

BEAM: A Peer-to-Peer Framework for Live Media Streaming

Darshan Purandare and Ratan Guha

Abstract

Peer-to-Peer (P2P) media streaming has emerged as a scalable method for content distribution in recent years. While recent measurement studies have shown the effectiveness of P2P network in media streaming, there have been questions raised about the Quality of Service (QoS), reliability of streaming services and sub optimal uplink utilization in particular. P2P streaming systems are inherently less reliable because of churn, internet dynamics, node heterogeneity and randomness in the swarm. In this paper, we present a new model for P2P media streaming based on clustering of peers, called *alliances*. In our approach, nodes cluster into alliances for a symbiotic association between them. We show that alliance formation is a loosely coupled and an effective way to organize the peers. We show that our model maps to a small world network, which form efficient overlay structures and are robust to network perturbations such as churn.

We present a simulation based study for our model and provide an empirical analysis under varying workloads and conditions. Evaluation of our model is based on the following metrics: QoS, uplink throughput, fairness in terms of end user's uplink contribution, robustness and scalability of the system. To evaluate efficiency of our model, we compare it with CoolStreaming/DONet and present a quantitative performance evaluation of the aforementioned metrics. Simulation results are promising and show that our model scales well under varying workloads and conditions, delivers near optimal levels of QoS, and for most cases, performs at par or even better than CoolStreaming/DONet.

1 Introduction

With the advent of multimedia technology and broadband surge, there has been an increasing use of Peer-to-Peer (P2P) networks. In particular file sharing, Video-on-Demand (VoD), live event webcasting and streaming of TV channels are more common now. Various paradigms for P2P streaming have been proposed in both academia and industry. However, design flaws and inefficient peering strategy in the earlier works have led to the development of newer P2P streaming models like CoolStreaming/DONet [6] and its derivatives, most of which are built on *chunk-driven* and *loosely coupled* peering philosophy.

Users in excess of tens of thousands are turning to live streaming of popular Asian TV channels through PPLive [2], SopCast [5] and others as of 2007. Though it has been shown that P2P has emerged as a successful medium for live streaming, the quality of service (QoS) and reliability of streaming service still needs additional improvement. Recent measurement studies have revealed the following shortcomings of current day P2P streaming systems:

1) A study [3] on PPLive showed that startup time of video before playback is in order of tens of seconds and sometimes even minutes, and needs to be minimized for a better viewing experience. The study further states that some nodes lag in their playback time by minutes as compared to their peers. We believe that this could be alleviated by a better peering strategy.

2) Another measurement study [4] on PPLive and SopCast has shown that these streaming services lack *tit-for-tat* fairness which leads to uneven distribution of uplink bandwidth among users. The study found that these approaches use greedy algorithms for peering without the consideration of peer locality that leads to huge cross ISP traffic. An important finding of this work is that due to random data distribution structures, the uplink bandwidth utilization is sub optimal.

These limitations serve as a motivation for our current work. In this paper, we attempt to study and counter these issues by proposing a P2P model for live media streaming based on a unique alliance based peering scheme in which nodes cluster in groups, called *alliances*, for mutual node benefit and share the content. Our work mainly focuses on leveraging the randomness of swarm like environments and imposing a few management policies at the node level to reduce the real time packet contention among the nodes. The peering strategy and internal policies in our model are unique as compared to earlier works.

We show that the node topology of our model forms a small world network [7], which are shown to be robust against network perturbations and have small overlay hops between the nodes. To evaluate the effectiveness of our P2P streaming model, we quantify QoS (in terms of jitter free transmission and latency), uplink bandwidth utilization, fairness (in terms of content served), robustness, reliability and scalability of the system. Using simulations, we provide a comparative performance evaluation and an empirical analysis under varying workloads and conditions of our model with Cool-Streaming/DONet (CS) [6] system. We chose to compare our model with CS for the following reasons. First, it is based on swarming technology, uses chunk-driven P2P streaming philosophy and can serve as a benchmark, and secondly many derivatives have evolved out of it that are extremely popular among audiences. Results show that *alliance formation* is an effective way of organizing peers and distributing content in the P2P overlay networks. We show that our model has scaled well while achieving near optimal levels of QoS. We call our model as BEAM (Bit strEAMing).

Related work is presented in Section 2. We describe the details of our model in Section 3. We present a graph theoretic analysis of our model in Section 4. We present the details of our simulation setup, experiments and discuss our results in the in Section 5. Section 6 elucidates impact of our peering strategy on cross ISP traffic. We conclude the paper with an overall discussion and future research directions.

2 RELATED WORK

Notable approaches to media streaming are: 1) Application level infrastructure overlay networks such as Akamai [9]. 2) IP multicast [8]. 3) P2P application layer multicast (ALM) trees such as ESM [1]. 4) Most recent chunk-driven P2P streaming systems based on loosely coupled peering philosophy like CoolStreaming/DONet [6]. Content Distribution Networks (CDN) like Akamai are infrastructure overlay of nodes deployed at various locations on the edge of the networks worldwide to serve the peers. However, owing to the high cost associated with them, they are limited to larger infrastructures. Foremost studies suggested the use of IP multicast, but due to technical and administrative issues, its deployment has been very sparse and is not considered to be a feasible solution. ALM is an alternative to IP mutlicast, wherein multicasting functionality is implemented at the end hosts instead of network

routers. Lately, P2P media streaming has evolved and is found to be a viable solution owing to its ease of use and deployment. Moreover, it can serve a large pool of users without the need for an additional infrastructure.

In P2P ALM, most overlay network construction algorithms form a tree like node topology. This is suitable for reliable network routers, but given the dynamics of internet, churn rate, strict time and bandwidth requirements of media streaming, these algorithms have been found to be less effective and vulnerable to failures. Noted approaches like NICE [10], ZIGZAG [11] and SpreadIT [12] distributively construct an overlay network of nodes and routing functionality to minimize the number of hops for content distribution. The internal nodes in the tree are responsible for forwarding the content and any failure in these nodes causes short term failures including jitters in that particular sub tree before any repair algorithm is used for recovery. End System Multicast (ESM) [1] is a mesh based tree approach to counter the problems in tree like structures. It has the capability to send multiple video streams at different qualities to counter node failures and degrading network links.

PRIME [13] is a mesh based P2P streaming approach to live media streaming that focuses on finding the global content delivery pattern to maximize the uplink utilization while maintaining the delivered quality. Recent P2P model CoolStreaming/DONet [6] is one of the most successful approaches to live media streaming. It is based on a data driven overlay network where a node periodically exchanges data availability information with a set of partners, and retrieves the unavailable data and helps peers with deficient content. New proprietary models like PPLive [2] and SOPCast [5] etc. that are similar to [6] have gained popularity for TV streaming of Asian channels.

BASS [14] is another recent P2P streaming technique for VoD that uses a hybrid approach of BitTorrent (BT) [15] and a client-server model that provides the swarm with an external media server. However, load on such server increases linearly with the number of users owing to its server centric design and hence does not scale well. BiToS [16] is a BT modified approach to VoD using the P2P network. Redcarpet [17] is yet another work that concentrates on providing near VoD and *play as you download experience* using different piece selection algorithms in BT. Since BT has been proven to be near optimal in achieving uplink utilization and mean download time, these approaches have modified BT protocol to suit the VoD needs. In this paper, we attempt to provide an efficient P2P media streaming model based on chunk driven philosophy and unique peering scheme to counter the short-

comings mentioned in [3, 4].

3 BEAM

BEAM consists of three main entities: nodes, a media relaying server and a tracker. Media relaying server is the origin of the stream content in the swarm. The tracker is a server that assists nodes in the swarm to communicate with other peers¹. It also communicates with the media relaying server to exchange important information about the current state of the system. As a new user arrives, it contacts the tracker and submits its IP address together with its bandwidth range. The tracker issues it a peer list, typically 40 nodes, from the set of nodes that are in similar bandwidth range. Alternatively, if it is not available, tracker provides the list of nodes in the closest bandwidth range. Small et al. [18] and Bharambe et al. [24] have shown that interaction of nodes in similar bandwidth range leads to optimal resources utilization in the swarm. The new node requests stream content from the nodes in its peer list, and then starts creating and joining alliances. Alliance formation is explained in detail in Section 3.1.

Since the media relaying server cannot stream the content to multiple users simultaneously due to the bottleneck in its uplink speed, it streams the content to a selected number of peers, termed as *power nodes*, which have higher contribution to the swarm in terms of content served. Initially, when the streaming starts, power nodes are chosen from the nodes with higher uplink bandwidth, since the contribution of nodes is yet undetermined. The power nodes in turn forward the content to the other peers in the swarm.

The tracker periodically (e.g. every 10 minutes) computes the rank of the nodes in terms of the content served to the swarm. If the media server can simultaneously stream the content to, say P nodes, then the P top ranked nodes become the power nodes. The tracker updates the media server about the new power nodes, which are then streamed the media content directly from the server. The rank is calculated on the basis of a *Utility Factor (UF)*, which is a measure of the node utility or contribution to the swarm. UF is computed using two parameters: *Cumulative Share Ratio (CSR)* and *Temporal Share Ratio (TSR)*. Share ratio is the ratio of the uploaded volume content to the downloaded volume content by an end user. CSR is the share ratio of a node since its arrival in the swarm, whereas TSR is the share ratio

¹Nodes and peers have been used interchangeably.

over a recent period of time. Thus, $UF = f(TSR, CSR)$. We formulate UF as follows:

$$UF = \alpha CSR + (1 - \alpha) TSR$$

where α is the weight of CSR and $(1 - \alpha)$ is the weight of TSR . For example, if a node has a $CSR = 2.0$, $TSR = 4.0$ and $\alpha = 0.75$, then $UF = 2.5$. In Section 6.2.7, Table 5 presents possible combinations of (CSR, TSR) values used in determining the power nodes. Only the nodes that have (CSR, TSR) values ≥ 2.0 (empirically obtained from Figure 6(a) and explained later) periodically update the tracker with their (CSR, TSR) . These account for less than 20% of the total nodes in the swarm (see Figure 6(a)). These 20% nodes are enough to generate the required number of power nodes in a streaming session and do not incur significant overhead since the remaining 80% nodes do not report to the tracker. This alleviates the tracker from receiving an overwhelming number of messages from the nodes in the system. We assume that nodes are honest and do not tamper with the data, protocol and the software at the client end. Similar concept of the gauging share ratio of registered users is used in popular BitTorrent clients like Azureus [32].

Since the power nodes are periodically computed based on their UF , they need to perform consistently well in terms of distributing the content to remain as power nodes, else they could be replaced by other well performing nodes. The purpose is two fold: 1) It serves as a natural incentive for the power nodes as well as the non power nodes to contribute to the swarm since this reward helps them to get the content early and directly from the server; the most reliable source in the swarm. Such altruism has been shown to be very effective in improving the overall swarm performance [24, 31]. 2) Nodes with higher uploading capacity are closer to the server. Small et al. [18] has proven that placing peers with higher uploading capacity closer to the source achieves optimal performance in terms of maximizing uplink utilization and minimizing average delay for all the peers in the swarm.

Live media streaming is time and resource constrained. Nodes contend within themselves for the same media content within a short period of time. The need to playback the media and procure the future content necessitates an effective management policy. We introduce the concept of *alliance formation* to counter these problems. Nodes cluster into small groups, typically between 4 to 8, called *alliances*, to form a symbiotic association with other peers. Members of an alliance are assumed to be mutually trusted and help

each other with sharing media content. Our model places an upper bound on two very important parameters: Maximum number of nodes in an alliance, h , and maximum number of alliances a node can join, k . A node can be a member of at most k alliances and this helps the node to form a stronger connectivity in the pool and gives an option to receive the stream content from different alliances (paths).

As a power node receives the media content from the server, it propagates the content within its alliances. While serving the content to its alliance members, a node serves different pieces of a packet² to its peers. A packet contains $(h - 1)$ pieces, which the power node distributes to the other $(h - 1)$ members of the alliance, i.e. each node gets one piece, which it shares with other alliance members and subsequently obtains other missing pieces from other members. In this process, a node downloads $(h - 1)$ pieces and uploads $(h - 2)$ pieces. This is done to leverage the uplink bandwidth of all the peers and make participation necessary so that no node gets a free ride. In case a node cannot get a particular piece because it could not fetch from peer in its alliance, it can request the power node for it. Nodes that only procure the content from alliance members and do not share are ignored by other alliance members in future and alliance member find another node to replace such non contributing node to be in the alliance. As a node gathers all the pieces of a packet, it starts the media playback, forwards the content among its other alliances like power node and procures the future stream content. A node uses *announce mechanism* to notify its alliance members the receipt of a new packet. This process of announcing and exchanging unavailable content can also be efficiently improved using Network Coding [19]. Periodically, nodes in an alliance serve a request that is out of the alliance to bootstrap a new node.

3.1 Alliance Formation

A node creates an alliance by sending an alliance join request packet to the nodes in its peer list. The receiving node can accept the alliance join request or reject it (depending on how many alliances it is currently a member of i.e. k). In case of rejection or no reply, node times out after a short time interval (e.g. 2 round trip times (RTT)) and continues to search for peers

²A packet refers to a collection of pieces and does not refer to an IP packet. A piece is the smallest data unit exchanged.

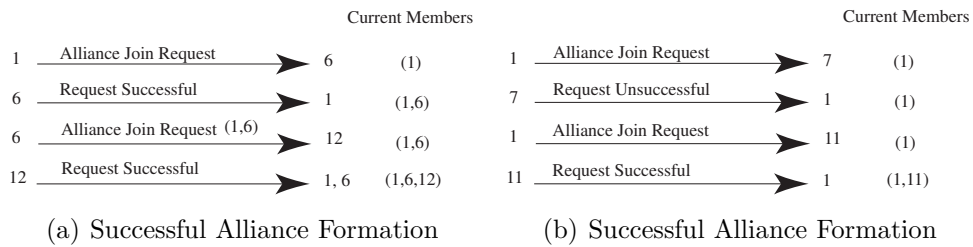


Figure 1: *Alliances Formation in BEAM*

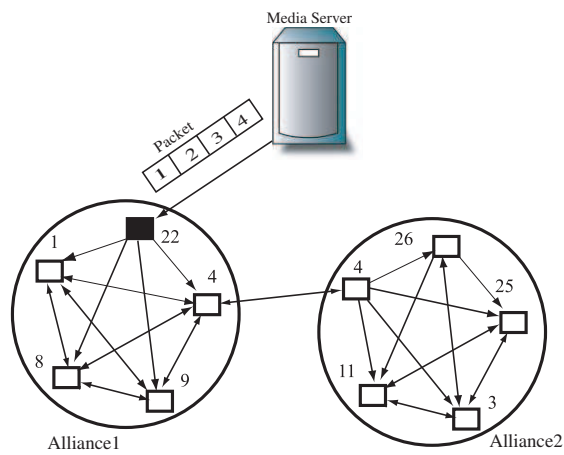


Figure 2: *Alliance Functionality*

to join their alliances. If a node accepts the alliance request, it issues a success packet back to the requesting node. These two members of the alliance can expand and grow the alliance further. The format of a request packet for an alliance is shown by: $[A_{ID}, Num, N_1, N_2, ..]$, where A_{ID} is the ID of the alliance, Num denotes number of current members in the alliance, and N_{id} is the ID of the present member(s) in the alliance. N_1 is the sender of the alliance join request. The format of a success packet is as follows: $[A_{ID}, Self_{ID}]$, where $Self_{ID}$ is the ID of the node that sends the success message to all the alliance members in A_{ID} . Figure 1 depicts the process of alliance formation. In Figure 1(a), following events occur:

- 1) Node 1 sends an alliance request to node 6.
- 2) Node 6 accepts alliance invitation and returns success packet to node 1.
- 3) Node 6 issues a request packet to node 12, that includes: alliance ID and IDs of nodes 1 and 6.
- 4) If 12 joins the alliance, 12 will send success packets to both, 1 and 6. Now all three nodes 1, 6 and 12 are members of the same alliance.

Similarly, in figure 1(b) following events occur:

- 1) Node 1 issues an alliance request to node 7.
- 2) Node 7 does not reply or rejects the request. This could be because node 7 has reached the maximum limit of k . Node 1 times out after a small time interval.
- 3) Node 1 issues request to some other node, say 11.
- 4) If 11 agrees to be part of the alliance, it sends success packet to node 1. Nodes 1 and 11 are members of the same alliance. Nodes expand the alliance till k is reached.

3.2 Alliance Functionality

A node can be member of multiple alliances (at most k). This is important to facilitate multiple paths for a node to obtain the stream content in case of node failures. As a member of an alliance procures a packet, it spreads it among its respective alliances. Consider the scenario in Figure 2, Alliance1 consists of nodes with IDs (1, 4, 8, 9, 22) and Alliance2 has nodes with IDs: (3, 4, 11, 25, 26) with node 4 being a member of both the alliances. Suppose, node 22 obtains a new packet from one of its other alliances or from media server, it then forwards it in Alliance1. It sends an announce packet to its

members as: $[A_{ID}, PNum, NPieces]$, where $PNum$ is the packet number in the streaming and $NPieces$ is the number of pieces in the packet. Nodes (1, 4, 8, 9) request for unavailable pieces that they need to procure to complete the download by sending a request in the form: $[A_{ID}, PNum, P_1, P_2, \dots]$, where P_1 and P_2 are piece number 1 and 2 respectively.

A packet comprises of $(h - 1)$ pieces. If all the members of a particular alliance simultaneously request all the pieces, the forwarding node randomly distributes the pieces of the requested packet among them. It is left to the peers to exchange the pieces within themselves. In case, a node requests specific unavailable pieces in a packet since it has already obtained some pieces from other alliances, the forwarding node sends only those specific requested pieces to avoid any redundancy at the requesting node. In the above example, if members of Alliance1 have procured distinct pieces, they exchange among themselves to complete their individual downloads. As nodes of Alliance1 procure the complete packet, they forward it in their other alliances. In the above case, node 4 (common node in both the alliances) forwards the content in Alliance2 by announcing the arrival of the packet and the subsequent process of forwarding the content is similar as explained above.

While leaving the network, a node sends a departure packet to its alliance members as follows: $[A_{ID}, SelfID, Flag_D]$, where $Flag_D$ is the departure flag. In case a node exits without sending a departure packet, the nodes within the alliance become aware of its inactivity and infer its departure. Other nodes in the alliance continue sharing the streaming content within themselves and/or can find another member for the alliance. In our model, we propose to use TCP connection in BEAM as the network links between peers. TCP detects the peer's departure from the pool and gracefully handles shutdown of the connection link. Li [20] has explained the benefits of using TCP over UDP/RTP, provided the initial buffer time is significantly larger than the Round Trip Time (RTT) between the peers. We elaborate the details in the simulation section.

4 SMALL WORLD NETWORK

In this section, we present an analogy between BEAM and Small World Network (SWN) [7] to show the effectiveness of BEAM's important properties such as near-optimal overlay distance and network robustness in the events of churn and node failures. SWN is a class of random graphs where: 1)

Every node has dense local clustering, i.e a high coefficient of clustering (μ_c , defined below) and some edges with far located nodes. 2) Every node can be reached from every other node by a small number of hops or steps. We present a graph theoretic analysis of our model and show that it generates a swarm of nodes, which when converted to a graph, end users as vertices and connection between them as edges, exhibits small world network characteristics. We compare our results with CS [6] which uses a random network topology. Random graphs [21] are known to generate near-shortest mean distance between all pairs of nodes in a graph.

We chose to show an analogy with small world network for the following reasons: 1) Overlay hops (path length) between any two nodes is short in SWN and partially reflects end to end latency [6,10,11]. 2) High local clustering means a close knit group ; in a media streaming scenario it ensures that once a packet is in the alliance, it can be readily obtained from the alliance members. The important group policies required in an alliance can also be readily applied. 3) SWN are robust to network perturbations like churn and hence provide an efficient overlay structure in events of nodes failures.

4.1 Alliances and Small World Network

μ_c is a local property of a vertex v in a graph and is defined as follows. Consider the vertex v and a set of neighboring vertices $V = (v_1, v_2, \dots, v_n)$ and a set of edges E , where e_{ij} denotes an edge between the vertex i and vertex j . The clustering coefficient(μ_c) of a vertex is the ratio of actual number of edges present to the total possible edges among those vertices:

$$\mu_c = \frac{|e_{ij}|}{\binom{n}{2}} = \frac{2|e_{ij}|}{n(n-1)}$$

In other words, μ_c is density of edges in the node's vicinity. Average of clustering coefficients of all the nodes is the clustering coefficient of the graph. Mean path length is the mean of path lengths between all pairs of vertices in the graph. The concept of SWN is counter intuitive as graphs with higher clustering coefficients would be dense locally and require more hops to traverse the other parts of the graph as compared to a random graph. Watts et al. [7] showed that routing distance in a graph is small if each node has edges with its neighbors (i.e. has high μ_c) as well as some randomly chosen nodes in the swarm. Similarly, Kleinberg [22] proved that if every node in

the swarm shares an edge with a far located node, the number of expected hops for routing between any pair of vertices becomes $O(\log^2 N)$.

Suppose a node is a member of k alliances (a_1, a_2, \dots, a_k) and each alliance has neighbors (m_1, m_2, \dots, m_k) , where $|m_i| \leq h$, and $1 \leq i \leq k$. Therefore, coefficient of clustering for such node would be:

$$\mu_c \geq \frac{\binom{m_1}{2} + \binom{m_2}{2} + \dots + \binom{m_k}{2}}{\binom{m_1+m_2+m_3+\dots+m_k}{2}}.$$

Figure 2 depicts the neighborhood of node 4. It is a member of two alliances and in each alliance, it is connected to four other members. Nodes in an alliance forms a clique (complete subgraph). Node 4 is completely connected to members of Alliance1 and Alliance2, though, members of Alliance1 and Alliance2 may or may not be connected with each other. With respect to Alliance1, node 4 forms four long distance links elsewhere in the network. Similarly, with respect to Alliance2, node 4 forms four long distance links elsewhere in the network. This property is analogous to small-world network, where nodes are well connected locally and also have some long distance links elsewhere in the network that helps to achieve a small path length between all pairs of nodes. Coefficient of clustering for node 4 in this case would be:

$$\mu_c \geq \frac{\binom{4}{2} + \binom{4}{2}}{\binom{8}{2}} = \frac{3}{7} = 0.428$$

We also consider the case that other alliance members can have edges between them. Similarly, other nodes in the graph would have μ_c of at least 0.428 since they have similar constitution of alliance and neighborhood. Clustering coefficient of 0.428 is relatively much higher than a random graph (μ_c for random graph of the same size was found to be 0.0019), and therefore it lies in the region of small world graphs as mentioned in [7].

4.2 Graph Theoretic Properties of Alliance

Graph density (ratio of number of edges to the total number of possible edges in the graph) is an important factor for the connectedness of a graph. We evaluate the graph density of a BEAM graph by abstracting the alliances as nodes. As a member of an alliance receives a packet, it forwards within its alliance members and hence we focus on alliance hops rather than individual

node hops in this scenario and compute the same. To simplify, we consider an alliance as a single node which we call super node. Suppose there are N nodes in the swarm viz (V_1, V_2, \dots, V_N) and they are spread in M alliances. Let D_{graph} be the density of the graph, $D_{alliance}$ be the density of the graph when alliances are abstracted as vertices i.e. super nodes as vertices, M be the number of super nodes in the swarm, O be the outdegree of a super node. Every node in the swarm is connected to $(h - 1)$ other nodes in every alliance and there are k such alliances. Therefore, we have

$$D_{graph} = \frac{\sum_{i=1}^N \sum_{j=1}^k (h_{ij} - 1)}{2 * \binom{N}{2}} = \frac{\sum_{i=1}^N \sum_{j=1}^k (h_{ij} - 1)}{N * (N - 1)}$$

where h_{ij} is the number of members in j^{th} alliance of node i , and $1 \leq i \leq N$, $1 \leq j \leq k$. In a steady state, when all the nodes have formed k alliances, and each alliance has exactly h members, we have

$$D_{graph} = \frac{(h - 1)k}{N - 1}$$

Since super nodes are formed by contracting the alliances, we have

$$M = \frac{N * k}{h}$$

Every super node is connected to other super nodes through its h members and their respective $(k - 1)$ alliances since every node is a member of k alliances each. Therefore, we have

$$O = h * (k - 1)$$

Since there are M super nodes and each has a outdegree of O , there are $\frac{M * O}{2}$ edges. Therefore, we have,

$$D_{alliance} = \frac{\frac{M * O}{2}}{\binom{M}{2}} = \frac{h^2(k - 1)}{(Nk - h)}$$

For $h = 5$, $k = 2$, i.e. node degree = $(h - 1) * k = 8$ and $N = 512$, the D_{graph} is approximately 0.004, while $D_{alliance}$ is approximately 0.025. We see that

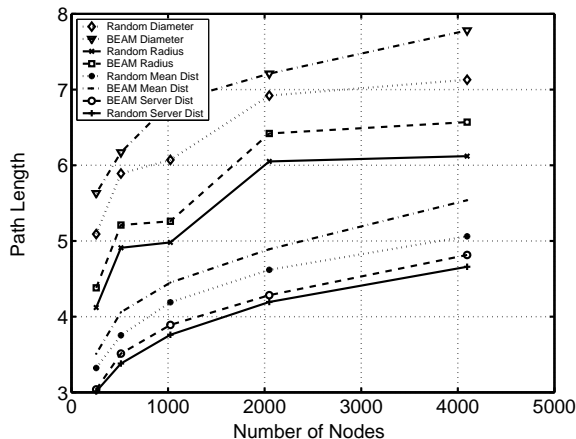


Figure 3: Path length versus Number of Nodes for various graph parameters in Random and BEAM graphs

the density of the graph at alliance level is relatively much higher than at the node level i.e. $D_{alliance} \gg D_{graph}$. The alliance formation and subsequently the topology of the network produces strongly connected graph and reduces the hop count during the communication. Similar abstraction is not possible for complete random graphs.

We are more interested in the mean path lengths from the server to the nodes rather than the mean path lengths between all pairs of nodes. Therefore, we limit our search to the length of all paths from the server to all other nodes in the swarm. To gauge the actual path length in a large swarm, we conducted experiments for finding the average path length between all nodes, average path length from server, radius, and diameter of a graph.

Table 1: A comparison of BEAM, Random and Hybrid graph for 512 nodes with node degree 8.

Graph Type	Diameter	Radius	Mean Distance	Server Distance	Clustering Coefficient
BEAM	6	5	3.37	3.19	0.42
Hybrid	5	4	3.33	2.87	0.014
Random	5	4	3.26	3.16	0.013

Table 1 illustrates results from a simulation in which we compare synthetically generated random graphs with BEAM graph having the same node degree and overall density in the graph. We use networkx python library [23]

for complex networks to obtain our simulation results. In these experiments, we have tested three kinds of graphs: completely random graph, BEAM network graph and hybrid BEAM graph where alliances acts as nodes. The graph density, node degree = 6, and the node count = 512 were the same in all three graphs. Hybrid graph's node count is reduced to $\frac{Nk}{h} = 256$ since $h = 4$, $k = 2$, and the degree of such hybrid nodes is equal to h . Server distance is calculated by picking a random node among the 512 nodes and then calculating distance from it. From the results in Table 1, it is seen that random graphs perform well as expected in all the metrics. BEAM graphs have performed at par with random graphs. Server distance in hybrid graphs is shorter than random graphs. Random graphs have relatively lower mean path length while hybrid graphs have the lowest mean server distance. This abstraction of BEAM graphs helps to analyze the topography and various other graph theoretic properties.

Figure 3 depicts the performance of random and BEAM graphs for higher number of nodes. Random graphs are known to perform better while traversing the graph. The figure shows that it has relatively shorter radius and diameter as compared to BEAM graph. BEAM has nearly matched random graphs in mean distance from the server, which is the most important criteria in our environment. It is an indicative of the overlay hops a node is away from the server. Thus, the BEAM graph forms a small world network with relatively high clustering coefficient and very comparable in mean path length to the random graphs.

5 Metrics

In this section, we define the metrics that are most important in a P2P streaming environment. We quantify these metrics and provide mathematical expressions for the same. In our simulation, we use the following expressions to evaluate our results.

Table 2: *This table explains all the Metrics related terms.*

Term	Expression	Description
Total Number of Nodes	N	Number of Nodes in swarm
Total Number of Packets	T	Total Packet in a Streaming Session
Number of Server Upstream Connection	P	Server can simultaneously upload to P nodes
Total Uploads by node i	UP_i	Volume of Uploads by node i
Total Downloads by node i	$DOWN_i$	Volume of Downloads by node i
Share Ratio of node i	SR_i	$UP_i/DOWN_i$
Media Playback Time at Server	T_{server}	Start Time of Media Playback at Server
Media Playback at Node i	T_i	Start Time of Media Playback at Node i
Piece Availability	F_i	0 if packet available before media playback else 1
Jitter Factor of Node i	J_i	$\left(\sum_{i=0}^T F_i\right) / T$
Latency of node i	L_i	$L_i = T_{Server} - T_{Node_i}$

Table 3: *This Table lists all the Media Streaming Metrics we have used in our simulations.*

Term	Expression	Description
Average Jitter Factor	A_j	$\left(\sum_{i=0}^N J_i\right) / N$
Average Latency	A_l	$\left(\sum_{i=0}^N L_i\right) / N$
Total Uplink Utilization	U	<i>Uplink Used/Uplink Available</i>

Metric 1. Average Jitter Factor = $\left(\sum_{i=0}^N J_i\right) / N$ where,

$$J_i = \left(\sum_{i=0}^T F_i\right) / T$$

$$F_i = \begin{cases} 0 & \text{if packet arrived before media playback} \\ 1 & \text{if packet not arrived before media playback} \end{cases}$$

Here, T and N denote the total packets in the streaming session and the total number of nodes in the swarm respectively.

If a packet is not received at its playback time, it is considered to be a jitter. Average jitter factor is critical to maintain high quality of streaming at user end; lower the jitter rate, better is the QoS. Since, it is averaged out among the total number of end users, it depicts a system wide measure of QoS. We compute the jitter factor for each individual node and then average it to compute the system wide jitter factor. Jitter factor is also called as continuity index in some previous works [6].

Metric 2. Average Latency = $\left(\sum_{i=0}^N L_i\right) / N$ where,

$$L_i = T_{\text{Server}} - T_{\text{Node}_i}$$

$$T_{\text{Server}} = \text{Media playback time at Server}$$

$$T_{\text{Node}_i} = \text{Media playback time at Node}_i$$

The difference in media playback time at user end and server end is the latency. Most live events and their streaming rely on minimizing the latency from the actual playback at server end to keep end users interested. We compute the latency of individual nodes and then average it to derive the system wide measure of average latency.

Metric 3. Uplink Utilization = $\frac{\text{Total Uplink Used}}{\text{Total Uplink Available}}$

The better the uplink utilization, better is the scalability of the system. Uplink bandwidth is the most sparse resource in the swarm and its maximization leads to optimal performance in the swarm in terms of minimizing delay and maximizing number of end users [18].

Metric 4. Fairness Factor = Variance($SR_1, SR_2 \dots, SR_N$) where, SR_i denotes the share ratio of node i and is defined as,

$$SR_i = \frac{\text{Uploads (Node}_i\text{)}}{\text{Downloads (Node}_i\text{)}}$$

Fairness can be defined in several different ways, for e.g. in terms of uplink bandwidth contribution, pricing etc. We believe that in such random swarm environments, it is extremely difficult to deliver services in proportion to their contribution. We define fairness in terms of share ratio of content served by nodes. Share ratio of end users over the period of simulation run depicts the contribution of the nodes quite fairly. Since, it is difficult to provide services in proportion to the contribution, the best we can do is to minimize the variance of share ratios of the nodes in the swarm by enforcing strict policies. An ideal system would have nodes with share ratios of 1.0, where an end user gets streaming content and it passes on equally to other end user. But given the dynamics of the internet, it is very difficult to achieve the same. More the number of nodes close to share ratio of 1.0, the fairer is the system.

Metric 5. Robustness Factor = Maximum(F) where,

$$\begin{aligned} F &= \text{Percent failure of nodes} \\ R_J &= \text{Average Jitter Factor} \\ R_L &= \text{Average Latency} \\ \Delta R_j &= \text{Threshold Jitter Factor} \\ \Delta R_l &= \text{Threshold Average Latency} \end{aligned}$$

and such that,

$$\begin{aligned} R_J &\leq \Delta R_j \\ R_L &\leq \Delta R_l \end{aligned}$$

To evaluate the robustness of BEAM with respect to achieving acceptable levels of QoS, while maximizing the node failure rate in the swarm, we assign a threshold of ΔR_j and ΔR_l to jitter factor and average latency respectively. We test the robustness and reliability of the underlying network architecture under increasing stress test, subjecting the system to varying percentages of node failures or departures. We determine the maximum node failures which the system can withstand, without degrading the QoS.

Metric 6. Scalability Number = Maximum(N) where,

$$\begin{aligned} N &= \text{Number of Nodes} \\ S_J &= \text{Average Jitter Factor} \\ S_L &= \text{Average Latency} \\ \Delta S_j &= \text{Threshold Jitter Factor} \\ \Delta S_l &= \text{Threshold Average Latency} \end{aligned}$$

and such that,

$$\begin{aligned} S_J &\leq \Delta S_j \\ S_L &\leq \Delta S_l \end{aligned}$$

To evaluate the scalability of BEAM with respect to achieving acceptable levels of QoS while maximizing the number of nodes, we assign a threshold of ΔS_j and ΔS_l to jitter factor and average latency respectively. The scalability number indicates the optimal number of users in the swarm, where the threshold is not exceeded and number of users are maximized.

6 SIMULATION SETUP AND EXPERIMENTS

To evaluate various aspects of BEAM which are normally difficult to study using logs of real world torrents or trace, we use packet-level simulation. Though it is difficult to capture all the Internet dynamics correctly in a discrete time event simulator, we model most of the real world aspects in our simulation. Further, we mention the assumptions and simplifications in the context of our experiments and their impact on the results. Table 2 explains all the related terms used in Table 3, which lists all the media streaming metrics we use in our simulation.

6.1 Simulator Details

We simulate both the models i.e. BEAM as well as CS [6] and compare their results based on the metrics defined in Table 3. We simulate all the components of CS i.e. 1) Node join and membership management algorithm. 2) Buffer map representation and exchange. 3) Intelligent scheduling algorithm. 4) Failure recovery and partnership refinement method. In BEAM, we

model the server, tracker functionality, and the nodes in the swarm. Server is the only source of streaming packets in the system. For comparing the two systems, we quantify QoS (in terms of jitter factor and latency), uplink utilization, fairness (in terms of content served by an end user). We also analyze robustness, reliability and scalability of the system by evaluating the QoS of the system under varying workloads and conditions like node failure, churn, larger swarms etc.

We used the BRITE universal topology generator [29] in the Top-Down Hierarchical mode to model the physical network topology of Autonomous Systems (AS) and the routers. All AS are assumed to be in the Transit-Stub manner. Overlay is assumed to be undirected. Unlike other simulators [13,24, 28], we assume that the bottleneck in the network can appear in the access links of source and destination (i.e. first-mile and last-mile hops) as well as the non access links that are in the interior of the network, in particular within or between carrier ISP networks. The nodes in the swarm are assumed to be of heterogeneous bandwidth classes namely:(512Kb, 128Kb), (768Kb, 256Kb), (1024Kb, 512Kb), (1536Kb, 768Kb), (2048Kb, 1024Kb) where first and second member of the tuple are the maximum downlink and uplink speed of a node respectively. The distribution of these bandwidth classes is uniform in the swarm. To simulate the congestion in the Internet, we induce 5% congestion in the non access links within the interior of the network. In such congestion scenarios, the available bandwidth to nodes is the minimum of the bottleneck at source or destination and the bottleneck in the non access links. The delay on inter-transit domains and intra-transit domains are assumed to be 100 ms and 50 ms respectively, while delay on stub-transit is assumed to be 30 ms and intra-stub transit links are randomly chosen between 5ms and 25ms. We simulate the TCP level dynamics like timeouts, slow start, fast recovery and fast retransmission by introducing a delay of 10 RTTs [27]. We model a flash crowd scenario for the arrival of users in the swarm, i.e. all users are present in the swarm when the live media streaming starts, as this is the most relevant and challenging scenario for the P2P streaming system.

In our experiment, the number of nodes typically vary from 128 to 4096 for most cases. For some large sets of experiments we have also considered nodes in excess of 16000. We consider a media file of duration 120 minutes, originating from a source, encoded with streaming rate of 512 Kbps and a file size of approximately 440 MB. In BEAM, we use the values of $(h, k) = (4, 2)$ to make the neighbor count = 6, similar to CS, for a fair comparison. Table 4 provides other values of (h, k) that can be considered for streaming. The

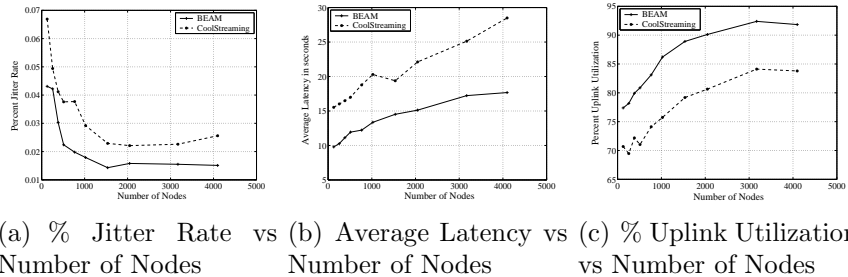


Figure 4: Comparison of QoS in BEAM and CoolStreaming

values of α is 0.75 from Table 5. Each piece size is 64 Kb and hence packet size is $(h - 1) * 64$ Kb = 192 Kb in our case. We maintain similar settings for the remainder of the paper. Any changes in the configuration settings are mentioned at respective sections.

6.2 Results and Discussion

6.2.1 QoS and Uplink Utilization

In the first set of experiments, we compare the effectiveness of alliance theory of BEAM on QoS and uplink utilization as against CS's random peer selection. Figure 4 depict these comparisons. In accordance with the conventional notion of scalability in P2P systems, it can be seen from figure 4(a) that BEAM and CS both perform better with the increasing swarm size, though jitter rate slightly increases after 1500 node mark but stabilizes around 2000 nodes. BEAM has a comparatively lower (approximately 0.01%) jitter rate than CS. The plausible reason is that in CS, the content delivery is random in nature rather than an organized flow. Sometimes an intermediate piece which could not be fetched, may increase the jitter rate. Due to alliance formation in BEAM, the stream content propagates in an organized fashion from one alliance to other; so chances of an intermediate piece missing are comparatively low. In BEAM, every node receives the content through the best possible channel among its various alliances, while the same cannot be commented for CS. An optimal jitter rate of 0 is difficult to attain in such random swarm environments because the content distribution is dynamic and, lasts for an extended time; network anomalies and congestions can cause unavailability of a packet at its playtime.

Figure 4(b) depicts the average latency in both systems. For the same

settings, average latency for BEAM varies from less than 10 seconds for a 128 node swarm to less than 18 seconds for a 4096 node swarm, while CS has considerably higher latency of 29 seconds for a swarm of 4096 nodes. An explanation for this could be that higher the number of hops in the overlay, greater are the chances of increased end to end latency [10,11]. CS has comparatively higher bootstrapping time before playing media and it could be attributed to the following facts: 1) It buffers more pieces in advance before playback. 2) Due to random nature of data exchange, a missing intermediate piece further increases jitter and hence latency. 3) Execution of intelligent scheduling algorithm causes both computation overhead and delay. On the contrary, in BEAM, the systematic flow of content from one alliance to another and near optimal overlay hops account for its lower latency. Moreover, if a packet has been procured by an alliance member, it implies that there are at least one or more sources for the content. This flow of packets indeed saves time as compared to CS. Playback starts 10 seconds after receiving the first segment in [6]. In our implementation of both BEAM and CS, the playback starts after 6 seconds, as 6 seconds of buffer time is long enough and is many times larger than the RTT between peers to counter the network anomalies like jitter and congestion within the network [20].

Uplink bandwidth is the most important resource of a P2P system. End users that are charged for bandwidth used per time unit want to maximize their utilization. Moreover, maximization of uplink bandwidth is a must for a scalable system [18]. From Figure 4(c) it is clear that uplink utilization increases with the swarm size in both of the systems. BEAM has approximately 7% higher utilization and this could be due to the fact that nodes with higher uploading capacity can effectively use their outgoing bandwidth in their other alliances, while the same may not be true for CS where a node with high uplink capacity may remain under utilized due to insufficient requests from its neighbors. In random peering (CS), neighboring peers may or may not request for pieces in the packer, while in an alliance (BEAM), members share pieces in every packet among themselves, ensuring that there are request for upload almost all the time, this increases BEAM's uplink utilization. An optimal utilization of 1.0 is near impossible because of node heterogeneity and lack of download requests from the low capacity peers.

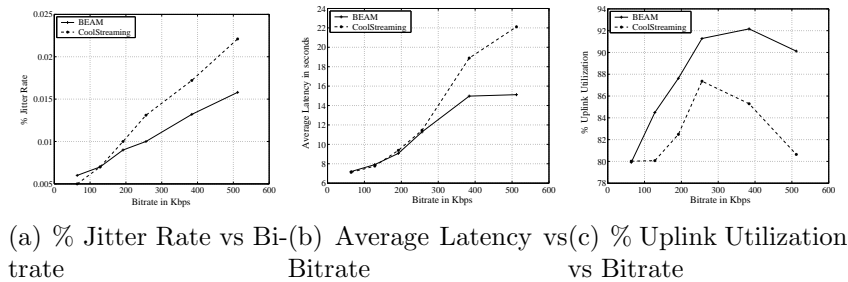


Figure 5: *Effect of Bitrate on the QoS for BEAM and CoolStreaming*

6.2.2 Streaming Rate

We vary the streaming rate from 64 Kbps to 512 Kbps in a 2048 node swarm and expect a comparative deterioration in QoS with increasing streaming rate as the nodes need to procure more content for the same playback time. For lower streaming rates, QoS is expected to be near optimal as additional packets are fetched much before their playtime and chances of jitter and hence latency become negligible. From Figure 5(a), the difference in average jitter rate between BEAM and CS is marginal for lower streaming rates but more prominent for higher streaming rates when the systems are subjected to stress test. In Figure 5(b), similar trends can be observed for average latency while analyzing the effect of varying streaming rate on BEAM and CS. For the same playback time, a node needs to obtain higher number of packets that incurs additional time overhead. Figure 5(c) shows the variation in uplink utilization of both the systems. They both peak around encoding rate of 256 Kbps. A plausible reason could be that at this encoding rate, the nodes are able to cater all the requests at optimum rate and this in turn increases the throughput. Node topology and peering partners are important in analyzing the utilization of bandwidth. Most commercial websites stream at rates between 225 Kbps and 450 Kbps as of 2007. Receiver should have $downlink \geq streaming\ rate$ and sender should have enough uplink to contribute. In our simulation, the bandwidth classes of lowest strata is 512 Kbps, so we have limited our discussion to streaming rate of 512 Kbps. 512 Kbps can be considered as a decent rate, though in near future streaming of DVD quality media will require additional bandwidth.

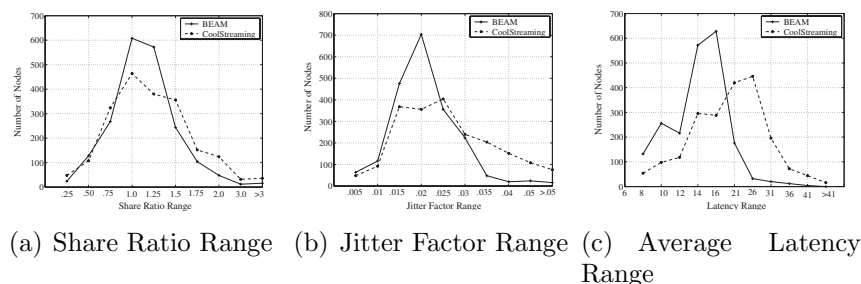


Figure 6: *Number of Nodes vs Share Ratio, Jitter Factor and Latency Range in BEAM and CoolStreaming.*

6.2.3 Fairness

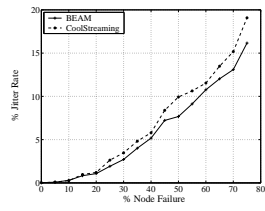
To the best of our knowledge fairness has been undermined in P2P streaming models. [24,25] have addressed fairness issues in Gnutella and BitTorrent like P2P systems. Chu et al. [30] have proposed a tax based model for fairness in P2P streaming. Uplink bandwidth is a sparse and most important resource in P2P streaming swarm and end users resort to methods like freeriding [25], whitewashing [26] etc. in order to save their uplink bandwidth. As a result, many nodes upload much more than what they should while others get a free ride. Recent measurement studies [3,4] confirm the same. In this paper, we quantify fairness in terms of content served by each node (uplink bandwidth) or equivalently by their share ratios. We compute the share ratio of all the individual nodes and analyze the correlation, if any, in the QoS perceived by the nodes. Also, we study the fairness of BEAM and CS towards distributing the load evenly among users.

In Figure 6(a), we depict the share ratios of nodes and their distribution in BEAM and CS. An ideal share ratio of 1.0 is not possible in such P2P systems due to the node bandwidth heterogeneity [3,4]. In such cases, the range of share ratios from 0.75 to 1.25 becomes more significant since it is closest to 1.0. Larger the number of nodes having share ratio close to 1.0, fairer is the system. In BEAM, around 1180 nodes out of 2048 have their share ratios in the range 0.75 to 1.25, which forms 57.61% of the total nodes, while the distribution is more spread out in CS with 41.21% nodes lying in the region of share ratios between 0.75 to 1.25.

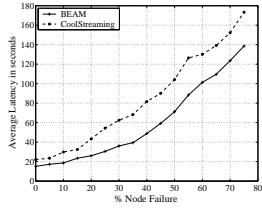
BEAM encourages user participation, wherein nodes in an alliance exchange pieces, i.e. nodes upload the content to their alliance members while

completing their individual downloads. For example, in an alliance of 5 nodes, if one of the nodes procures a packet (either from the server or through some of its other alliances), it forwards the content among its other four members of the alliance. These four members must exchange their pieces amongst themselves. Therefore, a node downloads 4 pieces and at least uploads 3 pieces in its current alliance which makes its share ratio = $3/4 = 0.75$. Further, after downloading the complete packet, it forwards (4 or fewer pieces) the content to its other alliances depending upon the number of requests it has and depending upon if those members have procured some pieces from their other alliances. This explains why BEAM has better uplink utilization and more nodes have share ratio around 1.0. However, some disparity can be seen in Figure 6(a) where some nodes upload more than 3 copies while others share less than one fourth of the entire content. This is because there are very few requests made to the low capacity peers, and power nodes distribute multiple copies of the content in the swarm. In case of CS, there are more nodes with higher share ratios and comparatively lesser nodes with share ratio closer to 1. This could be attributed to the fact that some nodes with high bandwidth always remain forwarding nodes, i.e. they upload much more than the lower bandwidth nodes either because of excess bandwidth or the topology of the node in which flow could be top-down.

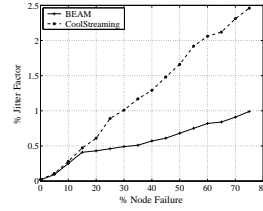
As shown in the Figure 6(b), the average jitter factor is 0.0158% and average latency is 15.12 seconds in BEAM, whereas they are 0.0221% and 22.11 seconds respectively for CS. It can be seen that most nodes in the swarm receive average values of QoS parameters for both the systems, i.e. the nature of graphs in Figures 6(b) and 6(c) are similar and show that nodes contributing fairly to the swarm receive the streaming content with an average jitter factor and latency. This can be seen in Figure 6(b) and Figure 6(c) where certain nodes have larger latencies and comparatively higher jitter rates. These nodes have lower contribution in the swarm. In BEAM, more than 80% nodes have jitter factor in the range of 0.001 to 0.003 and similarly more than 70% nodes have latencies in the range of 10 to 20 seconds. In case of CS, jitter factors are more spread out as compared to BEAM and there are many nodes with a high jitter factor of 0.040 and above. The initial bootstrapping time in CS is also higher as shown in Figure 6(c).



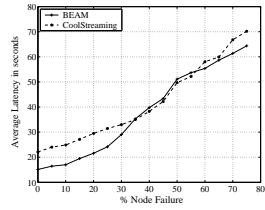
(a) % Jitter Rate vs % Sudden Failure



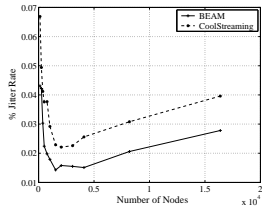
(b) Average Latency vs % Sudden Failure



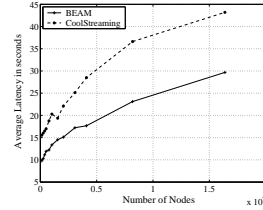
(c) % Jitter Rate vs % Gradual Failure



(d) Average Latency vs % Gradual Failure



(e) % Jitter Rate for Larger Swarm



(f) Average Latency for Larger Swarm

Figure 7: (a), (b) depict QoS under sudden node failure, (c), (d) depict QoS under gradual node failure, (e), (f) represent QoS for very large swarms in BEAM and CoolStreaming.

6.2.4 Robustness and Reliability

We conducted two types of experiment to evaluate the robustness and reliability of both systems: 1) We injected various percent of node failures after 50% of the simulation run time. 2) We injected one third of node failures at three different intervals: 25%, 50% and 75% of the simulation run. After the complete simulation run, we recorded the QoS factors viz percentage jitter rate and average latency. We study the impact of node failures on these metrics and the overall system performance. This simulation run comprises of 2048 nodes and maintains a 512 Kbps streaming rate.

Figure 7(a) shows the results for our first set of experiment where we inject node failures after 50% simulation run. It can be seen that for 0% node failure, the jitter rate is almost negligible for both the schemes. The jitter rate steadily increases to 1% for around 20% node failures. After injecting 50% node failures, we observe that the rise in jitter rate is steep and reaches 8% for BEAM and 10% for CS. This effect can be understood considering the impact of node failures on the alliances. As the number of nodes gradually decline in case of node failure or departure, number of alliance members (h) decrease, thereby weakening the overall graph connectivity. As nodes continue to falter, the alliance becomes sparse and sometimes is broken completely. The increase in time and consequently jitter rate is due to the time required to find alternate paths and receive the content. CS has displayed similar trends except that it has comparatively more jitters with failure. This can be similarly understood that a node requires more time to find new peer with the available pieces.

Figure 7(b) shows the impact of node failures on average latency. In BEAM, for 25% node failures, the average latency incurred is around 30 seconds. At 0% failure, the latency is well within 20 seconds and steadily increases with the node failure rate. The steep curve is prominent after 20% failures. Similar is true for CS except that it takes more time to start the media playback. The same average latency is maintained for the rest of the session. A plausible reason for the behavior of BEAM is that since a node cannot procure packet, it issues multiple requests to the node having desired content in the alliance. In case of a complete alliance failure, the nodes need to re-form an alliance, thereby increasing the average latency. In CS, for 25% node failures the latency increases to 56 seconds and rises steadily after that to almost 170 seconds for 75% node failure. Finding new peers after a node failure incurs an additional time in CS and this delay becomes the end to end

latency. The difference in results of BEAM and CS in Figures 7(a) and 7(b) can be understood in the light of SWN, which show robust behavior during churn.

In the second set of experiments we study the effect when node failure is gradual and occurs over three distinct time intervals: 25%, 50% and 75% of the simulation run. This setup depicts more realistic scenario and facilitates easy recovery. For example, to depict 30% node failures, we inject one third i.e. 10% node failures at 25%, 50% and 75% of the simulation times respectively. Figures 7(c), 7(d) depict the jitter rate and average latency for the above mentioned node failure rates. We observe that for 75% node failures the jitter rate is still under 1% for BEAM, though it has gone considerably high up to 2.5% for CS. The average latency for 75% node failure is around 60 seconds in BEAM and 70 seconds for CS. For lesser percent of node failures, the jitter rate and average latency are found to be under acceptable range of QoS. We observe that in the case where the node departure is gradual (at specific time intervals), the node recovery is comparatively easy and makes system inherently more stable since more time is available for recovery. For example, when the node failure occurs say within the first 25% of the simulation run time, the system is recovered much before another failure occurs at 50% simulation run time. Similarly, when another failure occurs at 75% simulation time the system is already stable.

6.2.5 Scalability

In this section, we extend the results obtained for QoS and uplink utilization to larger swarms. We evaluate scalability in terms of maximum number of nodes a streaming system can support without degrading the QoS. In this experiment, we vary the number of nodes from 128 to 16384. From Figure 7(e), we observe that for a swarm size of 16384 nodes, the average jitter rate is around 0.0278% in BEAM and almost around 0.04% for CS. With increasing nodes, jitter factor decreases and becomes steady after 1500 node count and marginally increases for very large swarms. However, even with a steep rise in the number of nodes in the swarm the average jitter factor is found to be under acceptable levels. The difference between CS and BEAM is more evident for larger swarms.

From Figure 7(f), in BEAM, the average latency is under 30 seconds for a 16384 node swarm. The peer lag is approximately than 20 seconds. As the number of users in the swarm increase, there are more alliances and

Table 4: A comparison of QoS for various h, k values for a 1024 node swarm, media encoded with 512 Kbps. Peering denotes peering scheme in BEAM in terms of h and k , N denotes number of neighbors, $BEAM_J$ and CS_J denote Average Jitter Rate for BEAM and CS, $BEAM_L$ and CS_L denote Average Latency for BEAM and CS, $BEAM_U$ and CS_U denote Uplink Utilization for BEAM and CS.

Peering	N	$BEAM_J$	CS_J	$BEAM_L$	CS_L	$BEAM_U$	CS_U
$h, k = 4, 2$	6	0.0158	0.0221	15.12	22.11	90.13	80.64
$h, k = 5, 2$	8	0.0156	0.0213	15.89	21.89	91.26	82.76
$h, k = 4, 3$	9	0.0162	0.0202	16.23	20.27	92.42	84.34
$h, k = 6, 2$	10	0.0164	0.0206	17.63	22.18	90.57	85.10
$h, k = 4, 4$	12	0.0164	0.0210	16.11	23.34	88.26	84.14
$h, k = 5, 3$	12	0.0159	0.0210	15.04	23.34	92.53	84.14
$h, k = 4, 5$	15	0.0176	0.0231	17.72	23.42	86.47	83.59
$h, k = 6, 3$	15	0.0177	0.0231	17.14	23.42	89.41	83.59
$h, k = 5, 4$	16	0.0186	0.0245	17.98	24.03	85.53	80.68
$h, k = 4, 6$	18	0.0181	0.0244	18.31	24.16	86.77	82.71
$h, k = 5, 5$	20	0.0190	0.0249	18.68	26.98	87.63	84.93

as the content is forwarded from one alliance to other, the number of total hops increase resulting in a higher latency. As mentioned previously, a high average latency is not acceptable in live media streaming as live media content is time sensitive and loses its importance if the delay is greater. CS and BEAM have a comparable performance except that CS takes a little more time to bootstrap. One of the important problems in CS like models is the high peer lag for media playback and high buffering time. BEAM has displayed considerable improvement in both aspects, i.e. reduced peer lag and reduced initial buffering time from more than half a minute to approximately 20 seconds. It is very difficult to achieve TV like switching because of the lack of a dedicated proxy during initial buffering time.

6.2.6 Control Overhead

Control overhead is the ratio of the total number of bytes expended in communication and control to the total bytes used for streaming data payload. An efficient system aims to minimize the control overhead (CPU time and bandwidth) and maximize resource utilization towards the streaming content. Figure 8 shows the communication overhead incurred in varying swarm

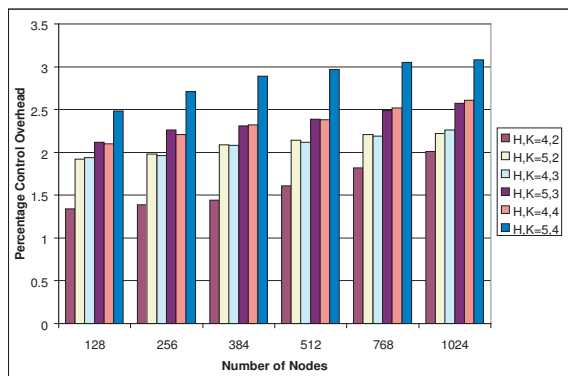


Figure 8: *Control Overhead in BEAM for various values of h and k .*

sizes ranging from 128 to 1024 nodes. We vary the values of h and k . Recall, that values of h and k denote the node degree. With higher node degree, additional resources are needed resulting in an increased communication and control overhead. For 1024 node swarm and $(h, k) = (5, 4)$ (node degree=16), the control overhead is slightly over 3%. For most other permutations of h and k values, the control overhead is around 2%. Table 4 shows affect of various values of h and k on the QoS parameters. We found $(h, k) = (5, 2)$, $(h, k) = (4, 3)$ and $(h, k) = (5, 3)$ as well performing schemes. In our experiments in the paper, we use $h = 4$ and $k = 2$, to show a comparison with CS which has a neighbor count of 6. As mentioned in [6], for a 200 node swarm in CS, the control overhead is around 2% for node degree of 6, which is quite comparable to BEAM. BEAM and CS almost incur similar overhead. In BEAM, a node sends announce request as it receives a packet, while in CS nodes send information packets to all their neighbors periodically about their buffer state.

6.2.7 Power Nodes

Table 5 shows the effect of power nodes on the whole system. For various values of α (the weight factor for *CSR* and *TSR*), the jitter factor, average latency and uplink utilization are compared. $pNodes$ is the number of distinct nodes that were chosen as power nodes at least once during the streaming session. The optimal choices were found to be $\alpha = 0.67$ and $\alpha = 0.75$, though other choices were also good with marginal overhead in jitter rate, latency or uplink utilization. This may lead to a very important question: Is it nec-

Table 5: *Evaluation of power nodes and their effect on the QoS factors. The swarm is composed of 2048 nodes and media encoded with 512 Kbps. α is the weight of CSR for calculating the UF. pNode denotes number of distinct power nodes during the streaming session. For various values of α we evaluate number of distinct power nodes active and their effect on overall QoS.*

α	pNodes	Average Jitter	Average Latency	Uplink Utilization
0.00	76	0.0175	17.12	90.18
0.20	73	0.0185	17.31	89.46
0.25	67	0.0186	18.23	91.42
0.33	63	0.0175	17.66	89.45
0.50	54	0.0174	17.11	88.29
0.67	48	0.0160	16.54	91.37
0.75	45	0.0158	15.12	90.13
0.80	36	0.0162	15.78	89.27
1.00	29	0.0182	17.95	86.22

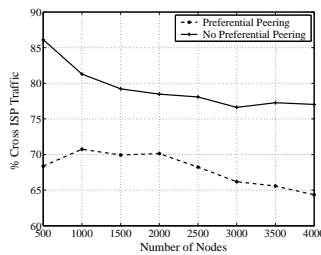


Figure 9: *% Cross Traffic vs Preferential Peering and No Peering.*

essary to change the power nodes at all during the streaming session? The answer could depend on many factors. What incentive do the high capacity nodes have in contributing the content altruistically? What if the already chosen power nodes decrease their uploading rate (in the absence of such a policy where best performers in terms of uploading are chosen as power nodes)? We believe that changing the power nodes brings altruism from the high capacity peers who have an interest of being served from the server. Altruism has a very important effect on the overall efficiency of the swarm and sometimes even more than *tit-for-tat* and any kind of forced fairness policies [31].

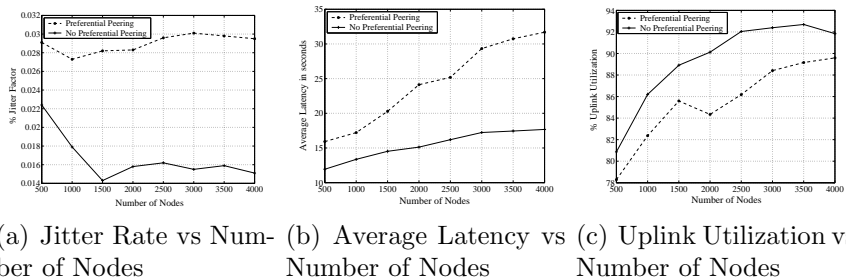


Figure 10: QoS vs Number of Nodes in Preferential Peering and No Peering for cross ISP traffic analysis

7 Reducing Cross ISP Traffic by using Preferential Peering

In this section, we study an important concern raised by Ali et al. [4] about excessive amount of cross ISP traffic generated as a result of greedy algorithms in choosing peers. Huge cross ISP traffic actually increases the operating cost of an ISP significantly. To overcome such revenue losses, some ISPs impose traffic throttling, where they limit the bandwidths of such P2P traffic. As a direct consequence, the QoS perceived at the user end is affected. Cross ISP traffic can be significantly reduced by using a biased neighbor selection policy [28] in BT like P2P file sharing systems and near optimal performance can still be achieved. In such systems, a node chooses more peers based on the locality of the peers (i.e. within the same ISP) and also chooses some far located peers for content diversity. But BT is a file sharing system, has different internal policies and mainly works on *tit-for-tat* mechanism. In this section, we focus on the problem of P2P media streaming cross traffic and evaluate the preferential peering technique using alliance theory to counter the huge cross ISP traffic and evaluate if reducing cross traffic affects the QoS. If so, by how much? Is it possible to achieve both the goals of good QoS and minimal cross ISP traffic? If yes, what are the optimal conditions to achieve both the goals?

In the preferential peering scheme as a node joins a swarm, the tracker intelligently assigns it a peer list using two main criteria: 1) Peers in the similar bandwidth range, if available. 2) Peers in the same ISP, if available and possible. A node while creating and joining alliances can effectively

choose its peers or alliance members based on the locality and peer bandwidth range. This serves two important purposes: 1) Cross ISP traffic is reduced. 2) Improving traffic locality ensures less probability of congestion within the interior of the network as compared to bandwidth congestion in the cross ISP links. A good combination of peers from the same ISP and peers in comparable bandwidth range can yield good performance in terms of reducing the cross ISP traffic but it may affect QoS and other parameters.

In this section, we present a set of simulation experiments where we consider 20 ISPs and nodes in the swarm belong to these ISPs. We attempt to find and compare percent of traffic between different ISPs and internal traffic within the ISPs. We consider two cases: 1) Using BEAM with regular alliance theory. 2) Using preferential peering and alliance theory as explained above. The number of nodes vary from 500 to 4000 (i.e. in each ISP, the number of nodes vary from 25 to 200), the streaming rate is 512 Kbps, $(h,k)=(4,2)$. We assume that there is a link between all ISPs and communication can be carried out between all the links.

Figure 9 depicts the cross ISP traffic in both scenarios. Preferential peering using alliance theory reduces the cross ISP traffic significantly. With increasing node count, the percent cross traffic has reduced in both schemes. While using no peering (i.e. just using regular alliance theory), cross ISP traffic has decreased with increase in node count because the nodes increasingly have better connections within the same ISP, though these connections are not intentionally created in the same ISP. While using preferential peering, the cross traffic has reduced considerably. For a 4000 nodes swarm, there is a reduction of around 13% in cross ISP traffic, which is approximately 230 GB (a significant volume considering 4000 nodes and a streaming rate of 512 Kbps).

Figure 10 depicts the QoS parameters between the two scenarios. It is clear that preferential peering affects the QoS, though, it reduces cross ISP traffic significantly. For smaller swarms the difference is negligible, however, for larger swarms, which are of interest to us, this difference is considerable. For a 4000 node swarm, there is an increase in percent jitter by around 0.02%, and latency increases by approximately 15 seconds. Such behavior is explained in terms of nodes not getting the best connection. Since more peers are clustered in similar ISPs, nodes miss out on useful connections which can fetch newer pieces, resulting in extra jitters and latency. Contrary to the intuition that uplink utilization in preferential peering should be better than normal alliance based peering, uplink utilization has indeed decreased in the

preferential peering scheme. This is because nodes in the same alliance are unable to get new stream content through the best possible connection. As a result the nodes are idle at various times and the uplink bandwidth is not as effectively used as it is in normal alliance based peering where a node receives the best connection, though at the expense of high cross ISP traffic. Figure 10 shows that achieving near optimal QoS and reducing cross ISP traffic are independent goals and pursuit of one affects other.

8 Challenges and Future Work

We enumerate some practical engineering challenges involved in the deployment of our proposed work, and possible solutions to overcome them. We also mention the related future work.

1. A significant number of nodes (mainly home users) that use P2P media streaming are behind the network address translator (NAT) boxes. This implies that these nodes primarily act as receivers and not as transmitters in P2P streaming applications. Successful TCP NAT traversal can overcome most of these problems, however, in cases that include multiple levels of NAT and when both hosts are behind NAT [34], alternative strategies need to be devised. A new technique called as TCP hole punching [33] has shown to resolve this problem, though its implementation is not widespread as of now. SST [36] provides the functionality of TCP and can traverse existing NATs effectively. Standardization of NAT behavior by vendors can further facilitate the implementation [35]. Lately, NAT vendors are supplying boxes with support for TCP NAT traversal. Manually configuring home routers is also seen as an option to overcome the NAT traversal problem [32].
2. Security issues such as malicious behavior by an end user tampering the software at client end or inserting garbage content are not covered in the paper. With growing popularity of P2P streaming applications, it is a necessary to identify and address the security concerns.
3. In this paper, our proposed work mainly focuses on architectural and organizational aspects of P2P media streaming and does not deal with the media content and its various attributes like compression, audio/video quality and media format types. Different media formats vary in their space requirements. There is on going research in making storage efficient and patent free formats for streaming by an end user. It is beyond the scope of our current work.

4. Our research is complementary to the advanced source and channel coding techniques such as layered coding, multiple description codes (MDC), fountain codes, and network coding. Such techniques can be suitably adopted in our work.
5. Our simulation results have shown the feasibility of alliance based peering scheme in P2P streaming systems. We intend to create a real world client for the actual deployment of BEAM.

9 Summary and Conclusion

We introduced a novel framework for P2P media streaming, *BEAM*, that uses alliance based peering scheme to solve some of the existing problems in chunk based P2P media streaming. In particular, our main contributions and findings are:

1. Peer lag (while media playback) can be significantly reduced from the order of minutes to approximately 10-20 seconds in the swarm. Initial buffering time has been reduced from around 30 seconds to 10-12 seconds for smaller swarms and less than 20 seconds for larger swarms (4096 nodes) in BEAM. Further reduction in such buffering time to a few seconds is extremely difficult because: a) Buffering time requires the time to find the path, stream content and possible forwarders of the stream content. b) Lack of any dedicated proxy during the initial (buffering) period. c) Heterogeneity of node bandwidths in the swarm.
2. BEAM has displayed robust and scalable behavior while delivering near optimal levels of QoS under varying workloads and conditions. Uplink utilization has improved considerably over CS and throughput is more than 90% for larger swarms. Alliance based peering theory encourages every node to contribute in order to receive the content, and indeed generates a fairer swarm.
3. Using preferential peering in combination with alliance theory can reduce significant volumes of cross ISP traffic, however, the delivered QoS is affected.

Finally, we believe that our results are promising and could provide research insight towards development of newer and efficient peering strategies in P2P media streaming systems.

References

- [1] Chu, Y-H., Rao, S.G., Seshan, S. and Zhang, H. (2002), ‘A Case for End System Multicast’, *IEEE Journal on Selected Areas in Communication (JSAC)*, *Special Issue on Networking Support for Multicast*, Vol 20, No 8, 2002.
- [2] URL:<http://www.pplive.com>
- [3] Hei, X., Liang, C., Liang, J., Liu, Y. and Ross, K. (2006), ‘A Measurement Study of a Large-Scale P2P IPTV System’, *Workshop on Internet Protocol TV (IPTV) services over World Wide Web in conjunction with WWW2006, Edinburgh, Scotland, May 2006*.
- [4] Ali, S., Mathur, A. and Zhang H. (2006), ‘Measurement of Commercial Peer-to-Peer Live Video Streaming’, *Workshop in Recent Advances in Peer-to-Peer Streaming, August, 2006*.
- [5] URL: <http://www.sopcast.org/>
- [6] Zhang, X., Liu, J., Li, B. and Yum, T-K.P. (2005), ‘CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming’, *In IEEE Infocom. IEEE Press, 2005*.
- [7] Watts, D. and Strogatz, D. (1998), ‘Collective dynamics of small-world network’, *Nature*, 440-442, 1998.
- [8] Deering, S. and Cheriton, D. (1990), ‘Multicast Routing in Datagram Internetworks and Extended LANs’, *In ACM Transactions on Computer Systems, 1990*.
- [9] URL:<http://www.akamai.com>
- [10] Banerjee, S., Bhattacharjee, B. and Komareddy, C. (2002), ‘Scalable Application Layer Multicast’, *In Proceedings of ACM SIGCOMM, 2002*, Pittsburgh, USA, 2002, pp.205–217.
- [11] Tran, D.A., Hua, K.A. and Do, Tai. (2003), ‘Zigzag: An efficient peer-to-peer scheme for media streaming’, *In IEEE Infocom. IEEE Press, 2003*.

- [12] Deshpance, H., Bawa, M. and Garcia-Molina, H. (2001), ‘Streaming live media over a peer-to-peer network’, *Stanford University, Tech Rep, August 2001*.
- [13] Magharei, N., Rasti, A., Stutzbach, D. and Rejaie, R. (2005), ‘Peer-to-Peer Receiver-driven Mesh-based Streaming’, *Proceedings of the ACM SIGCOMM, Poster Session, August 2005*.
- [14] Dana, C., Li, D., Harrison, D. and Chuah, C-N. (2005), ‘BASS: BitTorrent Assisted Streaming System for Video-on-Demand’, *IEEE International Workshop on Multimedia Signal Processing (MMSP) October 2005*.
- [15] Cohen, B. (2003), ‘Incentives build robustness in BitTorrent’, *P2P Economics Workshop, Berkeley, CA, 2003*.
- [16] Vlavianos, A., Iliofotou, M. and Faloutsos, M. (2006), ‘BiToS: Enhancing BitTorrent for Supporting Streaming Applications’, *In IEEE Infocom 2006 Global Internet Workshop, April 28-29, 2006*.
- [17] Annapureddy, S., Gkantsidis, C., Rodriguez, P. and Massoulie, L. (2005), ‘Providing Video-on-Demand using Peer-to-Peer Networks’, *Microsoft Technical Report MSR-TR-2005-147, 2005*.
- [18] Small, T., Laing, B. and Li, B. (2006), ‘Scaling Laws and Tradeoffs of Peer-to-Peer Live Multimedia Streaming’, *ACM Multimedia 2006, Santa Barbara, California, October 23-27, 2006*.
- [19] Gkantsidis, C. and Rodriguez, P. (2005), ‘Network Coding for Large Scale Content Distribution’, *IEEE Infocom, Miami, FL, USA, March 2005*.
- [20] Li, J. (2004), ‘PeerStreaming: A Practical Receiver-Driven Peer-to-Peer Media Streaming System’, *MSR-TR-2004-101, Sept. 2004*.
- [21] Bollobas, B. (2001), ‘Random Graphs’, *2nd Edition, 2001, Cambridge University Press*
- [22] Kleinberg, J. (2000), ‘The small-world phenomenon: An algorithmic perspective’, *32nd ACM Symposium on Theory of Computing, 2000*.

- [23] URL: <http://https://networkx.lanl.gov/>
- [24] Bharambe, A., Herley, C. and Padmanabhan, V.N. (2006), ‘Analyzing and Improving a BitTorrent Networks’s Performance Mechanisms’, *In IEEE Infocom, 2006*, Barcelona, Spain, 2006.
- [25] Adar, E. and Huberman, B. (2000), ‘Free riding on Gnutella’, *First Monday*, October 2000.
- [26] Lai, K., Feldman, M., Stoica, I. and Chuang, J. (2003), ‘Incentives for Cooperation in Peer-to-Peer Networks’, *First Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, June, 2003*.
- [27] Choi, S., and Kim, C. (2001), ‘Loss recovery time of Impatient variant of TCP NewReno’, *Electronic Letters, Vol 37, No. 25, December 6, 2001*.
- [28] Bindal, R., Cao, P., Chan, W., Medved, J., Suwala, G., Bates, T., and Zhang, A. (2006), ‘Improving Traffic Locality in BitTorrent via Biased Neighbor Selection’, *ICDCS 2006*.
- [29] Median, A., Lakhina, A., Matta, I., and Byers, J. (2001), ‘BRITE: Universal Topology Generation from a Users’ Perspective’, *BU-CS-TR-2001-003. April 05, 2001*.
- [30] Chu, Y-H., Chuang, J., and Zhang, H. (2004), ‘A Case for Taxation in Peer-to-Peer Streaming Broadcast’, *ACM SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS), Portland, OR, August, 2004*.
- [31] Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., and Venkatramani, A. (2007), ‘Do incentives build robustness in BitTorrent?’, *4th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2007)*.
- [32] URL:<http://azureus.sourceforge.net/>
- [33] Ford, B., Srisuresh, P., and kegel, D. (2005), ‘Peer-to-Peer Communication Across Network Address Translators’, *Proceedings of the 2005 USENIX Annual Technical Conference (Anaheim, CA, April 2005)*.

- [34] Guha, S, and Francis, P. (2005), ‘Characterization and Measurement of TCP Traversal through NATs and Firewalls’, *Proceedings of Internet Measurement Conference (IMC), Berkeley, CA, Oct 2005, pp. 199-211.*
- [35] URL: <http://www.ietf.org/html.charters/behave-charter.html>
- [36] URL: <http://pdos.csail.mit.edu/uia/sst/>