Trust and Reputation Multiagent-driven Model for Distributed Transcoding on Fog-Edge

Charles A. N. Costa
Computer Science Dept.
University of Brasilia
Brasília, Brazil
charles.costa@aluno.unb.br

Bruno Macchiavello
Computer Science Dept.
University of Brasilia
Brasília, Brazil
macchiavello@unb.br

Célia Ghedini Ralha
Computer Science Dept.
University of Brasilia
Brasília, Brazil
ghedini@unb.br

Abstract

Adaptive Bitrate Streaming is a popular technique for providing video media over the Internet. Nevertheless, the computational cost of transcoding a video in many formats can limit its application on live video streaming. Besides, the network overhead of transmitting simultaneously many versions of the same content is a problem. Offloading the transcoding job to the network edge can deal with the problem. Users and providers of live video could benefit from a joint scheme that allowed edge devices to do the transcoding with tolerable latency and delay. This work presents a multiagent-driven model to deal with the problem of distributed transcoding on fog-edge computing. Agents have well-defined roles relating to Broker, Transcoder, and Viewer Proxy. Trust and Reputation metrics derived from utility functions that take into account users' quality of experience (QoE) are defined and applied. The **Re**putation-based **No**de **S**election (ReNoS) algorithm is presented for selecting the best nodes to perform the transcoding tasks. The conducted experiments indicate that the proposed approach can afford utility gain keeping viewers' QoE having the potential to be applied in real edge computing environments.

1 Introduction

Adaptive Bitrate (ABR) streaming is a convenient way to distribute video to many users over the Internet. ABR splits the video into segments and then codes it in different bitrates so viewers' players can switch from a version to another to adapt to slow or unstable bandwidth conditions. The appeal of ABR is that experiences in subjective video quality evaluation suggest that quality of experience (QoE) is better as higher is the bitrate, but it is very penalized for interruptions. The transport protocol is HTTP, which facilitates its implementation and adoption ([Bin15]). HLS/MPEG-DASH is an industry pattern that relies on ABR ([Sto11]). Today, most providers have adopted ABR due to its advantages, but the delay it introduces in live video broadcast is still an open problem ([DKZ15]).

In: R. Falcone, J. Zhang, and D. Wang (eds.): Proceedings of the 22nd International Workshop on Trust in Agent Societies, London, UK on May 3-7, 2021, published at http://ceur-ws.org

The number of computational resources demanded transcoding is not negligible, representing significant costs for live video providers. Although pre-stored video transcoding could be postulated, for popular live video providers transcode every live channel is prohibitive. For example, [PS14] describes two strategies used by Twitch to select what channels will be transcoded to deal with the trade-off between QoE and computational resource consumption. Other works have tried to offload the transcoding task to users' devices as a way to alleviate the costs from providers and improve at the same time viewers' QoE ([HZML17], [BBE+19], [FKZY20]). In a Cloud-based architecture, the layer where end-user devices lay on is called the Edge or Fog. The Fog has indeed plenty of unused computational resources theoretically available at a small network latency, which can be good for live video transcoding. However, since these devices are neither in control of providers nor viewers, there is a risk of losing utility instead of gaining. Thus, in such an environment, open and semi-competitive, collaboration might be improved if nodes are guided to form overlay networks based on trust and reputation (T&R) ([HZJ08]).

Approaching the problem of coordinating the distributed transcoding tasks involving fog nodes and edge devices, the contributions of this work are the following:

- define a T&R multiagent model for dealing with the problem of distributed transcoding of live-video events over fog-edge computing.
- present the **Re**putation-based **No**de **S**election (ReNoS) algorithm for selecting the best nodes to perform the transcoding tasks.

The rest of the paper includes in Section 2 the main concepts; in Section 3 related work to solve the problem of distributed video transcoding; in Section 4 our approach to deal with the problem; in Section 5 the experiments with results; finally, in Section 6 the conclusions.

2 Background

Adaptive Bitrate Streaming

The central idea of ABR is transcoding the same input into segments of different bitrates. A typical workflow to deliver a live video event using ABR includes five steps ([DKZ15]): i) the video content is obtained and pushed up to a server; ii) the video stream is decoded and then re-coded to a convenient format for transmission; iii) the video stream is split into segments and they published on HTTP servers along with manifest files; iv) Content Delivery Networks (CDNs) are regionally employed to minimize latency; v) viewer's player performs buffering before decode and play the stream.

The buffering technique employed by viewer's players is the key characteristic in determining the predictability of video segment requests. There exist many algorithms that different players should apply. Following the work in [KCTV17], those algorithms can be classified into three categories: Throughput-based algorithms, Buffer-based adaptation, and Time-based adaptation. Throughput-based algorithms take constant measures of network bandwidth to evaluate TCP throughput and then schedule video chunk requests based on the current state of a buffer. The Buffer-based adaptation class observes the buffer occupancy itself to determine the time and bitrate version from at the video chunks should be requested. Finally, in Time-based adaptation downloading time is considered rather than TCP throughput, and then uses a pre-computed buffer-map to select the appropriate video representation. As commented, the audience of online videos do prefer high bitrate versions, but interruptions harm QoE. Considering this fact, the joint utility of an online video session can be defined as the combination of the average bitrate counterbalanced by the playback smoothness. Playback smoothness is a ratio between time spent rebuffering and the total time of video exhibition ([SUS16]).

Fog-Edge Computing

Fog-edge computing (FEC) is a complement of cloud computing that employs devices on the edge of the network, so improving the quality of service towards a service continuum ([BS19]). Hierarchically, the FEC can be divided into three layers: inner-edge, middle-edge, and outer-edge. The inner-edge layer corresponds to networks where the covered area is as large as a country or a state. In the inner-edge are placed the infrastructure for geo-distributed cache and the processing centers of WANs, as the CDN mentioned in [DKZ15]. The objective of the inner-edge layer is to improve QoE lowering the network latency. The middle-edge corresponds to the environment where MANs, LANs, Wireless LANs, and the cellular network are placed. Accordingly, with [BS19], the middle-edge is the common understanding of the fog computing layer. The outer-edge, which is also known

as the far-edge or things layer, is where we can find user's devices like mobile devices, as well as small devices like sensors and actuators. Figure 1 illustrates the FEC layer architecture.

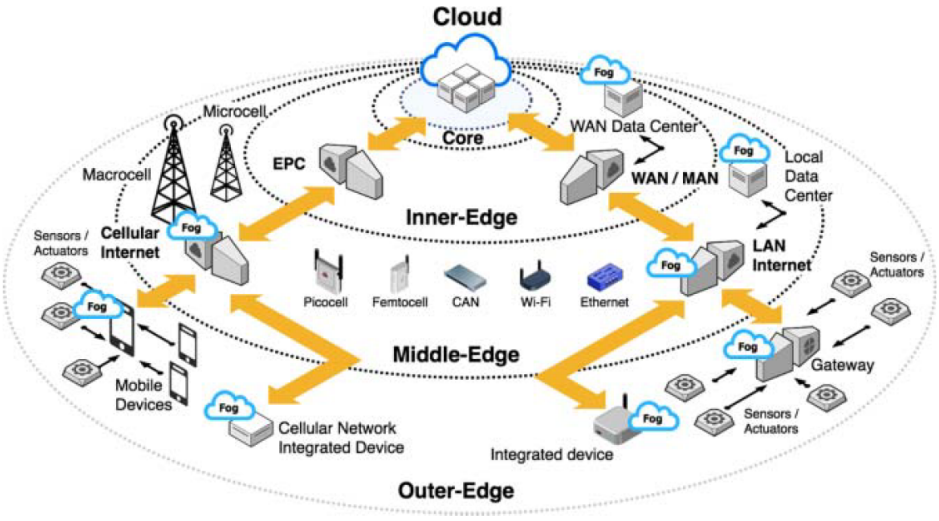

Figure 1: The FEC layer architecture [BS19].

As a complement of Internet infrastructure, the FEC inherits some of its characteristics ([BS19]). Since some FEC nodes are capable of making decisions to improve their performance, they can be described as agents. Adhering to the classification presented in [RN10], the FEC properties can be described as: (i) multiagent - it is expected that agents in the FEC could cooperate to improve overall performance; (ii) non-deterministic - agents would not be able to determine the next state only evaluating the effects of their actions; (iii) dynamic - assuming that nodes can join and leave the network unexpectedly; (iv) continuous - some environmental measures can be discretized but not all of them, and not for all the purposes; (v) sequential - as long as decisions made in the present will influence performance in the future; (vi) partially observable - information about the environment and other agents would remain outdated most of the time.

Trust and Reputation

To face the uncertainty involved in collaboration throughout FEC, a rational agent A, before delegating a task, which its welfare depends on, to an agent B, must somehow compute the probability of B completing the task successfully. This probability means how much agent A trusts B relative to the completion of the task. Nevertheless, an agent could not evaluate the trustworthiness of a counterpart if they have never met. A recurrent solution is to ask others about the opinion they have of the possible partner concerning the coted task. A weighting of the trust that a node receives from a collective is called its reputation. These definitions of T&R are in line with the ideas presented in [CF98]. Bringing up our problem domain, an agent that cannot complete the delegated tasks frequently, does not contribute to improving the delegators' QoE, should receive a low trust value. Since viewers of live-video streaming could obtain the stream directly from the stream provider at the cloud, there is no sense in continuing to select a bad transcoder in the FEC. In this scenario, the decision-making process proposed in [Mar99] could be adopted. Thus, nodes unable to maintain themselves above a trust threshold should be removed from the pool of possible partners.

There are many models of interaction based on T&R, like REGRET ([SS01]), March ([Mar99]), and FIRE ([HJS06]). Authors in [HR14] compared T&R models and proposed a meta-model that allows reasoning about the parameters of the models at run time. Considering the heterogeneity of the devices in the FEC, reasoning at the meta-level can be applied to adjust requirements for limited resources like storage, network, and energy. The FIRE T&R model presented in [HJS06] can use several sources of information to calculate the trustworthiness and reputation of a party. It can use four components for trust calculation: Interaction Trust (IR) is the result of direct interaction, which is also known as a direct trust; Witness Reputation (WR) is a weighting of trust evaluations provided by third parties; Role-based Trust (RT) is defined by role-based relationships between agents, e.g., agents owned by the same proprietary; Certified Reputation (CR) is evaluations made by

an authority and provided by target agents as proof of past good performance.

In FIRE T&R model, trust evaluations are calculated by Equation 1, where: k is one of the components for trust calculation; $T_k(a,b,c)$ is the trust that agent a has of agent b in relation to subject c; $R_k(a,b,c)$ is the set of all ratings collected; $\omega_k(r_i)$ is the relative weight of rating r_i and v_i is the value of the rating.

$$T_k(a,b,c) = \frac{\sum_{r_i \in R_k(a,b,c)} \omega_k(r_i) * v_i}{\sum_{r_i \in R_k(a,b,c)} \omega_k(r_i)} \tag{1}$$

An overall trust evaluation is calculated using Equation 2, which is a combination of all components of the FIRE T&R model. In that, k varies in the set of all existing components, e.g., $\{IR, WR, RT, CR\}$.

$$T(a,b,c) = \frac{\sum_k \omega_k * T_k(a,b,c)}{\sum_k \omega_k} \tag{2}$$

A different weighting function is defined for every information source. The weighting function for IR is defined in Equation 3 and represents the degradation of the rating reliability over time.

$$\omega_{IR}(r_i) = e^{-\frac{\Delta t}{\lambda}} \tag{3}$$

To calculate the weighting function of the WR component, the witness' credibility should be accounted for. In [HJS05] was proposed to compare witness' ratings with those directly obtained, then the deltas were applied to the FIRE T&R model to derive a credibility factor. Assuming that an agent a wants to evaluate the credibility of a witness w and IR ratings are available, the Equation 4 is applied, where v_k is the value by the witness and v_a by direct interaction. If the difference between the values is greater than an inaccuracy threshold i, the witness is penalized with the lowest possible value (-1).

$$v_w = \begin{cases} 1 - |v_k - v_a|, & \text{if } |v_k - v_a| < i \\ -1, & \text{if } |v_k - v_a| \geq i \end{cases} \tag{4}$$

The second step is to apply Equation 1 as if calculating the IR component. If direct interaction values were not available, a default credibility rate is assigned to every witness. Equation 5 demonstrates this reasoning, where T_{DWC} is the default credibility value.

$$T_{wc}(a,w) = \begin{cases} T_{IR}(a,w,c_{WC}), & \text{if } R_{IR}(a,w,c_{WC}) \neq 0 \\ T_{DWC}, & \text{otherwise} \end{cases} \tag{5}$$

Finally, Equation 6 is the weighting function for the WR component. It takes weights calculated with Equation 3, since rating reliability decays with time, but it is multiplied by the credibility factor. If T_{wc} for a witness is less than zero, it means that the witness is not trustful at all, then its report should not be taken into account.

$$\omega_{WR}(r_i) = \begin{cases} 0, & \text{if } T_{wc}(a,w) \leq 0 \\ T_{wc}(a,w) * \omega_{IR}(r_i), & \text{otherwise} \end{cases} \tag{6}$$

Considering a population where self-interested agents exist, FEC delegator nodes are vulnerable to some trust and reputation attacks, as fake feedback and unfair rating ([LS10]). Dishonest delegate nodes might plot with each other to artificially increase its evaluation and decrease the others. As advocated in [HJS05], evaluating witness credibility can mitigate the harmful effects of those practices, or, at least, it would take more time until a delegator node was tricked to give trust to a malicious partner.

3 Related work

In this section related work is presented. Since every work has a different approach employing a specific method, it is difficult to compare one by one in terms of performance. Table 1 summarizes related work's important aspects.

In [HZML17] is investigated an edge-fog clouding distributed transcoding scheme. The goal of the work is to minimize the delay experienced by the viewers. Candidate transcoding nodes are organized in a pool based on their stability, i.e., as likely they are to stay online until the end of the streaming. The pool is implemented

in a B+tree data structure which can guarantee that inclusion and exclusion in the pool occur in $O(logN)$ and selecting the most preferred node for transcoding can take $O(1)$. Some other heuristics for node selection are suggested, like video quality, but not further explored in the manuscript. In terms of architecture, regional data centers have the responsibility of distributing the transcoding task to candidate viewers. Thus, management is executed in the middle-edge while transcoding is performed by outer-edge devices.

The objective of joining nodes on the fog in a distributed transcoding system for live video streaming is modeled by [LDL19] as a Non-convex Integer Programming problem. The model is optimized through the expected QoE that viewers could obtain watching the stream from transcoder nodes. The success rate of a device transcoding is considered in the cost model. Anyway, the success rate is chosen to reflect the stability of the node online. Authors use Complementary Geometric Programming to provide a sub-optimal solution. Although no structure to where to place nodes in the Fog is provided, transcoding nodes are placed between regional data centers and CDNs, which locate them in the inner-edge.

Considering mobile network architecture, [BBE+19] delegates the transcoding jobs to Mobile Edge Computing (MEC) in Antenna Integrated Radios (AIR). MEC-AIR nodes are placed in the middle-edge. Only the highest bitrate available is requested to CDN servers and then transcoding to lower bitrates is performed by MECs. It is intended to minimize backhaul utilization and the number of CDN hits per stream. The most suitable bitrate is chosen by applying a greedy algorithm, which resulted from the relaxation of an Integer Linear Programming solution to the modeled problem.

The objective of [FKZY20] is to cope with the problem of distributed transcoding in vehicular fog computing. Both roadside units and vehicles are considered FEC nodes so that roadside units are placed in the middle-edge and vehicles in the outer-edge. The problem of selecting bitrate versions and transcoding nodes is modeled as a Markov Decision Problem and then an algorithm that employs actor-critic and deep-reinforcement learning is proposed.

In our approach, transcoding jobs will be dynamically assigned to nodes in every turn according to their past-observed performance. Hence, T&R metrics can represent an overall utility contribution including not only the node stability but also their competence for doing the assigned task. Besides, we propose that transcoding should be done in the outer-edge, nearby viewers' locations, so transcoded segments would be served with relatively low latency. Thus, the problem is addressed on two fronts: defining an architecture in terms of a multi-agent system; and reasoning about selecting appropriate nodes for transcoding jobs in an open environment.

Table 1: Related work comparison.

Work	**Transcoding in the outer-edge**	**Selecting by performance**	**Method**
[FKZY20]	✓	✓	actor-critic & deep reinforcement learning
[BBE+19]			integer linear programming & optimization
[LDL19]		✓	complementary geometric programming
[HZML17]	✓		B+Tree sorting from stability metric
This Work	✓	✓	T&R & multiagent models

4 Multiagent Model

The multiagent model comprises three well-defined agent roles: Viewer Proxy, Transcoder, and Broker. The Viewer Proxy is the agent for the audience of the live video stream, the ones that ask brokers for the adapted ABR stream. The Transcoder is interested in receiving transcoding jobs and being rewarded for them. The responsibility of the Broker is to manage the association of Viewer Proxies and Transcoders for the benefit of both. Concerning the network architecture, viewers and transcoders are expected to reside in the outer edge. However, it is important to point that nodes must be geographically close to each other to keep latency inside an acceptable range.

The model will act as an intermediary between viewers and the platform streaming. As an intermediary, the model will provide the viewers with the live-video content they want to consume with an advantage: It will manage to ensure that viewers could get the better QoE possible, providing all the bitrate versions needed. In exchange, is expected that viewers provide feedback on the quality obtained from each video segment. On the other side of the relations, the streaming platform is now depending on an external agent to maintain the viewers connected. The streaming platform must rely on our model's ability to coordinate transcoding jobs so

as the viewers could obtain a good QoE. The streaming platform could evaluate our model performance just by observing the viewers' tendency of staying connected or not.

Since the viewer, a human agent, would not be able to respond within the required time, we introduced the Viewer Proxy, a software agent represented by a role. The Viewer Proxy's responsibility is to interact with the other software agents using a protocol of messages, in the interest of the Viewer. Those agents who will perform the Transcoder role are interested in receiving the transcoding jobs. As a Viewer Proxy role, the Transcoder role is also suited to be performed by software agents. The Brokers are to select the better nodes to perform the transcoding tasks, offer them the transcoding jobs, and then inform Viewer Proxies where to find the transcoded video segments. Within a specific geographic region, a Broker should be able to interact with many different Transcoders and Viewer Proxies, coordinating the distributed transcoding of more than one live-video streaming at the same time.

To give an idea of where, in the FEC layered architecture described in Section 2, the agent roles should be placed, the Figure 2 is presented. Since different roles can be performed by the same node, the presented diagram is just a suggestion of network distribution. Roles were organized by the intended coverage area, thus the Streaming Platform was placed in the core of the cloud. The Directory Facilitator is an agent whose responsibility is to serve the Viewer Proxies and Transcoders with a way to locate near Brokers. The Broker role could be placed in the middle-edge so agents could be reached by the nodes in the layer below, without being too far from the client nodes.

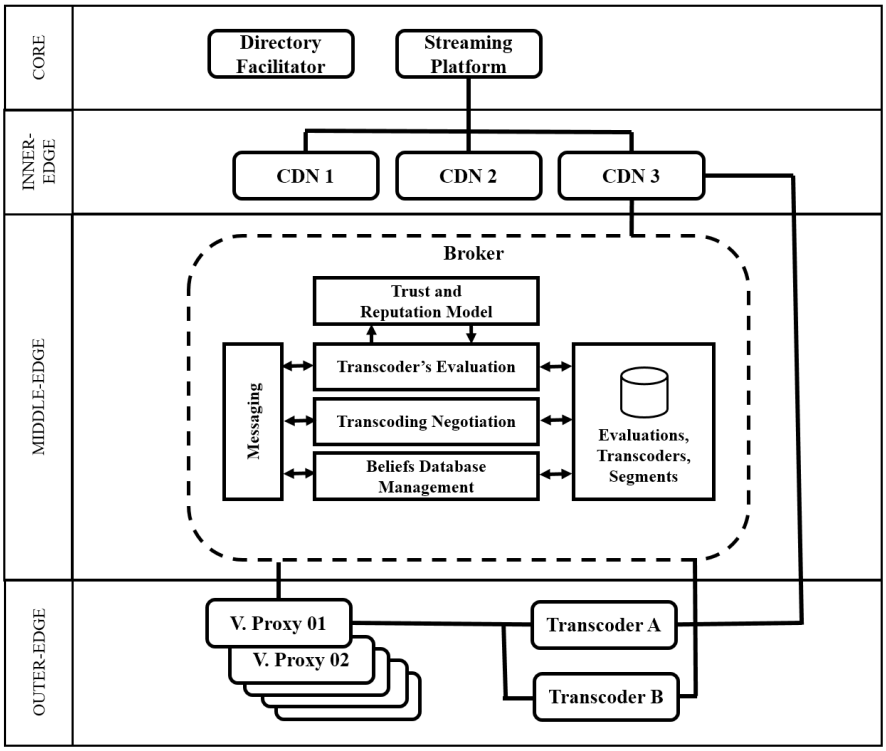

Figure 2: Multiagent architecture with detailed broker's layered architecture.

4.1 Performance Measures

In many aspects, the idea of the agents' rationality is linked to a performance measure ([RN10]). Not only for evaluating the effectiveness of a plan of action but also for learning, when agents need a way to measure their performance. In the ABR domain, it is usual to resort to utility functions as a way to evaluate performance, thus it is natural that the reasoning model of our three roles is based on maximizing a gained utility.

However, before defining the utility function, let us take the video being streamed and sliced into n segments with fixed duration T. We define a video stream S as a sequence of segments, $S = \{s_0, s_1, ..., s_n\}$. In the same way, we take the set of viewers under the influence of a broker as $V = \{v_0, v_1, ..., v_n\}$, and the set of

transcoders as $TC = \{tc_0, tc_1, ..., tc_n\}$. $B(v, tc, s)$ is a primitive function that denotes the bits that a viewer v downloaded from a transcoder tc when it required a segment s, and $I(v, tc, s)$ denotes the time interval passed during the association concerning s. The function $S(v, tc, s)$ says if the association was successful or not, i.e., if the segment was completely downloaded it returns 1, or 0 otherwise. Using these definitions, the utility that a viewer obtained downloading a segment from a transcoder can be defined as Equation 7. Since all the functions have the same arguments, they were omitted for readability. The constant β is empirically defined to adjust the equation considering how bad viewers evaluate video playback interruptions (QoE). The value returned by U can be negative if the association to download a segment was not successful.

$$U(v, tc, s) = \frac{B.S + \beta(T.S - I)}{T} \tag{7}$$

Viewer's utility can be calculated from U fixing a viewer and then iterating all over the transcoders and segments, as shown in Equation 8.

$$U_V(v) = \sum_{tc \in TC} \sum_{s \in S} U(v, tc, s) \tag{8}$$

For understanding how those utility functions should work, let us take the segment duration of 2s and β at 250 and then compare a stream at 3,000 Kbps with another at 6,000 Kbps[1]. This way, the resulted utility will improve by 1%. But if increasing the bitrate resulted in an interval of 1s per minute, then the accumulated bitrate will decrease by 1%, and then the user might prefer the lower bitrate version.

4.2 Reasoning model

Since we propose to deal with video processing on the fly during live video events, we identify our approach as a real-time constrained system. Our agents do not have much time for deliberation, which leads us to design agents where reactive behaviors are predominant. Two of our roles are implemented as purely reactive agents. These are Viewer Proxy and Transcoder roles. The Broker role architecture is hybrid since one of the components is a Trust and Reputation model. Figure 2 shows the internal architecture of the Broker role, besides an illustration of the location of roles in the FEC layers.

The transcoders' trustworthiness evaluation involves an interaction between the layers of Transcoder Evaluation (TE) and the Trust and Reputation Model (TRM). The Broker will collect Viewer Proxy feedback and then apply the Equation 9, where R is the rating concerning the association of Viewer Proxy v with Transcoder tc regarding segment s and M is the maximum utility possible and function U is the Equation 7. Viewer Proxies will give up the interaction if the utility reaches a minimum negative value.

$$R(tc, v, s) = max((U(v, tc, s) - 5 * M(s))/6 * M(s), -1) \tag{9}$$

Many T&R models can be implemented within the layer TRM. If the FIRE TRM is used, Broker agents can evaluate the transcoders' performance based on the feedback provided by the Viewer Proxies. From the point of view of Brokers, Viewer Proxies' feedback is the report of indirect interaction, so witness credibility should be considered. Thus, Brokers must test transcoders in order to obtain direct interaction ratings to be used in Equations 4 and 5.

For the Transcoding Negotiation layer, we introduce Algorithm 1, called ReNoS, on Reputation Node Selection. The algorithm was designed to balance exploration and exploitation so that the most trustworthy transcoders were not overloaded. When a new transcoder registers itself, it receives the maximum possible trust evaluation, therefore raising its chance to be selected in the next iteration. The input $Nodes$ are the set of available transcoders. The input $Factor$ must be equal or greater than 1 and represents how much is desired to explore the set of available transcoders. The default value for $Factor$ is 2. The input $Threshold$ is the interaction threshold defined in [Mar99] and explained before in Section 2.

We have empirically determined that the best trust threshold, below which transcoders should be disregarded, can be obtained by the equation $th(n) = 1 - 1/n$, where th is the desired threshold and n is the number of available transcoders. If a transcoder refuses a job, it is removed from available nodes set for two iterations.

[1]Using the Twitch's encoder, the bitrates of 3,000 Kbps and 6,000 Kbps correspond to 720p-30fps and 1080p-60fps quality versions, respectively. See https://stream.twitch.tv/encoding/.

Algorithm 1 Reputation-based Node Selection (ReNoS)

```
1: procedure ReNoS(Nodes, Factor, Threshold)
2:     Update T&R evaluations of Nodes
3:     Sort Nodes by Trustworth
4:     Distribution ← Empty dictionary
5:     Probability ← 1.0
6:     for Each N in Nodes do
7:         if Trustworth(N) ≥ Threshold then
8:             Distribution[N] ← Probability / Factor
9:             Probability ← Probability − Distribution[N]
10:            Last ← N
11:    Distribution[Last] ← Distribution[Last] + Probability
12:    Selected ← Draw a node by Distribution
13:    return Selected
```

5 Experiments

Real-time transcoding of a live event stream requires tight synchronization. Transcoders must download a part of the stream before starting the transcoding and only after the job is done the resulting segment will be available for the viewers. The contribution of a transcoder settles in close relation to its processing power and network bandwidth. Notwithstanding, our approach must guarantee that transcoded segments are ready on time for being requested. Experiments were conducted to answer three vital questions (Q1, Q2, and Q3).

Q1 - In Which frequency will viewer proxies require new segments to providers?

In the first experiment, the video Big Buck Bunny (720x480, 24fps, 9 minutes 57 seconds)[2] was prepared for a simulated live video streaming. Four segment durations were used: 1, 2, 4, and 6 seconds. The stream was played in VLC Media Player[3] and hls.js[4]. After the video execution, the logs analysis revealed two facts: i) segments are requested by order; ii) players tend to require segments more frequently in the beginning, before buffer occupancy is stabilized, but after that, the average frequency gets closer to segment duration.

The average time-frequency (Avg) and the standard deviation (SD) of the observed requests after 30% of the total video duration are presented in Table 2. Note that the Avg follows the video duration as cited in Fact 2. Comparing SD values, we note that VLC presents higher values than hls.js.

Table 2: Frequency of segment requests (milliseconds).

Duration	**VLC**		**HLS.js**	
	Avg	**SD**	**Avg**	**SD**
6000	6097	1282	6033	218
4000	4216	919	4096	395
2000	2199	576	2099	138
1000	1101	286	1101	128

The results allow the definition of an initial function to predict the moment when a specific segment might be requested. Considering that segments were ordered starting by zero and n identifies a segment by its order, a predictive recurrence is proposed in Equation 10. Q is the number of segments a player can store in the buffer, and D is the time required to download a segment. Same as in Equation 7, T represents segment duration.

$$M(n) = \begin{cases} 0, & \text{if } n = 0 \\ M(n-1) + D(n-1), & \text{if } n > 0 \text{ and } n < Q \\ M(n-1) + T, & \text{if } n \geq Q \end{cases} \tag{10}$$

[2]See https://peach.blender.org/.
[3]See https://www.videolan.org/vlc/index.html.
[4]See https://github.com/video-dev/hls.js/.

Q2 - How much processing power should an FEC Node have to perform as a transcoder?

In a practical approach, five FFmpeg transcoding methods were applied in three similar computers ([Tom06, Bra]). The subject video was taken from a CSGO match transmitted on Twitch (1280x720, 50fps, 60 seconds)[5]. The goal was to test if a typical domestic computer could be capable of transcoding a segment in a compatible amount of time. The used computers setups are as following: (i) i7-3770s, 3.10GHz, 8GB, HD ST100DM0003, Windows 10; (ii) i7-8565U, 1.8GHz, 16GB, SSD 256 G LiteOn Cv8-8e256, Windows 10, video card GTX 1650; and (iii) i5-6500T, 2.5GHz, 8GB, SSD SanDisk x400 256GB, Windows 10.

The results are summarized in Table 3. We relate the method used, the obtained bitrate reduction, peak signal-to-noise ratio (PSNR) ([Bin15]), average time spent (Avg), and the SD. Since the time spent is near half a second, and from the first experiment we expected that segments are requested every two seconds, results indicate that a common personal computer is capable of performing the transcoder role in our approach. Additionally, data comparison between bitrate reduction and PSNR does not show a clear correlation. Future experiments should be set to investigate how these transcoding methods influence measures of utility concerning Equation 7.

Table 3: Transcoding performance in common personal computers.

Method	**Bitrate Reduction**	**PSNR**	**i7 3770**		**i7 8565U**		**i5 6500U**	
			Avg	**SD**	**Avg**	**SD**	**Avg**	**SD**
Resize to 720x480	59.26%	30.66	321ms	0.72%	319ms	0.31%	415ms	1.45%
Reduce bitrate in 2/3	30.03%	33.15	328ms	0.81%	349ms	0.60%	467ms	11.15%
CRF 25	26.24%	40.21	382ms	1.57%	400ms	2.09%	489ms	1.78%
CRF 23	9.05%	42.16	388ms	0.59%	408ms	1.43%	500ms	1.75%
Default filter	12.39%	33.31	333ms	0.17%	354ms	1.63%	440ms	0.99%

Q3 - Can ReNoS algorithm perform as well as other selecting algorithms?

In this, Viewer Proxy agents have buffered players of fixed size, and transcoder agents have a similar processing power. Broker agents can select nodes using three algorithms: random choice, multi-armed bandits upper confidence bound (MAB UCB), and ReNoS. Random choice just draws a transcoder in every iteration by chance. The MAB UCB was implemented as defined in [Sli19], but using trustworthiness evaluations provided by the TRM layer as the average reward.

The experiment was designed to simulate our approach executing a distributed transcoding during a live video streaming session. An implementation of our approach was written using JADE, a popular open-source agent platform in Java ([BCG07]). The simulation parameters, as segments request frequency and time spent transcoding, were set up accordingly to previous experiments.

From Q1 observations, viewers will request segments at their duration, i.e., every 2 seconds. Viewers' players' buffers were set five-segment lengths. Transcoders are set up to one of the profiles of Table 4. Upload speed determines how long a viewer will wait for a segment but transcoding times include time spent downloading original segments from the server. In Table 4 values are consistent with the range observed in the Q2 experiment. Transcoders of A profile have more than sufficient resources to be good. B profile is just sufficient. C profile, otherwise, will seldom complete the tasks successfully. As a result, A and B transcoder agents are expected to accumulate positive utility, but the C kind is more prone to lose.

Table 4: Transcoder profiles.

Profile	**Upload speed**	**Transcoding Time (avg.)**
A	$5Mbps \pm 15\%$	$400ms \pm 25\%$
B	$2Mbps \pm 20\%$	$400ms \pm 25\%$
C	$1Mbps \pm 25\%$	$400ms \pm 25\%$

In the simulated live video streaming, a new segment is generated every 2 seconds. Despite in a real situation segments should have different bitrates, in the experiment every segment is 2s in length and has 1MB after transcoded. The total video length is 200 seconds or 100 segments. Also, agents are willing to exchange information, never reject an offer and do not lie about evaluations. Those are assumptions that hardily will

[5]See https://media.xiph.org/video/derf/.

be found in a real scenario, but our objective is to compare the three algorithms in fair conditions. Besides, assumptions are close to those used to test the FIRE T&R model in [HJS06].

JADE environment was populated with one broker, twelve identical viewers, and three transcoders, one of each profile. For bootstrapping the FIRE T&R model, trust evaluations were initially set to 1.0, and then transcoding jobs were randomly assigned for the first fifteenth segments, and the utility informed by the viewers was used to calculate transcoders' trust values. After this, the ReNoS algorithm was used. Regarding the utility evaluations, in Equation 7 the β factor used was 250, so the Equation 9 minimum rating value (-1) is reached after four iterations.

The experiment was repeated three times for each algorithm, and then the average utility gained was calculated for each iteration. Table 5 summarizes the experiment outcomes. In Figure 3 (a), we present the accumulated utility over the average utility gain by segment, where the radius represents the standard deviation. In (b), we present the accumulated utility obtained with each selection method over time, calculated using Equation 8. We can see that both MAB UCB and ReNos performed significantly better than a random choice, obtaining similar results. Comparing ReNoS and MAB UCB means with 95% confidence interval, we found out that the mean of one is inside the interval of another. Besides, comparing with a T-Test with the same confidence interval, we found a t-Value of $-0,563$, and a p-Value of 0.574. Thus, despite the accumulated utility using MAB UCB was higher, the statistical tests revealed that there is no significant difference between the two algorithms.

Table 5: Comparison of utility gain from Random Choice, ReNoS, and MAB UCB.

Algorithms	**Avg. Utility per Segment**	**Std. Dev.**	**Accumulated Utility**	**Improvement over Random Choice**
Random Choice	955.97	7,910.41	95,596.56	-
ReNoS	3,528.18	5,365.70	352,817.74	269%
MAB UCB	3,842.14	5,269.73	384,213.91	302%

However, analyzing the accumulated task assignments over time, as shown in Figure 4 (a) and (b), we can see that ReNoS was able to rapidly identify the harmful performance of profile C, avoiding assigning tasks to it in the early run. After this, the number of tasks was similarly assigned between Profiles A and B. On the other hand, the MAB UCB was too able to learn about profile's different performances, but the difference between assignments of profiles A and B is higher than that observed using ReNoS. We consider that in an environment where load balancing is important, our algorithm could be more interesting.

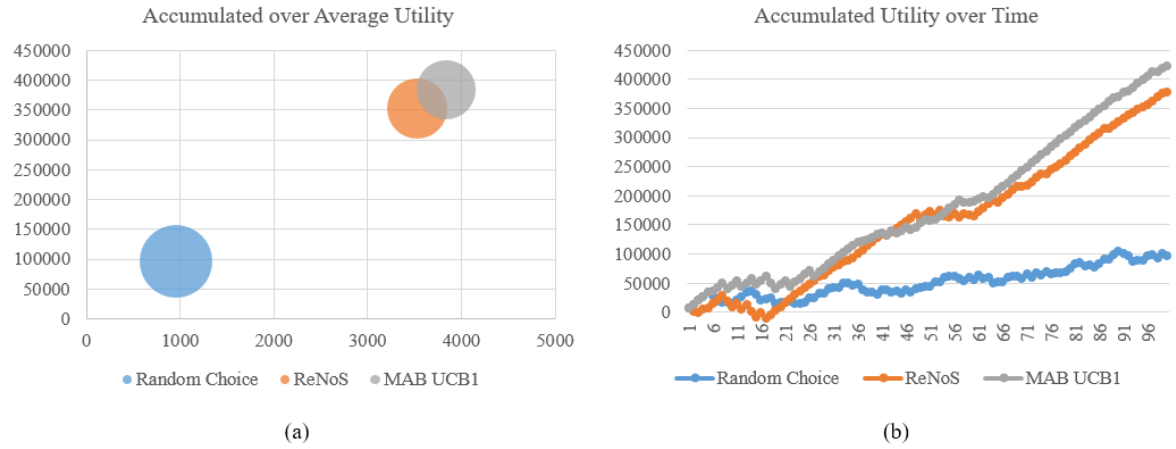

Figure 3: (a) – Accumulated utility over Average utility. (b) – Accumulated utility over time.

Still about reputation feedback from witnesses, in Figure 4 (c) it is possible to see the effect of outdated information in the Broker's trust evaluations. Observing the evolution of trustworthiness evaluations over time, it is notable that Profile C evaluation became way below the initial threshold (0.66). This occurs because of the delay between the moment when the Broker has to do the offloading of a transcoding task and the moment when it receives the Viewer Proxies' feedback.

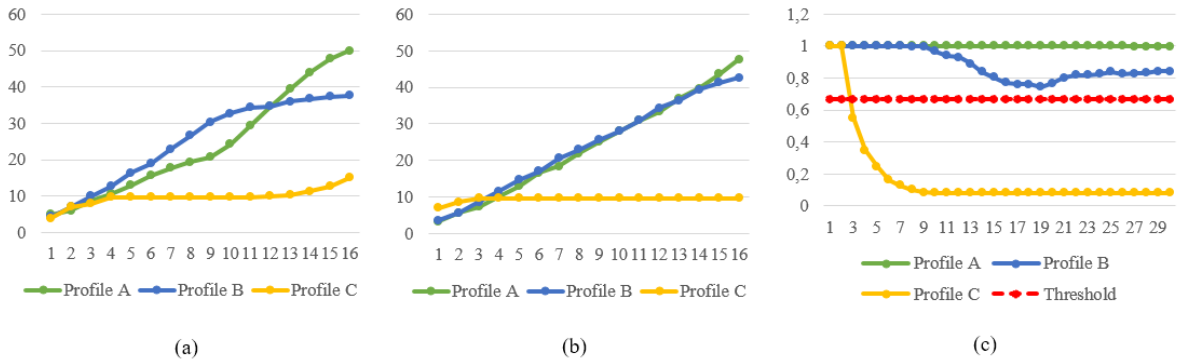

Figure 4: (a) – MAB UCB task assignments. (b) – ReNoS task assignments. (c) – Trustworthiness evaluations over time.

6 Conclusions

In this work, we presented a multiagent model for dealing with the problem of distributing live-video events transcoding throughout FEC nodes and edge devices. Our model related three software agent roles with well-defined responsibilities, the viewer proxy, the transcoder, and the broker. The broker role has the responsibility of coordinating the interaction of the other two. About the FEC layered network architecture, we suggested where to place agents to achieve the desired improvement on QoE.

Since delegating tasks in such an open and dynamic environment as FEC can be risky, we explained how T&R models should be applied to evaluate either transcoders performance as viewers' credibility as witnesses. Then, the algorithm ReNoS, which takes advantage of reputation reports from viewers to select the best nodes for performing the transcoding jobs, was presented.

Three experiments were conducted. The objective of the first two was to understand in which conditions distributed transcoding in the Fog using our model could be feasible, and then those raised parameters were used in a simulation that compared ReNos with two others, random choice, and MAB UCB. As an outcome, we could see that ReNos is much better than random choice and it is at least as performative as MAB UCB, with the advantage of balancing the transcoding load more evenly among the best transcoders.

In future work, other T&R models than FIRE should be tested, inclusively against questions about self-interested agents and their impact on trust evaluations. Interaction among T&R models and multi-armed bandits algorithms deserves further investigation. Simulations should be conducted in an environment closer to real FEC computing.

References

[BBE+19] K. Bilal, E. Baccour, A. Erbad, A. Mohamed, and M. Guizani. Collaborative joint caching and transcoding in mobile edge networks. *Journal of Network and Computer Applications*, 136:86–99, 2019.

[BCG07] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, Ltd, Chichester, West Sussex, England, 2007.

[Bin15] B. Bing. *Next-Generation Video Coding and Streaming*. John Wiley & Sons, Inc, Hoboken, New Jersey, 2015.

[Bra] M. Braun. How can I reduce a video's size with ffmpeg? https://unix.stackexchange.com/questions/28803/how-can-i-reduce-a-videos-size-with-ffmpeg. Online; accessed 08 January 2021.

[BS19] R. Buyya and S. N. Srirama. *Fog and Edge Computing*, chapter Internet of Things (IoT) and New Computing Paradigms, pages 3–21. John Wiley & Sons, Inc., Hoboken, New Jersey, 2019.

[CF98] C. Castelfranchi and R. Falcone. Principles of trust for MAS: cognitive anatomy, social importance, and quantification. In *Proc. Int. Conf. on Multi Agent Systems*, pages 72–79. IEEE Comput. Soc, 1998.

[DKZ15] M. Dabrowski, R. Kolodynski, and W. Zielinski. Analysis of video delay in internet TV service over adaptive HTTP streaming. In *Position Papers of the 2015 Federated Conf. on Computer Science and Information Systems*, pages 143–150, Lodz, Poland, oct 2015. PTI.

[FKZY20] F. Fu, Y. Kang, Z. Zhang, and F. R. Yu. Transcoding for live streaming-based on vehicular fog computing: An actor-critic drl approach. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1015–1020, Toronto, ON, Canada, 2020. IEEE.

[HJS05] T. D. Huynh, N. R. Jennings, and N. Shadbolt. On handling inaccurate witness reports. In *Proc. of 8th Int. Workshop on Trust in Agent Societies*, pages 63–77, 2005.

[HJS06] T. D. Huynh, N. R. Jennings, and N. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006.

[HR14] B. W. P. Hoelz and C. G. Ralha. Towards a cognitive meta-model for adaptive trust and reputation in open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 29(6):1125–1156, 2014.

[HZJ08] H. Huang, G. Zhu, and S. Jin. Revisiting trust and reputation in multi-agent systems. In *ISECS International Colloquium on Computing, Communication, Control, and Management*, pages 424–429, Guangzhou, China, 2008. IEEE.

[HZML17] Q. He, C. Zhang, X. Ma, and J. Liu. Fog-based transcoding for crowdsourced video livecast. *IEEE Communications Magazine*, 55(4):28–33, 2017.

[KCTV17] T. Karagkioules, C. Concolato, D. Tsilimantos, and S. Valentin. A comparative case study of HTTP adaptive streaming algorithms in mobile networks. In *Proc. of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video - NOSSDAV'17*. ACM Press, 2017.

[LDL19] X. Liu, M. Derakhshani, and S. Lambotharan. Joint transcoding task assignment and association control for fog-assisted crowdsourced live streaming. *IEEE Communications Letters*, 23(11):2036–2040, 2019.

[LS10] L. Liu and W. Shi. Trust and reputation management. *IEEE Internet Computing*, 14(5):10–13, 2010.

[Mar99] S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, 1999.

[PS14] K. Pires and G. Simon. DASH in twitch. In *Proc. of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, pages 13–18. ACM Press, 2014.

[RN10] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[Sli19] Aleksandrs Slivkins. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[SS01] J. Sabater and C. Sierra. REGRET. In *Proc. of the* 5^{th} *Int. Conf. on Autonomous Agents*. ACM Press, 2001.

[Sto11] T. Stockhammer. Dynamic adaptive streaming over HTTP: Standards and design principles. In *Proc. of the* 2^{nd} *Annual ACM Conference on Multimedia Systems*, MMSys'11, page 133–144, New York, NY, USA, 2011. ACM.

[SUS16] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. BOLA: Near-optimal bitrate adaptation for online videos. In 35^{th} *Annual IEEE Int. Conf. on Computer Communications (INFOCOM)*, pages 1–9, San Francisco, CA, 2016. IEEE.

[Tom06] Suramya Tomar. Converting video formats with ffmpeg. *Linux Journal*, 2006(146):10, 2006.