Kaleidoscope: Real-Time Content Delivery in Software Defined Infrastructures

Qi Zhang, Sai Qian Zhang, Jieyu Lin, Hadi Bannazadeh, Alberto Leon-Garcia
Department of Electrical and Computer Engineering,
University of Toronto, Toronto, ON Canada
{ql.zhang, sai.zhang, jieyu.lin, hadi.bannazadeh, alberto.leongarcia}@mail.utoronto.ca

***Abstract*—Real-time content delivery services such as live media streaming, news casting and real-time event subscription/publication systems have become popular in recent years. Unlike traditional content delivery applications, real-time content delivery requires live content to be processed and delivered to end users in a timely and efficient manner. Furthermore, as both content producers and consumers may change over-time, it is a challenge to provision resources for these applications to achieve high service quality while minimizing total operational costs.**

Fortunately, the recent development of Cloud computing and Software Defined Networking (SDN) enables efficient implementation of real-time content delivery systems. Recently, the concept of Software Defined Infrastructure aims at combining Cloud computing and SDN to provide an unified framework for application deployment and management. In this paper, we present Kaleidoscope, an architecture for real-time content delivery in Software Defined Infrastructures. Kaleidoscope leverages network virtualization, SDN-based broadcasting and dynamic cloud resource provisioning to achieve high resource efficiency and service performance. Specifically, we present a resource management scheme that controls Cloud resource allocation and network configuration at run-time in accordance with service demand. Experiments show that Kaleidoscope is able to achieve lower resource cost while providing high service quality.

I. Introduction

Real-time content delivery services such as live media streaming, web-casting and real-time monitoring have gained significant popularity in recent years. In these applications, a content producer publishes live contents to the content delivery system. The live content then undergoes several stages of processing (e.g. video transcoding and segmentation) and is delivered to content consumers in real-time. The capability of providing live content as it becomes available has generated considerable attention. For example, live video streaming applications like Twitch [1] has received 45% increase in monthly viewers compared to 2013 [2], and its peak Internet traffic has surpassed that of Facebook [3]. It has been reported that live-streaming has become the most popular activity for sports fans online [4], and Cisco predicted that the total live streaming and video on demand will account for more than 90% of Internet traffic by 2016 [5].

Unlike traditional content delivery systems, managing a live content distribution system introduces several key challenges. First, the content created by the content producer must be processed in real time and delivered to multiple content consumers. Therefore, sufficient bandwidth and processing capacities must be provisioned to avoid delay in the delivery of the content. Second, as both content producers and consumers may come from different geographical regions, the content must be delivered simultaneously to multiple locations in a responsive and cost-effective manner. Finally, since both content producers and consumers may join and leave the system over time, there is a need to provision resources dynamically to minimize total cost. However, dynamic service reconfiguration often incurs a cost in terms of resource usage and performance penalty. Therefore, it is necessary to minimize the reconfiguration cost during reconfiguration process.

Fortunately, the recent development of Cloud computing and Software Defined Networking (SDN) has enabled new techniques for implementing real-time content delivery systems. Cloud computing permits dynamic creation, deletion and migration of virtual servers, while SDN enables underlying networks to be dynamically reconfigured for both unicasting and broadcasting. Recently, SDN has also been used to control wide-area networks. For instance, Google has deployed SDN to interconnect data centers across multiple geographical regions [6]. While traditionally SDN and Cloud resources are managed by separate management systems, recently there is a trend towards integrated systems that provide unified management of both Cloud resources and SDN enabled-networks. In particular, *Software Defined Infrastructure* (SDI) [7] aims at providing open interfaces for heterogenous resources, including both compute resources and network resources controlled by SDN. This unified management framework brings numerous benefits for real-time content delivery services:

- Cloud computing enables rapid acquisition and release of computing resources, which allows live content delivery systems to scale up and down dynamically. Furthermore, leveraging heterogeneity in geographically distributed data centers, it is possible to reduce total bandwidth usage by allocating servers in data centers close to end users.
- SDN enables efficient implementation for multicasting at L2/L3 switches and routers. This enables the content delivery system to achieve better network efficiency compared to overlay-based multicasting schemes. In addition, SDN enables user-controlled routing of network flows, which allows for dynamic reconfiguration of multicast sessions. This can improve the overall efficiency of the content delivery system.
- SDI provides support for Network Function Virtualiza-

tion (NFV). The goal of NFV is to virtualize network node functions (e.g. middleboxes such as load balancers, firewall, intrusion detection systems) into building blocks that may be connected together to create communication services [8]. In SDI, NFV modules can be handled as virtual servers running in physical devices (e.g. servers and middleboxes).

Based on these observations, in this paper we present Kaleidoscope, an architecture for real-time content delivery in SDI. Kaleidoscope leverages network virtualization, SDN-based broadcasting and dynamic resource provisioning to achieve better resource efficiency and service performance. A key challenge in the design of Kaleidoscope is the design of the resource management scheme that jointly controls Cloud resource allocation and network configuration at run-time in accordance with service demand. In this paper, we provide our technical solution to this problem, and evaluate its performance using realistic simulations. While we are working on a full implementation of Kaleidoscope, our evaluation results already demonstrate that our system is able to achieve lower resource cost while ensuring high service quality.

The rest of the paper is organized as follows. Section II describes the architecture of Kaleidoscope. We formally present the resource allocation problem in Section III and provide our solution in Section IV. We evaluate our solution in Section V, and draw our conclusion in Section VI.

II. System Architecture

This section provides an overview of Kaleidoscope. We consider a geographically distributed Cloud that consists of multiple data centers of various sizes. We assume these data centers are connected through SDN-enabled networks. To keep our model simple, currently we only consider the case where the physical network is owned by a single Internet Service Provider (ISP). It is in our future work to consider the case where the physical network is operated by multiple ISPs.

The realization of a real-time content delivery system is shown in Figure 1. The system can be divided into 3 tiers. In the *Producer tier*, the content producer is responsible for registering and uploading its content to the system. The updated content is then processed in multiple steps by the various servers and NFV modules (e.g. transcoders) in the *Cloud tier*. In this paper, we used the term server to refer to either a server that runs in a Cloud, or a NFV module in the network. The output will then be distributed to end users using multicast mechanisms. In the ideal scenario, we would like to use network-level multicast supported by SDN to deliver content all the way to the end users. However, since not every network domain supports network-level multicast, In Kaleidoscope, the output content is first distributed to multiple distribution servers (i.e. content servers) using network-level multicast mechanisms. Then, the *User tier* which is beyond the control of Kaleidoscope finally delivers the content to end users using unicast mechanisms.

The implementation of Kaleidoscope is shown in Figure 2. In a typical scenario, the content producer submits a request

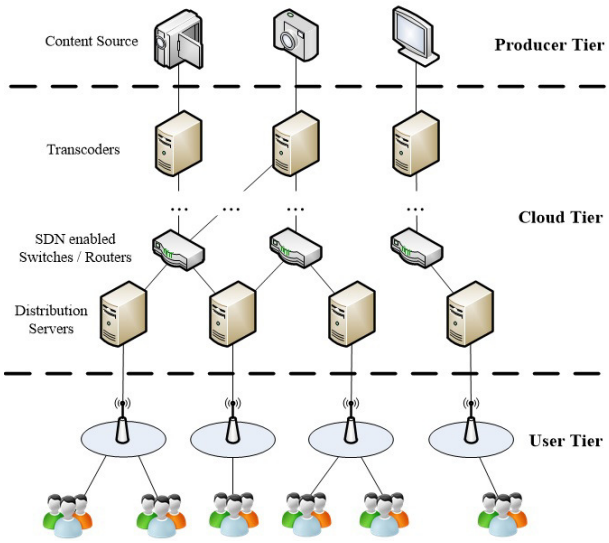

Fig. 1. Architecture of a Real-Time Content Distribution System

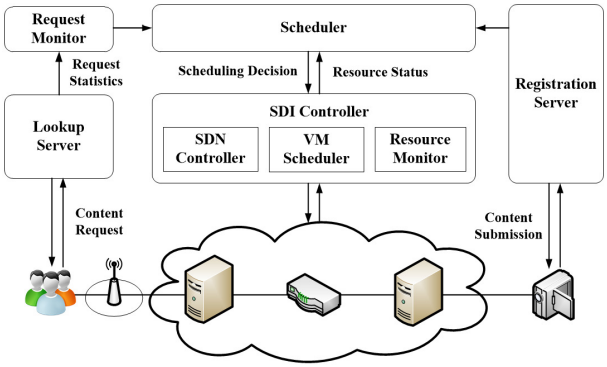

Fig. 2. Architecture of a Real-Time Content Distribution System

to the *Registration Server*, which registers the content in the system. The request also allows the content producer to specify the intermediary processing units as well as the bandwidth requirement after each processing step. The content submission request is then forwarded to the scheduler, which makes an initial scheduling decision. The scheduling then invokes the API of the SDI Controller, allowing the VMs and VN to be allocated in the physical infrastructure. At run-time, each end user may send its content request to a *Lookup server* (such as load balancers and DNS servers), which will then redirect the request to an appropriate distribution server. The end user then establishes the connection with the content server to start the streaming session.

Over time, the *Request Monitor* collects the statistics of content requests as well the resource status information. The scheduler then uses this information to make reconfiguration decisions. The dynamic reconfiguration is performed periodically. However, when demand spike occurs, the Request Monitor may trigger the Scheduler to make immediate scheduling decisions. In this work, we assume content demand is a known

for each location at a given time. The demand forecasting and estimation problem has been studied extensively in the literature [9], [10], and existing solutions can be applied to Kaleidoscope. Finally, it should be mentioned that many real-time streaming systems allow end users to assist each other by forming an overlay network [11], [12]. Kaleidoscope can handle this case by letting the Request monitor to capture the aggregate demand from each access network.

As end users may join and leave the system at any time, each multicast topology may need to be reconfigured over time. With the global view of network state, SDN allows flow migration to be done promptly within seconds. However, for live contents, even interruptions of a few seconds can be disruptive, not to mention that the time-consuming process of launching or migration of servers may also be involved during topology reconfiguration. To minimize service disruption, currently we do not perform server migration in Kaleidoscope. Rather, the multicast topology is gradually reconfigured as demand arrives and leaves the system. Specifically, when new demand joins the system that causes the multicast tree to route to a new distribution server, we construct a path from the new distribution server to multicast tree. Similarly, when demand leaves the system, we can gradually remove the the branch of the multicast tree that is no longer being used. In this case, the reconfiguration cost refers to the time of setup a new path to the distribution server, and the time to remove a path (and perhaps deallocate the distribution server). In the following sections, we shall describe the problem formulation as well as our technical solutions for the dynamic multicast tree scheduling problem in Kaleidoscope.

III. Problem Formulation

We now formally introduce the multicast tree scheduling problem. Given a set of real-time content delivery services, some of which has already been scheduled, our goal is to find a new resource allocation of services that maximizes the total net revenue. Mathematically, we model the network as a graph $G = (N, L)$, where each node $n \in N$ can be a server (either a virtual server or a NFV component), switch or router. In our model, there are R types of resources. Each node n has resource capacity C_{nr} for resource $r \in R$. Each link $l \in E$ has bandwidth capacity C_l and latency δ_l.

In our system, there are I contents to be published, each content $i \in I$ has a source (i.e. the content producer) s_i and a set of destinations D_i (i.e. access networks). Similar to existing work [11], we assume each destination $d \in D_i$ has an aggregate demand q_{id} in terms of bandwidth usage, regardless of whether overlay-based multicast is used in the user tier. Each content i from s_i is forwarded to D_i through a tree t_i that involve one or more servers (e.g. virtual servers and NFVs). We assume each content i requires K_i processing steps, each step $k \in \{1, 2, ...K_i\}$ needs to be performed in a server p_{ik}. For each content $i \in I$, let T_i denote the set of all feasible multicast trees. While in theory it is possible to have multiple multicast trees for the delivery of a single content i each carrying a fraction of the traffic, doing so at network level is difficult for several reasons. First, implementing multi-path routing using software such OpenFlow can be a complex task. Second, using multiple broadcast trees would introduce synchronization issues at both servers and end host. Thus, in this work we assume a single multicast tree rooted at the content producer is used. Let $x_{it} \in \{0, 1\}$ denote whether multicast tree $t \in T_i$ is selected for multicasting content i. The following constraint must be satisfied:

$$\sum_{t \in T_i} x_{it} \leq 1 \qquad \forall i \in I \tag{1}$$

For each multicast tree $t_i \in T_i$, define $a_{itkn} \in \{0, 1\}$ as a known constant that denotes whether tree t_i places the kth server at node n. Similarly, let a_{itkl} denote whether link $l \in E$ is used by multicast tree t_i between $k-1$th and kth server. For the sake of simplicity, let source s_i represent the 0th server, distribution servers as the K_ith servers and each destination as the one of the $(K_i + 1)$th servers. Furthermore, let b_{ik} denote the bandwidth usage by a multicast tree i between the $(k-1)$th and the kth server.

A. Modeling Capacity constraints

We now describe the server and network capacity constraints. Our model assumes there are M type of servers. For each content $i \in I$, define $b_{ik\tau} \in \{0, 1\}$ as a known constant that indicates whether of p_{ik} is of type τ. Let $y_{\tau n} \in \mathbb{N}^+$ denote the number of servers of type τ at location n, and p_τ denote the processing capacity of a type τ server. Essentially p_τ determines the maximum number of streams that can be processed by a type τ server. For example, a transcoder application may process at most 10 streams simultaneously. This implies the following constraint must be satisfied:

$$\sum_{i \in I} \sum_{k=0}^{K_i+1} \sum_{t \in T_i} x_{it} a_{itkn} b_{ik\tau} \leq y_{\tau n} p_\tau \quad \forall \tau \in M, n \in N. \tag{2}$$

Each server of type τ also has resource requirements. Let $c_{\tau r}$ denote the resource requirement of a type τ server for resource r. The following server capacity constraint states that the total resource usage should not exceed the capacity of each node:

$$\sum_{\tau \in M} y_{\tau n} c_\tau \leq C_{nr} \qquad \forall n \in N \tag{3}$$

Similarly, the link capacity constraint can be stated as:

$$\sum_{i \in I} \sum_{k=0}^{K_i+1} \sum_{t \in T_i} x_{it} a_{itkl} b_{ik} \leq C_l \qquad \forall \tau \in M, l \in L \tag{4}$$

B. Demand Constraint

As the distribution servers (i.e., the K_ith processing unit) delivers the content to end users using unicast, these leaf nodes must have sufficient capacity to deliver the content to end users. Let B_n denote the outgoing bandwidth capacity of node n that can be used to serve end users, and let B_{in} denote the bandwidth capacity of node n allocated for delivering content

i. Furthermore, let σ_{itdn} denote the demand from d assigned to distribution server at n for content i delivered using multicast tree t. We now have the following demand constraints:

$$\sum_{d\in D_i}\sum_{t\in T_i}\sigma_{itdn}b_{i(K_i+1)} \le B_{in} \quad \forall i\in I, n\in N \tag{5}$$

$$\sum_{i\in I} B_{in} \le B_n \quad \forall n\in N \tag{6}$$

$$\frac{B_{in}}{B_n} \le x_{it}a_{itK_in} \quad \forall i\in I, t\in T_i, n\in N \tag{7}$$

$$\sum_{t\in T_i}\sum_{n\in N}\sigma_{itdn} = q_{id} \quad \forall i\in I, d\in D_i, n\in N \tag{8}$$

Eq. (5) ensures that total bandwidth demand for content i assigned to n should be at most B_{in}. Eq. (6) specifies that the total bandwidth allocated should not exceed the total available bandwidth of the physical node. Eq. (7) states that the $B_{in} \ge 0$ can only be true if n is a leaf node of the multicast tree for content i. Finally, eq. (8) ensures that total demand from each location d is fully satisfied.

C. Optimization Objective

Given a set of real-time content delivery requests, some of which have already been scheduled, the goal of the scheduling problem is to find a new resource allocation configuration that maximizes the total revenue minus the total operation cost. Let R_i denote the revenue of serving a content $i \in I$. Total revenue R obtained from serving contents becomes

$$R = \sum_{i\in I}\sum_{t\in T_i} x_{it}R_i. \tag{9}$$

The operational cost consists of (1) resource usage cost, (2) run-time performance cost and (3) reconfiguration cost. Specifically, Let $\pi_{\tau n}$ denote the cost of allocating a type τ server at n, π_n denote the unit cost for the outgoing bandwidth in the user tier at node n, and π_l denote the bandwidth cost of link l in the Cloud tier, the resource cost C becomes

$$C = \sum_{\tau\in M}\sum_{n\in N} y_{\tau n}\pi_{\tau n} + \sum_{i\in I} B_{in}\pi_n + \sum_{i\in I}\sum_{k=0}^{K_i}\sum_{t\in T_i} x_{it}a_{itkl}b_{ik}\pi_l. \tag{10}$$

The performance cost refers to the penalty due to violation of performance constraints. Many types of real-time content delivery applications such as real-time monitoring have both bandwidth and end-to-end delay requirements. While the capacity constraints ensure sufficient bandwidth is allocated to each multicast tree, the penalty due to violation of delay requirement still need to captured in our model. In this case, we assume each content i has a maximum expected end-to-end delay $\bar{D}_i$ that is specified in the content request. We also assume each server τ has a processing delay δ_τ. For each multicast tree t rooted at s_i with destinations N_i, let $\mathcal{L}_{tn}$ denote the delay between the source node of multicast tree t and n. $\mathcal{L}_{tn}$ can be computed by summing up the link delay and processing delay between s_i and n in multicast tree t.

$$\mathcal{L}_{tn} = \sum_{k=1}^{K_i} x_{it}b_{ik\tau}\delta_\tau + \sum_{t\in T_i}\sum_{k=1}^{K_i}\sum_{l\in L} x_{it}a_{itkl}\delta_l \tag{11}$$

Similarly, let $\mathcal{L}_{dn}$ denote the average communication delay between $d \in D_i$ and a distribution server n in the user tier. The delay penalty cost P can now be expressed as

$$P = w\cdot\sum_{d\in D_i}\sum_{t\in T_i}\sum_{n\in N}\sigma_{itdn}\left(\frac{\mathcal{L}_{tn}+\mathcal{L}_{dn}}{\bar{D}_i} - 1\right), \tag{12}$$

where the term in the bracket captures the ratio between the violation in delay $\mathcal{L}_{tn}+\mathcal{L}_{dn}-\bar{D}_i$ and the expected maximum delay $\bar{D}_i$, and w represents the per user penalty cost.

Finally, as mentioned in Section II, the reconfiguration cost can be measured by the waiting time of setup a new path to the distribution server, and the time to remove a branch in the multicast tree. Let $\bar{t} \in T_i$ denote the the old multicast tree that is being used, and t denote the new multicast tree that is to be provisioned. We define $\Delta_{\bar{t}t}$ denote the cost of changing multicast tree $\bar{t}$ to t. $\Delta_{\bar{t}t}$ can be computed by summing up the difference in the two multicast trees weighted by reconfiguration time. The reconfiguration cost G becomes:

$$G = \sum_{i\in I}\sum_{t\in T_i}\Delta_{\bar{t}t}\pi_{\bar{t}t} \tag{13}$$

The dynamic multicast tree scheduling problem becomes

$$\max_{\{x_{it},\sigma_{itdn}\}} R - C - P - G, \tag{14}$$

subject to constraints (1)-(8). It is easy to show that this problem is $\mathcal{NP}$-hard. In fact, this problem generalizes multi-source unsplittable flow problem, which is $\mathcal{NP}$-hard to approximate with an approximation factor better than $\Omega(|L|^{\frac{1}{2}})$ [13].

It may seem that our problem shares many similarities with the connected facility location problem [14], whose goal is to find a set of servers to be allocated that have sufficient capacity to serve end users, and construct a Steiner tree to connect the servers such that the sum of server allocation cost, the cost of connecting end users to servers and cost of constructing the steiner tree is minimized. However, the connected facility location problem neither considers the intermediary servers nor the bandwidth capacity of the links in the Steiner tree. In particular, routing flows through intermediary servers may introduce loops [15] in the multicast tree, which is not captured by the classic Steiner tree problem.

IV. Solution Algorithm

In this section, we present our algorithm for the dynamic multicast tree scheduling problem. As this algorithm is reconfiguration-aware, it can be used for both scheduling a new content request or adapting existing multicast trees to new demand patterns.

It is easy to see that linear programming (LP)-relaxation based solutions such as [16] do not scale well for our problem due to $O(n^{n-2})$ multicast trees T_i for each $i \in I$. Therefore, we resolve to develop greedy algorithms for our problem.

Algorithm 1 Resource Allocation Algorithm

Estimate demand for content $i \in I$ at each location in $d \in D_i$
for each $i \in I$ do **do**
  $w \leftarrow 0$, $\kappa \leftarrow 0$, $q_{id}^{\kappa} \leftarrow q_{id}$ for all $d \in D_i$
  while $w \leq w_{max}$ **do**
    while $\exists d : q_{id}^{\kappa} \geq 0$ **do**
      for $n \in N$ that can schedule a K_ith server for i **do**
        Compute a set of demand $\{\sigma_{idn}^{\kappa}\}$ using eq. (17)
      $\{\sigma_{idn}^{*\kappa}\} \leftarrow \{\sigma_{idn}^{\kappa}\}$ with the lowest cost
      for $d \in D_i$ **do**
        $q_{id}^{\kappa} \leftarrow q_{id}^{\kappa} - \sum_{n \in N} \sigma_{idn}^{*\kappa}$
      $\kappa \leftarrow \kappa + 1$
  // now construct multi-cast tree
  $\kappa \leftarrow 0$
  while $\kappa \leq |n_{K_i}|$ **do**
    for $n \in N$ **do**
      for $k = 1...K_i$ **do**
        compute $score(n, k, \kappa)$ according to equation (17)
      $N_f \leftarrow$Top N_{th} nodes with lowest costs
    for each $\{n_1, n_2..., n_{K_i}\} \in N^{K_i}$ **do**
      Compute C_{n_κ} according to equation (16)
    $P_{\kappa}^{*} \leftarrow C_{n_\kappa}$ with the best cost
    Embed virtual links according to P_{κ}^{*}

However, unlike the traditional greedy virtual network embedding algorithm (e.g. [17]), our algorithm must find a right balance between minimizing total resource cost and meeting delay requirements. If we place servers based solely on resource cost, the resulting multicast tree may not satisfy the delay requirement for the content.

Motivated by this observation, in our algorithm we first identify the distribution servers to which end users should be connected. Once the distribution servers have been selected, we then construct a multicast tree to connect the distribution servers to the source. Finding the optimal set of distribution servers can be formulated as a special case of the capacitated facility location problem [18]. In this case, we propose a greedy algorithm similar to the approximation algorithm for the facility location problem. The greedy heuristic proceeds in iterations. In iteration κ, we select a content server at a node $n \in N$ and assign a set of demands σ_{idn}^{κ} from d to n to minimize a cost function $C_{\sigma n}^{\kappa}$:

$$C_{\sigma n}^{\kappa} = \min_{\{\sigma_{idn}^{\kappa}\}} \left\{ \frac{f_n + \sum \sigma_{idn}^{\kappa} \mathcal{L}_{dn}}{\sum_{\bar{m} \in \bar{N}} \sigma_{idn}^{\kappa}} \right\} \tag{15}$$

where f_n is the cost for operating a distribution server at n:

$$f_n = \begin{cases} w \cdot \mathcal{L}_{s_i n} + \pi_{n\tau(i,k)} + s_{n\tau(i,k)} & \text{There is no server at } n \\ w \cdot \mathcal{L}_{s_i n} + \pi_{n\tau(i,k)} & \text{Otherwise} \end{cases}$$

The value of $C_{\sigma n}^{\kappa}$ can be computed optimally in a greedy manner, by first sorting access networks in D_i in increasing order of distance to n, and then greedily assigning demand to n according to the order. Due to the $\sum_{\bar{m} \in \bar{N}} \sigma_{idn}^{\kappa}$ term in the denominator in equation (15), the value of $C_{\sigma n}^{\kappa}$ will decrease at first. But as $\mathcal{L}_{dn}$ increases over time, the value of $C_{\sigma n}^{\kappa}$ will eventually start to increase. Finding this optimal turning point that minimizes $C_{\sigma n}^{\kappa}$ can be done greedily using binary search.

Once we have identified the distribution servers to which the demand should be assigned, we then need to construct a multicast tree from the source to the distribution servers. Finding the optimal multicast tree can be modeled as a special case of the Steiner tree problem. However, the traversal of intermediary servers makes the traditional spanning tree-based approximation algorithm [18] ineffective, due to the possible occurance of loops. In our context, as we are also minimizing the height of the multicast tree, we apply a simple greedy heuristic as follows. We first sort the distribution servers in decreasing distance to s_i, and iteratively establish a path p_n from s_i to each node that leverages existing allocated servers and links as much as possible. Specifically, let $u_{ikn}^{\kappa} \in \{0, 1\}$ as a boolean variable indicates whether there is a server responsible for the kth processing step of content i at node n at the start of iteration κ, and let $u_{ikl}^{\kappa} \in \{0, 1\}$ indicates whether link l is used for transferring content after the kth processing step. Let $\tau(i, k) \in M$ represent the type of the kth server. As the κth iteration, we allocate a path from s_i to the n_κ, the κth node in the list that minimizes the allocation cost

$$C_{n_\kappa} = \sum_{k=1}^{K_i} \sum_{n \in p_n} u_{ikn}^{\kappa} \pi_{\tau(i,k)n} + \sum_{k=0}^{K_l} \sum_{l \in p_n} u_{ikl}^{\kappa} b_{ik} \pi_l \tag{16}$$

Finding the exact minimum cost path can be done by examining all combinations of K_i nodes and find shortest paths between them, which can be done in $O(K_i N^{K_i+1} \log N)$ time. This algorithm performs well when K_i is small. However, when K_i is large, this algorithm can take very long to run. To improve the running time of the algorithm, we can focus our search in a subset of the nodes. The intuition is that the node selected for hosting servers must either have low resource cost or near the shortest path between s_i and n_κ. Thus, we compute a score for each node in the subset, which is determined by

$$score(n, k, \kappa) = u_{ikn}^{\kappa} \pi_{\tau(i,k)n} + w\mathcal{L}_{s_i n_\kappa} + w\mathcal{L}_{s_i, n_\kappa} \tag{17}$$

and select the top N_{th} nodes for use of computing path between s_i and n_κ. The cost of computing the path becomes $O(N_{th}^{K_i} |N| \log |N|)$ time. Finally, the overall running time of our dynamic multicast tree scheduling algorithm is represented by Algorithm 1. The running time of the algorithm is $O(w|N| \log |N|(|N| + N_{th}^{K_i}))$.

V. Experiments

We have implemented a prototype of Kaleidoscope using OpenStack and OpenFlow, and evaluated the performance of our algorithm using simulations. We simulated a 100 nodes power-law topology where 15% nodes are selected as data centers of various resource capacities, and around 25% nodes as access networks. The resource capacities for CPU, memory and disk of each data center are randomly generated between $0 - 1000$ cores, $10 - 1000GB$ RAM, and $1 - 100$ TB storage, respectively. The link capacities are set between $100MB/s$ to $10GB/s$. As for the content requests, the CPU, memory and disk requirement of each type of server are generated randomly between $1 - 4$ cores, $1 - 10$GB RAM and $0.1 - 1$TB storage. These values are typical in production Cloud data

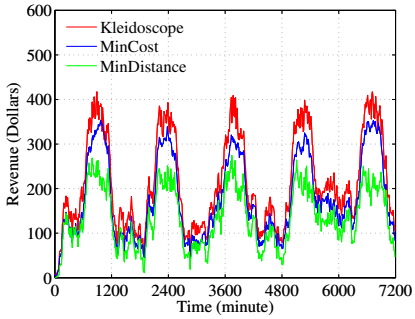

Fig. 3. Net income over time

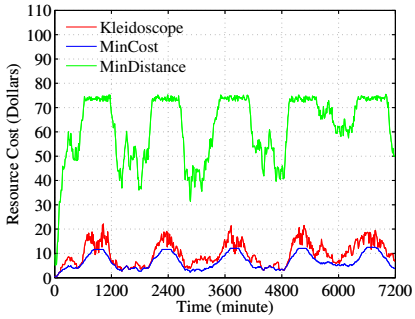

Fig. 4. Resource cost over time

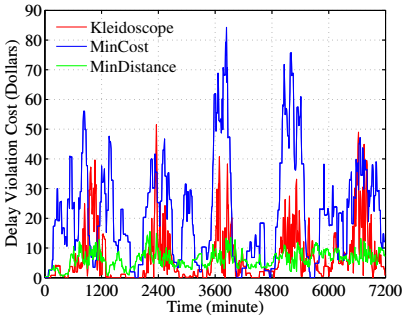

Fig. 5. Delay penalty cost over time

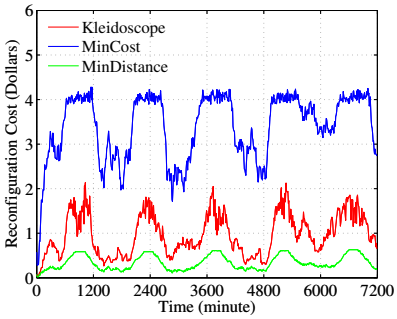

Fig. 6. Reconfiguration cost over time

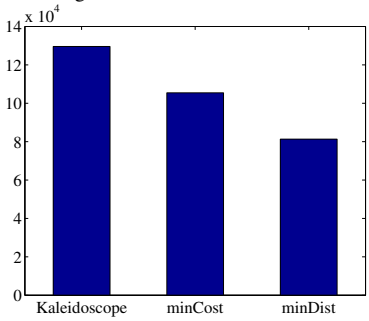

Fig. 7. Total Income

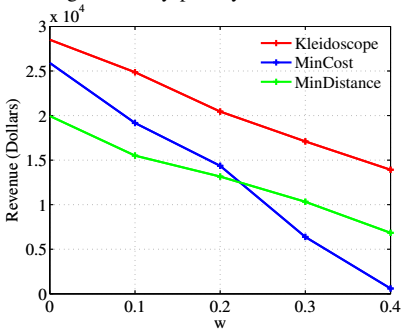

Fig. 8. Impact of w on net income

centers [19]. To simulate the heavy-tail distribution of real-time content popularity [20], we set 2% of the content sessions to be long (between 2-5 days), and the rest to be short (0.5-3 hours). The arrival rate of short content sessions follows a Poisson distribution with an average arrival rate of 18 requests per hour during normal working hours (i.e. between $9am$ and $5pm$), and 6 requests per hour during other hours.

We compared Algorithm 1 with two heuristics. The *minCost* heuristic assigns demand to servers solely based on resource costs, and ignores delay requirements. The *minDistance* heuristic gives high priority to satisfying delay requirements. Specifically, it greedily selects servers that minimize the sum of distances from each access network to the content source. We set per-user penalty to $w = 0.1$ in these experiments. The total net revenue is shown in Figure 3 for all 3 algorithms for a duration of 5 days. Clearly, our algorithm achieves the highest income, while minDistance achieves the lowest income. The reasons can be found in Figure 4 5. In particular, the minCost algorithm achieves much lower resource cost than the minDistance algorithm, but the delay penalty cost of minCost is higher than minDistance. In this case, our algorithm tries to balance the objective of minimizing resource cost and satisfying delay requirement. As a result, it incurs a small amount resources cost to reduce penalty cost and achieve higher net income. Figure 6 shows the reconfiguration cost of each algorithm. We see that minCost incurs the highest reconfiguration cost because it is most sensitive to demand fluctuation. Finally, Figure 7 shows the total income obtained by all 3 algorithms, our algorithm outperforms minCost algorithm by 18%.

To better understand the trade-off between resource cost and delay violation penalty, we varied the value of w between 0 to 0.5 and evaluated the total net income achieved by each algorithm for 24 hours. The results are shown in Figure 8. It is clear that as w increases, the revenue obtained by each algorithm decreases. However, minCost algorithm produces the sharpest drop due to lack of consideration of delay violation penalty. As a result, minDistance outperforms minCost when w is large. However, minCost still yields lower revenue then our algorithm as it does not take resource cost into consideration. In all cases, our algorithm consistently achieves $10-20\%$ gain in net income compared to the other algorithms.

VI. Conclusion and Future Work

In this paper, we present Kaleidoscope, an architecture for real-time content delivery that leverages both Cloud computing and SDN. Kaleidoscope utilizes network virtualization, SDN-based broadcasting and dynamic cloud resource provisioning to achieve better resource efficiency and service performance. We further present a dynamic resource management scheme that modifies Cloud resource allocation and network configuration at run-time in accordance with service demand. Simulations show our system is able to achieve lower resource cost while providing high service quality. Currently we are working on a full implementation of Kaleidoscope that will be deployed in a real SDI such as the SAVI testbed [7].

Acknowledgement

The work of this paper is funded by the Smart Applications on Virtual Infrastructure (SAVI) project under National Sciences and Engineering Research Council of Canada (NSERC) Strategic Networks grant number NETGP394424-10.

REFERENCES

[1] "Twitch," http://www.twitch.tv/.

[2] "Twitch dominated streaming in 2013, and here are the numbers to prove it," http://www.dailydot.com/esports/twitch-growth-esports-streaming-mlg-youtube-2013/.

[3] "Twitch.tv ahead of facebook in peak traffic," http://www.lazygamer.net/general-news/twitch-tv-ahead-of-facebook-in-peak-traffic/.

[4] "Research: Live-streaming most popular activity for sports fans online," http://iq.videonuze.com/content/view/18716.

[5] "Internet video homepage," http://www.cs.cmu.edu/ internet-video/.

[6] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013, pp. 3–14.

[7] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Savi testbed: Control and management of converged virtual ict resources," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 664–667.

[8] "Etsi isg on network functions virtualization (nfv)," http://portal.etsi.org/portal/server.pt/community/NFV/367.

[9] D. Niu, B. Li, and S. Zhao, "Understanding demand volatility in large vod systems," in *NOSSDAV*. ACM.

[10] D. Niu, Z. Liu, B. Li, and S. Zhao, "Demand forecast and performance prediction in peer-assisted on-demand streaming systems," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 421–425.

[11] F. Wang, J. Liu, and M. Chen, "Calms: Cloud-assisted live media streaming for globalized demands with time/region diversities," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 199–207.

[12] A. H. Payberah, H. Kavalionak, V. Kumaresan, A. Montresor, and S. Haridi, "Clive: Cloud-assisted p2p live streaming," in *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*. IEEE, 2012, pp. 79–90.

[13] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar, "Approximation algorithms for the unsplittable flow problem," in *Approximation Algorithms for Combinatorial Optimization*. Springer, 2002, pp. 51–66.

[14] C. Swamy and A. Kumar, "Primal-dual algorithms for connected facility location problems," in *Approximation Algorithms for Combinatorial Optimization*. Springer, 2002, pp. 256–270.

[15] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simplefying middlebox policy enforcement using sdn," in *ACM SIGCOMM*, 2013.

[16] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 783–791.

[17] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[18] V. V. Vazirani, *Approximation algorithms*. springer, 2001.

[19] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 2013, pp. 510–519.

[20] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 41–54.