A Real-Time Audio and Video Streaming Transmission Scheme for Social Media

Jianping Yu[1,3], Gang Zhao[2], Xiaohui Kuang[2], and Ruyun Zhang[4(✉)]

[1] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China
ppzepp@163.com

[2] National Key Laboratory of Science and Technology on Information System Security, Beijing 100093, China
zhao-gang20@126.com, xiaohui_kuang@163.com

[3] Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou 510006, China

[4] Zhejiang Lab, Hangzhou, Zhejiang, China
zhangry@zhejianglab.com

Abstract. Real-time video streaming transmission has been widely used in security monitoring field. However, audio function in the monitoring system has not attracted significant attention. As a well-known streaming media server in security monitoring field, Live555 projects have realized the real-time video capture and transmission with the secondary development, but they still lack the collection and transmission of real-time audio on IP camera. In this paper, we propose a streaming media transmission scheme to realize simultaneous transmission of real-time audio and video based on the Live555 project. In order to realize the above scheme, we add the real-time audio collection module, rewrite the related classes and methods in the project to transport real-time audio data with video. It is demonstrated by experiments that the real-time audio and video simultaneous transmission is stable, the delay is low and the quality is good. It can be played on mainstream players such as VLC and FFplay.

Keywords: Live555 real-time transmission · Audio and video · IP camera

1 Introduction

With the improvement of Internet of Things technology and people's concern about home security ecology, IP camera enters the field of home security from the professional field [1]. The new application scene brings new challenges. Most of IP cameras only support real-time video transmission, while in home, private and quiet environment requires IP camera can transport audio with video at the same time.

Live555 streaming media server plays an important role in security monitoring field [2]. However, the official source code only supports file streaming transmission rather than real-time audio and video transmission. Though there are already some secondary

Y. Xiang et al. (Eds.): SocialSec 2020, CCIS 1298, pp. 143–152, 2020.
https://doi.org/10.1007/978-981-15-9031-3_13

development Live555 projects to implement real-time video collection and transmission, audio is still a file transmission framework [3, 4].

In our paper, we propose a real-time audio and video streaming media transmission scheme for social media. Live555 project is based on the secondary development version which is in the SDK of Ambarella S2Lm chip. This version has achieved real-time video capture and transmission without real-time audio. So we add the real-time audio collection module to get real-time audio data, rewrite classes and methods related to audio to realize the real-time audio transmission, then add audio subsession to ServerMediaSession to merge audio with video. After cross-compiling and transplanting it to IP camera, we achieve normal real-time audio and video forwarding play. To summarize, we make the following main contributions:

- Real-time audio collection module is introduced to get real-time audio data.
- Classes and methods related to audio in Live555 are rewritten to realize the real-time audio transmission.

2 Related Work

The IP camera used in this paper has S2Lm processing chip produced by Ambarella [5], and is based on Live555 streaming media server, collects and transports H264 video and PCM audio in time. Live555 has a streamlined architecture and good portability, so it is easy to be used on multiple platforms through cross compilation, especially embedded systems [6].

However, the official source code of Live555 only supports file streaming transmission rather than real-time audio and video transmission. At present, there are two main solutions for Live555 real-time transmission. One is to use named pipe, the other is inheriting related classes and rewriting related methods. For the first method, it has been realized the real-time video transmission with named pipe [4]: use the mkfifo command to create a named pipe, then run the program, so that the collected real-time stream is continuously written into this FIFO [7], the Live555 server can run directly to see the real-time video. However, when the bitrate is large, the real-time video playing will have obvious jams and mosaics.

Therefore, the second method is commonly used, which is also the method we reference. By rewriting the relevant methods, the reading of the audio and video data is changed from files to memory, which avoids the overhead of reading and writing files by FIFO, Lu Shaojun [3] and others have initially implemented a real-time H264 streaming media transmission system by adding classes to LiveMedia library. The problems of delay and unstable data transmission in the Live555-based video transmission system has been solved [8], and the video transmission is smooth and stable. But including the researches mentioned above, most of the secondary development is done only for the streaming of h264 video data, they lack the attention on audio. Therefore, the goal of this article is to add the audio function of Live555, and to achieve the integration and simultaneous playback of audio and video.

3 Design and Implementations

The Live555 in Ambarella SDK has realized real-time video data collection and transmission. The BasicUsageEnvironment and BasicTaskScheduler are recreated separately in setup_streams. The real-time collection thread is added to store video data in a circular array, then enter doEventLoop to loop and wait for new client. When a client connects, RTSPClientSession class is created to process the client request [9]. A new subthread will be created in the process of interacting with client. In this subthread, enter doEventLoop to send real-time data with the independent BasicTaskScheduler object. The flow chart of real-time data is shown as Fig. 1:

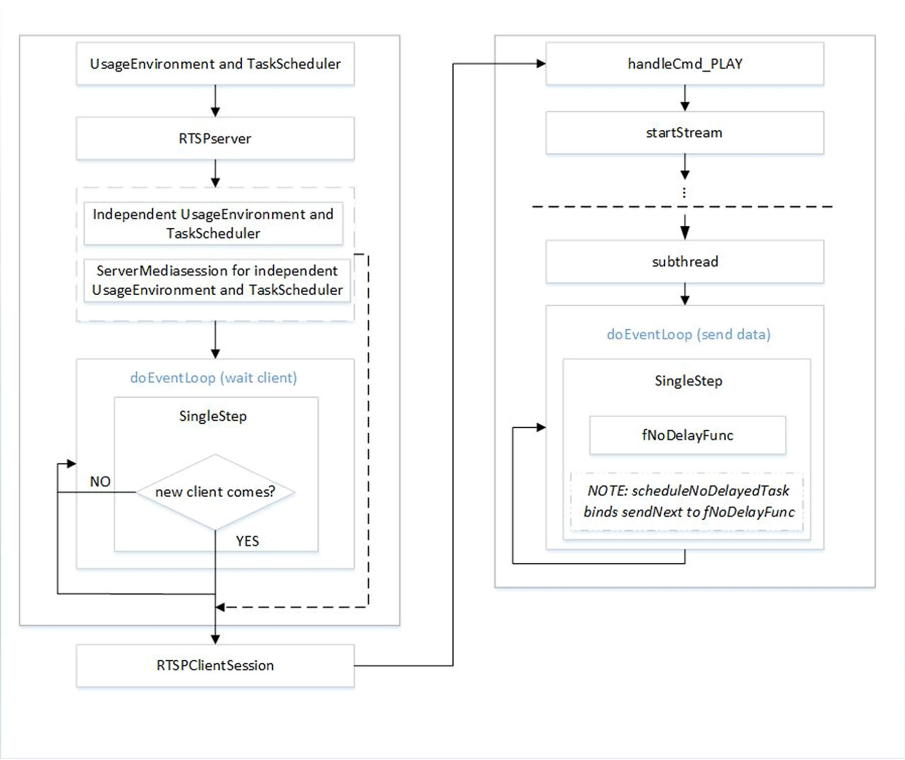

Fig. 1. Flow chart of real-time transmission in Live555.

3.1 Real-Time Audio Collection and Preparation

We use ALSA framework to achieve the collection of PCM audio [10], rewrite the WAVAudioFileSource and WAVAudioFileServerMediaSubsession to achieve the preparation of real-time audio. Key classes for audio are as shown in Table 1:

Table 1. Key classes for audio in LiveMedia

Class	Main functions
RTSPServer	Build RTSP server, create RTSPClientSession to handle individual client sessions
RTSPClient	Handle RTSP requests and responses, create RTP sessions
WAVAudioFileSource	Get PCM format audio data from buffers
SimpleRTPSink	Save and sent audio data to client
WAVAudioFileServerMediaSubsession	Save information of the streaming media, connect WAVAudioFileSource and SimpleRTPSink

Collection of Audio. We set audio format as SND_PCM_FORMAT_S16_LE, sample rate as 16000, the number of sample channels as 1. Then follow the ALSA audio acquisition process [11]. Formula for calculating the size of audio frame is as follows:

$$\text{FrameSize} = \text{sizeof (one sample)} * \text{nChannels} \tag{1}$$

So one frame occupies two bytes. We read chunk_size frames from the sound card one time, then store the audio data in buffer array buf_in, so we can read audio data from buffer other than sound card. The size of buf_in is calculated as follows:

$$\text{BufferSize} = \text{sizeof (FrameSize)} * \text{chunk\_size} \tag{2}$$

Preparation of Audio. PCM audio does not need to be encoded. So audio data stored in buf_in is read into fTo directly to wait to be consumed by Sink.

We assign empty data of the same length to fTo when the audio collection speed is lower than the reading speed. We also need to set the corresponding fFrameSize and calculate the corresponding playing time fDurationInMicroseconds. The relationship between audio capture thread and transport thread is as shown in Fig. 2:

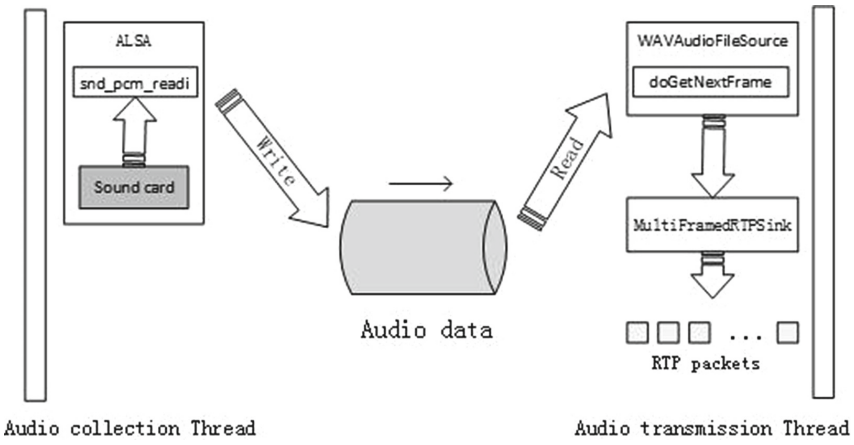

Fig. 2. The relationship between audio capture thread and transport thread.

3.2 Real-Time Audio Consumption

Consumption of real-time data is in the multiFramedRTPSink class actually. The process of packaging and sending is as follows:

In continuePlaying method, scheduleNoDelayedTask ((TaskFunc *) sendNext, this) use sendNext as the callback function to achieve the send task without delay. The live555 project on the S2Lm chip no longer uses the delay queue to send audio and video data. The scheduleDelayedTask method is no longer used, but the function ScheduleNoDelayedTask is recreated. The sendNext method calls BulidAndSendPacket to prepare the RTP header [12], and BulidAndSendPacket uses PackFrame to frame, PackFrame calls getNextFrame to continuously obtain data from Source, and sendPacketIfNecessary sends the data packet to the player. So far, the package is sent completely this time. Exit the singleStep function, enter doEventLoop and wait for the next packet sending process. This can effectively improve the efficiency of the server to send data, so as to obtain higher real-time and processing efficiency. This flow chart is as shown in Fig. 3:

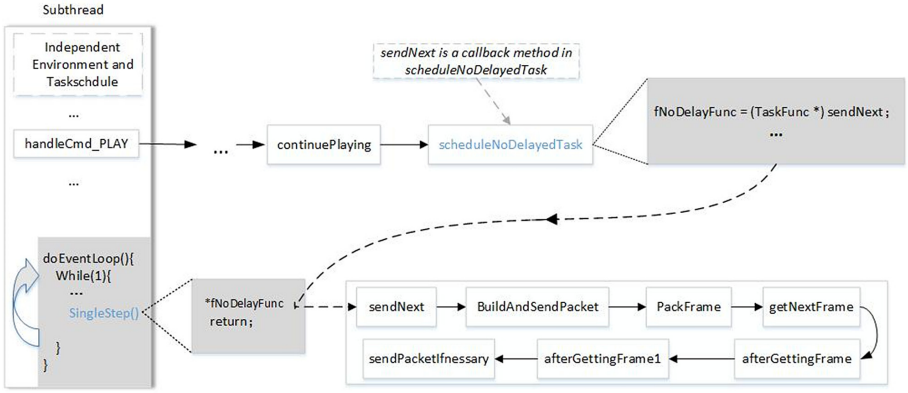

Fig. 3. Packaging and sending flowchart.

Create WAVAudioFileSource and SimpleRTPSink in the WAVAudioFileServerMediaSubsession class by implementing createNewStreamSource and createNewRTPSink, and set the corresponding audio parameters. Then add audio subsession into ServerMediaSession.

3.3 Redesign of Delayed Task Processing

Instead of using scheduleDelayedTask, ScheduleNoDelayedTask is called in continuePlaying method to deal with delayed tasks. ScheduleNoDelayedTask can deal task without delay. In order to have a deeper understanding of ScheduleNoDelayedTask, we compare the difference between ScheduleNoDelayedTask and delay queue [13].

The official live555 is a single-process, single-threaded server, but it can perfectly allow multiple clients to connect at the same time, delay queue is an important means [2]. ScheduleDelayedTask is called to add the transmitting work to the delay queue. In

singleStep, we need to actively go to the delay queue to check whether there is a timeout task, then execute the task of the timeout node, delete and synchronize the remaining time of the node in the queue [14].

This project improves to achieve multi-threaded concurrency. Each task thread creates independent TaskScheduler and UsageEnviroment classes, then executive doEventLoop independently to realize data processing. So the collected audio and video data can be continuously transmitted, and it is no longer necessary to call scheduleDelayedTask to add the sending work to the delay queue. Therefore, scheduleNoDelayedTask is introduced in the BasicTaskScheduler0 class to realize the transmission of real-time data immediately.

Because of the addition of audio stream, the data processing flow need to be refined. In the BasicUsageEnvironment0 class, there was originally only one pair of TaskFunc * fNoDelayFunc and void * fNoDelayClientData. This pair of member variables corresponds to the two parameters of scheduleNoDelayedTask, represents the nodelay task. If there is still only one pair of fNoDelayFunc and fNoDelayClientData, the latter caller will cover previous one. Therefore, add a new pair of member variables fNoDelayFunc2 and fNoDelayClientData2 to distinguish between audio and video; scheduleNoDelayedTask and singleStep in the BasicTaskScheduler class should handle the corresponding audio and video streaming tasks and data separately.

During the entire rtsp server operation process, multiple pairs of TaskScheduler and UsageEnviroment classes were created: one pair is mainly used to receive client requests and establish a connection with the client, the other pair is used to independently process audio and video data transmission. After one client connected, the main thread still calls select method to wait for the connection of new client, but it does not deal with the sending of the real-time data anymore—this task is processed in the subthreads created by audio and video subsessions, they send the real-time data in doEventLoop without any delay, and no longer handle the client's connection task. In fact, because we deal with the real-time data, unlike audio and video files, we no longer need to deal with operations such as pauses and fast forwards sent by the client, which can greatly improve the efficiency of the live555 server.

3.4 Thread Priority Setting

When creating the real-time transmission thread of audio, we noticed that the real-time transmission thread of video has a priority setting. In general, there are three kernel scheduling strategies in the Linux development environment [15]: SCHED_OTHER, SCHED_RR, SCHED_FIFO. The default priority of thread is SCHED_OTHER, which is allowed to be preempted by real-time tasks; SCHED_RR level threads are based on time slice rotation. When the time slice of a thread is reduced to 0, it will actively give up the CPU; For SCHED_FIFO level thread, once the CPU is occupied, it will continue to run until there is a higher priority task arrive or give up CPU on their own, so it will cause thread starvation [16].

The audio transmission thread we created here is a real-time thread, and it is not suitable for scheduling based on time slices—every time you send data, you must send all the data over in this time. You cannot give up data transmission just because the time slice is used up. Therefore, we set the audio transmission thread to the SCHED_FIFO

level with the same priority as the video transmission thread [17]. This will cause a problem that if the video transmission thread is called first, the video transmission thread will always occupy CPU. Therefore, we need to make the video transmission thread voluntarily give up CPU. Here we add a mutex to SingleStep. If the video thread gets the right to use the CPU first, after it executes the SingleStep method to release the lock, when it waits for the mutex resource in the next round, it will voluntarily give up the CPU use right, the thread task is deleted from the ready queue, and join the wait Queue; the next ready thread—the audio transmission thread will get the right to use the CPU, thereby achieving the concurrency of audio and video threads.

4 Evaluation

Use arm-linux-gnueabihf to cross compile the live555 project, transplant the executable file to the camera based on Ambarella platform, execute the script to start the service. The hardware parameters of IP camera are as shown in Table 2:

Table 2. hardware parameters of IP Camera

Nominal performance	Parameters
Hardware	Ambarella S2L
Operating system	Linux version 3.10.73
Crosstool	Ambarella Linaro Multilib GCC
CPU	ARMv7 Processor rev 1 (v7l)
RAM	103 MB

Enter the RTSP protocol playback address on the VLC client: rtsp://192.168.43.138/stream1, it can be played normally on player and audio codec information is as shown in Table 3.

Table 3. Media codec information on VLC

Audio codec	Parameters
Codec	PCM S16 BE (s16b)
Channels	Mono
Sample rate	16000 Hz
Bits per sample	16

We mainly evaluate this project from two aspects: system performance and real-time performance.

4.1 System Performance

For system performance, test CPU and memory usage of the camera during twelve hours. We first test the system resource occupation of the camera when only real-time video is captured and transmitted. Then add real-time audio to test the occupancy of system resources when audio and video are transmitted simultaneously, the result is as shown in Fig. 4:

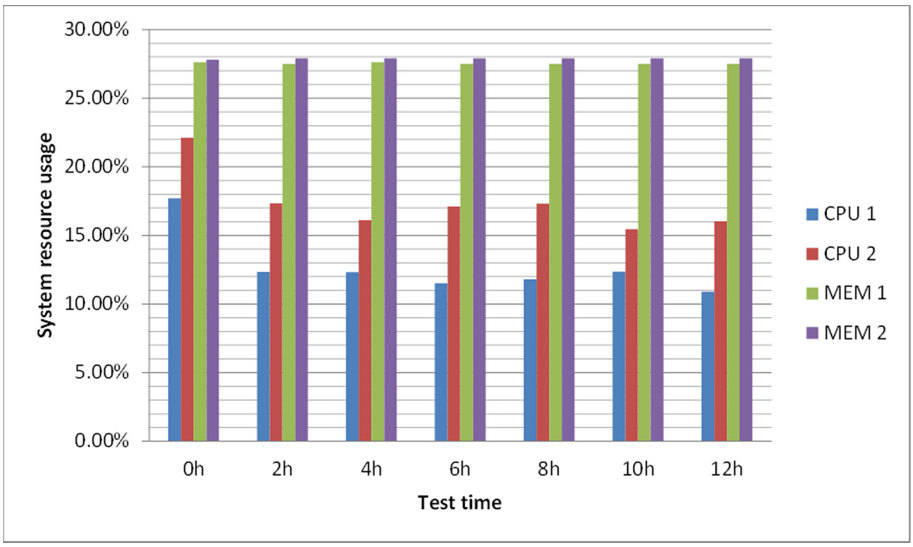

Fig. 4. System resource occupancy. CPU 1 means CPU usage when only video thread is working. CPU 2 means CPU usage when audio and video thread are working together. It is also suitable for MEM.

It can be seen from the above results that added audio collection and transmission threads can increase CPU occupancy rate, but even if the real-time audio and video collection and transmission threads are called at the same time, CPU resource utilization rate can be stabilized at about 20%; the CPU occupancy rate will not fluctuate greatly with time, indicating the thread are relatively stable. The proportion of memory space occupied by added audio thread is very small, only occupying about 0.2% more memory space; the memory occupancy rate is flat with time, which means that there is no memory leak in our project.

4.2 Real-Time Performance

For real-time testing, we use a stopwatch timer to intuitively test the delay of the Live555 project. We define the delay as the difference between the stopwatch time on the computer and the stopwatch time displayed on the player. In our LAN environment, the network download bandwidth can reach 5 MB/s and upload bandwidth can reach 1.4 MB/s; when

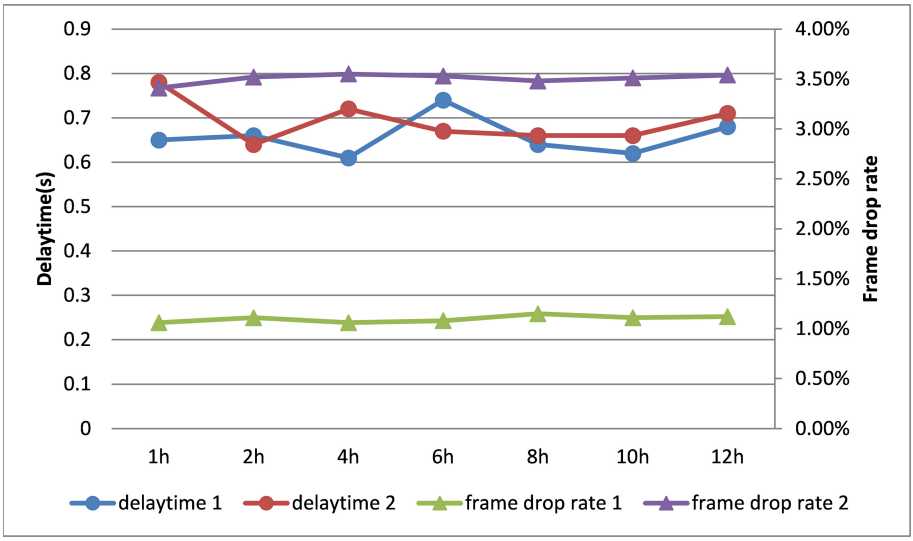

Fig. 5. Real-time performance. Frame drop rate 1 means frame drop rate when only video thread is working. Frame drop rate 2 means frame drop rate when audio and video thread are working together. It is also suitable for delaytime.

playing PCM audio and 1080p video, set the video code rate to 1200 kb/s. The frame drop rate and delay is as shown as Fig. 5:

From the results from above figure, delay can be controlled at about 0.6 s, real-time performance is good. Though frame drop rate has increased when the audio collection and transmission threads are added, the playback effect still has a good performance.

5 Conclusion

Most of the traditional real-time transformation schemes based on live555 project are only about video, there are few studies on the collection and transmission of real-time audio. In this paper, an audio collection module is added to get real-time audio data in Live555. Related classes are inherited and related methods are rewritten to realize the transmission of real-time audio and the synchronization of audio and video. We can use VLC player to play this stream in LAN environment. Besides the low delay, this Live555 can maintain long-term stable work. Experiments show that the real-time playback effect is good and can be used normally.

RTMP protocol is more widely used in the live broadcast industry [18, 19]. In future, we will build a cloud platform in the public network [20], try to convert RTSP protocol to RTMP protocol in the camera and push RTMP data to Public network to achieve forwarding.

Acknowledgment. This work was sponsored by Zhejiang Lab (NO. 2020LE0AB02).

References

1. Muruti, G., Rahim, F.A., Zawawi, N.A.: Motion activated security camera using raspberry Pi: an IoT solution for room security. Adv. Sci. Lett. **24**(3), 1698–1701 (2018)
2. LIVE555. http://live555.com/. Accessed 12 Apr 2020
3. Lv, S.-J., Zhou, Y.-P.: Real-time streaming media transmission system based on Live555. Jisuanji Xitong Yingyong – Comput. Syst. Appl. **24**(1), 56–59 (2015)
4. Huang, J., Yin, H.: An embedded multifunctional media system for mobile devices in terrestrial DTV relaying. J. Inf. Process. Syst. **14**(5), 1272–1285 (2018)
5. Ambarella. https://www.ambarella.com/. Accessed 25 Aug 2019
6. Huan, S., Li, P., Hao, W., Xuefang, Z.: Design of H.264 hard codec video transmission system based on ARM11. Comput. Meas. Control (2018)
7. Qian, J., Srisa-An, W., Seth, S., Hong, J., Pan, Y.: Exploiting FIFO scheduler to improve parallel garbage collection performance. ACM SIGPLAN Not. **51**(7), 109–121 (2016)
8. Wei, C.Y., Zhang, H.L.: Design and implementation of mobile real-time broadcast system based on Live555. Comput. Eng. Des. (2016)
9. Hao, F., Li, A., Liu, Y.: Design of embedded video monitoring system based on DM6437. Adv. Mater. Res. **1003**(01), 249–253 (2014)
10. ALSA. https://www.alsa.com/en/web/bus/home. Accessed 1 Sept 2019
11. Li, L., Mingjiang, W., Boya, Z., Anli, Y.: Design and implementation of embedded WM8960 audio driver and multi-thread player. In: The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings, pp. 292–295 (2015)
12. Zhang, X.-L., Zhou, Y.-P., Deng, C.-M.: Streaming media server based on android. Jisuanji Xitong Yingyong - Comput. Syst. Appl. **22**(12), 180–183 (2013)
13. Shibeshi, Z.S., Terzoli, A., Bradshaw, K.: An RTSP proxy for implementing the IPTV media function using a streaming server. Informatica **36**(1), 37–45 (2012)
14. Wei, C., Zhang, H.: Applications of a streaming video server in a mobile phone live streaming system. J. Softw. Eng. Appl. **7**(12), 975–982 (2014)
15. Kang, D., Lee, W., Park, C.: Kernel thread scheduling in real-time Linux for wearable computers. ETRI J. **29**(3), 270–280 (2007)
16. Choi, E., Jang, M., Kim, B.: Improving and evaluating real-time performance for smart devices. Int. Inf. Inst. (Tokyo) Inf. **16**(8), 5733–5743 (2013)
17. Hart, D., Stultz, J., Ts'o, T.: Real-time Linux in real time. IBM Syst. J. **47**(2), 207–220 (2008)
18. Aloman, A., Ispas, A.I., Ciotirnae, P., Sanchez-Iborra, R., Cano, M.D.: Performance evaluation of video streaming using MPEG DASH, RTSP, and RTMP in Mobile Networks. In: The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings, pp. 144–151 (2015)
19. Xue, Y., Tan, Y.-A., Liang, C., Li, Y., Zheng, J., Zhang, Q.: RootAgency: a digital signature-based root privilege management agency for cloud terminal devices. Inf. Sci. **444**, 36–50 (2018)
20. Zhang, Q., Gong, H., Zhang, X., Liang, C., Tan, Y.-A.: A sensitive network jitter measurement for covert timing channels over interactive traffic. Multimed. Tools Appl. **78**(3), 3493–3509 (2019)