Diurnal Availability for Peer-to-Peer Systems

Nashid Shahriar[1], Mahfuza Sharmin[1], Reaz Ahmed[1], Md. Mustafizur Rahman[1], Raouf Boutaba[2], Bertrand Mathieu[3]
[1]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
[2]School of Computer Science, University of Waterloo Ontario, Canada,[3]Orange Labs
Email: {nshahriar, sharmin, reaz, mustafiz_rahman}@cse.buet.ac.bd, rboutaba@bbcr.uwaterloo.ca, bertrand2.mathieu@orange-ftgroup.com

***Abstract*—Ensuring content availability in a persistent manner is essential for providing any consistent service over peer-to-peer (P2P) systems. This paper introduces an efficient protocol, called *DATA*, to design highly available P2P systems irrespective of peer uptime and churn. Our approach utilizes the diurnal pattern of globally dispersed peers to develop a grouping strategy where each group aims to ensure 24x7 data availability within the group. Simulation results reveal that our protocol converges fast and ensures high availability for each group with minimal overhead.**

I. Introduction

There has been a rapid increase in the popularity of peer-to-peer (P2P) systems in the last decade. Such systems typically lack dedicated infrastructure, centralized administration, but rather depend on the voluntary participation of individual computers often referred to as peers. These properties attract enormous number and heterogenous types of peers from around the world to share their otherwise unused resources such as computing power and storage. With the advancement of P2P technology, traditional client-server based systems are being investigated to be deployed in the P2P model.

One of the key challenges behind the success of P2P solutions is to ensure persistent availability of the content upon which the service is dependent. Existing availability oriented approaches are either bandwidth hungry or require complex predictive knowledge stored and computed during replica relocation. Frequently, these approaches burden the highly available peers resulting in a skewed load distribution-Another shortcoming lies in gathering information needed for the replication process in large and unstructured networks.

Cyclic diurnal pattern in peer availability has been observed in the previous studies of P2P systems [3], [4], [5], [6]. Rzadca et al. [7] have shown that diurnal behavior of peers can be a useful characteristic for improving availability if the system has truly global scope i.e. the participating peers are distributed in different time zones. When considering such a system under a Universal Time Standard, the cyclic behavior of the peers situated in different time zones can be found to be complementary or partially overlapping. For example, two cyclic peers that are usually down during the night, but located in two distinct places having 12-hour clock difference, show complementary availability pattern. Even the peers located in the same time zone may show diverse availability pattern depending on the individual's Internet habit or job nature. As shown in Figure 1, peers A, B, C, and D having similar daily Internet habit and located in four different time zones can together provide improved availability around the clock. Our proposed scheme takes advantage of this kind of behavior to improve availability with smaller replication overhead.

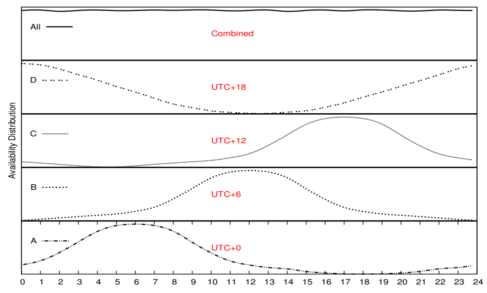


Fig. 1. Global Availability Pattern

The remainder of the paper is organized as follows. In section II, we propose an efficient peer grouping protocol for developing P2P systems leveraging 24x7 availability with high probability while minimizing the aforementioned shortcomings. Our protocol, called *DATA (Diurnal Availability by Temporal Assemblage)* of peers is based on the utilization of the diurnal availability patterns of geographically distributed peers. To evaluate the effectiveness of our proposal, we implement and evaluate our protocol using Peersim [1]. The simulation results in section III also validate our claim: *DATA* ensures persistent availability of the content by means of intelligent grouping of small number peers.

II. Diurnal Grouping

In our proposed protocol, *DATA*, peers having complementary availability behaviors form groups. A group consists of small number of peers and ensures that some members of the group are available with high probability at any given time. Collectively, such a group ensures overall increase in the system's availability. To develop the protocol, we first need to keep track of the peers' uptime history and to represent availability behaviors mathematically. We also need a measurement technique for optimizing complementary availability patterns. Subsection II-A presents a way to represent availability pattern as a probabilistic vector and II-B describes the metrics used by peers to select their best matching group mates. The next challenge is to devise a strategy to form groups in a distributed manner for an unstructured P2P system. If there was an Oracle providing availability patterns of all the peers across the P2P system, the optimal grouping would still be an NP-complete problem [7]. In addition, collecting and searching availability patterns of all the peers is not feasible in an unstructured P2P system. So, we propose a gossip based approach to construct a self-organizing gradient topology based on optimistic availability patterns in II-C. The information contained in the gradient topology is used to devise the grouping strategies as elaborated in II-D.

A. Availability Vector

The traditional definition of host availability simply measures it by the fraction of time a peer is online [9]. If a peer joins and leaves n times during the period $t - t_0$ and every time remains up for duration t_i, then host availability can be computed as follows

$$\mathcal{A}(t) = \frac{\sum_{i=1}^{n} t_i}{t - t_0} \tag{1}$$

However, this formula ignores the effect of time-of-day on host availability which is a function of time t. According to this formula, all peers A, B, C, and D in Figure 1 have the same value for availability though they largely differ in their pattern of availability. In [20] Yang et al. mathematically demonstrated the drawbacks of applying the above formulation in P2P system and argue for the necessity of a discrete model.

To discretize the traditional availability function, we propose to use a vector representation similar to [20]. As our proposal is based on the diurnal pattern of availability, we divide the 24 hours of a day into multiple, say K, slots of equal length Δt. So, $K = \frac{24}{\Delta t}$. We represent availability a_{ik} of a peer P_i in a particular slot k, by a historical probability of P_i to be online at k^{th} time slot which is the time duration from $\frac{24}{\Delta t}(k-1)$ to $\frac{24}{\Delta t}(k)$ of every day. By gathering the peer's most common availability information throughout a day, we get a_{ik} for all the K slots of a day. We represent availability behavior of a peer P_i as a K dimensional vector, named Availability Vector $\mathcal{A}_i$ as shown in Equation 2. *DATA* protocol layer deployed on a peer P_i, can easily compute and maintain $\mathcal{A}_i$ by recording its Internet connectivity history for a sufficient period of time.

$$\mathcal{A}_i = \{a_{i1}, a_{i2}, ..., a_{ik}, ..., a_{iK}\} \tag{2}$$

We, now define the availability vector, $\mathcal{A}_g$ for a group of peers formed using our protocol. The representation is the same as Equation 2 but the meaning of individual components is now different as more than one peer are involved. We define the availability of a group as the availability of at least 1 member of that group across time. For a group G, the availability at the k^{th} slot, represented by a_{gk} in $\mathcal{A}_g$, can be computed as follows:

$$a_{gk} = 1 - \prod_{\forall P_i \in G} (1 - a_{ik}) \tag{3}$$

B. Metrics for Peer Selection

We define, the contribution $C_{i,j}$ between two peers P_i and P_j as the improvement on availability after merging them in a new group. We derive two equations to compute $C_{i,j}$ from A_i and A_j of the two participating peers. Before posing the equations, we introduce some terminology as follows. Initially, the system starts with only individual peers but later both isolated peers and grouped peers may be present. For consistency, we consider a single peer as Singleton Group which has the peer as its sole member. The size of the group in which P_i resides is expressed by $|G_i|$, and $|G_i| = 1$ for a Singleton group. $|G_i \cup G_j|$ denotes the size of the new group consisting of the former two groups containing P_i and P_j. The joint availability of peers P_i and P_j at slot k is denoted by J_{ijk} and defined by $J_{ijk} = a_{ik} * a_{jk}$. Let C_{ijk} denote the contribution of any two peers P_i and P_j at slot k. We compute the value of $C_{i,j}$ using the following *"Conservative"* equation:

$$C_{i,j} = \sum_{k=1}^{K} \frac{C_{ijk}}{|G_i \cup G_j|} \tag{4}$$

where,

$$C_{ijk} = \begin{cases} 0 & \text{if } a_{ik} = a_{jk}; \\ J_{ijk}^{(\frac{a_{ik}}{a_{jk}})} - J_{ijk} & \text{if } a_{ik} < a_{jk}; \\ J_{ijk}^{(\frac{a_{jk}}{a_{ik}})} - J_{ijk} & \text{if } a_{ik} > a_{jk}. \end{cases}$$

We now explain the motivation behind this equation. We consider a time slot, say k, for two peers P_i and P_j to be complementary if $|a_{ik} - a_{jk}|$ is close to 1. According to our observation, for any two peers, P_i and P_j, complementary slots should contribute more in the result and to reflect this the value of J_{ijk} is deducted from the term $J_{ijk}^{(\frac{a_{ik}}{a_{jk}})}$ (or $J_{ijk}^{(\frac{a_{jk}}{a_{ik}})}$). To illustrate, consider three peers P_i, P_j,and P_l having availability a_{ik}, a_{jk}, and a_{lk} at slot k. For peer P_i, P_j is more attractive than P_l at slot k if $a_{jk} > a_{lk}$. To minimize the affinity towards large groups which should in general have higher slot availability, the term $\frac{1}{|G_i \cup G_j|}$ has been introduced in Equation 4 as a factor while summing up the slot wise contribution.

We call the above equation *Conservative* because it focuses on the complementary behavior to ensure group's availability across time. We also derive a "*General*" equation based on the concept of probability of at least one member of any group being present. The underlying explanation is relatively straightforward. If two peers P_i and P_j are merged into a group G, P_i or P_j should be benefitted by the utility, U_{ig} or U_{jg}. Mathematically,

$$U_{ig} = \sum_{k=1}^{K} (a_{gk} - a_{ik}), U_{jg} = \sum_{k=1}^{K} (a_{gk} - a_{jk})$$

The following *General* equation suffices to find the contribution $C_{i,j}$ satisfactorily.

$$C_{i,j} = \frac{(U_{ig} + U_{jg})}{|G_i \cup G_j|} \tag{5}$$

C. Availability Information

Peers in unstructured P2P systems keep little knowledge about the system and interact with a limited number of neighbors. In such a system, to find the current best candidate to group with, *DATA* needs to devise a search strategy that is efficient, scalable and that converges fast without any centralized component. To keep the search space relatively smaller, we propose to maintain a local list of current best candidates named as $knownlist$ in each peer. The search method exploits the above list to achieve a significantly better search performance than traditional search techniques, such as random walk, which requires communication with potentially all peers in the system. The absence of a centralized component requires that the construction and maintenance of the local list to be self-organized. In *DATA*, we propose a gossip based information exchange method named $Exploration$ to generate a list of current best candidates under a completely decentralized environment. Each group whether Singleton or not, has a leader, which on behalf of the group is responsible for communicating with other peers to process grouping activities. Among the alive members of a Group, the peer with the highest individual slot probability is chosen as the leader of the group. When the leader goes down, a new leader is elected according to the Bully Election Algorithm [21]. During exploration, each group leader collects availability information from as many peers as it can reach, i.e., its direct neighbors and the neighbors of its group mates. But it only keeps the information about a predefined $knowncount$ number of peers in its $knownlist$ whose contributions are attractive in the current context. As shown in Table I each entry of the $knownlist$ contains identity of a peer, its availability and grouping information.

TABLE I
DATA STRUCTURES CONTAINED IN PEER P_i

Data structure	Description
$neighborlist$	$\{P_j \mid \forall P_j, P_j \in neighbors(P_i)\}$
$knownlist$	$\{P_j, \mathcal{A}_j, \lvert G_j\rvert, attempted \mid \forall P_j \in P_i.knownlist\}$
$memberlist$	$\{P_j \mid \forall P_j \in G, P_i \in G, P_i \neq P_j\}$
$leader$	$\{P_j \mid P_j \in G, \forall P_i \in G, P_j \neq P_i, a_{jk} \geq a_{ik}\}$
$sentrequest$	$\{P_j \mid P_i$ has sent $GroupInvitation$ to $P_j\}$
$\mathcal{A}_i$	$\{a_{i1}, a_{i2}, ..., a_{ik}, ..., a_{iK}\}$

Algorithm 1 P_i.EXPLORATION()

```
1: for each $P_j \in P_i.neighborlist$ and $P_j$ is up
2:     $NeighborInfo$.add($P_j, \mathcal{A}_j$)
3: for each $eq \in ExploreQuery$ received
4:     $P_s \leftarrow eq$.getSource()
5:     $er \leftarrow$ exploreReply($NeighborInfo$)
6:     Send $er$ to $P_s$
7: if $P_i$ is a Leader
8:     for each $P_m \in (P_i.memberlist \cup P_i)$
9:         for each $P_j \in P_m.neighborlist$
10:            if $P_j \in P_i.memberlist$ and $P_j$ is down
11:                Continue to next iteration
12:            Send $ExploreQuery$ to $P_j$
13:            Compute $C_{i,j}$
14:            if $C_{i,j} >$ min($\{C_{i,k} \mid \forall P_k \in P_i.knownlist\}$)
15:                $P_i.knownlist$.add($P_j$)
16:    for each $er \in ExploreReply$ received
17:        for each $P_j \in er.NeighborInfo$
18:            Compute $C_{i,j}$
19:            if $C_{i,j} >$ min($\{C_{i,k} \mid \forall P_k \in P_i.knownlist\}$)
20:                $P_i.knownlist$.add($P_j$)
```

As shown in Algorithm 1, in the exploration process, the neighboring peers gossip with each other to exchange relevant information. Initially, a leader, P_i, sends $ExploreQuery$ to all its neighbors and the neighbors of its group mates who are alive in the current slot. At the same time, P_i initializes its $knownlist$ with the neighbors that are not member of its group. In the second phase, all peers upon reception of $ExploreQuery$, collect currently alive neighborhood information and bundle it in the $ExploreReply$ to be sent to the requesting peer. Finally, the exploring peer utilizes the peer availability information gathered from $ExploreReply$ to update its $knownlist$. Using the previously described equations, it compares the contribution for each of the collected peer's availability with the contribution of each peer in its current $knownlist$ and updates the $knownlist$ with the best set of peers having the highest contributions. By the end of its exploration, the peer is expected to know the best matching peers in its one or two hop neighborhood distance.

D. Group Construction

After gathering availability information and updating $knownlist$ through $Exploration$, a peer repeatedly executes a process called $PeerCycle$ to form and modify groups utilizing the complementary uptime distribution in the peers' availability patterns. Such a group is constructed incrementally, i.e., forming groups with two single peers initially then growing in size up to the maximum allowable group size. Later, two non-Singleton groups merge into a larger one such that the resultant availability of the new group in all the slots increases from the availability of the former two groups by a sufficient margin. Groups can revoke the membership of a peer if it is not any more contributing to the availability increase.

Algorithm 2 P_i.PEERCYCLE()

```
1: if $GroupUpdate$ received
2:     Update Grouping Information
3: if $Revoke$ received
4:     $|G_i| \leftarrow 1$
5:     $P_i.leader \leftarrow P_i$
6:     $P_i.knownList \leftarrow$ Exploration()
7: if $P_i$ is not a leader
8:     Forward all Incoming Messages to the leader
9:     exit
10: if SentRequestQueue not empty
11:     $P_m \leftarrow$ sentrequest.getPeer()
12:     if $Acceptance$ received from $P_m$
13:         MergeGroup($P_i, P_m$)
14:         sentrequest.Clear()
15:         exit
16:     else if $Denial$ received from $P_m$
17:         sentrequest.Clear()
18:         if $P_i.knonwlist.\mathcal{A}(P_m) \neq \mathcal{A}(P_m)$
19:             $P_i.knownList.\mathcal{A}(P_m) \leftarrow \mathcal{A}(P_m)$
20:             $P_m.attempted \leftarrow false$
21:         else
22:             $P_m.attempted \leftarrow true$
23:     else
24:         Initiate wait count
25:         exit
26: $P_{known} \leftarrow$ FindBestKnownPeer($P_i.knownlist$)
27: $P_{invitee} \leftarrow$ FindBestInviation($GroupInvitations$)
28: if $C_{i,P_{invitee}} > C_{i,P_{known}}$
29:     Send $Acceptance$ to $P_{invitee}$
30: else
31:     Send $GroupInvitation$ to $P_{known}$
32:     Send $Denial$ to $P_{invitee}$
33:     sentrequest.add($P_{known}$)
34: for each $P_j \in P_i.memberlist$
35:     Compute $C_{i,P_j}$
36:     if $C_{i,P_j} <$ ADAPTIVE_THRESHOLD
37:         Modify $\mathcal{A}_g$
38:         Send $Revoke$ to $P_j$
```

Algorithm 3 FINDBESTKNOWNPEER(KNOWNLIST)

```
1: $C_{max} \leftarrow$ ADAPTIVE_THRESHOLD
2: for each $P_j \in knownList$
3:     if $P_j.attempted = false$ and $|G_i \cup G_j| <$ MAXSIZE
4:         Compute $C_{i,j}$
5:         if $C_{i,j} > C_{max}$
6:             $C_{max} \leftarrow C_{i,j}$
7:             $P_{known} \leftarrow P_j$
8: return $P_{known}$
```

Algorithm 4 FINDBESTINVIATION(INVITATIONQUEUE)

```
1: $request_{max} \leftarrow$ ADAPTIVE_THRESHOLD
2: for each $gi$ in $InvitationQueue$
3:     $P_j \leftarrow gi$.getSource()
4:     if $P_j \in P_i.knonwlist$ and $P_i.knonwlist.\mathcal{A}(P_j) \neq \mathcal{A}(P_j)$
5:         $P_i.knownList.\mathcal{A}(P_j) \leftarrow \mathcal{A}(P_j)$
6:         $P_j.attempted \leftarrow false$
7:     compute $C_{i,j}$
8:     if $C_{i,j} > request_{max}$ and $|G_i \cup G_j| <$ MAXSIZE
9:         $request_{max} \leftarrow C_{i,j}$
10:        $P_{invitee} \leftarrow P_j$
11: Send $Denial$ to all inviting peers except $P_{invitee}$
12: return $P_{invitee}$
```

In the $PeerCycle$ process shown in Algorithm 2, a non-leader peer finishes its turn by just forwarding all incoming messages to its leader. On the contrary, a leader P_i, searches the entries in its $knownlist$ and the incoming requests to find its best matching peer. The selection of the highest contributing peer, P_{known} from the $knownlist$ is illustrated in Algorithm 3. The search on the incoming invitations is shown by the Algorithm 4. Here P_i finds the best peer, $P_{invitee}$ in terms of contribution among the inviting peers and sends rejections to all the inviting peers except $P_{invitee}$ through $Denial$ messages. P_i then compares the contribution of $P_{invitee}$ with that of P_{known} to ensure that they are the best matching from both ends. If the contribution of $P_{invitee}$ is greater, only then P_i accepts the invitation by sending an $Acceptance$ message to $P_{invitee}$. Otherwise P_i has found a better matching peer, P_{known} in its own $knownlist$. In this case, P_i rejects the invitation from $P_{invitee}$ through a $Denial$ and invites P_{known} to form a group through a $GroupInvitation$ message. After sending the $GroupInvitation$, P_i waits for the response from P_{known}, which can either be an $Acceptance$ or a $Denial$. To avoid inviting a unsuccessful peer repeatedly, a flag named $attempted$ is incorporated for each entry in the $knownlist$. It is marked after a $Denial$ is received and cleared after P_i detects change in the $\mathcal{A}_j$ of P_j. Upon receiving a $Denial$, P_i repeats the process by selecting one of the unattempted peers in its $knownlist$ in order of their contributions and inviting it as previously described. When sending a $GroupInvitation$ or a $Denial$, the originator of the message piggybacks its present availability and grouping information so that the receiver can update its $knownlist$ accordingly. Coalescing of two groups is allowed if the resultant group size does not exceed the predefined group size limit. To prevent unnecessary $GroupInvitation$s propagating across the network, an adaptive threshold has been incorporated whose value is computed depending on the current grouping state of the peer.

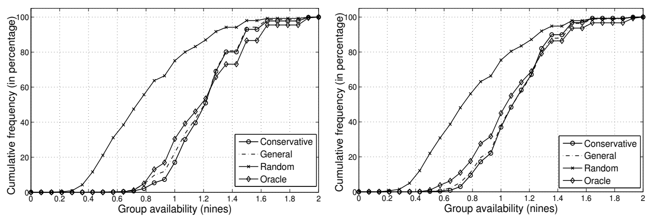
If none of the invited peers agree to form a group, P_i remains unchanged. On the contrary, upon receiving an $Acceptance$ from a peer P_m, P_i initiates the process of coalescing of the two groups into a larger one as shown in Algorithm 5. At this stage both peers reach an agreement to form a new group and exchange their $\mathcal{A}$s and $knownlist$s. The components of availability vector $\mathcal{A}_g$ of the group G_n is computed using the Equation 3. The best $knowncount$ number of peers with respect to the new $\mathcal{A}_g$ are chosen to construct the new $knownlist$. $GroupUpdate$ message containing these updated information is propagated to all members of the new group. When become alive, the members update their group related information from the $GroupUpdate$ and consider group's $\mathcal{A}_g$ and $knownlist$ for further grouping activities.

Algorithm 5 MERGEGROUP(P_i, P_m)

```
1: create new group $G_n$ where $|G_n| \leftarrow |G_i \cup G_m|$
2: $G_n.memberlist \leftarrow G_i.memberlist \cup G_m.memberlist$
3: $G_n.leader \leftarrow P_l$ s.t. $P_l$ is up and $\forall x \in \{G_n.memberlist\}$, $P_l.id \geq x.id$
4: for each slot $k$ from 1 to $K$
5:     $\mathcal{A}_g[k] \leftarrow 1 - ((1 - \mathcal{A}_i[k])(1 - \mathcal{A}_m[k]))$
6: $G_n.knownlist \leftarrow$ best $knowncount$ peers from $G_i.knownlist \cup G_m.knownlist$ w. r. t. the new $\mathcal{A}_g$
7: $gupdate \leftarrow$ groupUpdate($G_n, \mathcal{A}_g$)
8: for each $P_j \in G_n.memberlist$
9:     Send $gupdate$ to $P_j$
```

III. Experimental Results

To evaluate the performance of our protocol, we simulate an unstructured P2P network using Peersim simulator [1]. We have implemented four grouping strategies; one for each of the



(a) alpha=0.1 (b) alpha=0.5

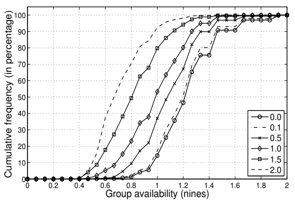
Fig. 2. Group availability measurements

two proposed methods in Section II-B, i.e., $Conservative$ and $General$; the third strategy follows random grouping among the currently alive two hop neighbors. To show the effect of global availability knowledge, we have also implemented a centralized Oracle based scheme. Time zone diversity is varied using the parameter $alpha$. To illustrate the effectiveness of our proposed protocol, we consider the following performance metrics:

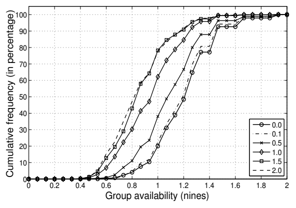
- *Group availability*: Group availability is measured in units of nines [5], defined as $-log_{10}(1-T)$, where T is the fraction of the total observed time when at least one member of the group is available. For instance, a group availability of 2 nines implies that the group is accessible during 99% of the total time.
- *Group count*: It is the total number of groups created by each strategy. For a fixed network size, group count is inversely proportional to the average group size which we can use as the replication factor.
- *Normalized message overhead*: It is a measure of total number of messages exchanged by the protocol normalized by the group count.

From the groups' availabilities in nines in Figure 2 we can see that for non-random strategies with $alpha = 0.1$, majority of the groups remain available more than 83% (0.75 nines) of total time whereas majority of groups formed by random strategy remain available only around than 50% (0.30 nines) of the time. If we increase the time zone disparity with $alpha = 0.5$, groups' available time decreases as finding matching peers becomes less probable. Figure 3 exhibits the impact of varying time zone disparity on different approaches. The figure clearly shows that impact of time zone diversity on random strategy is very small and the curves for different alpha tends to make close cluster. For all other three approaches, time zone diversity has strong impact with $alpha \rightarrow 0$ providing with the best availability for the group and $alpha \rightarrow 2$ with the worst result.

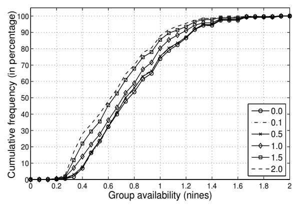
The improvement we gain on availability as shown in the above results comes at the cost of some overhead which is illustrated in Figure 4. The group count which is plotted along left axis can be a measure of replication overhead. The important aspect of the Figure 4 is that for a fixed network size the replication factor using Oracle has the lowest value, using random grouping has the highest, and using our methods lies in between the two. So far, all the graphs reveal the supremacy of the Oracle but its main bottleneck is shown in Figure 4 where normalized message overhead is plotted. The graph shows that normalized overhead for Oracle is significantly larger (5 to 9



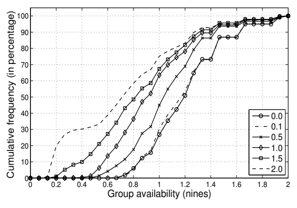
(a) Conservative



(b) General



(c) Random



(d) Oracle

Fig. 3. Group availability measurements

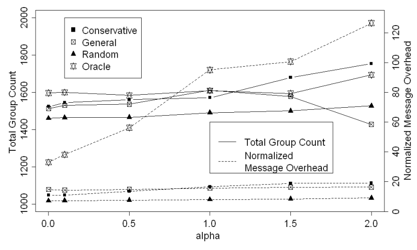


Fig. 4. Grouping overhead

times) than that of the other three strategies which is due to the network level search. As the Oracle requires communication with potentially all peers in the network, its normalized overhead increases with the increase of time zone diversity. The other three strategies involve communication with peers in a limited search space and so their normalized overhead remains somewhat constant. The normalized overhead for our methods is slightly greater than the random case. This stems from the fact that our methods require more attempts to find the best matching peers from both ends.

IV. Related Works

A number of approaches to improve availability in P2P systems can be found in the literature. These works vary in the type of redundancy, method of data regeneration, and the timing and number of peers storing redundant data. Bhagwan et al. [2] explored the issues of replication granularity, replica placement, and application characteristics. In terms of approach, redundancy is achieved either by replicating the complete data or fragmenting and encoding the data by erasure coding such that not all fragments are needed to reproduce it [2], [18]. Data replication is mainly done in two ways: reactive [9] or proactive [10]. Both approaches aim to optimally place the replicas in minimal number of peers so that overall availability of the data remains high. Existing solutions use information like peers' session time and churn [15], availability history [16], lifespan distribution [19], machines' uptime, downtime, lifetime, and correlation among them [13], Mean Time to Failure [8], up time score [18], recent up time [17], application specific availability [12], availability-prediction guided replica placement [9], [11], and probabilistic models [14] to tune the redundancy overhead.

All these redundancy based approaches rely on the cooperation of the hosts to achieve the desirable goal of availability. While *DATA* follows the same assumption of host cooperation it differs in the evaluation of replication criteria. Indeed, these schemes take into account only the current but single score of availability as the replication criteria whereas *DATA* disseminates the score across time to better capture the cyclic behavior and time-of-day effect of the P2P system. Also unlike other schemes *DATA* considers the fact of transient disconnections and utilizes the reintegration of data to significantly reduce the number of replicas needed.

V. Conclusion and Future Directions

In this paper we have described an efficient grouping scheme which irrespective of peer timing and churn ensures data availability around the clock. we plan to refine it and to store availability information using a Distributed Hash Table, which should result in a globally optimized and scalable group formation algorithm applicable to structured network. We also intend to investigate performance of *DATA* by deploying it on a real world P2P system and to ensure availability for specific application requirements. The success of *DATA* depends largely on the willingness and truthfulness of the peers. Tackling the untrusted behavior of peers and security issues of group formation is another prospective research issue.

References

[1] PeerSim: A Peer-to-Peer Simulator. http://peersim.sourceforge.net/.

[2] R. Bhagwan, D. Moore, S. Savage, and G. Voelker. *Replication strategies for highly available peer-to-peer storage systems.* In *Proc. FuDiCo,* 2002.

[3] D. Stutzbach, and R. Rejaie. *Understanding churn in peer-to-peer networks.* In *Proc. Internet measurement,* 2006.

[4] J. Chu, K. Labonte, and B. N. LevineH. *Availability and Locality measurements of Peer-to-Peer File Systems.* In *Proc. ITCom,* 2002.

[5] J. R. DOUCEUR. *Is remote host availability governed by a universal law. Performance Evaluation Review* Vol. 31, Issue 3, 25-29, 2003.

[6] S. Saroiu, P. K. Gummadi and S.D. Gribble. *A Measurement Study of Peer-to-Peer File Sharing Systems.* In *Proc. MMCN,* 2002.

[7] K. Rzadca, A. Datta, S. Buchegger. *Replica Placement in P2P Storage: Complexity and Game Theoretic Analyses.* In *Proc. DCS,* June 2010, pp. 599-609.

[8] K. Kim. *Time-related replication for p2p storage system.* In *Proc. ICN,* 2008. s

[9] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker. *Total Recall: system support for automated availability management.* In *Proc. NSDI,* 2004.

[10] A. Duminuco, E. Biersack, T. En-Najjary. *Proactive Replication in Distributed Storage Systems Using Machine Availability Estimation.* In *Proc. CoNEXT,* 2007.

[11] J. W. Mickens and B. D. Noble. *Exploiting Availability Prediction in Distributed Systems.* In *Proc. NSDI,* 2006.

[12] S. Shi, G. Yang, J. Yu, Y. Wu, and D. Wang. *Improving Availability of P2P Storage Systems.* In *Proc. APPT,* 2003.

[13] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. *Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs.* In *ACM SIGMETRICS,* 2000.

[14] K. Ranganathan, A. Iamnitchi, and I. Foster. *Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities.* In *Proc. GP2PC,* 2002.

[15] R. Mahajan, M. Castro, and A. Rowstron. *Controlling the cost of reliability in peer-to-peer overlays.* In *Proc. IPTPS,* 2003.

[16] S. Blond, F. Fessant, E. Merrer. *Finding Good Partners in Availability-aware P2P Networks.* In *Proc. SSS,* 2009.

[17] J. Sacha, J. Dowling, R. Cunningham, and R. Meier. *Discovery of stable peers in a self-organising peer-to-peer gradient topology.* In *Proc. IFIP* 2006.

[18] T. Schwarz, Q. Xin, and E. Miller. *Availability in Global Peer-To-Peer Storage Systems.* In *Proc. IPTPS,* 2004.

[19] F. E. Bustamante and Y. Qiao. *Friendships that last: Peer lifespan and its role in P2P protocols. Proc. Web Content Caching and Distribution*, pp. 233 - 246, 2004.

[20] Z. Yang, J. Tian, and Y. Dai. *Towards a More Accurate Availability Evaluation in Peer-to-Peer Storage Systems. International Journal of High Performance Computing and Networking,* Vol. 6 Issue 3/4, 2010.

[21] H. Garcia-Molina *Elections in a Distributed Computing System. IEEE Transactions on Computers,* Vol. 31 Issue 1, 1982, pp. 48-59.