Segment Tree Based Control Plane Protocol for Peer-to-Peer Streaming Service Discovery

Changxi Zheng[a], Guobin Shen[b] and Shipeng Li[b]

[a]Department of Computer Science and Technology,
Shanghai Jiaotong University,
Shanghai, P.R.China;
[b]Microsoft Research Asia, Beijing, P.R.China

ABSTRACT

As the intense academic interest in video streaming over peer-to-peer(P2P) network, more and more streaming protocols have been proposed to address different problems on this field, such as QoS, load balancing, transmission reliability, bandwidth efficiency, etc. However, to the best of our knowledge, an important component of any practical P2P streaming system, the streaming service discovery, is rarely granted particular considerations, which is to discover potential peers from which a newcomer could receive the requested stream. In this paper, we are inspired from a special data structure, named Segment Tree(ST), and propose a protocol to address this problem specifically. Furthermore, we fully decouple the control plane and data plane on video streaming, and hence provide more flexibilities in designing protocols on both of them.

Keywords: Peer-to-Peer Network, Video Streaming, Streaming Service Discovery, Segment Tree

1. INTRODUCTION

As the kinds of multimedia streaming applications become more and more popular, the research community pay more attention to this topic, and a deluge of streaming protocols is proposed.[1–4] In most of the P2P streaming protocols, a streaming session is logically composed of two planes: *data plane* and *control plane*. The data plane is responsible for transmitting actual payload packets while the control plane is responsible for the coordination among peers.

On the data plane, peers typically not only receive packets but also relay some of the received data to their downstream users. Due to the memory limitation, they usually cache only a portion of the latest received data and forward it to others continuously. The cached content on a peer is updated all the time along with the progress of the streaming session. Effectively, the cached content is from a sliding window on the original stream. The oldest received data is replaced by the newly coming one. This process of caching and relaying the content continuously was shown to scale well in terms of server as well as network loads.[1] On the other hand, the work of control plane usually consists of connection setup, teardown, topology maintenance and recovery, rate allocation and optimization.[4,5] Nevertheless, the first task on it is to discover potential peers from which a newcomer could receive the requested stream, which we called streaming service discovery(SSD) problem. In this paper, we mainly focus on the SSD problem and propose a protocol to address it.

The rest of the paper is organized as follows. Section 2 reviews the traditional aggregation and publish/subscrible service used in file sharing and presents the motivation and challenges for the streaming protocols. Section 3 introduces the basic organization and overlay topology in our protocol. We present the control plane protocol in details in section 4. In section 5, we present the evaluation results for our protocol, and compare the performance under different configurations. Finally, in section 6, we conclude the whole paper.

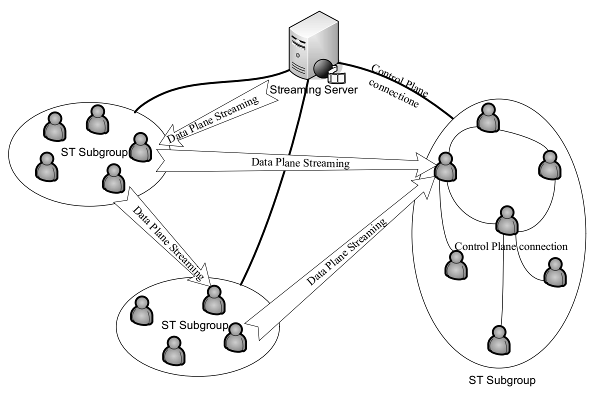

Figure 1. Macroscopical structure of our protocol

2. MOTIVATION AND CHALLENGES

For better user experience, the initial delay for the SSD should be shortened as much as possible while the resulting candidate senders be as accurate as possible. With little initial knowledge of others, a newcomer has to query somewhere, say aggregation points, to discover potential senders, and publish its state information. Essentially, this is a traditional aggregation and publish/subscribe service, for which the research community has proposed DHT-based approach[6] and hierarchical tree-based approach.[7] However, as observed in `Mercury`,[8] the traditional DHT-based approach is difficult for range query that is the most crucial demand need to be satisfied here, and from the perspective of video streaming, some particular barriers distinguish the SSD problem from the traditional ones for the sake of the following new challenges:

- *Peer Status:* Some of the status information, especially the content cached in the buffer, keeps on changing continuously. What's more, different data plane protocols may require a variety of peer status information, but we cannot make any assumptions for that(to ensure the flexibility in the data plane protocol design). Therefore an appropriate yet flexible way to represent the peer status is highly desired.
- *Efficiency of Updating Peer Status:* This is the critical point which hinders the feasibility of the traditional aggregation protocols in the streaming context. As peer status changes over time, it has to periodically update its status to aggregation points with whatever means. If a peer takes a long time to update the aggregation point about its new status, the updated status may become inaccurate since it is highly possible that the peers's status is changed again during the updating process.
- *Performance trade off:* Different from the traditional file sharing discovery(FSD), the peers always depart from the streaming session after finishing the playing back. According to the *Little's formula* in queuing theory, the average number of peers in the session is equal to the product of the average arrival rate and the average time spent in the system:

  $$E[N] = \lambda E[T] \tag{1}$$

  That means, in contrast with the most importance of scalability in FSD, the updating efficiency is more important here in the trade off between efficiency and scalability, since the population in the session is usually less than the case of file sharing.

Stemmed from these considerations, we place all peer status information in a structure, named *Segment Tree*, which is essentially a binary search tree with the facility to represent segments that are just the representation

Further author information:
Changxi Zheng: E-mail: cxzheng@ieee.org
Guobin Shen: E-mail: jackysh@microsoft.com
Shipeng Li: E-mail: spli@microsoft.com

of peers caching contents here. We will present this presentation in next section. All of the peers are divided into several groups according to their buffer range, and are organized as a 2-height tree as shown in Fig 1. The segment tree is divided into several subtrees, each of them are maintained by a group of peers cooperatively.

Furthermore, we noticed that in most of the streaming protocols, the delivery paths of control packets are tightly bundled with those for data packets. We argue that they should be decoupled as much as possible, so as observed in `Zigzag`.[2] Doing like this allows more flexibility and extensibility in designing protocols on both control plane and data plane. In our control plane protocol, it is flexible enough to provide any necessary information for data plane, so that it allows different protocols and optimization strategies on data plane.

3. SYSTEM ORGANIZATION

3.1. Cache Dependency Model

As described above, peers typically cache only a portion of the latest received data, and the contents of peers' buffer keep on updating continuously. Each peer need a structure to represent its buffer content and publish it for other peers' queries. Here we use the cache dependency model similar with the one introduced in `oStream`.[1]

Consider a media streaming have a time length of L seconds, a peer, say R_i, who starts to playback this media at time s_i from the offset o_i. It has a buffer with the length of l_i also represented by the time units. Then for another peer R_j, it starts at time s_j from the offset o_j. R_j can receive streaming from R_i if the following condition is satisfied.

$$\begin{cases} s_j - s_i + o_i < L \\ max\{s_j - s_i + o_i + l_i, s_i\} < s_j < s_j - s_i + o_i \end{cases} \tag{2}$$

where the first inequality represents that R_j should start to playback before R_i finish its playing, and the second inequality represents the starting position of R_j should be cached into the buffer of R_i. Here we assume all the peers playback the media with the same rate, so R_j can always receive stream from R_i as long as it can receive data from R_i at the starting position. Specially, We bring in two new terms here that would be referred in the following sections:

Definition 1. ***Absolutely Starting Time*** $|s|$***:*** *The value $|s|$ that is got from subtracting the starting offset from the starting time of a peer, $|s| = s - o$, is called the Absolutely Starting Time, which represents the starting time if the peer starts to play it from the beginning of the media stream.*

Definition 2. ***Caching Segment*** c***:*** *The caching segment of a peer is an time interval whose lower bound is the absolutely starting time, and whose length is the buffer length of that peer, say $c = [|s|, |s| + l]$.*

Based on the inequality (2) and definition (1) and (2), we notice the necessary condition for R_j receiving from R_i:

$$\begin{cases} s_i < s_j \\ |s_i| < |s_j| < |s_i| + l_i \end{cases} \tag{3}$$

That means the cached content of the peer R_i could be represented by its caching segment. A peer R_j who can receive stream from it should satisfy that the absolutely starting time of R_j is covered by the caching segment of R_i. In the following, we will present how to publish/query the caching segments and related status information on P2P network.

3.2. Segment Tree

As mentioned above, the caching content of each peer could be represented by the caching segment. Segment Tree(ST) is a data structure comes from the computational geometry, which is essentially a binary searching tree and is used to represent the segments with high efficiency as shown in Figure 2. Even through we do not use it as the overlay topology structure for the sake of the considerations mentioned in section 2, it is worthwhile to introduce it here to clarify our design philosophy.

As shown in Figure 2, the peer status information is maintained on the segment tree according to its caching segment. As an example in Figure 2, a peer with the caching segment of $[1, 3]$ is maintained at node $[1, 2]$ and $[2, 3]$. The maintained information consists of any information necessary for data plane protocol, which can be configured according to the specific data plane protocol. Furthermore, we notice that the high-level nodes on

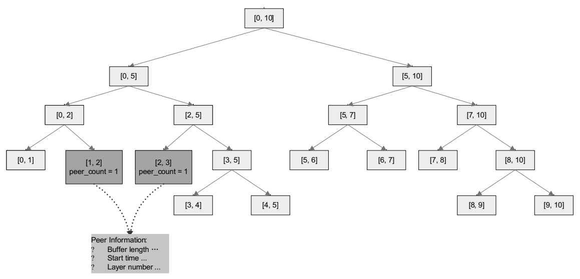

Figure 2. The Segment Tree

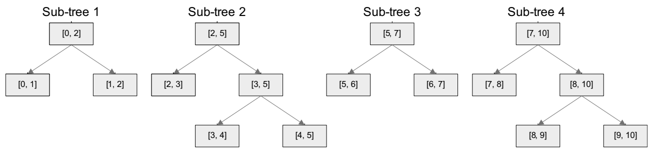

Figure 3. The Segment Sub-Trees

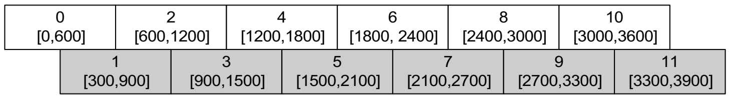

Figure 4. The overlapped ST subtrees

the segment tree, such as node $[0, 5]$ and $[0, 10]$ in Figure 2, rarely maintain peer information since the intervals represented by them are larger than the upper bound of the peers' caching segments. That means, as illustrated in Figure 3, the segment tree could be divided into several subtrees whose top interval–the interval represented by the root node of the subtree–is a bit larger than the upper bound of the peers' caching segments. And then, all the subtrees are distributed on the network, each of them is maintained cooperatively by a group of peers.

Figure 4 is a practical example for dividing the segment tree, where we have a media stream with the length of 1 hour(3600 seconds), and the upper bound of the length of peer caching segments is 5 minutes(300 seconds). We divide the segment tree into several subtrees with the length of 600 seconds, which is twice of the upper bound of caching segment. So for any caching segment, say $[a, b]$, there is always at least one, and at most two, subtrees that can fully cover the caching segment. We determine the subtrees it belongs to as follows, where

```
 10   a←a mod 3600;
 20   b←a + bufferLength;
 30   lid←a / 600;
 30   rid←b / 600;
 40   if ( lid = rid ) then subtreeId[0]←lid * 2;
 50   a←a - 300;
 60   b←b - 300;
 70   if ( a < 0 ) then
 80           a←a + 3600;
 90           b←b + 3600;
100   lid←a / 600;
110   rid←b / 600;
120   if ( lid = rid ) then subtreeId[1]←lid * 2 + 1;
```

Table 1. segment-subtree mapping algorithm

the `subtreeId` is the subtree id as shown in Figure 4. If there are two subtrees that both contain the peer's caching segment, one of them is selected randomly. For example, the caching segment $[450, 580]$ is covered by both $subtree_0$ and $subtree_1$, but the caching segment $[540, 720]$ can only be covered by $subtree_1$.

Suppose the peer arrival process is Poisson with rate λ, and the streaming length is L, the average number of peers in the network is $L\lambda$ [*]. If the upper bound of peers' buffer length is B, there are $2\lceil \frac{L}{2B} \rceil$ ST subgroups.

[*]We assume the peer arrival process is a Poisson precess with rate λ and the time of peer playing back is a exponential

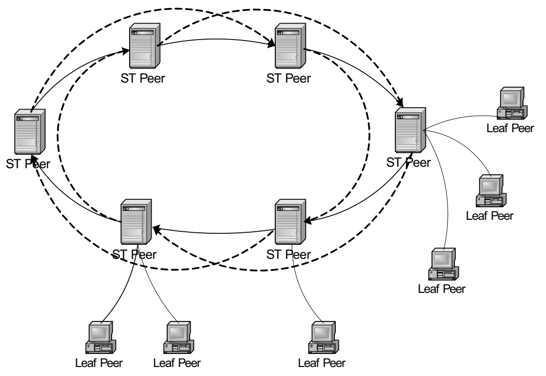

Figure 5. An example of the ST subgroup

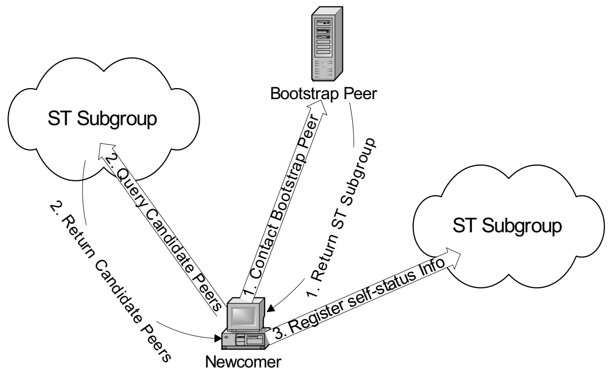

Figure 6. The procedure for a new peer arrival

Considering the continual peer arrival and the randomness of starting offset, all the peers is uniformly distributed on all the subgroups. The population of each subgroup is

$$\frac{L\lambda}{2\lceil \frac{L}{2B} \rceil} = \lceil \lambda B \rceil \tag{4}$$

3.3. ST Subgroup

An *ST subgroup* is a group of peers that cooperatively maintain a subtree mentioned above. It consists of *ST peers* and *leaf peers*. ST peers play the role as the aggregation points to collect the status information of leaf peers and publish it on the network. On the other hand, leaf peers hold connections with the ST peers and update the status information periodically.

All the ST peers in an ST subgroup is organized like a ring. Each ST peer on the ring maintain a table to record its next $k(k \geq 1)$ successors, where k is an adjustable parameter for the system. Usually, k is more than one in case of unforeseeable peer failures. In the next section, we will present that the circle is easy for adaption of group population, failure tolerance and recovery.

Each ST peer has several children, called leaf peers. The leaf peers periodically update their status information to the ST peers in order to publish to other peers. Periodical updating is important especially for a real-time streaming system, since the peer status is changed overtime, i.e. the length of the caching segment in time unit. Hence, here updating in time becomes more important, that is the reason why we give up a tree-based aggregation approach like many other traditional ones.

Meanwhile, there is a bootstrap peer on the network. Usually it could be the server, since the packets exchange on control plane wouldn't incur much of the burden on server. The bootstrap peer maintains a table about the t ST peers in each of the ST subgroup, where t is usually more than one in case of peer failures, so that it can tell the newcomers where to find the ST peers for queries and publications. The ST peers recorded by the bootstrap peer send the heartbeat message to it periodically, so that the bootstrap peer could notice the failure of these ST peers, and recruit the new ST peers in its table once some of them are failed or departed.

4. CONTROL PLANE PROTOCOL

Our proposed protocol on the control plane mainly deals with three cases, peer arrival, failure, and status updating. When a new peer arrives, the protocol is responsible for discovery of candidate peers that are potentially able to provide stream to it. Once a peer fails or departs from the current session, the overlay topology need to be recovered using this protocol. And periodically, each peer updates its published status information by this protocol.

random variable with rate $1/L$, where L is the streaming length. So the system is an $M/M/\infty$ queuing system, and the average number of peers is $L\lambda$.

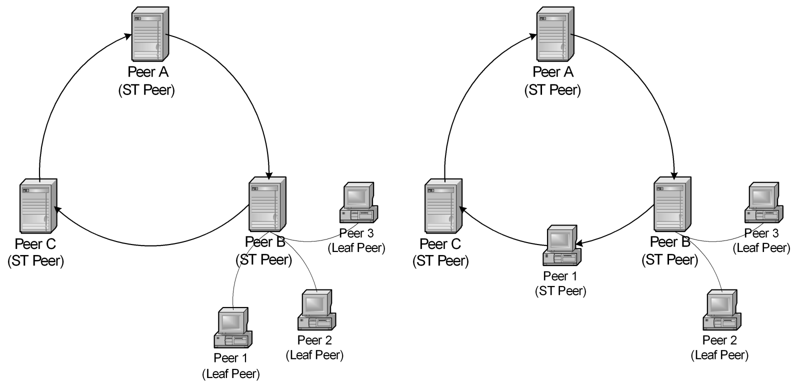

Figure 7. An example of expanding the ST subgroup ring

4.1. Peer Arrival

Anytime a new peer come to require the streaming, two things it need to do first of all. 1) Query the potentially parent senders. Once it knows the peers that are able to provide streaming for itself, transmission rate is allocated by some data plane optimization algorithm,[4] then streaming is started. Then 2) the newcomer publish its status information, including the caching content and some other necessary information, to the aggregation points in order to sometime provide streaming to others. Please see Figure 6 as an example of a new peer arrival.

Firstly, the newcomer gets contact with the bootstrap peer and get the response from there about two ST subgroups, called *query group*, that covers the absolutely starting point of the demander according to the constraint (3) and the ST subgroup , called *register group*, where the newcomer should register itself according to the algorithm in Table (1). Note that it only returns an ST peer id that belong to the corresponding ST subgroup. The returned ST peer in each ST subgroup is selected randomly for load balancing on all the recorded ST peers.

Then the newcomer sends the queries to the query group to ask for the candidate peers. Once an ST peer receives the query, it returns the peers' status information registered on itself which satisfy the constraint (2), and furthermore, it relay the query to its next k successors on the subgroup ring, where k is the parameter mentioned in section 3.3. So each ST peer on the subgroup ring would receive at most k same queries, each ST peer only responses the first query it received and simply discards the later ones. Even though some ST peers are failed abruptly, the query still propagate on the ring successfully, since the simultaneous failures of all the k successors is almost impossible. Based on this structure, the query would propagate on the subgroup ring at the rate of k. In other words, it needs $\lceil \frac{n}{k} \rceil$ hops to propagate a query to all the ST peers on the ring, where n is the number of ST peers on the ring. That means the number of hops increases linearly with the number of ST peers in the ST subgroup. We argue that, as mentioned in section 2, is because the updating efficiency is more important here rather than the scalability upon the tradeoff between them. In fact, our experimental results in section 5 show that the initial delay for query is still short.

At last, the newcomer register its status information on register group after it receives the response for candidate senders from the ST peers and starts to streaming. Please note that the register group is always one of the query group, which is easy to be deduced from the constraint (3) and algorithm in Table (1). From the attachment of the query response of each ST peer, the newcomer knows the number of leaf peers associated with each of the ST peers. So it selects the ST peer in the register group that has the least child leaf peers, registers on it, and starts to periodically update its status information there.

4.2. Peer Failure/Departure

4.2.1. Leaf Peer Failure/Departure

The failure/departure of the leaf peer doesn't influence any others. Its parent ST peer would see the failure for the lack of periodical status updating message, then remove the failed peer from its peer status table. That is the only thing need to do since the failure of leaf peer doesn't destroy the overlay topology on the control plane.

4.2.2. ST Peer Failure/Departure

Each ST peer on the ST subgroup ring exchange heartbeat message with all of its children as well as its next k successors. So the failure of an ST peer would be noticed by its children, its k predecessor on the ring. They also send the heartbeat messages to the bootstrap peer if they are selected into the bootstrap peer's local table, and hence they are also noticed by bootstrap peer for their failures.

Each time the leaf peer sends its status updating message as a heartbeat message to its parent ST peer, the ST peer responses it with the addresses of its next k successors. So once the ST peer is failed and is noticed by its child leaf peers, they randomly choose one of its successors to register themselves there. This is quite efficient, since there are no other queries or routings for recovery.

The ST peer on the ST subgroup ring knows the failure of any of its next k successors for the heartbeat message exchange. Once it notices this case, it removes the failed peers from its successor table, and ask the last living successors for its immediate living successor to replace the failed one into the successor table.

4.3. Expand and Shrink on ST Subgroup Ring

As the change of the population of the ST subgroup, the number of ST peers on the ST subgroup ring should be also be changed to adapt the subgroup population. We set the a parameter, *children upper bound*, to determine when to expand the subgroup ring. As more and more peers become the children of an ST peer, it periodically check the number of its children. Once the children number excesses the children upper bound, the ST peer starts to recruit one of its children to be a new ST peer.

Figure 7 shows this case as an example. Peer B selects peer 1 to be the new ST peer, adds it into its local successor table, and remove the last successor from that table. Peer 1 knows its next k successors from peer B, and creates connections with them.

On the other hand, if a ST peer has no children, and its immediate predecessor on the ring has the children less than the children upper bound, it require to become the child of its predecessor so that the ring is shrank. The predecessor updates its successor table from the successor table of the demoted ST peer. All the predecessors connecting with the demoted ST peer are notified to update their successor table.

5. EXPERIMENTAL RESULTS AND EVALUATION

We evaluate the performance of our scheme in this section. We generate the topologies for our simulation using the `BRITE`[9] topology generator based on *Albert-Barabasi* model. In all of our following simulations, we use the stream with the length of an hour. The buffer length of each peer is uniformly distributed from 20 seconds to 180 seconds. The peer arrival process is Poisson with rate of 5 peers per minute, and the time length of peer playing back is a exponential random variable with the average length of 1 hour.

5.1. Accuracy Evaluation

In this experiment, we use BRITE to generate 1000 peers based on *Albert-Barabasi* model. All of the peers arrive the session following a Poisson process. Each peer requests their potential senders on this network, and then registers itself status information. We use the Metric, *Accuracy Rate*, to evaluate the query accuracy, which is the ratio of the number of returned potential senders and the number of peers that could actually provide streaming to the newcomer. Figure 8 shows the results of the accuracy percentage. We can see that more than %85 peers could retrieve all of its potential senders. In the worst case, the peer could know %75 potential senders.

5.2. Size of ST Subgroups

We use the same configuration with the previous experiment, but observe the change on size of the ST subgroups. We sample the subgroup size every 30 seconds, and choose three of the ST subgroups to plot the samples on Figure 9. It shows that there are at most 18 peers in a subgroup with the arrival rate of 5 peers per minute, and at the steady state, there are about 12 peers in an subgroup on average, which is consistent with our theoretical analysis in equation (4).

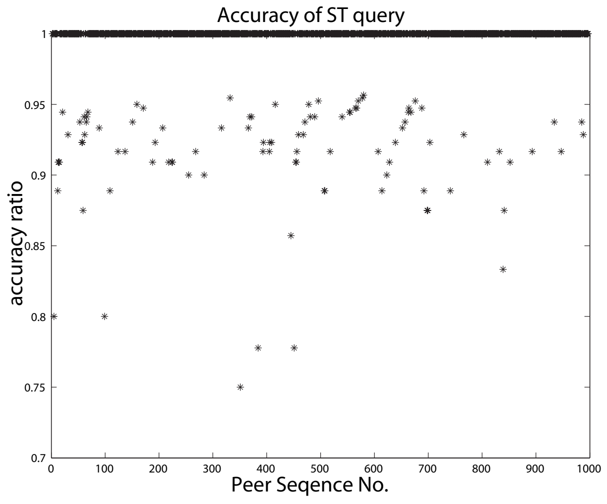

Figure 8. Query accuracy of peers

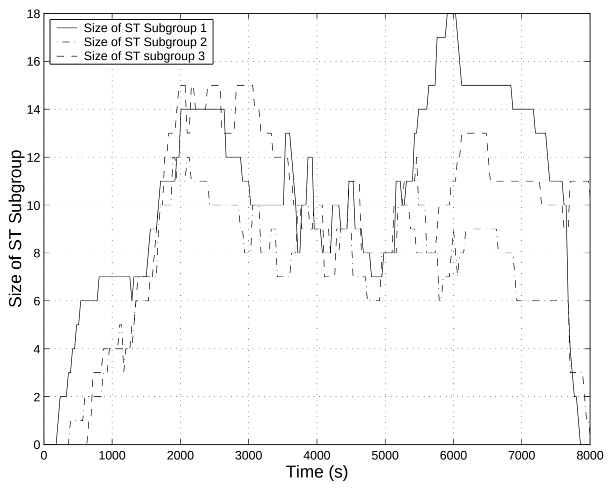

Figure 9. Change of ST subgroup size

5.3. Query Delay

In this experiment, we use BRITE to generate 300 nodes on the network and randomly select 100 peers of them. We observe the query delay under different configurations. Since the peers' queries would spend some time to propagate on the ST subgroup ring, we plot the minimum and maximum delay from sending out the query request to receiving the response from ST peers.

From the control plane protocol, there are two parameters that would influence the delays, the *children upper bound* that is the upper bound of children number of each peer, and *the number of next successors*(k) of each ST peer maintained. If the children number of a ST peer exceeds the upper bound, the ST subgroup ring would be expanded, which would prolong the propagation time of a query on the ring. On the other hand, the number of next successors also influence the propagation rate on the ring.

Figure 10 shows the minimum delay of a query, that is the time spent till the first ST peer response. We see that the minimum query delay is almost the same under different configurations. But Figure 11, which is plotted by the time spent till the last ST peer response, show that the more children and maintained successors by an ST peer, the shorter the maximum query delay. Figure 12 shows the time difference between the maximum and the minimum query delay is at most 3 seconds if the children upper bound and maintained next successors are both 1, and in other cases, the difference is about 0.5 second, more or less.

5.4. Overheads on Failure Recovery

We use 100 peers randomly selected from 300 nodes generated by BRITE. In this experiment, we observe the overheads used to recover the overlay topology after the failure of a peer. The overheads are counted by the number of packets exchanged between any two peers to recover the topology. So the overheads for the failure of a Leaf peer is always be zero. We also plot the overheads under 4 kinds of configurations like the above experiment in Figures 13,14,15 and 16. The peak overheads is the most when children upper bound is 6 and number of maintained next successors is 4, but the number of leaf peers is also the most in this case, so many of the failure recovery overheads is zero. Contrastively, the peak overheads is the least when children upper bound and number of maintained next successors are both 1, but overly a few overheads are zero, since the number of leaf peers is also the least.

6. CONCLUSIONS

In this paper, we present a protocol on control plane to provide efficient yet flexible service discovery mechanism for video streaming over P2P network. We first model the caching content of a peer into a segment, and then publish/subscribe the segments on the network for streaming service discovery. On local we maintain the segments in a segment tree, on network, we divide the segment tree into several subtrees, and distribute them. Each subtree is maintained cooperatively by a group of peers. Status aggregation points in each group are organized into a ring, and the leaf peers updates its status information there to publish to others. Our protocol fully decouple the control plane from the data plane, and provide more flexibility for the protocol on data plane.

REFERENCES

1. Y. Cui, B. Li, and K. Nahrstedt, "ostream: Asynchronous streaming multicast in application-layer overlay networks," *IEEE Journal on Selected Areas in Communications* **22(1)**, January 2004.
2. D. Tran, K. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," *in Proceedings of IEEE INFOCOM*, 2003.
3. M. Bansal and A. Zakhor, "Path diversity for overlay multicast streaming," *in IEEE International Packet Video Workshop*, 2004.
4. T. Nguyen and A. Zakhor, "Distributed video streaming with forward error correction," *in Packet Video Workshop*, (Pittsburgh, PA), April 2002.
5. T. Nguyen and A. Zakhor, "Multiple sender distributed video streaming," *IEEE Transaction on Multimedia* **6(2)**, pp. 315–326, April 2004.
6. R. Bhagwan, G. Varghese, and G. Voelker, "Cone: Augmenting dhts to support distributed resource discovery," Tech. Rep. CS2003-0755, UC, San Diego, July 2003.
7. R. Renesse and A. Bozdog, "Willow: Dht, aggregation, and publish/subscribe in one protocol," *in Workshop on Peer-to-Peer System*, February 2004.
8. A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," *in Proceedings of ACM SIGCOMM*, 2004.
9. A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," *in Proceedings of MASCOTS*, (Cincinnati, Ohio), August 2001.

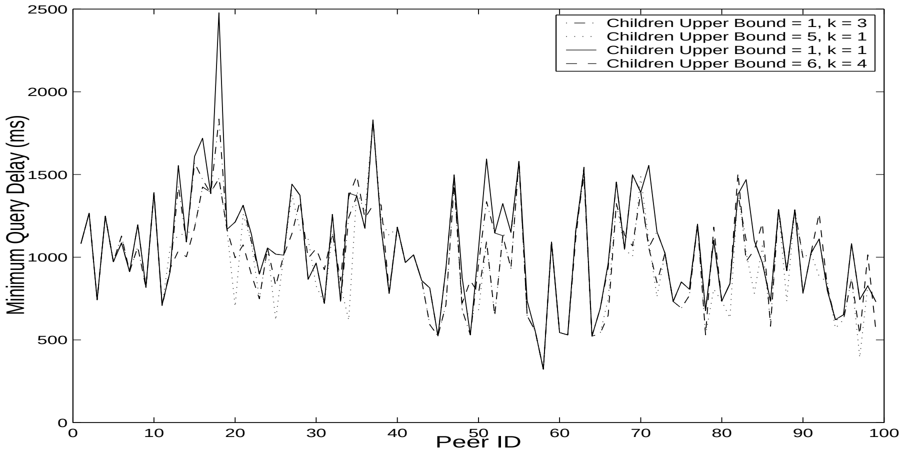

Figure 10. Minimum Query Delays

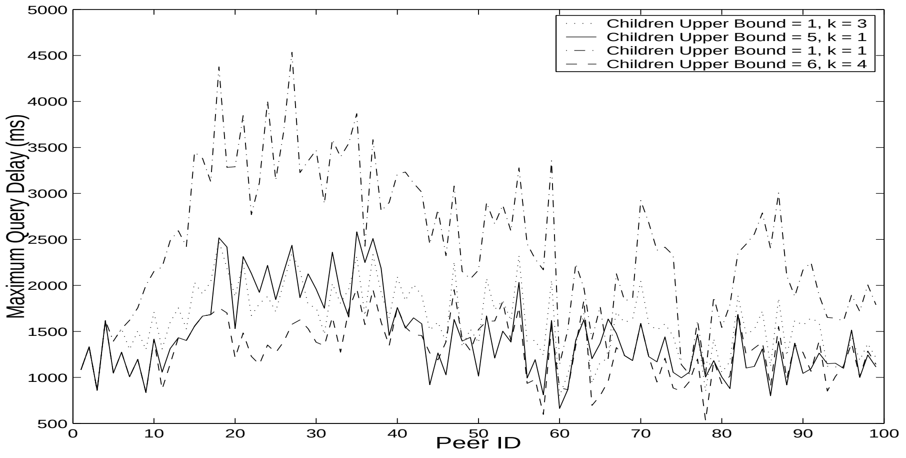

Figure 11. Maximum Query Delays

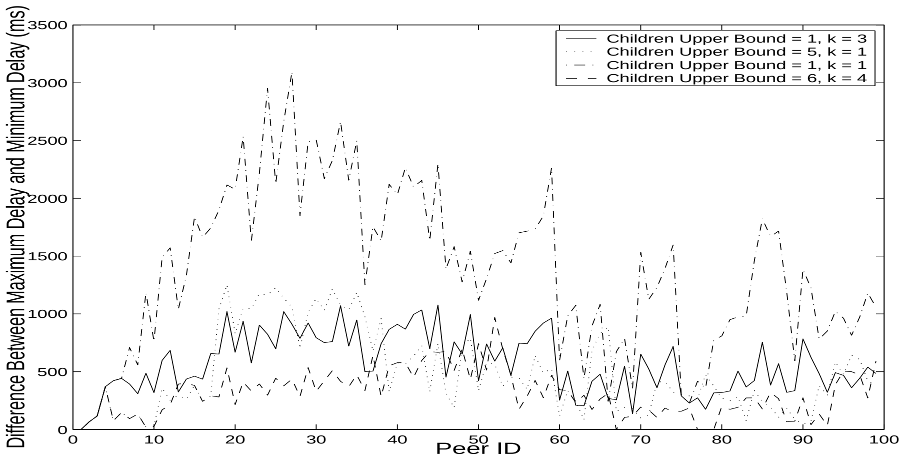

Figure 12. Time Difference Between Maximum Query Delays and Minimum Query Delays

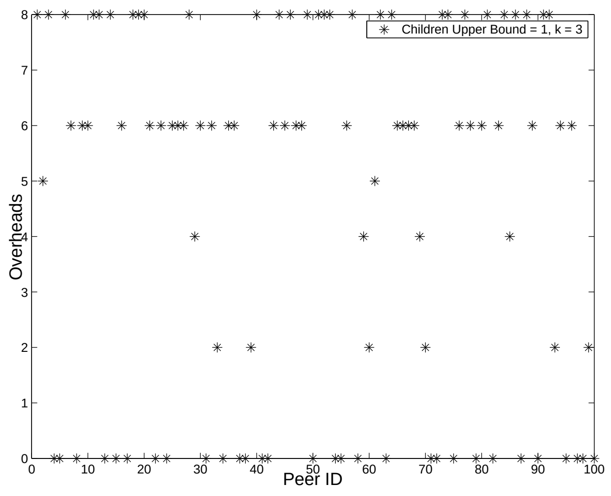

Figure 13. Failure Recovery Overheads: Children Upper Bound = 1, k = 3

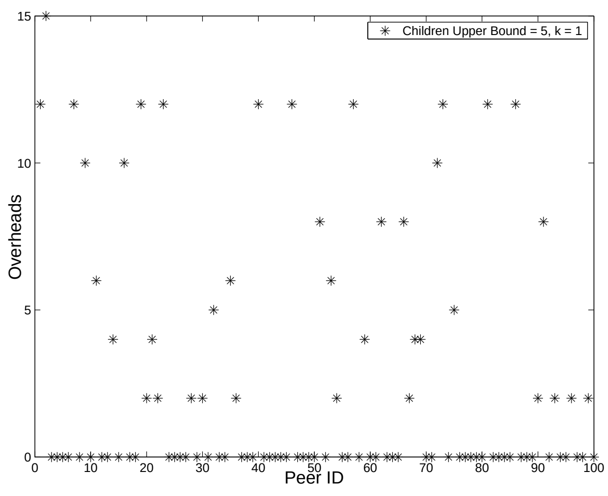

Figure 14. Failure Recovery Overheads: Children Upper Bound = 5, k = 1

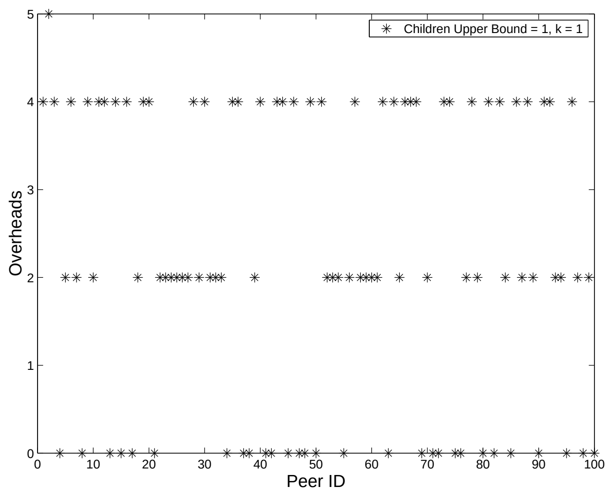

Figure 15. Failure Recovery Overheads: Children Upper Bound = 1, k = 1

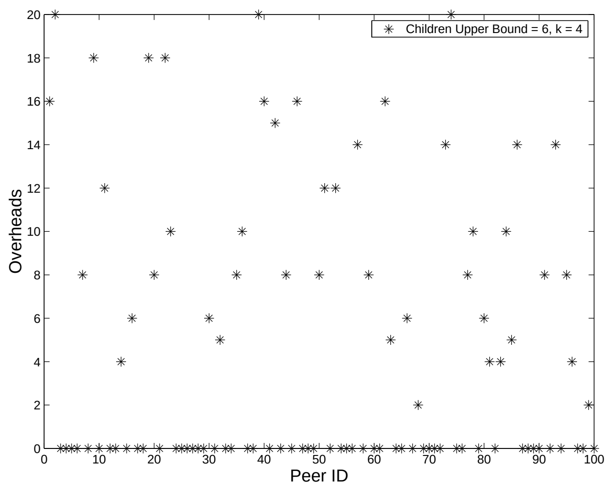

Figure 16. Failure Recovery Overheads: Children Upper Bound = 6, k = 4