WebSocket-Based Media Forking for Cloud Speech Services

Overview

From Cisco IOS XE Bengaluru 17.6.1a, CUBE can use WebSockets to handle media forking in a Cisco Unified Contact Center Enterprise (UCCE) solution deployment with Cloud Speech Services.

In a typical deployment, Cisco Unified Customer Voice Portal (Cisco Unified CVP) handles a customer call initially before it's transferred to an agent if necessary. If Cloud Speech Services are required, Cisco Unified CVP instructs CUBE to fork media (audio) to the Speech Server using WebSockets. While the call is transferred to the selected agent.

Forking Based on SIP Re-INVITE

When the call is transferred using a SIP Re-INVITE, Cisco Unified CVP includes details of the forking request in a JSON encoded MIME attachment. CUBE sends status information about the forked stream back to Cisco Unified CVP using INFO messages. When the WebSocket connection is successful, the RTP media packets are forked to the Speech Server by CUBE.

Forking Based on SIP INFO Message

Cisco Unified CVP can also trigger a WebSocket-based forking request through the SIP INFO message. The details of the forking request are included in a JSON body similar to a forking request received in re-INVITE. CUBE supports the INFO message with content type **application/x-cisco-record+json** for WebSocket forking.

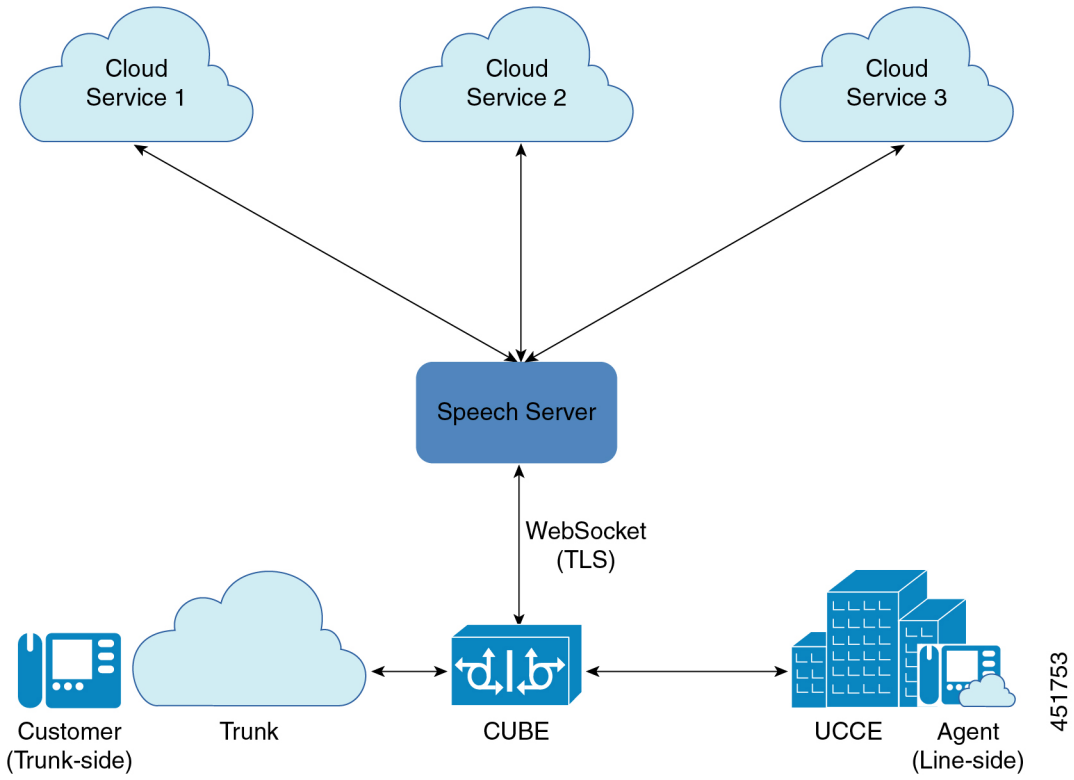
An INFO-based forking request is successful if the INFO message containing the forking request is received after a successful call setup.

Note For more information on the supported and unsupported content types for SIP INFO message, see https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/voice/cube/ios-xe/config/ios-xe-book/m_voi_unsupport_sipinfo_messages.html.

On receiving the forking request in the INFO message of the initial call, CUBE responds immediately to Cisco Unified CVP with a 200 OK message. Also, CUBE initiates a new WebSocket connection if there's no existing forking connection that can be reused for forking the incoming call. CUBE then identifies an existing WebSocket connection to transport the forked media. If there's no connection with available capacity, a new WebSocket connection is established. When a connection is identified, CUBE sends status information in an INFO message to CVP. Once the WebSocket connection is established, CUBE sends metadata to the speech server and starts forking the data packets.

Note If an INFO message is received outside of an existing call, CUBE rejects the request by responding with a 4XX message.

Figure 1: WebSockets in UCCE Solution Deployment



Note While WebSocket forking is enabled through configuration, CUBE only forks calls when explicitly requested with a SIP Re-INVITE or INFO message.

Feature Information

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 1: Feature Information

Feature Name	Releases	Feature Information
Secure WebSocket-based Media Forking on Cisco 4431, 4451-X, and 4461 Integrated Services Routers	Cisco IOS XE Bengaluru 17.6.2	WebSockets support media forking on Cisco 4431, 4451-X, and 4461 Integrated Services Routers.
WebSocket-based Media Forking for Cloud Speech Services	Cisco IOS XE Bengaluru 17.6.1a	WebSockets support media forking for Cloud Speech Services in CUBE.
GCM Ciphers for WebSocket-Based Media Forking	Cisco IOS XE Dublin 17.12.1a	This feature enables GCM cipher negotiation for TLS connectivity with the WebSocket server.

Prerequisites

Cisco IOS XE Bengaluru 17.6.1a or a later release that supports CUBE.

- WebSocket forking for CUBE is supported on the following platforms from Cisco IOS XE Bengaluru 17.6.1a:
  - Cisco Catalyst 8000V Edge Software (Catalyst 8000V)
- WebSocket forking for CUBE is supported on the following platforms from Cisco IOS XE Bengaluru 17.6.2:
  - Cisco 4431 Integrated Services Router
  - Cisco 4451-X Integrated Services Router

- Cisco 4461 Integrated Services Router

Benefits

WebSockets allow transportation of multiple media streams over a single reliable TLS connection, allowing more efficient use of network resources. As WebSockets use HTTPS ports, media streams can traverse firewalls without the need for special policies. Also, it's compatible with HTTP load balancers and proxies.

- CUBE supports a WebSocket deployment with HTTP 1.1 with the following characteristics:
  - Minimal state
  - Data format to stream multiple media sessions over a single connection.
  - No per-stream flow control and flow control is only at the TCP layer.

Restrictions

WebSocket-based forking doesn't support:

- Forking request in messages except Re-INVITE and INFO
- Doesn't support Websocket based forking on IPv6
- Both Re-INVITE and INFO message based forking in the same call. For example, CUBE cannot accept request to start forking (**START**) in Re-INVITE and stop forking (**STOP**) in the INFO message of the same call.
- Forking over Transport Layer Security (TLS) versions other than TLS 1.2
- Video forking
- Other forms of forking on the same CUBE instance
- Session Description Protocol (SDP) passthrough
- Codec payload buffering or VAD detection
- Secure Real-Time Transport Protocol (SRTP) passthrough calls.
- Forking isn't supported on an SRTP dial-peer.
- Call flows other than SIP to SIP.
- While WebSocket media forking can be used with CUBE High Availability architectures, if a switchover happens, active WebSocket media forks are cleared.
- ECDSA-based GCM ciphers isn't supported in the WebexCC Agent Answer infrastructure. Supports only RSA-based GCM ciphers.
- If the hostname gets resolved to multiple IP addresses, CUBE attempts connection on the first IP address only. That is, CUBE doesn't attempt connection with the remaining IP addresses if there’s a connection failure with the first IP address.

Licensing for WebSockets in CUBE

A call that involves successful WebSocket forking in CUBE is counted as an Enhanced trunk call. If WebSocket forking is invoked midcall during a Standard trunk call, it is then considered as an Enhanced trunk call.

Note A call that involves WebSocket-based forking is considered successful if START message is sent to Speech Server and a WebSocket forking session is created.

The following licensing tags are used to track call counts in CUBE:

- CUBE Standard Trunk—License Tag to count the Standard trunk calls in CUBE.
- CUBE Enhanced Trunk—License Tag to count the Enhanced trunk calls in CUBE.
- CUBE Aggregate Trunk—Sum of Standard and Enhanced trunk calls in CUBE.

CUBE Modes and Licensing

- Box-to-Box High Availability (B2BHA) Mode—The CUBE Enhanced Trunk licensing tag is used to record all Enhanced trunk calls in CUBE for the B2BHA mode.
- Standalone Mode—The CUBE Enhanced Trunk and CUBE Standard Trunk licensing tags are used to record all standard and enhanced trunk calls in CUBE for the Standalone mode.
- Inbox High Availability Mode—The licensing for Inbox High Availability mode is similar to Standalone mode. The CUBE Enhanced Trunk and CUBE Standard Trunk licensing tags are used to record all Standard and Enhanced trunk calls in CUBE for the Inbox High Availability mode.

Following are some of the scenarios that are associated with licensing of CUBE calls that use WebSocket forking:

- A call with WebSocket forking is considered as an Enhanced trunk call until the call disconnects. The call is considered as Enhanced trunk call even if the WebSocket forking is stopped midcall.
- In a Call Transfer scenario, if the anchor leg is disconnected, the call involving WebSocket forking is no more treated as an Enhanced trunk call. It is considered as a Standard trunk call.
- A call is counted as an Enhanced trunk call only once even if:
  - WebSocket forking is invoked multiple times on a call leg.
  - WebSocket forking is invoked on both (incoming and outgoing) legs.
- Once a call is counted as an Enhanced trunk call, the call count is not altered even if there are further messages (Update, STOP, START) on the same leg.

High Availability Scenarios

- B2BHA calls in CUBE are counted as Enhanced trunk calls. Hence, when WebSocket forking is invoked on a B2BHA call, the Enhanced trunk call count is not incremented.

- All existing WebSocket forking sessions are cleared if there is a Stateful Switchover (SSO) in CUBE. However, new forking requests that are received after SSO are supported.
- As forking sessions are cleared during SSO, only the Standard trunk calls continue on the new active router.
- Information related to call counting gets checkpointed to the CUBE standby router.
- A B2BHA call continues as an Enhanced trunk call even if the WebSocket forking is stopped or even after SSO.
- High Availability calls in CUBE are checkpointed for serviceability and reporting (before and after SSO).
- Information about a call-leg using enhanced feature (that is, WebSockets) is checkpointed to the standby router from the active CUBE router. This information is used by the standby CUBE router to increment and decrement the call information correctly as Enhanced or Standard trunk call.

Note Inbox High Availability Mode and Standalone Mode use the same call counting logic.

License Usage

The count of calls in CUBE for a specific interval is required for license usage calculation. The following types of call counts are maintained in CUBE:

- CUBE Standard Trunk call count—Number of CUBE trunk calls that does not involve WebSocket forking.
- CUBE Enhanced Trunk call count—Number of CUBE calls involved in successful WebSocket forking.
- CUBE Aggregate Trunk call count—Sum of Enhanced and Standard trunk calls in CUBE.

License Usage Reporting

License usage reporting for the different CUBE modes are as follows:

- Box-to-Box High Availability (B2BHA) Mode—Peak usage value for the periodicity interval that is defined is reported by:
  - CUBE Enhanced Trunk tag for CUBE Enhanced calls (peak value).
- Standalone Mode—Peak usage value for the periodicity interval that is defined is reported by:
  - CUBE Enhanced Trunk tag.
  - CUBE Standard Trunk tag (Difference of peak value of CUBE Aggregate Trunk calls and CUBE Enhanced Trunk calls).
- Inbox High Availability Mode—Peak usage value for the periodicity interval that is defined is reported by:
  - CUBE Enhanced Trunk tag.

- CUBE Standard Trunk tag (Difference of peak value of CUBE Aggregate trunk calls and CUBE Enhanced trunk calls).

Note License reporting happens simultaneously for all CUBE tags. Separate timers are not maintained for any of the tags.

CUBE prints call history with data on the call count. The following is a sample call history with Enhanced and Aggregate trunk call count during call connect:

```
CUBE Standard Trunk call count: 0 , CUBE Enhanced Trunk Call count: 1, CUBE Aggregate Trunk
 call count: 1 after WS_Fork_START
```

The following is a sample call history with Enhanced and Aggregate trunk call count during call disconnect:

```
CUBE Enhanced Trunk Call count: 0, CUBE Aggregate Trunk call count: 0 after call disconnected
```

Use the show command **show voice sip license stats table** to display the summary of last 10 license usage reports. For more information on license reporting periodicity and commands to verify the platform registration and license usage, see Verify Smart License Operation

Feature Characteristics

- WebSockets fork SIP-to-SIP calls using G.711ulaw or G.711alaw.
- CUBE establishes a TCP connection with the destination URL provided in a forking request. The host address that is provided in the URL may either be an IPv4 address or a fully qualified domain name that is resolved using DNS. Once a connection is made with a destination, it can be used for multiple forked media streams. Two media streams are created for each forked call, one from the calling and one from the called party.
- CUBE uses INFO messages to provide connection status information to a Unified CVP and to signal the start of a stream to the Speech Server. Each stream is identified by a unique channel identifer.
- CUBE initiates a WebSocket connection by sending an HTTP GET request to the Speech Server with **Upgrade: websocket**. If the connection is successful, the speech server responds with **HTTP/1.1 101 Switching Protocols**. CUBE validates the connection by matching the Accept keys (**Sec-WebSocket-Accept**) that are exchanged in **GET** and **101** messages.
- The maximum number of WebSocket connections is based on the router platform.
- Established WebSockets maintain a persistent connection for as long as they are being used. A WebSocket connection is closed in either of the following ways:
  - If idle for more that 30 minutes, a WebSocket connection is closed automatically. To define a custom duration for the connection timeout, configure **connection idle-timeout** *minutes*.
  - Use the **clear voip stream-service connection** *id* **forced** command to clear a WebSocket connection.
- The following show commands display information about WebSockets in CUBE:
  - **show voip stream-service callid** *callid* —Displays detailed information about a WebSocket fork using the call ID of the original call.

- **show voip stream-service connection**—Displays information about all the active WebSocket connections in CUBE.
- **show voip stream-service connection history**—Displays information about closed or stale WebSocket connections in CUBE.
- **show voip stream-service connection id** *id*—Displays detailed information about a specific WebSocket connection in CUBE.
- **show voip stream-service server** *ip:port*—Displays information about WebSocket connections corresponding to a specific Speech Server IP address and port.
- **show voip stream-service statistics**—Displays statistical information about WebSocket connections in CUBE.
- **show platform hardware qfp active feature sbc fork global**—Displays media forking statistics that are related to all the forking instances for an active Cisco Quantum Flow Processor (QFP) instance of CUBE.
- **show platform hardware qfp active feature sbc fork session**—Displays media forking statistics specific to a fork session for an active Cisco Quantum Flow Processor (QFP) instance of CUBE.

- Use the **clear voip stream-service statistics** command to reset the global WebSocket statistics in CUBE.

Load Balancing

CUBE supports load balancing of forked media streams across multiple WebSocket connections. Each connection to a Speech Server can accommodate a maximum number of calls (threshold).

When CUBE receives a forking request, it assigns the media stream to one of the less busy connections to the target Speech Server. You can configure the maximum number of calls that CUBE assigns to each connection using the command **connection calls-threshold** *calls*. The range is 1–20 calls. We recommend that you configure the default call threshold of three calls per connection.

Consider the following scenario. The threshold for the number of calls that a WebSocket connection can handle is ten. If all the WebSocket connections are handling ten calls each, then CUBE creates a new WebSocket connection for the incoming call.

Pause and Resume Forking

The codec that is used for a WebSocket fork must be the same as the one negotiated for the call. Consider a scenario in which CUBE receives an initial call with codec G711ulaw. If CUBE receives a forking request (re-INVITE with **START** message) with G711ulaw, then CUBE starts forking. However, if CUBE receives another re-INVITE with codec G711alaw during forking, a **PAUSE** message is sent to the Speech Server to suspend the forked stream.

CUBE sends a **PAUSE** message to the Speech Server to pause WebSocket-based forking in the following scenarios:

- A call with an ongoing forking session is placed on hold.
- A call with an ongoing forking session receives a re-INVITE with a codec other than the previously negotiated codec—In this scenario, codec information is available in the initial re-INVITE with the forking request.

WebSocket-based forking is resumed by sending a **RESUME** message to the Speech Server. The following are the prerequisites for CUBE to resume WebSocket-based forking:

- The forking is currently in the paused state.
- The call is currently not on hold.

The following are the scenarios in which CUBE resumes WebSocket-based forking.

- CUBE receives re-INVITE and the codec in the forking request is same as the negotiated codec.
- The call is placed on hold and the corresponding forking is paused. When the call is resumed, the forking is also resumed if:
  - The newly negotiated codec is same as the one received in the re-INVITE with a forking request.
  - The re-INVITE with a forking request doesn't have any codec information, and the call-negotiated codec is either G711ulaw or G711alaw.

Note WebSocket forking can only be used for g711ulaw and g711alaw codecs.

- CUBE sends a **PAUSE** message followed by a **RESUME** message even for a single re-INVITE forking request, in the following scenario. The initial re-INVITE with a forking request doesn't contain codec information. Call negotiation codec is either G711ulaw or G711alaw. Then CUBE uses either of these codecs (for example, G711alaw) to trigger forking. If CUBE receives a midcall re-INVITE with G711ulaw, it sends **PAUSE** followed by **RESUME** message. The **RESUME** message contains information on the newly negotiated codec.

Note
- CUBE receives **START** and **STOP** in the re-INVITE or INFO message from Unified CVP to start and stop WebSocket-based forking respectively.

INFO-Based Forking

Some of the primary call scenarios and the corresponding behavior for INFO-based forking include:

- CUBE receives a Re-INVITE (with codec change, media address change or session refresh) request after the initial call is established—The Re-INVITE is handled as for a normal call. As INFO with JSON is received in the initial call, the CUBE triggers the forking request.
- CUBE receives a Re-INVITE after the initial call is established. INFO with JSON is received before completing the Re-INVITE transaction—As INFO with JSON is received in the initial call, CUBE triggers the forking request.
  - Re-INVITE for codec change introduces PAUSE and RESUME into the forking scenario.
  - Re-INVITE for media address change is handled consistent to the handling of a normal call.
  - Re-INVITE for session refresh is handled consistent to the handling of a normal call.

- CUBE successfully triggers a forking request but Unified CVP does not respond with 200 OK and retransmits INFO with JSON—CUBE ignores the retransmitted forking request.

UPDATE Message

Unified CVP can request CUBE to update the metadata in an ongoing forking session using the UPDATE message type. The metadata for the ongoing forking is in the JSON body of a Re-INVITE or INFO message.

Note The UPDATE message is contained in the JSON body of a SIP INFO message.

Note It's not allowed to use UPDATE message to change the forking session or connection. If a change in forking session or connection is required, use the STOP message to stop the current forking and the START message to restart forking with the new details.

Alarms

Alarms are generated corresponding to the events logged for WebSocket-based forking in CUBE. Alarms are generated in the form of syslog messages for the following events:

- Syslog Alarms
  - TCP failures—Failures while establishing a TCP connection. For example, an error while trying to set up a WebSocket connection with a server by providing the wrong IP address or port.
  - HTTP failures—Failures while establishing an HTTP connection, including authorization.
  - Remote WebSocket closure—An alarm is generated when the Speech Server closes a WebSocket connection gracefully.
  - TCP closure—An alarm is generated when a TCP connection to the Speech Server is closed unexpectedly.
- Alarms in Event Trace Logging

The following is a sample syslog alarm format specific to WebSocket-based forking:

```
%SIP-3-STREAM_SERVICE:   1 HTTP connection failures
%SIP-3-STREAM_SERVICE:  50 TCP connection failures
%SIP-3-STREAM_SERVICE:  60 Remote WebSocket closures
%SIP-3-STREAM_SERVICE: 120 TCP closures
```

Note By default, the syslog alarm is generated for the first instance of TCP failure, HTTP failure, Remote WebSocket closure, and TCP closure. Thereafter, CUBE generates syslog alarms for these events after every ten occurences.

When you execute the command **clear voip stream-service statistics**, the statistics for the events TCP failure, HTTP failure, Remote WS closure and TCP closure are reset. The command **show voip stream-service statistics** displays these statistics.

Event Trace Logging

For WebSocket-based forking in CUBE, events are logged using the VoIP Trace and Event Trace Logging framework. While VoIP Trace logs the call events, Event Trace logs the global events and call events.

VoIP Trace logs call events that are related to WebSocket-based forking:

- Forking request for a call
- Forking initiated for a call
- Forking paused or resumed
- Forking stopped

Event Trace logs global events that are related to WebSocket-based forking:

- WebSocket creation
- WebSocket closure (Due to idle timeout)
- WebSocket closure (Due to clear command—**clear voip stream-service connection** *id*)
- WebSocket remote connection closure (Due to Speech Services closing the WebSocket connection gracefully)
- WebSocket TCP connection closure (Speech service outage is one of the identified causes)
- WebSocket connection creation failure
- Alarms. This is similar to Syslog Alarms.

The following is a sample output for Event Trace logging specific to WebSocket-based forking:

```
*Oct 14 07:18:10.913: CUBE_ET: TYPE = GLOBAL : WebSocket Connection creation successful for
 ws://10.64.86.215:8000
*Oct 14 20:25:12.160: CUBE_ET: TYPE = GLOBAL : WebSocket Connection closed locally due to
idle-timeout expiry for ws://10.64.86.215:8000
*Oct 14 07:20:08.786: CUBE_ET: TYPE = GLOBAL : WebSocket Connection closed locally using
clear command for ws://10.64.86.215:8001
*Oct 14 21:01:38.061: CUBE_ET: TYPE = GLOBAL : WebSocket Connection was remotely closed for
 ws://10.64.86.215:8002
*Oct 14 22:13:00.879: CUBE_ET: TYPE = GLOBAL : WebSocket Connection : 1 HTTP connection
failures
*Oct 14 07:18:31.205: CUBE_ET: TYPE = GLOBAL : WebSocket Connection was closed over TCP for
 ws://10.64.86.215:8000
```

Server Name Indication (SNI)

WebSocket-based forking in CUBE provides support for Server Name Indication (SNI). SNI is a Transport Layer Security (TLS) extension that allows a TLS client to indicate the name of the server that it's trying to connect with during the initial TLS handshake process. The server hostname that is received in JSON by the WebSocket is used to configure SNI. SNI isn't set if the IP address of the host is received in JSON. For WebSocket-based forking, SNI is always used for WebSocket forking where possible.

If CUBE uses a proxy server for setting up a WebSocket connection, then SNI is set using the hostname of the proxy server that you configure. However, SNI isn't set if the IP address of the proxy server is configured instead of the hostname.

Common Name and Subject Alternate Name

WebSocket-based forking in CUBE supports server identity validation through Common Name and Subject Alternate Name fields in the server certificate. Common Name and Subject Alternate Name validation is supported only when the hostname of the server is available. Common Name and Subject Alternate Name isn't validated in WebSockets if CUBE receives the IP address of the host or proxy server.

For more information on SNI, Common Name and Subject Alternate Name, see SIP TLS Support on CUBE.

Error Strings in WebSocket Forking

CUBE sends a SIP INFO message with the forking status to Cisco Unified CVP if an error occurs during an ongoing WebSocket forking.

Some of the forking related error strings originating from CUBE (toward Unified CVP) include:

Table 2: Error Strings in WebSockets

Error String	Error
Codec is not supported	Codec that is negotiated in the call or received in a forking request is not supported. WebSocket forking only supports G711ulaw and G711alaw codecs.
Connection failed	Unable to establish WebSocket connection with the Speech Server.
Internal error	Error occurred during the internal processing of WebSocket.
Connection closed	Closure of an idle WebSocket connection either locally or by the server. Closure of a WebSocket connection using a CLI command.
Authentication token invalid	Speech Server rejects the authentication token.
JSON parse failure	Parsing of the JSON content in the forking request failed.
Unexpected format or content in JSON	Unexpected value for a parameter in JSON. For example, **"codec": "g729"** or **"port": "8080" (instead of "port": 8080)**
Mandatory parameter missing in JSON	One or more mandatory parameters are missing. For example, **call-guid**.
Method not supported	Request for an unsupported method received.
Invalid Start Fork Request	Out of sequence START message from CVP. For example, a forking session is active but CUBE receives START message.

Error String	Error
Invalid Stop Fork Request	CUBE receives out of sequence STOP message from Unified CVP. For example, fork session is inactive but CUBE receives STOP message.
DNS lookup failed	DNS lookup for the hostname (as received in JSON) of the Speech Server is unsuccessful.
Proxy DNS lookup failed	DNS Lookup for the proxy server that is configured is unsuccessful.
Unsupported flow	Active call flow uses an unsupported feature with WebSocket-based forking. For example, SRTP, FAX, SIPREC, SDP Passthrough, SIP-TDM, and so on. WebSocket forking is not enabled on the corresponding dial-peer.
Forking stopped	CUBE stops forking in response to the STOP request received from Unified CVP.
TCP Connection closed	TCP connection with the Speech Server is closed.
High availability Switchover	Switchover happens on CUBE and forking is stopped. Not applicable for CSR.
GUID Mismatch	GUID received in UPDATE message is different from START the message.
Invalid Update Fork Request	Out of sequence UPDATE message from CVP. For example, when a fork session is inactive, CUBE receives UPDATE message.

Configure WebSocket-Based Forking

Perform this task to configure WebSocket-based forking in CUBE.

Before you begin

- Cisco IOS XE Bengaluru 17.6.1a or a later release that supports CUBE.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **media profile stream-service** *tag*
4. (Optional) **secure-ciphersuite** *list*
5. (Optional) **connection** {**calls-threshold** *calls* | **idle-timeout** *minutes*}
6. **description** *string*
7. **proxy** [**host** *host* **port** *port* | **ip** *word* **port** *port string*]
8. **source-ip** *ip-address*

9. **exit**
10. **media class** *tag*
11. **stream-service profile** *tag*
12. **exit**
13. **dial-peer voice** *tag* **voip**
14. **media-class** *tag*
15. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Device> enable`	Enters privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Device# configure terminal`	Enters global configuration mode.
Step 3	**media profile stream-service** *tag* **Example:** `Device(config)# media profile stream-service 99`	Enables stream-service on CUBE.
Step 4	(Optional) **secure-ciphersuite** *list* **Example:** `Device(cfg-mediaprofile)# secure cipher-suite aes-128-cbc-sha`	Configures the cipher suites (encryption algorithms) to be used for encryption over HTTPS for a WebSocket connection in CUBE.

	Command or Action	Purpose
	Example: `Device(cfg-mediaprofile)# secure cipher-suite ecdhe-ecdsa-aes-gcm-sha2`	**Note** The following cipher suites are supported by WebSockets: • aes-128-cbc-sha • dhe-rsa-aes-cbc-sha2 • ecdhe-rsa-aes-cbc-sha2 • rsa-aes-cbc-sha2 • ecdhe-ecdsa-aes-gcm-sha2 • ecdhe-rsa-aes-gcm-sha2 WebSocket forking with GCM cipher suites includes AES256, and AES128 encryption algorithms: • ecdhe-ecdsa-aes-gcm-sha2 includes ECDHE-ECDSA-AES256-GCM-SHA384 and ECDHE-ECDSA-AES128-GCM-SHA256. • ecdhe-rsa-aes-gcm-sha2 includes ECDHE-RSA-AES256-GCM-SHA384 and ECDHE-RSA-AES128-GCM-SHA256.
Step 5	(Optional) **connection** {**calls-threshold** *calls* | **idle-timeout** *minutes*} **Example:** `Device(cfg-mediaprofile)# connection idle-timeout 45`	Configures idle timeout and call threshold for a media profile in CUBE. • If you don't provide any configuration, the default values are applied for timeout. Default for **idle-timeout** is 30 minutes. • The default for **calls-threshold** is three, and is the recommended call threshold value.
Step 6	**description** *string* **Example:** `Device(cfg-mediaprofile)# description <text>`	Adds a description of up to 64 alphanumeric characters to a media profile.
Step 7	**proxy** [**host** *host* **port** *port* | **ip** *word* **port** *port string*] **Example:** `Device(cfg-mediaprofile)# proxy ip 1.1.1.1 port 3456`	Configures the IP address or hostname of a WebSocket proxy server in CUBE.
Step 8	**source-ip** *ip-address* **Example:**	Configures the local source IP address of a WebSocket connection in CUBE.

	Command or Action	Purpose
	`Device(cfg-mediaprofile)# source-ip 10.64.86.70`	
Step 9	**exit** **Example:** `Device(cfg-mediaprofile)# exit`	Exits media profile configuration mode and enters global configuration mode.
Step 10	**media class** *tag* **Example:** `Device(config)# media class 9`	Configures a media class and enters media class configuration mode.
Step 11	**stream-service profile** *tag* **Example:** `Device(cfg-mediaclass)# stream-service profile 99`	Associates the details specific to **stream-service** with **media class** for WebSocket.
Step 12	**exit** **Example:** `Device(cfg-mediaclass)# exit`	Exits to global configuration mode.
Step 13	**dial-peer voice** *tag* **voip** **Example:** `Device(config)# dial-peer voice 9090 voip`	Defines a dial peer and enters the dial peer configuration mode.
Step 14	**media-class** *tag* **Example:** `Device(config-dial-peer)# media-class 9`	Binds the media class to a dial peer. It's mandatory to configure **media-class** and bind it to a dial peer to enable WebSocket forking. Otherwise, WebSocket-based forking is disabled for your dial-peer.
Step 15	**end** **Example:** `Device(config-dial-peer)# end`	Exits dial peer configuration mode and enters privileged EXEC mode.

Configure CA Signed Certificates for SIP TLS Support in WebSockets

For WebSocket forking to be supported over TLS, CUBE must perform CA certificate exchange with the remote server. To import the CA certificate and to configure the cipher suite that CUBE supports, perform the following steps:

Step 1 Configure trustpoint.

Example:

The following is a sample configuration for RSA trustpoint:

```
configure terminal
crypto pki trustpoint websocket
     enrollment terminal
     revocation-check none
     exit
```

Example:

The following are the steps to configure ECDSA trustpoint:

a. ECDSA key regeneration:

```
crypto key generate ec keysize 256 label ECK256
```

b. Configure ECDSA trustpoint:

```
configure terminal
crypto pki trustpoint ECDSA-DM
    enrollment terminal pem
    serial-number
    subject-name cn=MYSUB
    revocation-check none
    eckeypair ECK256
    exit
```

c. CA certificate authentication:

```
crypto pki authenticate ECDSA-DM
```

For more information, see Configuring Certificate Enrollment for a PKI.

Step 2 Import CA certificate for WebSockets.

Open Certificate in Notepad and copy-and-paste content from BEGIN CERTIFICATE REQUEST to END CERTIFICATE REQUEST.

Note Ensure that the clock on the CUBE router and the WebSocket server are consistent.

Example:

```
Router (config)#crypto pki authenticate websocket
Enter the base 64 encoded CA certificate.
End with a blank line or the word "quit" on a line by itself
```

Step 3 Bind the trustpoint with an HTTP client to be used for secure communication channels.

Example:

```
configure terminal
http client secure-trustpoint websocket
```

Step 4 Configure the HTTP client to negotiate specified ciphers.

Note Starting from Cisco IOS XE Dublin 17.12.1a, GCM ciphers are also supported for WebSocket connections.

Example:

```
media profile stream-service tag
    secure-ciphersuite ecdhe-rsa-aes-gcm-sha2
```

Verify WebSocket-Based Forking

SUMMARY STEPS

1. **show run | sec media class**
2. **show run | sec dial-peer voice**
3. **show voip stream-service connection id** *id*

DETAILED STEPS

Step 1 **show run | sec media class**

Show command output to verify **media class** configuration for WebSockets.

Example:

```
#show run | sec media class

media class 8
 stream-service profile 80
```

Step 2 **show run | sec dial-peer voice**

Show command to verify **media class** binding with a dial peer.

Example:

```
dial-peer voice 900 voip
 session protocol sipv2
 session target ipv4:8.41.17.71:7051
 session transport udp
 incoming called-number 6777
 dtmf-relay rtp-nte
 codec g711alaw

dial-peer voice 901 voip
 destination-pattern 6777
 session protocol sipv2
 session target ipv4:8.41.17.71:7052
 session transport udp
 media-class 8
 dtmf-relay rtp-nte
 codec g711alaw
```

Step 3 **show voip stream-service connection id** *id*

The following is a sample output for the **show voip stream-service connection id** command displaying a GCM specific cipher secure WebSocket connection:

Example:

```
router# show voip stream-service connection id 60
Id: 60
Total session count: 1
Active session count: 1
Secure: Yes
TLS Version: TLS1.2
Cipher Suite: ECDHE-RSA-AES256_GCM-SHA384
Auth Token: e2238f3a-e43c-3f54-a05a-dd2e4bd4631f
Server Address: 10.1.40.50:8051
Local Address: 10.2.10.10:52642
State: Active
Connected at: Feb 7 07:47:27 UTC

Anchor leg cccallid        Data plane fork session id
       58                            2
```