions to an audience. These "Actions" can range from telling an audience to go to alternate content, restrict trick mode, or restrict resolution. For advertising specifically, these actions can contain information about the advertising decisioning service (ADS) to use for a particular audience, or various advertisement conflicting rules for a particular audience. The key to a ViewingPolicy is that if the "Audience" criteria is met, then the action must be taken. The SCTE 224 specification maintains a large list of actions allowed within the ViewingPolicy object, many of which are specific to addressable advertising.

3.5. SCTE 224 Audience

The Audience object is a set of characteristics that define a subset of viewers based on criteria such as device-based characteristics (tablet, phone, etc.), general characteristics (local storage, mobile, etc.), or location-based characteristics such as zip codes, postal codes, latitude/longitude, market areas, as well as roles such as distributor or Virtual Integrated Receiver Decoder (vIRD) identifiers that place the audience into groups to dictate specific actions. Audience objects can contain other Audience objects making them a compound Audience. Additionally, logic to associate a client with a viewer or viewers is based on matches of ANY, ALL, or NONE of the characteristics outlined in the Audience, so that characteristics can easily be included or excluded. For example, you can say Match="ANY" for a list of zip codes to characterize the audience within that area or use Match="NONE" to characterize an audience outside that area.

4. SCTE 224 Example Logic

So now let's take those five objects from Figure 1 and run through a scenario of a real-time decision request for the video playout. The trigger is a video signal acquisition system (SAS) seeing an in-band SCTE 35 signal or a user logging into an app. An events triggers the request to a signal decisioning system (SDS) to figure out what it should do. When the SAS calls the SDS it tells the SDS which source it was on, what time it saw the signal, the binary signal, and the client characteristics. Something like "I just saw the signal 'UhJeasfojoihe23edde' on source XYZ, at 1:00pm and I am encoding for zip code 80820." Or if it is an app user, they are requesting a determination of entitlement to see if the game they are requesting is or is not on the feed they are requesting it on. The SDS then looks through all the out-of-band SCTE 224 and its Media to find the one for that source (i.e., XYZ). Once it finds the correct Media it looks through its MediaPoints to find the list of MediaPoints that at that time falls within their effectiveness window. Once it has the list of MediaPoints it evaluates each one to see if there is a match with the signal. For the MediaPoint that matches, the decision is either to "Apply" or "Remove" the associated Policy. It then goes from that Policy to the ViewingPolicy where it gets the Audience and sees if the client's zip code is in or out of that Audience or not based on the Match criteria. If it is in that Audience, the SDS returns the "Action" of the ViewingPolicy to the SAS. In the case of alternate content, it might return to the SAS that it needs to switch over to another source and start encoding the alternate

source. Based on the client information and the construction of the Audience objects, API logic in this use case can give an entitlement answer to a requester.

5. SCTE 224 Indexing

As mentioned in the Introduction, SCTE 224 objects do not lend themselves to expedient searches. To provide a performant service, the SCTE 224 objects need to be transformed into some other form that does support expedient searches. Looking at the types of requests to be processed, there was no single type of transformation sufficient to provide the required performance. Thus, we needed multiple indexing schemes, with each type of request using an indexing scheme that best suits it for the SCTE 224 object. *This is a key concept, because even at the expense of potentially storing the same data in multiple structures, the executional performance outweighed the duplication.*

5.1. Scheduling

For linear playout channels, an SCTE 224 Media object typically represents a schedule for that channel, with MediaPoint objects representing individual shows or events. The SCTE 224 specification also allows for metadata about these shows to be contained within the MediaPoint object. Taken together, these two features allow a scheduling interface to determine what is currently playing and to gather metadata on a particular show or event.

The SCTE 224 specification does not make any comments about how MediaPoint objects are stored within a Media object, just how the document order of these MediaPoint objects dictate their precedence. As the MediaPoint objects are stored as a list within the Media object, doing a linear search to determine what is currently playing would take too long on average. For this system under discussion in this paper, the MediaPoints were set up in a true schedule fashion, with earlier events closer to the beginning of the Media object and later events farther down in the document. Additionally, the shows were set up to eliminate any overlap. These restrictions on the MediaPoint objects lead nicely to using a form of self-balancing binary search tree, such as an AVL tree, using time as the search criterion.

Searching for the metadata on a specific show would not lend itself well to using the same binary search tree described above. Therefore, a different form of indexing needs to be applied for this type of request. With the caller supplying the identifier for the show, a hash map object mapping the identifier to a MediaPoint would provide the desired performance.

5.2. Entitlement

Once the information for what is playing and the metadata has been gathered, now it's time to determine if the user is entitled to view the show or event. As with the request for gathering the metadata, the system is provided with the identifier for the show, lending this type of request to rely on an underlying map structure to pinpoint the MediaPoint object. However, unlike the metadata request, these entitlement requests are required to follow the chains of Policy, ViewingPolicy, and Audience objects to make the appropriate decision. Generally, both the Policy and ViewingPolicy objects are succinct, allowing them to be stored as simple data objects. The Audience objects, on the other hand, could contain a list of thousands of zip codes, calling for a more complex structure such as a hash map.

The flow for satisfying an entitlement request would look something like this:

1. Using the show identifier provided in the request, locate the desired MediaPoint object.
2. For each Policy object listed in the Apply elements of the MediaPoint object:

a. Locate the named Policy object, if it is only supplied as a reference in the MediaPoint object
b. For any ViewingPolicy associated with this Policy:
    i. Locate the named ViewingPolicy, if it is only supplied as a reference in the Policy object.
    ii. For each Audience associated with the ViewingPolicy object, use the zip code provided in the request to look for a match.
        1. If a match is found, use the ViewingPolicy to determine the action for a member of this Audience.
        2. If no match is found, continue searching until all Policy, ViewingPolicy, and Audience objects for the MediaPoint have been examined.
        3. If there is a match in any other Policy, it is handled like step #1 above.
        4. If there are still no matches, then nothing is found, and a default "Not Entitled" response is given.

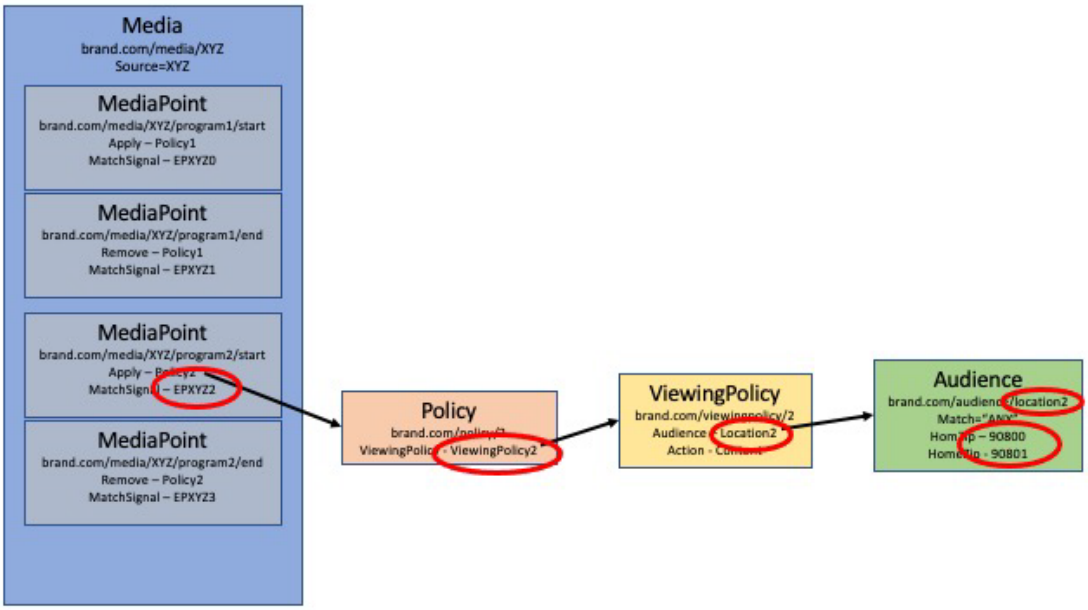

Figure 2 – Entitlement Logical Decision Flow

5.3. Pre-Conversion and Caching

As serverless functions used to supply this functionality are expected to be ephemeral, converting the raw SCTE 224 to structures optimized for SCTE 224, such as AVL or hash maps as described above upon start up excessively impacts the performance of the initial requests. The system was designed to do these conversions externally to the serverless functions providing the searching capabilities. When any update is received by the system, a processor generates the various structures that form the optimized SCTE with the updated data and stores it in a cache that is shared across all instances of the request-handling serverless functions.

6. Concurrency Scaling

Concurrency scaling is what controls the number of concurrent serverless functions that can be set up to handle all of the calls in a scaled event. As the number of connections reaches the limit, throttling of requests can occur during requests that are held until they can be serviced.

6.1. Rate-Limiting

The design choices explained in the indexing section of this paper laid the foundation of this highly performant system and during various load tests that were performed, the system scaled reasonably well until it hit a certain burst load, and requests were being limited.

Most serverless functions are not infinitely scalable and therefore have a configured concurrency limit. There are two types of concurrency limits with serverless functions.

- There is a limit on the number of function containers that can serve requests at a given time. In most cases this value is configurable.

- There is also a burst concurrency limit which is the number of new function containers that can be spun up at a given point in time. This is set at 3,000 in the area these systems were operating in.

In each of the cases above, if either limit is reached during execution, any more requests that need additional serverless functions would not be served. This is reflected during execution as limiting those requests so that available resources can serve the current requests.

The metrics graph below shows, as many as ~16k requests @~11.47AM were throttled, because we hit the burst concurrency limit of 3,000 instances.

Figure 3 – Requests Per Second

6.2. Solution Options

Because the burst levels of requests for these extreme events requests were running up against the limiting, we needed a way to allow the design to work. The goal then was to raise this concurrency limit in some fashion, but there are constraints on the limit, so we looked at two options:

- Increasing the concurrency limit of serverless compute serve. But our functions were being limited because they hit the burst concurrency limit, which is non-configurable.

- Take a closer look at behavior of the functions themselves and identify opportunities for optimization.

It was not enough to just throw more resources at it, because there were cost prohibitions for the majority amount of time when the system is not under load. Since the first option is not configurable, our course of action was to look at the second option around function behavior.

6.3. High Response Time Requests

We decided to take a closer look at:

- What are some of our slowest requests?
- Which part of the code are they spending significant time on?

We already had built some custom instrumentation in our functions, which were basically measuring time taken by different sections of the code and logging them in a structured format to analyze. Querying those logs using analysis tools gave us insights allowing us to answer the two questions above.

#	total_time_ms	db_conn_ms	execution_time_Fin…	execution_time_TreeBySta…
▶ 1	4127.7085	2005.5475	0.00173	33.6695
▶ 2	4100.0024	2002.9073	0.007616	43.5819
▶ 3	4089.9966	2003.8354	0.00227	51.5729
▶ 4	4088.0203	4002.5376	0.002546	64.7034
▶ 5	4085.9028	4003.8772	0.002401	40.0637
▶ 6	4084.3572	4003.59	0.00249	47.2985
▶ 7	4077.8228	4002.887	0.001731	31.4863
▶ 8	4077.6042	4002.8762	0.002939	34.8527
▶ 9	4076.252	3.3288	0.002005	38.6671
▶ 10	4075.7478	4002.984	0.002004	71.1681
▶ 11	4074.7073	4003.7002	0.002398	33.3264
▶ 12	4074.5022	4003.2598	0.0018	30.7119
▶ 13	4071.6487	4002.935	0.001934	36.5997
▶ 14	4071.4495	2002.521	0.002003	37.9416
▶ 15	4071.1333	4002.9124	0.002255	36.8048
▶ 16	4070.7996	2003.0397	0.001996	37.8136
▶ 17	4070.7488	4003.3252	0.00223	42.2515
▶ 18	4069.8464	4002.905	0.002119	38.5778
▶ 19	4069.3198	3.2697	0.005251	36.5757
▶ 20	4068.2205	4003.4956	0.001871	37.597
▶ 21	4067.6438	4003.4858	0.002094	30.18
▶ 22	4067.1667	2002.8424	0.007691	31.651

Table 1 Response Times

It allowed us to see a linear correlation between high processing times and:

i. Time taken to forward logs to a centralized logging system of choice. Yes, we were using two logging systems; one allowed us to correlate log-events from this system with related log-events from other systems that weren't using central logging tools.
ii. Time taken to establish connections to the database. A burst of requests meant many different serverless function containers were requesting "new" connections at the same time causing congestion on the server side.

We replaced logging synchronously to the second logging system with logging asyncrhonously. All the events were logged to the central logging tool, then forwarded to the second using another independent

serverless function.

To address (ii), we implemented a connection pooling mechanism and initialized connections to a database in the inital phase of serverless function, rather than on-demand.

6.4. Serverless Function Warm-up

With all those performance advances described above, there was a significant improvement in response times. However, given that the expected scaling for an event like the Super Bowl will be anything but linear (i.e., tens of thousands of users logging in to watch the broadcast just before the game starts), we knew we might still hit the non-configurable *burst limit* leading to requests being throttled. But an advantage is that it's easy to predict when to expect the burst (before game start).

One aspect of the request-processing life cycle that wasn't in our control, but was consuming significant amounts of time, was the server-less function container initialization time. In other words, the time required to perform a cold start of a new serverless function.

To avoid being limited by cold starts so that bursts could be gracefully handled, we leveraged the provisioned concurrency feature of the server-less function. This instructed the server-less function to spin up X number of serverless function containers, ready to handle the burst of requests as they come.

All of the sound design choices, optimizations for less-efficient parts of the code, and enabling provisioned concurrency allowed us to achieve a P95 response times of <50 milliseconds, when the system was under a peak load of 16-18k requests per second.

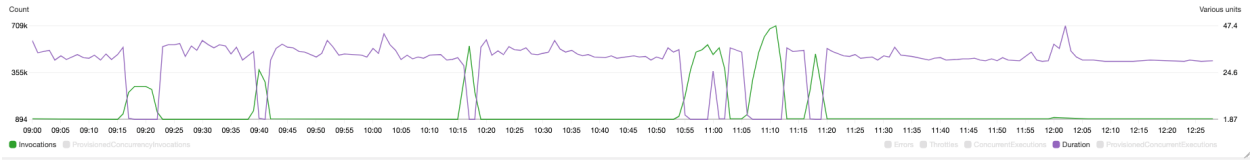

Figure 4 – Peak Loads

7. Final Architecture

So based on the information and the success of the above optimization, here is a view of an example overall architecture for an optimized solution.

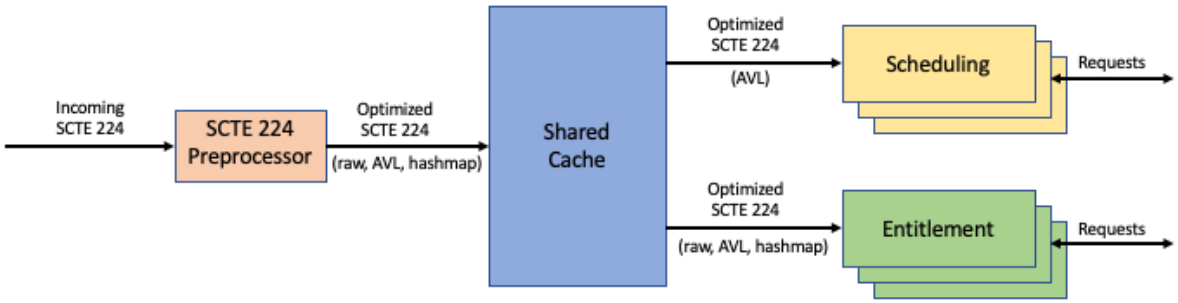

Figure 5 – Architecture View

8. Conclusion

With its increase in popularity, content providers and operators are now delivering linear television over the internet to millions of subscribers simultaneously. As a result, SCTE 224 decision systems can receive millions of simultaneous decision requests from individual playback devices. Scaling these systems is paramount to effective delivery and subscriber satisfaction. One scaling strategy is the deployment of distributed decision systems as "serverless" functions. However, for us, merely moving the decision logic to serverless functions did not achieve performant scale for loads expected for marquee events like the major sports championships. Even though these functions exhibited acceptable elasticity in launch, the decision logic was not performant at runtime, because SCTE 224 objects do not lend themselves to expedient searches for applicable viewing policies. We optimized the design by indexing the SCTE 224 objects in multiple methods and took advantage of other cloud features such as scaling concurrency and warming up serverless functions before use, to create an optimal and performant distributed decision engine that scales in a ready state, making it instantly responsive across the video subscriber footprint. Our highly performant infrastructure decision system expands automatically to accommodate peak loads. As a result, this distributed decision system delivers 50 milliseconds responses under a load of nearly 20K requests per second. Furthermore, the system scales back under times of lower viewership, reducing our cloud processing costs.

Abbreviations

ADS	ad decisioning service
API	application programming interface
AVL	Adelson-Velskii and Landis binary search tree
ESNI	Event Scheduling and Notification Interface
IRD	integrated receiver decoder
REST	representational state transfer
SAS	signal acquisition system
SCTE	Society of Cable Telecommunications Engineers
SDS	signal decisioning system
vIRD	virtual integrated receiver decoder
XML	Extensible Markup Language

Bibliography & References

SCTE 35, Digital Program Insertion Curing Message for Cable, 2020. https://scte-cms-resource-storage.s3.amazonaws.com/ANSI_SCTE-35-2020-1619708851007.pdf

SCTE 224, ESNI, Event Scheduling and Notification Interface 2021. https://scte-cms-resource-storage.s3.amazonaws.com/SCTE-224-2021-1620314764331.pdf

W3C XML Base (Second Edition). W3C Recommendation 28 January 2009. http://www.w3.org/TR/xmlbase/

W3C XML Schema Part 2: Datatypes Second Edition. W3C Recommendation 28 October 2004. http://www.w3.org/TR/xmlschema-2/

W3C XML Path Language (XPath) 2.0 (Second Edition). W3C Recommendation 14 December 2010. http://www.w3.org/TR/xpath20/

AVL Tree: https://en.wikipedia.org/wiki/AVL_tree

