Improving Video Ingestion and Serving in Cloud-Based Home Security Systems: Enhancing Cost and Latency Reduction

Sibin Thomas

Tech Lead
sibin_thomas15@hotmail.com

Abstract
In this study, the important parts of managing video data in cloud-based home security systems are looked at in depth. We look at how video is usually loaded and served, focusing on the roles of video compression, fragmentation, and adaptable streaming [1]. We list the main problems with this design, such as the time it takes to check permissions and the extra work that comes with looking up metadata. We suggest ways to improve things by using time-limited access keys, hash-based validation, and encoding URLs for video fragments in manifests [2]. These changes make things run more smoothly, cut down on latency, and keep costs low for service companies. We also talk about how these strategies can be used in a wider range of video-based applications besides home security, focusing on how they can improve speed and the user experience in today's constantly changing digital world.

Keywords: Smart home security, video streaming, media processing, cloud architecture, regionalization, edge computing, caching, latency optimization

INTRODUCTION

With so many home security cameras that are linked to the cloud, we are now creating more video data than ever before. Keeping track of, sending, and getting back all this extra info is proving to be very difficult. This essay goes into detail about the important parts of managing video data in cloud-based home security systems. It focuses on the details of video input, serving, and the building blocks that make these things possible.
To start, we'll give you a full rundown of how video is usually received and sent in a home security camera system. This includes looking at how video compression, fragmentation, and adaptable streaming can be used to make sure that data is handled efficiently [3]. Then, we look at the main problems with this design, mainly the extra time and work that is needed for multiple authorizations and metadata database searches.
To deal with these problems, we look into possible ways to make things better, such as using time-limited access tokens, hash-based validation, and putting real video fragment URLs in the manifest [4]. The goal of these improvements is to lower latency, cut costs, and make the system work better overall.
In addition, we talk about more than just home security cameras. We look at how these optimization methods can be used in other video-based applications that have similar problems. This includes looking at how they might be useful in areas like medical imaging, videoconferencing, cloud gaming, and online learning tools.
The point of this paper is to explain in detail the difficulties and difficulties that come with managing video data in cloud-based home security systems. It will also give useful tips for improving these systems and show how they could be used in a wider range of situations as technology changes.

BACKGROUND OF VIDEO INGESTION AND SERVING IN CLOUD-BASED HOME SECURITY SYSTEMS

This section provides context for a typical workflow of video ingestion and serving in a cloud-based home security camera system, providing a foundational understanding for subsequent discussions on challenges and optimizations.

Video Ingestion

The video processing part of the camera is the first part of the video ingestion system. Taking raw video and audio data straight from the camera sensor is what this part does. This raw data is encoded into a compressed file, usually H.264 (Advanced Video Coding), so that it can be stored and sent more efficiently [5]. A lot of people use H.264 to compress videos because it strikes a good mix between how well it compresses and how hard it is to program. It compresses video data by using spatial and temporal redundancy to cut down on the amount of data needed to show the video without lowering the quality of the image.

After the video has been compressed, it is cut up into smaller pieces, usually just a few seconds long. This method to fragmentation has a number of benefits, including [6]:

Resilience to Network Disruptions: Breaking the video up into smaller pieces makes sure that if the network goes down, only the current piece being sent is lost. This keeps data loss to a minimum and lets the camera start uploading again as soon as the link is back up and running.

Efficient Bandwidth Utilization: Because each section can be uploaded separately, fragmentation makes better use of bandwidth. In this way, the camera can adjust to different network conditions and send important parts first, like those with motion or interesting events.

Simplified Cloud Storage: Cloud storage systems are better at handling smaller files, which makes uploading broken-up videos easier and lowers the amount of space that is needed.

An upload request is sent by the camera to the cloud or "device frontend" when a piece of video is ready to be sent and there is a network link to the cloud. After that, the device frontend does the following:

Authorization: To keep things safe and stop people from getting in without permission, the front end of the gadget first authorizes each piece coming from the camera [7]. This authorization process checks that the camera is who it says it is and that it has the right powers to send data to the cloud. Usually, this step is given to a different component called a "authorization service," which is in charge of verifying devices' identities and controlling access.

Region Selection: Once authentication is complete, the device frontend uses predefined reasoning to find the best storage area for the video fragment. This choice is based on things like the user's position, the server's load, and the need for data redundancy. Putting data closer to the user can speed up retrieval and lower latency. Spreading data across different regions can make it more resilient and redundant.

Storage: Finally, the front end of the device saves the video fragment in the right place in an object storage database [8]. You can store a lot of unstructured data, like video fragments, in object storage, which is scalable and cheap. The URL or link to the stored fragment in the region-specific object storage is kept in a separate metadata database to make it easier to find and control. This metadata library is like an index; it lets the system quickly find and get specific parts of videos when they are needed.

Video Serving

When a user wants to view a video recorded by their home security camera, they typically use a mobile app provided by the service provider. The user selects the desired camera and specifies the duration of the video they wish to view. The app then sends a request to the "cloud frontend," requesting all video fragments within the user-selected duration.

The cloud frontend handles this request by performing the following steps:

Authorization: The cloud frontend makes the permission of the user's request go to the authorization service, which is similar to how video is loaded. This makes sure that only people who are allowed to can get to the video footage and watch it.

Manifest Generation: Once the user's request is approved, the cloud server makes a manifest file. Depending on the user's device and network conditions, this file is usually made using either HTTP Live Streaming (HLS) or Dynamic Adaptive Streaming over HTTP (DASH) [9]. The manifest file is basically a playlist with a list of URLs or links to the video clips that make up the desired length.
It is important to keep in mind that the manifest links do not go straight to the video snippets in the object storage. Instead, they lead to records in the database of metadata, which then have links to the video fragments. By using this indirection, you can change and manage where videos are stored without having to change the manifest itself.
The cloud frontend makes sure that the video fragment period links or URLs in the manifest are set up in the right order so that the video plays smoothly when the user watches it on the app.
Manifest Delivery: The user's app gets the manifest that was made. The app then uses this manifest to ask for the download of certain video snippets based on how the user interacts with and watches videos. For instance, if the user wants to find a certain part of the video, the app will ask the manifest for the fragment that goes with it.
The real requests to download video fragments are sent to the "media frontend." These things are done by the media frontend:
Authorization: The media frontend also gives the authorization service control over the app's request [10]. This makes sure that the video parts can only be downloaded by apps that are allowed to.
Metadata Lookup: Once permission is approved, the media frontend uses the video fragment period link from the manifest to search the metadata database for the right link to the video fragment in the object storage.
Retrieved Video Fragment: The media frontend gets the video fragment from the object store and sends it to the app.
Lastly, the app's built-in video player renders the downloaded video snippets so that the user can watch them without any problems. The app usually downloads and stores a few fragments ahead of where the current playback point is. This way, even if there are temporary problems with the network, playback will still go smoothly.
With its focus on efficient video compression, fragmented ingestion, and adaptive streaming, this design makes sure that video uploads are strong, storage is managed in a flexible way, and video delivery is optimized for a smooth user experience. It also lays the groundwork for adding more improvements and advanced features, like AI-powered video analysis and smart video uploading strategies, which will be talked about in later parts of this paper.

CHALLENGES IN CURRENT CLOUD-BASED VIDEO INGESTION AND SERVING ARCHITECTURES FOR HOME SECURITY

While the architecture described in the previous section provides a functional framework for video ingestion and serving in home security systems, it presents certain challenges that can impact efficiency, cost, and user experience. This section delves into these challenges, providing a detailed analysis and setting the stage for potential optimizations.

Multiple Authorizations and Associated Latency and Cost

One big problem with the design described is that different people need to be authorized at different points in the video serving process. Authorization checks happen both when the manifest is first requested and, more importantly, when each video clip is downloaded [11]. This multiple-level authorization provides strong security by stopping unauthorized access to video data, but it adds delay and computational load.
For every authorization request, the media frontend and the authorization service have to talk back and forth. This adds delay, especially if the authorization service needs to get authorization data from a database or an outside service or do complicated authorization logic. This latency can add up if a user asks multiple video

fragments at once, which could slow down video playback and make the user experience worse, especially in situations where time is of the essence, like in home security.

Furthermore, the authorization service has to pay for handling each request and doing database searches, which adds to the overall cost of the service. When these costs are spread out over millions of devices and people, they can go up quickly, which could make the service less cost-effective overall.

This various levels of authorization are based on two main needs:

Ensuring User Authorization: Only authorized users will be able to view the video data if their access permissions change between the manifest request and the fragment download request. This is because user authorization is checked at both the manifest and fragment levels.

Maintaining Data Integrity: During fragment download, an extra authorization check makes sure that the user can still access the requested video fragment, even if the fragment's status or access rights have changed since the manifest was made. This could happen if the user's subscription plan deletes videos or if another approved user deletes the video fragment.

It is important to think about security and data stability, but the extra time and computing power that comes with multiple authorizations mean that we need to look into ways to make the service more efficient and improve the user experience.

Database Lookup for Each Video Fragment Download

Another challenge lies in the requirement for a metadata database lookup for each video fragment download request. The media frontend needs to translate the video fragment period link from the manifest into the actual URL or link to the video fragment in the object storage. This lookup process adds extra work to the computer and might cause delays, but it is needed to keep things flexible and make sure the data is correct [8].

These are the main reasons for this secondary addressing scheme:

Efficient Video Navigation: The design lets users navigate and choose specific parts of the video within the app by including links to video fragment periods in the manifest. This lets users quickly find and watch parts of the movie that interest them without having to download the whole thing, which saves time and bandwidth.

Handling Video Fragment Relocation: The mapping between video fragment period links and actual fragment locations allows the system to handle potential relocation of video fragments in the object storage. This could occur due to storage optimization or data migration operations. By performing a metadata lookup, the media frontend can ensure that the user receives the correct video fragment even if its storage location has changed since the manifest was generated.

However, performing a database lookup for each video fragment download request adds computational load and potential latency, especially when scaled across a large user base. This necessitates exploring optimization strategies to minimize the overhead associated with metadata lookups and ensure efficient video fragment retrieval.

Although the current design works for video ingestion and serving in home security systems, it needs more research and improvement because of the problems that come with multiple authorizations and metadata lookups. In what follows, we'll look at some possible answers to these problems, with a focus on making things more efficient, cutting down on latency, and keeping costs as low as possible while still ensuring strong security and data integrity.

OPTIMIZATIONS FOR EFFICIENT VIDEO SERVING IN HOME SECURITY SYSTEMS

This section delves into potential optimizations to address the challenges identified in the previous section, focusing on reducing latency, minimizing costs, and enhancing the overall efficiency of video serving in cloud-based home security systems.

Optimizing Authorizations and Reducing Latency

To mitigate the latency and computational overhead associated with multiple authorizations for each video fragment download request, we propose an optimization strategy that leverages time-limited access tokens and hash-based validation [12].

Time-Limited Access Tokens: When the app requests a manifest for a specific duration, the media frontend performs the standard authorization flow, verifying the user's identity and permissions before generating and serving the manifest with video fragment period links. In addition to this standard flow, we introduce the concept of time-limited access tokens.

For each video fragment period URL within the manifest, the media frontend appends a time-limited access token. This token, with a duration of "x" minutes, grants the user temporary access to the corresponding video fragment. This "x" minute duration represents a window during which the user is considered authorized to access the fragment without requiring further authorization checks.

Hash-Based Validation: To ensure the integrity and prevent tampering of the URL, the media frontend also generates a unique hash for each URL, incorporating the time-limited access token. This hash is calculated using a secure hashing algorithm, such as SHA-256, applied to the concatenated URL and access token.

When the app requests to download a video fragment using the modified URL containing the access token and hash, the media frontend performs the following validation steps:

1. **Hash Verification:** The media frontend calculates the hash of the received URL and compares it with the hash provided by the app. If the hashes match, it confirms that the URL has not been tampered with.
2. **Access Token Validation:** If the hash verification is successful, the media frontend checks if the request falls within the x-minute access token window. If the request is within the valid time frame, the media frontend can safely assume that the user's request is authentic and has been recently authorized. Therefore, it can bypass the authorization workflow and proceed directly to the video serving flow.

Benefits and Considerations: This optimization strategy significantly reduces the number of authorization checks performed for each video fragment download request, thereby reducing latency and improving user experience. By eliminating redundant authorization checks, the system can deliver video fragments more quickly, allowing users to access critical security footage with minimal delay.

Furthermore, this approach reduces the computational load on the authorization service, minimizing processing and database query costs associated with each authorization request. This translates to cost savings for the service provider, especially when scaled across a large user base.

The selection of the access token duration ("x" minutes) is crucial. A shorter duration enhances security but may require more frequent authorization checks. Conversely, a longer duration reduces authorization overhead but increases the risk of unauthorized access if permissions change. The optimal duration can be determined based on security requirements, user behavior patterns, and service provider preferences. Machine learning algorithms can be employed to personalize the access token duration based on individual user behavior and risk profiles.

It's important to acknowledge the potential limitation of this approach: a user might gain access to a video URL for a short period beyond the intended retention period. However, this is a rare edge case with minimal practical impact. In most scenarios, either the video fragment would have been deleted, or the additional access duration would be negligible compared to the overall retention period, which is typically days, weeks, or even months.

Optimizing Database Lookups for Efficient Fragment Retrieval

We suggest an optimization approach that includes encoding the actual video fragment URL within the video fragment period URLs in the manifest. This will avoid the extra work that comes with looking up metadata databases for each video fragment download request.

Encoded URLs in Manifest: Instead of adding the video fragment period URLs straight to the manifest, the media frontend encodes the actual video fragment URL within each period URL. To make sure the data is correct and can't be changed, this encoding can be done with a reversible encoding method like Base64 enco-

ding. In order to make things even safer, a unique hash is made for each encoded URL.
URL Decoding and Validation: The media frontend does the following when it gets a request for a video fragment time URL:
URL Decoding: To make sure the URL is correct, the media frontend generates the hash of the received URL and compares it to the hash that was given.
URL Decoding: When the hash check goes well, the media frontend gets the encoded real video fragment URL from the period URL and decodes it to get the direct link to the video fragment in the object storage.
Benefits and Considerations: This optimization gets rid of the need to search a metadata library for every request to download a video fragment. The media frontend can skip the metadata database query by including the URL of the real video fragment directly in the manifest. This cuts down on latency and computational load.
This method also makes the metadata library less busy, which lowers the cost of queries and makes the system work better overall. This saves the service provider money, especially when they have to deal with a lot of requests for video fragments.
But it's very important to make sure that the compressed URLs are safe and correct. Using secure encoding and hashing stops people from changing or accessing the video snippets without permission.
Finally, these optimization techniques help with the problems that come up with cloud-based home security camera systems that need to deal with multiple authorizations and metadata lookups. Using time-limited access tokens, hash-based validation, and encoded URLs, the system can make big improvements in speed, low latency, and low cost, all while keeping strong security and data accuracy. These improvements make the user experience smooth and responsive, so people can get to their security video quickly and reliably.

EXTENDING OPTIMIZATION STRATEGIES TO SIMILAR APPLICATIONS

The previous part talked about optimization strategies that were specifically made for home security cameras. These strategies have a lot of potential to be used in other areas with similar architectures and problems.
These optimizations, primarily focused on reducing latency and computational overhead associated with authorization and metadata lookups, can be effectively applied to other video-centric applications that involve:

- **Frequent user requests for video segments**
- **Large-scale video data management**

Video Conferencing and Collaboration Platforms:

- Time-limited access tokens can be used to grant temporary access to specific meeting recordings or video segments, reducing the need for repeated authorization checks.
- Encoding the actual video segment URLs within the manifest can optimize video retrieval and minimize metadata database lookups, improving loading times and overall user experience.

Medical Imaging and Telemedicine:

- Time-limited access tokens can be used to grant temporary access to medical images and videos, ensuring safe and efficient sharing of data between healthcare providers and patients [14].
- Encoding the actual image or video URLs within the manifest can optimize retrieval and viewing of medical data, facilitating timely diagnosis and treatment.

Online Education Platforms:

- Time-limited access tokens can be used to grant temporary access to online course videos or lecture recordings, reducing authorization overhead and improving access speed for students.
- Encoding the actual video URLs within the manifest can optimize video streaming and minimize metadata database lookups, enhancing the learning experience.

Video Editing Platforms:

- Temporary access to video editing projects or individual clips can be given with time-limited access codes.

This speeds up the editing process and cuts down on authorization delays.

- Encoding the URLs of the real video clips in the manifest can speed up video retrieval and playback while editing, making the process more efficient and improving the user experience.

Different video-focused apps can get big boosts in efficiency, latency, and cost-effectiveness by adapting and using these optimization methods. They can also keep strong security and data integrity. The fact that this is the case shows how important and useful these improvements could be in the changing world of cloud-based video apps.

CONCLUSION

This paper has given a thorough look at how video is stored and sent in cloud-based home security camera systems. It shows how important it is to handle data correctly to get the best performance, cost-effectiveness, and user experience.

We started by going over the usual steps for importing and serving videos, focusing on how important video compression, fragmentation, and adaptive streaming are to this process. Then we looked at the problems that come with this design, mainly the extra time and work that is needed for multiple authorizations and metadata database lookups.

To deal with these problems, we came up with two main improvement strategies:

Optimizing Authorizations and Reducing Latency: We showed how to reduce the number of unnecessary permission checks by using hash-based validation and time-limited access tokens. This cut down on latency and made the user experience better.

Optimizing Database Lookups for Efficient Fragment Retrieval: Getting the most out of database searches for quick fragment retrieval: We showed how to get rid of the need for metadata database lookups for each fragment download request by encoding the real video fragment URLs in the manifest. This cut down on latency and computational overhead even more.

These improvements not only make home security camera systems more fast and efficient, but they also help service providers save a lot of money by lowering the amount of bandwidth, storage, and processing that is needed.

We also looked into how these optimization methods could be used in other situations besides home security, showing that they could be useful for video-based apps like online education platforms, cloud gaming, video conferencing, and medical imaging.

This study shows how important it is to keep improving and coming up with new ways to handle video data in order to keep up with the changing needs of cloud-based video applications. By using the tips and strategies in this paper, service providers and producers can make their video-focused solutions work better, be more affordable, and last longer. This will give users a better experience in the ever-growing digital world.

REFERENCES

1. T. Wiegand, et al., "Overview of the H. 264/AVC video coding standard," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 7, pp. 560-576, 2003.
2. Y. Wang, et al., "Privacy-preserving cloud-based video surveillance framework for smart cities," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 7, pp. 4608-4618, 2020.
3. Huang, Q., Yang, J., & Wang, B. (2012). Cloud Mobile Media: Reflections and Outlook. Microsoft Research, Redmond, WA, USA, Tech. Rep. MSR-TR-2012-82.
4. Chen, M. Y., & Lian, D. (2012, October). Efficient video event recognition using cloud computing. In 2012 IEEE 12th International Conference on Data Mining Workshops (pp. 826-831). IEEE.
5. Wiegand, T., Sullivan, G. J., Bjøntegaard, G., & Luthra, A. (2003). Overview of the H. 264/AVC video coding standard. IEEE Transactions on circuits and systems for video technology, 13(7), 560-576.

6. Stockhammer, T. (2011). Dynamic adaptive streaming over HTTP-: standards and design principles. In Proceedings of the second annual ACM conference on Multimedia systems (pp. 133-144).
7. Skarmeta, A. F. G. (2012). The internet of things. In Green ICT: Trends and challenges (pp. 1-24). River Publishers.
8. Li, S., & Wang, X. (2011, December). Cloud storage techniques. In 2011 International Conference on Computational and Information Sciences (pp. 127-130). IEEE.
9. Pantos, R., & May, W. (2012). HTTP live streaming. IETF Internet Draft, draft-pantos-http-live-streaming-12, 20.
10. Akhshabi, S., Begen, A. C., Dovrolis, C., & Fitzek, F. H. (2011, September). An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In Proceedings of the second annual ACM conference on Multimedia systems(pp. 157-168).
11. Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, 25(6), 599-616
12. Ebert, J., & Vollmer, M. (2012, October). Towards a more efficient authorization service for cloud applications. In 2012 IEEE 12th International Conference on Data Mining Workshops (pp. 798-803). IEEE.
13. Sanchez, V., Abugharbieh, R., & Schätzler, L. (2008). Real-time H. 264 video encoding for telemedicine applications. In 2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (pp. 2116-2119). IEEE.