Peer-to-Peer Networking and Applications (2021) 14:2038–2060
https://doi.org/10.1007/s12083-021-01134-1

Delay - aware bandwidth estimation and intelligent video transcoder in mobile cloud

S.P. Tamizhselvi[1] · **Vijayalakshmi Muthuswamy**[2]

Received: 16 May 2020 / Accepted: 19 March 2021 / Published online: 1 May 2021

Abstract

In recent years, smartphone users are interested in large volumes to view live videos and sharing video resources over social media (e.g., Youtube, Netflix). The continuous streaming of video in mobile devices faces many challenges in network parameters namely bandwidth estimation, congestion window, throughput, delay, and transcoding is a challenging and time-consuming task. To perform these resource-intensive tasks via mobile is complicated, and hence, the cloud is integrated with smartphones to provide Mobile Cloud Computing (MCC). To resolve the issue, we propose a novel framework called delay aware bandwidth estimation and intelligent video transcoder in mobile cloud. In this paper, we introduced four techniques, namely, Markov Mobile Bandwidth Cloud Estimation (MMBCE), Cloud Dynamic Congestion Window (CDCW), Queue-based Video Processing for Cloud Server (QVPS), and Intelligent Video Transcoding for selecting Server (IVTS). To evaluate the performance of the proposed algorithm, we implemented a testbed using the two mobile configurations and the public cloud server Amazon Web Server (AWS). The study and results in a real environment demonstrate that our proposed framework can improve the QoS requirements and outperforms the existing algorithms. Firstly, MMBCE utilizes the well-known Markov Decision Process (MDP) model to estimate the best bandwidth of mobile using reward function. MMBCE improves the performance of 50% PDR compared with other algorithms. CDCW fits the congestion window and reduces packet loss dynamically. CDCW produces 40% more goodput with minimal PLR. Next, in QVPS, the M/M/S queueing model is processed to reduce the video processing delay and calculates the total service time. Finally, IVTS applies the M/G/N model and reduces 6% utilization of transcoding workload, by intelligently selecting the minimum workload of the transcoding server. The IVTS takes less time in slow and fast mode. The performance analysis and experimental evaluation show that the queueing model reduces the delay by 0.2 ms and the server's utilization by 20%. Hence, in this work, the cloud minimizes delay effectively to deliver a good quality of video streaming on mobile.

Keywords Mobile bandwidth estimation · Dynamic congestion window · Video processing · Quality of service · Queueing delay · Intelligent video Transcoder · Mobile cloud

1 Introduction

The traditional mobile phones play a significant role in effective communication, especially in transmission media. Although they support heterogeneous networks, the mobile phones do not support many multimedia applications [1, 2]. Hence, smartphone networks, such as 5G and 6G support video streaming. In this work, the author discusses briefly the 6G wireless network which operates with the current telecommunication technologies such as IoT, cloud and edge computing, and Big data. To handle the smart and secure building, the author proposed a new mechanism called Secure Cache Decision System (CDS) in the wireless network. This provides a safer internet service for sharing and scalability of data [3]. Kaium et.al discuss the future view of IoT management in the context of cloud, fog, and mobile edge computing [4]. Ahmed et. al provides a different mechanism for the benefit of IoT applications to handle the fog computing services like low latency. This model is mainly designed to achieve scalability and partial

✉ S.P. Tamizhselvi
tamizh8306@gmail.com

Vijayalakshmi Muthuswamy
vijim@annauniv.edu

[1] Department of Computer Science & Engineering, Faculty of Information & Communication Engineering, KCG College of Technology, Anna University, Chennai, India

[2] Department of Information Science & Technology, Faculty of Information & Communication Engineering, College of Engineering Anna University, Chennai, India

validation is committed only in read and write transactions [5].

The prerequisite of a smartphone is to deliver a good quality of the video to the viewers. Some of the multimedia challenges faced by a streaming video are delay and network performance. [6]. Apart from these QoS(Quality of Service) challenges, smartphones need to solve network QoS issues such as a battery, energy consumption, low bandwidth, latency, and packet loss. Therefore, the solution is integrating smartphones with the cloud. Mobile Cloud Computing (MCC) defines on-demand service and resource-constrained mobile devices offloads tasks from mobile devices to cloud. Thus it saves energy, power, the battery in mobile. This work proposes the offloading process of the mobile application in the cloud. The code is going to be offloaded to the cloud. Therefore, the cloud is responsible to parallel execute the task independently [7]. This will reduce the battery consumption and execution time of mobile [8–10]. In recent years cloud-based IoT applications useful for day to life activities. For example, this work introduced blockchain-based authentication and authorization services for smart city applications. In this work, the security information is decentralized for the smart city ecosystem, and blockchain technology is used to detect the management and provide the access control which is integrated within the FIWARE platform [11]. The author proposed a novel algorithm for watermark embedding and extraction by using a synergetic neural network. In this work, the optimal PSNR is achieved efficiently compared with other models [12].

Lai et al. focus mainly on delay and packet loss as network parameters [13]. Hence, the need is to estimate the available bandwidth based on the profile of the mobile. Profile means history of the mobile which is already registered in the cloud server. The mobile user activates the cloud monitors services, and the device manager maintains the various QoS parameters such as bandwidth, link capacity, network status, signal strength, buffering stream, and energy. Whenever, a device state change occurs, the device information updates in the cloud server for the registered mobile users and the estimated bandwidth fits the congestion window. There is a lot of research on bandwidth estimation [14], and TCP congestion control algorithms exist in the literature [15–20]. Lai et al. shows that the buffering in the transport layer reduced to deliver a good quality of video live streaming in mobile [13]. As far as cloud networks are concerned, Leong et al. input the necessary parameters as Round Trip Time (RTT) and video segments for bandwidth estimation in heterogeneous networks [21].

We propose MMBCE to estimate the bandwidth of mobile using MDP. Here, MDP defines the available bandwidth of each video request of mobile with different actions and reward functions for each region or state. The reward function derives with Best First Reward Search(BFRS) algorithm using heuristic value, and the cloud server dynamically fits the congestion window. After estimating the bandwidth, in the cloud server, the next proposed QVPS schedules the video process as per the estimated bandwidth, arrival rate of video request, and queue length of individual video server with the M/M/S queueing model. QVPS identifies the minimum queue length of a video server for processing the video request. The server significantly minimizes the computational workload and processing time of each video processor, which directly implies the reduction in a video processing delay. To calculate the average output video chunk size and average transcoding time of the server, we propose IVTS for deciding the different transcoding modes by comparing maximum delay with average delay, maximum chunk size with average chunk size. After mode selection, the transcoding server selects the optimal transcoder. Thus, IVTS reduces both transcoding delay and resource utilization.

1.1 Contribution

1. We propose the MMBCE algorithm to estimate the bandwidth of the mobile in the cloud network.
2. CDCW is the new congestion algorithm proposed to fit the congestion window dynamically in the cloud.
3. We compared MMBCE with an existing TCPHas, TCPW, Cubic, and Reno algorithms to evaluate the performance metrics of bandwidth, packet delivery ratio, and throughput.
4. Our CDCW algorithm compared with the existing congestion algorithm to reduce PLR and improve the congestion window size.
5. We proposed QVPS using the M/M/S queueing model to schedule the video process, minimize video server utilization and process time.
6. IVTS intelligently selects an optimal transcoder server, and comparative analysis made with existing work, which reduces the transcoding delay.
7. Overall, the above-proposed algorithms improve the network QoS parameters and deliver a quality of video streaming in the mobile cloud.

This paper is organized as follows: Section 2 briefly covers the background on three areas (1) Bandwidth estimation and congestion window, (2) Queueing model, (3) Transcoding. Section 3 gives a detailed description of the proposed framework of delay – aware bandwidth estimation and intelligent video transcoder in the mobile cloud. Next, Section 4 deals with three components such as experimental setup, case study and performance metrics, then we present the results and discussion in Section 5. Finally, Section 6 concludes with future work.

2 Related work

In this section, we review the Network QoS parameters, namely, Bandwidth estimation and congestion window, Queueing Model for video processing, and Transcoding.

2.1 Bandwidth estimation and congestion window

We first address the bandwidth estimation in a wireless network and study their influence on the parameters of TCP congestion control variants. The famous TCP variants such as Reno [22], Binary Increase Congestion control (BIC) [23], CUBIC [24], TCPWestwood (TCPW) [25] and TCPHas [26] have been analyzed and compared with proposed work. The detailed working of each algorithm mentioned above discussed below.

Reno Among different TCP congestion control variants, TCP Tahoe, detects the packet loss after the time interval expires. Then the performance of the algorithm becomes slow, and transmission flow also decreases. [15, 27, 28]. Since Slow Start, Congestion Avoidance, and Fast Retransmit techniques followed in Tahoe. Reno is similar to Tahoe but applicable only when three duplicate acknowledgments (ACKs) receives at the sender, then it incur packet loss in a network. A Fast recovery algorithm overcomes the drawback of this method sets a threshold value to retransmit the lost packet. After receiving the third duplicate ACKs, the congestion window is fit to half, based on the threshold value. The threshold has added to three times of Maximum Segment Size (MSS) [29], which assigns to the congestion window to reduce packet loss. However, in experimental scenarios, when multiple packet loss occurs in the same window, Reno exhibits the capability to improve the throughput. To address these challenges, the authors introduce a new algorithm, namely, a queueing delay-based rate control algorithm. Lin et al. and Wang et al. observe that Reno does not show the ability to identify the impact of a packet loss. Hence, we believe that full link utilization may not achieve [30, 31]. The analysis of RENO incurs the low bandwidth with minimum delay. To improve this, we will move on to BIC.

BIC Next, we discuss the packet loss-based TCP congestion control algorithm is BIC [23, 32]. Based on the maximum window size, a binary search technique fits the destination window. The window selects the midpoint of the maximum and minimum level, to enhance the window size than BIC, CUBIC. To simplify the window size and friendliness, BIC enhances to CUBIC.

Cubic Cubic, also considered a packet loss-based congestion algorithm fixes the Congestion window size (cwnd) through a cubical function. Two shaping approaches explain in CUBIC. While TCP handles congestion by slow start mechanism, CUBIC handles utilizing a Hybrid Slow-Start mechanism [33]. Packet loss is identified based on ACK and RTT. The main drawback of CUBIC is time consumption and more bandwidth requirement.

TCPWestwood (TCPW) In this method, the sender analyzes the ACKs arrival rate to evaluate the RTT and bandwidth. If RTT is minimum, the congestion is said to be low and vice-versa. In some situations, packet drops occur at the receiver. In such cases, TCP-Westwood [25] determines a new BDP (Bandwidth Delay Product) and allocates it to a threshold values instead of assigning half of the limit. Later, Yet another scheme, namely Persistent Non-Congestion Detection, was introduced to determine the congestion [34]. To improve throughput, Westwood adjusts the congestion window. Instead of fixing the window, we strive to incorporate TCPHas, to fit the congestion window.

TcpHas TCPHas is an entirely new TCP congestion algorithm that helps estimate bandwidth and fits the congestion window. In TCPHas, the quality levels of video expect the bandwidth of the network, queueing delay, a historic level of video, and bit rates. This rate calculates with the help of shaping function. The sending rate of this function should be suitable for the encoding bitrate of the estimated optimal quality level. Since the reduction happens in the queueing delay and packet loss rate, the QoS improves live streaming [26, 35]. In our work, we intend to reduce the same parameter with the help of the queueing model. The above survey examines the TCP congestion control variants for estimating the bandwidth. Then, the video packets are processed in the video server as per the bandwidth by utilizing the different queueing models. Further, the discussion proceeds with the survey of the queueing model.

2.2 Queueing model for video processing

The theory of queueing is a mathematical model defined for measuring the capacity of the queue length and the waiting time to be produced in a network [36, 37]. M/M/1, M/M/S, M/G/1, M/G/N are some of the different types of queueing model. M/M/1 is the queueing model where the arrival of the video request follows a Poisson distribution with exponential service rate. The drawback of the model is always single server handles only one queue at a time. As a result, the waiting time increases. To solve this problem,

M/M/S was applied to increase system performance by having multiple servers. Therefore, this M/M/S and M/G/N model are used for the application of multi-service cloud computing. The resulted in processing delay minimization and response time, maximizing the average queue length compared to other models [38, 39]. The video packets in the queue are ready to enter in the transcoding server. To process, the server operation, the analysis of existing transcoding work has been explained as follows.

2.3 Transcoding

Nearly every corporate public cloud service, such as Spritdsp, Amazon Web Services (AWS) and Bitcodin helps manage the transcoding service of a user to create a different solution for uploading video files. In multiple devices, a user can experience online video content with varying sizes of resolution [40].

The work in [41] jointly measured transcoding and delivery of video based on user priorities in the geographical areas and video varieties of the Content Delivery Network (CDN). The existing literature survey [42, 43] does not contain the capacity scaling problem [44]. In microdata center, based on the broadcast request, Heuristic schedules the live transcoding jobs dynamically. The schedule allows small degradation in the output [45]. To overcome these problems, Lei Wei.et al. introduced a new Cloud-based Online Video Transcoding system (COVT) [46] to provide a QoS guaranteed video, using the profiling approach to achieve the different metrics for performing tasks with separate infrastructures. Based on the queueing model, the QoS metrics design, the findings from the experimental analysis show that a COVT dynamically resolve the resource limitation, and the task is programmed at runtime to implement an assured QoS.

2.4 Motivation

In the literature, we find that TCP Tahoe detects the packet loss only after a timeout occurs. Whereas, Reno does not detect packet loss within a single RTT. As far as the rest of the techniques like BIC, CUBIC, TCP Westwood are concerned, they commonly face fairness issues. On the other hand, TcpHas, a reasonably new method requires slight performance improvement in delivering live streaming service, especially when the network is stable. Hence, our motivation provides a solution for the issues of packet loss, delay, network performance in existing techniques. Thus the need for a framework to handle bandwidth estimation, video processing, transcoding challenges and stream the quality of the video in mobile.

3 Delay – aware bandwidth estimation and intelligent video transcoder in mobile cloud

The proposed framework for delay – aware bandwidth estimation and intelligent video transcoder in a mobile cloud has two major components. (1) Mobile Client, (2) Cloud Server.

3.1 Mobile client

This component has two significant elements, namely, a video player and a decoder. Smart mobile users request for different types of video from heterogeneous networks. According to video requests, a decoder decodes video content from the cloud server with good quality. Then, video streaming happens in a real-time player of mobile.

3.2 Cloud server

The cloud server manages three main components, namely, proposed MMBCE, QVPS, and IVTS. The mobile profile manager maintains the historical information of mobile and helps to estimate the bandwidth. MMBCE applies Markov Decision Process (MDP) to achieve available bandwidth. Then, BFRS in server estimates the bandwidth of mobile in the cloud environment. To stream good quality video, we define the action and reward function in each state to calculate the goal state. Then fits the congestion window dynamically by using CDCW. Then, QVPS processes the video as per the bandwidth. After video processing, IVTS transcodes the video content. Once the transcoding process completes, the cloud server streams the video to the mobile client. We will discuss the three proposed components briefly in the following section. Firstly, the working principle of MMBCE explains in the next subsection.

3.2.1 Bandwidth estimation process formulation

The cloud server estimates bandwidth and analyzes the mobile profile manager. By considering the available bandwidth, the current bandwidth estimation assigns to an individual mobile, and it will act as an agent. The agent works on the principle of reinforcement learning task [47]. In the cloud environment (Fig. 1), the actual bandwidth passes to the agent of each mobile (State S). Then, the agent identifies the best bandwidth correspondingly. For each bandwidth, the cloud environment, in turn, responds with a reward to the bandwidth estimation agent. The Markov Decision Process (MDP) develops to provide better

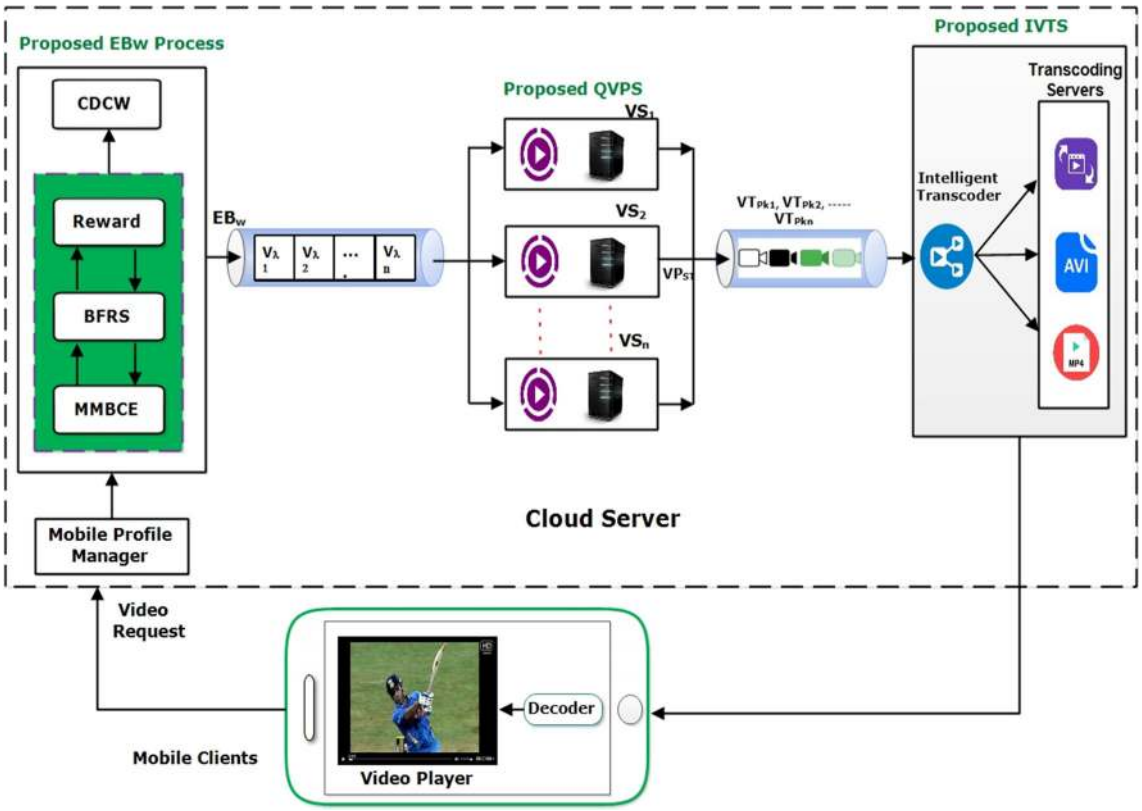


Fig. 1 Framework for the delay – aware bandwidth estimation

bandwidth estimation, and the transition state behavior of the same Markov process [48]. The Markov chain is defined to derive the transition state model as follows:

Markov Chain (Definition) Let State X denote the state of the available bandwidth of all mobiles with different locations. X ∈ d(a,b) of the Euclidean distance for heterogeneous mobile. The dynamic movement of mobile measures with the discrete-time interval [49].

$$P(X_{Mt+1} = X)|X_t, X_{t-1}, ...X_0) = P(X_{Mt+1} = X|X_{Mt}) \quad (1)$$

$$TP(y_{Mt}|x) = P(X_{Mt+1} = y|X_{Mt=x}) \quad (2)$$

Markov Decision Process (MDP) is defined as a four tuples MDP = (X, A, TP, r). X, A are the state and action space. TP(y/x, a) is the transition probability with

$$TP((y|x), a) = P(X_{Mt+1} = y|X_{Mt} = x, a_t = a) \quad (3)$$

where r(x, a, y) is reward of transition probability (x, a, y). The estimated bandwidth is mapped with the actual B_w bandwidth for each step.

The study of MDP helps to develop our bandwidth estimation algorithm MMBCE. The new bandwidth algorithm description is as follows:

Algorithm 1 Markov mobile bandwidth cloud estimation (MMBCE).

Input: Number of Video Request (NV_{Req}), Segment Size (S_z), Round Trip Time (ΔT)
Output: Best Available Bandwidth (AB_w)

```
Initialize action a ← - 1, MaxMB_w ← - ∞, state S ← NV_Req
//Calculate all Available bandwidth AB_w(s) for state S
for i = 1 to S
ΔT = T_i - T_{i-1}
AB_w[i] = S_zi / ΔT
EB_w[i] ← Best_FirstRewardSearch (S, AB_w[i])
if EB_w[i] ≥ MaxMB_w[i] then
    MaxMB_w[i] ← EB_w[i]
    AB_w[i] ← a[i]
end
end for
return AB_w[i]
```

Algorithm 1 Description In the cloud server, to estimate bandwidth, we propose MMBCE, to measure the best available bandwidth of each video request of the mobile

user. This performance of the available bandwidth has evaluated; three inputs passed to the MMBCE algorithm are the number of video requests, segment size, and RTT. Here, the number of requests considers the number of states. Therefore, based on the segment size and time interval, each state of mobile's available bandwidth is calculated. Then, the output of the state and the available bandwidth are input to the Best-First Search() procedure (Algorithm 2). The output of Algorithm 2 assigns EB_w in Algorithm 1(MMBCE). The next step is to compare the estimated bandwidth of each mobile and maximum bandwidth. If EB_w is greater than the maximum, the estimated bandwidth assigns to maximum bandwidth and action of each state to available bandwidth. This comparison repeats for all the states of the mobile. Next, we move on to determine the best first reward from the transition matrix and states. The BFRS function is as follows:

Algorithm 2 Best first reward search Algorithm.

Input: Number of State (N_s), Available Bandwidth (AB_w))
Output: Estimate Bandwidth (EB_w))
1 **Function** `Best_FirstRewardSearch` (N_s, AB_w, I_s) **:**
2 Initialize TPM [ ] [ ] ← 0, TAB_w [ ] [ ] ← 0, N_s ← 0, I_s ← 0
3 SC ← Array of State Count for mobile transition
4 C ← Individual Count of mobile state
5 TPM [ ][ ] ← TPM [N_s] [N_s]
6 **foreach** $i = I_s; i \leq N_s; i$++ **do**
7 **foreach** $j = 1; j \leq N_s; j$++ **do**
8 TAB_w[i][j] ← AB_w[i]
9 **end**
10 **end**
11 **foreach** $i = I_s; i \leq N_s; i$++ **do**
12 **foreach** $j = 1; j \leq N_s; j$++ **do**
13 $SC[i][j] \leftarrow \sum_{i=1}^{Ns} C[i][j]$
14 $TPM[i][j] \leftarrow \frac{C[i][j]+1}{SC[i][j]+Ns}$
15 **end**
16 **end**
17 R ← Reward(N_s, TPM, I_s)
18 $EB_w \leftarrow R + \sum_{i=1}^{Ns} TPM[i][j]$ **return** EB_w;
19 **End Function**

Description of Algorithm 2 The mobile network faces different challenges in traffic and time intervals. When different mobile users request the video in the cloud, the bandwidth estimation of every mobile becomes a complicated task. To overcome this issue, the Markov channel model proposed to estimate with a time-varying random variable. The Markov model has two elements (1) Transition matrix, (2) state model. To generate a transition matrix, TPM (Refer 2) represents the total number of mobile transitions for each state i to j. According to the TPM, the state model is indicated by n with n x n matrix. The number of states, the available bandwidth are received from algorithm 1 given as input to Algorithm 2. The steps 6 to 10 makes a corresponding entry in the transition matrix of available bandwidth start from an initial state to the total number of states. Then step 11 through 16 counts the number of requests for each state and fills the TPM. To assign the reward for each state, step 17 calls the reward function (Algorithm 3). Next, the reward adds along with TPM. Finally, EB_w returns as the output of Algorithm 2.

Description of Algorithm 3 and 4 According to the MDP, a reward is added to the estimated bandwidth. To select the reward, algorithm 3 determines the best reward, which falls under the heuristic or informed search calculation. The two inputs, such as many states and TPM, assigns to algorithm 3. A priority queue also initializes for vertices, and the maximum value assigns as 999. There are two states introduced in this algorithm, namely computing state and goal state. Step 8 through 11 determines the heuristic function for all vertices until it reaches the goal state. The level 13 through 14 inserts a new vertex as a computing node for each neighboring vertex v in the priority queue and is marked as unvisited.

Here, the initial state is added in the priority queue to identify the computing state and labeled as a visited node and marked as 1. Step 16 through 26 calculates the minimum heuristic value for every computing state, and the goal state is marked as visited. Step 16 checks the priority queue is empty or not. If it is not empty, determine the computing state using getStateWithMin_Heuristic() procedure. Now, the goal state assigns 1. Step 19 compares the goal state with many states. If it is less than or equal to a number of states, it checks the step 21. If this condition is satisfied, that particular vertex is added to the priority queue and marked as visited. The goal state is assigned 1. Next, the goal state increments and repeats until the priority queue empty. Then the minimum value is removed from the priority queue, and that reaches a final goal state.

All the above four algorithms have devised to calculate the best bandwidth estimation at different states in the heterogeneous network. Next, to minimize the packet loss, the cloud dynamically adjusts the congestion window as per the estimated bandwidth. The steps involved are explained in the CDCW algorithm as follows:

Algorithm 3 Reward search Algorithm.

Input: Number of State (N_s), Transition Probability Matrix (TPM)
Output: Best Reward (R)

Function `Reward` (N_s, *TPM*, I_s) **:**
  Initialize PriorityQueue<Vertex>pq, Hv [] = 0
  MAX_VALUE ← 999, n ← N_s
  pq ← new PriorityQueue<Vertex>((N_s, new Vertex()) C_s ← Computing state, G_s ← Goal State
  N_s ← 0
  Initialize visited[N_s]
  /* Calculate heuristic values */
  for $vertex \leftarrow 1$; $vertex \leq n$; $vertex$++ **do**
    f(vertex) ← Distance(G_s)
    Hv[vertex] ← f
  end
  $CH_v \leftarrow H_v$
  pq.add(new Vertex(I_s, $CH_v[I_s]$))
  visited[I_s] ← 1
  /* To reach Goal State */
  while *! pq.isEmpty()* **do**
    C_s ← getStateWithMin_HeuristicValue(pq)
    G_s ← 1
    while $G_s \leq N_s$ **do**
      Vertex v ← Vertex(G_s, $CH_v[G_s]$)
      if *(TPM[*C_s*][*G_s*] ≠ MAX_VALUE && Cs ≠ Gs) && visited[Gs] = 0)* **then**
        pq.add(v) visited [G_s] ← 1
      end
      G_s++
    end
  end
  return G_s;
End Function

Algorithm 4 HeuristicValue.

Input: PriorityQueue<Vertex>pq
Output: Goal State (G_s)

Function `getStateMin_HeuristicValue` (*PriorityQueue<Vertex>pq*) **:**
  Vertex v ← pq.remove();
  return v
End Function

Algorithm 5 Cloud dynamic congestion window (CDCW).

Input: Estimate Bandwidth (EB_w)
Output: Dynamic Congestion Window (Dcwnd)

Initialize $MaxM_W$ ← 64 Kbps
if *3 DUPACKs are received* **then**
  $MaxS_z \leftarrow (EB_w * \Delta T) / S_z$
end
/* congestion avoid. */
if $MaxM_W > MaxS_z$ **then**
  Dcwnd = $MaxS_z$
end

Description of Algorithm 5 CDCW algorithm takes the estimated bandwidth as input from Algorithm 1. The maximum mobile window size initializes to 64 kbps. The cloud server receives three duplicate acknowledgments to calculate $MaxS_Z$. The dynamic congestion window fits to avoid congestion by comparing the $MaxM_W$ with $MaxS_Z$. The cloud server minimizes the packet loss as per the estimated bandwidth and congestion window through this algorithm. Then, the video server in the cloud receives the packets in the queue. In the next section, we elaborately explain the operation of the video processing server.

3.2.2 Video processing server

Assume that mobile users are in a heterogeneous network. These users send video requests to the cloud server. Since multiple mobile clients send the request, they fall under the M/M/S queueing model, also known as buffering. Consequently, video processing happens through the server. In this work, we applied a queueing model named multi-server system to accept the request from the mobile and issue the appropriate services in the cloud. This architecture works on the open Jackson network [38, 50]. The symbols and description list are given in Table 1:

Video Processing Server using M/M/S Queueing model formulation Model : (M/M/S): (∞/ FIFO) [51] This model is based on the following assumptions:

- The number of video request arrivals follows a Poisson distribution with a mean arrival rate V_λ.
- The response time has an exponential distribution with the average response rate $AV\mu$.
- Arrivals are infinite population α.
- Number of video requests follows First-in First-out basis (FIFO).

Table 1 Table of notations

Symbols	Description
V_λ	Mean of Video arrival rate
AV_μ	Average response rate of video request
A	Video arrivals are infinite population
ρ	Traffic intensity of the Video Processing Server
N_{VP}	Expected number of process in video server
QL_n	Number of process expected in the server queue
W_q	Mean waiting time in queue
W_S	Average waiting time in the server
EL_q	Expected length of non-empty queue
P_n	Probability of n process in the server
VS	Video server
VP_{ST}	Video Processing service time
$C(n, \rho)$	Erlang's C distribution video processing
V_{SZ}	Video Segment Size
$MaxM_W$	Maximum mobile window size 64kB

This model provides a solution for mobile users to satisfy demand. It depends on the values of V_λ and AV_μ. If $V_\lambda \geq AV\mu$, then the waiting line of the Video Processing Server (VPS) would increase without limit. Therefore, it must satisfy a condition as $V_\lambda \leq AV\mu$.

As mentioned earlier, traffic intensity of the VPS $\rho = V_\lambda$ / S * $AV\mu$ which refers to the probability of time. High traffic intensity makes the server busy. The probability of VPS is idle, or there are no video processes in the system, $VP_0 = 1- \rho$. The probability of a single process in the server is $VP_1 = \rho\ VP_0$. Similarly, the probability of two processes in the server $VP_3 = \rho\ VP_1 = \rho 2\ VP_0$. The probability of n process in the server is

$$VP_n = \rho n VP_0 = \rho n(1-r) = (V_\lambda / AV_\mu) n VP_0 \tag{4}$$

The expected number of processes in the video server is given by,

$$N_{vp} = \sum_{n=1}^{\alpha} nVP_n = \sum_{n=1}^{\alpha} n(1 - V_\lambda / AV_\mu)(V_\lambda / AV_\mu)^n \tag{5}$$

$$N_{vp} = \frac{V_\lambda}{AV_\mu - V_\lambda} \tag{6}$$

The expected number of process in the server queue is given by,

$$QL_n = \sum_{n=1}^{\alpha} (n-1) VP_n \tag{7}$$

$$= \sum_{n=1}^{\alpha} nVP_n - \sum_{n=1}^{\alpha} VP_n \tag{8}$$

$$QL_n = \frac{V_\lambda^2}{AV_\mu(AV_\mu - V_\lambda)} = \frac{\rho^2}{1-\rho} \tag{9}$$

With an average arrival rate of video V_λ, the average time between the arrivals is $\frac{1}{V_\lambda}$. Therefore, the mean waiting time in the queue, Wq, calculates the average time between the arrivals of video packets and the average queue length.

$$W_q = \left[\frac{1}{V_\lambda}\right]\left[\frac{1}{1-\rho}\right] \tag{10}$$

$$= \left[\frac{1}{V_\lambda}\right]\left[\frac{V_\lambda}{AV_\mu - V_\lambda}\right] \tag{11}$$

Similarly the average waiting time in the server, W_s

$$W_s = \left[\frac{1}{V_\lambda}\right]\left[\frac{1}{1-\rho}\right] \tag{12}$$

$$W_s = \left[\frac{1}{AV_\mu - V_\lambda}\right] \tag{13}$$

Expected length of non-empty queue,

$$EL_q(m/m > 0) = \frac{AV_\mu}{AV_\mu - V_\lambda} \tag{14}$$

Probability that there are n video segments in the server,

$$VP_n = \left[\frac{1}{V_\lambda}\right]^n \tag{15}$$

Probability that there is no video process in the cloud server,

$$VP_0 = \frac{1 - V_\lambda}{AV_\mu} \tag{16}$$

Video Server (VS) queue is assigned to provide the service Video Processing Service Time (VP_{ST}) given by VP_{ST1} .... VP_{STn}. This is formulated in the following equation as

$$VP_{ST} \leftarrow \frac{1}{AV_\mu} + \frac{c(S,\rho)}{n\left(AV_\mu - V_\lambda\right)} \tag{17}$$

where n is the number of VS. Then, μ_{VSR} and γ_{VAR} are the parameters of the M/M/S model which represents μ_{VSR} video service rate and γ_{VAR}, video arrival rate. Therefore, γ_{VAR} and $\mu_{VSR} = \mu_i$. where i=1,2 ... S. In some cases, the new video request which is forwarded to the video server, the Erlang's C distribution will be denoted as [52]

$$C(S,\rho) \leftarrow \frac{\left(\frac{(S*p)^S}{S!}\right)\left(\frac{1}{1-\rho}\right)}{\sum_{k=0}^{S-1}\left(\frac{(S*p)^k}{k!}\right)\left(\frac{(S*p)^S}{S!}\right)\left(\frac{1}{1-\rho}\right)} \tag{18}$$

The above M/M/S model equation is used to calculate the total video processing service time in QVPS algorithm. The steps are explained as follows:

Algorithm 6 Queue based video processing server(QVPS).

Input: No. of Video Processing Request (V_λ), Average response rate (AV_μ)
Output: Total video processing service time (VP_{ST})

```
Initialize n ← V_λ
Traffic intensity ρ ← V_λ / AV_μ
foreach VP_i from 1 to n do
    if V_λ ≥ AV_μ then
        /* To Calculate the video processing service time */
        a. Calculate expected no of process in the video server (N_VP)
        b. Calculate expected no of process in the server queue is
        QL_n ← Σ_{n=1}^{∞} (n-1) P_n
        c. Calculate the mean waiting time in queue W_q
        d. The aveage waiting time in the server (W_s)
        e. Probability for n process in the server P_n
    else
        Probability that there is no video request in the server,
        P_0 ← 1 - V_λ / AV_μ
    end
    /* calculate the total video processing service time */
    /* // In some cases, if the new video request is forwarded to the video server by using Erlang's C distribution C (S, ρ) by using */
    C(S,ρ) ← ((S*p)^S/S!)(1/(1-ρ)) / Σ_{k=0}^{S-1}((S*p)^k/k!)((np)^n/n!)(1/(1-ρ))
    VP_ST ← 1/AV_μ + c(S,ρ)/S(AV_μ - V_λ)
end
```

According to the bandwidth prediction of the cloud server, the video packets arrive in the VPS. Table 2 diminishes the arrival rate, the queue length of the server, and the probability of an idle server. The arrival rates of video packets examine with the servers (2, 4, and 6). In general, whenever the arrival video packet arises, the probability of idle server decreases. Table 2 shows for the number of servers is 2. Similarly, for the same arrival rate, the VPS increases, which increases the probability of idle server by an average of approximately 2%. The same arrival rate process with more number of servers. Hence, we have chosen the number of servers as 6. Then, there is an automatic decrease in queue length. This reduction happens due to the implementation of the M/M/S Queueing model in the VPS. The overall advantage is to reduce the computational overhead and energy consumption. Thus, we significantly minimize video processing delay. After processing the video packets, the cloud server deals with the transcoding server as follows.

3.3 Transcoding server (TS)

The TS mainly focuses on two parameters, namely Transcoding Mode(T_M) and different video types (V_d).

Table 2 Measurement of Queue length and Probability of idle server with different arrival rate

No. of Server	Arrival rate (No. of request /time)	Queue length in Q	Probability idle server (%)
2	100	0.083	50.0000
	105	0.098	48.1480
	110	0.113	46.3130
	115	0.128	44.4780
	120	0.143	42.6430
	125	0.158	40.8080
	130	0.173	38.9730
4	100	0.066	51.3310
	105	0.082	49.6450
	110	0.098	47.9973
	115	0.114	46.3498
	120	0.13	44.7023
	125	0.146	43.0548
	130	0.162	41.4073
6	100	0.042	51.3420
	105	0.058	49.6570
	110	0.074	47.9870
	115	0.09	46.3170
	120	0.106	44.6470
	125	0.122	42.9770
	130	0.138	41.3070

Frequently, mobile users request different types of video, like popular videos, news, live sports streaming, etc. In the Federation International de Football Association (FIFA), live streaming happens. The same request is made by different mobile clients in various networks around the world, across regions (Higher to a lower network). Herein, the networks specify the mode of operation. Slow mode indicates a low spectrum, which results in delay. It follows that higher mode is faster than the slower mode. Hence the two parameters are well defined. Now the received video is fragmented into chunks. The TS must process the Average Output of Video chunks (AOV_c), and the following formula [36, 46] calculate the Average Video Transcoding time (AVT_T).

$$AVT_T = \left\{ t_q^v \right\} \tag{19}$$

$$AOV_c = \left\{ w_q^v \right\} \tag{20}$$

In the above equations, 19 and 20, the transcoding mode q ranges from 1 to T_M and video type limits from 1 to V_d. The obtained values help find the total time spent for transcoding the video packets based on historical data. In the following section, the AVT_T and AOV_c mapped with each other. Both are directly proportional. The growth of AVT_T, AOV_c makes the transcoder to work in a faster mode. Thus, there is no possibility of a delay. The overall transcoding mode distribution evaluates the maximum delay of Max_D and maximum transcoding video size $MaxV_{sz}$. The following conditions give this:

$$T_d \leq Max_D \tag{21}$$

$$TV_{SZ} \leq MaxV_{SZ} \tag{22}$$

Where T_d represents the transcoding delay of individual T_S, and TV_{SZ} is the transcoder video size of T_S.

By applying the above equation, we propose the IVTS algorithm to choose the optimal transcoding server with a

Table 3 Total delay with two servers

No. of request	VPS Delay (ms)	VTS Delay (ms)	Output Delay (ms)	Total Delay (ms)
100	0.016	0.013	0.09	0.119
200	0.049	0.048	0.54	0.637
300	0.082	0.083	0.99	1.155
400	0.115	0.118	1.44	1.673
500	0.148	0.153	1.89	2.191
600	0.181	0.188	2.34	2.709
700	0.214	0.223	2.79	3.227
800	0.247	0.258	3.24	3.745
900	0.28	0.293	3.69	4.263
1000	0.313	0.328	4.14	4.781

different mode. Then determine the transcoding delay and video segment size of the individual transcoder.

Algorithm 7 Intelligent Video Transcoding Server (IVTS).

Input: q:Transcoding modes,
T:Average response rate,
W:Histoy of Average segment size,
Max_D- Maximum Delay,
MaxV_{sz} - Maximum transcoding video segment size
Output: N – Optimal transcoding server,
E – Estimate probability distribution of T_M,
T_d - Transcoding delay of individual server,
T_S – Transcoding delay of video segment size of individual server

1 Initialize N ← 0 T_d ← -1,
2 TV_{SZ} ← -1
3 n ← no of transcoding request
4 N ← No of transcoding server
5 **foreach** *i from 1 to n* **do**
6 **foreach** *j from 1 to n* **do**
7 **while** $T_d > Max_D > || \, d < 0 \, || \, TV_{SZ} > MaxV_{sz} \, || \, S_Z < 0$ **do**
8 N ← N + 1
9 **while** $T_d > 0$ **do**
10 $\sigma \leftarrow \sum_{q=1}^{Q} P_q \sum_{v=1}^{V} \left(\frac{b_v t_v^q}{N} - \frac{1}{(\mu N)} \right)^2$
11 $\text{T}_d \leftarrow \frac{N^2\lambda^2\sigma^2+\rho^2}{2\lambda N(N-\rho)} + \frac{1}{\mu N}$
12 **end**
13 **end**
14 /* To select transcode mode */
15 **if** $T_d > Max_D \, || \, T_s > MaxV_{sz}$ **then**
16 T_M ← 1 /* fast mode */
17 **else**
18 T_M ← 0 /* slow mode */
19 **end**
20 **if** $T_M == 1$ **then**
21 CT_{qi} ← T_{qi} - λV_{Zi}
22 Min_C_{qi} = V_{SZi} /* CT - Current q^{th} transcoder */
23 **else**
24 /* put the Vsz are waiting in Queue*/
25 **if** $CT_{qi} < Max\_T_{qi}$ **then**
26 $\text{WQ}_i(\text{V}_{SZ})$ CT_{qi} ← V_{SZi}
27 **end**
28 **end**
29 **end**
30 **end**

Description of IVTS The intelligent video transcoder server proposed in this framework helps to choose a transcoder. The algorithm's input is transcoding mode, history of transcoding time with a different mode, and prehistoric average segment size. Apart from this, maximum delay and maximum transcoding video segment size are given as input in our algorithm. IVTS mainly focuses on two processes. The first process continuously receives the number of video requests from 1 to n and video segment size of transcoder checks with maximum video segment size. Based on the condition in step 7, the transcoding server increments. Step 9 through 12 compares the transcoding delay with 0. If it is greater than zero, calculate the transcoding delay of individual server (T_d) and that of video segment size. The second process deals with the selection of the transcoding mode, which consists of two modes, namely fast mode and slow mode. Step 17 compares the maximum delay, V_{SZ} with T_d and T_S. If this criterion becomes true, then the mode is selected as a fast or slow mode. In the fast mode, the current transcoder queue length has computed using step 23. The minimum current transcoder queue is assigned as a video segment size. Similarly, in the slow mode, the input of the video segment size waits in the queue. As per the condition in step 27, the video segment resides in the waiting queue and is assigned to V_{SZ}. Hence, this IVTS algorithm minimizes the transcoding delay by intelligently distributing the workload for individual transcoder. In turn, it reduces the burden for a single server. Thus, it also reduces the transcoding delay in the transcoder server.

3.4 Delay

The framework aims to achieve the QoS parameter of delay in all three components. According to the video request, MMBCE estimates the bandwidth. Then, the below equation calculates the overall delay. The result of VPS, VTS, and Output delay is tabulated as follows:

$$Delay = VPS_{Delay} + VTS_{Delay} + OP_{Delay} \tag{23}$$

Table 4 Total delay with four servers

No. of request	VPS Delay (ms)	VTS Delay (ms)	Output Delay (ms)	Total Delay (ms)
100	0.008	0.0065	0.045	0.0595
200	0.0245	0.024	0.27	0.3185
300	0.041	0.0415	0.495	0.5775
400	0.0575	0.059	0.72	0.8365
500	0.074	0.0765	0.945	1.0955
600	0.0905	0.094	1.17	1.3545
700	0.107	0.1115	1.395	1.6135
800	0.1235	0.129	1.62	1.8725
900	0.14	0.1465	1.845	2.1315
1000	0.1565	0.164	2.07	2.3905

Table 5 Total delay with six servers

No. of request	VPS Delay (ms)	VTS Delay (ms)	Output Delay (ms)	Total Delay (ms)
100	0.004	0.00375	0.0275	0.03525
200	0.01225	0.012	0.185	0.20925
300	0.0205	0.02025	0.3425	0.38325
400	0.02875	0.0285	0.5	0.55725
500	0.037	0.03675	0.6575	0.73125
600	0.04525	0.045	0.815	0.90525
700	0.0535	0.05325	0.9725	1.07925
800	0.06175	0.0615	1.13	1.25325
900	0.07	0.06975	1.2875	1.42725
1000	0.07825	0.078	1.445	1.60125

In Table 3, the no. of servers fixed as 2, then response delay is 0.05 ms with the transcoding queue capacity 25. Tables 3, 4 and 5 demonstrates the delay parameters for video processor, transcoding server and the time taken to deliver the video packets in the mobile client. Therein, the delay affects the delivery of good quality, and buffering happens on the mobile. To overcome this issue, we propose an intelligent transcoder using M/G/N queueing model. The M/M/S video processing server reduces the delay as 0.82 ms, which improves the QoS in the video.

4 Experimental setup and performance metrics

This section provides a detailed description of the experimental setup, implementation of the mobile client, and public cloud in Table 6.

To better evaluate the performance of our proposed video streaming algorithm, we established a realistic testbed to measure the performance with a real video stream and an Android mobile phone. We implemented a video streaming application that can request the video contents from the web server through the HTTP/1.1 protocol. The proposed framework is implemented using Samsung Galaxy mobile phones [53] and AWS cloud server [54]. We used a Samsung Galaxy S7 edge smartphone with Android 6.0 (Marshmallow) OS as the client. The Samsung Galaxy is equipped with a dual-core 1.2 GHz Cortex-A9 CPU and 14GB RAM in Configuration 1. Samsung Galaxy Note20 Ultra 5G mobile phone with Android 10 OS, and Octa-core 2x2.73 GHz Mongoose M5 & 2x2.50 GHz Cortex-A76 & 4x2.0 GHz Cortex-A55 CPU and 12GB RAM in Configuration 2. The minimum RTT between the smartphone and the proxy is about 25.83 ms, and between the proxy and the server is about 1.79 ms, respectively.

Our experimental setup machine was utilized in the research lab setup with the system following system configuration as shown below in Table 6. Twitch TV and YouTube live are the two major real-time live streaming service providers which were used in this work as video datasets. The results are aimed at synchronizing smartphones with the cloud environment by effectively utilizing the estimated bandwidth of mobile, efficient queue-based video processing, and intelligent transcoding, smoothly download the video without any packet loss.

In the cloud server, the mobile profile manager maintains the historical information of mobile which helps MMBCE to estimate the best bandwidth. The proposed queuing theory model QVPS was implemented using AWS. AWS goes beyond a classic hypervisor (i.e., VirtualBox3, Xen4, and VMware5), provides the creation and setup of virtual machines dynamically, which are required for computational resources. This enables the way to handle QoS constraints in traffic peaks. Based on traffic conditions, the arrival rate of the requests that need to be served increases. When traffic decreases, AWS creates new instances automatically and current servers are destroyed and shut down. The structure composed for this experiment was formed with different servers created as virtual machines (VMs) containing the Entering Server (ES), servicing nodes, $VPS_1 \ldots VPS_n$, and a MySQL6 database,

Table 6 Experimental setup

Configuration	Mobile	Public cloud
Processor	Octa-core (4x2.3 GHz Mongoose)	Intel ® Xeon ® CPU E5 2686 v4 2.3 GHz
RAM	4	32 GB
Storage	64 GB	1 TB
Operating System	Android 6.0 (Marshmallow)	Ubuntu 14.04.5 LTS Server
Video Standard	MP4	H.265
Software	MX Player Pro	AWS Device Farm, S3, Amazon CloudFront, Amazon EBS

Database Server(DS), and the Output Server (OS) and client servers. Twitch API provides parallel online video channels in Twitch which helps to enable online video services for transcoding servers. We use the arrival rates on different days as the arrival rates for the different types of videos. After video processing, the IVTS algorithm minimizes the transcoding delay by intelligently distributing the workload for individual transcoder using M/G/N queuing model.

Herein, the cloud server uses public cloud: Amazon Web Service (AWS). The results observed and experimental analysis has been made with Amazon Elastic Cloud Computing (EC2) [55] to demonstrate the working principle in a public cloud. In this cloud environment, the availability zone (AZ) specifies the regions(R). There are 8 regions globally. For our experimental setup, two regions namely, the Asia Pacific and USWest (Northern California) have been considered to demonstrate the live video streaming. The reason for choosing the particular two regions is to examine network traffic between short and far geographic distances.

In the demo shown in appendix 1, in step1, the user can choose the AMI. In step 2, the user can create and launch an instance. In step 3, the user can execute the existing algorithm and our proposed algorithms in this framework. Finally, to test the application, AWS is used to provide the Device farming testing forum. By using this service, the Samsung Galaxy S7 edge has been tested successfully. Finally, we could view our video outputs smoothly with good quality in the mobile screen AWS platform as shown in the output screen. To provide a demo of our system, we have added a set of screenshots in Appendix 1 of this paper.

4.1 Analysis of Industrial application for video streaming

Cloud-based mobile live streaming is executed in many different applications such as enterprise and corporate streaming, health care, social media live streaming, telecommunication public safety, and law enforcement and government. Among these applications, we focus on health care, social media, and telecommunication. For example, the world faces pandemic situations due to coronavirus and people are utilizing the telehealth sevices effectively. Nowadays, doctors in many healthcare organizations are adopting live video streaming services for medical purposes like surgical and diagnostic processes. Doctors are also practicing live streaming to guide and educate new doctors, broadcasting surgical procedures, respond to the patients via live chat. For example, The U.S. Department of Health and Human Services (HHS) practices live video to broadcast workshops, advisory council meetings, and other events to create awareness for covid 19 and improve medical fitness.

The above industrial application for video streaming uses the resolution, frame rate, video bit-rate settings are the basic requirement to stream the video on platforms like YouTube Live, Twitch.tv, Netflix, Beam, and Hitbox. The stream settings are different on each platform. We aim to deliver a good quality of video on different streaming platforms with estimated bandwidth. To achieve this goal, we analyzed the two different configurations of mobile standards with different resolutions and frame rates as shown in Table 7. To evaluate the better performance of our proposed algorithms, we made a comparative study with the existing TCP variants.

We used a Samsung Galaxy S7 edge smartphone with Android 6.0 (Marshmallow) OS as the client. The Samsung Galaxy is equipped with a dual-core 1.2 GHz Cortex-A9 CPU and 14GB RAM in Configuration 1. Samsung Galaxy Note20 Ultra 5G mobile phone with Android 10 OS, and Octa-core 2x2.73 GHz Mongoose M5 & 2x2.50 GHz Cortex-A76 & 4x2.0 GHz Cortex-A55 CPU and 12GB RAM in Configuration 2. To better test, the usefulness and robustness of our methods, two different mobile configurations with the encoded videos are employed, which are shown in Table 7. The frame rate of the two configurations 1 and 2 are set to 24 frames and 30 frames. Hence, the duration of the individual segment is about 1000 ms and 3000 ms for the two configurations, respectively.

Table 7 Different mobile client video streaming configuration

Mobile client	Resolution	Frame rate (FPS)	Video bitrate (Kbps)	Bandwidth (Mbps)
Samsung Galaxy S7 edge	640×360	24	522.96	0.8112
	1280×720	24	2211.84	2.8848
	1440×2560	24	8847.36	11.179
Samsung Galaxy Note20 Ultra 5G	640×360	30	691.2	0.984
	1280×720	30	2764.8	3.576
	1440×2560	30	11059.2	13.944
	1440×3088	30	13436.16	16.789

In our scenario, 200 segments are elected for configuration 1 and 100 segments for configuration 2. In Table 7, we notice how much bandwidth is required for two different mobile configurations with varying resolution and frame rates. The novel idea of our approach is to estimate the bandwidth effectively to minimize packet loss and delay. The requirement of the bandwidth is available in Table 7.

4.1.1 Case 1

In configuration 1, with 24 frame rate and 640×360 resolution, the exact bandwidth which needs to stream the good quality video is shown in Table 7. Hence, the bandwidth is necessary to stream the video. In case, if bandwidth is not available, then packet loss occurs with delay and the quality of the video also becomes low. To avoid the case, the cloud server plays a vital role to estimate the bandwidth of mobile. We implemented the different existing TCP variants in cloud server such as Cubic, TCPW, and TCPHas with the given configuration. Experimental results demonstrate the estimated bandwidth of different TCP variants and our approach. TCP Cubic, TCP Westwood, TCPHas achieves the estimated bandwidth as 0.4, 0.52, and 0.78 with low resolution. The required bandwidth is 0.8112 but it is not estimated efficiently by the existing TCP variants. Hence, in our work, we used the Markov decision process to estimate the bandwidth. The bandwidth of each link will be divided into several regions. In the cloud environment, the actual bandwidth from Table 7 passes to the agent of each mobile (State S). Then, the agent identifies the best bandwidth correspondingly. For each bandwidth, the cloud environment, in turn, responds with a reward to the bandwidth estimation agent. The Markov Decision Process (MDP) develops to provide better bandwidth estimation and the transition state behavior of the same Markov process. The MMBCE achieves 0.8 Mbps. Similarly, with the same procedure, MMBCE obtains 11.002 Mbps than other TCP variants. The analysis made with the high resolution 1440×2560 and with frame rate 24 in the same configuration 1. The required bandwidth is 11.179 to stream the good quality video. We made the comparative study with other TCP variants and obtains 10.452, 9.28, 10.56 for cubic, TCPwestwood and TCPHas. The existing work does not reach the actual bandwidth. In order to solve the issue, MMBCE proposed in this work and it achieves 11.002 as estimated bandwidth respectively.

4.1.2 Case 2

In configuration 2, we added one more high resolution, and the frame rate is fixed as 30. We analyzed with a low-resolution 640×360 and 30 frame rate. The required bandwidth is 0.984 Mbps to stream good quality video but the existing algorithm Cubic, TCPW, and TCPHas as 0.60, 0.75, 0.84. Mbps respectively. To improve the bandwidth, our MMBCE proposed and it achieves 0.9 Mbps. Next, the comparative study is made with high resolution 1440×3088 and with 30 frames. Table 7 shows the required bandwidth as 16.7 Mbps. But 13.125, 13.810, 14.065 Mbps are the bandwidth which is achieved by Cubic, TCPW, and TCPHas. Our proposed algorithm MMBCE attains 15.992 Mbps estimated bandwidth which nearly occupies the actual bandwidth to deliver good quality video.

For both mobile configurations, to minimize the packet loss, the CDCW algorithm in the cloud dynamically adjusts the congestion window as per the estimated bandwidth. The main advantage of CDCW dynamically adjusts the congestion window to improve the goodput with minimal PLR. According to the bandwidth prediction of the cloud server, the video packets arrive in the VPS. The arrival rates of video packets examine with the servers (2, 4, and 6). In general, whenever the arrival video packet arises, the probability of an idle server decreases. Similarly, for the same arrival rate, the VPS increases, which increases the probability of idle servers by an average of approximately 2%. The overall advantage is to reduce the computational overhead and energy consumption. Thus, we significantly minimize video processing delay. After processing the

Table 8 Comparison of bandwidth estimation for different segment size

Segment Size (MB)	MMBCE	TCPHas	TCPW	Cubic
30	0.5	0.4	0.3	0.3
60	0.8	0.6	0.5	0.4
90	1	0.8	0.7	0.6
120	1.3	1.1	0.9	0.8
150	1.4	1.2	1	0.9
180	1.7	1.4	1.2	1.1

Table 9 Comparison of TCP Variants for PDR with time

Time (ms)	Cubic	Reno	TCPW	TCPHas	MMBCE
100	38	40	40	45	50
200	42	45	50	50	55
300	47	50	55	55	60
400	50	55	60	65	70
500	60	65	70	75	80
600	65	70	80	85	90

Table 10 Analysis of Goodput Vs. Packet Loss Rate

PLR	CDCW	TCPHas	TCPW	Cubic
0	4.5	4	3.8	3.5
0.00001	3.8	3.5	3.1	3
0.0001	3.5	3.1	2.8	2.6
0.001	3.2	2.7	2.5	2.2
0.01	2.9	2.3	2.2	1.8
0.1	2.6	1.9	1.9	1.4
0.2	2.3	1.5	1.4	1
0.3	2	1.1	1	0.6
0.4	1.7	1	0.9	0.2
0.5	1.4	0.9	0.7	0.2
0.6	1.1	0.7	0.4	0.2
0.7	0.8	0.6	0.4	0.2

video packets, the cloud server deals with the transcoding server. We propose the IVTS algorithm to choose the optimal transcoding server with a different model. Then determine the transcoding delay and video segment size of the individual transcoder. It always monitors and schedules the performance of transcoders to reduce the workload of the individual transcoder and delay.

4.2 Performance evaluation metrics

The bandwidth estimation, packet delivery ratio, packet loss rate, congestion window, and throughput are the network QoS parameters used to evaluate the proposed algorithms. Tables 8, 9 and 12 show the performance of MMBCE compare with other TCP algorithms. Our CDCW reduces the PLR in Table 10 and improves the congestion window size in Table 11.

Table 11 Comparison of the congestion window among TCP variants

Time (sec)	CDCW	TCPHas	TCPW	Cubic	Reno
0	32	30	34	30	31
50	266	165	98	33	27
100	269	166	103	63	34
150	300	204	112	54	35
200	264	170	123	55	46
250	258	162	122	65	39
300	263	173	123	54	50
350	230	136	121	77	40
400	283	189	128	53	42
450	282	188	130	61	43
500	330	255	130	61	41
550	244	148	128	59	36
600	211	150	122	62	44

Table 12 Performance of throughput for a different time interval

Time (sec)	Cubic	Reno	TCPW	TCPHas	MMBCE
2	67	98	100	118	128
5	562	823	723	823	860
7	613	781	760	815	854
10	749	883	748	848	900
13	798	854	805	906	990
15	815	833	825	908	995
17	814	765	755	804	840
20	814	803	690	790	885
23	816	833	725	782	880
25	826	858	740	799	900
27	830	847	800	830	910
30	816	829	800	809	900
33	818	823	760	791	890
35	800	831	760	813	920
37	789	820	778	815	925
40	789	804	790	792	900

Packet Delivery Ratio (PDR) Packet Delivery Ratio (PDR) defines the sum of accepted video packets to the total no. of video packets sent from the cloud server during different RTT.

Goodput Goodput is defined as the ratio of the total number of video packets received by the mobile in sequence order to the total number of video packets sent by the cloud. The unit of goodput is bytes per second. When any packet loss occurs, retransmission is performed in the goodput parameters based on bandwidth.

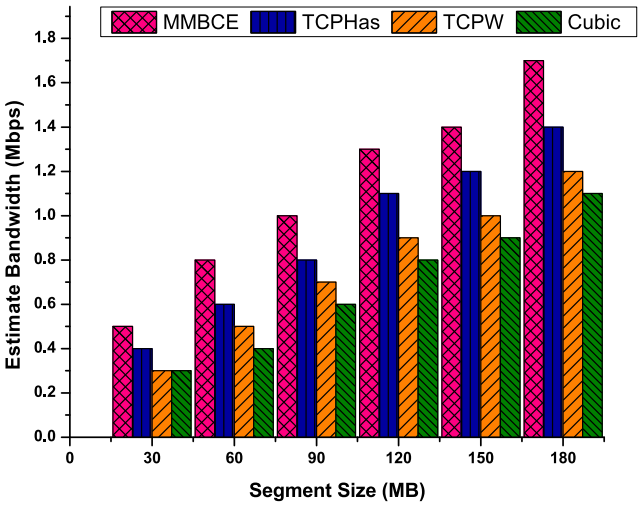


Fig. 2 Comparison of bandwidth estimation over actual bandwidth

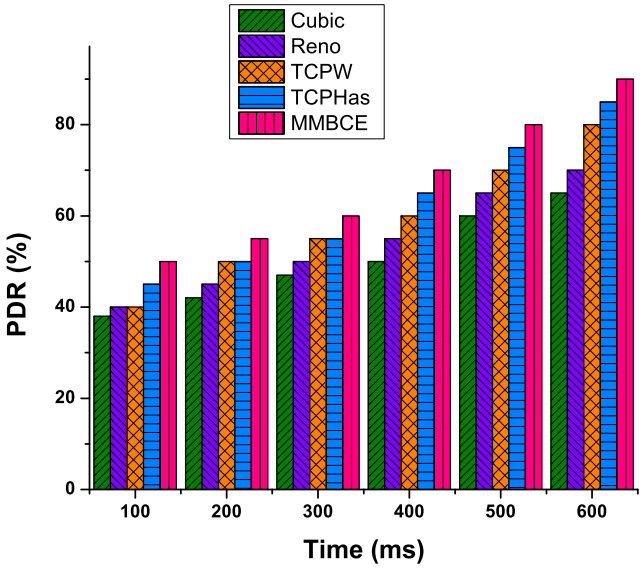


Fig. 3 Packet Delivery Ratio for TCP variants

Packet Loss Rate (PLR) PLR, defines the number of losing video packets per unit of time. In some cases, the video packets are retransmitted from the sender because they are either damaged or dropped.

Throughput The throughput is defined as the transmission rate of video packets from the cloud server to the mobile client at the given time.

$$Throughput = \frac{Packet\ Transfer\ Rate}{Packet\ Transfer\ Time} \tag{24}$$

5 Results and discussion

In this Section, the framework evaluates the performance of estimated bandwidth, Packet Delivery Ratio(PDR), Packet Loss Ratio(PLR), goodput, dynamic congestion window, and throughput for MMBCE. The performance analysis of video processing server utilization and delay carried out in QVPS. Next, we discuss the transcoding parameters such as workload, mode, average output size, and delay.

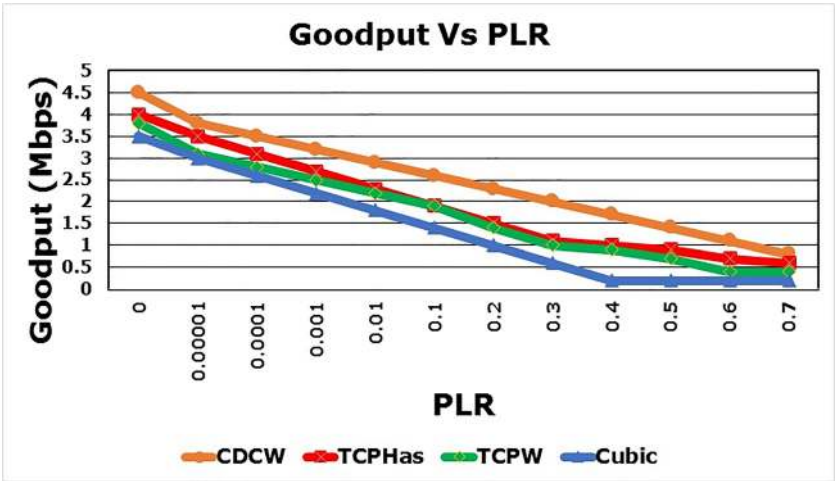
5.1 Mobile bandwidth estimation

Figure 2 depicts the estimated bandwidth of different TCP variants. When the file size and RTT vary, the estimation of actual bandwidth also shows variations in the multi-flow distributed cloud environment. Our approach MMBCE applies the Markovian mathematical model to achieve high bandwidth estimation compared to other TCP algorithms. To make it more transparent, for the file size 30 MB with 50 ms, RTT, the *MMBCE reaches 0.5 Mbps*, and TCPHas utilizes 0.4 Mbps. The slight modification that occurs between the two algorithms is *0.1 Mbps*. Our algorithm provides the best bandwidth estimation using a reward function. The same analysis is also applicable to larger file sizes.

5.2 Packet delivery ratio

PDR is an important parameter to improve network performance. The higher rate of PDR provides a functional network capacity. The unit of PDR metrics measures in terms of percentage. Figure 3 depicts PDR concerning RTT. For example, the lowest RTT 100 ms, the *PDR for MMBCE*, and TCPHAS is *50%* and 48%. The rest of the congestion control schemes TCPW, Reno, and Cubic, provides 40% and 38%. The results show that MMBCE is more efficient and depicts the highest value of PDR. The same analysis performed with the highest value of 600 RTT, MMBCE, provides a high PDR rate since the congestion window dynamically fits in the cloud as per the bandwidth of mobile. Hence, MMBCE improves the performance of PDR compared with other algorithms.

Fig. 4 Performance analysis of goodput and PLR



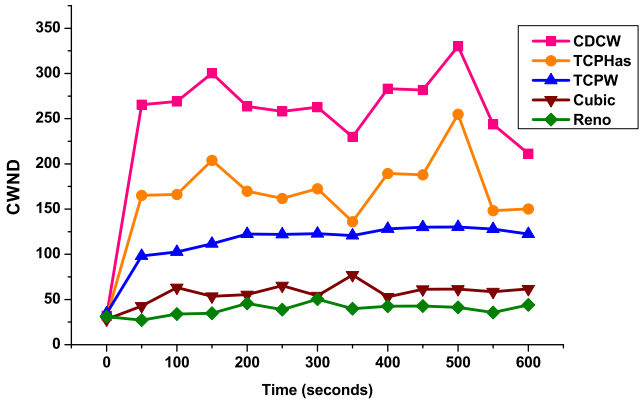


Fig. 5 Congestion Window Size of different TCP variants

5.3 Packet loss rate

Figure 4 experimentally analyzed the four TCP variants of various congestion algorithms, such as Cubic, TCPW, TcpHas, and CDCW. This result shows the better goodput with different Packet Loss Rate (PLR). For example, the lowest 0.00001 PLR, the goodput for the TcpHas, TCPW, and Cubic is 25%, 23%, and 12%, whereas *our approach CDCW achieves 28% improvemen*t. TCPHas, TCPW, and our proposed algorithm comparatively made a slight modification in the goodput. The reason behind this result is, all three algorithms deal with high Bandwidth Delay Product (BDP), and it handles the packet loss issue. A similar analysis examines with the highest PLR as 0.7%. We notice that the proposed algorithm, CDCW, produces 40% more goodput than other algorithms. For a better understanding as PLR increases, the goodput decreases gradually, as both deal with congestion loss and dynamically adjust the congestion window. The congestion window sets the reduction in packet loss. A cubic function fixes the CWND as a low percentage in TCP Cubic.

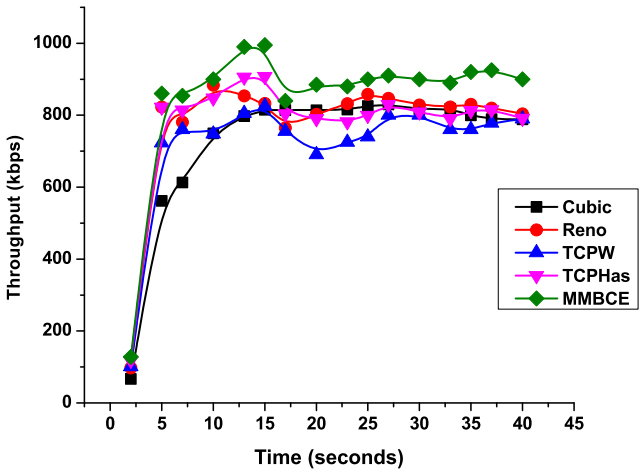


Fig. 6 Comparison of Throughput

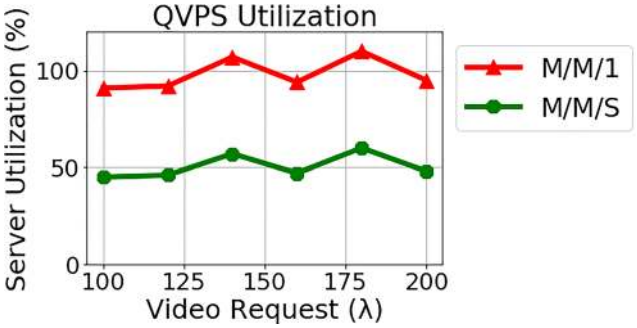


Fig. 7 Queue Utilization of different servers

According to the estimated bandwidth, dynamic adjustment of the congestion window happens in the sender side itself. The main advantage of CDCW dynamically adjusts the congestion window to improve the goodput with minimal PLR.

5.4 Dynamic congestion window

The congestion window is flow control forced through the sender, whereas the receiver inflicts the advertised window flow control. Figure 5 outlines the window size deviation for various algorithms at different RTT. In Fig. 5 CDCW window size steeply improves by adjusting the window dynamically based on segment size and estimated bandwidth. The window of other TCP variants such as TCPW, Cubic, Reno individually increased slowly, but TCPHas has a slight variation compare with proposed CDCW.

5.5 Throughput

Figure 6 shows the throughput performance for our proposed MMBCE compared with other TCP variants. Our approach improves throughput performance than other existing algorithms due to the reduction of PLR. Figure 6

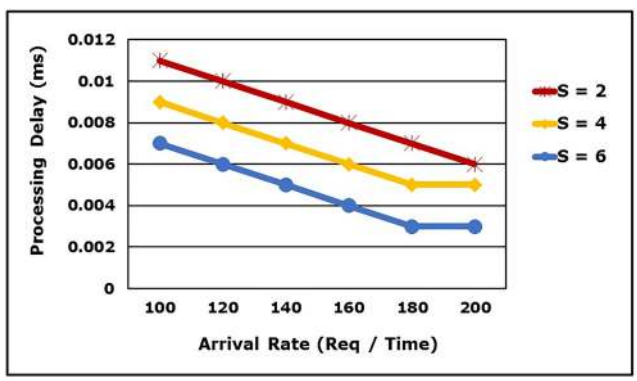


Fig. 8 Video Processing delay of different servers

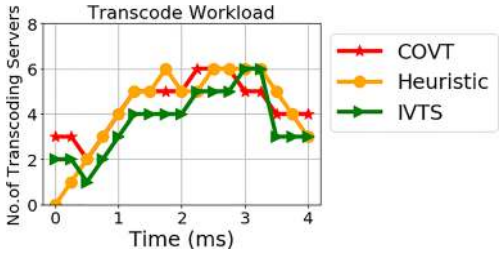


Fig. 9 Comparison of transcoding workload for different Algorithm

observed that the proposed *MMBCE algorithm attempts maximum throughput than other TCP Variants*.

5.6 Video processing server queue utilization

Figure 7 illustrates the evaluation of two different queueing models as M/M/1 and M/M/S for the utilization of video server based on the arrival of request. In the graph, we noticed the arrival rate starts from 100 to 200 with different service rate and utilization of queue are calculated in percentage. The VPS utilizes 91% of the server for the M/M/1 queueing model at 100 arrival rate. Whereas for the same arrival rate, our proposed M/M/S queueing model in *QVPS reduces as 45% of server utilization*. This variation predicts until 200 requests. Hence, whenever the arrival rate increases, the server utilization also gradually increases in M/M/1. Using M/M/S, the server utilization significantly reduced due to the processing server increasing in the proposed model. The next figure shows the processing delay changes with an increase in the arrival rate of many servers.

5.7 Video processing delay

Figure 8 shows the processing delay for the different number of servers. We noticed that the increase in the number of servers (S=6) reduces the video processing delay. The result achieves the goal of the QVPS algorithm. Once video processing over, the next step is transcoding. Here, for the heterogeneous network, experiments were performed for different workloads and modes.

5.8 Transcoding server workload

Figure 9 illustrates the comparison of transcoding servers utilization for different algorithms. To understand transcoding server utilization, we analyzed three approaches, namely Heuristic, COVT, and IVTS. Our algorithm utilizes 29%

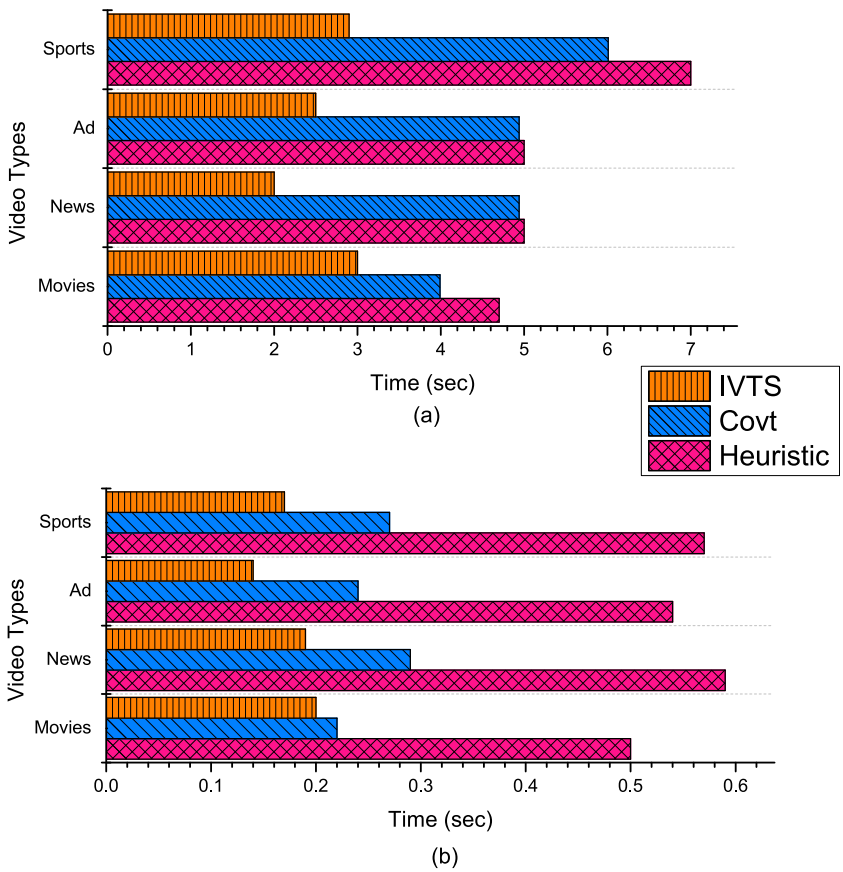


Fig. 10 Comparison of transcoding mode for various methods

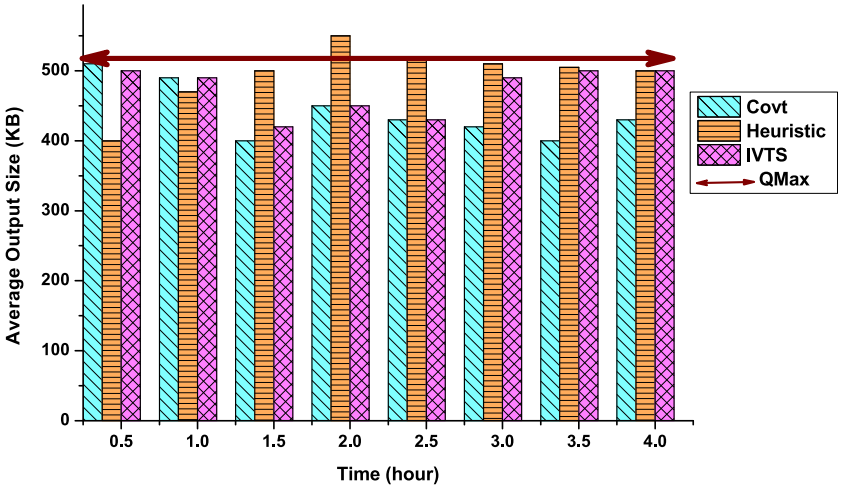


Fig. 11 Comparison of output segment size

transcoder, and COVT employs 35%. Hence, our IVTS method reduces 6% utilization of transcoding workload, and overall, it uses only 13 transcoders. This reduction happens due to the intelligent decision making of IVTS. It always monitors and schedules the performance of transcoders to reduce the workload of the individual transcoder.

5.9 Transcoding mode

Figure 10 investigates with three different video transcoding algorithms, namely IVTS, COVT, and Heuristic. The analysis shows various video data types with varying intervals of time. While transcoding, sports video occupies more transcoding time due to on-demand of different viewers with the heterogeneity of mobiles heavy network traffic, different encoding standards, and billions of user's demands make the delay in slow mode. The time takes for IVTS will always be less in both modes, such as slow and fast. Here, the transcoder operates on the workload instead of applying the heavy load to the particular transcoder. Our algorithm IVTS equally schedules the workload and optimize the time.

5.10 Transcoding average output size and delay

Figure 11 illustrates the comparison of the video output size and average delay for a different algorithm in video streaming. The utilization of video transcoder correlates with IVTS to select the video transcoder based on the output segment size, and thereby, it reduces the transcoding delay. Initially, the maximum segment size fixes as 500 KB, and the delay sets at 2 hours. As per the algorithm 3 and 4, the parameters Max_D and MaxV_{SZ} map with QoS (S_{Max}) and (D_{Max}) of the given figure. The above analysis of Figs. 11 and 12 gives the result of average output size and average delay for different video transcoding algorithms. In this, we noticed, at the initial time of 0.5 hours, COVT executes 510 KB, whereas our proposed IVTS takes 500 KB, which is equal to S_{Max}. The figure clearly states that always IVTS maintains the execution of transcoder output segment size to the QoS(S_{Max}). But, the existing approach Heuristics exceeds the maximum segment size at a specified time interval. Then, it reflects the transcoding delay in the execution process. Since our algorithm maintains a constant level, to minimize the delay and buffering. Next, we move on to the average delay. Once, IVTS identifies the segment size of individual video transcoder, the execution begins.

Herein, the video transcoder is scheduled based on the video output segment size. In turn, it depicts the delay in Fig. 12. The average delay is plotted with a different

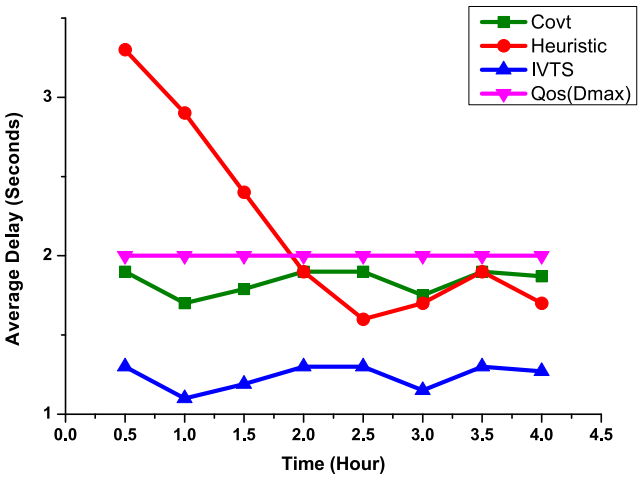


Fig. 12 Flow analysis

time interval, as shown in Fig. 12. Here, the comparison analysis performs with the traditional approaches of COVT, Heuristic, IVTS, and QoS (D_{Max}). The maximum delay for the execution of the different algorithm fixes as an average of 2 seconds. COVT and IVTS take the lower bound of QoS (D_{Max}) of minimum average delay. But heuristic executes with upper bound of D_{Max}. Hence, this figure clearly shows that the *IVTS delay is minimal compared with COVT and Heuristic*.

6 Conclusion

In this paper, we proposed an MMBCE, CDCW, QVPS, and IVTS approach to improve the network and multimedia QoS. The first approach, MMBCE and CDCW estimated the bandwidth with MDP's help to avoid congestion and made a comparative analysis with the existing approach. The second approach, QVPS, involves the M/M/S queue to reduce the delay and server utilization of video processing. Thirdly, IVTS works intelligently for selecting the transcoding server using M/G/N queueing model, which reduces the transcoding time efficiently. The experimental evaluation shows that the overall system delay has reduced by using the above approaches. The queue model in the cloud helps to deliver a good quality of the video. In the future, we intend the cloud server to minimize congestion avoidance in the router for mobile live video streaming.

Appendix 1

This section provides the details on the implementations carried out in this work with suitable screenshots and demos. The proposed framework in cloud server (AWS) Fig. 13 depicts the screen taken from the Samsung Galaxy S7 used in the experiment. Here, the user requests for a video from cloud side. The video is retrieved and delivers good quality of live streaming.

Figure 14 shows integration of Samsung mobile with the public cloud namely AWS. Figure 15 has been taken while testing the proposed algorithms in a public cloud environment with AWS device farm. Figure 16 shows the video delivery using the public cloud AWS farm to the user by applying the proposed algorithms and download the video contents with better response.

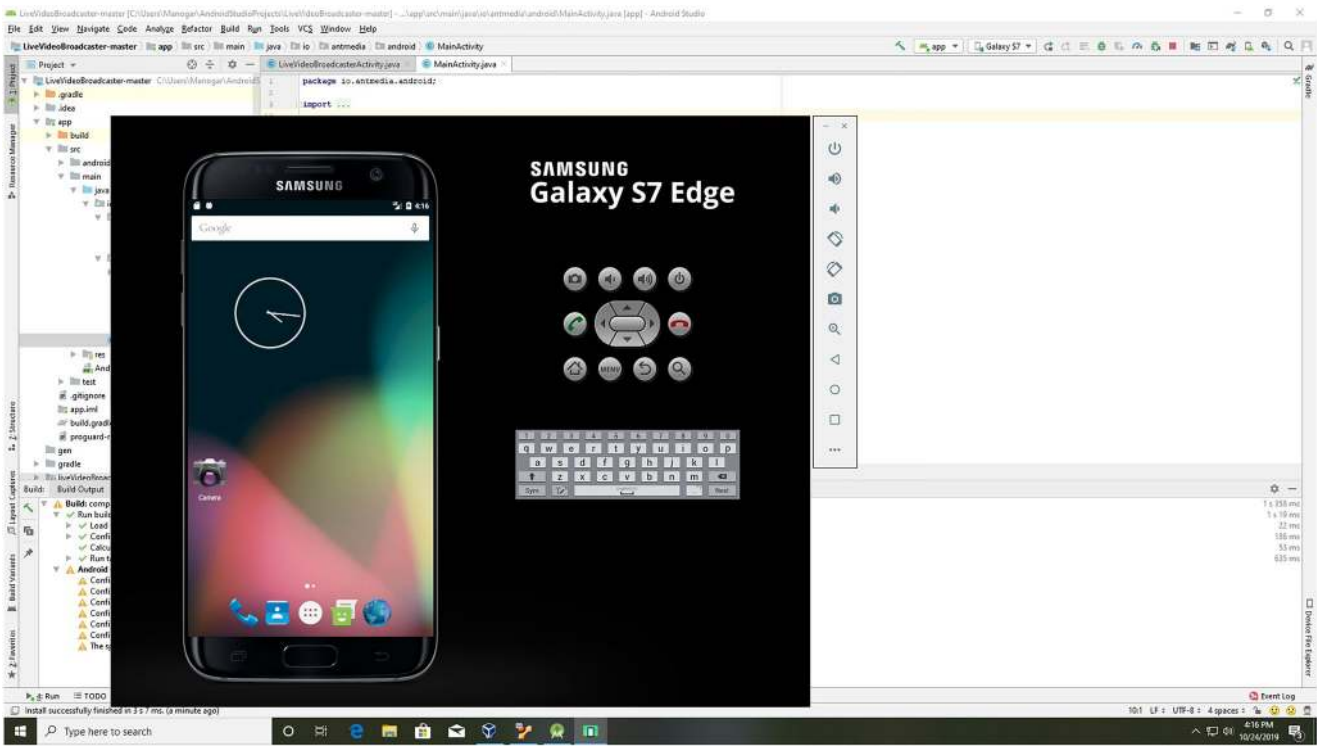


Fig. 13 Samsung Galaxy S7

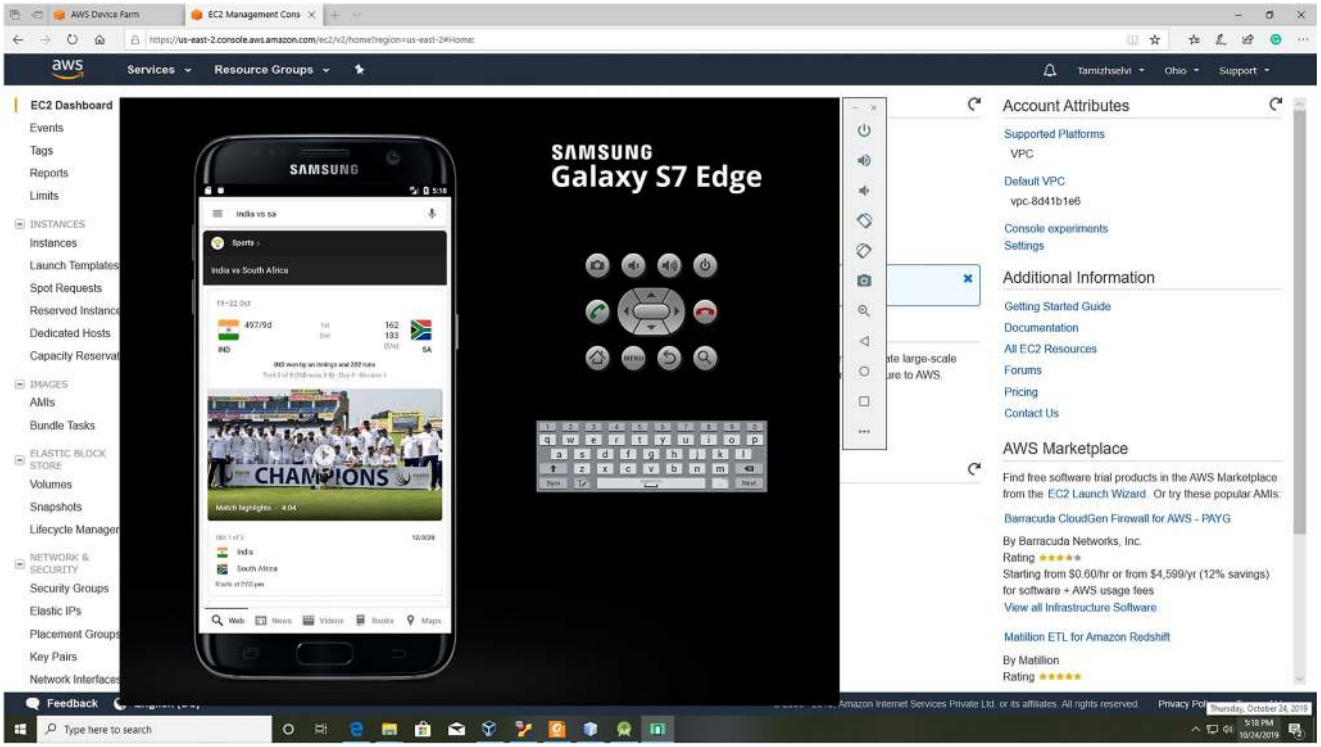


Fig. 14 Public Cloud AWS Device Farm tested output

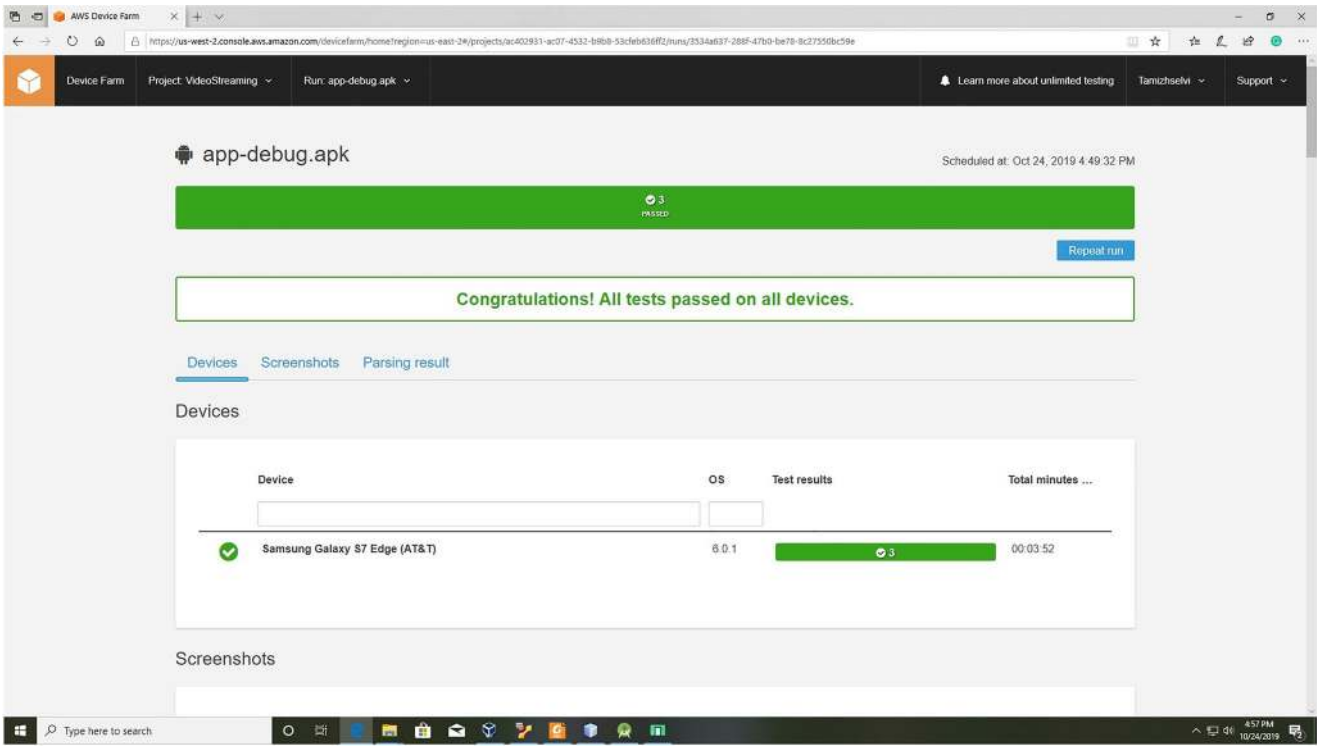


Fig. 15 Public Cloud AWS Device Farm tested

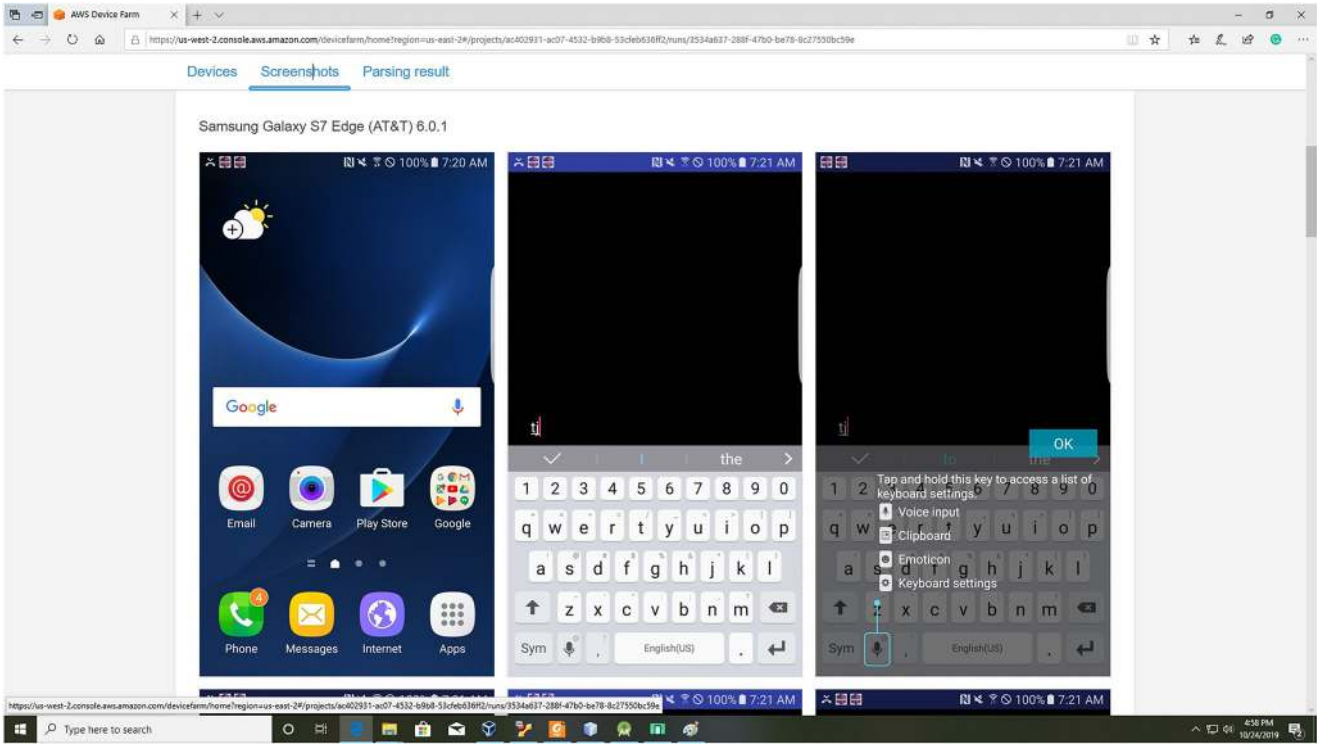


Fig. 16 Public Cloud AWS Device Farm tested output

References

1. Dinh HT, Lee C, Niyato D, Wang P (2013) A survey of mobile cloud computing: architecture, applications, and approaches. Wirel Commun Mobile Comput 13(18):1587–1611
2. Sanaei Z, Abolfazli S, Gani A, Buyya R (2014) Heterogeneity in mobile cloud computing: taxonomy and open challenges. IEEE Commun Surv Tutor 16(1):369–392
3. Stergiou CL, Psannis KE, Gupta B (2020) Iot-based big data secure management in the fog over a 6g wireless network. IEEE Internet Things J
4. Hossain K, Rahman M, Roy S (2019) Iot data compression and optimization techniques in cloud storage: current prospects and future directions. Int J Cloud Appl Comput (IJCAC) 9(2):43–59
5. Al-Qerem A, Alauthman M, Almomani A, Gupta B (2020) Iot transaction processing through cooperative concurrency control on fog–cloud computing environment. Soft Comput 24(8):5695–5711
6. Tamizhselvi S, Muthuswamy V (2014) Adaptive video streaming in mobile cloud computing. In: 2014 IEEE International conference on computational intelligence and computing research. IEEE, pp 1–4
7. Jadad HA, Touzene A, Day K, Alziedi N, Arafeh B (2019) Context-aware prediction model for offloading mobile application tasks to mobile cloud environments. Int J Cloud Appl Comput (IJCAC) 9(3):58–74
8. Ahmed E, Naveed A, Gani A, Ab Hamid SH, Imran M, Guizani M (2019) Process state synchronization-based application execution management for mobile edge/cloud computing. Futur Gener Comput Syst 91:579–589
9. Noor TH, Zeadally S, Alfazi A, Sheng QZ (2018) Mobile cloud computing: Challenges and future research directions. J Netw Comput Appl 115:70–85
10. Abdo JB, Demerjian J (2017) Evaluation of mobile cloud architectures. Pervasive Mob Comput 39:284–303
11. Esposito C, Ficco M, Gupta B (2021) Blockchain-based authentication and authorization for smart city applications. Inf Process Manag 58(2):102468
12. Li D, Deng L, Gupta B, Wang H, Choi C (2019) A novel cnn based security guaranteed image watermarking generation scenario for smart city applications. Inform Sci 479:432–447
13. Lai CF, Wang H, Chao HC, Nan G (2013) A network and device aware qos approach for cloud-based mobile streaming. IEEE Trans Multimed 15(4):747–757
14. Tamizhselvi S, Muthuswamy V (2015) A bayesian gaussian approach for video streaming in mobile cloud computing. Adv Nat Appl Sci 9(6 SE):470–478
15. Wang Z, Zeng X, Liu X, Xu M, Wen Y, Chen L (2016) Tcp congestion control algorithm for heterogeneous internet. J Netw Comput Appl 68:56–64
16. Lukaseder T, Bradatsch L, Erb B, Van Der Heijden RW, Kargl F (2016) A comparison of tcp congestion control algorithms in 10g networks. In: 2016 IEEE 41st conference on local computer networks (LCN). pp. IEEE, 706–714
17. Wang J, Wen J, Zhang J, Xiong Z, Han Y (2016) Tcp-fit: An improved tcp algorithm for heterogeneous networks. J Netw Comput Appl 71:167–180
18. Jiang X, Jin G (2015) Cltcp: an adaptive tcp congestion control algorithm based on congestion level. IEEE Commun Lett 19(8):1307–1310

19. Parichehreh A, Alfredsson S, Brunstrom A (2018) Measurement analysis of tcp congestion control algorithms in lte uplink. In: 2018 Network Traffic Measurement and Analysis Conference (TMA). IEEE, pp 1–8
20. Callegari C, Giordano S, Pagano M, Pepe T (2014) A survey of congestion control mechanisms in linux tcp. In: Distributed computer and communication networks. Springer, pp 28–42
21. Leong WK, Wang Z, Leong B (2017) Tcp congestion control beyond bandwidth-delay product for mobile cellular networks. In: Proceedings of the 13th international conference on emerging networking experiments and technologies. ACM, pp 167–179
22. Jacobson V (1990) Modified tcp congestion control and avoidance alogrithms. Technical Report 30
23. Xu L, Harfoush K, Rhee I (2004) Binary increase congestion control (bic) for fast long-distance networks. In: Ieee Infocom, vol 4. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), pp 2514–2524
24. Ha S, Rhee I, Xu L (2008) Cubic: a new tcp-friendly high-speed tcp variant. ACM SIGOPS Operat Syst Rev 42(5):64–74
25. Mascolo S, Casetti C, Gerla M, Sanadidi MY, Wang R (2001) Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In: Proceedings of the 7th annual international conference on Mobile computing and networking. ACM, pp 287–297
26. Metzger F, Liotou E, Moldovan C, Hoßfeld T. (2016) Tcp video streaming and mobile networks: not a love story, but better with context. Comput Netw 109:246–256
27. Jacobson V (1995) Congestion avoidance and control. ACM SIGCOMM Comput Commun Rev 25(1):157–187
28. Fall K, Floyd S (1996) Simulation-based comparisons of tahoe, reno and sack tcp. ACM SIGCOMM Comput Commun Rev 26(3):5–21
29. Alrshah MA, Othman M, Ali B, Hanapi ZM (2014) Comparative study of high-speed linux tcp variants over high-bdp networks. J Netw Comput Appl 43:66–75
30. Lin S, Zhang X, Yu Q, Qi H, Ma S (2013) Parallelizing video transcoding with load balancing on cloud computing. In: 2013 IEEE International symposium on circuits and systems (ISCAS2013). IEEE, pp 2864–2867
31. Wang Z, Sun L, Wu C, Zhu W, Zhuang Q, Yang S (2015) A joint online transcoding and delivery approach for dynamic adaptive streaming. IEEE Trans Multimed 17(6):867–879
32. Liu S, Baṣar T., Srikant R (2008) Tcp-illinois: A loss-and delay-based congestion control algorithm for high-speed networks. Perform Eval 65(6-7):417–440
33. Mudassar A, Asri NM, Usman A, Amjad K, Ghafir I, Arioua M (2018) A new linux based tcp congestion control mechanism for long distance high bandwidth sustainable smart cities. Sustain Cities Soc 37:164–177
34. Chen Z, Liu Y, Duan Y, Liu H, Li G, Chen Y, Sun J, Zhang X (2015) A novel bandwidth estimation algorithm of tcp westwood in typical lte scenarios. In: 2015 IEEE/CIC International conference on communications in China (ICCC). IEEE, pp 1–5
35. Ameur CB, Mory E, Cousin B, Dedu E (2017) Tcphas: Tcp for http adaptive streaming. In: 2017 IEEE International conference on communications (ICC). IEEE, pp 1–7
36. Taha HA (2011) Operations research: an introduction, vol 790. Pearson/Prentice Hall, Upper Saddle River
37. Kleinrock L (1976) Queueing systems, volume 2: Computer applications, vol 66. Wiley, New York
38. Vilaplana J, Solsona F, Teixidó I., Mateo J, Abella F, Rius J (2014) A queuing theory model for cloud computing. J Supercomput 69(1):492–507
39. Nan X, He Y, Guan L (2014) Queueing model based resource optimization for multimedia cloud. J Vis Commun Image Represent 25(5):928–942
40. Baik E, Pande A, Zheng Z, Mohapatra P (2016) Vsync: Cloud based video streaming service for mobile devices. In: IEEE INFOCOM 2016-The 35th annual ieee international conference on computer communications. IEEE, pp 1–9
41. Cheng R, Wu W, Lou Y, Chen Y (2014) A cloud-based transcoding framework for real-time mobile video conferencing system. In: 2014 2nd IEEE international conference on mobile cloud computing, services, and engineering. IEEE, pp 236–245
42. Zheng Y, Wu D, Ke Y, Yang C, Chen M, Zhang G (2017) Online cloud transcoding and distribution for crowdsourced live game video streaming. IEEE Trans Circ Syst Vid Technol 27(8):1777–1789
43. Gao G, Zhang W, Wen Y, Wang Z, Zhu W (2015) Towards cost-efficient video transcoding in media cloud: Insights learned from user viewing patterns. IEEE Trans Multimed 17(8):1286–1296
44. Aparicio-Pardo R, Pires K, Blanc A, Simon G (2015) Transcoding live adaptive video streams at a massive scale in the cloud. In: Proceedings of the 6th ACM multimedia systems conference. ACM, pp. 49–60
45. Oikonomou P, Koziri MG, Tziritas N, Loukopoulos T, Cheng-Zhong X (2019) Scheduling heuristics for live video transcoding on cloud edges. ZTE Commun 15(2):35–41
46. Wei L, Cai J, Foh CH, He B (2017) Qos-aware resource allocation for video transcoding in clouds. IEEE Trans Circ Syst Vid Technol 27(1):49–61
47. Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT Press, Cambridge
48. Xing M, Xiang S, Cai L (2014) A real-time adaptive algorithm for video streaming over multiple wireless access networks. IEEE J Sel Areas Commun 32(4):795–805
49. Wang HS, Moayeri N (1995) Finite-state markov channel-a useful model for radio communication channels. IEEE Trans Veh Technol 44(1):163–171
50. Jackson JR (1963) Jobshop-like queueing systems. Manage Sci 10(1):131–142
51. Nair AN, Jacob M, Krishnamoorthy A (2015) The multi server m/m/(s, s) queueing inventory system. Ann Oper Res 233(1):321–333
52. Kleinrock L (1975) Theory, volume 1, Queueing systems. Wiley-interscience, Hoboken
53. Samsung galaxy s7 edge android emulator. https://medium.com/@ssaurel/install-samsung-galaxy-s7-and-s7-edge-skins-in-your-android-emulator-846bbcd84d24
54. Amazon web services, inc.,amazon elastic compute cloud. http://aws.amazon.com/ec2/
55. Palmer M (2017) Amazon EC2 instances explained. [online] Available: https://24x7itconnection.com/2017/01/10/amazon-ec2-instances-explained

S.P. Tamizhselvi is currently working as Assistant Professor (Sr.Gr) in Department of Computer Science and Engineering, KCG College of Technology, Chennai. She received her M.E. and B.E in the Department of Computer Science and Engineering from Anna University. Her area of interest includes Mobile Cloud Computing, Computer Network and Multimedia.

Vijayalakshmi Muthuswamy is currently working as Associate Professor in Anna University, Chennai. She received her M.E. and Ph.D. from Anna University and B.E. degree from NIT Trichy. She has published 50 articles in journals and conferences. Her area of interest includes Mobile Database, Computer Networks, Mobile Cloud, Cloud Computing and Security.