Research Article

International Journal of
Distributed
Sensor Networks

International Journal of Distributed Sensor Networks
2016, Vol. 12(9)
© The Author(s) 2016
DOI: 10.1177/1550147716665514
ijdsn.sagepub.com
SAGE

Quality of experience provisioned mobile streaming protocol for the hyper-connected Internet of Things devices over the heterogeneous wireless access networks

Min Ho Park[1,2], Jun Kyun Choi[1] and JungYul Choi[3]

Abstract
5G-based hyper-connected Internet of Things environment is emerging, where various connectivities such as BlueTooth, WiFi, and 3/4/5G are provided. Since they have different communication characteristics, it is difficult to provide the user quality of experience when the handoff takes place according to the movement of the mobile devices between wireless access networks. For solving this problem, we propose a seamless video streaming protocol to satisfy the user quality of experience regardless of handoff. The proposed protocol can dynamically adjust streaming bit rate according to the network condition and also maintain fixed quantity of buffered audio–video data for the stable quality of experience level. The proposed protocol is designed independently of OS versions and CPU performance of terminals, which makes it easy to be implanted in various types of Internet of Things devices. Real-world experimental results show that seamless video streaming service can be guaranteed by carefully predicting the network dynamics over the heterogeneous wireless access networks. The protocol is now commercially deployed for over 5 million Long-Term Evolution subscribers of a South Korea telco, LG U + .

Keywords
Quality of experience, mobile streaming, wireless network, adaptive streaming, Internet of Things, 4G, 5G

Date received: 4 March 2016; accepted: 22 June 2016

Academic Editor: Mikhail Gofman

Introduction

Rapid increase in mobile and Internet of Things (IoT) devices and increased traffic accelerate the adoption of 5G networks. 5G network technology is being developed and standardized with the following three features: (1) high data rate, (2) low latency, and (3) massive connectivity.[11–14] Requirement for the high data rates should ensure peak transmission rates at 20 Gbps and guarantee user-experienced data rate at 100 Mbps.[11] Requirement for the low latency is to reduce one-way transmission delay in the air section to 1 ms. It makes it possible to support mission critical services such as autonomous car and tele-medicine.[11,13] Massive connectivity requirement for 5G networks should have the capacity to connect IoT devices up to 100 million/km^2.[11]

[1]Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea
[2]LG U + Research Institute of Technology, Daejeon, Republic of Korea
[3]Division of Computer Engineering, Sungkyul University, Anyang, Republic of Korea

Corresponding author:
Min Ho Park, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, Republic of Korea.
Email: minho.park@kaist.ac.kr

With the development of IoT technology, a number of sensors and consumer electronics in homes have the connectivity to the Internet.[16,19] Depending on the applications, IoT devices require different network access capability. Many IoT devices such as TVs, refrigerators, drones, and phones have multimedia transmission and reception functions. Devices with a large screen such as a TV and a smart-mirror have to be provided high bit rates to provide a high-quality image with a high resolution. Refrigerators or washing machines may require low bit rate for relatively low resolution. On the other hand, drones, vehicles, and wearable devices should be provided with adaptive bit rates by reflecting the network state due to the user's mobility. 5G networks are expected to be present in the overlay form over 3G or 4G networks and also expected to co-exist with a variety of unlicensed band wireless access networks, such as WiFi and BlueTooth. Accordingly, the IoT-based multimedia devices should adopt video streaming protocols that reflect the device capabilities and network characteristics.

Although various video streaming mechanisms have been proposed to provide quality of experience (QoE) in wireless network environment,[20–23] they failed to solve the poor bandwidth problem at the cell edge area. In order to solve the bandwidth fluctuation problem, scalable video codec (SVC) has been proposed to encode video data in a scalable manner.[24] While scaling encoded streams deliver to the destination, one or multiple stream layers are extracted and dropped by the intermediate node to adjust streaming rate according to the network bandwidth. In spite of the technical benefit, deployment of SVC is not welcomed in the industry. The main obstacle of the deployment is difficulty of nodes' replacement located in a middle of the networks. To solve the deployment problem, adaptive streaming protocols which adjust streaming bit rate considering the network conditions have been proposed.[11,12,25,26,30,31] Different from the SVC mechanism, adaptive streaming only requires modification of server and client. The server encodes a video stream into multiple bit streams in advance and the client requests a bit rate suitable for the network condition. However, the mechanism could not sensitively respond to the dramatic network fluctuation due to the handoff event over heterogeneous network. It can cause buffer underflow event and consequently degradation of QoE.

Based on the observation of the previously proposed mechanisms, this article aims to propose an adaptive streaming protocol to provide user QoE regardless of the network fluctuation. This article first analyzes the behavior of legacy video streaming protocols and finds out the weakness of the protocols when the network bandwidth is drastically fluctuated due to the handoff events between different networks. Based on the observation, we design and implement a novel mobile video streaming protocol which overcomes the weakness of the legacy streaming protocols. The proposed protocol dynamically adjusts streaming bit rate against the network fluctuation and tries to maintain fixed quantity of buffered audio–video (AV) data for the stable QoE level. This protocol has also been designed for the IoT devices which have various types of OS version and low computing power.

The rest of this article is organized as follows. In section "Observation of the legacy streaming protocols over the heterogeneous wireless networks," we observe and analyze the behavior of legacy streaming protocols. The weaknesses of the protocols over the heterogeneous networks environment are also addressed under the real-world condition. Section "Proposed network adaptation and buffer management protocol" proposes a new adaptive streaming algorithm, which overcomes the weak points of the previous protocols. Section "Design principles of the proposed protocol" describes design principles of the proposed protocol. Section "Implementation of the proposed protocol" describes implementation details of the proposed protocol. Next, we measure the behavior of the proposed protocol and present performance results by the extensive real-world experiments. Conclusion and direction of the future work are made in section "Conclusion."

Observation of the legacy streaming protocols over the heterogeneous wireless networks

In this section, we observe the behavior of the legacy streaming protocols when network is dramatically fluctuated due to the handoff event between different networks. We consider two streaming protocols. The first one is a fixed bit rate streaming protocol, where streaming bit rate is fixed by 700 kbps and the audio–video (AV) data are streamed by constant bit rate manner. The second one is an HTTP adaptive streaming, which dynamically adjusts the streaming bit rate in accordance with network condition. Source media file is encoded as six different bit rates (200, 300, 500, 700, 1000, and 1500 kbps). The target duration for each chunked file is set to 5 s. The initial bit rate for the first download chunk is set to 200 kbps.

Figure 1 illustrates the experimental topology. The server side platform is located in Anyang city and the client side is located in Daejeon city of South Korea. Each side is connected to high-speed multimedia backbone and access network. The maximum streaming bit rate is limited to 9 Mbps at the server side. In order to emulate terminal movement and handoff event as a network fluctuation condition, we utilize a shield room, where Long-Term Evolution (LTE) signal is shielded and only 3G signals are available as shown in Figure 2.

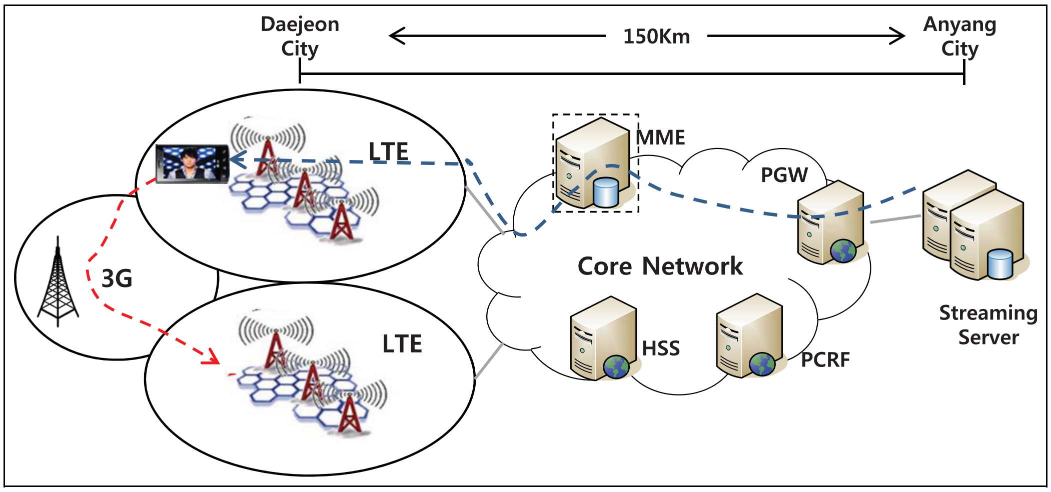


Figure 1. Experimental topology.

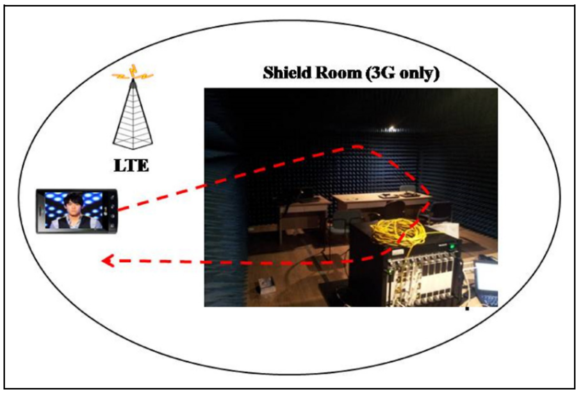


Figure 2. LTE and 3G handoff emulation environment using shield room.

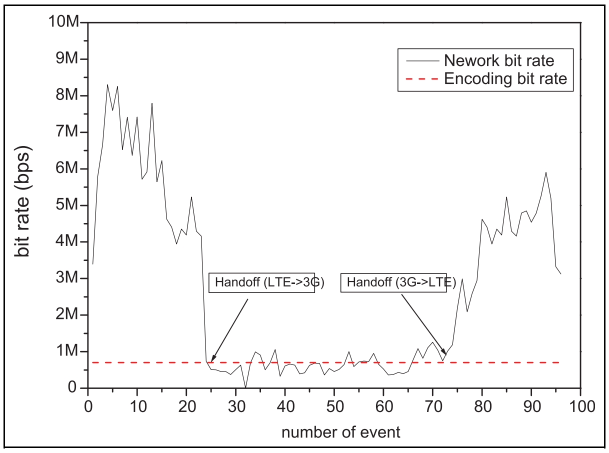


Figure 3. Behavior of fixed streaming protocol.

Outside of the shield room, bandwidth is provided over the 20 Mbps average. On the other hand, average 1 Mbps bandwidth is provided in the shield room.

Fixed bit rate streaming protocol

Figure 3 illustrates the behavior of the fixed streaming protocol when handoff event occurs between LTE and 3G networks. The solid line plots calculated the network bandwidth and dashed line plots the streaming bit rate. When a terminal is attached to the LTE network, which provides wide bandwidth, AV buffer is maintained with enough buffered data. However, when the terminal moves to the shadowing area of LTE and attaches to the 3G network, the providing bandwidth is drastically degraded and fluctuated under the video encoding bit rate. In this environment, it is difficult to maintain AV buffer enough and so buffer underflow event can occur, resulting in poor QoE.

Figure 4 displays the cached AV data in the terminal from the 23rd to 77th events of Figure 3. Guard buffer size 0 implies that buffer underflow event occurs. Buffer underflow event occurs 19 times during this experiment. It implies that the user may suffer from service discontinuity 19 times and it degrades user QoE seriously.

Adaptive bit rate streaming protocol

The adaptive streaming is an enhanced version of the legacy streaming protocol such as Real Time Streaming Protocol (RTSP) and Hypertext Transfer Protocol (HTTP) progressive download. It typically pre-encodes source streams into the multiple bit rate streaming files and the files are divided into the fixed size chunk files. A client adaptively requests the chunk file in the different bit stream according to network condition. Figure 5 illustrates the behavior of the adaptive streaming protocol when the handoff event occurs between LTE and

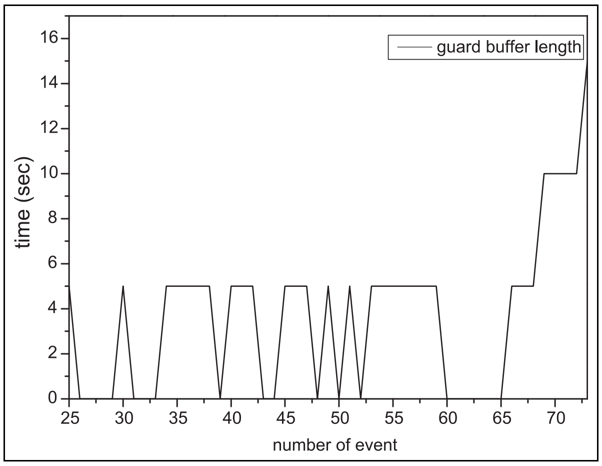


Figure 4. Cached buffer size of fixed streaming protocol.

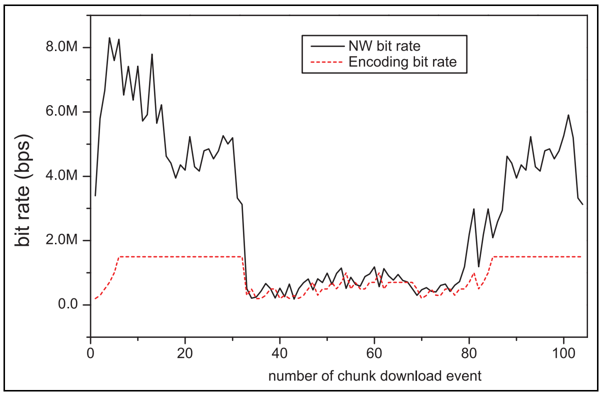


Figure 5. Behavior of adaptive streaming protocol.

3G networks. In this figure, before the 31st download event at the *x*-axis, the terminal attached to the LTE networks and the terminal may download the chunk file encoded as maximum bit rate, 1.5 Mbps. Between the event numbers 32 and 71, the terminal moves to the cell edge area (shadowing area) and handoff to the 3G network. After the event number 72, the terminal returns to the LTE network.

From this figure, we can observe that the algorithm keeps track of the network bandwidth change, but it follows the network fluctuation one step later. Since the decision criteria of the next download bit rate depend on the past network status, this algorithm could not respond to the changing network condition immediately.

Figure 6 represents the chunk downloading time and the size of cached AV data from the 32nd to 77th events of the Figure 5. The guard buffer length reaches zero 20 times and the chunk downloading time gets large at every point. A user then gets QoE degradation such as display broken or still phenomenon. This can be analyzed as follows. When the network bandwidth is

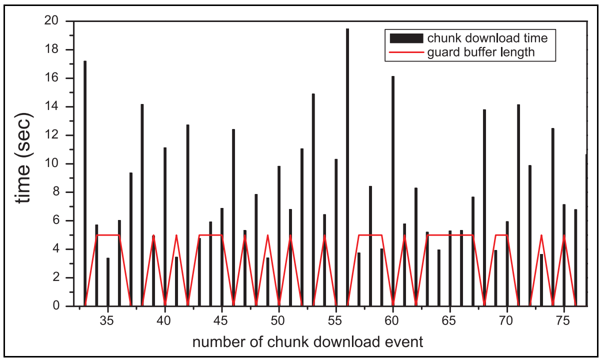


Figure 6. Cached buffer size and chunk downloading time of adaptive streaming protocol.

sufficient to stream the highest encoded data, the terminal requests the highest bit rate chunk file. If the network status is drastically degraded by the handoff event between different networks, the chunk downloading time increases. At the 55th and 56th events, 5-s chunk file is downloaded during 10 and 20 s, successively. The large downloading time consumes AV cached data and causes buffer underflow event. This is basically caused by wrong forecasting of the future network status based on the past information.

Proposed network adaptation and buffer management protocol

One of the problems of the adaptive streaming protocol is mis-forecasting of future network status based on the past information. It causes large chunk downloading time and it consequently leads to buffer underflow event. We propose a new adaptive streaming protocol for overcoming the weakness of the existing streaming protocols. The proposed protocol consists of the network adaptation block and the buffer management block. Network adaptation block conservatively increases streaming bit rate when network status gets better and sharply decreases when it gets worse. The protocol dynamically adjusts minimum threshold (*min_th*), maximum threshold (*max_th*), and maximum bit rate (*max_rate*) depending on the average of the past bandwidth information. The *min_th* and *max_th* are the criterion parameter of the buffered data. If the buffered data are larger than *max_th*, we could estimate that the network status is favorable. If the buffered data are located between *max_th* and *min_th*, we could estimate that the network status is changeable. If the buffered data are smaller than *min_th*, it indicates that the network status is bad. If the data are larger than *max_th* or smaller than *min_th*, it performs with highest or lowest streaming bit rate, respectively. Otherwise, it conservatively increases the streaming bit

Table 1. Pseudo code of the algorithm.

```
Given values:
recent_event, MIN_LOW, MAX_LOW, MIN_HIGH, MAX_HIGH, bit rate changing overhead (BCO)
 1:  Start chunk file download with current index
 2:  While (chunk file download complete)
 3:      if (download time > target duration of chunk file || buffered AV data < min_th)
 4:          if (remaining_chunk_data >= lowest encoded bit rate)
 5:              download chunk = lowest encoded bit rate
 6:              Go to (line 1)
 7:          End if
 8:      End if
 9:  End While
10:  Calculate average BW of the recent_event BW value
11:  if (average_BW of recent_event value> highest encoded bit rate * BCO)
12:      max_rate = highest encoded bit rate
13:      min_th = MIN_LOW
14:      max_th = MAX_LOW
15:  End if
16:  Else
17:      max_rate = round (number of layer/2)
18:      min_th = MIN_HIGH
19:      max_th = MAX_HIGH
20:  End else
21:  If (guard_buffer > max_th)
22:      next_download_chunk = highest encoded bit rate
23:  End if
24:  else if (MIN_TH < buffered_data < MAX_TH)
25:      next_download_chunk = current_download_layer + + ;
26:  End if
27:  Else
28:      next_download_chunk =lowest encoded bit rate
29:  End else
```

rate when network status gets better. The *max_rate* limits the highest streaming bit rate depending on the average of the past bandwidth information, thereby minimizing the impact of the dramatic bandwidth degradation. The average bandwidth information is the hysteresis of the past bandwidth information calculated by the cumulative chunk download bandwidth. These three parameters are updated by every chunk file reception. The number of past chunk download events, which is used to calculate the average bandwidth, is set to recent event (*recent_event)*. If the value of *recent_event* is set to too large, the algorithm could not sensitively detect recent network status. If the value is set to too small, the algorithm too sensitively reacts to the network status.

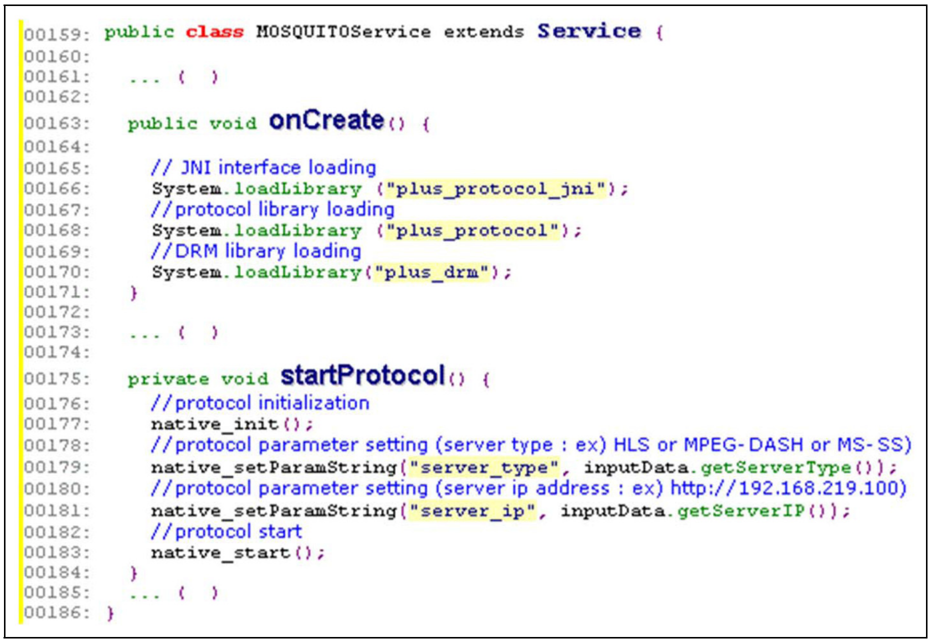
Buffer management block tries to its effort to equalize chunk file downloading time and maintains fixed quantity of buffered data. When the block detects that the chunk downloading time exceeds the target duration or buffered data being less than *min_th*, the block compares the size of left data to download chunk file and the whole data size of the lowest encoded chunk file. After that, the block requests a smaller one between the two of them. When it calculates the size of lowest encoded chunk file, the protocol considers Bandwidth Changing Overhead (BCO) which is HTTP session release and establishment overhead and compensation value for the AV quality degradation. This value is given constant between 1.2 and 1.5. The protocol can guarantee chunk downloading time as two times of target duration when the network bandwidth is provided at least lowest encoding bit rate. Table 1 shows the pseudo code of the algorithm.

Design principles of the proposed protocol

This section presents four key design principles of the proposed protocol. Design principles and the things to be considered for the overall platform architecture, considerations for application developer and rapid deployment are discussed.

Development transparency

Development transparency is one of the most important design principles of the protocol. Application developers do not know additional application programming interfaces (APIs) possible to develop. Application developers without having the domain knowledge of the protocol should be able to develop the service. For this reason, we hide the detail interfaces of protocols such as behavior of streaming protocol or DRM



```
public class MOSQUITOService extends Service {

  ... (  )

  public void onCreate() {

    // JNI interface loading
    System.loadLibrary ("plus_protocol_jni");
    //protocol library loading
    System.loadLibrary ("plus_protocol");
    //DRM library loading
    System.loadLibrary("plus_drm");
  }

  ... (  )

  private void startProtocol() {
    //protocol initialization
    native_init();
    //protocol parameter setting (server type : ex) HLS or MPEG-DASH or MS-SS)
    native_setParamString("server_type", inputData.getServerType());
    //protocol parameter setting (server ip address : ex) http://192.168.219.100)
    native_setParamString("server_ip", inputData.getServerIP());
    //protocol start
    native_start();
  }
  ... (  )
}
```

Figure 7. Source codes for protocol initialization and parameter setting.

interaction. By only updating and loading the protocol library, streaming and digital rights management (DRM) agents could be extended such as MPEG-DASH or PlayReady DRM. As shown in Figure 7, an application developer only loads the library and calls the initialization and sets the protocol parameters by just adding 10 lines. The protocol runs inside the device as shown in Figure 8 and the application developer would not be involved in the protocol procedure any more. It makes the protocol transparent to the application developer.

Independency of OS kinds and versions

It should not modify a kernel or a platform for the development of the proposed protocol and there shall be no dependency on the OS version or type. However, real-time AV decoding and rendering functions require low-level primitive APIs and the APIs have high dependency of operating system and OEM software of the target device. If we modify the low-level APIs to develop some functions, we have to customize in a vast number of cases. It will disrupt the deployment of the protocol also. For this reason, we design a proxy server for mobile devices and utilize a built-in media player of the terminal as shown in Figure 8. The built-in player interacts only with the local proxy server, which is loaded by code of Figure 7. This approach abstracts the heterogeneity of operating systems and OEM software of the mobile devices.

Independency of CPU performance

Since video decoding and rendering consume a lot of computing resources, it will cause many constraints on the performance of the terminal. IoT devices having low computing power could be affected by limited processing capability. Thus, the protocol utilizes the hardware decoder and a renderer mounted in the SoC at the IoT device instead of utilizing CPU. We utilize the default android media player, which exploits Hardware (HW) decoder embedded on the terminal. As a result of this, the proposed protocol does not have dependency of CPU performance.

Deployment simplicity

For rapid deployment of the protocol, modification and development of the protocol are restricted to the terminal side. The protocols are developed in the user-level space but not kernel-level space. Any change in the intermediate nodes such as routers and switches is exempted. Therefore, we can adopt legacy adaptive streaming protocols as a media delivery protocol such as HTTP live streaming, MPEG-DASH, and Microsoft Smooth Streaming. These protocols are HTTP-based media delivery protocols. HTTP-based delivery enables effortless streaming services by avoiding network address traversal (NAT) and firewall traversal issues and provides a simple means to seamlessly change content rate on-the-fly in reaction to the changes in the available bandwidth without requiring the negotiation with the streaming server. We encrypt the media contents with common encryption algorithm (CENC) in order to store a piece of copy for every content not an every copy version of DRM kinds. Also, we adopt de facto standards for the DRM system such as Google Widevine and Microsoft PlayReady DRM as shown in Figure 8.

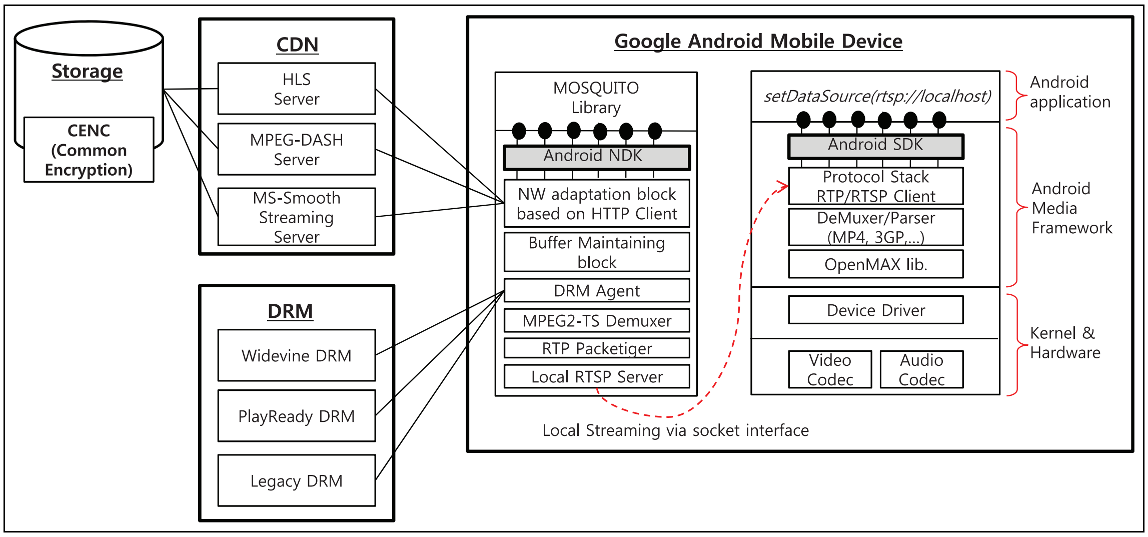


Figure 8. Design principles and SW architecture of the proposed protocol over the Google Android platform.

Implementation of the proposed protocol

We have developed the proposed protocol over the up-to-date Google Android mobile phone, which is equipped with LTE, WiFi, and 3G network interfaces. Since the Android platform basically supports RTSP for the multimedia delivery, we implemented the proposed protocol as well as local RTSP server, which communicate with built-in player through the RTSP standard interface. Through this interface, application developers can utilize the proposed protocol by using the legacy Android API transparently. Since the built-in player interworks with local RTSP server through the standard socket interface, platform dependency is eliminated.

The implementation of the proposed protocol in the terminal side is mainly composed of network adaptation block based on HTTP adaptive streaming protocol, buffer and download time monitor, buffer manager, DRM agent, MPEG2-TS demuxer, realtime transport protocol (RTP) packetizer, and local RTSP server as illustrated in Figure 8. The network adaptation block basically interworks with the HTTP server. The block begins by fetching the index file based on a URL identifying the stream. For the selected stream, the block downloads each available chunk file in sequence. The block calculates the current and average network bandwidth for every chunk file and updates the parameters as explained in the previous section. The block extracts the MPEG2-TS file from the chunk file also. Buffer management block observes the chunk downloading time. If the downloading time exceeds the target duration time, the block compares the data size between the remaining chunk file size and the whole lowest bit rate chunk file size. Then it selects a smaller one. At the comparing time, *BCO* is multiplied by the size of lowest bit rate file to reflect session change overhead. The block manages buffer status and dynamically updates buffer managing parameters upon the bandwidth fluctuation. The whole buffer size is set to 12 times of the chunk file. DRM agent obtains entitlement management message (EMM) key from the DRM key issuer server and decrypts the encrypted AV data. MPEG2-TS parser extracts AV element stream from the multiplexed TS packet. RTP packetizer packetizes AV element stream into RTP network adaptation layer (NAL) units and RTSP server streams out packetized RTP packets to the local RTSP player, which is the embedded Android media player. Thus, a service developer only handles API of the Android software development kit (SDK) related to RTSP-based media player without any concern about the proposed protocol.

The server side platform consists of four main generic components as shown in Figure 9. The first one is media trans-coder that gets real-time live streams or pre-encoded AV files from program provider (PP). The trans-coder encodes a media stream into multiple media streams with variable bit rates. We generate six different bit rates: 1.5, 1.2, 1, 0.7, 0.5, and 0.3 Mbps. Trans-coding should be set to a standardized format supported by the client device, such as H.264 video and HE-AAC audio. The second one is the MPEG2 Transport System (TS) multiplexer. It multiplexes encoded audio and video element streams into 188 bytes fixed size MPEG2-TS Transport Stream. The third one is a scrambler that receives plain AV data from the demuxer and encrypts the data. The fourth one is a HTTP delivery server. Since the proposed protocol basically adopts HTTP as a media delivery platform, control of streaming session entirely depends on the client process. The server is a web server or a web caching system that

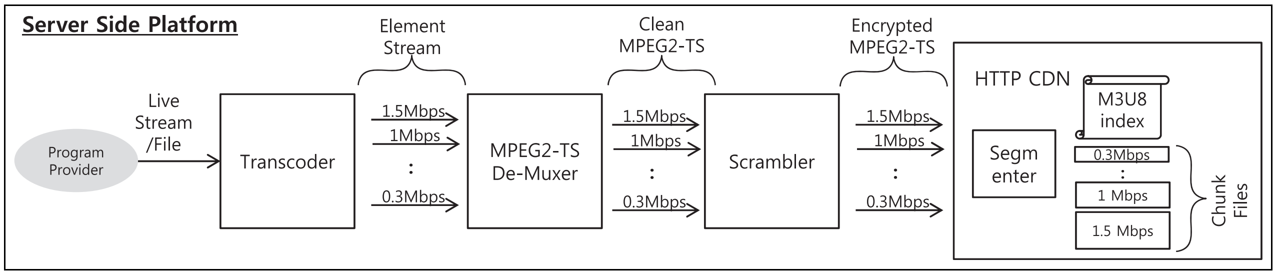


Figure 9. Main components of server side platform.

delivers the media files and index files to the client over HTTP. No new modules are required to deliver the content and typically very little configuration is needed on the generic HTTP server.

The segmenter is a process that reads the encrypted Transport Stream from the scrambler and divides it into a series of small media files of equal duration called chunk file. Here, the equal duration is called the target duration. Even though each chunk file is in a separate file, video files are made from a continuous stream, which can be reconstructed seamlessly. The segmenter also creates an index file containing references to the individual media file. Each time the segmenter completes a new media file and the index file is updated. The index is used to track the availability and location of the media files. Index files are saved as M3U8 or Media Presentation Description (MPD) playlists. Table 2 is a very simple example of an index file, in the form of an MPD playlist, that a segmenter might produce for the entire stream.

Performance evaluation

In this section, we analyze and compare the performance of the proposed protocol with network adaptation block only, and with network adaptation and buffer maintaining block. Figure 10 shows the behavior of the proposed protocol with only network adaptation block when the network bandwidth fluctuates severely. The protocol reacts conservatively when the bandwidth fluctuates compared with a legacy adaptive streaming protocol as shown in Figure 5. The protocol fails to respond sensitively when the network status changes radically. The downloading time of chunk files increases when the network bandwidth degrades as shown in Figure 10(b). Even though the target duration is set to 5 seconds, the downloading time takes over 25 seconds. The buffered AV data at the device may exhaust, display may stop, and QoE degrades finally. System resource efficiency also degrades because the variance of the chunk downloading time and buffered AV data changes drastically.

Figure 11 shows the behavior of the proposed protocol with buffer management block, which keeps a watch on chunk file downloading time and buffer status. The protocol parameters used in this experiment are shown in Table 3. The unit of the first four parameters is the playback duration of the buffered data. When the protocol detects that the chunk downloading time exceeds the target duration time, the block compares the data size between the remaining chunk file size and the whole lowest bit rate chunk file size. Then it selects a smaller one. At the comparing time, *BCO* (1.2) is multiplied by the size of lowest bit rate file to reflect session change overhead. The proposed protocol assumes these situations as network status being degraded. Through this algorithm, the protocol keeps small variance ratio of the chunk downloading time and AV buffered data when the network bandwidth fluctuated as shown in Figure 11(a). Also, there is no buffer underflow event. Especially, we can notice that the chunk file downloading time would not exceed two times of the target duration, 10 s in this experiment. For this reason, we can expect proper size of AV buffer and maximize system resources such as memory and network redundancy. Since we set the maximum size of AV buffer as 12 times of the target duration, little downloading time is taken til the 16 downloading events, and after that the point reaches steady state. Figure 11(c) compares the measured network bit rate and the calculated average network bit rate while changing *recent_event* values as 2, 4, and 8. If *recent_event* value is set to a small value (2), the result traces almost the same as the result of the measured network bit rate. In this case, the protocol sensitively reacts to the network status. When the value of *recent_event* is set to large value (8), the protocol reacts slowly because the protocol conservatively reacts against the real network status. When *recent_event* is set to an appropriate value, such as 4, it properly responds to the hysteresis of actual network conditions calculated by cumulative chunk download bandwidth. Figure 11(d) and (e) are the diagrams showing that the *max_rate, min_th*, and *max_th* values are changed adaptively in accordance with the calculated average bandwidth when the *recent_event* value is set to 4. As can be seen in Figure 11(d), when the network state is deteriorated, *max_rate* value is set to 700 kbps to limit

Table 2. Simple example of an MPD format index file.

```
  <?xml version=“1.0” encoding=“UTF-8” standalone=“no”?>
  <MPD xmlns:xsi=“http://www.w3.org/2001/XMLSchema-instance” xmlns=“urn:mpeg:dash:schema:mpd:2011”
xsi:schemaLocation=“urn:mpeg:dash:schema:mpd:2011” profiles=“urn:mpeg:dash:profile:isoff-live:2011“ type=”static”
mediaPresentationDuration=“PT58S” minBufferTime=“PT2S”>
  <Period start=“PT0S” duration=“PT58.3583S”>
  <AdaptationSet segmentAlignment=“true”>
  <SegmentTemplate timescale=“10000000” duration=“584533111” startNumber=“1” media=“audio_und_$Bandwidth$/
$Time$.m4s” initialization=“audio_und_$Bandwidth$/init.mp4”>
  <SegmentTimeline>
  <S d=“25173222”></S>
  <S t=“25173222” d=“25173334”></S>
  <S t=“50346556” d=“24746666”></S>
  :
  :
  <S t=“550826556” d=“25173444”></S>
  <S t=“576000000” d=“8533111”></S>
  </SegmentTimeline>
  </SegmentTemplate>
  <Representation mimeType=“audio/mp4” codecs=“mp4a.40.24” startWithSAP=“1” id=“1” bandwidth=“602915”></
Representation>
  </AdaptationSet>
  <AdaptationSet segmentAlignment=“true”>
  <SegmentTemplate timescale=“10000000” duration=“583582778” startNumber=“1” media=“video_und_$Bandwidth$/
$Time$.m4s” initialization=“video_und_$Bandwidth$/init.mp4”>
  <SegmentTimeline>
  <S d=“25025000”></S>
  <S t=“25025000” d=“25025000”></S>
  :
  :
  <S t=“550550000” d=“25025000”></S>
  <S t=“575575000” d=“8007778”></S>
  </SegmentTimeline>
  </SegmentTemplate>
  <Representation width=“720” height=“480” mimeType=“video/mp4” codecs=“avc1.64001f” startWithSAP=“1” id=“2”
bandwidth=“661481l”></Representation>
  </AdaptationSet>
  </Period>
  </MPD>
```

MPD: Media Presentation Description.

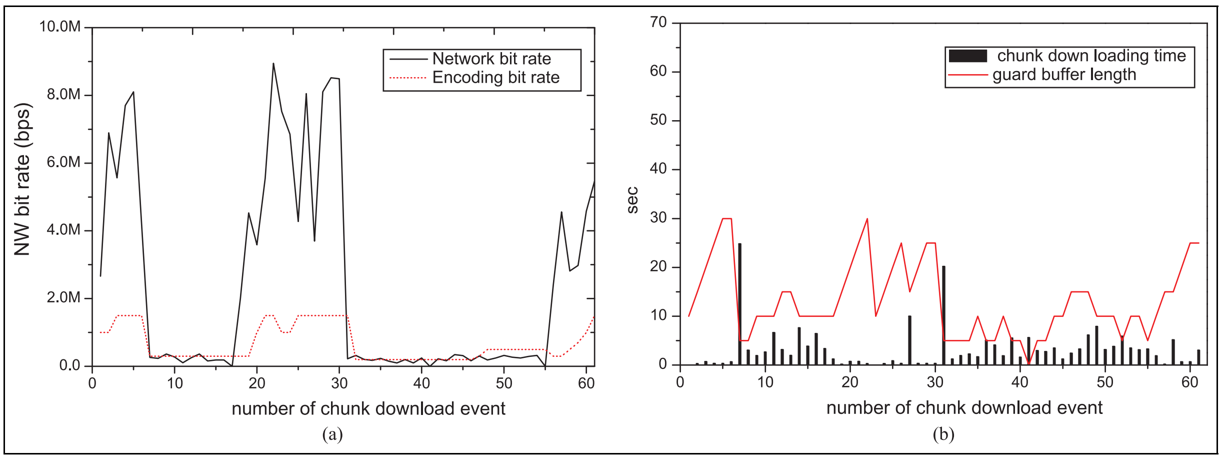


Figure 10. Performance evaluation of the proposed protocol with network adaptation block: (a) trajectory of network adaptation while changing network status and (b) correlation of chunk downloading time and buffered AV data

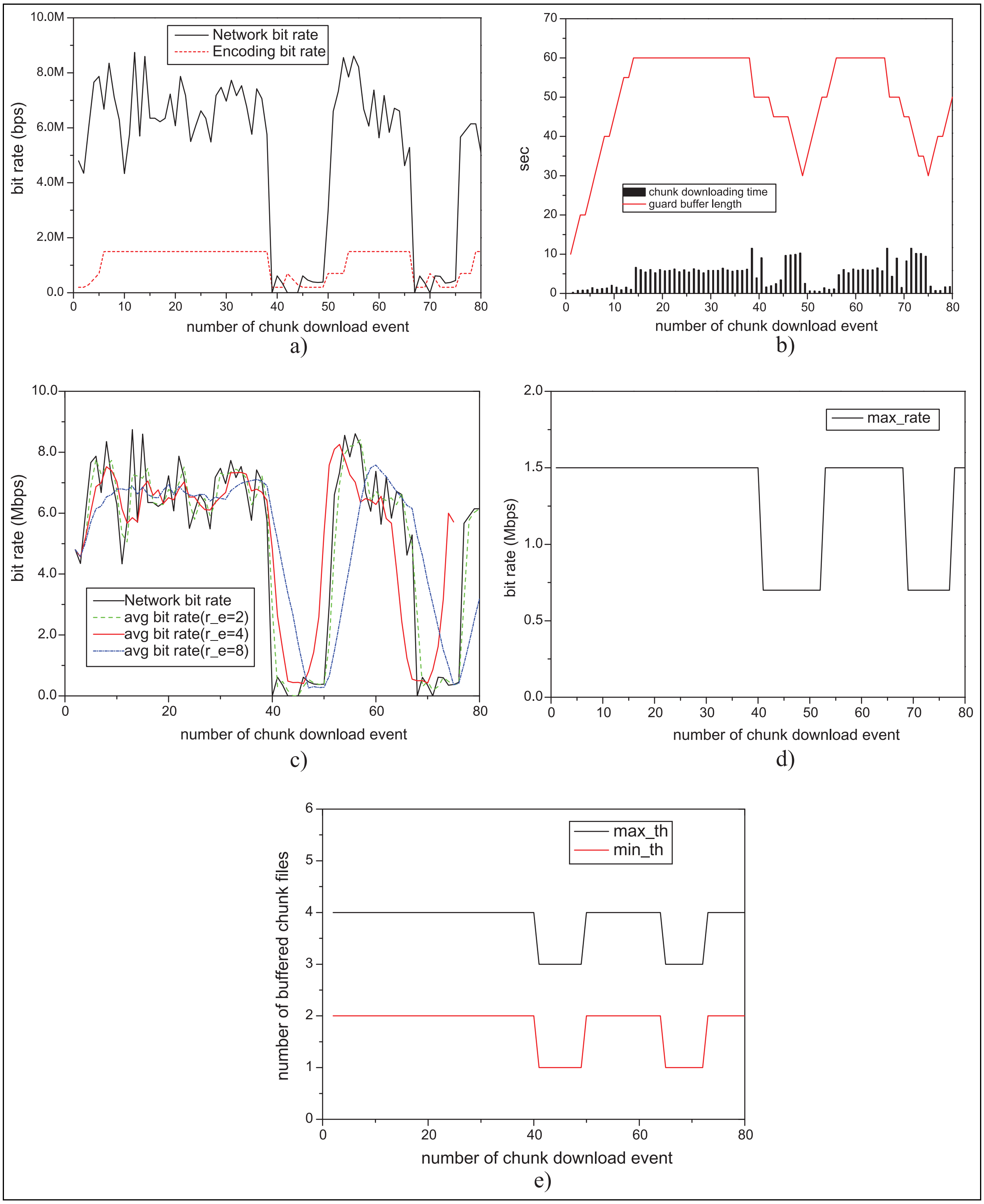


Figure 11. Performance evaluation of the proposed protocol with network adaptation and buffer maintaining block: (a) trajectory of network adaptation while changing network status, (b) correlation of chunk downloading time and buffered AV data, (c) calculated average bandwidth while changing *recent_event* = 2, 4, 8, (d) trajectory of *max_rate* value while changing network status, and (e) trajectory of *max_th* and *min_th* value while changing network status.

maximum increase bit rate and equalize chunk download time. Here, 700 kbps is calculated as the rounding value of the number of encoded bit rate layer/2 as described in Table 1. When the network status is recovered, *max_rate* is set highest encoded bit rate again. As shown in Figure 11(e), the *min_th* and *max_th* values

Table 3. Protocol parameters used in Figure 9 experiment.

MIN_LOW	*MAX_LOW*	*MIN_HIGH*	*MAX_HIGH*	BCO	*recent_event*
5 s	15 s	10 s	20 s	1.2	4

BCO: Bandwidth Changing Overhead.

are also dynamically set to a value as *MIN_HIGH, MAX_HIGH* or *MIN_LOW, MAX_LOW* depending on a network status. As a result, it can provide seamless video watching for user QoE.

Conclusion

This article studied a video streaming protocol over the 5G-based hyper-connected IoT environment. Especially, we focused on QoE issue when multimedia-based IoT devices are moved over the heterogeneous wireless network environment. The behavior of legacy video streaming protocols was analyzed and the weakness of the protocols was found out when the network bandwidth is drastically fluctuated due to the handoff event between heterogeneous wireless networks. Based on the observation, we proposed a novel mobile video streaming protocol. The proposed protocol dynamically adjusts streaming bit rate according to the network fluctuation and attempts to maintain the fixed quantity of buffered AV data for the stable QoE level. The proposed protocol is designed to be independent of the OS version and CPU performance of terminals to be easily portable to various types of IoT devices. We also provided development transparency for the application developer without the knowledge of the proposed protocol. For the rapid deployment, protocol modification and development is only restricted by the terminal side. Even in the terminal side, all the operations of the protocol were developed in the user-level space, not kernel-level space. We have developed the protocol over the many kinds of Android versions such as JellyBean, Kitkat, Lollipop, and Mashmallow and many manufacturer devices such as Samsung, Huawei, and LG. Over the 5 years, the protocol has been running on the commercial LTE/3G/WiFi network with about 5 million users and improved based on the field test and users' experiences. This utilitarian approach has led us to not only better understand their behavior but also optimize the proposed algorithm in real-world conditions. Extensive experimental results proved that the proposed protocol could provide better QoE than legacy streaming protocols for the seamless service when dramatic network fluctuation occurs.

We believe that our protocol design and performance analysis are useful for researchers, application developers, and service providers who consider providing high-quality AV streaming services over the 5G-based hyper-connected IoT environment and try to understand the behavior of streaming protocols in real network conditions.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

References

1. Dahlman E, Ekstrom H, Furuskar A, et al. The 3G long-term evolution—radio interface concepts and performance evaluation. In: *Proceedings of the 63rd IEEE vehicular technology conference*, Melbourne, VIC, Australia, 7–10 May 2006, pp.137–141. New York: IEEE.
2. Ying Z. Antennas in cellular phones for mobile communications. *P IEEE* 2012; 100(7): 2286–2296.
3. IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999), 2007, pp.1–1076S.
4. Hoadley J and Maveddat P. Enabling small cell deployment with HetNet. Wireless Comm 2012; 19(2): 4–5.
5. Hu RQ, Qian Y, Kota S, et al. Hetnets—a new paradigm for increasing cellular capacity and coverage (Guest Editorial). *Wireless Comm* 2011; 18(3): 8–9.
6. Singhrova A and Prakash N. Vertical handoff decision algorithm for improved quality of service in heterogeneous wireless networks. *IET Commun* 2012; 6(2): 211–223.
7. Azhari VS, Smadi M and Todd TD. Fast client-based connection recovery for soft WLAN-to-cellular vertical handoff. *IEEE T Veh Technol* 2008; 57(2): 1089–1102.
8. Choi Y, Ji HW, Park J-y, et al. A 3W network strategy for mobile data traffic offloading. *IEEE Commun Mag* 2011; 49(10): 118–123.
9. Zhuo X, Gao W and Cao G. Win-coupon: an incentive framework for 3G traffic offloading. In: *Proceedings of the 19th IEEE international conference on Yiqi Dai Network Protocols (ICNP)*, Vancouver, BC, Canada, 17–20 October 2011, pp.206–215. New York: IEEE.
10. Andrews JG, Claussen H, Dohler M, et al. Femtocells: past, present, and future. *IEEE J Sel Area Comm* 2012; 30(3): 497–508.
11. Agyapong PK. Design considerations for a 5G network architecture. *IEEE Commun Mag* 2014; 52(11): 65–75.

12. Boccardi F, Heath RW, Lozano A, et al. Five disruptive technology directions for 5G. *IEEE Commun Mag* 2014; 52(2): 74–80.
13. Gupta A and Jha RK. A survey of 5G network: architecture and emerging technologies. *IEEE Access* 2015; 3: 1206–1232.
14. Ziegler V, Theimer T, Sartori C, et al. Architecture vision for the 5G era: cognitive and cloud network evolution. In: *Proceedings of the IEEE 81st vehicular technology conference*, Glasgow, 11–14 May 2015, pp.1–6. New York: IEEE.
15. Kyriakidou A, Karelos N and Delis A. Video-streaming for fast moving users in 3G mobile networks. In: *Proceedings of the 4th ACM international workshop on data engineering for wireless and mobile access*, Baltimore, MD, 12 June 2005, pp.65–72. New York: ACM.
16. Seema A and Reisslein M. Towards efficient wireless video sensor networks: a survey of existing node architectures and proposal for a Flexi-WVSNP design. *IEEE Commun Surv Tutorials* 2011; 13(3): 462–486.
17. Oyman O and Singh S. Quality of experience for HTTP adaptive streaming services. *IEEE Commun Mag* 2012; 50(4): 20–27.
18. http://www.iis.net/downloads/microsoft/smooth-streaming
19. Tavli B, Bicakci K, Zilan R, et al. A survey of visual sensor network platforms. *Multimed Tools Appl* 2012; 60(3): 689–726.
20. Fitzek F and Reisslein M. A prefetching protocol for continuous media streaming in wireless environments. *IEEE J Sel Area Comm* 2001; 19(10): 2015–2028.
21. Li B and Wang KH. Nonstop: continuous multimedia streaming in wireless Ad Hoc networks with node mobility. *IEEE J Sel Area Comm* 2003; 21(10): 1627–1641.
22. Anastasi G, Conti M, Gregori E, et al. An energy-efficient protocol for multimedia streaming in a mobile environment. *J Pervasive Comput Comm* 2005; 1(4): 301–312.
23. Wu S-y and He C-e. QoS-aware dynamic adaptation for cooperative media streaming in mobile environments. IEEE T *Parall Distr* 2011; 22(3): 439–450.
24. Schierl T and Wiegand T. Mobile video transmission using SVC. *IEEE T Circ Syst Vid* 2007; 17(9): 1204–1217.
25. http://www.adobe.com/products/httpdynamicstreaming/
26. http://developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/StreamingMediaGuide.pdf
27. Li M and Lin H-J. Design and implementation of smart home control systems based on wireless sensor networks and power line communications. *IEEE T Ind Electron* 2015; 62(7): 4430–4442.
28. Kelly SDT, Suryadevara NK and Mukhopadhyay SC. Towards the implementation of IoT for environmental condition monitoring in homes. *IEEE Sensors J* 2013; 13(10): 3846–3853.
29. Palattella MR, Dohler M, Grieco A, et al. Internet of things in the 5G era: enablers, *architecture and business models*. *IEEE J Sel Area Comm* 2016; 34: 1.
30. 3GPP TS 26.247 v10.1.0. Transparent end-to-end packet switched streaming service (PSS); progressive download and dynamic adaptive streaming over HTTP (3GP-DASH), 10 June 2011.
31. ISO/IEC DIS 23009-1. Information technology—dynamic adaptive streaming over HTTP (DASH)—part 1: media presentation description and segment formats.