Video Management White Paper

1 Introduction

Video streaming, broadcasting, storage and associated user facing technologies are very mature as exhibited by Netflix and YouTube. These organizations operate from the same cloud-based infrastructure that is accessible at the swipe of a credit card. The challenges faced by an enterprise wide video surveillance system are similar to those faced by the likes of YouTube with some additional needs:

- Enterprise video surveillance does not have to manage massive scale broadcast. This means that the video storage and playback formats and protocols, mass storage services, authorization and authentication and network backbones are all in place. There is little need for content delivery networks in this application.
- Adaptive bandwidth streams are probably not needed. This means that the bulk of the video can be streamed directly from camera to mass storage with a computationally cheap repackaging of the video stream rather than a full recording of the stream for different resolutions.
- The specifications are aimed at serving organizations that operate over multiple sites as well as single site operations that may have limited or unreliable connectivity.
- The system will have to manage individual cameras which includes registration, configuration and commands such as PTZ.
- It must have a pluggable architecture to allow advanced machine learning and artificial intelligence algorithms to be applied to video streams.

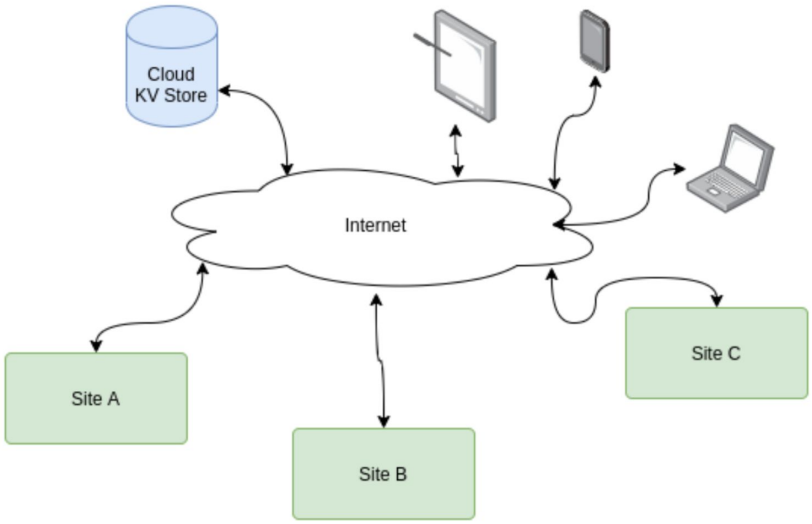
These additional requirements can be implemented with careful design. There are some caveats with the approach taken in this design. Video surveillance requires quite low latencies (less than 10 seconds) which is achievable using the standards envisaged but present some risks that should be evaluated early in the project.

The overall design philosophy used here is one of a fully distributed application and “use what is out there”. From this comes the focus on the standards used by the likes of YouTube with modern product packaging and the distributed API driven architecture inspired by applications such as git. Being API driven means that the application can be accessed from client applications (mobile, tablet, browser, smart TV) as well as by machine to machine services using a published API.

2 Design overview

The high-level view of this architecture is a distributed application that functions as a syndicated network of peers (equals). There is no concept of a central server or centralized storage which means a user at site A can manage and view video at site B if they are suitably authorized. It also means that the storage used can be anything from a local hard drive to a cloud based bulk key-value store such as Azure Blob Store or AWS S3. A camera at site A

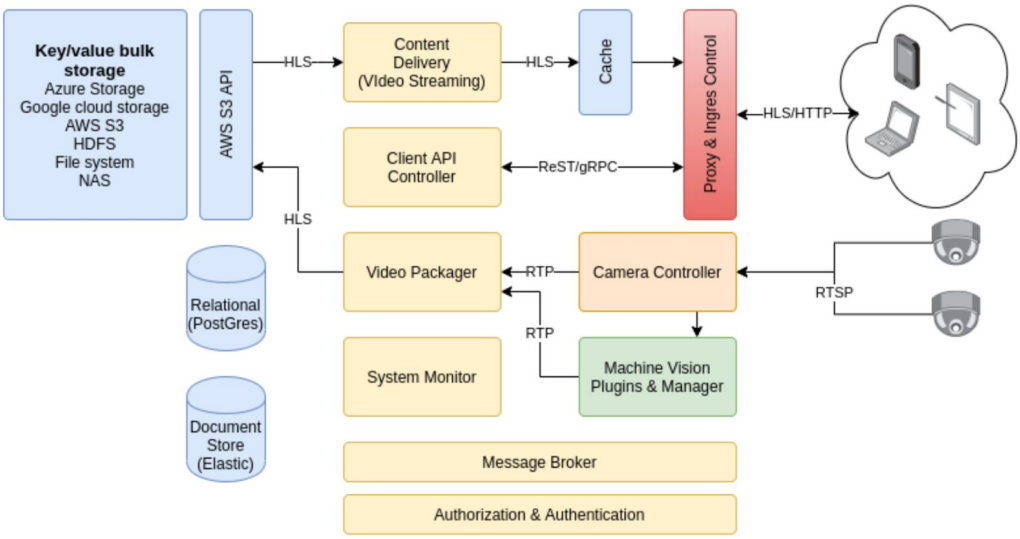
could be set up to send its stream to a storage device at site B. Clients (users) would access the system using an application that runs on all modern devices (tablets, smart phones, desktops and even smart TVs) in much the same way as applications like Netflix functions. They would authenticate to some service such as the corporate Active Directory or some cloud-based service such as Amazon's Cognito. These credentials would be issued with a set of scopes and capabilities which will provide fine grained access control over sites, individual camera feeds and recordings as well as a multi-tiered administrator hierarchy that would grant suitable users capabilities such as adding new users. Note that the only centralised component is the authorization service.
Finally, it also means that the same application can be deployed to a cloud vendor or to on-premises hardware as needed by the deployment.



In keeping with the overall philosophy, Internet security will be managed by standard best practices including secure socket layers, virtual private networks, firewalling and distributed authentication that is available as a commodity product.
The sites referenced in the diagram above are entities such as factories, office blocks or other properties that require surveillance. Obviously, the cameras are on-site as is any camera related hardware and services but the management and storage services could be on-site or cloud based.

3 Video Management Service

Each site will have access to a fully-fledged video management service. This entity will enable all the capabilities associated with the service including camera management, video storage, streaming to applications and a pluggable machine vision architecture as illustrated in the figure below. In the following sections, we elaborate on each of the major components in the system and will illustrate how these get applied as well as look at the state of the art for video streaming technology and associated user facing tools



4 Video streaming

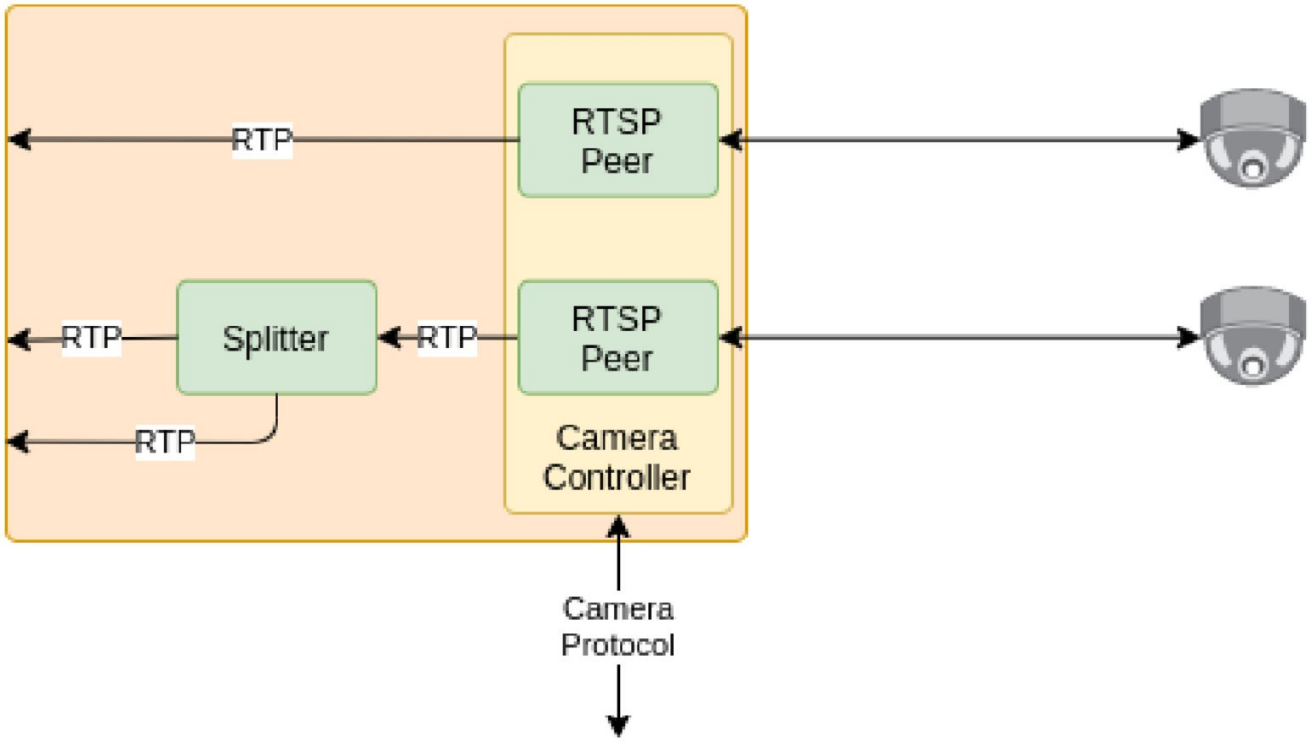
Modern commercial video streaming systems are able to accept video in a limited number of formats and an almost countless number of devices. These formats include RTSP/RTP and RTMP. The video source is configured to send its stream to a target URL which then trans-multiplex and transcode for scrubbing and at multiple bitrates into a packaging format suitable for scrubbing and at multiple bitrates (e.g. HLS and DASH) to allow for adaptive streaming. These streams can easily be consumed by mobile and desktop applications alike. Formats such as HLS are really just video streams broken up into chunks that always start with a key frame (AKA a GOP or group of pictures) which represent 2-10s of video. The client simply pulls the next chunk as it needs it using the HTTP protocol and displays it for the viewer. Key components here are the repackaging and possible recording of the video source, the bulk storage until and the content delivery mechanism (effectively a web server with a distributed cache). We follow the video path from the camera to the client in the following sections.

5 Video upload

Video is streamed from IP capable cameras using a format called RTSP which is really just a control protocol. The actual video and/or audio is streamed using RTP. RTP is usually transmitted over UDP so it is inherently unreliable but has very low latency. We envision that streams from individual cameras will be required by multiple consumers. These are then trans-multiplexed which maps the RTP over to HLS and any machine vision plugins that have been connected to the particular camera. The subsystem for managing this is called the camera controller and has a ReST/gRPC interface to drive camera registration and set up the RTP stream splitters and destinations. It will store its configurations on a database that is either local or cloud based as per its configuration. This will be a custom application which could repurpose software already developed.

6 Camera controller

The camera controller will manage the camera registration, streaming destinations and camera controls such as PTZ and general settings. The control will be via a ReST API that will map requests to RTSP for the camera control. The controller will also route the RTP to one or more specified URLs for repackaging, machine vision or direct viewing.



7 Trans-multiplexer/transcoder

This application simply takes an RTP feed and repackages it as an HLS stream and writes this stream to the key/value bulk storage system. The trans-multiplexer function (a.k.a. transmuxing) can be achieved by something like ffmpeg or one of the many commercial video streaming applications out there (e.g. Nimble, Wowza or the cloud offerings from Azure or Amazon). Repackaging is a low-cost operation as no actual video is processed. Whatever is sent by the camera is simply reassembled into video frames and those frames are then packaged as HLS. However, recording (a.k.a. transcoding) can be very expensive. This step is required for streams which need to be resized or if the stream's key frames are too far apart or not equally spaced or if a different codec is required in the destination stream. It requires a full decode of the stream (relatively cheap) followed by a full encoding into the new stream which is computationally very expensive. The big cloud vendors all offer recording

as a service (normally as a prelude to live streaming at multiple bit rate streams). In this application, we believe simple repackaging is sufficient.

8 Distributed storage

The distributed storage system is a key/value store and effectively the network video recorder (NVR) for the system. It will use a protocol such as Amazon's S3 protocol (it is widely used for KV stores) to write HLS video streams. By using an appropriate application such as MinIO, the store can be made robust against failures and allow transparent streaming directly to cloud storage offerings such as Azure or AWS.

9 Client API controller

The client API is the access point through which all external applications access the service. The goal is a purely externally accessible API which serves all the user applications, manages the video streams and is also available for machine to machine applications. It will also be used to allow peer sites to communicate with each other. It will be a ReST interface and will contain all the business logic including role and scope management.

10 User application

The complete user experience happens on the user application which will run on multiple platforms including tablets, mobile phones, laptops, browsers and even smart TVs. The goal is a single code base for the UI which will be packaged and deployed using technologies such as Angular, React or Flutter. The functionality available will be dependent on the role and scopes assigned to the authenticated user of the application. It will therefore be the place where people watch video streams, configure the system and monitor system events such as intrusions etc

11 Support systems

The support systems underly all the functional components noted above. These systems include databases (one relational, one a document store), a message broker to manage events and stream logging messages. It will also have a dashboard component from which the system can be monitored.

We recommend that the entire site be deployed using containers and some orchestration system such as Kubernetes. This will allow deployments from something as small as a laptop computer through a fault tolerant deployment over a server farm to deployment to any one of the major cloud vendors. The overall development should be based on modern best practices for testing and continuous integration and deployment. Seamless flows from source code through a staging site to production are common practice and eliminate risky upgrade procedures.