Key Functions and Sequences

1. Initialization and Connection

```
python
Copy code
import logging
import requests
import ari
import socket
import threading
import signal
import sys
import time

logging.basicConfig(level=logging.INFO)

client = ari.connect('http://10.0.0.1:8088', 'asterisk', 'password')
```

- Sets up logging and connects to the Asterisk ARI using the provided credentials.

2. Port Management

```
python
Copy code
PORT_RANGE_START = 26500
PORT_RANGE_END = 26999
available_ports = list(range(PORT_RANGE_START, PORT_RANGE_END + 1))
mapping_table = {}

def get_free_port():
    if available_ports:
        return available_ports.pop(0)
    else:
        raise Exception("No available ports")

def release_port(port):
    available_ports.append(port)
    available_ports.sort()
```

- Manages a range of UDP ports to be used for external media connections. Functions to get and release ports ensure ports are reused efficiently.

3. Safe Resource Handling

```
python
Copy code
def safe_hangup(channel):
    try:
        channel.hangup()
        logging.info("Hung up {}".format(channel.json.get('name')))
    except requests.HTTPError as e:
        if e.response.status_code != requests.codes.not_found:
            logging.error("Error hanging up channel: {}".format(e))
```

```
def safe_bridge_destroy(bridge):
    try:
        bridge.destroy()
        logging.info("Destroyed bridge {}".format(bridge.id))
    except requests.HTTPError as e:
        if e.response.status_code != requests.codes.not_found:
            logging.error("Error destroying bridge: {}".format(e))
```

- Provides functions to safely hang up channels and destroy bridges, handling HTTP errors appropriately.

4. Master Cleanup

```
python
Copy code
def master_cleanup(port, reason=""):
    logging.info("Initiating master cleanup for port {}. Reason:
{}".format(port, reason))
    if port in mapping_table:
        resources = mapping_table[port]
        if resources.get('stop_event'):
            resources['stop_event'].set()
        if resources.get('sock'):
            resources['sock'].close()
        for channel_key in ['inbound_channel', 'dialed_channel',
'external_channel']:
            if resources.get(channel_key) and
resources[channel_key].json.get('state') != 'DESTROYED':
                safe_hangup(resources[channel_key])
        if resources.get('bridge'):
            safe_bridge_destroy(resources['bridge'])
        del mapping_table[port]
        release_port(port)
    logging.info("Master cleanup for port {} completed.".format(port))
```

- Ensures that all resources (channels, sockets, bridges) associated with a specific port are cleaned up properly.

5. RTP Echo Server

```
python
Copy code
def run_rtp_echo_server(host, port, stop_event):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_address = (host, port)
    try:
        sock.bind(server_address)
    except Exception as e:
        logging.error("Failed to bind socket on port {}: {}".format(port,
e))
        master_cleanup(port, "Socket bind failure")
        return
    try:
        while not stop_event.is_set():
            sock.settimeout(1.0)
            try:
                data, address = sock.recvfrom(2048)
                if data:
```

```
                    sock.sendto(data, address)
            except socket.timeout:
                continue
            except Exception as e:
                if stop_event.is_set():
                    break
                logging.error("Error during receive/send on port {}:
{}".format(port, e))
                if isinstance(e, socket.error) and e.errno == 9:
                    break
    finally:
        sock.close()
    if port in mapping_table:
        mapping_table[port]['sock'] = sock
        if port in mapping_table and
mapping_table[port].get('external_channel') is None:
            master_cleanup(port, "Socket closed")
    else:
        logging.error("No mapping table entry for port {}".format(port))
```

- Implements a simple RTP echo server that binds to a UDP port, receives data, and sends it back to the sender.

6. Event Handling

```
python
Copy code
def handle_channel_event(channel, event_type):
    channel_id = channel.id
    for port, resources in mapping_table.items():
        if channel_id in [getattr(resources.get(key), 'id', None) for key
in ['inbound_channel', 'dialed_channel', 'external_channel']]:
            if event_type == 'StasisEnd':
                if resources.get('external_channel') and
resources['external_channel'].id == channel_id:
                    if not resources.get('external_media_hangup_by_script',
False):
                        logging.info("ExternalMedia channel {} hung up
unexpectedly".format(channel_id))
                        resources['external_channel'] = None
                else:
                    master_cleanup(port, "Channel {} hung
up".format(channel_id))
            break
```

- Handles channel events, particularly focusing on the `StasisEnd` event to clean up resources.

7. Stasis Application Callbacks

```
python
Copy code
def stasis_start_cb(channel_obj, ev):
    channel = channel_obj.get('channel')
    args = ev.get('args')
    if args and args[0] == 'inbound':
        thread = threading.Thread(target=main_call_flow,
args=(channel_obj,))
        thread.start()
```

```
def stasis_end_cb(channel, ev):
    handle_channel_event(channel, 'StasisEnd')
```

- Defines callbacks for Stasis events to initiate the main call flow and handle channel endings.

8. Main Call Flow

```
def main_call_flow(channel_obj):
    channel = channel_obj.get('channel')
    free_port = get_free_port()
    try:
        external_host = "127.0.0.1:{}".format(free_port)
        external_channel = client.channels.externalMedia(app='voicebot1',
external_host=external_host, format='alaw')
        stop_event = threading.Event()
        thread = threading.Thread(target=run_rtp_echo_server,
args=("0.0.0.0", free_port, stop_event))
        thread.start()
        bridge = client.bridges.create(type='mixing')
        bridge_id = bridge.id
        mapping_table[free_port] = {
            'thread': thread,
            'stop_event': stop_event,
            'external_channel': external_channel,
            'inbound_channel': channel,
            'bridge': bridge,
            'bridge_id': bridge_id,
            'dialed_channel': None,
            'external_media_hangup_by_script': False
        }
        bridge.addChannel(channel=[channel.id, external_channel.id])
        time.sleep(7)
        mapping_table[free_port]['external_media_hangup_by_script'] = True
        safe_hangup(external_channel)
        mapping_table[free_port]['external_media_hangup_by_script'] = False
        outbound_channel =
client.channels.originate(endpoint='Local/4438007', app='voicebot1',
appArgs='dialed', formats='alaw')

        def outbound_start_cb(outbound_channel_obj, ev):
            outbound_channel = outbound_channel_obj.get('channel')
            logging.info("{} answered; bridging with
{}".format(outbound_channel.json.get('name'), channel.json.get('name')))
            try:
                bridge = client.bridges.get(bridgeId=bridge_id)
                bridge.addChannel(channel=[channel.id,
outbound_channel.id])
                mapping_table[free_port]['dialed_channel'] =
outbound_channel
            except requests.exceptions.HTTPError as e:
                logging.error("Error adding channel to bridge:
{}".format(e))
                master_cleanup(free_port, "Bridge not found")

        outbound_channel.on_event('StasisStart', outbound_start_cb)
    except requests.HTTPError as e:
        logging.error("HTTP error: {}".format(e))
        master_cleanup(free_port, "HTTP error in main call flow")
```

- Manages the entire call flow, including setting up the external media, running the echo server, and handling outbound calls.