Computer Networks 55 (2011) 3318–3331

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Impact of user selfishness in construction action on the streaming quality of overlay multicast ☆

Dan Li [a,*], Jianping Wu [a], Yong Cui [a], Jiangchuan Liu [b], Ke Xu [a]

[a] Computer Science Department of Tsinghua University, Tsinghua National Laboratory for Information Science and Technology (TN List), China
[b] Simon Fraser University, Canada

ARTICLE INFO

Article history:
Received 6 December 2010
Received in revised form 20 June 2011
Accepted 22 June 2011
Available online 1 July 2011

Keywords:
Overlay multicast
User selfishness
Construction action

ABSTRACT

A majority of the existing overlay multicast proposals have assumed that the nodes (users) are cooperative and thus focus on the global topology enhancement. However, a unique and important characteristic of overlay nodes is that, as application-layer agents, they can be selfish with their own interests. To achieve higher Quality-of-Service (QoS) in the streaming application, an overlay node can behave selfishly in the neighborhood information collection stage or in the construction action stage. While the former has recently been widely investigated, the impact of selfishness in the construction action remains unclear.

In this paper, we present the systematic study on the impact of user selfishness during construction action on the streaming quality of overlay multicast, in both tree and mesh based structures. Our investigation considers multiple QoS measures, including stream latency, resolution, and continuity. Our contribution is twofold. First, we discuss the construction-action policy a selfish overlay node chooses to improve its individual multi-metric QoS. Second, we demonstrate according to our model, that the selfishness-aware policy in the construction action is consistent with the cooperative policy required by overlay multicast protocols to improve the QoS of the global multicast session. The implication is that we can leverage the user selfishness in the construction-action stage to form a desirable overlay topology.

1. Introduction

Given the multi-receiver nature of video programs, multicast is a natural vehicle for supporting video streaming applications [1]. It is known that network-layer multicast, or IP multicast, is efficient, but its reach and scope remain very limited due to many practical and political reasons. In the past years, application-layer overlay multicast has emerged as a promising alternative. Overlay multicast realizes routing and data transmission in the application layer, which is remarkably easier to implement and deploy, though less efficient [14,26]. Overlay multicast is also more flexible, because it is decoupled from the network-layer routing, and end systems support much richer semantics in the application layer.

Building an overlay multicast structure with high streaming quality is clearly critical to streaming applications. Existing proposals on overlay multicast structures can be broadly classified into two categories [26], namely, *tree-based* and *mesh-based*. The former follows a well-ordered parent/children relation for data delivery. On the contrary, the latter does not maintain such a fixed relation, but let each node keep a small set of partners to exchange their data availability information, and accordingly fetch expected data.

☆ Some preliminary results of this work was published at INFOCOM'07. In this version, we add the user selfishness model, the up-to-date related works, extensive discussions, as well as more simulations. The work is supported by the National Basic Research Program of China (973 Program) under Grants 2011CB302900 and 2009CB320501, and the National Natural Science Foundation of China (Nos. 60970104 and 60911130511).

* Corresponding author.
E-mail address: tolidan@gmail.com (D. Li).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.06.019

In both tree and mesh overlays, the structure establishment generally consists of two logical stages, that is, *information collection* and *construction action*. In the first stage, overlay nodes learn the information of other nodes and the virtual overlay links, such as the outgoing bandwidth, pair-wise delay, and etc. In the construction action stage, based on the available information, each node selects a long-time parent to receive stream data in tree based overlay multicast, or selects a segment-providing node to fetch a certain segment in mesh based sessions.

A majority of the existing overlay multicast proposals assume that the nodes (users) are cooperative and thus focus on the global topology enhancement [15,16]. However, a unique and important characteristic of overlay nodes is that, as application-layer agents, they can be selfish with their own interests. To achieve better QoS, an overlay node can behave selfishly in the information collection stage or construction action stage. The impact of user selfishness in the information collection stage has recently been largely examined [14,17–20], but that for the construction action stage remains unclear yet. In fact, in many P2P streaming systems such as P2P-Next [27], DistribuStream [28] and VidTorrent [29], users indeed have both incentives and opportunities to modify the source codes to improve their own construction-action policies.

In this paper, we conduct a systematic study on the impact of user selfishness during construction action in both tree and mesh overlays. We consider multiple QoS measures for live streaming applications, including latency, streaming rate, and continuity. Our contribution is twofold. First, we discuss the construction-action policy a selfish overlay node chooses to augment its individual multi-metric QoS. Second, we demonstrate that according to our model, the selfishness-aware policy in the construction action is consistent with the cooperative policy required by overlay multicast protocols, which aims to enhance the streaming quality of the global multicast session. The implication is that we can leverage the user selfishness in the construction-action stage to form a desirable overlay topology.

The rest of this paper is organized as follows. Section 2 introduces the background and related work. Section 3 establishes the model of user selfishness in overlay multicast. Sections 4 and 5 discuss the impacts of user selfishness in construction action on the streaming quality of tree and mesh overlays, respectively. Section 6 makes extensive discussions. Finally, Section 7 concludes the whole paper.

2. Background and related work

Given the obstacles in deploying IP multicast in the global Internet, overlay multicast has emerged as a promising alternative, particularly for live media streaming. In this section, we introduce the background and related works on overlay multicast architectures and its user selfishness.

2.1. Overlay multicast structures

Existing overlay multicast proposals can be broadly classified into two categories according to the data-dissemination structures [26], namely, *tree-based* and *mesh-based*. The former includes NARADA [2], NICE [3], ZIGZAG [4], TAG [5], ALMI [6] and etc. The latter category is represented by DONet/CoolStreaming [7], Chainsaw [8], SplitStream [9], Bullet [10], CoopNet [11], PRO [12], PROMISE [13] and etc. There are also proposals combining the two types of solutions together, such as Pulsar [34], GridMedia [37]. In what follows, we simply describe the two basic overlay structures.

Generally speaking, in a tree-based overlay, each node selects a parent from the neighbors to receive the streaming data, and the parent/children relationships form a multicast tree, as illustrated in Fig. 1. Once the multicast tree is established, the data is continuously propagated along the tree (although there is still stream transmission unit named blocks) and there is no additional control overhead unless the tree is to be updated. In particular, when a node leaves or fails, the tree has to be repaired and its descendants may suffer from data outage.

Unlike the tree-based case, in a mesh-based overlay, there is no fixed parent/children relationship among overlay nodes. Instead, each node selects a number of other nodes as partners. The partner relationships among all nodes form a mesh structure. In a typical mesh, the original stream is divided into a series of segments; partners exchange the segment availability information with each other, and each node fetches a certain segment from a partner that holds the segment. Therefore, it is the data availability that drives the propagation of the stream. An example of a mesh overlay is shown in Fig. 2, with node *a* being the source.

Compared with tree-based overlay, the segment notification and segment requests introduce additional control overhead. Yet mesh-based overlay tolerates node dynamics better. A node observes stream pause only when a seg-

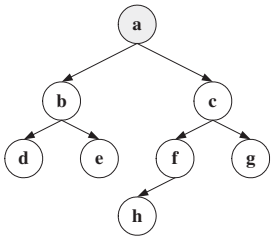


Fig. 1. An example of tree-based overlay multicast. Each receiver has a long-term parent node and several long-term children nodes.

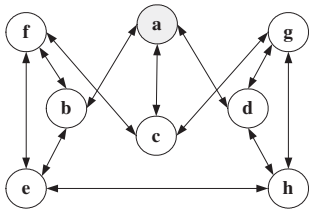


Fig. 2. An example of mesh-based overlay multicast. There is no pre-fixed long-term parent/children relationship between the nodes.

ment is not available in any of its partners before the playback deadline.

Both tree and mesh have shown success in theory and practical deployment [26]. A thorough comparison between them is out of the scope of this paper (Interested readers may find one such comparison in [26]).

Improving the user-experienced streaming quality has constantly been the primary design objective for either tree-based or mesh-based overlay multicast protocols. Many existing protocols have focused on the global topology enhancement and supposed that all the overlay nodes (users) are cooperative. Under this assumption, Sripanidkulchai et al. [15] discuss different parent-selecting methods, like randomly policy, minimum-depth-first policy, and longest-first policy. Bishop et al. [16] disclose the advantage of preemption in favor of nodes with higher priorities by experiments.

2.2. User selfishness in overlay multicast

A key difference between overlay multicast and IP multicast, however, is that the overlay nodes are strategic application-layer agents, which can be selfish with their own interests. Naturally, a node would like to receive as high QoS as possible. There have been many works examining the impact of user selfishness when advertising the private information for tree construction. Mathy et al. [14] demonstrate the negative impact of distance cheating among overlay nodes on the stretch and link stress of the multicast tree. Li et al. [17] further study the impact of this kind of cheating on the stability of multicast tree. Habib et al. [18] point out that the QoS of overlay multicast might be negatively influenced if some overlay nodes are not cooperative to contribute resources, which can also be viewed as the information cheating about outgoing bandwidth or available data. Yuen et al. [19] propose a VCG-based strategyproof algorithm to defend cheating about node throughput. Wang et al. [20] study the cheating about link cost in non-cooperative multicast protocols, and also design distributed payment algorithms against this kind of cheating. Li et al. [30] also find that buffer map cheating in DONet/Coolstreaming will negatively affect the streaming quality of the overlay network, and design service-differentiation based incentive algorithms to defend it. Lately, there is investigation suggesting obtaining the information of other overlay nodes or virtual links by a trustworthy infrastructure, like RandPeer [21].

The user selfishness problem is also widely studied in P2P file sharing application. It is noticed that a large fraction of free riders exist in the file sharing network Gnutella [38]. Thomas et al. [23,32] investigate the P2P topology formed by selfish users, where a peer exploits locality properties to minimize the latency of lookup operations and maintain limited links to other peers. They find that the resulting topologies can be much worse than if peers collaborated. Moreover, the network may never stabilize, even in the absence of churn. In some P2P file sharing systems such as BitTyrant [35] and BitThief [36], it is also found that selfish users can easily make strategic modifications in obtaining better services, while causing the average performance degradation of the whole system. To overcome the free-rider problem in P2P networks, either contribution-based system is designed [39], or micro-monetary infrastructure is introduced [40].

In this paper, we focus on the selfish behaviors of users in overlay multicast when they choose policies to join the streaming-dissemination structure. The most related works to ours are probably [22,31,33]. In [22], the authors impose a payment mechanism to the overlay, in which a node consumes *points* to request for a segment from another node, and earns *points* by sending a segment to another node. Our work does not impose any additional mechanisms but focuses on the selfishness-aware construction. Specifically, we present a systematic study on the impact of user selfishness during construction action in both tree and mesh overlays, and consider a comprehensive set of QoS metrics, including stream latency, resolution, and continuity. In the work of [31], the authors use a repeated-game model to analyze the cooperation behavior of selfish users. The cooperation is based on the tradeoff between a user's short-term desire for quality and long-term desire for the network's continued existence. But in our work, we go one step further by assuming that users primarily care about its current benefit. In [33], the authors propose a mechanism for live streaming swarms to identify free riders quickly and to guarantee that these nodes receive a restricted amount of data such that it is not worthwhile for them to remain in the system. In this paper, we analyze the impact of user selfishness in the construction-action stage on the streaming quality.

Different from most previous work in this area, we find that user selfishness in the construction-action stage can help improve the streaming quality of the overlay multicast sessions. The fundamental reason behind is, if each selfish user is rational and adopts a construction-action policy which is in favor of those nodes with higher upload contributions to it, the nodes with higher capacity will be served better than other nodes; the better positions the stronger nodes lies in the overlay topology, usually the streaming quality of the overlay multicast session will be better. Hence, we can leverage the selfishness in the construction-action stage to form a desirable overlay topology.

3. User selfishness model

Like in existing overlay systems, we assume that each node maintains only a partial view of other nodes, called *neighbors*. When a node joins a multicast session, it obtains the neighbor list from a dedicated node (e.g., the source or a node designated by the source), and this list is dynamically updated to accommodate network changes. The neighboring relationships among all nodes form a *control structure* of the system. We define it neighbor density as the average number of neighbors that each overlay node maintains over the total number of nodes in the control structure. The final data-delivery structure established for an overlay multicast session is called the *overlay structure*.

To facilitate our discussion, we summarize the major system parameters used throughout this paper in Table 1. The actual meaning of each notation will also be revisited later.

Table 1
Major notations in this paper.

Notations	Definitions
T	Playback duration of a block or segment of the stream
E	Stream encoding rate on the source node
A	Set of receiver receiver nodes of the multicast session
e	Average node degree of the overlay control structure
Q	Set of all segments of the stream (for mesh overlay only)
U	Overall QoS of the multicast session
u_i	QoS of node i
u_i^q	Segment QoS of node i for segment q (for mesh overlay only)
d_i	Source-to-end latency on node i
d_i^q	Source-to-end latency of segment q on node i (for mesh overlay only)
f_i^q	Source-to-end latency of segment q on node i's partners that hold it (for mesh-based overlay only)
m_{ij}	Distance from node i to node j
r_i	Received stream rate on node i
r_i^q	Received stream rate of segment q on node i (for mesh overlay only)
v_i	Total incoming bandwidth of node i
o_i	Total outgoing bandwidth of node i
l_i	Past duration of node i in the multicast session
t_i	Expected remaining duration of node i in the multicast session
s_i	Average interval between stream pauses on node i
α	User's weights on stream latency
β	User's weights on stream resolution
γ	User's weights on stream continuity

There are two logical stages towards forming the overlay structure for an overlay multicast session, i.e., information collection stage and construction action stage. In the information collection stage, neighboring nodes exchange necessary control information with each other. In the construction action stage, the nodes issue requests to form the overlay structure from scratch or join an existing overlay. In real systems, the two stages are periodically refreshed. For example, at some time a node i collects the private information of other nodes and makes a construction-action decision to join the overlay; after a while, the information of other nodes can be refreshed and node i makes another round of construction-action decision based on the updated information. For most overlay multicast protocols, the two stages completely compose the overlay structure. Note that we do not consider the data tamper or similar selfish behaviors during data transmission over the overlay topology. From the logical view, the information collection stage and construction action stage are separated with each other.

Note that nodes (users) are selfish in overlay multicast. On one side, each selfish node i can adopt its autonomous policy in the two stages above when establishing the overlay structure, so as to augment its own QoS. In the information collection stage, a selfish node can advertise fake private information to neighboring nodes, in order to find a better position in the overlay structure. Similarly, in the construction action stage, a selfish node can use its own policy to select the upstream or downstream nodes to achieve better QoS. On the other side, the overlay multicast protocol is always trying to improve the overall QoS of the multicast session, requiring the cooperative policies of all receivers. So a simple question is raised: **are the autonomous policies of individual selfish receivers consistent with the cooperative policies required by overlay protocol?**

Let U denote the QoS of the overall multicast session, u_i denote the QoS of a receiver i,IC_i represent the selfish policy of a node i in the information collection stage, and CA_i represent the selfish policy of a node i in the construction action stage. We have the following two equations.

$$U = \frac{\sum_{i \in A} u_i}{|A|}, \tag{1}$$

$$U = F(IC_1, IC_2, \ldots, IC_n, CA_1, CA_2, \ldots, CA_n). \tag{2}$$

Eq. (1) defines the overall QoS of a multicast session as the average QoS of all individual receivers, where A is the set of all receivers of the multicast session. Since the multicast session as well as the overlay structure can be dynamical, throughout this paper, the QoS of a node or the multicast session refers to the QoS at a certain time. Eq. (2) shows that the overall QoS of a multicast session is an output of the autonomous policies of all selfish receivers. It is natural since the user policies in information collection and construction action determine the overlay structure established. However, the QoS of an individual receiver i,u_i, is usually dependent on not only its own policies, but also other receivers' policies.

We observe that though the user policies in information collection stage and construction action stage are usually combined to form the overlay structure, they are orthogonal. It is easy to explain by showing that user policies in the two stages are independent with each other. First, when a receiver i is advertising its private information to other nodes in the information collection stage, it can send out arbitrary information, without knowledge of the construction-action policies of other nodes and itself. Hence, user policies in information collection stage is independent with that of construction action stage. Second, when a receiver i decides to join the overlay structure in the construction stage, it can choose arbitrary nodes as upstream or downstream nodes from its neighbors, no matter what private information of other nodes' is collected. Therefore, user policies in construction action stage is also independent with that of information collection stage.

Therefore, we can separately study the impact of user selfishness in information collection stage and construction action stage. Many works [14,17–20] have disclosed that the autonomous policy of each selfish node i in the information collection stage, IC_i, is inconsistent with the cooperative policy required by overlay protocol. The overall streaming quality will be negatively affected by such kind of selfish behaviors. Consequently, additional defensive policies are necessary to lead selfish nodes towards truthfully advertising their actual private information.

We try to answer the other half of the question, i.e., what is the relationship between the autonomous policy of individual receivers and the cooperative policy required by overlay protocol in the construction action stage? We will investigate this problem by studying the two kinds of policies respectively, and make a comparison between them. The implication of this investigation is significant

for robust overlay multicast protocol design. If the two kinds of policies are inconsistent, an integrated mechanism should be introduced to defend against user selfishness in both information collection and construction action. Otherwise, we only need to focus on selfish user behaviors in information collection stage, and can leverage the user selfishness in construction action stage to lead the overlay topology towards desirable streaming quality.

4. Tree-based overlay multicast

We begin our discussion on the tree-based overlay multicast. We will first give a multi-metric QoS expression, and then discuss the selfish policy of individual nodes and the cooperative policy required by overlay protocol respectively.

4.1. Multi-metric QoS

In a tree overlay, the stream latency and resolution on each node can typically be evaluated as the source-to-end latency and the streaming rate experienced by the node; the stream continuity can be evaluated as the average interval between stream pauses during playback. Hence, we detailize the meaning of u_i. Assume that the playback duration of a block (transmission unit of the stream) and the stream encoding rate at the source node are T and E, respectively. Given d_i, the source-to-end latency on node i, r_i, the received stream rate, l_i, the duration of node i in the session so far, and s_i, the average interval between stream pauses on node i so far, the QoS of node i is shown in Eq. (3).

$$u_i = \frac{\alpha u_1 + \beta u_2 + \gamma u_3}{\alpha + \beta + \gamma}, \tag{3}$$

in which $u_1 = \log\left(1 + \frac{T}{T+d_i}\right)$, $u_2 = \log\left(1 + \frac{r_i}{E}\right)$, $u_3 = \log\left(1 + \frac{s_i}{l_i}\right)$.

Parameters α, β, and γ represent the node's (user's) weights on stream latency, stream resolution, and stream continuity, respectively. The log (.) function is concave, suggesting that the marginal QoS increment diminishes with shorter source-to-end latency, higher stream rate, and longer interval between pauses. It is easy to prove that the resultant value of u_i is within [0,1].

The intuition of this compound QoS expression can be further explained under some boundary cases:

(1) If node i only cares about stream latency, i.e., $\beta = 0$ and $\gamma = 0$, its QoS expression becomes $u_i = \log_2\left(1 + \frac{T}{T+d_i}\right)$. In this case, the QoS measure approaches 1 when the source-to-end latency approaches 0, and its QoS approaches 0 if the source-to-end latency is excessive.

(2) If node i only cares about stream resolution, i.e., $\alpha = 0$ and $\gamma = 0$, its QoS expression becomes $u_i = \log_2(1 + \frac{r_i}{E})$. It approaches 1 when the received stream rate is close to the stream encoding rate on the source node, and its QoS approaches 0 when the received stream rate is too low.

(3) If node i only cares about stream continuity, i.e., $\alpha = 0$ and $\beta = 0$, its QoS expression becomes $u_i = \log_2\left(1 + \frac{s_i}{l_i}\right)$. In this case, the QoS of node i equals 1 when the pause interval is just its duration in the session (i.e., there is no pause), and approaches 0 with very short intervals, i.e., frequent pauses.

4.2. Selfish policy of individual nodes

We first study the selfish construction-action policy adopted by individual nodes. Selfish nodes seek to improve its own QoS. In tree-based overlay multicast, it includes both parent-selection policy and children-acceptance policy.

4.2.1. Parent selection

For node i to select the parent from its neighbors, there are two representative autonomous policies:

Random policy. Randomly selects a neighbor as its parent; denoted as x^R.
QoS-aware policy. Selects the neighbor that can maximize node i's QoS as its parent; denote as x^S.

From node i's selfish perspective, the QoS-aware policy is obviously better since this policy enhances its own QoS. We claim that the QoS-aware policy is also better for improving the overall QoS of the overlay structure, which will be demonstrated by the simulations later in this section.

The key issue then becomes how for the parent-selecting node i to estimate its compound QoS if it were to select neighbor j as its parent.

- *Estimation of the source-to-end latency*. In the information collection stage, neighbor j tells its source-to-end latency, d_j, to its neighbors including node i, and node i measures the distance from neighbor j to itself as m_{ji}. The source-to-end latency on node i if selecting neighbor j as the parent can thus be estimated as $d_i = d_j + m_{ji}$.
- *Estimation of the received stream rate*. Suppose the total incoming bandwidth of node i is v_i. In the information collection, node i learns that the total outgoing bandwidth of neighbor j is o_j, and the received stream rate on neighbor j is r_j. The received stream rate on node i if selecting neighbor j as the parent can be estimated as $r_i = min(r_j, o_j, v_i)$.
- *Estimation of the average interval between stream pauses.* We focus on stream pauses due to ancestor changes, which is the main cause in a tree overlay. There are two reasons for the changes: 1) the ancestor is preempted by another node; and 2) it fails or leaves the session. It is known that, if a node is frequently preempted by other, it will likely be preempted again in future [16]. In other words, a node that has experienced longer average interval between stream pauses is likely to have longer average interval in future as well.

More explicitly, in information collection, node i learns the average interval between stream pauses on neighbor j

as s_j. The average interval between stream pauses on node i if selecting neighbor j as the parent can thus be estimated as $s_i = s_j$.

It is worth noting that the duration of a neighbor is also implicitly included in this expression. For illustration, if node i is to select its parent from two neighbors, j_1 and j_2. Neighbor j_1 stays in the multicast session for 10 s and observes 1 stream pause, while j_2 stays in the multicast session for 4 seconds and observes no stream pause. Then neighbor j_1 will be selected, which potentially enables longer interval between ancestor changes for node i.

Given the estimations and a predicted residual duration for node i as t_i, we express the parent priority of neighbor node j to node i, P_{ji} as,

$$P_{ji} = \frac{\alpha P_1 + \beta P_2 + \gamma P_3}{\alpha + \beta + \gamma}, \tag{4}$$

where $P_1 = \log_2\left(1 + \frac{T}{T + d_j + m_{ji}}\right)$, $P_2 = \log_2\left(1 + \frac{min(r_j, o_j, v_{ij})}{E}\right)$, $P_3 = \log_2\left(1 + \frac{min(s_j, t_{ij})}{t_i}\right)$.

After assigning the parent priorities, the parent-selecting node i will issue a parent request with a expected rate (the minimum of its incoming bandwidth and the stream encoding rate) to the neighbor with the highest parent priority. If this request is rejected due to competition from other nodes or bandwidth constraints, node i will issue another request to the the neighbor with the second highest priority, and so on.

We should emphasize that the decision is online. When node i compares the parent priority of the current parent with that of another neighbor, it should take the additional pause to switch to a new parent into consideration.

4.2.2. Children acceptance

A straightforward policy for node i to accept children depends on its outgoing bandwidth: if it is enough, then all the parent requests can be accepted; otherwise, some requests have to be rejected or some existing children have to be preempted.

On the other hand, a strategic node may choose to reject all the parent requests. Unfortunately, if all nodes choose this policy, the system simply will not work, which in turn hurts the strategic node.

A close investigation suggests that there are four typical children-acceptance policies that a selfish node i might choose:

Negative policy. Not accept any children; denoted as y^N.
Random policy. Randomly accept children within its total outgoing bandwidth; denoted as y^R.
Capacity-aware policy. Prioritize neighbors that might provide better expected QoS to node i if becoming node i's parent in future; denoted as y^P. For example, the neighboring nodes that have higher outgoing capacity, shorter RTT to i and longer living time are more likely to provide better QoS to i if it becomes the parent of i in future.
Contribution-aware policy. Prioritize neighbors that have since forwarded more data to node i; denoted as y^C. For example, in BitTorrent-oriented streaming systems, the tit-for-tat incentive mechanism ([24,25]) favors neighboring nodes from which node i has downloaded more data when i makes the uploading decision.

Since a node receives parent requests from neighbors, and will also select its parent from neighbors, its QoS in future is determined by the children-acceptance policies of its neighbors and its own. The choice of children-acceptance polices of selfish nodes can thus be modeled as a multi-player game. In this game, the players are the overlay nodes, the strategy space is $\{y^N, y^R, y^P, y^C\}$, and the payoff to each node is its expected QoS in future. Please note that we do not count in the forwarding cost of having children nodes here, since we assume the streaming quality is the dominate factor end users care about. Before locating the equilibrium of the multi-player game, we give two definitions from game theory:

Definition 1 (*Dominant strategy*). If s_i is the strategy of player i, s_{-i} is the strategy set of all players except player i, and the payoff of player i is $w_i(s_i, s_{-i}), s_i^\star$ is called the dominant strategy for player i if it satisfies

$$w_i(s_i^\star, s_{-i}) \geqslant w_i(s_i', s_{-i}), \quad \forall s_{-i}, \quad \forall s_i' \neq s_i^\star. \tag{5}$$

Definition 2 (*Dominant strategy equilibrium*). If $s^\star$ is the strategy set of all players, it is called a dominant strategy equilibrium if $s_i^\star$ is the dominant strategy for each player i.

Theorem 1. *The dominant strategy equilibrium in the game of choosing children-acceptance policy is that every node adopts policy y^P.*

Proof. Note that among all parent-requesting neighbors, in future node i will most likely send parent requests to those with higher QoS, since QoS-aware policy is the preferred parent-selection policy. We call such neighbors as *better potential parents* for node i.

For node i, its payoff depends on the children-acceptance policies of its neighbors and its own. The neighbors of node i can adopt different policies from each other. However, for simplicity of node i to estimate its payoff, it can assume that all its neighbors use the same policy out of the four possible ones. In other words, node i assumes the dominant policy of its neighbors and makes the corresponding counter-policy for each one.

Let y_i^k (k = N, R, P, C) denote that node i adopts policy y^k, and y_{-i}^k (k = N, R, P, C) denote that the neighbors of node i adopt the policy y^k, the payoff of node i is shown in Table 2.

The payoff numbers in Table 2 only represent the *relative* value, not the *absolute* QoS of node i in future. However, it is sufficient for us to find the dominant strategy of node i. We now explain Table 2 as follows:

(i) Second column: If the neighbors choose policy y^N, the payoff of node i is obviously 0, indicating that node i cannot find any parent from these neighbors in future.

Table 2
Payoff of node i.

	y^N_{-i}	y^R_{-i}	y^P_{-i}	y^C_{-i}
y^N_i	0	2	2	1
y^R_i	0	2	2	2
y^P_i	0	2	2	4
y^C_i	0	2	2	3

(ii) Third column: If the neighbors choose policy y^R, the payoff of node i is 2, no matter which policy node i adopts. Because the child priority of node i to any neighbor node j (including the better potential parents) in future is random and can be viewed as identical with other neighbors of node j.

(iii) Fourth column: If the neighbors choose policy y^P, the payoff of node i is also 2, no matter which policy node i adopts. This is because, compared with other neighbors of node j, the relative capacity of node i to any neighbor node j (including the better potential parents) in future is also random.

(iv) Last column: If the neighbors choose policy y^C, the payoff of node i depends on its own policy. We will discuss the four cases when node i chooses policy y^N, y^R, y^C, y^P:

Case 1: If node i chooses policy y^N, its payoff is 1. Node i still has the chance to become the child of any neighbor node j in future, if node j has available outgoing bandwidth; but the child priority of node i to node j is the lowest, because it provides no data to node j. Thus, the payoff of node i is more than 0 but less than when it adopts policy y^R.

Case 2: If node i chooses policy y^R, its payoff is 2. Since node i chooses the random policy, its contribution to any neighbor node j (including the better potential parents) is also random. Thus, the child priority of node i to the contribution-aware neighbor j in future can be viewed as random.

Case 3: If node i chooses policy y^C, its payoff is 3. The nodes that have since contributed more to node i will be better potential parents for node i in future (recall the previous discussion about estimating average pause intervals). The QoS of node i in future will be higher than if it adopts policy y^R, because the better potential parents are likely to be served better from a contribution point of view.

Case 4: If node i chooses policy y^P, its payoff is 4. The payoff of node i is higher than if adopting policy y^C, because the prioritized neighbors are just the better potential parents, to which parent requests will be sent first in future.

Therefore, according to Definition 1, y^P is the dominant strategy of node i. According to Definition 2, the dominant strategy equilibrium in the game of choosing children-acceptance policy is that all nodes choose policy y^P. □

The next issue is how for node i to assign the child priorities to the parent-requesting nodes under policy y^P. The expected QoS of node i if selecting some parent-requesting node j as the parent in future is estimated by two factors, that is, the QoS of node i if node j accepts it as a child in future and the probability of node j accepting node i as a child in future.

For estimating the QoS of node i if neighbor j accepts it as a child in future, node i does not have the information of source-to-end latency of node j, the received stream rate of node j, nor the average interval between stream pauses on node j when node j becomes node i's parent in future. The information available for node i to estimate includes the distance from node j to node i, the total incoming bandwidth and outgoing bandwidth of node j, and the passed duration of node j in the multicast session.

On estimating the probability of node j accepting node i as a child in future, the nodes that have higher outgoing bandwidth and longer duration in the multicast session are prioritized. We can just use the product of the two parameters since only the relative value of the probability is useful.

Therefore, the child priority of node j to node i, C_{ji}, is assigned as Eq. (6).

$$C_{ji} = \frac{\alpha C_1 + \beta C_2 + \gamma C_3}{\alpha + \beta + \gamma} * H, \tag{6}$$

where $C_1 = \log_2\left(1 + \frac{T}{T + m_{ji}}\right)$, $C_2 = \log_2\left(1 + \frac{min(E, v_j, o_j, v_i)}{E}\right)$, $C_3 = \log_2\left(1 + \frac{min(l_j, t_i)}{t_i}\right)$, and $H = o_j * l_j$.

Eq. (6) suggests that the overlay nodes with higher outgoing bandwidth, higher incoming bandwidth and longer staying time, and those closer to other nodes in the overlay network will be assigned with higher child priorities in construction action. We find that prioritizing the nodes with higher capacities also helps improving the overall QoS, which will be demonstrated by the simulations in the following subsection.

4.3. Cooperative policy required by overlay protocol

In this subsection, we use simulation to study the cooperative policy required by tree-based overlay protocols. The overlay protocol always tries to maximize the overall QoS of the multicast session. We set the policy space the same as the one used by individual nodes, i.e., random policy (x^R) and QoS-aware policy (x_S) for parent selection, and negative policy (y^N), random policy (y^R), capacity-aware policy (y^P) and contribution-aware policy (y^C) for children acceptance.

We use the GT-ITM toolkit to generate network-layer topologies. Unless otherwise specified, the following default settings are used in the simulation. There are 2000 routers in the network-layer topology and the link distances between connected routers are uniformly distributed in [10 ms, 500 ms]. The overlay nodes as well as the source node are attached to different routers randomly selected from the 2000 routers. The outgoing bandwidth of each node is within [0 kbps, 8000 kbps], and the incoming bandwidth of each node is within [500 kbps, 10000 kbps].

The playback duration of a block is 5 s, and the stream encoding rate at the source node is 1 Mbps. To mitigate randomness, we generate 10 different network-layer topologies for each simulation on the above settings and the results presented below are their average.

To evaluate the overall QoS of the multicast session under different network situations, we test various system configurations. For each configuration, the receiver node with lower sequence number joins earlier in the multicast session and leaves later from the multicast session, which is an important assumption in this paper stated previously.

We compare the overall QoS of the multicast session under the combination of different parent-selection policies and children-acceptance policies, that is, $x^R + y^R$, $x^S + y^R$, $x^R + y^P$, $x^S + y^P$, $x^R + y^C$, and $x^S + y^C$. The children-acceptance policy y^N, i.e., not accepting any children, is not considered here because, as mentioned, it does not work for practical systems. We first fix the total number of receiver nodes, n, as 200, and vary the average node degree of the overlay control structure, e, from 10% to 90%. We then fix the average node degree as 20%, and vary the total number of receiver nodes from 100 to 700.

For QoS evaluation, we put different weights on stream latency, resolution, and continuity. For a combination of α, β, and γ as 0.6, 0.2 and 0.2, we plot the overall QoS of the multicast session for the two network settings above in Fig. 3. For combination of 0.2, 0.6 and 0.2, the result is shown in Fig. 4. And finally, for combination of 0.2, 0.2, and 0.6, the result is in Fig. 5.

From the simulation results, we have the following observation:

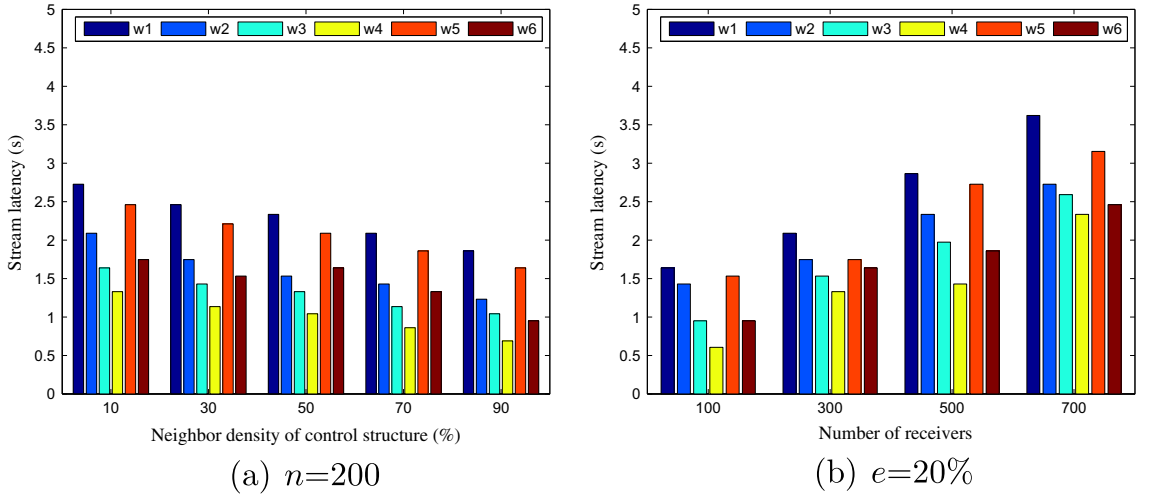


Fig. 3. Overall QoS of the tree-based overlay multicast session. $w1 = x^R + y^R$, $w2 = x^S + y^R$, $w3 = x^R + y^P$, $w4 = x^S + y^P$, $w5 = x^R + y^C$, and $w6 = x^S + y^C$. $\alpha = 0.6$, $\beta = 0.2$, and $\gamma = 0.2$.

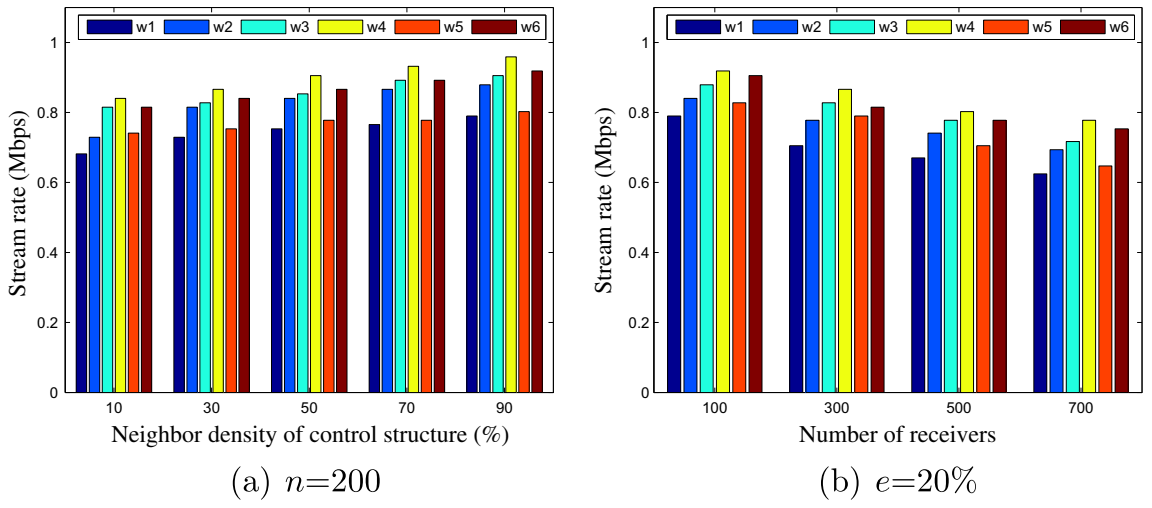


Fig. 4. Overall QoS of the tree-based overlay multicast session. $w1 = x^R + y^R$, $w2 = x^S + y^R$, $w3 = x^R + y^P$, $w4 = x^S + y^P$, $w5 = x^R + y^C$, and $w6 = x^S + y^C$. $\alpha = 0.2$, $\beta = 0.6$, and $\gamma = 0.2$.

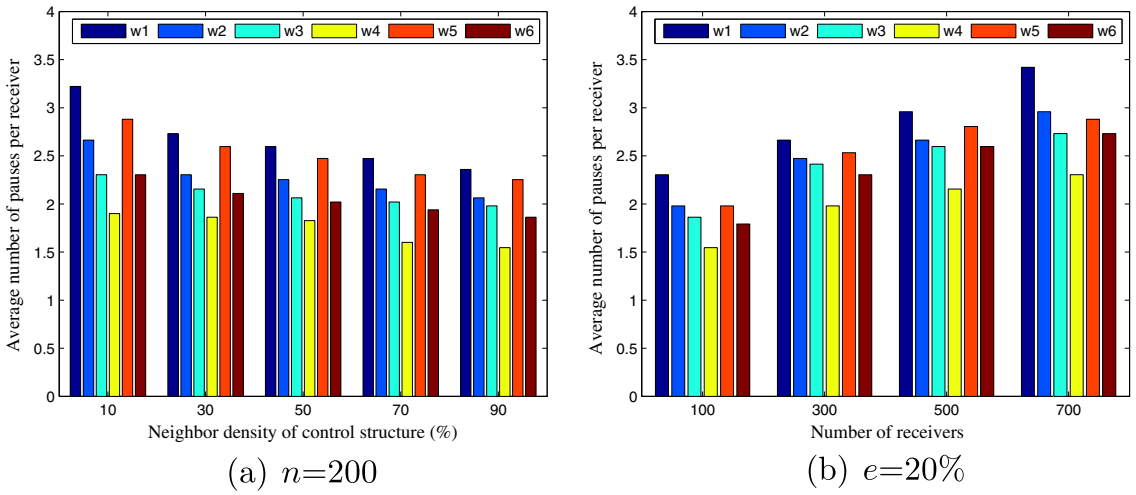


Fig. 5. Overall QoS of the tree-based overlay multicast session. $w1 = x^R + y^R$, $w2 = x^S + y^R$, $w3 = x^R + y^P$, $w4 = x^S + y^P$, $w5 = x^R + y^C$, and $w6 = x^S + y^C$. $\alpha = 0.2$, $\beta = 0.2$, and $\gamma = 0.6$.

- Given the same children-acceptance policy, the overall QoS under QoS-aware parent-selection policy is generally better than that under random parent-selection policy. This is because enhancing the QoS of an individual selfish node i potentially enables better QoS for other nodes that might select i as parent.
- Given the same parent-selection policy, the overall QoS under capacity-aware children-acceptance policy is the best among the three children-acceptance policies. Intuitively, a higher-capacity node i has more opportunities to be the parent of other nodes, and the improvement of QoS on node i will also help improving the QoS of nodes that select i as parent.

Therefore, the cooperative policy required by tree-based overlay protocol is QoS-aware parent-selection policy and capacity-aware children-acceptance policy. It is exactly the same as the selfish construction-action policy used by individual nodes. In other words, user selfishness in construction action will lead to a desirable tree overlay.

5. Mesh-based overlay multicast

We next discuss the mesh-based overlay construction. There are again two stages toward establishing the overlay structure. In the information collection stage, each node learns the information of partners, including the segment availability, the outgoing bandwidth, the distances from partners to it, and etc. In the construction action stage, each node sends a segment request to the partner selected as the segment-providing node for a certain segment, and responds to the received segment requests with the corresponding segments if its outgoing bandwidth allows.

5.1. Multiple-metric QoS

The multi-metric QoS for a mesh overlay can be expressed similarly to that in the tree case. The stream latency and resolution are respectively evaluated as the average source-to-end latency and the average streaming rate across of all the segments. The stream continuity is still evaluated as the average interval between stream pauses, but a pause is caused by the segment scarcity in partners, for there are no predefined ancestors as in the tree case.

Let T, E, and Q be the playback duration of a segment, the stream encoding rate at the source node, and the segment set of the stream, respectively. Let d_i^q be the source-to-end latency of segment q on node i, r_i^q be the streaming rate of segment q,l_i be the duration of node i in the session so far, and s_i be the average pause interval so far. Like in the tree case, we express the segment QoS of node i for segment q, u_i^q as

$$u_i^q = \frac{\alpha u_1^q + \beta u_2^q}{\alpha + \beta}, \tag{7}$$

where $u_1^q = \log_2\left(1 + \frac{T}{T + d_i^q}\right)$, $u_2^q = \log_2\left(1 + \frac{r_i^q}{E}\right)$.

And the QoS of node i, u_i, is expressed as

$$u_i = \frac{\alpha u_1 + \beta u_2 + \gamma u_3}{\alpha + \beta + \gamma}, \tag{8}$$

where $u_1 = \frac{\sum_{q \in Q} \log_2\left(1 + \frac{T}{d_i^q + T}\right)}{|Q|}$, $u_2 = \frac{\sum_{q \in Q} \log_2\left(1 + \frac{r_i^q}{E}\right)}{|Q|}$, $u_3 = \log_2\left(1 + \frac{s_i}{l_i}\right)$.

5.2. Selfish policy of individual nodes

As in the tree-based overlay protocol, we start from analyzing the selfish policy of individual nodes in this subsection, followed by demonstrating the cooperative policy required by overlay protocol in the next subsection. For mesh-based overlay multicast, the construction-action policy is comprised of both segment-request policy and segment-response policy.

5.2.1. Segment request

In the mesh, each node exchanges segment availability information with partners, and finds which partners hold certain segments. If a segment is held by multiple partners,

a segment-providing node is selected among these partners for the segment. There are also two representative policies for node i to select the segment-providing node for segment q.

Random policy. Randomly selects a partner that holds segment q as the segment-providing node; denoted as $x^{R'}$.

QoS-aware policy. Selects the partner that can maximize the QoS of node i for segment q as the segment-providing node; denoted as $x^{S'}$.

Because of the same reason in tree-based overlay multicast, QoS-aware policy is the preferable choice of selfish overlay nodes. We claim that the QoS-aware policy is also better for improving the overall QoS of the multicast session, which will be demonstrated by the simulations later in this section.

The remaining issue here is how for the segment-requesting node i to estimate its segment QoS for segment q if selecting partner j as the segment-providing node.

Since the requesting segment q is wholly available in the segment-providing node, when node i decides to request segment q, the source-to-end latencies of segment q on the partners that hold it at that time are identical, denoted as f_i^q.

In the information collection, node i has measured the distance from partner j to it, m_{ij}, and learned the outgoing bandwidth of partner j, o_j. Based on these information, the segment-request priority of node j to node i for segment q, R_{ji}^q, is assigned as Eq. (9).

$$R_{ji}^q = \frac{\alpha R_1^q + \beta R_2}{\alpha + \beta}, \tag{9}$$

where $R_1^q = \log_2\left(1 + \frac{T}{T + m_{ji} + f_i^q}\right)$, $R_2 = \log_2\left(1 + \frac{min(E, o_j, v_i)}{E}\right)$.

The segment-requesting node i will issue a segment request with a required rate (the minimum of its incoming bandwidth and the stream encoding rate) to the partner of the highest priority for the segment. If the request is rejected, node i will contact the partner with the second highest priority, and so on.

5.2.2. Segment response

When a node receives multiple segment requests from partners, it needs a segment-response policy to accept the requests within its total outgoing bandwidth. Like in tree, there are four representative segment-response policies for selfish node i.

Negative policy. Not respond to any segment request; denoted as $y^{N'}$.

Random policy. Randomly respond to segment requests within its total outgoing bandwidth; denoted as $y^{R'}$.

Capacity-aware policy. Prioritize partners that might provide higher expected segment QoS to node i if becoming node i's segment-providing nodes in future; denoted as $y^{P'}$.

Contribution-aware policy. Prioritize partners that have ever forwarded more segments to node i; denoted as $y^{C'}$.

We can again apply the multi-player game to examine the choice of segment-response policy. In this game, the players are the overlay nodes, the strategy space is $\{y^{N'}, y^{R'}, y^{P'}, y^{C'}\}$, and the payoff of each node is its expected segment QoS in future.

Theorem 2. *The dominant strategy equilibrium in the game of choosing segment-response policy in mesh-based overlay multicast is that each node adopts the policy* $y^{P'}$.

The proof of Theorem 3 is similar to that for Theorem 2. We now focus on practically how for node i to assign the segment-response priority to a segment-requesting node j. To predict the future segment QoS of node i if choosing node j as the segment-providing node, node i needs to learn the distance from node j to it and the total outgoing bandwidth of node j in the information collection stage. In addition, node i needs to estimate the probability of its segment request being accepted by node j in future. The nodes that have higher outgoing bandwidths and stay longer in the multicast session will likely accept the segment request of node i with higher probability in future (as in the tree base, we just multiply the two parameters). Therefore, the segment-response priority of node j to node i, G_{ji}, can be assigned as Eq. (10).

$$G_{ji} = \frac{\alpha G_1 + \beta G_2}{\alpha + \beta} * H, \tag{10}$$

where $G_1 = \log_2\left(1 + \frac{T}{T + m_{ji}}\right)$, $G_2 = \log_2\left(1 + \frac{min(E, o_j, v_i)}{E}\right)$, $H = o_j * l_j$.

Eq. (10) suggests that the overlay nodes with higher outgoing bandwidth and longer duration, and those closer to other overlay nodes are assigned with higher segment-response priorities in the mesh construction action. Our following simulation results also show that prioritizing these nodes enhances the overall QoS of the overlay structure.

5.3. Cooperative policy required by overlay protocol

Again we use simulation to study the cooperative policy required by mesh-based overlay protocols. We also set the policy space the same as the one used by individual nodes, i.e., random policy ($x^{R'}$) and QoS-aware policy ($x_{S'}$) for segment request, and negative policy ($y^{N'}$), random policy ($y^{R'}$), capacity-aware policy ($y^{P'}$) and contribution-aware policy ($y^{C'}$) for segment response.

The default simulation settings for mesh are similar to those for tree. We assume that the stream is divided into 600 segments, each segment is of 5 s, and the number of partners each node maintains is 4, as recommended in [7].

We again compare the overall QoS of the multicast session under different combinations of segment-request policies and segment-response policies, that is, $x^{R'} + y^{R'}$, $x^{S'} + y^{R'}$, $x^{R'} + y^{P'}$, $x^{S'} + y^{P'}$, $x^{R'} + y^{C'}$, and $x^{S'} + y^{C'}$. Likewise, the negative segment-response policy, $y^{N'}$, is not considered. To evaluate the overall QoS under different network configurations, we first fix the total number of receiver nodes, n, to 200, and vary the average node degree of the overlay control structure from 10% to 90%; we then fix the average

node degree to 20%, and change the total number of receiver nodes from 100 to 700.

As in the simulation of tree overlay, we also consider different weights combinations on stream latency, resolution, and continuity. The corresponding results for (α,β,γ) of (0.6,0.2,0.2), (0.2,0.6,0.2), and (0.2,0.2,0.6) are shown in Figs. 6–8, respectively.

From all these figures, we have the following observations, which are consistent with those made in the tree case:

- Given the same segment-response policy, the overall QoS under QoS-aware segment-request policy is always better than under random segment-request policy. Because the QoS-aware policy not only improves the segment QoS of the requesting node itself, but will also help improve the segment QoS of the nodes that select it as segment-providing node.
- Given the same segment-request policy, the overall QoS under capacity-aware segment-response policy is the best among the three segment-response policies. This is because a higher-capacity node i has more opportunities to be a segment-providing node, and augmenting its QoS thus helps with improving the QoS of nodes that select i as the segment-providing node.

We can see that the cooperative policy required by mesh-based overlay protocol is also QoS-aware segment-request policy and capacity-aware segment-response policy, which is consistent with the selfish construction-action policy

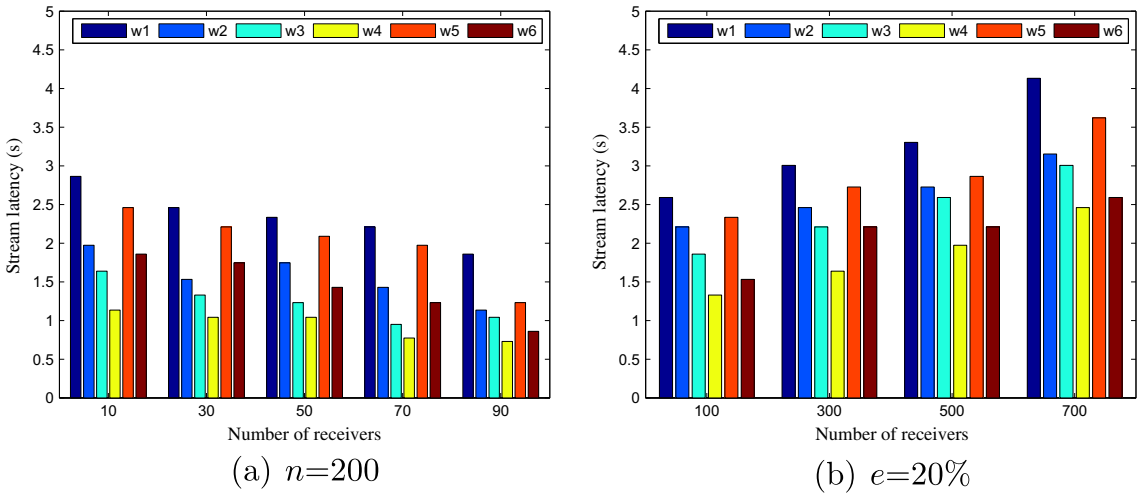


Fig. 6. Overall QoS of the mesh-based overlay multicast session. $w1 = x^{R'} + y^{R'}$, $w2 = x^{S'} + y^{R'}$, $w3 = x^{R'} + y^{P'}$, $w4 = x^{S'} + y^{P'}$, $w5 = x^{R'} + y^{C'}$, and $w6 = x^{S'} + y^{C'}$. $\alpha = 0.6$, $\beta = 0.2$, and $\gamma = 0.2$.

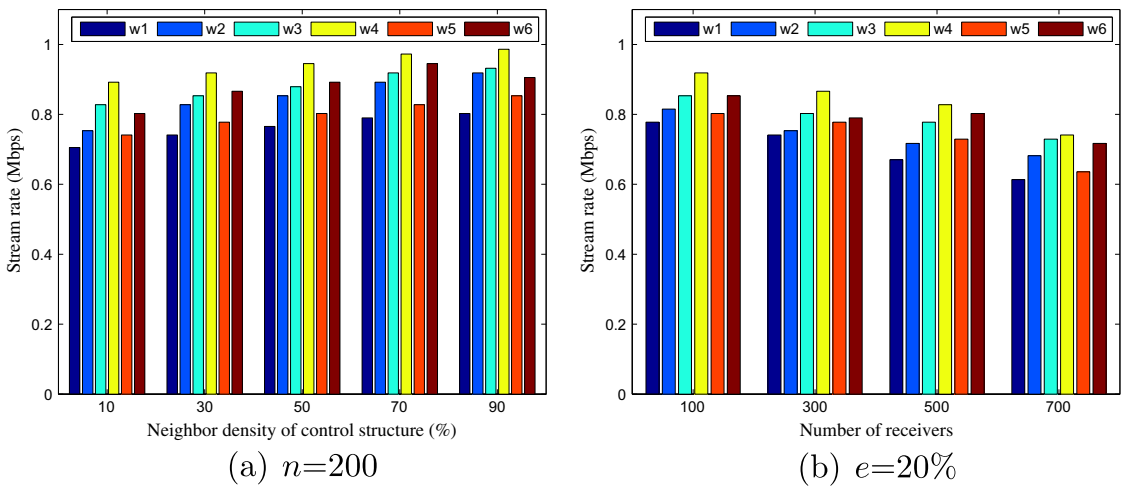


Fig. 7. Overall QoS of the mesh-based overlay multicast session. $w1 = x^{R'} + y^{R'}$, $w2 = x^{S'} + y^{R'}$, $w3 = x^{R'} + y^{P'}$, $w4 = x^{S'} + y^{P'}$, $w5 = x^{R'} + y^{C'}$, and $w6 = x^{S'} + y^{C'}$. $\alpha = 0.2$, $\beta = 0.6$, and $\gamma = 0.2$.

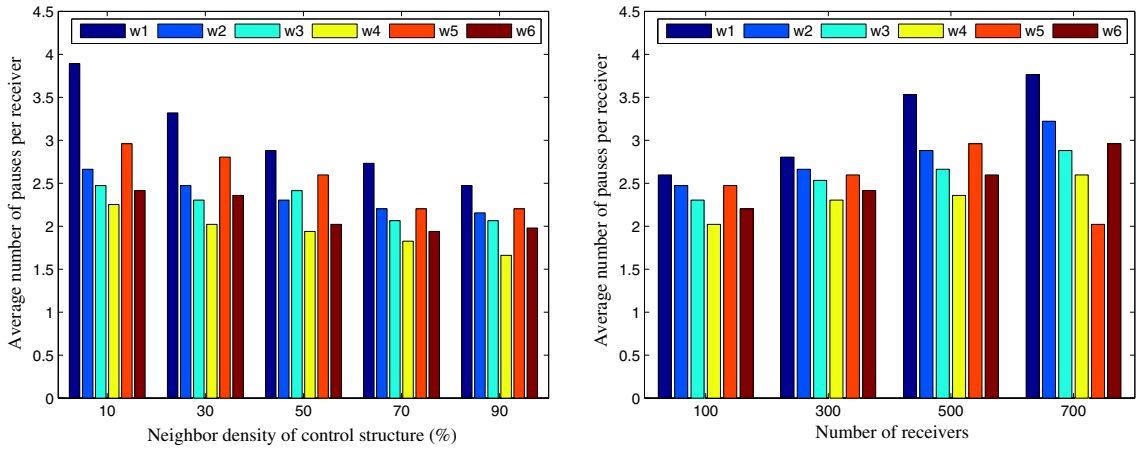


Fig. 8. Overall QoS of the mesh-based overlay multicast session. $w1 = x^{R'} + y^{R'}$, $w2 = x^{S'} + y^{R'}$, $w3 = x^{R'} + y^{P'}$, $w4 = x^{S'} + y^{P'}$, $w5 = x^{R'} + y^{C'}$, and $w6 = x^{S'} + y^{C'}$. $\alpha = 0.2$, $\beta = 0.2$, and $\gamma = 0.6$.

used by individual nodes. In other words, user selfishness in construction action can help improve the streaming quality of the mesh overlay.

6. Discussions

In this section we make more discussions on our work.

First, we have answered the question proposed in Section 3. In either tree or mesh based overlay multicast, the autonomous policy of selfish individual users is consistent with the cooperative policy required by overlay protocols. Hence, it releases us from designing defensive mechanism against user selfishness in construction action. Instead, for robust and trustworthy overlay multicast design, we should focus on tackling with user selfishness in information collection stage, since previous studies have shown that the fake information advertised by selfish users brings negative impact on the global multicast session.

Second, when proving the autonomous policies an individual user chooses in the construction action stage, we assume that all its neighbors adopt the same construction-action policy for simplicity. In fact, this assumption does not lose any generality. If considering that neighbors adopt different construction-action policies, the payoff table will be much more complex. But we can divide the neighbors into several groups, each group with identical policy. And the conclusion still holds.

Third, we do not consider collusion among individual users in this paper. In real systems, two or more users can collude by knowing the construction-action policies of each other. We illustrate a simple example for this kind of collusion. Given there are n receivers, $n-1$ receivers can collude by not accepting the parent request or segment-sending request of the left one receiver. The $n-1$ receivers can benefit from this collusion since some of them may get closer to the source, or get higher streaming rate. Although we do not want to thoroughly solve collusion problem in this paper, we think one possible approach is to introduce a third-party authority, and some *punishment* mechanism is used.

7. Conclusion

In this paper, we investigated the impact of user selfishness in construction action on the streaming quality of overlay multicast sessions, for both tree and mesh overlays. Our study considers multiple QoS measures for live streaming applications, including stream latency, stream resolution, and stream continuity. We discuss the construction-action policy a selfish overlay node chooses to enhance its individual multi-metric QoS. The analytical and simulation results reveal that the selfish policies chosen by individual nodes are consistent with improving the overall QoS of the multicast session. Therefore, given that users advertise true private information in the information collection stage, the user selfishness in the construction action stage can lead to a desirable overlay multicast topology, and no additional defensive mechanism is required for user selfishness in this stage.

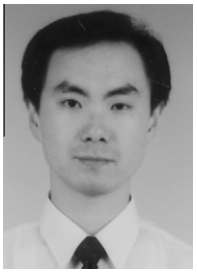
References

[1] J. Liu, B. Li, Y.Q. Zhang, Adaptive video multicast over the internet, IEEE Multimedia 10 (1) (2003) 22–31.
[2] Y.H. Chu, S.G. Rao, H. Zhang, A case for end system multicast, in: Proceedings of ACM SIGMETRICS'00, Santa Clara, CA, USA, 2000.
[3] S. Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable Application Layer Multicast, in: Proceedings of ACM SIGCOMM'02, Pittsburgh, PA, USA, 2002.
[4] D. Tran, K. Hua, T. Do, Zigzag: an efficient peer-to-peer scheme for media streaming, in: Proceedings of IEEE INFOCOM'03, San Franciso, CA, USA, March/April 2003.
[5] M. Kwon, S. Fahmy, Topology-aware overlay networks for group communication, in: Proceedings of NOSSDAV'02, Miami Beach, FL, USA, 2002.

[6] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel, ALMI: an application level multicast infrastructure, in: Proceedings of USITS'01, San Francisco, California, USA, March 2001.
[7] X. Zhang, J. Liu, B. Li, T.P. Yum, CoolStreaming/DONet: a data-driven overlay network for efficient live media streaming, in: Proceedings of IEEE INFOCOM'05, Miami, FL, March 2005.
[8] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A.E. Mohr, Chainsaw: eliminating trees from overlay multicast, in: Proceedings of IPTPS'05, Ithaca, NY, USA, February 2005.
[9] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, A. Singh, SplitStream: high-bandwidth content distribution in cooperative environments, in: Proceedings of ACM SOSP'03, Bolton Landing, NY, October 2003.
[10] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In Proceedings of ACM SOSP'03, Bolton Landing, NY, October 2003.
[11] V. Padmanabhan, H. Wang, P. Chou, K. Sripanidkulchai, Distributing streaming media content using cooperative networking, in: Proceedings of NOSSDAV'02, Miami Beach, FL, USA, May 2002.
[12] R. Rejaie, S. Stafford, A framework for architecting peer-to-peer receiver-driven overlays, in: Proceedings of NOSSDAV'04, Cork, Ireland, June 2004.
[13] M. Hefeeda, A. Habib, B. Botev, D. Xu, B. Bhargava, PROMISE: peer-to-peer media streaming using CollectCast, in: Proceedings of ACM Multimedia'03, Berkeley, CA, Nov 2003.
[14] L. Mathy, N. Blundell, Impact of simple cheating in application-level multicast, in: Proceedings of IEEE INFOCOM'04, Hong Kong, China, March 2004.
[15] K. Sripanidkulchai, A. Ganjam, B. Maggs, H. Zhang, The feasibility of supporting large-scale live stream applications with dynamic application end-points, in: Proceedings of ACM SIGCOMM'04, Portland, Oregon, USA, August/Septamber 2004.
[16] M. Bishop, S. Rao, K. Sripanidkulchai, Considering priority in overlay multicast protocols under heterogeneous environments, in: Proceedings of IEEE INFOCOM'06, Barcelona, Spain, April 2006.
[17] D. Li, Y. Cui, K. Xu, J. Wu, Impact of receiver cheating on the stability of ALM tree, in: Proceedings of IEEE GLOBECOM'05, St. Louis, Missouri, USA, November/December 2005.
[18] A. Habib, J. Chuang, Incentive mechanism for peer-to-peer media streaming, in: Proceedings of IWQOS'04, Montreal, Canada, June 2004.
[19] S. Yuen, B. Li, Strategyproof Mechanisms for dynamic multicast tree formation in overlay networks, in: Proceedings of IEEE INFOCOM'05, Miami, Florida, USA, March 2005.
[20] W. Wang, X. Li, Z. Suny, Y. Wang, Design multicast protocols for non-cooperative networks, in: Proceedings of IEEE INFOCOM'05, Miami, Florida, USA, March 2005.
[21] J. Liang, K. Nahrstedt, RandPeer: membership management for QoS sensitive peer-to-peer applications, in: Proceedings of IEEE INFOCOM'06, Barcelona, Spain, April 2006.
[22] G. Tan, S.A. Jarvis, A Payment-based incentive and service differentiation mechanism for peer-to-peer streaming broadcast, in: Proceedings of IWQOS'06, Yale University, New Haven, CT, USA, June 2006.
[23] T. Moscibroda, S. Schmid, R. Wattenhofer, On the topologies formed by selfish peers, in: Proceedings of IPTPS'06, Santa Barbara, USA, Feb 2006.
[24] D. Qiu, R. Srikant, Modeling and performance analysis of BitTorrent-like peer-to-peer networks, in: Proceedings of ACM SIGCOMM'04, Portland, OR, USA, August 2004.
[25] A.R. Bharambe, C. Herley, Analyzing and Improving BitTorrent Performance, Microsoft Research Report, No. MSR-TR-2005-03, Microsoft Research, 2005.
[26] J. Liu, S.G. Rao, B. Li, H. Zhang, Opportunities and challenges of peer-to-peer Internet video broadcast, Proceedings of the IEEE, Special Issue on Recent Advances in Distributed Multimedia Communications (2007).
[27] Shaping the Next Generation of Internet TV. <http://www.p2p-next.org/>.
[28] Distriu Stream. <http://distribustream.org/>.
[29] Viral communicationis. <http://viral.media.mit.edu/index.php?page=vidtorrent>.
[30] D. Li, J. Wu, Y. Cui, Defending against buffer map cheating in DONet-like P2P streaming, IEEE Transactions on Multimedia 11 (3) (2009) 535–542.
[31] M. Afergan, R. Sami, Repeated-game modeling of multicast overlays, in: Proceedings of IEEE INFOCOM'06, Barcelona, Spain, April 2006.
[32] T. Moscibroda, S. Schmid, R. Wattenhofer, Topological implications of selfish neighbor selection in unstructured peer-to-peer networks, Algorithmica 57 (4) (2010).
[33] T. Locher, R. Meier, R. Wattenhofer, S. Schmid, Robust live media streaming in swarms, in Proceedings of NOSSDAV'09, New York, NY, 2009.
[34] T. Locher, R. Meier, S. Schmid, R. Wattenhofer, Push-to-pull peer-to-peer live streaming, Springer Lecture Notes in Computer Science 4731 (2007) 388–402.
[35] M. Piatek, T. Isdal, T. Anderson, Do incentives build robustness in BitTorrent? in: Proceedings of NSDI'07, 2007.
[36] T. Locher, P. Moor, S. Schmid1, Free riding in BitTorrent is cheap, in: Proceedings of HotNets'06, 2006.
[37] M. ZHANG, J. LUO, L. ZHAO, S. YANG, A peer-to-peer network for live media streaming – using a push-pull approach, In Proceedings of ACM Multimedia (2005).
[38] E. Adar, B. Huberman, Free riding on Gnutella, First Monday 5 (2000) 10.
[39] Kazza. <http://www.kazaa.com/>.
[40] F. Garcia, J. Hoepman, Off-line karma: a decentralized currency for peer-to-peer and grid applications, in: Proceedings of Third Applied Cryptography and Network Security (ACNS).

Dan Li is currently an assistant professor in Tsinghua University. His research interests include Internet architecture and protocol design, P2P streaming, data center networks and green computing. In these areas, he has published more than 20 technical papers in referred conferences and journals. He received Ph.D degree from Tsinghua University in Dec 2007.

Jiangping Wu is now a full professor in computer science department of Tsinghua University. He received master and doctor degrees from in computer science Tsinghua University. In the research areas of the network architecture, high performance routing and switching, protocol testing and formal methods, he has published more than 200 technical papers in the academic journals and proceedings of international conferences. He is a senior member of IEEE.



Yong Cui, Ph.D Associate Professor in Tsinghua University, Council Member in China Communication Standards Association. He directed several international and national R&D projects. He had the honor to win National Science and Technology Progress Award (Second Class) in 2005 and Influential Invention Award of China Information Industry in 2004. Having published more than 80 papers, he also holds more than 20 patents. He is one of the authors in RFC 5747 and RFC 5565 on IPv6 transition technologies. His major research interests include computer network architecture and mobile wireless computing.

Jiangchuan Liu (S'01-M'03-SM'08) received the BE degree (cum laude) from Tsinghua University, Beijing, China, in 1999, and the PhD degree from The Hong Kong University of Science and Technology in 2003, both in computer science. He is a recipient of Microsoft Research Fellowship (2000), Hong Kong Young Scientist Award (2003), and Canada NSERC DAS Award (2009). He is a co-recipient of the Best Student Paper Award of IWQoS'2008, the Best Paper Award (2009) of IEEE Com Soc Multimedia Communications Technical Committee, and Canada BC Net Broadband Challenge Winner Award 2009. He is currently an Associate Professor in the School of Computing Science, Simon Fraser University, British Columbia, Canada. His research interests include multimedia systems and networks, wireless ad hoc and sensor networks, and peer-to-peer and overlay networks. He is a Senior Member of IEEE and a member of Sigma Xi. He is an Associate Editor of IEEE Transactions on Multimedia, an editor of IEEE Communications Surveys and Tutorials, and an Area Editor of Computer Communications. He is TPC Vice Chair for Information Systems of IEEE INFOCOM'2011.

Ke Xu received the B.S., M.S. and Ph.D degrees in computer science from Tsinghua University, China in 1996, 1998 and 2001 respectively. Currently he is Professor in the department of computer science of Tsinghua University. His research interests include next generation Internet, switch and router architecture, P2P and overlay network. He is a senior member of IEEE and member of ACM.