

Server tuning recommendations

- [REST client tuning](#)
 - [Excessive logging suppression](#)
- [UDP tuning](#)
- [Channel load optimization](#)
 - [Publisher bitrate limiting](#)
 - [Server bitrate limiting](#)
- [Changing dynamic ports range in Linux](#)
- [Adjusting the maximum number of opened files](#)
 - [Legacy settings \(before build5.2.762\)](#)
 - [Using environment variable \(since build5.2.762\)](#)
 - [Using service parameter while launching from non-root user \(since build5.2.801\)](#)
- [Traffic encryption in a separate thread for each client session](#)
- [Stream distribution optimization](#)

Server default settings are mostly universal and need to be tuned to certain client case.

REST client tuning

When [REST hooks](#) are used, on every WCS server action (establishing client connection, publishing and playing a stream, making a SIP call etc) HTTP REST connection to backend server is established. With a large number of simultaneously publishing clients or subscribers, with the default WCS settings it is possible to exhaust the WCS REST client thread pool, that leads to deadlocks. Then, server stops to publish and play streams.

By default, a maximum number of simultaneous REST connections is set to 200 with the following parameter in [flashphoner.properties](#) file

```
rest_max_connections=200
```

To escape thread pool exhausting and deadlocks this value should be reduced, for example

```
rest_max_connections=20
```

If [REST hooks](#) are not used, REST client can be disabled with the following parameter

```
disable_rest_requests=true
```

Excessive logging suppression

When [REST hooks](#) are used, REST client operations, EchoApp default backend operations and REST API server operations are written to WCS core logs. That leads to large number of entries in the log file and, therefore, increases the server load. The excessive logging may be decreased if necessary using the following parameters in [log4j.properties](#) file:

```
log4j.logger.RestClient=WARN
log4j.logger.EchoApp=WARN
log4j.logger.RestApiRouter=WARN
```

UDP tuning

Streaming mediadata are transferred with UDP packets. Those packets can be dropped, for example if server does not have enough time to parse packet queue, that leads to picture quality loss and freezes. To escape this it is necessary to tune UDP sockets buffers with the following settings in [flashphoner.properties](#) file

```
rtp_receive_buffer_size=131072
rtp_send_buffer_size =131072
```

and to tune system queues with command

```
ip link set txqueuelen 2000 dev eth0
```

To diagnose UDP problem, it is necessary to track UDP packets dropping with command

```
dropwatch -l kas
>start
```

Channel load optimization

Users' playback picture quality depends on bitrate: the higher the bitrate, the higher the quality. However, the higher the bitrate, the higher data transfer channel load and, if the bandwidth between the server and clients is limited, there is a possibility that the channel will be fully loaded. This leads to the bitrate dropping and a sharp decline in quality.

In this regard, it is necessary to limit the bitrate to ensure sufficient picture quality with an acceptable channel load.

Publisher bitrate limiting

To reduce the load to the channel from publisher to server, maximum and minimum bitrate values in kbps may be set in publisher script with JavaScript API

```
session.createStream({
  name: streamName,
  display: localVideo,
  constraints: {
    video: {
      minBitrate: 500
      maxBitrate: 1000
    }
  }
  ...
}).publish();
```

Server bitrate limiting

Minimum and maximum bitrate values in bps on server may be set with the following parameters in [flashphoner.properties](#) file

```
webrtc_cc_min_bitrate=500000
webrtc_cc_max_bitrate=1000000
```

To exclude fast bitrate rise buy browser, the following parameter should be set

```
webrtc_cc2_twcc=false
```

Stream decoding on demand only must be switched on to reduce server load:

```
streaming_video_decoder_fast_start=false
```

Changing dynamic ports range in Linux

Dynamic or ephemeral port is a temporary port that is opened when establishing IP-connection from certain range of TCP/IP stack. Many Linux kernel versions use ports range 32768 — 61000 as dynamic ports. Enter the following command to check what range is used on server

```
sysctl net.ipv4.ip_local_port_range
```

If this range overlaps with WCS standard [ports](#), it should be changed with the following command

```
sysctl -w net.ipv4.ip_local_port_range="59999 63000"
```

Adjusting the maximum number of opened files

Legacy settings (before build [5.2.762](#))

In the launch script `webcallserver` that is in subfolder `bin` in WCS home folder, for example

```
/usr/local/FlashphonerWebCallServer/bin/webcallserver
```

in `start()` function the maximum number of opened files is set

```
function start() {
    ...
    echo -n "$PRODUCT: starting"

    ulimit -n 20000
    if [[ "$1" == "standalone" ]]; then
        ...
    fi
    ...
}
```

By default, this value is set to 20000, but it may be increased if necessary, following the limitations of the operating system used.

Using environment variable (since build [5.2.762](#))

Since build [5.2.762](#), maximum opened files limit can be set using the following environment variable

```
WCS_FD_LIMIT=20000
```

in `setenv.sh` file. When updating WCS from previous builds, this variable should be added to `setenv.sh` manually, for example

```
export WCS_FD_LIMIT=100000
```

Unlike the `webcallserver` startup script, the `setenv.sh` file is not overwritten on subsequent updates, therefore it is not necessary to restore this setting after every update.

Using service parameter while launching from non-root user (since build [5.2.801](#))

Since build [5.2.801](#), WCS is launching from 'flashphoner' user for better security. In this case, maximum opened files limit can be set using service parameters

```
sudo nano /etc/systemd/system/webcallserver.service
```

Maximum opened files limit is set with `LimitNOFILE` parameter, for example

```
[Service]
User=flashphoner
Group=flashphoner
LimitNOFILE=100000
...
```

Traffic encryption in a separate thread for each client session

By default, one CPU thread encrypts media traffic for all the client sessions. This leads to one CPU core overload by such thread, especially on low-power servers, for big subscribers amount. Then, server can not send media packets to all subscribers, and streams viewed are degrading, FPS lowering and freezing.

To distribute the load evenly across the CPU cores, it is necessary to enable traffic encryption in a separate thread for each client session with the following parameters

```
rtp_paced_sender=true
rtp_paced_sender_initial_rate=200000
rtp_paced_sender_increase_interval=50
rtp_paced_sender_k_up=0.9
```

and restart WCS.

Stream distribution optimization

A stream playback quality may drop when a number of subscribers are viewing it simultaneously (from 100 and more): low FPS, freezes. However, server capacity and channel bandwidth may be enough. In this case it is recommended to enable multithreaded stream distribution to subscribers using the following parameter

```
streaming_distributor_subgroup_enabled=true
```

In this case, audio and video client sessions are distributed by groups.

Maximum number of video sessions per group can be set with the following parameter

```
streaming_distributor_subgroup_size=50
```

Maximum number of audio sessions per group can be set with the following parameter

```
streaming_distributor_audio_subgroup_size=500
```

Frame queue size per group and maximum frame waiting time (in milliseconds) are set by the following parameters

```
streaming_distributor_subgroup_queue_size=300
streaming_distributor_subgroup_queue_max_waiting_time=5000
```

for video and

```
streaming_distributor_audio_subgroup_queue_size=300
streaming_distributor_audio_subgroup_queue_max_waiting_time=5000
```

for audio sessions respectively.