GB28181 Restreamer - Complete Documentation

Overview

GB28181 Restreamer is a comprehensive video streaming solution that implements the GB28181 surveillance standard protocol. It enables streaming of video content from various sources (RTSP streams, video files, or live camera feeds) to GB28181-compatible platforms like WVP-Pro using SIP signaling and RTP streaming.

Key Features

- **GB28181 Protocol Compliance**: Full support for device registration, catalog queries, and media streaming
- **Multiple Streaming Modes**: File-based streaming, RTSP source streaming, and real-time frame processing
- **Advanced Video Processing**: Appsink/Appsrc pipeline for real-time frame manipulation
- **Time Series Recording**: Recording management with historical video playback capabilities
- **Flexible Configuration**: JSON-based configuration with multiple preset options
- **Robust Health Monitoring**: Automatic stream recovery and health monitoring
- **Multi-stream Support**: Handle multiple concurrent video streams
- **ARM Architecture Support**: Optimized for ARM64 systems including Raspberry Pi

Table of Contents

1. System Requirements
2. Installation
3. PJSUA Setup
4. Configuration
5. Usage
6. Advanced Features
7. Troubleshooting
8. API Reference

System Requirements

Minimum Requirements

- **Operating System**: Ubuntu 20.04+, Debian 11+, or Raspberry Pi OS (64-bit)
- **Architecture**: x86_64 or ARM64 (aarch64)
- **RAM**: 2GB minimum, 4GB recommended
- **Storage**: 10GB free space minimum
- **Network**: Stable internet connection
- **Python**: 3.8 or higher

Required System Packages

```
# Core dependencies
sudo apt update
sudo apt install -y python3 python3-pip python3-dev
sudo apt install -y pkg-config libcairo2-dev gcc python3-dev libgirepository1.0-dev

# GStreamer dependencies
sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
sudo apt install -y libgstreamer-plugins-good1.0-dev libgstreamer-plugins-bad1.0-dev
sudo apt install -y gstreamer1.0-plugins-base gstreamer1.0-plugins-good
sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly
sudo apt install -y gstreamer1.0-libav gstreamer1.0-tools gstreamer1.0-x

# Additional video processing tools
sudo apt install -y ffmpeg v4l-utils

# Development tools (if compiling from source)
sudo apt install -y build-essential cmake git
```

Installation

Quick Installation

1. **Clone the Repository** `bash git clone https://github.com/your-org/gb28181-restreamer.git cd gb28181-restreamer`
2. **Install Python Dependencies** `bash pip3 install -r requirements.txt`

3. **Install PJSUA** (See detailed instructions below)
4. **Configure the Application** `bash cp config/config.json config/my_config.json # Edit config/my_config.json with your settings`
5. **Run the Application** `bash python3 src/main.py`

PJSUA Setup

PJSUA is a critical component for SIP communication in GB28181. Follow these detailed instructions based on your system architecture.

For x86_64 Systems (Intel/AMD)

Option 1: Package Manager Installation (Recommended)

```
# Ubuntu/Debian
sudo apt update
sudo apt install -y pjsip-tools

# Verify installation
pjsua --help
```

Option 2: Build from Source

```
# Install dependencies
sudo apt install -y build-essential libasound2-dev

# Download and compile PJSIP
wget https://github.com/pjsip/pjproject/archive/2.14.1.tar.gz
tar -xzf 2.14.1.tar.gz
cd pjproject-2.14.1

# Configure and build
./configure --enable-shared --disable-video --disable-opencore-amr
make dep && make
sudo make install
sudo ldconfig

# Verify installation
pjsua --help
```

For ARM64 Systems (Raspberry Pi, ARM servers)

Installing PJSUA on ARM64

ARM64 systems often require building PJSUA from source due to limited package availability.

Step 1: System Preparation

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install build dependencies
sudo apt install -y build-essential
sudo apt install -y libasound2-dev libssl-dev
sudo apt install -y autoconf automake libtool
sudo apt install -y pkg-config cmake

# For Raspberry Pi specifically
sudo apt install -y libraspberrypi-dev
```

Step 2: Download and Build PJSIP

```
# Create build directory
mkdir -p ~/build
cd ~/build

# Download PJSIP source (use stable version)
wget https://github.com/pjsip/pjproject/archive/refs/tags/2.14.1.tar.gz
tar -xzf 2.14.1.tar.gz
cd pjproject-2.14.1

# Configure for ARM64
export CFLAGS="-O2"
export CXXFLAGS="-O2"

# Configure with ARM optimizations
./configure \
    --host=aarch64-linux-gnu \
    --enable-shared \
    --disable-video \
    --disable-opencore-amr \
    --disable-silk \
    --disable-opus \
    --disable-speex-codec \
    --disable-ilbc-codec \
    --disable-l16-codec \
    --disable-gsm-codec \
    --disable-g722-codec \
    --disable-g7221-codec \
    --disable-speex-aec \
    --enable-epoll \
    --prefix=/usr/local

# Build (this may take 20-30 minutes on Raspberry Pi)
make dep
make -j$(nproc)
sudo make install

# Update library path
echo '/usr/local/lib' | sudo tee /etc/ld.so.conf.d/pjsip.conf
sudo ldconfig

# Create symlink for easier access
sudo ln -sf /usr/local/bin/pjsua /usr/bin/pjsua
```

Step 3: Verify Installation

```
# Test PJSUA installation
pjsua --help

# Check library linking
ldd /usr/local/bin/pjsua
```

Step 4: Configure for GB28181

```
# Test SIP registration
pjsua --id "sip:test@your-sip-server.com" \
      --registrar "sip:your-sip-server.com:5060" \
      --username "test" \
      --password "password" \
      --auto-answer 200 \
      --log-level 4
```

Alternative PJSUA Installation Methods

Using Docker (All Architectures)

If compilation fails, you can use Docker:

```
# Create a PJSUA container
docker run -it --rm \
  -v $(pwd):/workspace \
  --name pjsua-builder \
  ubuntu:22.04 bash

# Inside container
apt update && apt install -y build-essential wget
wget https://github.com/pjsip/pjproject/archive/2.14.1.tar.gz
tar -xzf 2.14.1.tar.gz
cd pjproject-2.14.1
./configure --enable-shared --disable-video
make dep && make
cp pjsip-apps/bin/pjsua-* /workspace/pjsua
exit

# Make executable
chmod +x pjsua
sudo mv pjsua /usr/local/bin/
```

Pre-built ARM64 Binaries

```
# Download pre-built binary (if available)
wget https://github.com/your-org/pjsua-arm64/releases/download/v2.14.1/pjsua-arm64
chmod +x pjsua-arm64
sudo mv pjsua-arm64 /usr/local/bin/pjsua
```

PJSUA Configuration for GB28181

Create a PJSUA configuration file for your GB28181 setup:

```
# Create PJSUA config directory
mkdir -p ~/.pjsua

# Create configuration file
cat > ~/.pjsua/gb28181.conf << EOF
--id sip:YOUR_DEVICE_ID@YOUR_SIP_SERVER:5060
--registrar sip:YOUR_SIP_SERVER:5060
--username YOUR_DEVICE_ID
--password YOUR_PASSWORD
--realm *
--local-port 5080
--auto-answer 200
--log-level 4
--app-log-level 4
--duration 0
--null-audio
--no-vad
EOF
```

Configuration

Basic Configuration

The main configuration file is `config/config.json`. Here's a complete example:

```
{
  "sip": {
    "device_id": "81000000465001000001",
    "username": "81000000465001000001",
    "password": "admin123",
    "server": "your-gb28181-server.com",
    "port": 5060,
    "local_port": 5080,
    "transport": "udp"
  },
  "local_sip": {
    "enabled": false,
    "port": 5060,
    "transport": "udp"
  },
  "stream_directory": "/path/to/recordings",
  "rtsp_sources": [
    {
      "id": 1,
      "name": "Camera 1",
      "url": "rtsp://192.168.1.100:554/stream1",
      "username": "admin",
      "password": "password123"
    }
  ],
  "srtp": {
    "key": "313233343536373839303132333435363132333435363738393031323334"
  },
  "logging": {
    "level": "INFO",
    "file": "./logs/gb28181-restreamer.log",
    "console": true
  },
  "pipeline": {
    "format": "RGB",
    "width": 1920,
    "height": 1080,
    "framerate": 25,
    "buffer_size": 33554432,
    "queue_size": 3000,
    "sync": false,
    "async": false
  }
}
```

Configuration Parameters

SIP Configuration

- **device_id**: Your unique GB28181 device identifier (20 digits)
- **username**: SIP username (usually same as device_id)
- **password**: SIP password for authentication
- **server**: GB28181 platform server address
- **port**: SIP server port (usually 5060)
- **local_port**: Local SIP port (recommended: 5080)
- **transport**: Transport protocol (udp/tcp)

Streaming Configuration

- **stream_directory**: Directory for storing/reading video files
- **rtsp_sources**: Array of RTSP source configurations
- **pipeline**: GStreamer pipeline parameters

Encoding Presets

The system includes predefined encoding presets in `config/streaming_presets.json`:

```
{
  "presets": {
    "high_quality": {
      "width": 1920,
      "height": 1080,
      "framerate": 25,
      "bitrate": 4000,
      "keyframe_interval": 25
    },
    "medium_quality": {
      "width": 1280,
      "height": 720,
      "framerate": 15,
      "bitrate": 2000,
      "keyframe_interval": 15
    },
    "low_bandwidth": {
      "width": 640,
      "height": 480,
      "framerate": 10,
      "bitrate": 500,
      "keyframe_interval": 10
    },
    "mobile_optimized": {
      "width": 480,
      "height": 320,
      "framerate": 10,
      "bitrate": 200,
      "keyframe_interval": 10
    }
  }
}
```

Usage

Basic Streaming

1. Start the Application

```
python3 src/main.py
```

2. File-based Streaming

```
from media_streamer import MediaStreamer

config = {...}  # Your configuration
streamer = MediaStreamer(config)

# Start streaming a video file
success = streamer.start_stream(
    video_path="/path/to/video.mp4",
    dest_ip="192.168.1.100",
    dest_port=9000,
    ssrc="1234567890"
)
```

3. RTSP Source Streaming

Configure RTSP sources in your config file and they will be automatically available as GB28181 channels.

Advanced Features

Appsink/Appsrc Frame Processing

The system supports real-time frame processing using GStreamer's appsink and appsrc elements:

```
def custom_frame_processor(frame, timestamp, stream_info):
    """
    Custom frame processing function

    Args:
        frame (numpy.ndarray): Video frame in RGB format
        timestamp (float): Frame timestamp
        stream_info (dict): Stream information

    Returns:
        tuple: (processed_frame, timestamp)
    """
    # Example: Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    processed = cv2.cvtColor(gray, cv2.COLOR_GRAY2RGB)

    return processed, timestamp

# Start stream with processing
streamer.start_stream_with_processing(
    video_path="/path/to/video.mp4",
    dest_ip="192.168.1.100",
    dest_port=9000,
    frame_processor_callback=custom_frame_processor
)
```

Available Frame Processors

1. **Grayscale Conversion** `python def grayscale_processor(frame, timestamp, stream_info): gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY) return cv2.cvtColor(gray, cv2.COLOR_GRAY2RGB), timestamp`
2. **Edge Detection** `python def edge_detection_processor(frame, timestamp, stream_info): gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY) edges = cv2.Canny(gray, 100, 200) return cv2.cvtColor(edges, cv2.COLOR_GRAY2RGB), timestamp`
3. **Text Overlay** `python def text_overlay_processor(frame, timestamp, stream_info): frame_copy = frame.copy() timestamp_str = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(timestamp)) cv2.putText(frame_copy, f"Time: {timestamp_str}", (20, 40), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2) return frame_copy, timestamp`

Testing Appsink/Appsrc Mode

Use the provided test script:

```
 # Test basic frame processing
python3 test_appsink_appsrc.py --video /path/to/test.mp4 --processing grayscale

# Test with custom destination
python3 test_appsink_appsrc.py --video /path/to/test.mp4 --dest-ip 192.168.1.100 --dest-port 9000

# Test edge detection
python3 test_appsink_appsrc.py --video /path/to/test.mp4 --processing edge

# Test text overlay
python3 test_appsink_appsrc.py --video /path/to/test.mp4 --processing text
```

Time Series Recording and Playback

The system supports GB28181 RecordInfo queries for historical video access:

```
from recording_manager import RecordingManager

# Initialize recording manager
config = {...}
recorder = RecordingManager(config)

# Query recordings by time range
recordings = recorder.query_recordings(
    device_id="81000000465001000001",
    start_time="20240101T000000Z",
    end_time="20240131T235959Z",
    recording_type="all"
)

# Start playback of a specific recording
for recording in recordings:
    streamer.start_recording_playback(
        recording_info=recording,
        dest_ip="192.168.1.100",
        dest_port=9000,
        start_timestamp=recording["timestamp"],
        end_timestamp=recording["timestamp"] + recording["duration"]
    )
```

Directory Structure for Recordings

Organize your recordings for optimal time series query performance:

```
recordings/
├── 2024-01-01/
│   ├── 12-00-00.mp4
│   ├── 12-30-00.mp4
│   └── 13-00-00.mp4
├── 2024-01-02/
│   ├── 09-15-30.mp4
│   └── 14-45-00.mp4
└── metadata.json
```

Troubleshooting

Common Issues and Solutions

1. PJSUA Installation Issues

Problem: `Cannot install pjsip-tools / pjsua`

Solution for ARM64:

```
# If package installation fails, build from source
sudo apt install -y build-essential libasound2-dev
wget https://github.com/pjsip/pjproject/archive/2.14.1.tar.gz
tar -xzf 2.14.1.tar.gz
cd pjproject-2.14.1
./configure --enable-shared --disable-video
make dep && make
sudo make install
sudo ldconfig
```

Solution for older Ubuntu/Debian:

```
# Add universe repository
sudo add-apt-repository universe
sudo apt update
sudo apt install -y pjsip-tools
```

2. GStreamer Issues

Problem: `Cannot read rtsp stream, Gstreamer error: Internal data stream error`

Solutions:

1. **Install additional GStreamer plugins**: `bash sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly sudo apt install -y gstreamer1.0-libav`
2. **Check RTSP source compatibility**: `bash # Test RTSP stream directly gst-launch-1.0 rtspsrc location="rtsp://your-camera-ip:554/stream" ! decodebin ! videoconvert ! autovideosink`
3. **Update pipeline configuration**: `json { "pipeline": { "format": "RGB", "width": 640, "height": 480, "framerate": 15, "buffer_size": 16777216, "sync": false } }`

3. ARM Architecture Issues

Problem: `RTSP server doesn't work on ARM (rtsp-simple-server is amd64)`

Solution - Use MediaMTX (ARM64 compatible):

```
# Download ARM64 version of MediaMTX (successor to rtsp-simple-server)
wget https://github.com/bluenviron/mediamtx/releases/download/v1.5.0/mediamtx_v1.5.0_linux_arm64v8.tar.gz
tar -xzf mediamtx_v1.5.0_linux_arm64v8.tar.gz
chmod +x mediamtx

# Create configuration
cat > mediamtx.yml << EOF
rtspAddress: :8554
rtmpAddress: :1935
webrtcAddress: :8889
api: yes
apiAddress: :9997
paths:
  all:
    source: publisher
EOF

# Run MediaMTX
./mediamtx mediamtx.yml
```

Alternative - Use GStreamer RTSP Server:

```
# Install GStreamer RTSP server
sudo apt install -y libgstrtspserver-1.0-dev

# Use python gst-rtsp-server for ARM
pip3 install gst-rtsp-server
```

4. Registration Issues

Problem: `WVP-pro device registration issues`

Solutions:

1. **Check SIP configuration**: `bash # Test SIP registration manually pjsua --id "sip:YOUR_DEVICE_ID@YOUR_SERVER:5060" \ --registrar "sip:YOUR_SERVER:5060" \ --username "YOUR_DEVICE_ID" \ --password "YOUR_PASSWORD" \ --duration 30`
2. **Verify network connectivity**: ```bash # Check if SIP server is reachable telnet YOUR_SERVER 5060

```
# Check UDP connectivity
nc -u YOUR_SERVER 5060
```
```
```

3. **Check device ID format**: - Must be exactly 20 digits - First 8 digits: Administrative region code - Next 2 digits: Device type (01 for camera) - Last 10 digits: Device serial number

5. Video Catalog Issues

Problem: `Videos not showing in WVP catalog`

Solutions:

1. **Verify catalog response**: `bash # Check if catalog XML is generated ls -la catalog_response.xml cat catalog_response.xml`
2. **Check channel configuration**: `python # Ensure channels are properly configured { "rtsp_sources": [ { "id": 1, "name": "Camera 1", "url": "rtsp://192.168.1.100:554/stream1" } ] }`

6. Performance Issues on ARM

Problem: High CPU usage or poor performance on ARM devices

Solutions:

1. **Optimize encoding settings**: `json { "pipeline": { "width": 640, "height": 480, "framerate": 15, "bitrate": 1000 } }`
2. **Enable hardware acceleration** (Raspberry Pi): ```bash # Enable GPU memory split sudo raspi-config # Advanced Options -> Memory Split -> 128

```
# Use hardware encoder
export GST_DEBUG=3
gst-inspect-1.0 | grep omx
```

3. **Use lower quality presets**: `python encoder_params = { "width": 480, "height": 320, "framerate": 10, "bitrate": 200 }`

Debugging and Logging

Enable Debug Logging

```
# In your configuration
{
  "logging": {
    "level": "DEBUG",
    "file": "./logs/debug.log",
    "console": true
  }
}
```

GStreamer Debug

```
# Set GStreamer debug level
export GST_DEBUG=3

# Debug specific elements
export GST_DEBUG=rtspsrc:5,rtpbin:5

# Run with debug
python3 src/main.py
```

PJSUA Debug

```
# Run with verbose logging
pjsua --log-level 5 --app-log-level 5 --log-file pjsua_debug.log
```

Network Debugging

```
# Monitor SIP traffic
sudo tcpdump -i any -n port 5060 -A

# Monitor RTP traffic
sudo tcpdump -i any -n portrange 10000-20000

# Check port usage
sudo netstat -tulpn | grep python3
```

API Reference

MediaStreamer Class

Methods

`start_stream(video_path, dest_ip, dest_port, ssrc=None, encoder_params=None)`

Start streaming a video file.

Parameters:

- `video_path` (str): Path to video file
- `dest_ip` (str): Destination IP address
- `dest_port` (int): Destination port
- `ssrc` (str, optional): RTP SSRC identifier
- `encoder_params` (dict, optional): Encoding parameters

Returns: `bool` - Success status

`start_stream_with_processing(video_path, dest_ip, dest_port, frame_processor_callback=None, ssrc=None, encoder_params=None)`

Start streaming with real-time frame processing.

Parameters:

- `frame_processor_callback` (function): Frame processing function
- Other parameters same as `start_stream`

Returns: `bool` - Success status

`stop_stream(stream_id=None)`

Stop a specific stream or all streams.

Parameters:

- `stream_id` (str, optional): Stream identifier

`get_stream_status(stream_id=None)`

Get status of streams.

Returns: `dict` - Stream status information

RecordingManager Class

Methods

`query_recordings(device_id, start_time, end_time, recording_type=None, secrecy=None, max_results=100)`

Query recordings by time range.

Parameters:

- `device_id` (str): Device identifier
- `start_time` (str): Start time (ISO format or GB28181 format)
- `end_time` (str): End time
- `recording_type` (str, optional): Recording type filter
- `secrecy` (str, optional): Secrecy level filter
- `max_results` (int): Maximum results to return

Returns: `list` - List of recording metadata

`get_recording_stream_uri(recording_id)`

Get streaming URI for a recording.

Parameters:

- `recording_id` (str): Recording identifier

Returns: `str` - Streaming URI

Performance Optimization

For ARM Devices

1. **Use appropriate quality settings**: `json { "pipeline": { "width": 640, "height": 480, "framerate": 15, "bitrate": 1000 } }`
2. **Enable hardware acceleration** (when available): `bash # Check for hardware encoders gst-inspect-1.0 | grep -i "h264\|h265"`
3. **Optimize buffer sizes**: `json { "pipeline": { "buffer_size": 16777216, "queue_size": 1000 } }`

Memory Management

- Monitor memory usage: `htop` or `free -h`
- Limit concurrent streams on low-memory devices
- Use appropriate buffer sizes for your hardware

Network Optimization

- Use UDP transport for better performance
- Configure appropriate RTP port ranges
- Monitor network bandwidth usage

Support and Contributing

Getting Help

1. Check the troubleshooting section above
2. Review log files in `./logs/`
3. Enable debug logging for detailed diagnostics
4. Check GStreamer installation: `gst-inspect-1.0 --version`

Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Test on both x86_64 and ARM64 architectures
5. Submit a pull request

Reporting Issues

When reporting issues, please include:

- System architecture (`uname -a`)
- Python version (`python3 --version`)
- GStreamer version (`gst-inspect-1.0 --version`)
- PJSUA version (`pjsua --version`)
- Complete error logs
- Configuration file (with sensitive data removed)

License

Last Updated: May 2025 **Version**: 2.0.0 **Compatible Architectures**: x86_64, ARM64