TCP-ROME: A Transport-Layer Parallel Streaming Protocol for Real-Time Online Multimedia Environments

Ju-Won Park, Roger P. Karrer, and Jongwon Kim

Abstract: **Real-time multimedia streaming over the Internet is rapidly increasing with the popularity of user-created contents, Web 2.0 trends, and P2P (peer-to-peer) delivery support. While many homes today are broadband-enabled, the quality of experience (QoE) of a user is still limited due to frequent interruption of media playout. The vulnerability of TCP (transmission control protocol), the popular transport-layer protocol for streaming in practice, to the packet losses, retransmissions, and timeouts makes it hard to deliver a timely and persistent flow of packets for online multimedia contents. This paper presents TCP-real-time online multimedia environment (ROME), a novel transport-layer framework that allows the establishment and coordination of multiple many-to-one TCP connections. Between one client with multiple home addresses and multiple co-located or distributed servers, TCP-ROME increases the total throughput by aggregating the resources of multiple TCP connections. It also overcomes the bandwidth fluctuations of network bottlenecks by dynamically coordinating the streams of contents from multiple servers and by adapting the streaming rate of all connections to match the bandwidth requirement of the target video.**

Index Terms: **Parallel streaming protocol, parallel TCP, rate and content coordination, real-time online streaming.**

I. INTRODUCTION

Real-time multimedia streaming has become the dominant source of the Internet traffic today. The Web 2.0 has moved the file sharing idea to a novel level where videos are easily uploaded, shared, and viewed in real time. In November 2010, 172 million U.S. Internet users watched online video content and engaged in nearly 5.2 billion viewing sessions.[1] The recent acquisition of YouTube by Google emphasizes the market potential in home-made videos. Similarly, the Web content of traditional sites includes increasingly embedded videos or flash animations. The long-predicted multimedia application of content in the Internet has been enabled by three components. First, easy-to-use and free distribution mechanisms, such as the Web 2.0 and peer-to-peer (P2P) systems, allow users to easily upload, share, and view video content. Second, the technical advances in providing broadband access to homes enable users to download and view content in real time. Third, the vast amount of available storage, either by a portal or by building distributed systems such as content distribution networks (CDNs), e.g., Akamai, and P2P systems, make petabytes of content available to everyday users.

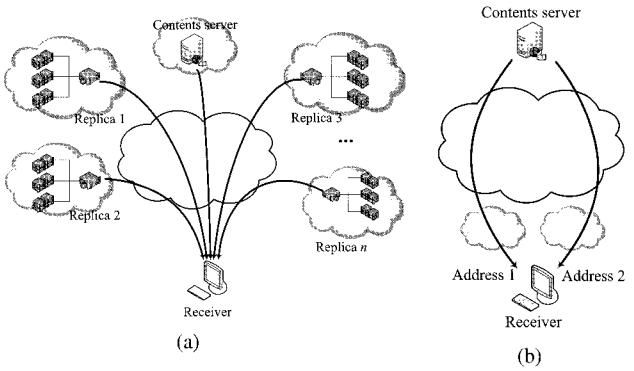


Fig. 1. Enhancing QoE for online media streaming: Streaming over multiple paths either (a) from replicated servers or (b) via client-side multihoming or a combination of both.

Unfortunately, these three enablers are not sufficient to allow a high-quality, disturbance-free quality of experience (QoE) to users. More specifically, the sharp reactions to congestion, the retransmissions, and the timeouts make TCP (transmission control protocol) largely unsuitable to guarantee the promised QoE. In spite of these limitations, TCP has become the dominant transport-layer protocol even for real-time streaming because of its fairness and reaction to congestion [1]. For example, a measurement study has shown that, for both stored-video and live streaming, a significant fraction (around 66%) of the commercial traffic uses TCP [2].

However, the design of the transport layer, in particular today's form of TCP, is still one of the main bottlenecks that prevents a high QoE of multimedia application. To address current limitations of existing TCP-based solutions, parallel streaming approaches based on multiple connections have emerged [3]–[10]. Fig. 1 shows that a parallel streaming protocol can be used in different scenarios. In the simplest and well-known case, parallel connections can be established between a single client-server pair [6]–[8]. However, in modern scenarios, more cases are possible. First, the increasing content replication to build scalable systems and to reduce round-trip times, e.g., CDNs for web applications enables a parallel streaming from multiple replica servers. Alternatively, a parallel streaming can be envisioned when the client is multi-homed. For example, mobile devices may use multiple wireless technologies in parallel, and homes are increasingly equipped with multiple Internet service

Manuscript received February 24, 2010; approved for publication by Richard J. La, Division III Editor, December 6, 2010.

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2011-(C1090-1111-0004)).

This paper is a substantially revised version of a paper that appeared in the Proc. of IEEE International Workshop on Quality of Service (IWQoS) 2008.

J.-W. Park and J. Kim are with the Networked Media Laboratory, Gwangju Institute of Science and Technology (GIST), Gwangju, Korea. J.-W. Park is currently affiliated with Korea Telecom, email: {jwpark, jongwon}@gist.ac.kr.

R. P. Karrer is with the Deutsche Telekom Laboratories TU Berlin, Berlin, Germany, email: roger.karrer@gmail.com.

[1] [Online]. Available: http://www.comscore.com.

provider connections as well as wireless connections [9], [10]. In [9], multi-path connections for cellular links is proposed to cope with channel diversity. Also, in [10], multi-path streaming connections are used to increase the tolerance to packet loss and delay due to network congestion.

In this paper, we propose TCP-real-time online multimedia environments (ROME), a parallel streaming protocol for ROM. TCP-ROME establishes many-to-one parallel TCP connections between a client with one or multiple multi-homing addressed, and one or multiple distributed replica servers. It extends [11] by providing an improved fine-grained mechanism that coordinates the content stream according to the monitored individual connection status (e.g., round-trip time (RTT) and packet loss probability) in order to ensure a real-time high-quality media streaming experience. More specifically, TCP-ROME coordinates the content streaming at the granularity of individual segments over the different connections and merges them in real-time at the receiver. The aggregated bandwidth of multiple disjoint paths fulfills one of the requirements to improve QoE. Moreover, TCP-ROME also ensures that content segments are transmitted over these connections where the bandwidth and delay conditions promise a timely delivery. Finally, when segments are lost or expected to be delivered late, TCP-ROME can re-request the segments over alternative connections. In addition, TCP-ROME provides rate adaptation that adjusts the aggregated rate to match the requested video streaming rate. This rate adaptation ensures that the finite client buffer neither overflows nor starves. Experimental results via simulations show that TCP-ROME can provide a sustained throughput and therefore, significantly increase the QoE of video streaming.

The remainder of this paper is organized as follows. Section II overviews parallel streaming protocols. Then, detailed descriptions of TCP-ROME are provided in Section III. Next, Section IV describes the performance evaluation of TCP-ROME. Finally, we conclude this paper in Section V.

II. PARALLEL STREAMING PROTOCOLS

In this section, we first describe the requirements and challenges for parallel streaming protocols. Then, we provide some previous work about parallel streaming protocol designed for real-time media.

A. Requirements & Challenges

In this paper, a parallel streaming protocol is investigated to leverage n parallel TCP connections and to stream a content concurrently over multiple paths. We expect the parallel TCP approach can significantly improve the QoE of users for the following reasons. First, a parallel protocol aggregates the bandwidth of multiple connections. If the connections use disjoint paths, the aggregated bandwidth is a multiple of the bandwidth of a single flow. Second, it can compensate throughput degradation of one connection by increasing the throughput of another connection. Third, it can ensure a timely delivery of the segment. If one connection is not able to deliver the segment in time, e.g., due to TCP congestion control reactions, the segment can be streamed using alternative connections. Finally, it can provide a faster startup time because multiple streams initiate their slow-start phase.

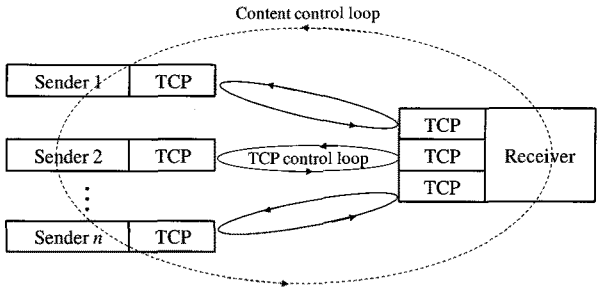


Fig. 2. Challenge: How can the two control loops interact?

To design a parallel streaming protocol for multimedia environments, we will address the following protocol requirements.

- **Bandwidth heterogeneity:** The protocol must provide a high QoE in a large variety of physical bandwidth rates, ranging from kb/sec connections up to high-speed networks with Gb/sec speed [12], [13]. The available bandwidth is thereby a difficult-to-predict function of network conditions and server load.
- **Size heterogeneity:** The protocol must ensure an efficient streaming for different size distributions. The streamed content may range from small multimedia contents of 100 kB to several MB (today's videos) up to future high-definition videos of several GB.
- **Reliability:** The protocol must guarantee to stream even if servers suddenly and unexpectedly terminate connections and when network connections fluctuate due to network congestions.

Parallel TCP streaming protocols should determine how to split the segment such that only a fraction of the content is streamed over each path, and it ensures that eventually the entire content is streamed. If the content is hosted on a single server, this coordination is a lot easier than if the content is replicated. In the latter case, the content is typically split into multiple segments (a.k.a., chunks) and segments are streamed from different locations. It also keeps track of the successfully streamed segments as BitTorrent [14]. However, additional challenges arise for real-time parallel streaming because the stream must be coordinated to ensure that the content is received *in time*, and not just *eventually* as with BitTorrent.

Thus, as depicted in Fig. 2, it is necessary to integrate two control loops: the TCP control loop and the content control loop. The TCP control loop dynamically adjusts to network congestion and ensures that the segments are delivered reliably. The content control loop is responsible to coordinate the segment delivery of each server to avoid overlaps and to ensure that the entire video is delivered. These two loops are traditionally independent, separated by the socket interface. However, the separation of control loops has severe drawbacks for real-time streaming. If the delivery of one segment is blocked on one connection due to congestion, the video is stopped until video segment is delivered. With a minimum TCP retransmission timeout of 1 sec, the streaming interruption can not be hidden from the user. If the congestion is persistent, the viewing experience may be intolerable. Therefore, to ensure a pleasant viewing experience, we argue that a joint reliablity/congestion and content control is

necessary.

B. Related Work

The use of parallel TCP connections has been addressed in [8] and [15]. In [8], TCP-based multi-path live streaming is proposed to distribute packets over multiple paths based on the available bandwidth. To analyze the performance gain of multiple paths, it develops an analytical model and validates this model by using extensive network simulations and Internet experiments. In [15], a *MultiTCP* scheme provides the resilience against short-term bandwidth bottlenecks due to network congestion by using multiple TCP connections. For that, it adjusts the receiver window size of each connection to precisely control the sending rate. However, these studies focus only on the path diversity between a single client-server pair.

Parallel streaming approaches from multiple servers to a single client are exploited in [4] and [10]. In [4], a streaming mechanism from multiple servers is proposed by using multiple description (MD) encoding. They present a distortion model for MD encoding and use this model to estimate the effect of server and content placement on the delivered quality. Similarly in [10], a streaming video framework from multiple mirror senders to a single receiver is proposed by using multiple connections. In this framework, the receiver periodically reports the throughput and delay of all sender connections back to them. Then, the senders run a distributed algorithm to determine which one should send each packet.

By using the P2P approach, other parallel streaming ideas are also proposed [16]–[18]. In [16], a multimedia streaming extension of BitTorrent download is proposed by replacing the policy for chunk download for time-sensitive streaming. In [17], cooperative networking (CoopNet), where clients cooperate to distribute contents while alleviating the server load, leverages MD encoding to send different MD descriptions to separate trees of interested peers. Also, in [18], a new P2P streaming adaptation scheme, called as peer-to-peer adaptive layered streaming (PALS), is proposed to cope with unpredictable throughput variation. The quality-adaptive playback of layer-encoded streaming media is attempted from a group of congestion-controlled sender peers to a single receiver peer.

The proposed TCP-ROME is a parallel TCP-based streaming protocol for real-time multimedia by establishing multiple TCP connections from multiple servers to a single client. Then, each TCP connection is coordinated to alleviate the instantaneous throughput fluctuation and to ensure timely delivery of segment. Although the proposed coordination is not totally new by itself, the tight integration of TCP congestion control and retransmission with the rate and content coordination is new. Also, as a transport-layer solution, TCP-ROME is readily applicable to many applications without re-implementations.

III. TCP-ROME: PARALLEL STREAMING PROTOCOL FOR REAL-TIME ONLINE MULTIMEDIA ENVIRONMENTS

This section first describes the overall design of TCP-ROME, a novel parallel streaming protocol. Then, we discuss the core components, in particular the content coordinator and the rate coordinator, in detail.

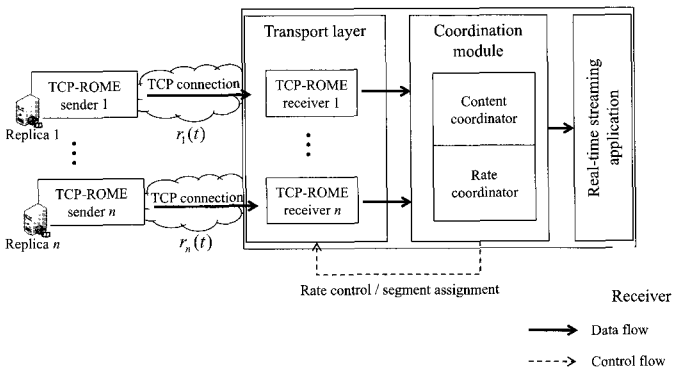


Fig. 3. Multimedia streaming via TCP-ROME.

A. Overall Design

TCP-ROME extends TCP-PARIS [19][2] by providing a framework that allows a dynamic adjustment of segment allocation and transmission rate of each connection to support real-time media streaming. TCP-ROME connection consists of n connections that connect a single receiver with n senders via point-to-point TCP connections, as depicted in Fig. 3. The parallel connections can be established between single client-server pair. However, to achieve path diversity and improve the probability of enhancing QoE, TCP-ROME is primarily designed to work between multiple replica servers and a single client, or between a multi-homed client and one or multiple senders. Multi-homing addresses bandwidth problems when the bottleneck is at the client side, whereas servers deal with server-side and network bottlenecks.

From the viewpoint of application, the communication between a client and servers with TCP-ROME is almost identical to the communication with a single TCP connection. The client passes one or multiple addresses to TCP-ROME to establish parallel connections. The returned end-point abstraction (socket) is the same as that of traditional TCP. Requests sent from the client are duplicated and sent to the servers. Then, TCP-ROME receiver starts coordinating the parallel streaming by assigning every connection individual segments of contents to deliver. Every server sends only those segments that are requested by the receiver. At the receiver side, TCP-ROME merges the segments before it is passed to the application. Therefore, from the perspective of application, TCP-ROME hides the fact that a content is streamed over multiple paths.

For this, TCP-ROME partitions the content into the smallest unit, a segment[3] of contents, as shown in Fig. 4. Then, the receiver determines *which* segments are sent from each replica and *how fast* they are sent. Within TCP-ROME, two components are responsible to these issues: A rate coordinator (RC) and a content coordinator (CC). RC is responsible for coordinating the

[2]TCP-PARIS is a parallel download protocol that splits a file into smaller units and downloads disjoint units from different replicas.

[3]Actually, the size of segment is an important issue. However, in this paper, we simply assume that a content can be partitioned into multiple segments of the same size.

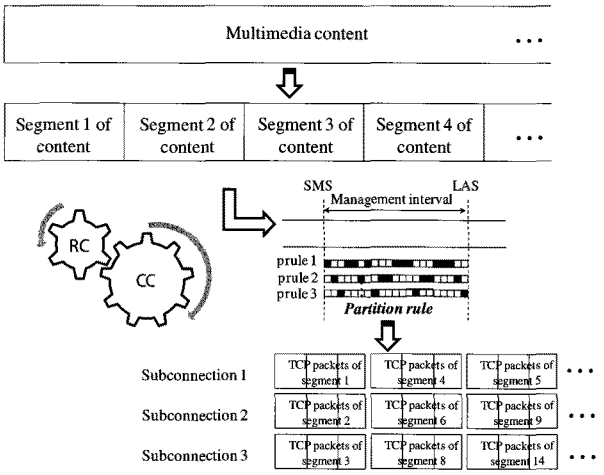


Fig. 4. Rate and content coordination of TCP-ROME.

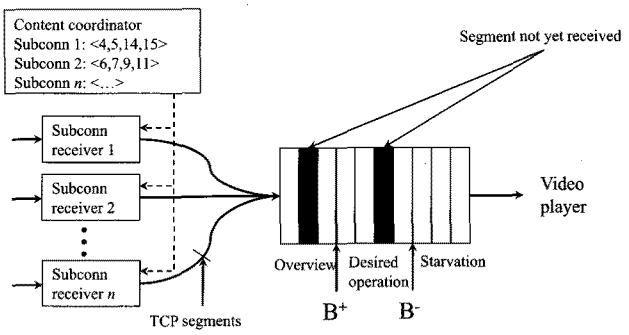


Fig. 5. Buffer management at the receiver.

bandwidth among different connections. CC is responsible for ensuring that all segments are received at the client at the right time. The amount of segments requested from a server is proportional to the actual congestion window size during the entire streaming time. Therefore, servers with a high throughput deliver more segments than slow servers, and the streaming rate is dynamically adapted to the available bandwidth variations. The next sections describe the behavior of TCP-ROME by separating two phase, namely *start-up phase* and *dynamic resource adaptation phase* in detail.

B. Start-up Phase

In a start-up phase, RC opens the congestion window of every TCP-ROME connection until the aggregate throughput of TCP-ROME achieves the video display rate, $\bar{R}_v$. In this phase, the segment assignment is an important issue. For this, CC ensures that (i) every segment is delivered only once and (ii) all segments are eventually delivered. To handle this assignment, CC maintains a structure called as *partition rule* [19]. The partition rule stores which segment is assigned to which connection. The detailed assignment is only stored for a subset of segments that is bounded by 2 variables: *SMS*, which stands for the smallest missing segment, and *LAS*, the last assigned segment. All segments with segment numbers smaller than *SMS* have been successfully delivered to the receiver, and all segments larger than *LAS* are not assigned to either server. Thus, the partition rule can be seen as a global sliding window (i.e., a sliding window that covers all connections).

At the same time, TCP-ROME measures RTT and packet loss probability to check the performance of individual connections. To obtain RTT of given TCP flow, smoothed RTT is used with smoothing factor α [20].[4] In addition, the receiver counts the number of lost and received packets to estimate the packet loss probability.

How does TCP-ROME know the video display rate $\bar{R}_v$? There are two options. First, the user knows the rate and sets it as a parameter. Second, if the rate is unknown, TCP-ROME can dynamically monitor the receiver buffer and infer the display rate. In particular, denote $r_v(t)$ as the actual frame rate required by the video at the client at time t. The rate is time dependent as the video content may vary even if the frame rate remains constant. Segments arrive from the network at a streaming rate $r_s(t)$. The ratio between $r_v(t)$ and $r_s(t)$ determines the fill degree of the client-side buffer, depicted in Fig. 5. TCP-ROME must manage how much the receiver buffer is filled in order to prevent media playout interruptions. To this end, TCP-ROME continuously monitors the buffer level of receiver buffer, denoted as $b(t)$, which is measured by the number of bits filled in the receive buffer at time t. A simple strategy to adjust the streaming rate of the connections is to define two thresholds B^+ and B^-. The objective is to maintain $B^+ < b(t) < B^-$. If $b(t) < B^-$, the buffer dries out and the video will be halted. Therefore, the RC must increase the download rate of its connections. In contrast, if $b(t) > B^+$, the buffer is filling up, and the RC may reduce the download rate.

C. Dynamic Resource Adaptation Phase

For real-time streaming, we need to guarantee that a segment is timely delivered as well as eventually delivered. Assume that one connection experiences congestion and therefore goes into a timeout. Then, the assigned segments are therefore not transmitted until the timeout expires–which may be too late for the real-time stream. The impact of a missing segment depends on the application. The video streaming may have to be stopped until the segment arrives, or the video omits the frame to which the segment belongs.

Thus, to ensure that the segments arrive in time, CC jointly monitors the partition rule and the buffer. Specifically, CC searches for missing segments within the sliding window and checks with the buffer (Fig. 5) to estimate the display time of individual segments. Those segments that should be displayed soon but have not yet been received must be delivered immediately. In particular, denote $s_i(t)$ as the location of the ith segment in the client-side buffer at a given time t. The 0th segment is the segment that will be fetched next by the video player. Over time, the index i decreases as the video plays. The expected play time of a segment s_i, can be estimated as

$$t_p(s_i) = \frac{i \times ssize}{\bar{R}_v} \tag{1}$$

with $\bar{R}_v$ as the average video play rate and $ssize$ as the segment size. We define that a segment is missing if $t_p(s_i) < rtt(s_i)$,

[4] $srtt_{i+1} = \alpha \times srtt_i + (1-\alpha) \times \mathrm{RTT}_i$ where $srtt_i$ is previous smoothed RTT, $srtt_{i+1}$ is new computed value, and RTT_i is current measured RTT.

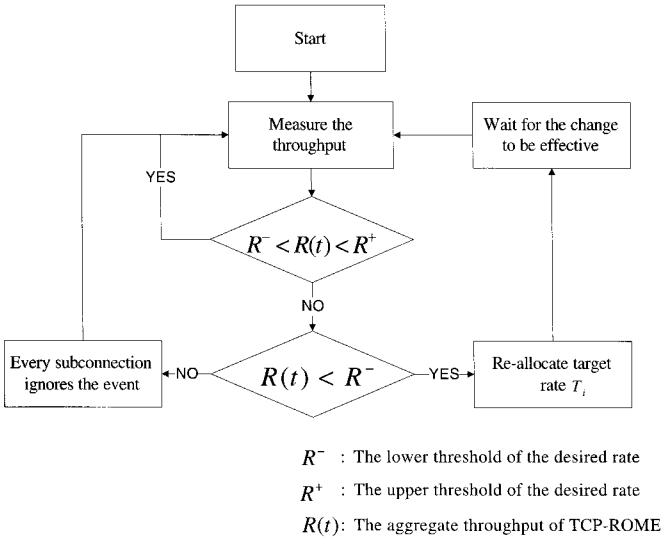


Fig. 6. Flowchart of rate adjustment mechanism of RC.

$s_i \notin B$ where B and $rtt(s_i)$ denote the buffer and round-trip time of connection which is assigned with s_i, respectively. If a missing segment is detected, CC re-requests it over the connection with lower RTT than the expected playback time. Thus, the case of re-requesting a missing segment may violate the original concept of TCP-ROME that every segment is delivered over exactly one connection. By re-assigning a segment to an alternative connection, it is possible that the same segment eventually arrives over both connections. However, this tradeoff is necessary to maintain the real-time streaming properties. Should a connection be responsible for a significant number of missing segments, TCP-ROME can consider abandoning this connection.

In addition, TCP-ROME needs to adjust the aggregate throughput to prevent buffer underflow/overflow. For this, RC allows a dynamic adaptation of the streaming rate of each connection based on the condition of flow. Specifically, RC is an optional component that is only needed if TCP-ROME should achieve an aggregated throughput that is different from the sum of the throughput of all connections. We currently envision two scenarios where RC is required. First, assume that the throughput of TCP-ROME should be approximated, but not exceed the streaming rate of the video by a pre-defined threshold. Such a limitation is useful to keep an upper threshold on the client-side buffer. Second, TCP-ROME is inherently unfair towards flows that use a single connection only, such as traditional TCP applications, but also delay-sensitive applications like voice over Internet protocol (VoIP). To avoid that single flows degrade or even starve, TCP-ROME may trade off between throughput and fairness towards single flow. In the remainder of this paper, we consider the first case.

For dynamic adaptation of the streaming rate, we propose a simple rate coordination mechanism based on the thresholds as shown in Fig. 6. First, RC measures the R at the fixed interval, ϕ.[5] Denote R^+ and R^- as the upper and lower threshold of the desired rate such that $R^- < R(t) < R^+$. If $R(t) > R^+$, the parallel streaming rate exceeds the necessary streaming rate. Therefore, RC signals the connections to omit throughput increases. In particular, if a connection receives an event to open its congestion control window but RC has signaled that a rate increase is not necessary, the connection ignores the event and maintains the congestion window at its actual size. Alternatively, if $R(t) < R^-$, the streaming rate is no longer maintained. Here, RC has two options that can be set by the user. The first option is to strictly adhere to TCP fairness and accept that the streaming rate may not be sufficient. An alternative is a "compensation" model. In times of high available bandwidth when connections ignore events to open the congestion window, RC may compensate insufficient throughput of TCP-ROME by adjusting the throughput of TCP connections. In this model, the key concept of rate coordination is proportional rate adjustment based on estimated throughput. For that, we leverage the methodology used by Padhye et al. [21] that models TCP throughput. By using the throughput modeling, we can estimate the throughput of the ith connections from measured RTT $srtt_i$ and packet loss probability p_i as

$$\tilde{r}_i(srtt_i, p_i) = \frac{1}{srtt_i\sqrt{\frac{2bp_i}{3}} + T_0 \min(1, 3\sqrt{\frac{3bp_i}{8}})p_i(1+32p_i^2)} \tag{2}$$

where b and T_0 denote the number of packets that are acknowledged by a received ACK and a period of sender's time-out, respectively. Then, RC calculates the desired rate T_i of ith TCP-ROME connection based on the proportional estimated throughput of each connection as

$$T_i = \frac{\tilde{r}_i(srtt_i, p_i)}{\sum_{j=1}^{n} \tilde{r}_j(srtt_j, p_j)} \bar{R}_v. \tag{3}$$

After an adjustment, TCP-ROME has to wait, φ, for the change to be effective before the next adjustment. In [1], the minimum value for φ is discussed and dictated by $\varphi > \text{RTT}+\phi$, where ϕ is throughput estimation period. In addition, after a sending rate adjustment of each connection, the past smoothed RTT value and loss probability become irrelevant. Therefore, the receiver simply sets $srtt_0$ as initial measured RTT and re-records the states of loss and received packets.

D. TCP-ROME Steady-State Throughput

In this subsection, we derive an expression for the expected throughput of TCP-ROME. Let's denote T^i_{ROME} and T_{ROME} as the steady-state throughput of ith TCP-ROME connection and TCP-ROME, respectively, based on the throughput model developed in [21]. T_i derived in (3) can be considered as an upper bound of ith TCP-ROME connection since each TCP-ROME connection does not exceed the desired rate. To this end, we have to determine a maximum congestion window size, $W^i_{\max} = \frac{\text{MTU}}{srtt_i \times T_i}$, where MTU denotes the TCP maximum transfer unit. Therefore, T^i_{ROME} can be derived as

$$T^i_{\text{ROME}}(srtt_i, p_i) = \min\left(T_i, \tilde{r}_i(srtt_i, p_i)\right) \tag{4}$$

[5] ϕ is a crucial parameter to ensure the convergence and stability of RC. However, in this paper, for the sake of simplicity, we simply assume that RC measures the aggregate throughput of TCP-ROME at every RTT to ensure fast convergence.

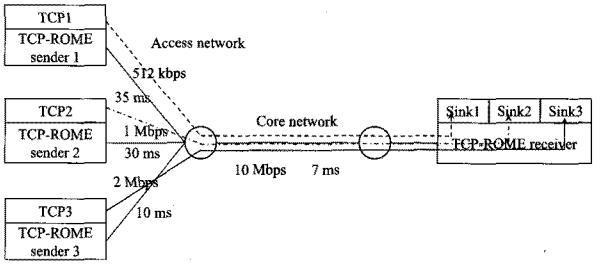


Fig. 7. Simulation topology for the microscopic behavior of RC and CC.

The steady-state throughput of TCP-ROME is the throughput summation of the individual connections as shown in (5) when a TCP-ROME connection consists of n TCP connections.

$$T_{\mathrm{ROME}}(srtt, P) = \sum_{i=1}^{n} T^{i}_{\mathrm{ROME}}(srtt_i, p_i) = \sum_{i=1}^{\theta} T_i + \sum_{j=1}^{\eta} \tilde{r}_j(srtt_j, p_j), \tag{5}$$

where $srtt = \{srtt_1, srtt_2, \cdots, srtt_n\}$ and $P = \{p_1, p_2, \cdots, p_n\}$ denote the set of measured RTT and packet loss probability of each TCP-ROME connection, respectively. In addition, θ and η denote the number of connections where $\tilde{r}_i > T_i$ and $\tilde{r}_i \leq T_i$, respectively, such that $n = \theta + \eta$.

IV. EVALUATION

In this section, we systematically study the performance of TCP-ROME using ns-2. First, we investigate the details of RC to maintain a given streaming bandwidth and CC to ensure a timely delivery of the segment. Next, we assess the throughput and fairness of the TCP-ROME in a distributed system with 25 clients and servers.

A. Verification of Microscopic Behavior of RC and CC

To assess the microscopic behavior of RC and CC, we make a very simple topology depicted in Fig. 7. The server-side access network has a lower bandwidth than the core and the receiver-side network to prevent that bottleneck is near the client–in which case the usefulness of parallel streaming is limited. The server-side networks have different bandwidths, allowing for different strategies to be investigated. In this experiment, we assume that bitrate and frame rate of video stream are 3 Mbps and 30 fps, respectively. The parallel streaming application receives the video packets with 3 connections and 2 seconds initial buffering time.[6] The bitrate of 3 Mbps is chosen because (i) no single connection is able to support this bandwidth but (ii) the aggregated bandwidth is larger than the required rate.

Fig. 8 shows the throughput of TCP-ROME when concurrent TCP flows are added in the same access network. The cross traffic interferes with the TCP-ROME flows and forces RC to adjust its rates. The legends in Fig. 8 denote the activation times of the cross flows and average throughput. Specifically, we

[6]Actually, the decision of initial buffering time is an important issue. However, in this paper, we assume preset value for the sake of simplicity.

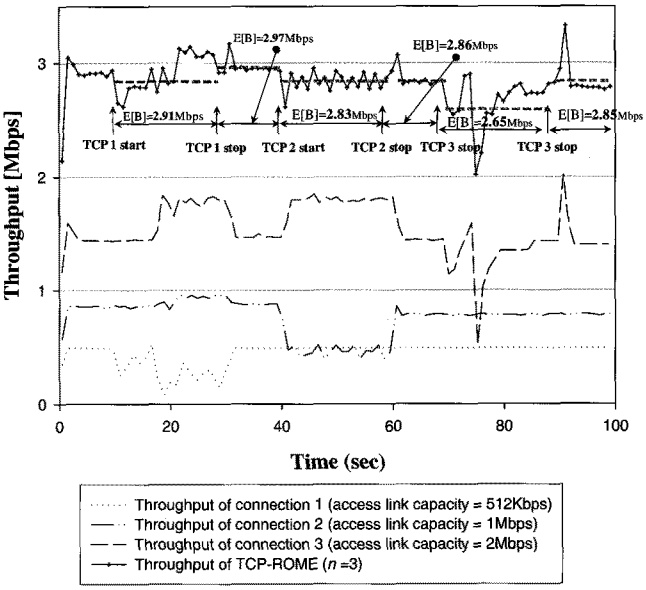
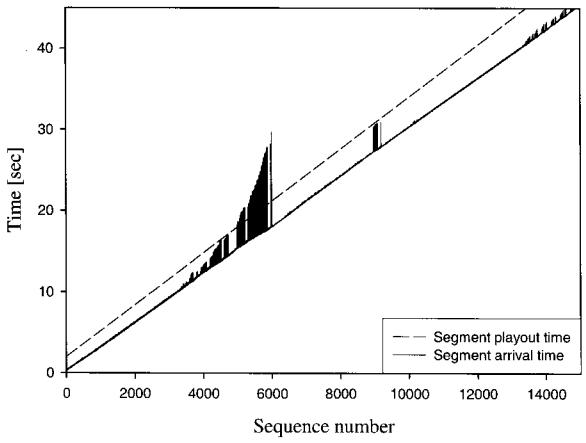


Fig. 8. Bandwidth aggregation with cross traffic.

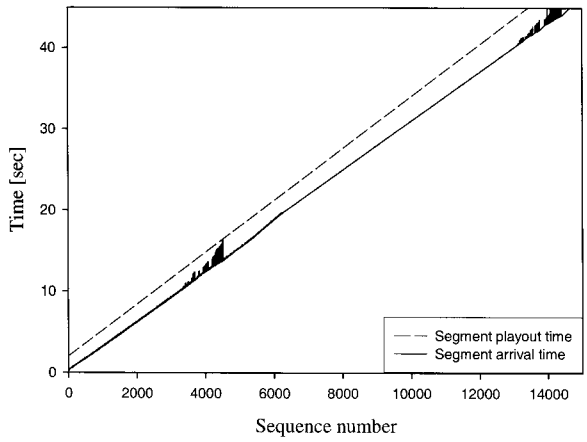
note that RC initially stabilizes the aggregated throughput at roughly 3 Mbps. After 10 seconds, the cross-traffic TCP flow 1 is added. The throughput of connection 1 degrades since the TCP flow 1 interferes with connection 1. After 15 seconds, RC compensates this degradation by allowing connection 3 to increase its throughput. The average aggregated throughput remains therefore at roughly 3 Mbps. After 40 seconds, the cross-traffic TCP flow 2 is added into connection 2. The throughput of TCP-ROME degrades sharply as shown in Fig. 8. Therefore, RC immediately adjusts the rates by increasing the throughput of connection 3. With interfering with TCP flow 2, the average throughput of TCP-ROME is 2.83 Mbps. After 70 seconds, the cross-traffic TCP flow 3 is added into connection 3. As a result, the throughput of connection 3 degrades. In this case, RC does not adjust its rate since the connection 1 and connection 2 cannot make up for a decline of connection 3. With interfering with TCP flow 3, the average throughput of TCP-ROME is 2.65 Mbps.

The results shown with this simulation setup are coarse-grained in the sense that the rate changes are significant since it is difficult for RC to find equilibrium. In more realistic setups where a large number of TCP flows interact and compete, it can be expected that rate changes are more frequent and potentially less drastic. In addition, this simulation shows that the parallel streaming does not guarantee a sustained bandwidth. With TCP flow 3 interference, the long-term aggregated throughput is roughly 2.65 Mbps between 70 and 90 seconds. However, the rate is still far above the throughput obtained from a single-flow TCP streaming.

Fig. 9(a) shows the time on the y-axis as a function of the acknowledged sequence number in the case where no CC is present. The dotted line shows the expected play time of segments as derived in (1). On the other hand, the solid line indicates the practical arrived time of segments. After 10 seconds,



(a)



(b)

Fig. 9. Segment arrival time (a) without CC and (b) with CC.

the almost horizontal line is visible since cross-traffic TCP flow 1 is added. Between 10 and 20 seconds, many segments are late because connection 1 looses packets and retransmits them. The effect of late segments on the real-time streaming application is disastrous: Since TCP-ROME delivers a content to the application as a stream, i.e., it only delivers segments that have been received in sequence, the video streaming rate is delimited in Fig. 9(a). Therefore, during this period, the video stops frequently and the viewing experience is low. And a large number of frames are delivered that have to be buffered at the client. After 20 seconds, the assignment and the delivery are synchronized and the video starts playing smoothly. Instead, with CC, the video experience is quite different, as Fig. 9(b) shows. Every segment is arrived before the expected playout time since CC re-assigns segments that are missing in the sequence. Therefore, the video can be playing smoothly. There are, of course, small variations in the delivery pattern, but at the presented (relevant) resolution, CC can guarantee that a segment is timely delivered as well as eventually delivered. Therefore, we conclude that CC is a necessary component of parallel streaming protocols and that TCP-ROME significantly improves the viewing experience compared to other parallel streaming protocols.

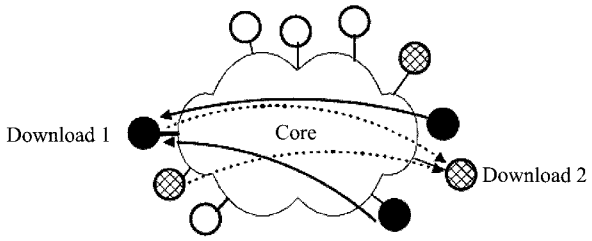


Fig. 10. Simulation setup for multiple interacting TCP-ROME.

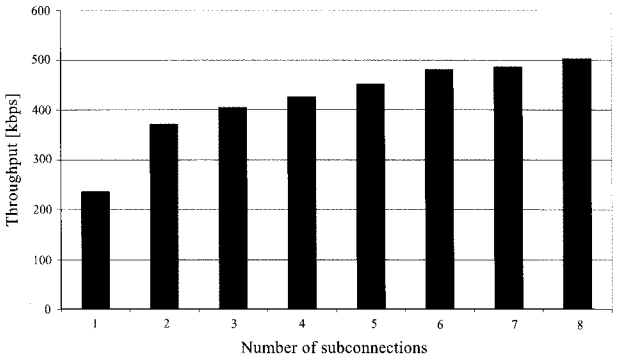


Fig. 11. Parallel streaming performance.

B. *Throughput Evaluation of Parallel Streaming*

Now we address the potential to increase the video experience by using parallel TCP streaming. For this, we choose content replication on multiple distributed servers because this scenario (i) has not been addressed in previous work such as [8], and (ii) does not require support from the network (multipath routing) and is therefore readily applicable. Our objective is to quantify the benefit by simulating a distributed system depicted in Fig. 10 in ns-2. The system consists of a core network with 20 backbone routers that are connected via 100 Mbps links. 100 intermediate systems are randomly attached to this core. The access link bandwidth between the end system and the core router has a uniform distribution of among 56, 128, 256, 512 kbps, and 1 Mbps. Given this network topology, we randomly placed 50 clients on the end systems. These clients stream files that vary in the size from 10–100 MB using n parallel connections. Similarly, $n \times 50$ servers are placed randomly on the end systems. Therefore, clients and servers may share connections to increase the dynamics in the network. Thus, this setup generates a highly dynamic and heterogeneous system where parallel streaming flows interact.

Fig. 11 shows the average streaming rate in kbps as a function of n. The results show that the rate is doubled from using $n = 1$ connection to $n = 8$. For this particular simulation, the biggest improvement is from $n = 1$ to $n = 2$. This improvement shows that a parallel streaming is able to balance the load on the network and make use of unused capacity. For $n > 2$, improvement are only due to temporary usage of links bandwidths and therefore add less improvement. We argue that the general ability to significantly improve the streaming performance is valid for various scenarios in real networks, however, the exact performance improvement depends on multiple factors, such as the

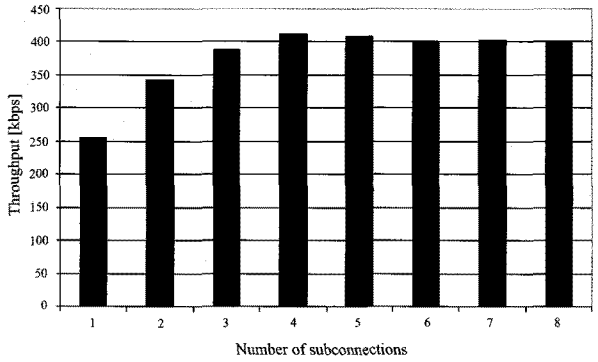


Fig. 12. TCP-ROME streaming performance (required streaming rate = 400 kbps).

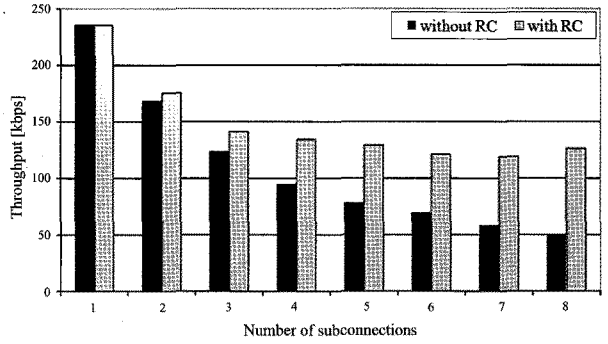


Fig. 13. Performance of concurrent single flow stream.

total capacity of the network, the location of the bottleneck or the number of flows sharing a bottleneck.

C. TCP-ROME Throughput and Fairness Evaluation

The previous experiment shows that the throughput can be improved using parallel streaming. However, an obvious drawback of parallel streaming is the unfairness towards other, concurrent flows. Here, we assess two issues: 1) the ability to maintain the required bit rate for media streaming, 2) the impact of the rate coordination on the real-time streams as well as concurrent single flow streaming.

We reuse the setup of the distributed system of Fig. 10 with 2 modifications. First, only half of the connections (25) use parallel streaming, whereas the others remain single flow streaming. By having concurrent parallel and single flow streaming, we can assess the mutual influence. Second, we define a required streaming rate of 400 kbps. We choose this value because of the access-link bandwidth distribution.

Fig. 12 shows the parallel streaming throughput of TCP-ROME as a function of the number of connections used. However, we report only those connections where the access link from the client to the Internet is larger than 400 kbps. Nodes with slower access links are not considered because they have an upper bound that is given by the access link, not the parallel streaming protocol. The main insight from Fig. 12 is TCP-ROME's ability to maintain a streaming rate that is close to the required streaming bandwidth once sufficient parallel connections are available. That is, with $n > 3$ connections, the protocol is able to aggregate sufficient bandwidth to maintain a constant streaming rate. By adding more connections, the streaming rate remains stable. In fact, it even matches the required bandwidth even better because the protocol has more opportunities to play with the resources.

Fig. 13 depicts the impact of a rate-coordinated parallel streaming protocol on single-flow streaming as a function of the number of connections used by TCP-ROME on the x-axis. Without RC (i.e., without limiting the aggregate throughput of TCP-ROME), the throughput of a single flow decreases as a function of the number of connections, from 235 kbps down to a dismal 50 kbps with $n = 8$ connections. In contrast, with RC, the throughput levels off at roughly 125 kbps. Thus, single flow receive a significantly higher throughput with RC while the real-time streaming achieves the necessary throughput to maintain a perfect streaming quality.

The lessons from these experiments are the ability to trade off performance and fairness in parallel streams. In particular for Web downloads where rich media objects are only one of multiple contents of a Web page and must therefore be streamed concurrently with these other objects, such a trade off is vital to provide a high QoE to the user.

V. CONCLUSIONS

This paper presents TCP-ROME, a transport-layer parallel streaming protocol to support real-time online streaming for multimedia environments. TCP-ROME particularly exploits the potential of the modern Internet to improve the user's QoE, e.g., by streaming content in parallel from multiple distributed replica servers or by leveraging multihoming. We have shown that TCP-ROME is able to improve the QoE via several concepts: The coordination of the content at the granularity of segments, the ability to re-request single segments, and the ability to regulate the streaming rate.

The design of TCP-ROME as a transport-layer solution emphasizes the need to support modern application via a paradigm shift from a point-to-point TCP protocol to a multi-party communication. Parallel streaming advantages are not limited to improving the performance of a single application, but they also increase reliability and reachability of a client, e.g., in wireless networks, and provide load balancing efforts on the network resources.

REFERENCES

[1] P. Mehra, C. de Vleeschouwer, and A. Zakhor, "Receiver-driven bandwidth sharing for TCP and its application to video streaming," *IEEE Trans. Multimedia*, vol. 7, no. 4, pp. 740–752, Aug. 2005.

[2] J. van der Merwe, S. Sen, and C. Kalmanek, "Streaming video traffic: Characterization and network impact," in *Proc. WCW*, 2002.

[3] L. Golubchik, J. Lui, T. Tung, A. Chow, W. Lee, G. Franceschinis, and C. Anglano, "Multi-path continuous media streaming: What are the benefits?," *Perform. Eval.*, vol. 49, no. 1, pp. 429–449, 2002.

[4] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee, "On multiple description streaming with content delivery networks," in *Proc. INFOCOM*, 2002, pp. 1736–1745.

[5] Y. Liang, E. Steinbach, and B. Girod, "Real-time voice communication over the internet using packet path diversity," in *Proc. ACM Int. Conf. Multimedia*, 2001, pp. 431–440.

[6] H. Sivakumar, S. Bailey, and R. Grossman, "Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks," in *Proc. ACM/IEEE SC*, Nov. 2000, pp. 38–38.

[7] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data management and transfer in high-performance computational grid environments," *Parallel Comput.*, vol. 28, no. 5, pp. 749–771, 2002.

[8] B. Wang, W. Wei, Z. Guo, and D. Towsley, "Multipath live streaming via TCP: scheme, performance and benefits," in *Proc. ACM CoNEXT*, 2007, pp. 1–12.

[9] J. Chesterfield, R. Chakravorty, I. Pratt, S. Banerjee, and P. Rodriguez, "Exploiting diversity to enhance multimedia streaming over cellular links," in *Proc. INFOCOM*, Mar. 2005, pp. 2020–2031.

[10] T. Nguyen and A. Zakhor, "Multiple sender distributed video streaming," *IEEE Trans. Multimedia*, vol. 6, no. 2, pp. 315–326, Apr. 2004.

[11] J.-W. Park, J. Kim, and R. Karrer, "TCP-ROME: A transport-layer approach to enhance quality of experience for online media streaming," in *Proc. IWQoS*, June 2008, pp. 249–258.

[12] P. Saroiu and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proc. MMCN*, Jan. 2002, pp. 38–38.

[13] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy, "An analysis of internet content delivery systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 315–327, 2002.

[14] B. Cohen, "Incentives build into robustness in bittorrent," May 2003. [Online]. Available: http://bittorrent.com/bittorrentcon.pdf

[15] T. Nguyen and S. Cheung, "Multimedia streaming using multiple TCP connections," in *Proc. IPCCC*, Apr. 2005, pp. 215–223.

[16] P. Shah and J.-F. Paris, "Peer-to-peer multimedia streaming using bittorrent," in *Proc. IPCCC*, Apr. 2007.

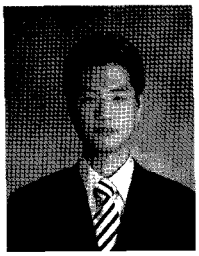
[17] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributed streaming media content using cooperative networking," in *Proc. NOSSDAV*, May 2002.

[18] R. Rejaie and A. Ortega, "PALS: Peer-to-peer adaptive layered streaming," in *Proc. ACM NOSSDAV*, June 2003, pp. 153–161.

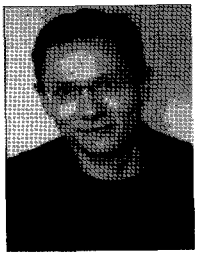
[19] R. P. Karrer and E. W. Knightly, "TCP-PARIS: A parallel download protocol for replicas," in *Proc. WCW*, 2005, pp. 15–25.

[20] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, pp. 66–74, 1995.

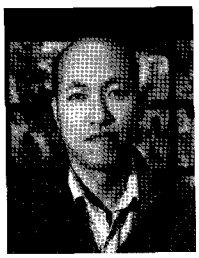
[21] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling tcp throughput: A simple model and its empirical validation," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 303–314, 1998.



Ju-Won Park received his B.S. degree in information and telecommunication engineering from Hankuk Aviation University in 2002. He received the M.S. and Ph.D. degrees from Information and Communications Engineering Department at Gwangju Institute of Science and Technology (GIST) in 2004 and 2010, respectively. From September 2010, he has joined at KT (Korea Telecom). His research interests are focused on multipoint communication and networking issues including real-time parallel streaming and multicast routing over optical networks.



Roger P. Karrer received his Ph.D. in computer science from ETH Zurich, Switzerland. He spent 2 years as a PostDoc at Rice University. Since 2005, he has been a Senior Research Scientist at Deutsche Telekom Laboratories in Berlin. His research interests include the design of networked systems, including wired and wireless protocols and multimedia applications.



JongWon Kim received the B.S., M.S., and Ph.D. degrees from Seoul National University, Seoul, Korea, in 1987, 1989 and 1994, respectively, all in control and instrumentation engineering. In 1994–1999, he was with the Department of Electronics Engineering at the Kongju National University, Kongju, Korea, as an Assistant Professor. From 1997 to 2001, he was visiting the Signal and Image Processing Institute (SIPI) of Electrical Engineering - Systems Department at the University of Southern California, Los Angeles, CA. USA, where he has served as a Research Assistant Professor since Dec. 1998. From September 2001, he has joined as an Associate Prof. at the Department of Information & Communications, Gwangju Institute of Science and Technology (GIST, formerly known as K-JIST), Gwangju, Korea, where he is now serving as a Professor. He is now focusing on networked media systems and services to support dynamic composition of immersive media-centric services over the wired/wireless IP convergent networks. Dr. Kim is a Senior Member of IEEE, a Member of ACM, SPIE, KICS, IEEK, KISE, and KIPS.