Statistically Quality Assured Streaming Architecture For Dynamic Peer To Peer Networks

A. Rajesh[1], S. Ravi[2]

Department of Information Technology, Padmasri B.V Raju Insitute of Technology, Hyderabad, A.P, INDIA.
rajesh.atteli@gmail.com, ravi.s@bvrit.ac.in

Abstract

In a peer to peer streaming system, the server usually provides multiple channels. Peers may form multiple groups, each corresponding to a channel for content distribution. A peer can freely switch from one channel to another. We propose a shared or distributed overlay framework (called SMesh or subset mesh) for dynamic groups where users may frequently hop between different groups. SMesh first builds a relatively stable mesh consisting of all hosts for control messaging and supports dynamic joining and leaving host in between the groups. And will guide the construction of overlay delivery trees. Through simulations on Internet, we show that SMesh achieves low delay and low link stress with efficient and low cost maintenance. Secondly we focus on providing statistically guaranteed streaming quality at channel level and individual peer level.

Keywords

Shared overlay mesh for peer to peer network, Delivery trees, Bandwidth allocation, Admission control, Chunk scheduling

1. Introduction

Peer-to-peer streaming has emerged as viable business model and systems architecture for Internet-scale applications [1]. It is an effective way to build applications that connect millions of users across the globe without reliance on specially deployed (centralized) servers.

Peer-to-peer (P2P) streaming has been widely deployed over the Internet. With the penetration of broadband Internet access, there has been an increasing interest in media streaming services. Recently, P2P streaming has been proposed and developed to overcome the limitations of traditional server-based streaming. P2P streaming system cooperative peers self-organize themselves into an overlay network via unicast connections and adapt to changing peer population.

For example fig, one of the most popular P2P streaming systems, PPLive streaming system (www.pplive.com), has provided over 400 channels. The total number of peers in PPLive during a day varies from around 50 thousand to 400 thousand, and the number of peers in a single channel, e.g., CCTV1, varies from several hundred to several thousand.

DOI : 10.5121/ijma.2011.3406

In these applications, as peers may dynamically hop from one group to another, it becomes an important issue to efficiently deliver specific contents to peers. One obvious approach is to broadcast all contents to all hosts and let them select the contents. Clearly, this is not efficient in terms of bandwidth and end-to-end delay; especially for unpopular channels. Maintaining a separate and distinct delivery overlay for each channel appears to be another solution. However, this approach introduces high control overhead to maintain multiple dynamic overlays. When users frequently hop from one channel to another, overlay reformation becomes costly and may lead to high packet loss.

Most of the literatures on peer-to-peer (P2P) live streaming focuses on how to provide *best-effort streaming quality* by efficiently using the system bandwidth; however, there is no guarantee about the provided streaming quality.

The issues mentioned in above two paragraphs are sorted out in this paper. In section 2, we propose a single shared overlay mesh construction for streaming process and in section 3, we propose algorithms which guarantees streaming quality statistically.

2. A Single Shared Overlay Mesh For Peer To Peer Streaming In Dynamic Groups

In this application, we consider building a data delivery tree for each group. To reduce tree construction and maintenance costs, we build a single shared overlay mesh. The mesh is formed by all peers in the system and is, hence, independent of joining and leaving events in any group. This relatively stable mesh is used for control messaging and guiding the construction of overlay trees. With the help of the mesh, trees can be efficiently constructed with no need of loop detection and elimination. Since an overlay tree serves only a subset of peers in the network, we term this framework Subset-Mesh, or SMesh.

Our framework may use any existing mesh based overlay network. In this application, we use Delaunay Triangulation (DT) [2]. The traditional DT protocol has the following limitations: Inaccuracy in estimating host locations, Single point of failure, Message looping. We propose several techniques to improve the DT mesh, e.g., for accurately estimating host locations and distributed partition detection. The two important issues in construction SMesh: *Mesh formation and maintenance, Construction of data delivery trees.*

SMesh does not rely on a static mesh. In the case of host joining or leaving, the underlying DTmesh can automatically adjust itself to form a new mesh. The trees on top of it will then accordingly adjust tree nodes and tree edges. Also note that in SMesh a host may join as many groups as its local resource allows. If a host joins multiple groups, its operations in different groups are independent of each other.

2.1 Mesh formation and maintenance

*SMesh uses GNP to estimate host locations in the Internet space and builds a DT mesh based on the estimated host coordinates. Since GNP estimation is based on network distances between hosts, the resultant mesh can achieve lower end-to-end delay than the traditional DT mesh.

*SMesh uses a distributed algorithm to detect and recover mesh partition, thereby eliminating the need for a central server from the system.

*The distributed algorithm is able to detect whether a message destination is in a partitioned mesh or not and hence solves the message looping problem.

GLOBAL NETWORK POSITIONING (GNP)

GNP estimates host coordinates in a multidimensional Euclidean space such that the distance between two hosts in the Euclidean space correlates well with the measured roundtrip time between them [3]. In GNP, a few hosts are used as landmarks. Landmarks first measure the round-trip time between each other and forward results to one of them. The landmark receiving results uses the results to compute the landmark coordinates in Euclidean space. He coordinates are then disseminated back to the respective landmarks. More specifically, to estimate landmark coordinates, he following objective function is minimized.

$$J_{landmark}(L_1, L_2, \ldots, L_M) = \sum_{L_i, L_j \in \{L_1, \ldots, L_M\} | i>j} \left(\|L_i - L_j\| - RTT(i,j) \right)^2$$

Where M is the number of landmarks, L_i and L_j are the coordinates of landmarks i and j in the Euclidean space, and RTT (i,j) is the round-trip time between i and j. As shown, *Jlandmark* is the sum of the estimation error between the measured round-trip time and the logical distances in the Euclidean space among the landmarks. Therefore, we seek a set of landmark coordinates such that the sum is minimized. If there are multiple sets of {L1; L2; . . . ; LM} to minimize *Jlandmark*, any one set can be used.

Given the landmark coordinates, a normal host estimates its coordinates by minimizing a similar objective function:

$$J_{host}(H_u) = \sum_{L_i \in \{L_1, \ldots, L_M\}} \left(\|H_u - L_i\| - RTT(u,i) \right)^2$$

Where H_u is the coordinates of host u, and RTT (u,i) is the measured round-trip time between host u and landmark i. Note that landmarks do not have to be permanent. It is easy to modify to remove a failed landmark or add a new landmark. Each host can obtain its coordinates by pinging *O* (1) landmarks and using *O* (1) messages. It is highly efficient and scalable.

DELAUNAY TRIANGULATION (DT)

In the traditional DT protocol, each host knows its geographic coordinates [2]. Hosts form a DT mesh based on their geographic coordinates. Compass routing, a kind of local routing, is then used to route a message along the mesh [4]. In this approach, a host only needs to know the states of its immediate neighbors to construct and maintain the mesh, and the mesh is adaptive to dynamic host joining or leaving.

DT protocol connects hosts together so that the mesh satisfies the DT property, i.e., the minimum internal angle of the triangles in the mesh is maximized [5], [6]. Here angles are computed according to the coordinates of hosts as in traditional geometry. It has been shown that a mesh formed in this way connects close hosts together. We illustrate the triangulation process in Figure 1. Suppose that hosts a; b; c, and d form a convex quadrilateral *abcd.* Two possible ways to

triangulate it are shown in Figs. 1a and 1b, respectively. Clearly, the minimum internal angle of Δabc and Δacd is smaller than that of Δ*abd* and Δ*bcd*. DT protocol then transforms the former configuration into the latter one. To achieve this, a host periodically sends *HelloNeighbor* messages to its neighbors to exchange their neighborhood information. It removes a host from its neighbor list if the connection to that host violates the DT property. Similarly, a host adds another host into its neighbor list only if the addition does not violate the DT property. Given a set of N hosts in the network, a DT mesh among them can be constructed with *O (N log N)* messages. The detailed construction mechanism and complexity analysis can be found in [5].

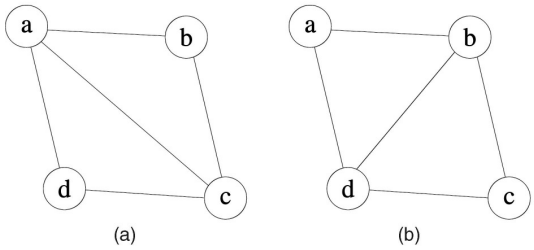

Figure 1. (a) Two adjacent triangles in a convex quadrilateral (Δabc and Δadc) violate the DT property and (b) restore the DT property by disconnecting a from c and connecting b and d.

In each connected DT mesh, a host is selected as the leader, which periodically exchanges control messages with the server. If the mesh is partitioned, more than one host will claim to be leaders. The server then requests them to connect to each other.

DISTRIBUTED ALGORITHM FOR PARTITION DETECTION AND RECOVERY

We now present a distributed algorithm to detect and recover mesh partition. We define some notations as follows. Given a graph, define *L+abc* as the clockwise angle from edge ab to edge bc and *L-abc* as the counterclockwise angle from edge ab to edge bc. They are both between 0 degree and 360 degrees (i.e., the angle is not negative). We further define the undirected angle *Labc* as the smaller one of *Labc* and *Labc*, which is certainly between 0 degree and 180 degrees. We show the examples of *L+, L-,* and *L* in Figure 2. respectively.

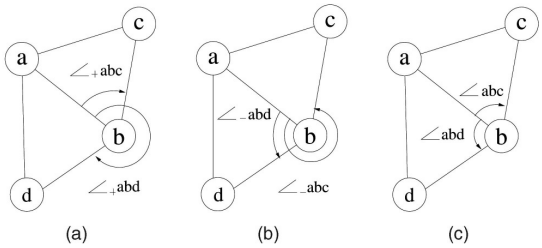

Figure 2.Examples of (a) clockwise angles *L+*, (b) counterclockwise angles *L-*, and (c) undirected angles*L*.

We further consider two connected hosts b and c, and another host a in the graph (whether a is a neighbor of b or c is irrelevant here). We say that c is the clockwise neighbor of b with respect to a if and only if *L+abc* is less than 180 degrees and is the minimum among all the neighbors of b (i.e., $L+abc \leq L+abx$; for all x ϵ neighbors of b). In this case, we write *N+b,a=c* (one can imagine that the edge ba with b fixed, when sweeping clockwise by less than 180 degrees would first

touch c among all b's neighbors). For example, in Fig. 2a, c is the clockwise neighbor of b with respect to a *(i.e., N+b,a=c).* Similarly, we say that c is the counterclockwise neighbor of b with respect to a, denoted as *N-b,a=c*, if and only if *L-abc* is less than 180 degrees and is the minimum among all the neighbors of b. In Fig. 2b, d is the counterclockwise neighbor of b with respect to a *(i.e., N-b,a=d).* Note that host b may not have any clockwise neighbor (or counterclockwise neighbor) with respect to a. For example, in Figure 2, host b does not have any clockwise neighbor with respect to c, since angles *L+cbd* and *L-cba* are larger than 180 degrees.

Theorem 1*: Given the above definitions and host coordinates, a host u detects that a destination t is partitioned from the mesh if and only if one of the following conditions is satisfied:*

1. $N^{+}_{u,t} = \emptyset$, *or*
2. $N^{-}_{u,t} = \emptyset$, *or*
3. $\angle_{+} N^{-}_{u,t} u N^{+}_{u,t} > 180°$, *or*
4. $\angle_{+} N^{+}_{u,t} t N^{-}_{u,t} > 180°$.

Proof: There are two possible cases for t's location:

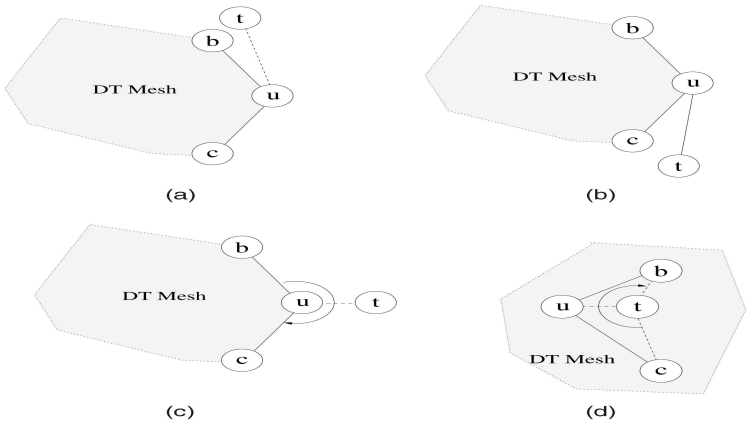

Figure 3 (a), (b), and (c) Host u is on the boundary of the overlay mesh, and host t lies outside the mesh. (d) u is interior host of the mesh. Host t lies inside Δubc but doesn't belong to the mesh.

* t is outside the mesh (see Figs. 3(a,b,c)). By definition, a DT mesh is a convex polyhedron where only the external angles are larger than 180 degrees. As t falls outside the mesh, a message with destination t must be finally forwarded to a boundary host u in the mesh. The possible positions of t are given in Figs. 3(a,b,c) where hosts b and c are two neighbors of u on the boundary of the mesh. Fig. 3a corresponds to condition 1. Fig. 3b corresponds to condition 2. Fig. 3c corresponds to condition 3, where *L+N-u,tuN+u,t* is as indicated.
* t is in the interior of the mesh (see Fig. 3d). If t is in the interior of the mesh, the position of t must fall inside a certain triangle Δubc (as shown in Fig. 3d). When a host u receives a message with destination t, if it finds that *L+N+u,ttN-u,t* > 180 degrees and there is no connection with t, it can conclude that t is not in the mesh.

Therefore, a host u checks whether the destination has been partitioned from its mesh before forwarding a message. If so, u directly forwards the message to t to avoid message looping, and

asks t to join the mesh through itself (using the joining mechanism below) so as to recover the partition.

JOINING MECHANISM (RECOVERY)

A joining host, after obtaining its coordinates, sends a *MeshJoin* message with its coordinates to any host in the system. *MeshJoin* is then sent back to the joining host along the DT mesh based on compass routing. Since the joining host is not a member of the mesh yet, it can be considered as a partitioned mesh consisting of a single host. The *MeshJoin* message finally triggers the partition recovery mechanism at a particular host in the mesh, which helps the new host join the mesh. We illustrate the host joining mechanism in Figure 4. Suppose that u is a joining host.

The following steps show how u joins the mesh (corresponding to Figure 4):

1. u first retrieves the list of landmarks by querying a host b with a *GetLandmark* message.
2. Then u measures the round-trip time to the landmarks and estimates its coordinates.
3. After that, u sends a *MeshJoin* message to b.
4. The message is then forwarded from b to c based on compass routing.
5. Since u falls into Δacd; c knows that u is in another partitioned mesh. c then adds u into its neighbor listNc to recover the partition. Note that the minimum internal angle of Δauc and Δabc is less than that of Δbuc and Δabu. Therefore, the connection between c and a violates the DT property, and c will remove a from Nc and notify a to remove the connection. c then broadcasts its neighborhood information to its neighbors through *HelloNeighbor* messages. (In DT, each host needs to periodically send *HelloNeighbor* messages to its neighbors to exchange the neighborhood information.)
6. Upon receiving *HelloNeighbor* messages from c; b, and d discover u. They add u into their neighbor lists since such connections do not violate the DT property. In the meantime, u also discovers b and d and adds them into its neighbor list. Suppose that b is the next to broadcast *HelloNeighbor* messages. Upon receiving the message, a discovers u and adds u into its neighbor list.
7. The resultant overlay mesh after the joining of u still satisfies the DT property.

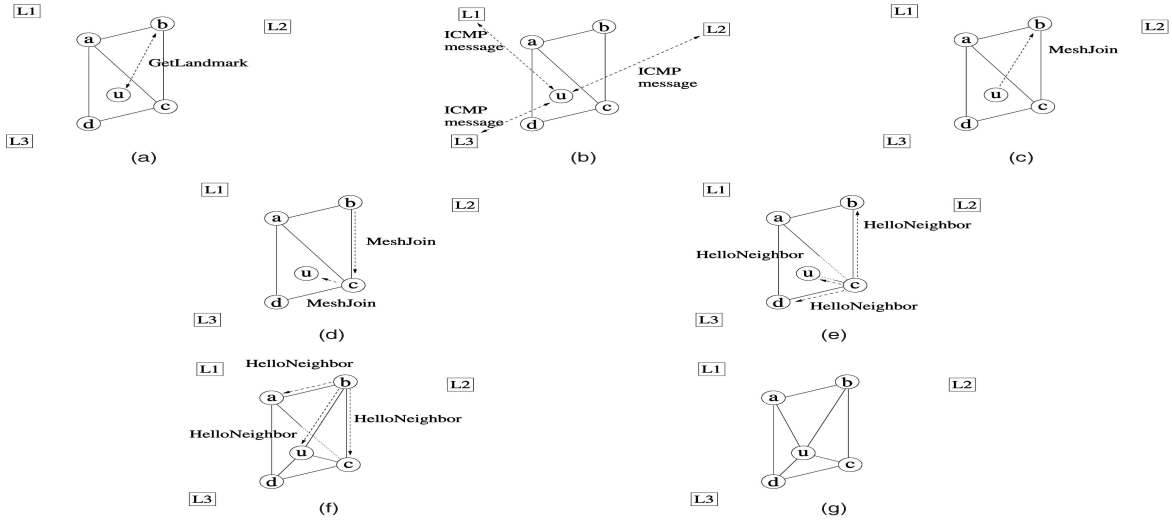

Figure 4. An example of host joining in SMesh.

2.2. Construction of data delivery trees

We construct source specific data delivery trees on top of the mesh. The first type of tree is called an embedded tree, where all tree edges are part of the overlay mesh. When forming the tree, nonmember hosts may be included. The second one builds an overlay tree that covers only group members without having to use mesh edges. We call it a bypass tree. All tree nodes in a bypass tree are members of the group. This is similar to traditional overlay tree construction, where a node relays packets only for other members in its groups. However, the construction of a bypass tree has to rely on the underlying mesh. The third one is termed as intermediate tree, which lies between embedded and bypass trees. In the following, we call a nonleaf host in an overlay tree a forwarder, which needs to forward data messages to its children in the tree. We elaborate the details as follows:

Embedded Tree: To join an embedded tree, a joining host first sends a *TreeJoin* message to the group source along the DT mesh using compass routing. All hosts along the message routing path become forwarders for the tree no matter whether they are group members.

In Algorithm 2, we show how the *TreeJoin* message is handled by a host in the mesh: A host first adds the joining host into its children table for the specified group. Then, it checks whether itself is already a forwarder of the group. If so, the host has already started to forward messages for the tree and known the overlay path along the mesh to the group source. So, it suppresses the forwarding of the *TreeJoin* message and does nothing. Otherwise, it turns itself into a forwarder and relays the *TreeJoin* message to the group source. The *TreeJoin* message will eventually discover a path along the mesh to the group source.

Algorithm 2.

```
TREEJOINHANDLER_EMBEDDEDTREE (TreeJoin)
1  Me.Child[TreeJoin.InterestGroup] ← Me.Child[TreeJoin.
   InterestGroup] ∪ TreeJoin.JoinHost
2  if TreeJoin.InterestGroup ∉ Me.InterestGroups
3     then Me.InterestGroups ← Me.InterestGroups ∪
          TreeJoin.InterestGroup
4          TreeJoin.JoinHost ← Me
5          CompassRoute(TreeJoin, TreeJoin.GroupSource);
```

Bypass Tree: All forwarders in a bypass tree are the group members. Similarly, in order to join a bypass tree, a joining host needs to send a TreeJoin message to the group source using compass routing.

In Algorithm 3, A nonmember host receiving the TreeJoin message simply relays the message to the next hop without turning itself into a forwarder. Such a host will not forward data packets for the group in the future. On the other hand, if the host receiving the message is a member of the group, it accepts the joining host as its child by adding the joining host into its children table. Clearly, such a host has already joined the tree and known the path to the group source. So, it stops forwarding the TreeJoin message.

Algorithm 3.

```
TREEJOINHANDLER_BYPASSTREE (TreeJoin)
1 if TreeJoin.InterestGroup ∈ Me.InterestGroups
2   then Me.Child[TreeJoin.InterestGroup] ←
     Me.Child[TreeJoin.InterestGroup] ∪ TreeJoin.JoinHost
3   else CompassRoute(TreeJoin, TreeJoin.GroupSource);
```

Intermediate Tree: We observe that an embedded tree requires the participation of nonmember hosts, and a host may need to serve multiple hosts of different groups. As compared to a bypass tree, it consumes more network resources and suffers from higher delay, especially for sparse groups. On the other hand, a host in a bypass tree may have a high node stress and heavy load for data forwarding (e.g., a star-like topology rooted at the source for a sparse group). Therefore, we propose an intermediate tree which trades off between an embedded tree and a bypass tree. In an intermediate tree, a nonmember host is included in the tree if it receives more than a certain number of joining messages. Such a host resides in many routing paths, and we expect high delivery efficiency by including it in the tree.

In Algorithm 4, we show how the TreeJoin message is handled by a host: A host handles the message as in a bypass tree if the number of received messages is less than a certain threshold. Otherwise, the host forwards the message as in an embedded tree.

Algorithm 4.

```
TREEJOINHANDLER_INTERMEDIATETREE (TreeJoin)
1 Received Message ← Received Message + 1
2 if Received Message ≤ Message Threshold
3    then TreeJoinHandler_BypassTree (TreeJoin);
4    else TreeJoinHandler_EmbeddedTree (TreeJoin);
```

We give three illustrative examples of the trees. Overlay trees are inherently looped free. This is because compass routing in DT is a greedy algorithm, where the distance from a host to the message destination strictly decreases along the path [4]. As a result, in a data delivery tree, the distance from the source to a host is always smaller than the distance from the source to any of its descendants. This property leads to the loop-free characteristic of SMesh trees.

One possible issue of bypass tree is that a host may have a high node stress by having many children. This is also an issue for intermediate tree. As a comparison, an embedded tree does not have this issue. Because all edges of an embedded tree are part of the mesh, while in a DT mesh, a host has on average six neighbors. In order to address this issue in bypass and intermediate trees, it is possible to set a degree bound for each host. If the number of children of a host reaches its degree bound, the host will not accept new joining hosts as its children. Instead, it forwards new joining hosts to other hosts in the tree (e.g., its parent). Clearly, this is a trade-off between fan-out and performance. It may incur a higher delay.

PATH AGGREGATION FOR QOS PROVISIONING

We note that the traditional DT protocol may result in high network resource consumption. For example, if host a belongs to domain A, and hosts b and b` belong to domain B, usually the

delays of interdomain paths ab and ab` are much higher than that of intradomain path bb`. In other words, angle *L*bab` is small. As a result, using compass routing, if either b or b` is a child of a, the other one is also likely to be a child of a. Therefore, two independent connections across domains A and B are set up, which leads to high usage of long paths and hence high network resource consumption. Furthermore, in the traditional DT protocol, a host may have many children. However, a host often has a node stress threshold K for each group depending on its resource. To address these problems, we require that the minimum adjacent angle between two children of a host should exceed a certain threshold T. If the condition on K or T is violated, SMesh modifies its overlay tree through aggregation and delegation.

Algorithm 5
:

PATHAGGREGATION(u)
1 $[c, c'] \leftarrow$ a pair of children with the minimum adjacent angle
2 **while** $\angle cuc' < T$ OR number of children $> K$
3 **do if** $\|c - s\| < \|c' - s\|$
4 **then** delegate c' to c
5 **else** elegate c to c'
6 $[c, c'] \leftarrow$ a pair of children with the minimum adjacent angle.

Consider a source s and a host u in the network. Once u accepts a child, u checks whether its node stress exceeds K or whether the minimum adjacent angle between its children is less than T. If so, it runs the path aggregation algorithm, as shown in Algorithm 5. It selects a pair of children with the minimum adjacent angle and delegates the child farther from the source to the other. Note that after aggregation, the overlay tree is still loop free because hosts are still topologically sorted according to their distances from the source. We show an example in Figure 5, where *L*buc is the smallest angle among all u's children. If it is smaller than the threshold T and because 11 s – b 11 < 11 s - c 11, u delegates c to b.

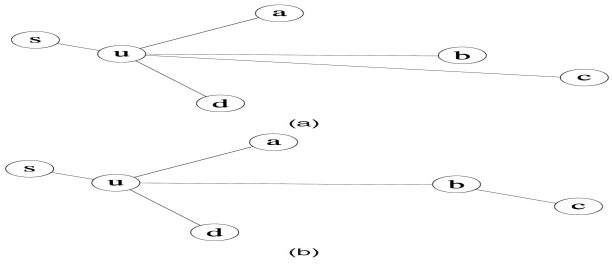

Figure 5. Host u delegates its child c to child b,
since *L*buc < T and | | s - b| | < | | s - c| | .

SMesh avoids tree partition during aggregation by temporarily setting up backup paths. If u delegates its child c to another child c',u would keep forwarding data to c unless it receives an acknowledgment from c'. (This way, a backup path uc is set up.) Backup paths are also present when a host leaves its group. For example, in a bypass tree, a leaving host u sends a *TreeLeave* request to its parent p with the information of its children C= {c1, c2, c3….cm}. P keeps forwarding data to u until p has handled (either accepted or delegated) all the hosts in C (i.e., backup paths uc1, uc2,.. ucm are set up).

- **Illustrative Example**

We show in Figures 6, 7 and 8 how embedded, bypass, and intermediate trees are constructed. White circles in figures denote hosts belonging to the same group. The joining sequence is $\{d,b,f,c\}$ and s is the source.

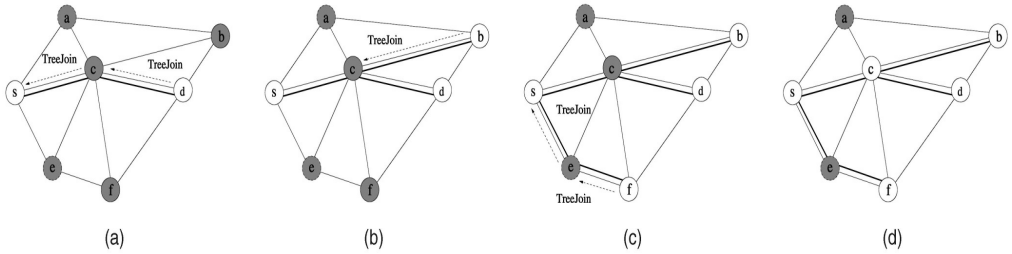

Figure 6. An example of building an embedded tree. Tree branches are indicated by bold lines.

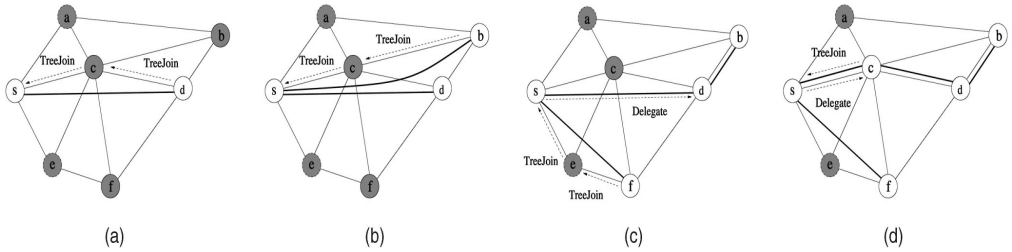

Figure 7. An example of building a bypass tree. Tree branches are indicated by bold lines.

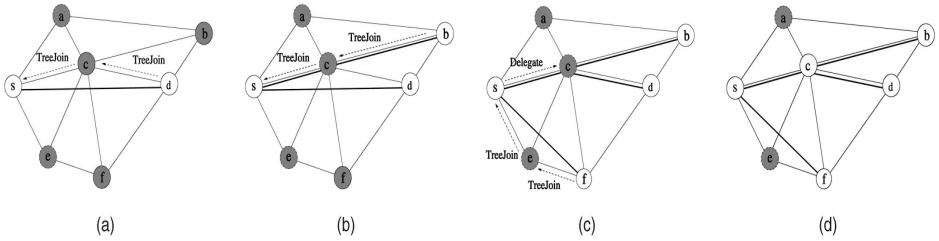

Figure 8. An example of building an intermediate tree. Tree branches are indicated by bold lines.

3. Providing Statistically Guaranteed Streaming Quality For P2P Streaming Systems

There are two different ways to provide statistically guaranteed streaming quality. First at the channel level, this depends on overall bandwidth. Second at the individual peer level, it heavily depends not only on the overall system bandwidth but also on the underlying overlay construction method (we showed the best and cost effective overlay construction in section.1) and block scheduling algorithm.

3.1 Providing channel-level guarantee

We assume that the upload capacity of users is the only bottleneck for a P2P live streaming system i.e., the download capacity of a user is higher than the streaming rate, and bandwidth

bottlenecks are located at the edges instead of the core of the Internet, which are reasonable assumptions [15] for the current Internet. With this assumption, *the statistical bandwidth guarantee problem becomes how to guarantee that the probability for a channel to have sufficient overall upload bandwidth is higher than a threshold.*
In order to achieve statistically bandwidth guarantee, we study a *class of admission control algorithms* which admits or rejects a user based on the information and the channel state.

We are particularly interested in the user-behavior insensitivity of an admission control algorithm, which is whether the algorithm performance is insensitive to the fine statistics of user behaviors including both the distribution of user inter-arrival times and the distribution of user lifetimes. This is because we believe that user-behavior insensitivity is the key to designing an admission control algorithm that is robust and has predictable bandwidth guarantee in a dynamic and heterogeneous P2P system.

we model a channel of a P2P system by the queuing model shown in Figure 9, which captures two fundamental properties of P2P streaming, i.e., heterogeneous upload capabilities and peer churn.
The notation is summarized in Table 1.

Notation	Description
r	streaming rate
c^S	the upload capacity of the streaming server
λ^H	average arrival rate of super users
λ^L	average arrival rate of ordinary users
$1/\mu^H$	average life time of super users
$1/\mu^L$	average life time of ordinary users
c^H	upload bandwidth of a super user
c^L	upload bandwidth of an ordinary user
N^H	number of super users
N^L	number of ordinary users
C	total upload bandwidth of a system
R	total required bandwidth of a system
δ	required bandwidth guarantee probability

Table 1: Notations

Our model considers two classes of users (and can be extended to more classes). Class 1 contains a group of super users each capable of uploading at a high rate of c^H, and class 2 contains a group of ordinary users each capable of uploading at a low rate of c^L. We have $c^H > r > c^L$. A new user arrives at the system randomly with an average rate of λ^H and λ^L for a super user and an ordinary user, respectively. A new user may be admitted or rejected (also called blocked) by an admission control algorithm based on the upload bandwidth of the user and the current state of the system. If a user is admitted, it stays in the system for a random lifetime with average $1/\mu^H$ and $1/\mu^L$ for a super user and an ordinary user, respectively. Each class is modeled as a state-dependent processor-sharing (PS) queuing node [16] shown in Figure 9. The service rate of a queuing node depends on the current node state. For example, the service rate of the super user node (i.e., the top node in the figure) is $N^H \mu^H$, where N^H is the current number of super users.

We say that a system has sufficient upload bandwidth if $C \geq R$, where C and R denote the total upload bandwidth and the total required bandwidth, respectively. The total upload bandwidth C of the system is a function $f_C(\cdot)$ of the current system state as defined below

$$C = f_C\left(N^H, N^L\right) = N^H \times c^H + N^L \times c^L + c^S \qquad (1)$$

where c^S is the upload capacity of the streaming server.

The total required bandwidth R of the system is a function $f_R(\cdot)$ of the current system state as defined below

$$R = f_R\left(N^H, N^L\right) = \left(N^H + N^L\right) \times r \times \varepsilon \qquad (2)$$

where $\varepsilon \geq 1.0$ indicates the control overhead and bandwidth inefficiency of the system, and depends on the underlying overlay architecture and block scheduling algorithm of the system. For example, packet-level simulation results [17] show that ε is about 1.15 for an overlay with mesh-based overlay architecture and a random block scheduling algorithm.

Finally, the statistical bandwidth guarantee problem is to determine whether a new user is admitted or rejected in order to guarantee $\mathbb{P}[C \geq R] \geq \delta$, where δ is the required bandwidth guarantee probability.

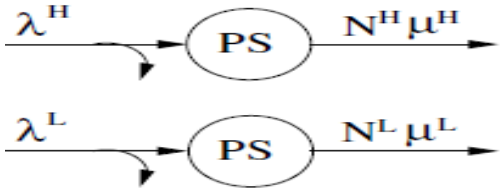

Figure 9 A state-dependent processor-sharing (PS) queuing model for a channel of a P2P live streaming system with two types of users: super users and ordinary users.

ADMISSION CONTROL ALGORITHMS

We propose three admission control algorithms for achieving statistical bandwidth guarantee.

1. Static User Admission Control (SUAC) admits all super users into the channel, and randomly admits an ordinary user with probability β_{SUAC}, where $\beta_{SSUAC} \in [0, 1]$.

2. Semi-Static User Admission Control (SSUAC) admits all super users, and admits an ordinary user if the following condition is true, where $\beta_{SSUAC} \in [0, 1]$.

$$\frac{f_R\left(\mathbb{E}[N^H], N^L + 1\right)}{f_C\left(\mathbb{E}[N^H], N^L + 1\right)} \leq \beta_{SSUAC} \qquad (3)$$

3. Dynamic User Admission Control (DUAC) admits all super users, and admits an ordinary user if the following condition is true, where $\beta_{DUAC} \in [0, 1]$

$$\frac{f_R\left(N^H, N^L + 1\right)}{f_C\left(N^H, N^L + 1\right)} \leq \beta_{DUAC} \qquad (4)$$

Since the upload bandwidth c^H of a super user is greater than the streaming rate r, all three algorithms always admit a super user. But they make different admission decisions for an ordinary user. SUAC is "static" in the sense that its admission decision for an ordinary user does not depend on the current system state (i.e. N^H and N^L), whereas DUAC is "dynamic" in that its admission decision depends on the current system state. SSUAC is "semi-static" since its admission decision depends only on the current state of ordinary users (i.e. N^L) but not on the current state of super users (i.e. N^H).

Below, we compare these three admission control algorithms according to the performance metrics.

• *Implementation Difficulty:* SUAC is the easiest to implement, since it does not need to measure anything. DUAC is the hardest to implement, since it is not trivial to accurately and quickly measure the current N^H and N^L. SSUAC is in the middle, since the average value of N^H can be obtained by using the history in- formation and then it only needs to accurately and quickly measure the current N^L.

• *Blocking Rate:* Intuitively, since DUAC makes a dynamic decision based on the current channel state, it should achieve the lowest blocking rate for a given δ, whereas SUAC makes a static decision, it should achieve the highest blocking rate. The performance of SSUAC should fall somewhere in between. This is verified in the next section by numerical results.

• *Retry Robustness:* Both SSUAC and DUAC are robust in case of user retries, since for a given system state, no matter how many times a rejected ordinary user retries its admission request, it will always be rejected by both SSUAC and DUAC. However, with SUAC, a rejected ordinary user can keep retrying its admission request until it is finally admitted. One possible solution for SUAC is to keep track of all recently rejected users (E.g. their IP addresses).

• *User-Behavior Insensitivity:* Intuitively, since DUAC is more dynamic than SUAC (i.e., more dependent on the channel state), DUAC is more sensitive to the fine statistics of user behaviors than SUAC. Specifically, we have the following insensitivity theorem.

We say that an admission control algorithm is insensitive to the user lifetime distribution, if the steady state distribution of a P2P live streaming system using this algorithm depends only on the average lifetime (i.e. $1/\mu^H$) of super users and that (i.e., $1/\mu^L$) of ordinary users, but does not depend on the lifetime distribution of super users and that of ordinary users. We have the following theorem.

Theorem6 *Under the assumption that a super user arrives as a Poisson process and an ordinary user arrives as a Poisson process, the sufficient and necessary condition for an admission control algorithm to be insensitive to the life- time distribution is that its admission decisions do not depend on the current number of super users (i.e., N^H).*

Proof: Let $a^H(N^H, N^L)$ and $a^L(N^H, N^L)$ denote the arrival rate of admitted super users and that of admitted ordinary users, respectively, when there are N^H supers users and N^L ordinary users.
According to the insensitivity theory of processor-sharing queuing networks developed by Bonald and Proutere [19, 18], the queuing model shown in Figure 9 is insensitive to the user lifetime distribution if and only if

$$a^H(N^H,N^L)a^L(N^H+1,N^L) = a^H(N^H,N^L+1)a^L(N^H,N^L)$$

Since every super user is admitted, we have $a^H(N^H, N^L) = \lambda^H$ for any N^H and N^L, and thus the sufficient and necessary condition for insensitivity becomes

$$a^L(N^H + 1, N^L) = a^L(N^H, N^L)$$

That is, the admission decision is independent of N^H, but can be dependent on N^L.

It is then easy to see that both SUAC and SSUAC are insensitive to the user lifetime distribution under the Poisson user arrival assumption, but DUAC is sensitive to the user lifetime distribution. This implies that the bandwidth guarantee probability achieved by SUAC and SSUAC depends on the user lifetime distribution through the mean only.

We say that an admission control algorithm is insensitive to the user arrival process, if the steady state distribution of a P2P live streaming system using this algorithm depends only on the average arrival rate (i.e. λ^H) of super users and that (i.e., λ^L) of ordinary users, but does not depend on the inter-arrival time distribution of super users and that of ordinary users.

3.2 Providing peer level guarantee

The P2P design philosophy seeks to utilize peer's upload bandwidth for reducing server's workload. However, the upload bandwidth utilization might be suppressed by the so called *content bottleneck* where a peer may not have any content that can be uploaded to its neighbors even if its link is idle. The content bottleneck causes more severe problems in VoD (video on demand) system, due to free user's control (forward, resume, pause etc). To make things worst peers are interested only in a small portion of chunks and their priorities changes more frequently as compared to live streaming. One way to resolve this problem is to compromise user viewing quality. For example, a lower video playback rate has lower peer bandwidth utilization requirement. Allowing a longer playback delay also allows a larger set of chunks to be exchanged among peers. The other solution lies in designing more efficient prefetching strategies and chunk scheduling methods.

In this paper, we propose a differentiated chunk scheduling mechanism that can achieve high peer bandwidth utilization. Using queue-based signaling between peers and the content source server, the amount of workload assigned to a peer is proportional to its available upload capacity, which leads to high bandwidth utilization. In VoD system, the chunks closer to the current playing position have more importance; therefore a queuing model is designed for the segregation of "urgent" and "prefetching" traffic in VoD system. More specifically our paper provides following three fold contributions.

1. We investigate the server's side of peer, and classified the content requests into separate queues. We then proposed different scheduling policies for these queues considering the importance of each type of chunk.
2. We proposed a link sharing mechanism, to prioritize the "urgent downloading" target. The bandwidth sharing among the queues therefore follows a logical pattern.
3. We evaluate the properties of our algorithms, through real test bed.

DISTRIBUTED CHUNK SCHEDULING

The ability to achieve higher streaming rate in P2P VoD system is highly desirable. Higher streaming rate provides better quality and perception of stream. It also provides a cushion to absorb the bandwidth variations caused by peer churn and network congestion. The key to achieve high streaming rate is to better utilize the peer's upload bandwidth.

In this section we propose a differentiated chunk scheduling mechanism that can achieve maximum upload bandwidth of peers in P2P networks. We discuss the scheduling mechanism when peer is acting as a content source or content provider (server side scheduling). We assume a fully connected mesh topology, in which peers sends pull request to obtain the desired content

from other peers or server. The availability of upload capacity is conditional to the queue status. The following sub-sections will explain in detail the proposed differentiated chunk scheduling policy.

A. Server Side Scheduling

The queuing model is specifically designed for the case of co-existence of "*urgent downloading*" and "*prefetching*" requests on each peer. *Prefetching has been proposed as a technique for reducing the access latency*. In this technique, peers prefetch and store various portions of the streaming media ahead of their playing position. Each peer in the overlay providing the content to other peers is considered as content server in our case. We used two different queues for two different types of requests. Before sending a request for chunk, each peer sets an identifier for making distinction between the two types of content requests. On each peer (content server), there is a classifier which checks the *request-type* and sends it to appropriate queue. The urgent downloading target requires higher priority because the requested chunks are closer to the current position of playing window. There is also a scheduler which determines the order of packets to be transmitted from the queues. Figure 10, shows the model of queue based chunk scheduling. In this figure, the serving peer receives different types of chunk requests. These requests are classified into different queues according to their identifier.

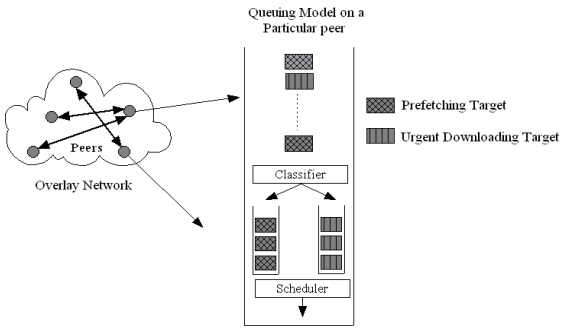

Figure 10. Queuing model in a particular Peer

We used two different types of scheduling policies for each queue. For urgent downloading target, the chunks whose deadline is near to expire have given priority. Thus *earliest deadline first* (EDF) is adapted in this case. This allows the timely availability of chunks to the requesting peer. If a peer requests multiple chunks at different time interval, the latest request will be served while the earlier chunk request would be forwarded to prefetching queue. Let two chunks are requested from a same peer at time t_e and t_c , where t_e is earlier time and t_c denotes the current time. If the difference between the two time (t_e and t_c) is greater than certain threshold, then this situation suggests that peer has performed a seek operation and now it's playing position have been changed. We define the time threshold equals to 10 seconds which is same as the length of window for urgent downloading [20] In this case, we give priority to the chunk closer to current position, thus the latest request (at time t_c) has been fulfilled. This scheduling scheme allows the peer to obtain the chunks nearer to playback position. On the other hand we used simple *first come first serve* (FCFS) policy for prefetching queue. This type of content doesn't have a specific deadline. These content are used to reduce the delay latency when a user performs a seek operation. Therefore FCFS policy is sufficient for this type of content.

B. Bandwidth Distribution

We design a scheduler to determine the order of packets to be transmitted from the queues according to the bandwidth ratio "*br*" for each type of traffic. *The bandwidth ratio "br" represents the amount of bandwidth dedicated to urgent downloading and prefetching*.

Moreover, both classes can borrow bandwidth from each other when one of the two types of traffic is nonexistent or under the limit. This b_r value is also used to calculate the service rate for both types of traffic on that particular peer with b_{ri} and μ_i- b_{ri} being respectively the service rate for urgent downloading and prefetching for peer *i*. μ_i is the total available bandwidth of peer *i*. In order to calculate the value of b_r we monitor the first queue (urgent downloading) in periodic interval. We calculate the total size of data chunks requested and their corresponding deadlines. Let CS_i represents the chunk size requested by peer *i* with deadline t_i then,

$$\text{Bandwidth ratio } (\text{br}_i) = \frac{\sum_{i=0}^{n} CS_i}{\sum_{i=0}^{n} t_i}$$

This value of b_r is used to distribute the upload capacity of the peer among the two types of traffic. The urgent downloading target has higher priority therefore b_{ri} is the outgoing capacity of this link. The remaining bandwidth μ_i-b_{ri} is assigned to the prefetching queue. The peers upload bandwidth doesn't remain constant and fluctuates over time. The periodic calculation of the bandwidth ratio allows handling the dynamicity of the network.

Algorithm 7

Input:
$G = (V, E)$
Chunk request: $r_{i,j}$ for $i, j \in V$
Time Interval : t_i
Upload Bandwidth: μ_i for $i \in V$
Video Chunk : c

Output:
Video Chunk Schedule;

Algorithm:

```
1.    for each r_{i,j} ∈ R
2.      if (ReqType = Urgent)      //Urgent download
3.        Push r_{i,j} to UrgentQueue
4.        Update UrgentQueue (r_{i,j})
5.      else
6.        Push r_{i,j} to PrefetchQueue
7.    for each t_i ∈ T
8.        Calculate bandwidth ratio
9.    for each r_{i,j} ∈ UrgentQueue
10.       Sort r_{i,j} according to deadline
11.       Push C_{j,i} to i
12.   If (μ_i - br > 0)
13.       for each r_{i,j} ∈ PrefetchQueue
14.           Sort r_{i,j} according to FCFS
15.           Push C_{j,i} to i
```

C. Client Side Scheduling

We divide the client buffer window into two different stages; according to play back time of segments as shown in Figure 11 The client side structure is similar to most P2P VoD implementations [20]. The *adjacent stage* contains the segments which are closer to the current playing position of the window. Thus the segments in this window are considered extremely important and therefore given higher priority. The prefetching stage contains the block with the latest playback time. We utilize cooperative prefetching[21] to prefetch the content from different peers. This technique fetches the maximum unavailable segments into session thus reducing the

inter-session transfer delay. The other segments to be prefetched are given lower priority as a request identifier.

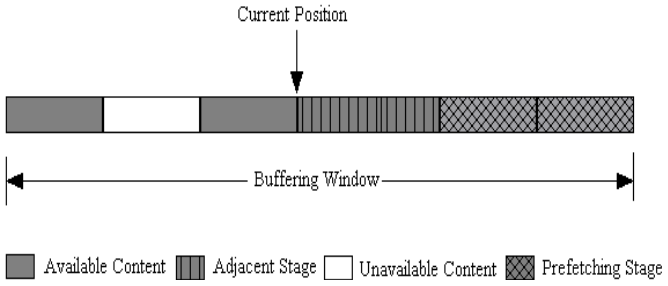

Figure 11 Sliding Window in VoD System

4. Conclusion

In P2P streaming networks, users may frequently hop from one group to another. The proposed novel framework called SMesh serves dynamic groups for Internet streaming. SMesh supports multiple groups and can efficiently distribute data to these dynamic groups. It first builds a shared overlay mesh for all hosts in the system. This stable mesh is then used to guide the construction of data delivery trees for each group. We construct three types of data delivery trees, i.e., embedded, bypass, and intermediate trees. We also propose and study an aggregation and delegation algorithm to balance the load among hosts, which trades off end-to-end delay with lower network resource usage.

To provide statistically guaranteed quality we proposed algorithms at two stages. First one at channel level i.e., admission control algorithms and second one at peer level i.e., algorithm for differentiated queuing and bandwidth allocation.

ACKNOWLEDGEMENTS

I want to thank all my professors, friends for their constant support.

REFERENCES

[1] F John, Buford, Heather Yu, Eng Keong Lua, (2008), *P2P Networking and Apllications*, Morgan Kaufmann publications.

[2] J. Liebeherr, M. Nahas, and W. Si, (Oct 2002) "Application-Layer Multicasting with Delaunay Triangulation Overlays", IEEE J. Selected Areas in Comm., vol. 20, no. 8, pp. 1472-1488.

[3] T.S.E. Ng and H. Zhang, (June 2002), "Predicting Internet Network Distance with Coordinates-Based Approaches", Proc. IEEE INFOCOM '02, pp. 170-179.

[4] E. Kranakis, H. Singh, and J. Urrutia, (Aug. 1999), "Compass Routing on Geometric Networks", Proc. Canadian Conf. Computational Geometry (CCCG '99), pp. 51-54.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, (2000), *Computational Geometry: Algorithms and Applications*, Springer- Verlag.

[6] R. Sibson, (1978), "Locally Equiangular Triangulations", Computer J., vol. 3, no. 21, pp. 243 245.

[7] Y.H. Chu, S. Rao, S. Seshan, and H. Zhang, (Oct. 2002), "A Case for End System Multicast", IEEE J. Selected Areas in Comm., vol. 20, no. 8, pp. 1456-1471.

[8] S.E. Deering, (Aug. 1988), "Multicast Routing in Internetworks and Extended LANs", ACM SIGCOMM Computer Comm. Rev., vol. 18, no. 4, pp. 55-64.

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, (Aug. 2002), "Scalable Application Layer Multicast", Proc. ACM SIGCOMM '02, pp. 205- 217.

[10] W.-P. Yiu, K.-F. Wong, S.-H. Chan, W.-C. Wong, Q. Zhang, W.-W. Zhu, and Y.-Q. Zhang, (Apr. 2006), "Lateral Error Recovery for Media Streaming in Application-Level Multicast", IEEE Trans. Multimedia, vol. 8, no. 2, pp. 219-232.

[11] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, (Apr. 2006), "Resilient Multicast Using Overlays", IEEE/ACM Trans. Networking, vol. 14, no. 2, pp. 237-248.

[12] X. Jin, K.-L. Cheng, and S.-H.G. Chan, (July 2006), "SIM: Scalable Island Multicast for Peer-to Peer Media Streaming", Proc. IEEE Int'l Conf. Multimedia & Expo (ICME '06), pp. 913-916.

[13] X. Jin, W.-C. Wong, and S.-H.G. Chan, (May 2005), "Serving Dynamic Groups in Application-Level Multicast", Proc. Int'l Workshop High Performance Switching and Routing (HPSR '05).

[14] Xing Jin, S.-H. Gary Chan, Wan-Ching Wong, and Ali C. Begen, (Feb. 2010), "A Distributed Protocol to Serve Dynamic Groups for Peer-to-Peer Streaming" , IEEE transactions on parallel and distributed systems, vol. 21, no. 2.

[15] C Wu, B Li, and S Zhao, ((Dec. 2007)), "Characterizing peer-to-peer streaming flows", IEEE Journal on Selected Areas in Communications, vol 25,no 9, pp1612–1626.

[16] T Bonald, and AProutiere, (May 2003), "Insensitive bandwidth sharing in data networks", Queueing Systems, vol 44, no 1, pp69–100.

[17] M Zhang, Q Zhang and S Yang, (2007), "Understanding the power of pull-based streaming protocol: Can we do better?", IEEE Journal on Selected Areas in Communications, vol 25, no 8, pp1678–1694.

[18] T Bonald, M Jonckheere, and A Proutiere, (Jun 2004), "Insensitive load balancing", In Proceedings of ACM SIGMETRICS/Performance (New York).

[19] TBonald, and A Proutiere, (Sep 2002), "Insensitivity in processor-sharing networks", Performance Evaluation, vol 49, no 1-4, pp 193–209.

[20] Bin Cheng et al. (Oct. 2008), "GridCast: Improving peer sharing for P2P VoD", in ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP), Vol 4 , Issue 4.

[21] U Abbasi., T Ahmed, (2009), "COOCHING: cooperative prefetching strategy for P2P video-on-demand system", In: Lecture Notes in Computer Science; "Wired-Wireless Multimedia Networks and Services Management", Springer, Berlin, vol 58, no 42, pp. 195–200.

Authors

A Rajesh, pursuing Masters Degree in Software Engineering from Department of Information Technology, Padmasri B V Raju Institute of Technology. Received Bachelor Degree in Information Technology from TRR Engineering College. Both institutions are affiliated to JNTU Hyderabad, Andhra Pradesh, INDIA.

S Ravi, M.Tech, Asst. Professor, Dept. of IT, Padmasri B V Raju Institute of Technology, JNTU Hyderabad, A.P, INDIA.