Maximizing real-time streaming services based on a multi-servers networking framework

Tian Wang\textsuperscript{a,}\textsuperscript{*}, Yiqiao Cai\textsuperscript{a}, Weijia Jia\textsuperscript{b}, Sheng Wen\textsuperscript{c}, Guojun Wang\textsuperscript{d,e}, Hui Tian\textsuperscript{a}, Yonghong Chen\textsuperscript{a}, Bineng Zhong\textsuperscript{a}

\textsuperscript{a} College of Computer Science and Technology, Huaqiao University, 668 Jimei Avenue, Xiamen, Fujian Province 361021, China
\textsuperscript{b} Department of Computer Science and Engineering, Shanghai Jiaotong University, 800 Dongchuan Road, Minhang District, Shanghai 200240, China
\textsuperscript{c} School of Information Technology, Deakin University, Victoria, Australia
\textsuperscript{d} School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, China
\textsuperscript{e} School of Information Science and Engineering, Central South University, Changsha 410083, China

A R T I C L E   I N F O

Article history:
Received 13 July 2015
Revised 15 October 2015
Accepted 26 October 2015
Available online 30 October 2015

Keywords:
Streaming
Multi-servers
Services
Real-time
Backup

A B S T R A C T

In recent years, we have witnessed substantial exploitation of real-time streaming applications, such as video surveillance system on road crosses of a city. So far, real world applications mainly rely on the traditional well-known client–server and peer-to-peer schemes as the fundamental mechanism for communication. However, due to the limited resources on each terminal device in the applications, these two schemes cannot well leverage the processing capability between the source and destination of the video traffic, which leads to limited streaming services. For this reason, many QoS sensitive application cannot be supported in the real world. In this paper, we are motivated to address this problem by proposing a novel multi-server based framework. In this framework, multiple servers collaborate with each other to form a virtual server (also called cloud-server), and provide high-quality services such as real-time streams delivery and storage. Based on this framework, we further introduce a \((1 - \epsilon)\) approximation algorithm to solve the NP-complete “maximum services” (MS) problem with the intention of handling large number of streaming flows originated by networks and maximizing the total number of services. Moreover, in order to backup the streaming data for later retrieval, based on the framework, an algorithm is proposed to implement backups and maximize streaming flows simultaneously. We conduct a series of experiments based on simulations to evaluate the performance of the newly proposed framework. We also compare our scheme to several traditional solutions. The results suggest that our proposed scheme significantly outperforms the traditional solutions.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

In recent years, we have witnessed the popularity of real-time streaming applications in the various environments, such as the video surveillance systems [1–3]. Not only can the police monitor the risky streets and road crosses, but also general public can employ these applications to enhance security and safety in their business and life. In these applications, there will be many cameras installed in order to cover as large area as one can. For example, Nasution and Foroughi et al. proposed methods to detect and record various posture-based events of interest in elder monitoring application at various home surveillance scenarios [4–6]. In particular, due to the popularity of smart phones and wide accessibility of
wireless connections, people are able to use portable devices to attain the events of interest from remote cameras [7].

Due to the limited resources on each terminal device in the applications, current schemes of the real-time streaming applications cannot well leverage the processing capability and storage space between the source and destination of the video traffic. For this reason, many QoS sensitive application cannot be supported in the real world. For example, there are traditionally mainly two general architectures for real-time services. The first one is the client–server (C/S) scheme where the terminals stream the data captured in the places of interest to the rendezvous server. Remote users can then retrieve the data from the server. This scheme is simple but incurs heavy burdens on the server. It also does not scale well when more terminals of monitoring get involved [8]. Another widely-adopted scheme is the peer-to-peer scheme (P2P). The P2P scheme greatly alleviates the burdens on rendezvous servers, and improves the applications scalability [8,9]. For example, Pudlewski [10] proposed a point-to-point real-time video transmission scheme over the Internet in conjunction with a compression method that is error resilient and bandwidth-scalable. However, any terminal in the P2P scheme acts as both client and server. This leads to the heavy burden on the transmission source. Moreover, because the lightweight server (every terminal) is in short of storage space and processing capability, the P2P scheme cannot support streaming data retrieval. The situation becomes worse when the connections are not stable. The quality of service (QoS) will become unacceptable. To date, it has long been a big challenge to realize the real-time streaming over best-effort packet networks. On the other side, some real-time streaming data is required to be backed up for later retrieval. For example, the data produced by video surveillance system is usually required to be backed up for a long time, which can be retrieved for taking evidence.

In this paper, we proposed a novel framework based on multi-server to address the above problem. In this framework, we take the real-time data streaming and storage into account at the same time, and multiple servers will incorporate with a virtual server (also called cloud-server) to provide both forwarding and storage services. Since our newly proposed framework can fully utilize the resources from dynamic networks [11], the multiple servers provide better robustness in case one transmission channel becomes congested, and achieve high throughput on end users. We carried out a series of experiments in order to test and evaluate the performance of our proposed framework. Compared to the traditional single server scheme, the best server and route/path could be picked out in our solution. For the perspective of users, the service will be provided by a super server, and the multiple servers will coordinate with each other to maximize the simultaneous services.

We summarize the contributions of this paper as follows:

- We proposed a novel framework based on multi-servers to collaboratively provide services for each end user. This framework exploits the diversity of network and efficiently realize the real-time streaming and backup at the same time.
- Based on the proposed framework, we innovatively formulated a brand-new problem in providing real-time streaming services: Maximum Services (MS). This problem aims to maximize the total number of streaming flows and/or services. We proved that this problem is actually an NP-complete problem.
- For the real working applications of real-time streaming services, we innovatively inserted a two-relay restriction into the framework, and therefore simplified the MS problem. We designed an approximate algorithm with foreseeable performance bounds to the optimal solution. Extensive simulations were carried out to validate the effectiveness of the proposed methods for real working applications.
- A new metric Possible Loss (PL) was introduced to evaluate the storage and backup performance. An efficient storage algorithm is designed to achieve the minimized PL.

The rest of this paper is organized as follows: Section 2 presents related work. The details of the framework are presented in Section 3. The approximate algorithm of MS problem is presented in Section 4, followed by Section 5, which is about the performance evaluation. Section 6 concludes this paper.

2. Related work

Clearly, the amount of data in our industry and the world is exploding. Data is being collected and stored at unprecedented rates. For example, video streaming has become a popular application. In 2010, the number of video streams increased 38.8% to 24.92 billion even without counting the user generated videos [12]. Video news clips, movies and TV shows, and videos made and shared by the general public are watched by millions of people every day [13,14]. A considerable amount of data produced by the public needs to be delivered in real-time such as online chatting, online games, video surveillance and so on [15,16]. The challenge is not only to store and manage the vast volume of data, but also to deliver the “Big Data” in real-time.

Traditional client–server based data streaming solutions incur expensive capacity provision cost on the server and do not scale well [8]. On the other hand, peer-to-peer (P2P) streaming greatly alleviates the burdens on servers with good scalability [8,9] where users act as both clients and servers, which then incurs burden for data sources. Many people consider them as suitable infrastructures for supporting real-time streaming. However, P2P networks possess dynamic characteristics that can decrease drastically the performance of these real-time applications. If the communication link between peers becomes bad, the QoS cannot be maintained. In contrast to traditional work, we combine multiple servers into a cloud-server to collaboratively provide reliable and efficient services.

The authors in [17,18] also utilized “multiple servers” to stream media over a lossy packet network. In their work, media packets are typically characterized by different deadlines, importance and interdependencies. Using this information, the client is able to request the transmission of media packets from the multiple servers. They proposed a client-driven rate-distortion optimal packet scheduling algorithm that decides which packet(s) are to be requested from which server(s) at a given request opportunity. The difference between their work and ours is, we do not stream one data
Fig. 1. The multi-server based framework. Sources stream flows to any of the servers and users can retrieve the flows from any of the servers. These servers collaboratively form a group. For example, $r_1$ forwards its data flow to $s_1$. If the connection from $s_1$ to the corresponding user (say $u_1$) is unstable, $s_1$ can select to forward the flow to another server (say $s_2$) and $s_2$ finally forwards the video flow to $u_1$.

flow simultaneously from multiple servers for reliable reasons. Our work is also different from [11] where the source splits one stream into several sub-flows and each sub-flow is streamed separately to the individual users. In our work we assume the media flow is just streamed from one source to the destination fully through only one path which can guarantee the real-time requirement.

On the other hand, the storage of streaming data is another big problem [19]. Most of the real-time streaming data is not the “one-time consumer goods”, which is required to be backed up. For example, the data produced by video surveillance system is usually required to be backed up for a long time, which can be retrieved for taking evidence [20]. Traditional P2P scheme cannot provide this backup service. C/S scheme just provides one backup, with high risk of data loss if this backup server is broken. In this paper, we take data storage and real-time streaming into consideration simultaneously. The introduced multiple servers framework is with the streaming backup ability and during the streaming process the data copies are generated naturally. Streaming data is backed up on at least two servers, which gives two advantages. First, the probability of data loss is lowered. Secondly, while more than one users require the same streaming, the streaming which is already stored on servers can be forwarded to users directly, which decreases the delay and alleviates the stress on a single machine (source node for P2P scheme or the single server for C/S scheme). In our former work (conference poster) [21], we proposed the prototype of the framework without elaborating the motivations behind in more detail. Furthermore, this backup method and extensive experiment results are not introduced.

3. Multi-server based framework

In this section, we first introduce the multi-server based framework which includes multiple cooperative servers. Based on this framework, the problem aiming at maximizing the number of simultaneous streams is formulated.

3.1. The framework

The multi-server based framework is shown in Fig. 1, which includes three sides: sources, cloud-server and end users. Sources are the origins of streams such as IP cameras while end users are requesting these streaming data for service. Multiple servers connect to each other in the networks and collaboratively combine into a cloud-server. In this paper, we assume that the topology of the servers is a fully-meshed network since it will yield the best performance. However, our proposed framework and algorithm can be easily extended to other topologies such as a ring topology. Data flows are first forwarded to one of the servers and then will be further forwarded directly to end users or via other servers for further forwarding, i.e., multiple servers can cooperatively forward the flows based on the exchanges of capacity information of themselves. From the perspective of users, the services are provided by only one server. The advantages are two-fold. One is to provide the user a network channel with adequate capacity and the other is to produce data copies at multiple servers for storage.

Our objective is to maximize the total number of data flows (or the number of services). For each data flow requested by the user, we assume that the capacity of the transmission link should be greater than a fixed threshold $f_i$. For example, to ensure continuous streaming for MPEG-1 data, the capacity at which flows are transported over the network must be no less than 1.5 Mbps [22]. In this paper, we use $f_i$ to denote a data flow itself as well as its capacity requirement. Another assumption is that all of the link capacity could be measured by using the tools such as Pathload [23] or AProb-ing [24].

Before formulating the problem, we first introduce a definition below.

Definition 1 (Flow Integrality (FI) constraint). A flow is under FI constraint, if this flow must be sent from one source to the destination via only one path.
That is to say, the original data flow produced should not be sliced into several separated flows and go separate paths. This is the case for most of real-time applications.

3.2 Problem description

We now formally define the general Maximum Services (MS) problem.

Definition 2 (MS problem). Let a set of source nodes be \( R = \{r_1, r_2, \ldots, r_m\} \), a group of servers be \( S = \{s_1, s_2, \ldots, s_h\} \) and a set of corresponding destination users be \( U = \{u_1, u_2, \ldots, u_m\} \). Source nodes send flows to servers, which users retrieve through this link \((v, w)\). User node \( u_i \) expects/requires the streaming service from a specific source \( r_i \). The data flow delivered to the user should be under the Fl constraint and data flows can be forwarded among servers arbitrarily. The objective is to maximize the total number of data flows simultaneously served (services).

This problem can be further formulated by linear programming. User node \( u_i \) requests the data flow \( f_i \) which is generated by \( r_i \). For each link \((v, w)\), we define a binary variable \( x_{v,w}^k \) where \( x_{v,w}^k \) is equal to 1 if the entire flow \( f_i \) will go through this link \((v, w)\) and is equal to 0 otherwise. The problem is thus formulated as:

\[
\text{maximize } \sum_{k=1}^{m} x_{v,w}^k \text{ s.t. } \sum_{k=1}^{m} f_k x_{v,w}^k \leq c_{v,w} \]
\[
\sum_{k=1}^{m} x_{w,z}^k = \sum_{k=1}^{m} x_{z,v}^k \text{ } \forall z \in S
\]
\[
x_{r_k,u_k}^k = x_{w,u_k}^k \text{ } \forall r_k \in R, \forall u_k \in U
\]
\[
x_{v,w}^k \in \{0, 1\} \text{ } \forall (v, w)
\]

The first constraint represents the capacity constraint on each link, whereas the second constraint represents the flow conservation constraint. The third constraint is to guarantee the user gets the corresponding flow requested. The problem formulation is similar to the multi-commodity flow problem [25] but the Fl constraint makes the difference since one flow cannot be split into multiple sub-flows going separate paths.

The proposed MS problem seems to be a Maximum Flow problem, where flows are also under capacity constraint. However, one of the differences is that our problem aims at maximizing the service number while the Maximum Flow problem aims at maximizing total flows. Another main difference is that in our problem one specific flow must be sent to one specific destination while in the Maximum Flow problem flows are indiscriminate. For these reasons, our MS problem is more complicated than the traditional Maximum Flow problem. Fig. 2 illustrates a simple example for the MS problem. There are only two source nodes: \( r_1 \) and \( r_2 \); two servers: \( s_1 \) and \( s_2 \); and two users: \( u_1 \) and \( u_2 \). The link capacity is also shown beside each link. Users \( u_1 \), \( u_2 \) request flows produced by \( r_1 \), \( r_2 \), respectively. The flows produced by \( r_1 \) and \( r_2 \) are denoted by \( f_1 = 7 \) and \( f_2 = 6 \), respectively. Source nodes \( r_1 \) and \( r_2 \) can forward their data flows to any of the servers. One server can further forward the flow to another server with adequate capacity. In this case, \( r_2 \) may first send its flow \( f_2 \) to \( s_2 \), since the link capacity between \( r_2 \) and \( s_2 \) satisfies the capacity requirement. But, the link capacity between \( s_2 \) and \( u_2 \) cannot support flow \( f_2 \). Then, \( s_2 \) may ask \( s_1 \) for forwarding help. Finally, the optimal network path for flows \( f_1 \) and \( f_2 \) are \( r_1 - s_1 - u_1 \) and \( r_2 - s_2 - s_1 - u_2 \), respectively.

3.3 Complexity discussion

In this subsection, we discuss the complexity of the MS problem. In fact, the general MS problem, where multi-hop forwarding among servers are permitted, is very complicated even when there are only several servers. Obviously, the decision version of the problem is to ask whether a subset of these flows can go through the network to corresponding users while the number of flows simultaneously served is greater than a constant \( C \). In the next, we prove its NP-completeness.

Theorem 1. The decision version of the MS problem is NP-complete.

Proof. Obviously, this problem is in NP. Below, we only prove its NP-hardness.
To prove the problem is NP-hard, we consider a special case of the original problem, which is shown in Fig. 3. There are $k$ source nodes, either connected to $s_1$ or $s_2$. Suppose that the flow $f_k = 1$ for any $k$ and all link capacities between sources and servers are 1. On the other side, there are $k$ users, which are all connected to $s_3$ and $s_4$. The link capacities between users and servers are also 1. These four servers are connected by a server network, which consists of multiple servers. The “virtual connection” means there may be multiple connection paths between the two sides. Obviously, for this special case, the problem can be described as: whether we can forward these $k$ flows from $s_1$ and $s_2$ to the destination $s_3$ and $s_4$, under the flow capacity constraints. This problem is exactly the D2CIF problem [25], which is proved to be NP-hard. □

4. Algorithms for realistic applications

Since the MS problem is NP-complete, a polynomial time algorithm cannot be found. Therefore, we seek to simplify the problem for a practical solution. We note that in reality, if there are too many forwarding hops among the servers, it will lead to both long delay and a high loss rate that may not satisfy the real-time requirement. Moreover, multi-hop forwarding among servers will lead to extra communication cost between servers. In this section, we introduce several special cases of the MS problem, according to the maximum number of hops forwarded by servers. We first simplify the MS problem to a MS-1 problem by permitting only one hop forwarding. Similarly, the MS-2 problem means a maximum of two hop forwarding can be permitted. To be consistent with MS-1 and MS-2, MS-0 represents that there are no intermediate forwarding between the source and the corresponding user. Obviously, this special case MS-0 is exactly the P2P scheme. Moreover, the general MS problem is denoted by MS-$n$ since any number of forwarding may exist. After these relaxations, we design an approximation algorithm with provable performance bound for the MS-2 problem. Simulation results in Section 5 also validate that the performances achieved by this relaxed MS-2 solution is very close to that of the optimal algorithm, which means that the MS-2 algorithm can satisfy the most practical situations.

4.1. MS-1: optimal algorithm for 1-forwarding circumstance

With the multiple servers model, we allow only a one hop forwarding, that is to say, the source node sends its data flow to a specific server and this server directly sends to the end user. The optimal algorithm is easy to attain. For each flow $f_k$, we just need to choose one of the servers as the intermediate server, which can serve the corresponding flow. Where one server can “serve” one flow it means that the capacities from the source and to the end user are both greater than the flow requirement $f_i$. This algorithm is shown as Algorithm 1 and will be used for designing the approximation algorithm for the MS-2 problem in Section 4.3.

**Algorithm 1** MS-0: optimal algorithm to solve the no-forwarding case of the MS problem.

*Input*: Three-party Graph $N(R, servers, U)$ that includes a set of source nodes, a set of servers, and a set of users. 

*Output*: Optimal server selected by each flow.

1: for each source $r_i$ do
2: Output one of the servers (say $s_k$), which can serve the corresponding user. If no such server, output NULL for this flow.
3: end for
4.2. MS-2: a more realistic problem

The two-hop forwarding case problem (MS-2) is defined below.

**Definition 3.** MS-2 problem is a special case of the MS problem where a data flow only allows a maximum of two hops forwarding.

In other words, a data flow can traverse a maximum of two servers before finally arriving at the users. We have the following theorem regarding the complexity of the MS-2 problem.

**Theorem 2.** The decision version of the MS-2 problem is NP-complete.

**Proof.** Obviously, this problem is in NP. In the next, we only prove its NP-hardness.

If only one-hop forwarding is permitted, once the flow arrives at one server: this flow is either forwarded directly from this server to the corresponding user or can be forwarded to another server which will further forward this flow to the corresponding user. In the former case, the flow can be served directly. In the latter case, the situation is more complicated. As shown in Fig. 4, there are 3 servers and all the source nodes are connected to S1. S1 connects to S2 and S3, respectively, with the link capacity c_{12} and c_{13}. S2 and S3 both connect to a set of users. Suppose that the capacity of links from source nodes to S1 and the capacity of the link from S2 and S3 to users is infinity, then the capacity bottleneck only exists among the links between servers. For this special case, we can regard the flows arriving at S1 as items to be packed, and take the link capacity c_{12} and c_{13} as the size of the bin. This problem can be transformed as follows: whether a list of items of size f_1, f_2, ..., f_k can be packed into the 2 bins of the size c_{12} and c_{13}. This problem is exactly the Bin Packing problem, which is a well-known NP-hard problem. □

4.3. Approximation algorithm for MS-2 problem

Unfortunately, the simplified problem MS-2 is still NP-complete. But the two-hops forwarding restriction contributes to the design of the approximation solution for the MS-2.

Before designing the approximation algorithm, we first preprocess the original data flows using Algorithm 1. After the optimal no-forwarding algorithm, the flows directly served by only one server have been picked out (assigned to one of the servers) and we denote the flows left (not assigned already) by f_1, f_2, ..., f_m (m ≤ m). Without loss of generality, we assume that these flows are sorted by non-decreasing order. For each link l_k, which is between two servers (denoted by s_i and s_j), we denote the set of flows that may be served by link l_k by λ(l_k). Flow f_k belongs to set of λ(l_k) if the following three requirements are satisfied: (1) there is adequate capacity for source node r_k to s_j; 2) the capacity of link l_k is greater than f_k; 3) there is adequate capacity from s_j to end user u_q. For each link l_k, we define a variable f^*_k = f_k. Initially, all the f^*_k = NULL. During the running of the algorithm, there are two statuses for one link l_k: not-full or full. Here the “full” means that there is no available capacity left (i.e. its f^*_k ≠ NULL), otherwise it is not full (f^*_k = NULL). If we assign f^*_k to one link and then this link becomes full, we set f^*_k = f_k. We assign flows to the link between servers one by one. Our basic idea is that, one flow f_k cannot be assigned if and only if all the link l_k where f_k ∈ λ(l_k) is full and the flows already assigned to l_k do not belong to any λ(l_j) where the state of link l_j is not-full. The algorithm is shown in Algorithm 2 in detail.

**Lemma 1.** After running the algorithm, the size of the flow not assigned (denoted as f_k) is no smaller than any of flow assigned to the link l_k where f_k ∈ λ(l_k).

**Proof.** For the link l_k which f_k ∈ λ(l_k), there must be a f^*_k = f_k according to our algorithm; otherwise f_k will become the f^*_k. Obviously, f_k is the maximum flow in the link l_k. If f_k is smaller than f_k, according to our algorithm f_k will be prior to f_k to become the f^*_k, which contradicts our assumptions. □

**Lemma 2.** After running the algorithm, if all the links are full, denote N_{opt} as the optimal solution of our problem, then N_U ≥ N_{opt} − n_l, where n_l is the number of links among servers.

**Proof.** Obviously, after running our algorithm, it is impossible to assign any flow f_k, which was not assigned, to any links without removing the flow already assigned. That is, if we add any of the flows not assigned to any link, some flow already assigned must be removed. Moreover, according to Lemma 1, f_k is larger than any of the flows, which were assigned to the link l_k where f_k ∈ λ(l_k). If we assign f_k to any l_k, it means that some f_l ≤ f_k must be removed from that link, which will not increase the total number of flows assigned. For each link, there is at most one flow, which is assigned but cannot be served, and our output N_U is the number of flow that can be served. Obviously, the optimal method can serve at most all the flows already assigned, which means N_U ≥ N_{opt} − n_l. □

**Theorem 3.** Algorithm 2 is a 1 − ε approximation algorithm, where ε is equal to \( \frac{n_l}{N_{opt}} \). The computation time is O((m)^2 + 2m^2), where m (m ≤ m) is the total number of flows not assigned in Algorithm 1 and n_l is the number of servers (servers).
Algorithm 2 MS-2 algorithm.

**Input:** flow set \(\{f_i\}\); links among servers: \(\{l_k\}\); \(B(l_k)\) which refers to the available capacity left for \(l_k\) and the corresponding \(\lambda(l_k)\), which refers to the flows may be forwarded by that link (candidate flows).

**Output:** \(N_u\) - The number of users can be served; \(\Omega(l_k)\) - The flows assigned to each link \(l_k\)

1. \(N_u = 0\); \(i = 0\);
2. sort the flows in the order of nondecreasing by their sizes: \(f_1, f_2, ..., f_m\);
3. for \(i = 1 \rightarrow m\) do
4. if \(f_i\) can be served by one of the links \(l_k\) then
5. assign \(f_i\) to link \(l_k\) (i.e. add \(f_i\) to \(\Omega(l_k)\)) and \(B(l_k) = B(l_k) - f_i\);
6. else if exist some link \(l_k\) who still has available capacity (i.e. \(f^*_k = \text{NULL}\) \&\& \(f_i \in \lambda(l_k)\)) then
7. assign \(f_i\) to link \(l_k\) (i.e. add \(f_i\) to \(\Omega(l_k)\)) and \(B(l_k) = B(l_k) - f_i\);
8. \(f^*_k = f_i\);
9. else
10. for the link \(l_k\) who have no available capacity left (i.e. \(f^*_k \neq \text{NULL}\)) do
11. assign any of the flow \(f_\theta\) already assigned to any other link \(l_p\) who still has available capacity (i.e. \(f^*_p = \text{NULL}\) \&\& \(f_\theta \in \lambda(l_p)\));
12. \(B(l_k) = B(l_k) + f_\theta\); \(B(l_p) = B(l_p) - f_\theta\);
13. if the link \(l_p\) has no available capacity then
14. the maximum flow assigned to \(l_p\) becomes its \(f^*_p\);
15. end if
16. if the link \(l_k\) still has available capacity then
17. assign \(f_i\) to \(l_k\) (i.e. add \(f_i\) to \(\Omega(l_k)\)) and \(B(l_k) = B(l_k) - f_i\);
18. \(f^*_k = f_i\);
19. GOTO step 3;
20. end if
21. end for
22. \(N_{\text{NS}} = N_{\text{NS}} + 1; \|f_i\|\) cannot be assigned
23. end if
24. end for
25. \(N_u = m - |\text{flow whose } f^*_i \neq \text{NULL}| - N_{\text{NS}}\);
26. output \(N_u\) and \(\Omega(l_k)\) for each link \(l_k\);

**Proof.** The flows can be divided into two subsets \(F_x\) and \(F_y\), where \(F_x\) is the subset containing the flows assigned to the links which are not-full (these links are denoted as \(L_x\)) and \(F_y\) contains the flows assigned to the links which are full and the flows not assigned (these links are denoted as \(L_y\)). Thus, the original problem can be divided into two subproblems \(x\) and \(y\), whose input flows are \(F_x\) and \(F_y\), respectively. According to our algorithm, all of the flows in \(F_x\) can be served and the flows in \(F_y\) are not in any \(\lambda(l_k)\), where \(l_k\) is the link in \(L_x\). Denote our output for solving the subproblem \(x\) by \(\theta\). Then, the optimal solution can be denoted as

\[
N_{\text{opt}} = N_{\text{opt}} + N_{\text{opt}}
\]  

where \(N_{\text{opt}}\) is the optimal solution for the subproblem \(x\), while \(N_{\text{opt}}\) is the optimal solution for the subproblem \(y\).

Obviously,

\[
N_{\text{opt}} = |F_x|
\]  

where \(|F_x|\) is the cardinality of \(F_x\). According to Lemma 2, we have:

\[
N_u \geq N_{\text{opt}} - |L_y|
\]

where \(|L_y|\) is the cardinality of link set \(L_y\). Then, we have:

\[
N_u = |F_x| + N_u \geq N_{\text{opt}} - |L_y| = N_{\text{opt}} - n_l \geq \frac{N_{\text{opt}} - n_l}{N_{\text{opt}}}
\]  

where \(\epsilon\) is equal to \(\frac{n_l}{N_{\text{opt}}}\) and can be infinitely small with the increasing of simultaneous online users. We also draw a figure to show the value of \(\epsilon\) in Section 5.

4.4. Streaming backup algorithm

As mentioned before, the multi-server framework can not only achieve real-time streaming but also provide backup services for streaming data. In this section, we analyze the backup performance of the proposed framework. Intuitively, when data flows arrive at a server, this server can keep a copy for later retrieval, so during the delivery process we can achieve multiple backups at the same time. Since we use multiple servers to collaboratively forward flows, data flows also can be stored in multiple servers. The advantages are two-fold: (1) multiple copies of streaming data can be kept on the servers. The video streaming data is backed up in the cloud for retrieving later. For example, the data produced by video surveillance system is usually required to be backed up for a long time, which can be retrieved for taking evidence. (2) While more than one users require the same streaming, the streaming which is already stored on servers can be forwarded to users directly. Users do not need to require the streaming from the original nodes, which decrease the delay.

Intuitively, during the real-time streaming process, too many backups on servers will lead to multiple transmission hops among servers, which increases the delay. In our algorithm, only one time forwarding among servers is permitted.
In other word, a maximum of 2 backups are kept on servers. The algorithm should be designed to fulfill the real-time streaming and complete the backup simultaneously. In order to evaluate the backup performance, several definitions are given first. We define the missing probability of a stream as the probability that there is no one backup on the server. Suppose that there are \( n \) servers and these servers may malfunction with some probability \( p \), which is i.i.d. (independent and identically distributed). Suppose that there are \( \sigma \) broken-down servers and a stream has \( \eta \) backups in several servers, the missing probability \( p_m \) for this stream can be computed by:

\[
p_m = \begin{cases} 
0 & \sigma = 1, 2, \ldots, \eta - 1 \\
\frac{C_{\eta-1}^{n-\sigma}}{C_{\eta}^{n}} & \sigma \geq \eta
\end{cases}
\]

where \( C_m^n \) can be written as: \( \frac{m!}{n!(m-n)!} \).

For traditional multi-server schemes, one flow (stream) is stored in only one server the missing probability is \( \frac{C_{\eta-1}^{n-\sigma}}{C_{\eta}^{n}} \) if \( \sigma \) servers are broken. Obviously, \( \frac{C_{\eta-1}^{n-\sigma}}{C_{\eta}^{n}} \leq \frac{C_{\eta-1}^{n-\sigma}}{C_{\eta}^{n}} (\eta \geq 1) \).

Moreover, our solution outperforms traditional multi-server scheme by:

\[
\frac{C_{n-1}^{\sigma-\eta} - C_{n-1}^{\sigma-\eta}}{C_{n-1}^{\sigma-1}} = \frac{(\sigma - \eta + 1) \times \ldots \times (\sigma - 1)}{(n - \eta + 1) \times \ldots \times (n - 1)} \tag{11}
\]

which validates the backup capability of our solution.

We now propose a "Minimum Missing Probability (MMP)" problem. We first introduce a metric-"Possible Loss" (PL for short), which denotes the missing probability of one stream, that is, the probability the flow is not backed up on the server. The missing probability is the most critical factor for data backup. Obviously, if one flow \( f_i \) is stored in \( \eta_i \) servers, its PL is \( p_i^\eta \), where \( p \) is the malfunction probability for a server. The objective of the MMP problem is to minimize \( \sum \eta_i \). This problem can also be proved to be NP-hard (Appendix A).

The proposed heuristic algorithm is shown in Algorithm 3. The basic idea is to minimize the number of flows backed up in low levels. In this algorithm, we exploit the MS-2 algorithm by introducing 2 parameters, which can be denoted by MS-2(\( S, \lambda(\ell_k) \)). The set \( S \) denotes the collection of flows to be backed up and \( \lambda(\ell_k) \) denotes the collection of candidate flows which may be forwarded by link \( \ell_k \). Initially, we consider broadcasting the flows stored in only one server, i.e., \( \eta_i \) is equal to 1. We assume that a link \( \ell_k \) connects server \( s_x \) to \( s_y \). A flow \( f_i \) is added in \( \lambda(\ell_k) \) if flow \( f_i \) is already stored at \( s_x \) but is not stored at \( s_y \) yet. We then call MS-2(\( S, [\lambda(\ell_k)] \)) algorithm to maximize the number of flows which can be backed up in two servers. As mentioned before, for real-time reasons, only two backups are stored in servers. That is to say, for one flow, there is only one time forwarding among the servers.

### 5. Performance evaluations

This section presents the evaluation of the algorithms using MATLAB. We compare the proposed algorithms (denoted as MS-1 and MS-2) with the traditional "one-server" method and P2P method (MS-0). As a baseline, we also run the brute-force method to get the optimal solution when multi-hop forwarding are allowed among servers, which is denoted as MS-n. In the simulations, by saying "P2P capacity", we mean the capacities from sources to servers or the capacities from users to servers. To simulate the dynamics of network capacities, we assume the P2P capacity obeys log-normal distribution [26] and so do the video flows since, based on Kun-chan's [27] research, flow rates can often be described by a log-normal distribution. Unless otherwise specified, we set the mean of P2P capacity and the mean size of video flows to 1000 kbps (close to 1.06 Mbps [28]) and 600 kbps (in Haakon’s research, videos are encoded in six quality levels which ranges from 250 kbps to 3000 kbps) [29]. Considering the real-time transmission in networks, here we adopt 600 kbps as the mean value and set their corresponding standard deviations as 50,000 and 10,000 respectively. There are 2000 users requesting 2000 flows respectively in the simulations. We ran the corresponding simulation program 5 times to get average results.

We first simulated the environment where servers are deployed isolated in the network. It means that the distribution of capacities between servers is identical to P2P capacities and this capacity may not be adequate for forwarding. Fig. 5 shows the ratios of failed media flows to the total number of flows when the mean value of P2P capacity varies from 800 to 1000. In this simulation, we set the number of servers by 3. This figure shows that the performance is not so good for all methods when the P2P capacity is weak; for example the users are using 3G networks to request flows. But, all methods yield less numbers of failed flows when the P2P

```plaintext
Algorithm 3 MMP algorithm.
Input: flow set \{\( f_i \) \} (1 \leq i \leq m), which is to be backed up at servers \{\( s_j \) \} (1 \leq j \leq n); \{\( S(f_i) \)\}, which refers to the backup servers for flow \( f_i \); links among servers: \{\( \ell_k \)\};
Output: \( \sum \eta_i \), \{\( S(f_i) \)\} and \( \Phi(\ell_k) \)-The flows assigned to each link \( \ell_k \)
1: \( S = \phi \);
2: set \( \Phi(\ell_k) = \phi \) for each link \( \ell_k \);
3: for each flow \( f_i \) do
4: set \( \eta_i = 1 \) if flow \( f_i \) is stored in only one server otherwise \( \eta_i = 2 \);
5: add \( f_i \) to set \( S \) if \( \eta_i = 1 \);
6: suppose \( f_i \) is stored in server \( s_y \), then add \( f_i \) to \( \lambda(\ell_k) \) if \( \ell_k \) starts at server \( s_y \) and remove \( f_i \) in \( \lambda(\ell_k) \) if \( \ell_k \) ends at server \( s_y \);
7: end for
8: call MS-2(\( S, [\lambda(\ell_k)] \));
9: \( S = \phi \);
10: for each \( f_i \), which is the flow can be served in step 9 do
11: add \( f_i \) to \( S \);
12: \( \eta_i = \eta_i + 1 \);
13: suppose \( f_i \) is forwarded from server \( s_x \) to \( s_y \), then add \( f_i \) to \( \lambda(\ell_k) \) if \( \ell_k \) starts at server \( s_y \) and remove \( f_i \) in \( \lambda(\ell_k) \) if \( \ell_k \) ends at server \( s_y \);
14: end for
15: for each link \( \ell_k \) do
16: \( \Phi(\ell_k) = \Phi(\ell_k) \cup \Omega(\ell_k) \);
17: end for
18: OUTPUT \( \sum \eta_i \), \{\( S(f_i) \)\} and \( \Phi(\ell_k) \);
```


capacity is increased. This is because, with the increase of capacity more flows can be forwarded from sources to users. The MS-2 method always yields the best performance among all algorithms. In particular, MS-2 outperforms C/S method and P2P method by 235–1400%, which validates the effectiveness of the proposed multi-server based server structure as well as the algorithm proposed. The C/S method yields the worst performance because one single server should further forward flows to users which increases the failure probability of delivery, compared with P2P method. We also simulated the scenarios where intermediate forwarding between servers are not permitted. This figure shows that the differences between MS-1 and MS-2 is small if servers are deployed isolated. Moreover, there is little difference between the MS-2 and MS-n, which means that a one-hop forwarding among servers is enough for most of the flows.

We then changed the deviation of flows to 1000, which means the size of flows are roughly equal. Fig. 6 shows the evaluation results. This figure is similar to Fig. 5 which validates our method for the application with the same size of flows. The difference is that, the ratios of failed flows have all decreased comparing with that shown in Fig. 5. It shows that the application with stable size of flows can support more users.
Fig. 7 shows the ratios of failed flows when the number of servers varies from 1 to 5. Obviously, when there is only one server, our method performs almost the same as that of the C/S method while the P2P method outperforms the C/S method by more than 200%. But with the increase of the number of servers, the ratio of failed flows decreases exponentially while those performed by C/S and P2P method are constants. This is because, more servers will lead to more links between them, which further leads to more chances for forwarding flows. It means our methods are more efficient with multiple servers because of the collaboration among them.

We also simulated the environment with more dynamics, such as wireless/mobile end users. Fig. 8 shows the ratio of failed flows when the deviation of P2P capacity increases from 10,000 to 50,000, which means the capacity fluctuations become more and more intense. As expected, the ratios of failed flows are growing for all solutions, which means the dynamics of networks would not be a good thing for streaming. However, our proposed methods are more tolerant of this network dynamics, at least much better than the traditional C/S and P2P methods. Even when the deviation of P2P capacity reaches 50,000, the ratio of failed flow is just 9. These simulation results show that our method is better when
applied in dynamic networks, such as wireless or mobile environments.

The first 4 of the following figures show the scenarios when servers are deployed isolated in the network. It reveals the effectiveness of the proposed multi-server based structures. But the differences between MS-1 and MS-2 is small. This is because isolated servers are connected from different places and the limited capacity between two servers limits the number of forwarded flows. In the next, we will compare our methods MS-1 and MS-2 in the scenarios where servers are deployed locally. That is, the servers are connected by LAN or private networks with adequate capacities.

We set the capacity between servers to 100 Mbps to simulate the LAN environment. Fig. 9 shows the ratios of failed flows when P2P capacity is increasing from 800 kbps to 10,000 kbps. Similarly to the former figures, all methods yield better performance when capacity of network increase. MS-2 outperforms MS-1 by 300–700%, which validates the effectiveness of our methods. We also found that there are almost no differences between MS-2 and MS-n, which further validates our one-hop forwarding algorithm especially for scenarios where servers are connected locally.

Finally, we evaluate the backup performance of the proposed framework and algorithm. Obviously, with the
traditional one server C/S method, all of the videos are stored in one server and if this server fails all of the data are lost while with P2P method there is no video backup. On the other hand, with our proposed method, original video data is stored in at least two servers because of the intermediate forwarding between servers. In this simulation, there are 10 servers in LAN and we suppose 5 servers may be breakdown with the same probability. Fig. 10 shows the missing probabilities of flows for traditional CS method and the proposed MMP method. It is shown in the figure that the probability of data loss performed by MMP method outperforms that performed by the traditional one server scheme by 50% or so.

Fig. 11 shows the missing probabilities of flows when the failing servers are increased from 1 to 5. It is shown in the figure that with the increase of failed servers, the probability of data loss performed by both methods increases. In particular, the PL performed by C/S scheme increases linearly while that performed by MMP method has no significant increase. This means our backup method outperforms that performed by the traditional method with one single server.

6. Conclusions

In this paper, we studied the networking aspect of real-time streaming applications. We designed a novel real-time streaming framework where multiple servers form into a cloud-server to collaboratively provide services. Our framework combined the advantages of both the traditional client/server scheme and P2P scheme. First, a server group acts as the streaming servers, which are more powerful. Secondly, the multiple servers can backup streaming data, which can improve the data security if a system breakdown or disaster is encountered. We formulated a problem-MS, which aims to maximize streaming services. We proved that this problem is NP-Complete. Moreover, for application in reality, we permit one-hop forwarding between servers and get the relaxed MS-2 problem. We also proved its NP-Completeness and designed a (1 − ε) approximation algorithm, where ε can be infinitesimally small. The simulation results showed that our framework plus the designed approximation algorithm can provide better services to support more simultaneous streams.

Acknowledgment

Above work was supported in part by grants from the National China 973 Project no: 2015CB352401 and National Science Foundation of China (NSFC) Key Project Fund No. 61532013 and the National Natural Science Foundation (NSF) of China under Grant nos. 61572206, 61202468, 61305085, 61472451, 61272151, 61370007, 61302094 and the Natural Science Foundation of Fujian Province of China (No. 2014J01240) and the Promotion Program for Young and Middle-aged Teacher in Science and Technology Research of Huqiao University (No. ZQN-PY308).

Appendix A

We have the following theorem regarding the complexity of the MMP problem.

**Theorem 4.** MMP problem is NP-hard.

**Proof.** If the value of p is small enough, for example $p \gg p^2$, $\sum_i p^\eta_i$ is only decided by $\Sigma_i p$, where i satisfies $\eta_i = 1$. In other words, to minimize $\sum_i p^\eta_i$ is to maximize the number of flows which are backed up in at least two servers. Obviously, this problem is exact the MS-2 problem, which was proved to be NP-Hard in Theorem 2. □

References

Hui Tian received his BSc and MSc degrees in Wuhan Institute of Technology, Wuhan, China in 2004 and 2007, respectively. He received his PhD degree in Huazhong University of Science and Technology, Wuhan, China. He is now an associate professor in the National Huaqiao University of China. His research interests include network and multimedia information security, digital forensics and information hiding.

Yonghong Chen received B.Sc. degrees from Hubei National University, and M.Eng. and Ph.D. degree degrees from Chongqing University, Chongqing, China, in 2000 and 2005 respectively. He is currently the professor of College of Computer Science and Technology, Huaqiao University, Xiamen, China. His research interests include network security, watermarking and nonlinear processing.

Bineng Zhong received the B.S., M.S., and Ph.D. degrees in computer science from the Harbin Institute of Technology, Harbin, China, in 2004, 2006, and 2010, respectively. From 2007 to 2008, he was a Research Fellow with the Institute of Automation and Institute of Computing Technology, Chinese Academy of Science. Currently, he is an Associate Professor with the School of Computer Science and Technology, Huaqiao University, Xiamen, China, and he is also a Post-Doc with the School of Information Science and Technology, Xiamen University, Xiamen, China. His current research interests include pattern recognition, machine learning, and computer vision.