9-2007

Variable Neighbor Selection in Live Peer-to-Peer Multimedia Streaming Networks

Jagannath Ghoshal
University of Nebraska-Lincoln, jghoshal@cse.unl.edu

Byrav Ramamurthy
University of Nebraska-Lincoln, bramamurthy2@unl.edu

Lisong Xu
University of Nebraska-Lincoln, xu@cse.unl.edu

Variable Neighbor Selection in Live Peer-to-Peer Multimedia Streaming Networks

Jagannath Ghoshal, Byrav Ramamurthy and Lisong Xu
Department of Computer Science and Engineering,
University of Nebraska-Lincoln, Lincoln, NE 68588-0115
Email: {jghoshal, byrav, xu}@cse.unl.edu

***Abstract*—Data-driven (or swarming based) streaming is one of the popular ways to distribute multimedia streaming traffic over live Peer-to-Peer (P2P) networks. In data-driven streaming networks, each peer independently selects its neighbors based on gossip-style overlay construction and then exchanges streaming data with the neighbors using data scheduling. In a P2P network the major advantage is that each peer contributes its own resources to the network. As a result, there is an increase in the amount of overall resources of the network, such as bandwidth, storage space, and computing power. However such type of networks exhibit end user bandwidth heterogeneity. Consequently, the overall bandwidth increase requires proper distribution of the number of neighbors for every peer, so that the upload capacity of high bandwidth peers is efficiently utilized. In this paper we improve a fixed random neighbor-selection algorithm *FRNS* (implemented in *p2pstrmsim* simulator), in order to achieve proper distribution of neighbors to the peers. We propose *VRNS*, a variable random neighbor-selection algorithm by modifying *FRNS* and find that there is efficient use of upload capacities at high bandwidth peers. Our work is evaluated by performing simulations with the improved *p2pstrmsim* simulator. Our results show that there is significant improvement in overall network performance and also around 60% decrease in peer elimination by using *VRNS*.**

I. Introduction

Peer-to-Peer (P2P) media streaming networks, motivated by the huge success of P2P file downloading networks, have recently attracted a lot of research interest. However, it is challenging to design P2P media streaming networks because of the stringent time constraints on the delivered media streams, which require more efficient and resilient overlay architectures [3]. In this paper, we focus on live P2P media streaming networks, a promising application flourishing in the Internet and which requires the distribution of live (not stored) multimedia content to subscribers.

Due to the presence of end user bandwidth heterogeneity, the objectives of good peer upload capacities, high scalability and high bandwidth demand of multimedia applications are difficult to achieve. Thus proper neighbor selection algorithm should be implemented to alleviate such problems and make the network more scalable. The limited bandwidth capacities in peer-to-peer networks pose a significant technical challenge to peer-to-peer media streaming. As nodes reside at the edge of the Internet, they usually have limited availability of upload and download capacities. In addition, these peers have greater download capacities than upload capacities. Therefore this causes a problem at low bandwidth peers, since there is inefficient utilization of the upload capacities at the high bandwidth peers.

The neighbor selection strategy implemented in *p2pstrmsim* [1] selects a peer at random as its neighbor. In addition, every peer has the same number of neighbors, which means that the low bandwidth peers have the same number of neighbors as high bandwidth peers. This causes a problem since the upload bandwidth capacity of high bandwidth peers are not used efficiently and the overall performance of the network is degraded. Hence, our conclusion is that, these low bandwidth peers should not have the same number of neighbors as high bandwidth peers, since they do not have the ability to handle the same number of connections as high bandwidth peers.

In this paper we address this challenge by modifying the fixed random neighbor-selection (*FRNS*) algorithm which is implemented in *p2pstrmsim* such that the peers achieve better streaming quality and to increase the overall performance of the network. Our contribution has proved to be effective as shown by the simulation results in Section IV with the help of a P2P live streaming simulator called *p2pstrmsim* [1]. The overall performance of the network using our enhancement is compared to the earlier performance and the results qualifies our modifications to the algorithm.

The rest of the paper is organized as follows. Section II presents some related work in this area. We present the problem description in Section III where we explain the modification to the algorithm. Section IV represents the simulation results and discussion and finally Section V concludes the paper.

II. Related Work

The purpose of peer discovery is to find the information about other peers in a P2P streaming network, when a peer initially joins the network. Since the obtained peer information will be used to select appropriate neighbor peers for a peer, the desired information includes the following: which peers are currently in the network, what the bandwidth usage of a specific peer is, what the workload of a specific peer is, etc.

One of the popular ways to perform neighbor-selection is based on packet delays. Since packet delay is relatively easy to measure, many P2P streaming networks use packet delay to select appropriate peers. A peer may select geographically nearby peers in order to minimize the packet delay between its parents and itself as implemented in DagStream [5], or it may select those peers that can minimize the total delay from the source peer to itself as in ZIGZAG [9] and AnySee [6]. However, the delay between two peers cannot reflect the available bandwidth between them. In other words, it is possible that the delay of a path is short, however, its available bandwidth is not high enough for media streaming.

Peer selection based on delay or bandwidth may achieve good performance, if the network and peer conditions do not change. However, it is possible that the constructed overlay may have a large number of shared links or shared parent and children peers, and it performs poorly in dynamic network environments. For this reason, some P2P streaming networks such as PRO [8] randomly select parent peers with the purpose of providing more diverse paths and diverse parents. During peer selection routine, peers should consider selecting some resilient nodes as their parents according to some standards. Note that the goal to be more resilient to peer and network dynamics does not necessarily conflict with other goals. For example, PRO randomly selects a parent peer according to a probability proportional to its available bandwidth.

PRIME [7] has addressed the neighbor-selection problem based on bandwidth bottleneck. The authors minimized the bandwidth bottleneck by finding out the total number of incoming and outgoing degrees that a peer should have based on its bandwidth capabilities. Additionally their work was based on a directed mesh-based overlay, where there is a parent-child relationship among the peers. However, their work has not been experimented with in an unstructured, undirected live P2P networks, where we encounter no parent-child relationships and have larger number links to manage.

In networks using the gossip-based method such as [10], CoolStreaming [11], and AnySee, a peer maintains the information about a small list of randomly selected peers, initially obtained from a starting peer, and the information is updated periodically by exchanging messages with other peers in the list. The gossip-based method has good scalability and is suitable for large-scale P2P streaming networks. However, a weakness of the gossip-based method is that the media quality cannot be guaranteed, since a group of randomly selected peers may not have enough resources to provide the desired media quality. Additionally, these random neighbor-selection algorithms do not provide efficient use of bandwidth capacities at the peers.

Our work is based on an undirected, unstructured mesh-based live P2P network, where we have more number of links to manage. We implement a variable random neighbor-selection *VRNS* algorithm, which is an improved version of *FRNS* algorithm in order to determine how many neighbors should be assigned to a peer based on its outgoing bandwidth. Our objective is to ensure that the upload capacities of these peers are efficiently utilized to achieve better performance of the network.

III. Problem Description

As mentioned earlier the major problem with the random neighbor-selection as implemented in [10] is that low bandwidth peers could become a bottleneck. Additionally the upload capacity of a high bandwidth peer is not utilized in an efficient manner. Our improvement to the random neighbor-selecton strategy is evaluated by performing numerous simulations with *p2pstrmsim* [1], used in [10] and showing that through such modifications we can achieve a better performance of the network. Our simulation results as shown in Section IV demonstrates a better performance due to our improved algorithm.

A. Fixed Random Neighbor-Selection Algorithm (FRNS)

The general random neighbor-selection algorithm (implemented in [1]) assigns neighbors to peers randomly. Hence we also refer to it as Fixed Random-Neighbor Selection algorithm (*FRNS*). The number of neighbors assigned to each peer is fixed for static environments. For dynamic environments the number of neighbors does not exceed a fixed value. In a static environment, peers do not frequently leave or join the network, that is, it does not have a high churn rate. However in a dynamic environment (see Section IV-B) peers join and leave the network frequently resulting in a high churn rate. This dynamic nature is simulated by using the traces from ‘Gridmedia’ [4]. The *FRNS* algorithm is presented as Algorithm 1 below.

Algorithm 1 Fixed Random-Neighbor Selection Algorithm

```
1:  if number of online nodes < 0 then
2:    return 0;
3:  end if
4:  while number of online nodes > 0 do
5:    node id = random(mod number of online node size)
6:    if node id ≠ self node id then
7:      if find(node id) not in the list then
8:        if node id neighbors < neighbor count then
9:          add neighbor nodes to node id
10:       else if some node id timer expires then
11:         eliminate node id
12:       end if
13:     end if
14:   end if
15: end while
```

FRNS starts by searching for online nodes that could possibly become a neighbor of the searching peer. Once it retrieves this information it creates a list which contains these retrieved online nodes. The list is denoted by a variable ‘node_idx’ in *p2pstrmsim*, which retrieves the position of these nodes in the list. Line 5 helps to retrieve a random position number from the list that points to a node id. This node is checked to see if it is a self-node and if it is not then we check to see if it is still available in the list, because the timer assigned to a node might expire before it can serve as a neighbor to the requesting peer. Once the above conditions are satisfied as shown in Lines 6, 7 and 10, the retrieved peer then serves as a neighbor to the requesting peer. The random behavior for peer selection is kept intact in the modified version, but the number of neighbors assigned is changed as discussed in Section III-B.

B. Variable Random Neighbor-Selection Algorithm (VRNS)

Since there are heterogeneous bandwidth peers, the above strategy fails to provide good performance because the upload bandwidth capacity of high bandwidth peers is not used efficiently. Hence the overall performance of the network is degraded. We propose a modified algorithm called Variable Random Neighbor-Selection algorithm *VRNS* (see Algorithm 2) alleviates the above problem by assigning a proper number of neighbors to the peers based on their bandwidth capacity.

Algorithm 2 Variable Random-Neighbor Selection Algorithm

```
Require: neighbor count < online nodes
1:  if number of online nodes < 0 then
2:    return 0;
3:  end if
4:  while number of online nodes > 0 do
5:    Calculate total number of Links (using $L_{FRNS}$)
6:    $L_{VRNS} = L_{FRNS}$ {Since, Link size is same}
7:    Calculate the values of $n$'s based on Equations 8, 9, 10.
8:    node id = random(mod number of online node size)
9:    if node id $\neq$ self node id then
10:     if find(node id) not in the list then
11:       if bandwidth type then
12:         if node id neighbors < neighbor count of the bandwidth type then
13:           add neighbor nodes to node id
14:         else if some node id timer expires then
15:           eliminate node id
16:         end if
17:       end if
18:     end if
19:   end if
20: end while
```

In *VRNS*, Line 5 calculates the total number of online nodes in the network. This is required at every run because the number of online peers changes overtime in case of dynamic environments, whereas for static environments the calculation is done only once. The distribution ratios of the peers shown in Table I are known beforehand. Since the number of links is equal, the total number of links for *VRNS* should be equal to the total number of links for *FRNS* (as shown in Line 6). Based on this, we can compute the number of neighbors, shown in Line 7 for different types of peers. Then a random selection (Line 8) of a node is performed in the constructed list. Lines 9 and 10 checks to see that it is not a self node and still exists in the list. Finally, the algorithm checks the bandwidth of the selected node and assigns the proper number of neighbors based on their n values (as shown in Lines 11 to 15).

C. Analysis of the Algorithm

The following analysis illustrates how to determine n_1, n_2, and n_3, by utilizing the same number of links, to ensure that *VRNS* has the same number of links as *FRNS*. This is maintained since we do not want to increase the overhead in the network.

Consider a P2P network $P2P\_N$, based on *FRNS*, which has P online peers and each peer has N neighbors, where,

$$N < P \tag{1}$$

Therefore the total/maximum number of links in *FRNS* (or L_{FRNS} in $P2P\_N$) is,

$$L_{FRNS} = (P \times N)/2 \tag{2}$$

P consists of three types of peers with different upload bandwidth capacities (see Table I).

p_1 peers have a bandwidth of b_1
p_2 peers have a bandwidth of b_2
p_3 peers have a bandwidth of b_3

where,

$$P = p_1 + p_2 + p_3 \tag{3}$$

Next we consider the number of neighbor assignment in *VRNS*: Let,

p_1 has n_1 neighbors
p_2 has n_2 neighbors
p_3 has n_3 neighbors

Since the upload bandwidth is not same for every peer, the total number of links for a group of peers with same upload bandwidth should also be different. Therefore we divide the number of neighbors for a peer based on its bandwidth ratio. So,

$$b_1 : b_2 : b_3 = x_1 : x_2 : x_3 \tag{4}$$

Similarly for neighbors,

$$n_1 : n_2 : n_3 = x_1 : x_2 : x_3 \tag{5}$$

Therefore, the total/maximum number of links,

$$L_{VRNS} = ((p_1 \times n_1) + (p_2 \times n_2) + (p_3 \times n_3)) \div 2 \tag{6}$$

Since, we keep the same number of links, we can say that:

$$L_{VRNS} = L_{FRNS} \tag{7}$$

Therefore the total number of neighbors are,

$$n_1 = (2L_{VRNS} - (p_2 \times n_2) - (p_3 \times n_3)) \div p_1 \tag{8}$$
$$n_2 = (2L_{VRNS} - (p_1 \times n_1) - (p_3 \times n_3)) \div p_2 \tag{9}$$
$$n_3 = (2L_{VRNS} - (p_1 \times n_1) - (p_2 \times n_2)) \div p_3 \tag{10}$$

Each of the above values can be computed by proper substitution of n_1, n_2 and n_3 ratio.

Thus from the above equations we can assign appropriate number of neighbors based on the upload bandwidth capacity ratio. Section IV shows the performance evaluation and comparison of the two algorithms.

IV. Simulation Experiments and Results

A. Simulation Setup

Our simulation is based on six parameters namely, 'Average Playback Delay', 'Average Packet Delay', 'Average Hop Count', 'Average quality', 'Total Eliminated Peers', and 'Request Success Ratio'. Each of these is related to the dynamic environment of a Live P2P mesh-topology. These characteristics help us to compare the performance of a live P2P network based on neighbor selection in the two algorithms. The simulation duration is not fixed and can be set to a specific time, which signifies the total time that the network needs to be live. In every simulation, the statistics or performance of the network is monitored on all the online nodes available at every 10-second interval. Additionally after every 50 seconds of simulation time, the above parameters related to performance of the network are written to the output. It should be noted that the simulation time is not the same as the actual time.

p2pstrmsim [1] is a discrete event-based simulator used to simulate the present work and it is capable of simulating large-scale live data-driven (or swarming based) P2P media streaming networks. The simulator supports single channel streaming in an unstructured network and uses mesh-topology for overlay construction and it simulates two environments, static and dynamic. For the underlying topology, it uses the random model of GT-ITM [2] to generate a topology with 2000 routers and sets delays proportional to the distance metric of the resulting topology within a range [5ms, 300ms].

The simulation is carried out with a network topology of 2500 nodes with a pull-based random protocol and a broadcasting peer sending data at a streaming rate of 300 Kbps with 2 Mbps upload bandwidth. The network consists of three types of peers as shown in Table I and a requesting window of size 20 seconds in every peer. This window denotes the latest generated segments and only these packets are requested if missing. The total simulation duration for every run is set to 500 seconds. A random user behavior algorithm is used in order to simulate a dynamic network. Two environments namely, static and dynamic, are simulated with three different settings of bandwidth ratio distribution, which are 15%-25%-60%, 30%-40%-30% and 60%-25%-15% representing the number of different types of DSL/Cable peers as shown in Table I in the network respectively. Due to the limitation of the paper only the dynamic environment with the first bandwidth ratio distribution is presented. The other scenarios also perform similarly to the presented results.

B. Dynamic Environment:

In our simulation, we study the effectiveness of the *VRNS* algorithm in a dynamic P2P environment where the peers join and leave frequently, i.e., means the network can have a very high churn rate. The arrival rate (or joining rate) of the peers is set to 10 users per second. The churn rate of network is simulated by referring to the traces from 'Gridmedia' [4]. The elimination of a peer is based on a threshold condition, such as, a peer should have the downloading rate of more than 15 Kbps. Every 10 seconds the behavior of the network is documented until the total number of peers reach a fixed value; in our case it is 2500 peers in the network. All the results shown below are obtained by changing the number of neighbors assigned. Only Figures 4 and 5 are based on individual number of neighbor assignment, that is 10 and 40 neighbors respectively.

TABLE I
Bandwidth Ratio Distribution Setup

Type	Inbound (Kbps)	Outbound (Kbps)	Ratio
DSL/Cable	3000	1000	15% or 30% or 60%
DSL/Cable	1500	384	25% or 40% or 25%
DSL/Cable	784	128	60% or 30% or 15%

1) Average Playback Delay: The simulator assumes that the node will play back the data in the buffer when the quality reaches 99%. The delay computed here is between the time when the packet is sent out from the source node and when this packet is played back at the respective peer. The parameter is important to calculate since we can know the average difference of the playback time (or lagging) between the source node, which has the actual playback time, and rest of the peers in the network. Every block that is received at a peer contains a timestamp, which indicates its playback time at the source node. This playback time is compared with its playback time at a peer and the difference obtained in the playback time is said to be the delay of the block. This delay is calculated every 50 seconds at the peers and averaged to obtain the average playback delay.

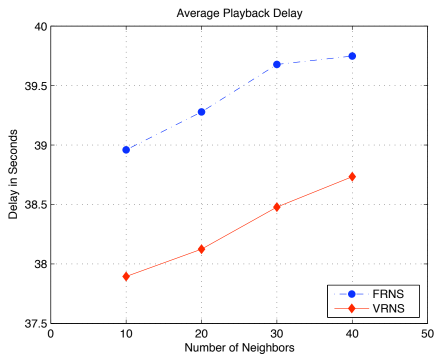

Fig. 1. Average Playback Delay with Bandwidth Ratio of 15% 25% 60%

The average playback delay is affected by the churn rate of the network. Fundamentally this delay depends on the frequency of peer elimination and time needed for a new peer to join the network and also on the packet losses occurring in the network. As shown in Figure 1, *VRNS* performs significantly better than *FRNS* since the average number of peers eliminated from the network is less than the in the case of *FRNS*; the cause of such a behavior is explained in Section IV-B5. Thus our modifications have helped to achieve a lower delay.

2) Average Packet Delay and Average Hop Count: The packet delay and hop count of a packet are inter-related. Since a packet needs to travel several hops before it can reach the destination node from where it is requested. As described in Section IV-A, the propagation delay of a link is fixed and therefore the performance of the average packet delay shown in Figure 2 shows a similar trend to the performance of the average hop count in Figure 3. This signifies that, the hop count of a packet to travel has decreased as a result of which the delay of the packet from the source node to the destination peer is also minimized.

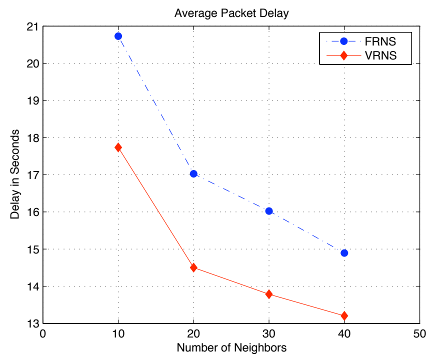

Fig. 2. Average Packet Delay with Bandwidth Ratio of 15% 25% 60%

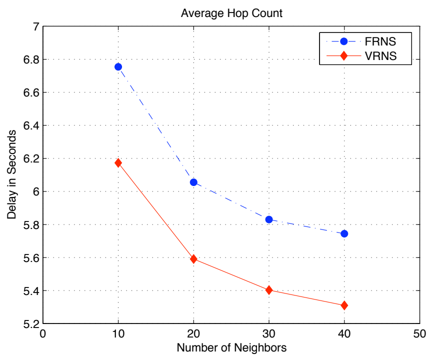

Fig. 3. Average Hop Count with Bandwidth Ratio of 15% 25% 60%

The average packet arrival delay at each node is the delay between the time when the packet is sent out from the source node and when this packet arrives at the destination node. Similar to the playback delay, the time difference between packet's departure time its arrival time at a node is computed and averaged to obtain a value for every 50 seconds of simulation duration until there are 2500 nodes. Finally these values are further averaged to obtain the average packet delay for every number of neighbors as shown in Figure 2.

All the streaming packets that are present in the network are calculated based on their arrival time at the online nodes that are available in the network. The hop count of a packet is the total number of hops it takes to reach a peer from the source node. Similar to the average packet delay, Figure 3 averages all the values and shows the performance by changing the number of neighbors assigned.

From our simulation result we can see that the number of average hops decreases, because the number of links present with the high bandwidth peers have increased. Consequently these peers help in decreasing the diameter of the network, which helps in building shorter routes to other peers of the network.

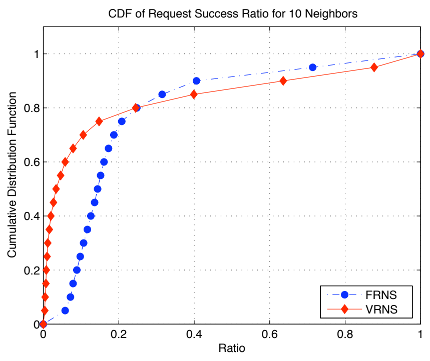

Fig. 4. CDF of Request Success Ratio with 10 Neighbors and Bandwidth Ratio of 15% 25% 60%

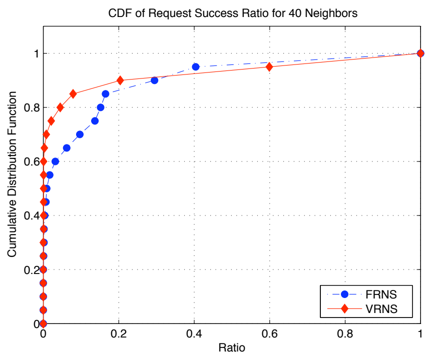

Fig. 5. CDF of Request Success Ratio with 40 Neighbors and Bandwidth Ratio of 15% 25% 60%

3) Average Success Ratio: Figures 4 and 5 show the performance of our algorithm in terms of 'Average Success Ratio' for 10 and 40 neighbors respectively and the performance for 20 and 30 neighbors are similar and hence not presented due to the limited length of the paper. The request success ratio is defined as the probability that a packet is successfully requested from a peer after a request is sent. In *p2pstrmsim* [1], for a packet to be requested there is a request period of 1-6 seconds. If any packet requested by a peer is not received then, the peer can again request the same packet within this time period. The request is only made to its neighboring peers and not the whole network. If the packet is not available at its neighboring peer then these peers further send the request to their neighbors. However, this operation is only carried out if the packet is said to lie within the respective requesting window, otherwise the request is not met. So the request success ratio is measured based on the total number of requests successfully met to the total number of request made. As shown in the Figures 4 and 5 the cumulative distribution function is computed for these ratios for every request made by each peer in the network. Therefore the results shows that the performance *VRNS* is better than that of *FRNS*, since the upload capacities of high bandwidth peers is efficiently utilized. Thus *VRNS* helps in making these packets readily available so that the number of requests made at each interval is decreased.

4) Average Quality: The quality or delivery ratio of a peer is calculated on the total number of blocks arriving at this peer before the playback deadline to the total number of blocks available in the encoded stream. The total number of streaming blocks remains constant and hence this value is affected based on encoding and packetization [10]. Fundamentally, it represents the throughput of this peer from the source node to itself. Thus the values obtained (as shown in Figure 6) is the average quality (or delivery ratio) of all the online peers in the network.

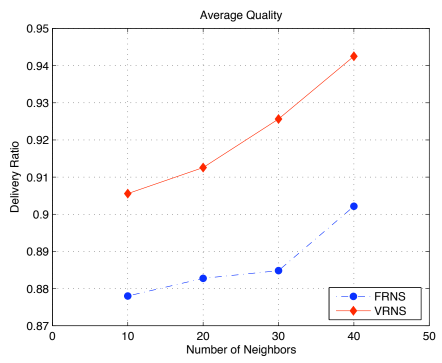

Fig. 6. Average Quality with Bandwidth Ratio of 15% 25% 60%

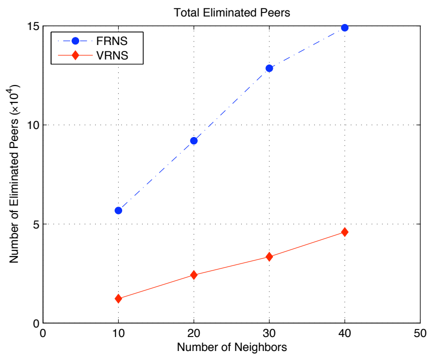

Fig. 7. Total Eliminated Peers with Bandwidth Ratio of 15% 25% 60%

5) Peer Dynamics: The elimination of a peer is based on a threshold condition, such as that, a peer should have the minimum receiving rate of more than 15 Kbps when the streaming rate is set to 300 Kbps. As shown in Figure 7, our *VRNS* algorithm performs better (around 60% decrease) than *FRNS* and shows a consistent trend when we have an increase in the link size. This is so because in the case of *FRNS*, the total number of links for a peer is fixed and so the low bandwidth peers have the same number of neighbors as the high bandwidth peers, which obviously becomes a problem in streaming for low bandwidth peers because the buffer size is the same for all the peers in the network. So if these peers have large buffer sizes in order to stream the data properly, then they would suffer from large packet delays. But in this work we maintain the same buffer size and try to achieve a better streaming rate at individual peers. Thus *VRNS* distributes these links appropriately based on the ratio, which helps in less number of peers getting eliminated.

V. Conclusions

Live P2P streaming networks, as a promising Internet application, have attracted a lot of research interest. In this paper, we have provided *VRNS* an improved neighbor-selection algorithm for unstructured, mesh-based data-driven networks over the *FRNS* a fixed random-neighbor selection algorithm. We have shown that neighbor assignment based on the ratios of upload capacities of the peers can significantly achieve a better performance. An implementation of this algorithm is possible if we have rich information about the network such as the number of different types of peers etc. We have not derived the optimal values of the number of neighbors assigned and would like to address this challenge in our future work.

Acknowledgements

The authors would like to thank Meng Zhang (Tsinghua University, China) whose efforts helped make this study possible.

References

[1] Peer-to-peer streaming simulator, available online at: http://media.cs.tsinghua.edu.cn/∼zhangm/.
[2] K. C. Ellen, W. Zegura, and S. Bhattacharjee. How to model an internetwork. In *IEEE Infocom*, 1996.
[3] Jagannath Ghoshal, Lisong Xu, Byrav Ramamurthy, and Miao Wang. Network architectures for live peer-to-peer media streaming. Technical report, University of Nebraska-Lincoln, http://lakota.unl.edu//facdb/TechReportArchive/TR-UNL-CSE-2007-0020.pdf, August 2007.
[4] Gridmedia. http://www.gridmedia.com.cn.
[5] J. Liang and K. Nahrstedt. DagStream: Locality aware and failure resilient peer-to-peer streaming. *Proceedings of MMCN*, January 2006.
[6] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng. AnySee: Peer-to-peer live streaming. In *Proceedings of IEEE INFOCOM*, April 2006.
[7] N. Magharei and R. Rejaie. PRIME: peer-to-peer receiver-driven mesh-based streaming. In *Proceedings of IEEE INFOCOM*, May 2007.
[8] R. Rejaie and S. Stafford. A framework for architecting peer-to-peer receiver-driven overlays. In *Proceedings of NOSSDAV*, June 2004.
[9] D. A. Tran, K. A. Hua, and T. Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proceedings of IEEE INFOCOM*, March 2003.
[10] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang. Optimizing the throughput of data-driven peer-to-peer streaming. *Under review, available online at http://media.cs.tsinghua.edu.cn/∼zhangm/*, 2006.
[11] X. Zhang, J. Liu, B. Li, and T. P. Yum. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of IEEE INFOCOM*, March 2005.