March 2021

# REAL-TIME SECURE SHARING OF LIVE MEETING CONTENT THROUGH VOICE COMMANDS OR USER CLICKS

Rajarshee Dhar

Deepesh Arora

Ram Mohan R

# REAL-TIME SECURE SHARING OF LIVE MEETING CONTENT THROUGH VOICE COMMANDS OR USER CLICKS

AUTHORS:
Rajarshee Dhar
Deepesh Arora
Ram Mohan R

## ABSTRACT

With the dramatic increase in working from home due to the COVID-19 situation, the reliance on online collaborative sessions has increased manifold and there is a significant need to have an enhanced immersive sharing and collaborative experience in live meetings. Existing sharing and collaboration mechanisms (including the sharing of meeting presentations ahead of time, etc.) all suffer from different weaknesses. To address these types of challenges, techniques are presented herein that support a means whereby, among other things, an authorized participant in an online meeting may request (through, for example, clicking on a shared meeting screen, using a voice input, etc.) that the meeting system, on demand, provide access to additional meeting content such as, for example, links, images, etc.

## DETAILED DESCRIPTION

With the dramatic increase in working from home due to the COVID-19 situation, the reliance on online collaborative sessions has increased manifold and there is a significant need to have an enhanced immersive sharing and collaborative experience in live meetings. For example, one of the common challenges in live meetings where a presenter is sharing content is to get on-demand access to some of the content being shared or obtain additional information (such as, for example, metadata) about the content that can help the non-presenter audience to develop more clarity, to browse links, etc.

The existing means to solve these problems include sharing the meeting presentation ahead of time to all participants thereby allowing other audience members on the bridge to use the material offline and fetch additional content, browse links, etc. or to share content on demand through a meeting chat window or an external real time collaboration client (such as, for example, an Internet Messaging Program (IMP) client).

The problem is more relevant in meetings where the meeting material cannot be shared with the audience ahead of time or the meeting material content cannot be shared at all. Examples of such circumstances may include, possibly among other things, a company's financial review meeting, board meetings, or similar highly confidential discussion-oriented conferences.

Use cases illustrating aspects of the above may include, for example:

1. Online University lectures. In a university lecture session, a professor may employ several dynamic pieces of content in his lecture. He may use a terminal or an integrated development environment (IDE) to demonstrate a code example. He may open a web browser and show an Application Programming Interface (API) definition. Additionally, predefined presentation slides, etc. may be utilized. Thus, sharing ahead of time may not help in such contexts.
2. Collaborative teams. During any brainstorming session within teams, one person may share his screen and go through various links or documents. Other team members can simply fetch that data in real-time without interrupting the presenter and can continue the research further at their respective ends.
3. Board meetings. In a typical financial or board meeting, the presenter reveals some information live. Such information cannot be shared ahead of time with the participants, which in turn will defeat the purpose of such meetings.
4. Confidential content in slides. There may be some confidential content (e.g., links, etc.) in slides that are meant to be shared just within a particular group of people (e.g., individuals within the same company). The sharing of such data ahead of time will defeat the purpose and give access to everyone.
5. Better interview experience and live sessions. Frequently during an interview, the interviewee needs to request a question, or during a live class the students need to request an assignment. A media server may be pre-fed with such information before a meeting starts so that there are no hiccups during the interview or during the live sessions.

To address the types of challenges that were described above, techniques are presented herein that support a means whereby, among other things, an authorized participant in an online meeting may request (through, for example, clicking on the shared meeting screen, using a voice input, etc.) that the meeting system, on demand, provide access to additional meeting content such as, for example, links, images, etc.

Figure 1, below, provides a high-level depiction of aspects of the techniques that are presented herein and which are described in the narrative below.

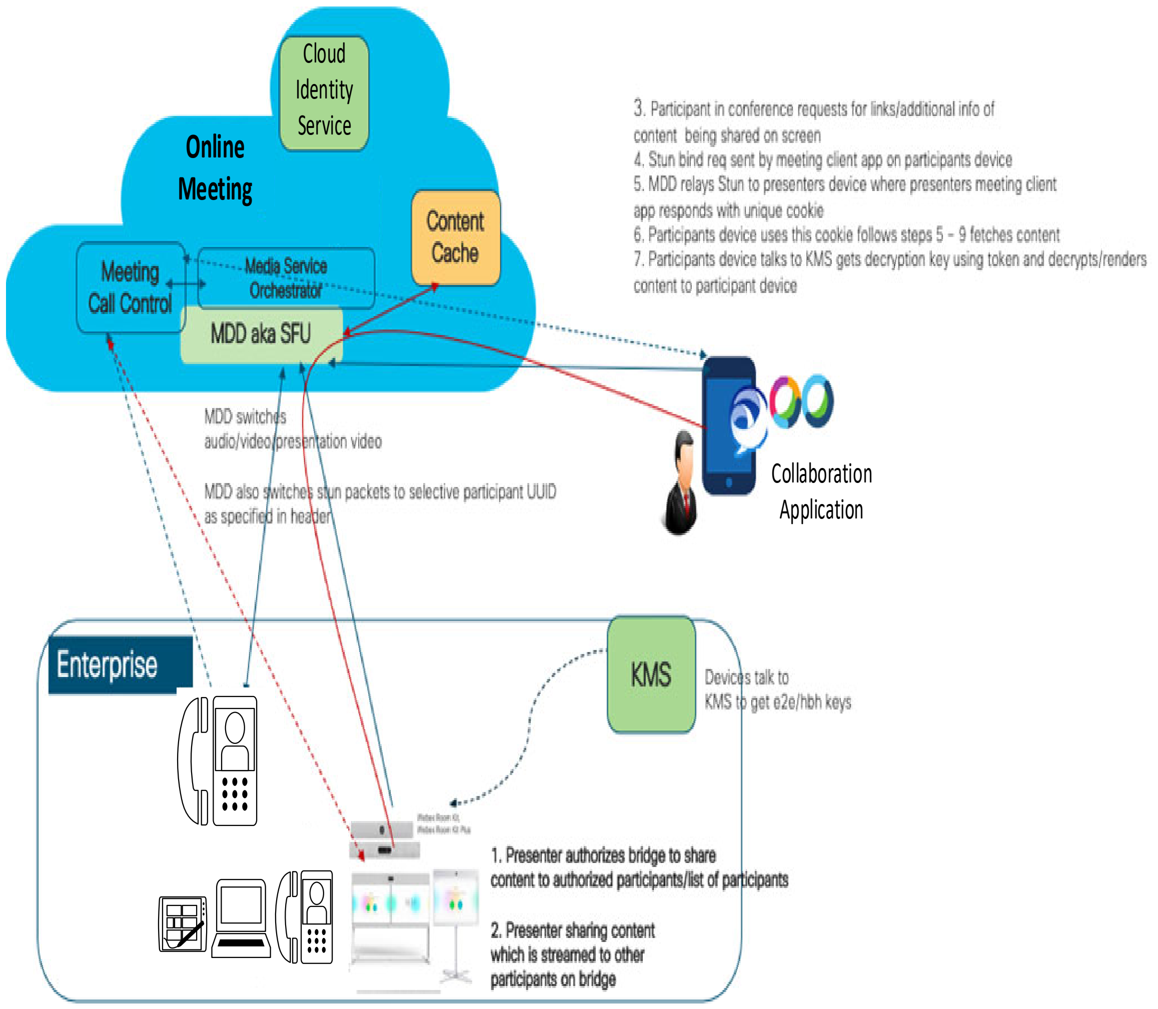

Cloud Identity Service
Online Meeting
Content Cache
Meeting Call Control
Media Service Orchestrator
MDD aka SFU
3. Participant in conference requests for links/additional info of content being shared on screen
4. Stun bind req sent by meeting client app on participants device
5. MDD relays Stun to presenters device where presenters meeting client app responds with unique cookie
6. Participants device uses this cookie follows steps 5 - 9 fetches content
7. Participants device talks to KMS gets decryption key using token and decrypts/renders content to participant device
Collaboration Application
MDD switches audio/video/presentation video
MDD also switches stun packets to selective participant UUID as specified in header
Enterprise
KMS
Devices talk to KMS to get e2e/hbh keys
1. Presenter authorizes bridge to share content to authorized participants/list of participants
2. Presenter sharing content which is streamed to other participants on bridge

*Figure 1: High Level Overview*

Through aspects of the techniques presented herein, the functionality of a web service may be added to online meeting screen shares. To aid in the description of various of the techniques that are presented herein, a series of exemplary steps will be illustrated and described in the narrative below. Note that, in the discussion below, the first step through the ninth step focus primarily on a mechanism to serve real-time content requests (e.g., online meeting screen share as a web service) whereas the final step, step ten, extends the described mechanism to include pre-fed content sharing.

Under a first step, during a screen sharing session an option may be provided to a presenter inquiring whether he wants to enable uninterrupted content sharing. If the presenter allows such sharing then remote users may be provided with a range of options for the different kinds of data that they may request. Different data types may include, for example:

- Generic data types such as, among other things, phone numbers, email addresses, hyperlinks, raw text, images, etc.
- Custom data types which have been pre-fed to the online meeting server by a host prior to a meeting. One possible example might be:
  - Question 1 (From interview perspective)
  - Assignment 1 (From Live-Class perspective)
  - Image Diagram 1 (Generic use-case).

A second step supports the circumstance where a user makes a request for a hyperlink. First, the participant may send a Session Traversal Utilities for network address translation (NAT), or STUN, bind request (that is multiplexed along with a Datagram Transport Layer Security (DTLS) - Real-time Transport Protocol (SRTP) stream) to the presenter for content request authorization.

The participant device meeting client that is requesting the content from a share may insert a new STUN attribute that will carry the sender's and the recipient's universally unique identifier (UUID) (e.g., a participant Address-of-Record (AoR) or other unique identifier) and the synchronization source (SSRC) of the participant to whom the STUN request needs to be routed. The UUID (e.g., an AoR or other identifier) of the participant may be obtained through a conference package subscription (as described, for example, under the Internet Engineering Task Force (IETF) Request for Comments (RFC) 4575.

The sender participant also encrypts their UUID along with the requested content type using end-to-end (E2E) encryption keys so that only the presenter participant device may decrypt the information. Note that E2E keys are shared only with participants, and a Media Distribution Device (MDD) or the cloud will not have access to same.

Figure 2, below, depicts elements of a new attribute (e.g., X-XYZCORP-REQUEST-CONTENT-TOKEN) that may support aspects of the above narrative.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             SSRC                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          receipient UUID(multiple of 32-bit words....)        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          EncVal (sender UUID:nonce)using e2ekey               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

*Figure 2: Exemplary New Attribute*

The MESSAGE-INTEGRITY will be computed using a hop-by-hop (HBH) key so that the MDD can validate and then forward the message to the presenter participant's device. A STUN packet will likely be multiplexed along with DTLS-SRTP packets. The MDD will route the STUN packet on the media channel that corresponds to the SSRC to only the indicated recipient UUID participant.

A third step supports various access control policies. In an online meeting, there may be participants from various organizations and the presenter may not want all of the participants to access of all the content that he is remotely sharing. The presenter in this step may determine various access control policies when he begins the meeting. Exemplary policies may include:

- Policy 1. Allow access to all data to XYZCorp.com employees.
- Policy 2. Allow access to links only to non-XYZCorp.com employees who are subject to confidentiality or data protection checks.

For Policy 2, it is possible to perform two sub-policies:

- A) Have the XYZCorp data protection or confidentiality module decide whether or not the data is confidential. Multiple mechanisms are available that support determining whether or not a given data is confidential.
- B) Have the presenter only choose signatures which he feels are confidential. For example, "opengrok.XYZCorp.com" and "doccentral.XYZCorp.com" may be confidential according to the presenter.

It is important to note that while the above example focuses on links, the underlying concepts are not limited just to links. For example, a presenter may even select write signatures for anything (e.g., diagrams, shapes, text, etc.), even an entire slide, and mark them as confidential.

During a fourth step, based on the policies that were defined by a presenter the online meeting server may categorize the incoming STUN requests, based on the access control policies defined by the presenter, and generate a unique cookie (that is specific to the defined policies) which will then be served in the STUN bind responses.

For instance, a STUN bind request coming from XYZCorp employees (which will have a UUID that identifies the participant uniquely like, for example, a CEC or SIP, an AoR, etc.) will match Policy 1. The presenter device may optionally prompt the presenter to authorize this access or if the presenter has already preauthorized, the device may simply proceed and generate a STUN bind response

The presenter participant device may generate a unique cookie that will be encrypted using an E2E key along with a content type and a nonce in a new attribute. Figure 3, below, depicts elements of such an attribute (e.g., X-XYZCORP--CONTENT-TOKEN-VALUE) that may support aspects of the above narrative.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              SSRC                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          receipient UUID(multiple of 32-bit words....)        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          EncVal(sender UUID:nonce:cookie)using e2ekey         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

*Figure 3: Exemplary New Attribute*

The sender UUID may be fetched from a bind request (after decrypting the encrypted portion using E2E keys) and this may be inserted as a recipient UUID in a bind response. The cookie is generated using policies that are defined to help a content service fetch the right material and give it back to the requestor who has the cookie. The cookie, along with the nonce received in the request, and the sender UUID (here presenters) will all be encrypted using an E2E key such that only other participants who receive this can decrypt the information.

It is important to note that in conference solutions where a single shared key is used across all participants as an E2E key, this STUN bind request/response may use a public and private key pair for exchanging this data between a presenter participant and recipient participants.

Under a fifth step, the participant's meeting clients must have received their corresponding STUN bind responses along with the unique cookie. All of the corresponding data or content fetch requests must make use of the unique cookie. Without the unique cookie, the presenter meeting client will not serve any requests.

During a sixth step, one of the participants may issue a request to the presenter for the links that are currently displayed on the screen that the presenter is sharing. Accordingly, the participant's meeting client will generate a Hypertext Transfer Protocol (HTTP) GET request over QUIC or Transport Layer Security (TLS) with a request type of hyperlink. This request will be sent directly to media server. For example:

```
HTTP GET OnlineMtg.XYZCorp.com/content_request/content_type=hyperlink
x-xyzcorp-content-cookie: XXXX
x-content-type: hyperlink
Accept: */*
User-Agent: Online Meeting Client v1.5
Accept-Encoding: gzip, deflate, br
```

In the above:

- `x-xyzcorp-content-cookie` identifies the cookie which was served by the presenter in a STUN bind response based on the access control policies.

- `x-content-type` contains the requested content type. In the above example, the participant is requesting all of the hyperlinks that are present on the current shared screen.

From a user experience perspective, there may be a number of ways that a participant can request content. For example:

- The participant can employ a left mouse click on his screen and select the content option that he is looking for.
- The participant can also just speak and say "hey online meeting, fetch me the links." The online meeting server may employ speech-to-text processing, which may further trigger various representational state transfer (REST) API calls.

During a seventh step, for this particular session the media server will be notified by the meeting client that the presenter in this session has indicated that the uninterrupted content sharing mode is on. As depicted in Figure 4, below, this basically means that the media server needs to participate as a cache manager along with its media server or mixer role.

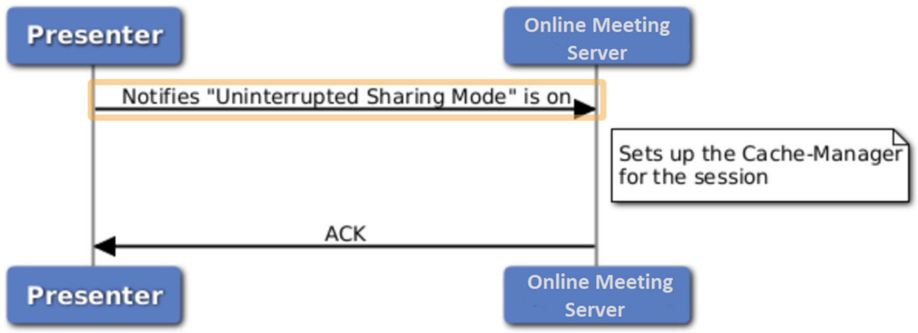
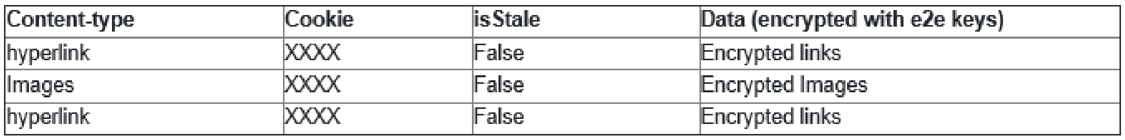

Presenter
Online Meeting Server
Notifies "Uninterrupted Sharing Mode" is on
Sets up the Cache-Manager for the session
ACK
Presenter
Online Meeting Server

*Figure 4: Illustrative Content Sharing Enablement*

As a cache manager, the media server will maintain a map of key-value pairs along with an isStale tag. In a key-value pair the key will be the encrypted request type and the value will be the data that was provided by the presenter's online meeting client which will also be in encrypted form. An isStale tag will indicate whether the data is fresh or stale.

The cache manager may be implemented as an in-memory database for fast access and may include details, as shown in Table 1, below, for a particular online meeting session.

TABLE 1: Example Cache Manager Database Information



| Content-type | Cookie | isStale | Data (encrypted with e2e keys) |
|---|---|---|---|
| hyperlink | XXXX | False | Encrypted links |
| Images | XXXX | False | Encrypted Images |
| hyperlink | XXXX | False | Encrypted links |

Various information, as shown in Table 1, may include:

- Content-type information, which is the content type requested by the participant.
- Cookie information, which is the unique identifier that was served as an access-control mechanism during a STUN bind response.
- A Boolean value 'isStale', which determines whether or not the content is still valid. It may be possible that the presenter has moved to a different screen, then the isStale value will be True. The isStale value will stay False as long as the presenter has not changed his screen, slides, etc.
- Data, which identifies the data that is stored in the meeting server, encrypted with E2E keys so that no one in the middle may interfere with or get access to the data. E2E keys will be negotiated for the DTLS transmission amongst participants and the presenter for the actual media and the same key will be used to also encrypt the content.

An eighth step supports the circumstance where a media server receives a request from a remote user. Three scenarios that may be possible under this step will be presented and discussed below. For simplicity, in the below scenarios the following abbreviations are employed:

- A. The presenter's online meeting client.
- B. A remote user's or requestor's online meeting client.

- E2E Key. The key that is used for the media over a DTLS communication link between the presenter and the participants.
- P2P Key. The key that will be used by participants and the presenter to communicate with the meeting server in their respective legs. It is important to note that this key will be unique to each leg.
- Cathy. An online meeting participant.

Note that, in general, in any data that is conveyed from a presenter to a participant and vice versa (e.g., a STUN bind) the content will be encrypted using an E2E key. Whereas in a HTTP GET request from a participant, notifications from the presenter would be encrypted using a P2P key.

Under a first scenario, consider that the media server does not have the requested key. In this case, the media server will forward the request to 'A' by encrypting it using the P2P key and 'A' will decrypt the request and understand the kind of request initiated by 'B'. Next, 'A' will then perform an optical character recognition (OCR) operation on the current frame being shared in the meeting (using one or more of the many existing cloud APIs) and will fetch the text data that matches the request (in this case hyperlinks). Next, A' will encrypt the data with the E2E key and then send same back to the media server.

The media server will forward the encrypted data to 'B' and will also save it in its dictionary, just in case some other user also asks for the same data until the frame has been changed. Figure 5, below, illustrates aspects of this scenario.

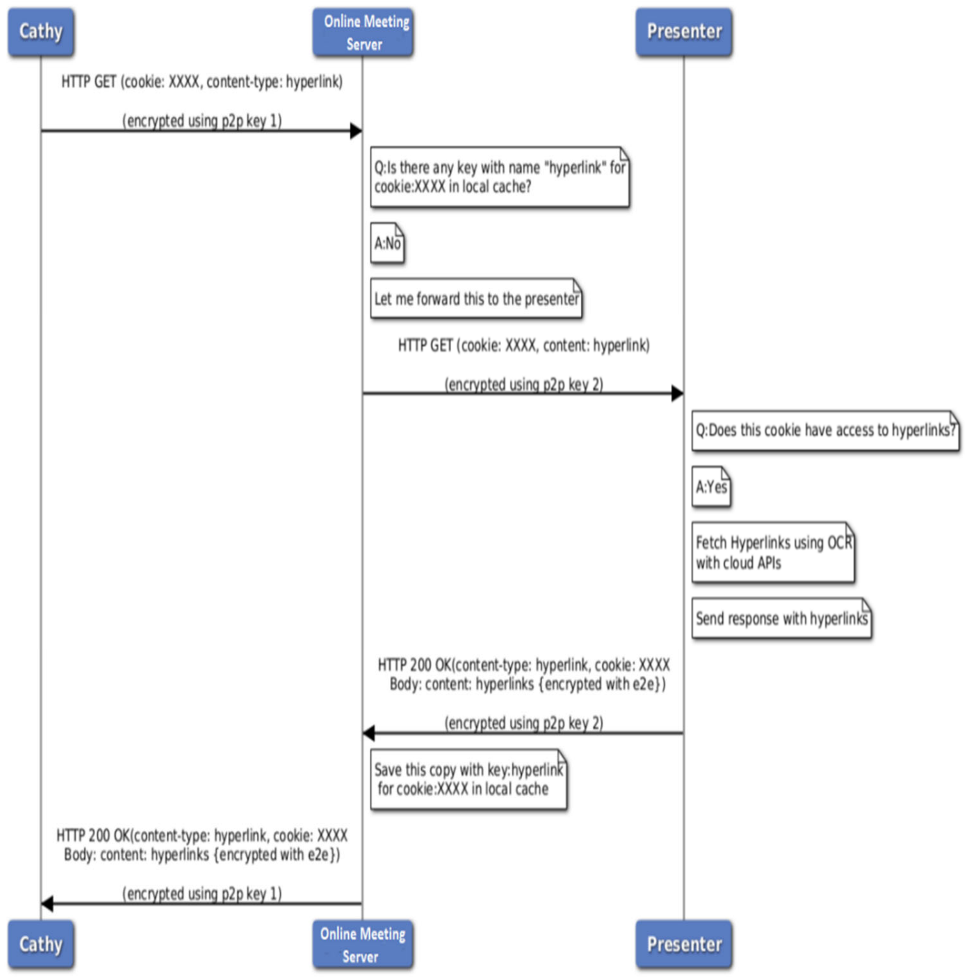

Cathy
Online Meeting Server
Presenter
HTTP GET (cookie: XXXX, content-type: hyperlink)
(encrypted using p2p key 1)
Q:Is there any key with name "hyperlink" for cookie:XXXX in local cache?
A:No
Let me forward this to the presenter
HTTP GET (cookie: XXXX, content: hyperlink)
(encrypted using p2p key 2)
Q:Does this cookie have access to hyperlinks?
A:Yes
Fetch Hyperlinks using OCR with cloud APIs
Send response with hyperlinks
HTTP 200 OK(content-type: hyperlink, cookie: XXXX Body: content: hyperlinks {encrypted with e2e})
(encrypted using p2p key 2)
Save this copy with key:hyperlink for cookie:XXXX in local cache
HTTP 200 OK(content-type: hyperlink, cookie: XXXX Body: content: hyperlinks {encrypted with e2e})
(encrypted using p2p key 1)
Cathy
Online Meeting Server
Presenter

*Figure 5: Scenario One – Media Server Does Not Have Requested Data*

Under a second scenario, the media server has the requested key but the isStale tag is True. This is the case when the media server has already received the data from 'A' but the frame has been changed recently thus marking the data as stale. As depicted in Figure 6, below, which captures aspects of this scenario, the media server will again have to request new data to the media server as in the first scenario.

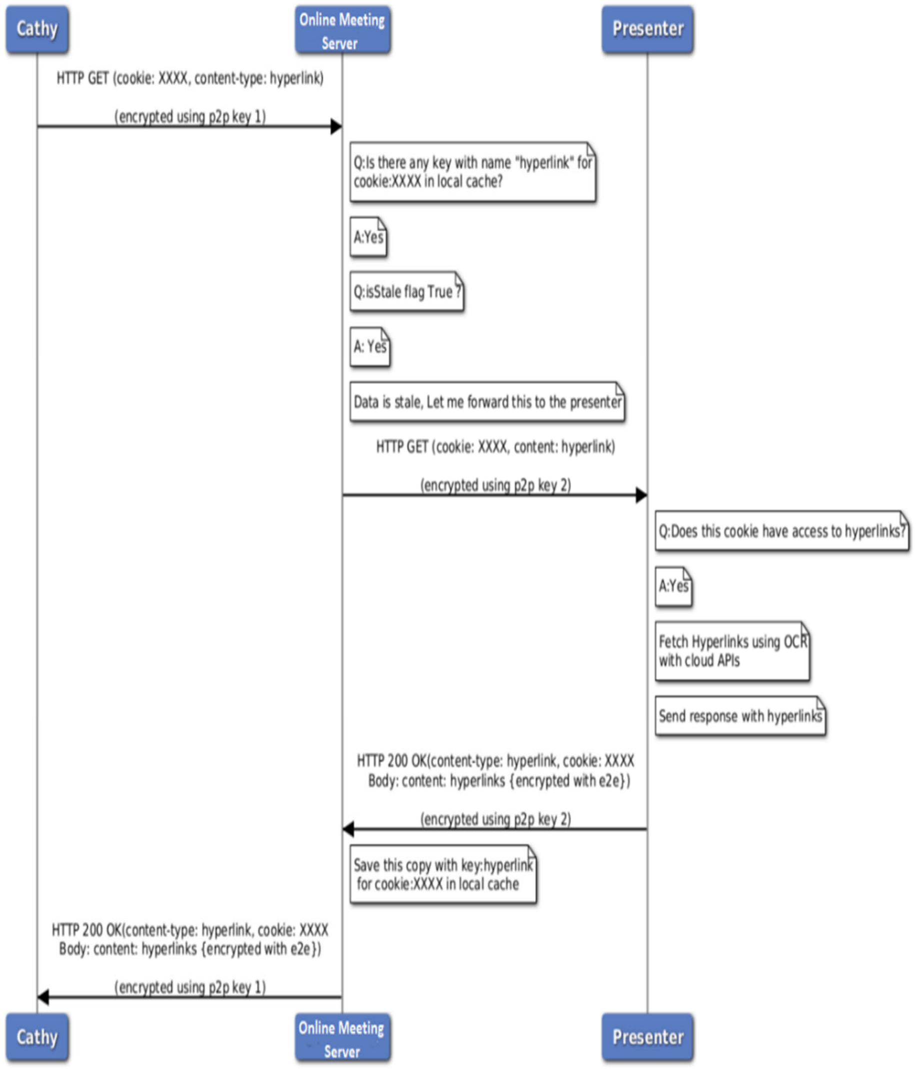

Cathy
Online Meeting Server
Presenter
HTTP GET (cookie: XXXX, content-type: hyperlink)
(encrypted using p2p key 1)
Q:Is there any key with name "hyperlink" for cookie:XXXX in local cache?
A:Yes
Q:isStale flag True ?
A: Yes
Data is stale, Let me forward this to the presenter
HTTP GET (cookie: XXXX, content: hyperlink)
(encrypted using p2p key 2)
Q:Does this cookie have access to hyperlinks?
A:Yes
Fetch Hyperlinks using OCR with cloud APIs
Send response with hyperlinks
HTTP 200 OK(content-type: hyperlink, cookie: XXXX Body: content: hyperlinks {encrypted with e2e})
(encrypted using p2p key 2)
Save this copy with key:hyperlink for cookie:XXXX in local cache
HTTP 200 OK(content-type: hyperlink, cookie: XXXX Body: content: hyperlinks {encrypted with e2e})
(encrypted using p2p key 1)
Cathy
Online Meeting Server
Presenter

*Figure 6: Scenario Two – Media Server has Stale Data*

Under a third scenario, the media server has fresh data. If the media server has an entry in its dictionary for the requested key and the isStale tag is not set to True (indicating that the data is not stale), then the media server is capable of fulfilling this request without any need to query the presenter's online meeting client. Figure 7, below, illustrates aspects of this scenario.

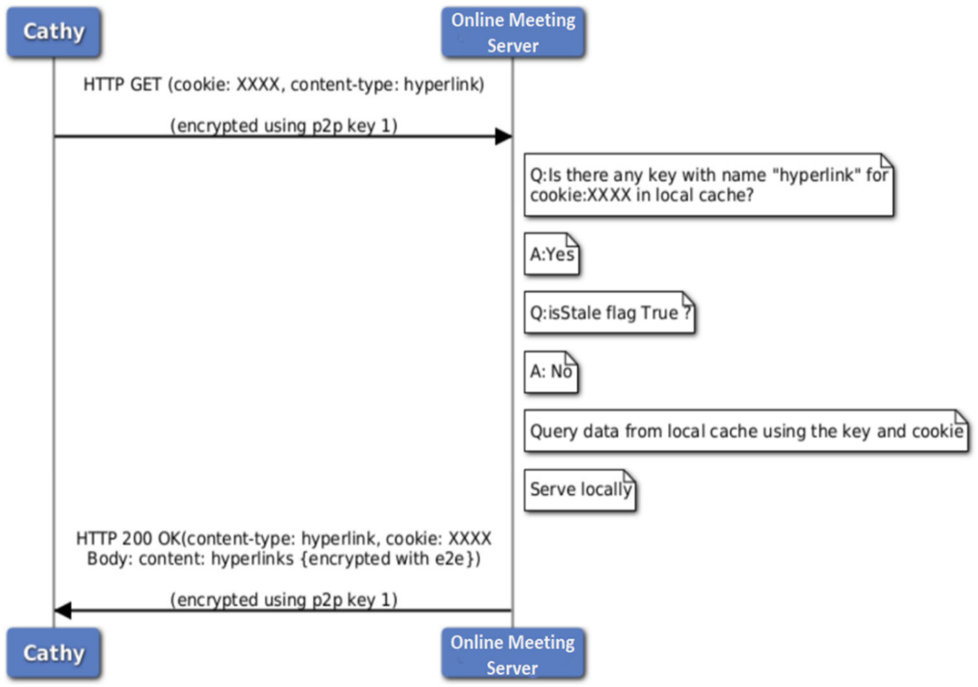

Cathy
Online Meeting Server
HTTP GET (cookie: XXXX, content-type: hyperlink)
(encrypted using p2p key 1)
Q:Is there any key with name "hyperlink" for cookie:XXXX in local cache?
A:Yes
Q:isStale flag True ?
A: No
Query data from local cache using the key and cookie
Serve locally
HTTP 200 OK(content-type: hyperlink, cookie: XXXX Body: content: hyperlinks {encrypted with e2e})
(encrypted using p2p key 1)
Cathy
Online Meeting Server

*Figure 7: Scenario Three – Media Server Has Fresh Data*

A ninth step addresses the issue of synchronization. In the different scenarios that were explained in step eight, above, obviously there needs to be a continuous synchronization between the presenter and the meeting or media server. Whenever there is be a frame change in the screen, the presenter will initiate a lightweight message (e.g., an HTTP PUT request with a flag "isStale") to the media server to indicate that all the previous data stored at the media server is now stale. The media server will need to ask for new data from A if it receives any new request after this point (see scenario number two under step eight, as described above).

Events like mouse clicks and keyboard strokes may be considered as the trigger for a change of frame and hence initiate the lightweight message to the media server to update the isStale tag. There may be a default set of keystrokes that indicate the change of frame but the presenter can modify them during the initial setup of the meeting. The online meeting server just needs to listen to such events and trigger the message for updating the isStale tag. Figure 8, below, depicts aspects of the above synchronization discussion.

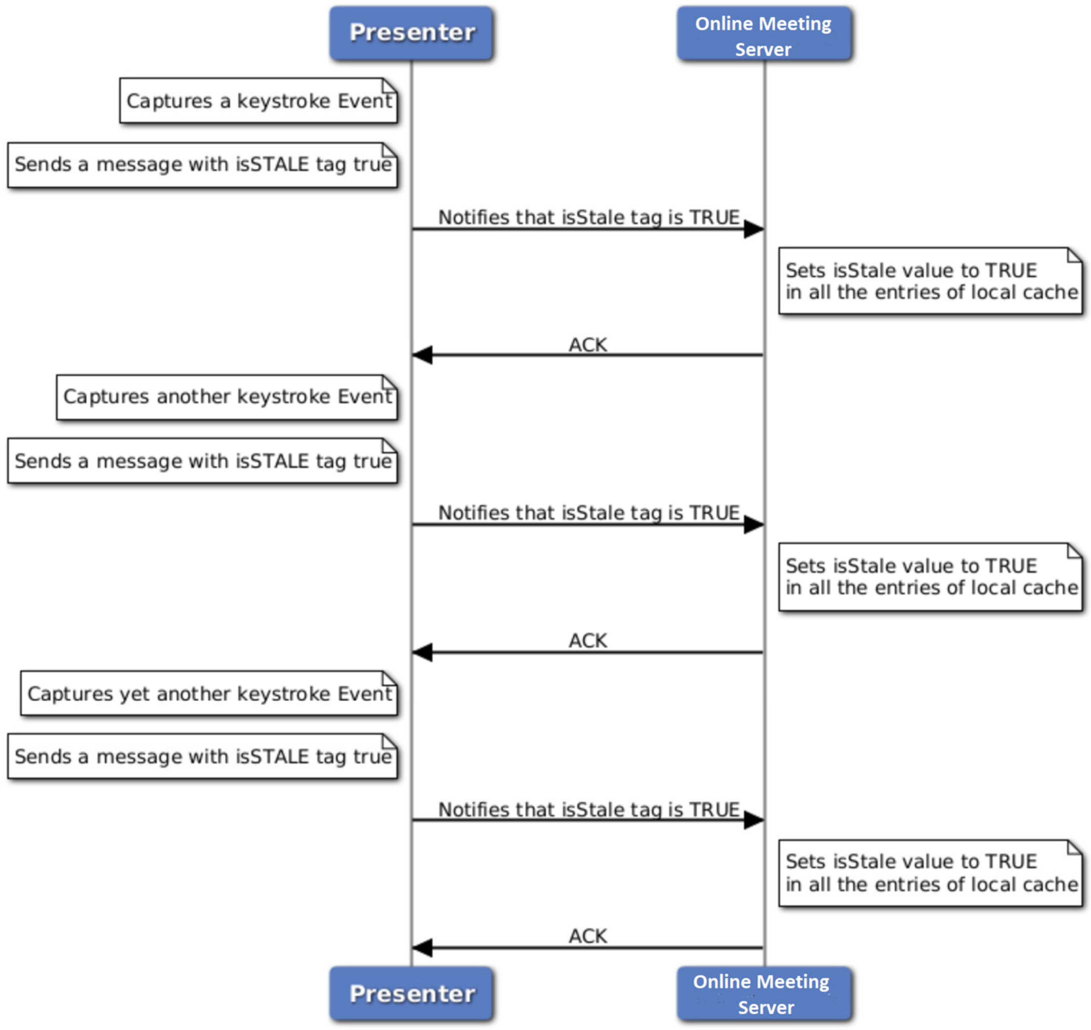

Presenter
Online Meeting Server
Captures a keystroke Event
Sends a message with isSTALE tag true
Notifies that isStale tag is TRUE
Sets isStale value to TRUE in all the entries of local cache
ACK
Captures another keystroke Event
Sends a message with isSTALE tag true
Notifies that isStale tag is TRUE
Sets isStale value to TRUE in all the entries of local cache
ACK
Captures yet another keystroke Event
Sends a message with isSTALE tag true
Notifies that isStale tag is TRUE
Sets isStale value to TRUE in all the entries of local cache
ACK
Presenter
Online Meeting Server

*Figure 8: Illustrative Synchronization*

An alternate approach may employ voice feeds where the presenter sets the voice feedback option in the meeting. Under this option, the online meeting server can listen to key events as specified by the user during his setup (e.g., "moving to the next slide," etc.) or the online meeting server can use natural language processing (NLP) to understand the context from the extracted text from the presenter's speech (e.g., raw audio →speech to text → NLP on the extracted text → trigger event). The presenter may even define a combination of keystrokes and voice feeds to denote the trigger for a change of frame.

It is important to note that the presenter may choose not to define keystrokes. In this case, all the subsequent content requests will need to be served only from the presenter meeting client instead of a meeting client's local cache (see scenario number one under step eight, as described above).

A tenth step supports the pre-feeding of content in a screen share. Before a meeting starts, the host of the meeting may feed information in key-value pairs where a key can be any string value (such as, for example, "Question 1," "Assignment 1," etc.) and a value can be anything (such as, for example, a link, a file, raw text, etc.).

When a user completes a left click during the meeting, these custom fields will also be shown to the presenter which host fed to the online meeting client along with other predefined fields like emails, hyperlinks, etc. Once the host of the meeting gives a voice command (e.g., "Ok online meeting, start Question 1") after this point any student can request this question. Before this point a media server will not serve the question.

For this functionality, a lightweight transaction needs to happen during the start of the meeting in which the host's client will POST all of the key-value pairs to the media server which in turn will POST just the keys to all of the other clients so that the clients would know what data can be requested.

From a user experience perspective, when a student clicks on their screen they can see, for example, "Quiz1: Question1, Question 2, Question 3..." They can select Qestion1 but it won't be served unless the host commands such by saying, for example, "start question 1" as explained above.

Of interest and note in connection with the techniques that are presented herein, and which were discussed above, are, possibly among other things:

- On-demand content request during a live session, which may provide the ability to retrieve content either through voice-activated trigger options (where the participant can just say, for example, "hey online meeting, get me the links from the screen") or through a click on the screen.
- Making an online meeting screen share act as a web service. One of the key elements of the techniques presented herein encompass allowing for the dynamic requests of content from a live share, essentially making an online

meeting screen act as a web service (and make requests for content via GETs) in parallel to being a traditional media sharing application.

- Using STUN for authorization or access control. More particularly:
  - STUN requests may contain information about the participants in the proprietary headers. There may be a chain of headers that may contain information beyond just the UUID of the participants, which may depend upon how the online meeting infrastructure is set up by the administrator. STUN packets may be selectively forwarded to presenter devices.
- Tracking of a presenter's screen through voice feedback or keystrokes. Aspects of the techniques presented herein provide a mechanism by which the online meeting infrastructure can keep a track of a presenter's screen by monitoring keystroke events, or listening to a voice feed from the presenter, or a combination of both (as described in step nine, above). The identified triggers may notify the meeting server and create a synchronization between the meeting server and the presenter's screen.
- Cache manager role of a media server. Data requests may be optimized by, for example, the caching of data that is extracted from the OCR output of a real time frame which has previously been requested. This extends the traditional functionality of a media or meeting server into a cache manager, aligning the entire online meeting infrastructure into a web service.

In summary, techniques have been presented herein that provide for an authorized participant in an online meeting to request (through, for example, clicking on a shared meeting screen, using a voice input, etc.) that the meeting system, on demand, provide access to additional meeting content such as, for example, links, images, etc.