Efficient Multi-source Data Dissemination in Peer-to-Peer Networks

Zhenyu Li[1,2], Zengyang Zhu[1,2], Gaogang Xie[1], Zhongcheng Li[1]

[1]Institute of Computing Technology, Chinese Academy of Sciences
[2]Graduate School of Chinese Academy of Sciences
Beijing, China
{zyli, antonial, xie, zcli}@ict.ac.cn

Abstract. More and more emerging Peer-to-Peer applications require the support for multi-source data dissemination. These applications always consist of a large number of dynamic nodes which are heterogeneous in terms of capacity and spread over the entire Internet. Therefore, it is a challenge work to design an efficient data disseminate scheme. Existing studies always ignore the node proximity information or have high redundancy or high maintenance overhead. This paper presents EMS, an efficient multi-source data dissemination in P2P networks. By leveraging node heterogeneity, nodes are organized in a two-layer structure: the upper layer is based on DHT protocol and composed of powerful and stable nodes, while the nodes at the lower layer attach to physically close upper layer nodes. Data objects are replicated and forwarded along implicit trees, which are based on DHT route table. The average path length for delivering objects to all nodes converges to $O(\log n)$ automatically, where n is the number of nodes in the application. We perform extensive simulations to show the efficacy in terms of delivery path length, delay, redundancy and the effect of proximity-awareness.

Key words: Multi-source multicast, Data Dissemination, P2P Networks

1 Introduction

A large number of emerging Peer-to-Peer applications, such as distributed multiplayer games, teleconferencing, remote collaboration and multimedia chat groups, require support for multi-source data dissemination. In these applications, each node is a potential data source. When a new data object emerges on a node, it should be propagated to all the interested members as soon as possible. Since these applications always consist of a large number of dynamic nodes which are heterogeneous in terms of capacity and spread over the entire Internet, it is a challenge work to design an efficient data disseminate scheme. In our opinion, an efficient multi-source data dissemination scheme should meet at least the following basic requirements.

- **Scalable performance.** The scheme should be decentralized and support node churns. As the system size grows, the efficiency should degrade gracefully.

- **Complete coverage.** A data object from any source should be delivered to all the other interest nodes.
- **Proximity aware.** In order to reduce the bandwidth consumption and data transfer delay, the scheme should account for node proximity information.
- **Heterogeneity aware.** The protocol should take the heterogeneity nature of nodes into account so that the load on a node is proportional to its capacity.
- **Low overhead.** Both the maintenance overhead of the P2P overlay network and the data message redundancy should be low.

Existing studies [3][4][5] on multi-source data dissemination do not meet all the requirements. The probabilistic gossip based hybrid push/pull scheme [3] brings lots of duplicate messages and only guarantees probabilistic convergency. CAM [4] builds capacity aware multicast on top of structured P2P network. So, it always has no redundancy. However, the maintenance overhead is considerable and it considers the proximity information only in a limited extent due to the rigid structures of structured overlay networks. ACOM[5] implicitly specifies a delivery tree for each source. However, it suffers a tradeoff between delivery delay and message redundancy. When the average path length is reduced to $O(\log n)$, the average number of duplicate messages grows to $O(n/\log n)$, where n is the number of nodes in the overlay network.

In this paper, we propose *EMS*, an Efficient Multi-Source data dissemination scheme. EMS is motivated by two facts. First, although the structured overlay networks need considerable maintenance overhead and have relative rigid structures, their predefined structures (e.g. ring, hypercube) give us useful information to extract data delivery trees. Second, measurement results in [11] [2] have shown that there exists some relative powerful and stable nodes in P2P applications. Therefore, we organize nodes in a two-layer structure: the upper layer is based on DHT protocol and composed of powerful and stable nodes, while the nodes at the lower layer attach to physically close upper layer nodes. Obviously, the upper layer node and the lower layer nodes attaching to it constitute a cluster. Data objects are replicated and forwarded on the upper layer along *k-ary* implicit trees, which are constructed using DHT route table. Thus, no control message is required to built the tree structures. The data objects are also delivered within the clusters using implicit tree structures. And in each step, a node tries to transmit the objects to as many new nodes as it can. We analyze the average path length for delivering objects to all the nodes and the results show that it automatically converges to $O(\log n)$, where n is the number of nodes in the P2P network. We also evaluate the performance of our scheme through comprehensive simulations. The results demonstrate that our scheme is more time- and cost-effective when compared with others.

The rest of the paper is organized as follows. Section 2 provides a survey of related work. Section 3 describes the construction of hierarchical structure in detail. Section 4 details the data dissemination and in section 5, we evaluate our scheme through simulation experiments. Finally, we conclude our work in section 6.

2 Related Work

Probabilistic gossip based schemes [3] are reliable, scalable and easy to deploy. Their major limitations are considerable duplicate messages and only probabilistic convergency guarantees.

Data-driven based methods [6] are simple and bring no redundancy. However, they suffer a control overhead and delay tradeoff [7], and therefore are not suitable for fast multi-source data dissemination applications. Tree-based schemes always build one tree or several disjoint trees [12] to propagate data. They are optimized for single source and not applicable to multi-source applications directly. NICE[8] forms nodes in a hierarchical structure to achieve high scalability. However, it is heterogeneity ignorant and fragile due to lack of redundancy in overlay networks. Narada[9] extracts source specific distribution trees from overlay network. It requires each node to maintain the information of all other nodes. Thus, it is only suitable for small scale systems.

Several methods have been proposed for multi-source data dissemination in P2P systems. [1] and [4] are based on structured P2P networks. [1] assumes each node has same capacity and is not suitable for heterogeneous systems. [4] takes node capacity into account. A disadvantage of structured overlay based schemes is the considerable maintenance overhead. Moreover, the overlays are always too strict to optimize due to the predefined structures, e.g. ring or multi-dimension coordinates. Although our scheme also leverages DHT protocols to organize the upper layer nodes, the maintenance overhead is relative small because upper layer only consists of small portion of nodes which are more stable.

ACOM[5], on the other hand, organizes nodes in a ring structure to support any-source multicast. Each node has a nearby neighbor and several random neighbors. Data object is first forwarded via random neighbors within several hops. Then it is transmitted down to the ring by the aware nodes, until all the nodes become aware of the object. The major contribution of ACOM is that the data object can be broadcasted to all the members in $O(\log_c n)$ hops while no explicit tree is maintained, where c is the average node capacity and n is the number of members. However, this is at the cost of non-negligible redundancy. When the average path length is reduced to $O(\log_c n)$, the average number of duplicate messages grows to $O(n/\log_c n)$. In addition, the constant coefficient in $O(\log_c n)$ always is 2, which indicates the hop complexity is larger than other schemes.

As to hierarchy architecture, Shen and Xu in [10] propose a landmark clustering based hierarchical structure to account for proximity information. However, they directly use node Hilbert number as logical node ID in the auxiliary DHT-based structure. Thus, the identifier randomness in the structure does not exist any more, which is a necessary condition in our scheme. In addition, no upper bound is imposed on the number of regular nodes that can be assigned to a supernode. Therefore, the supernodes may be overloaded.

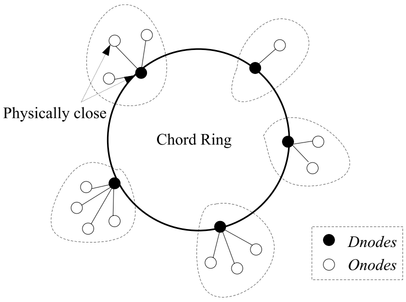


Fig. 1: Two-layer structure.

3 Hierarchical Structure Construction

Fig. 1 captures the hierarchical structure in our scheme. Powerful and stable nodes are organized in a Chord ring, labeled as *Dnodes*. Ordinary nodes, labeled as *Onodes*, attach to physically close Dnodes. Although we use Chord [13] as a representative DHT protocol, it is straightforward for other DHT protocols.

Obviously, a Dnode and the Onodes attaching to it constitute a cluster and the Dnode acts as a rendezvous point of the cluster. In order to avoid overloading Dnodes and relieve the single point of failure problem, an upper bound on the number of Onodes that a Dnode can be attached should be imposed. We call this upper bound as *cluster capacity* and denote it as m_c. The default value of m_c is 16.

Nodes have different capacities. Node x's capacity is denoted as c_x, which specifies the number of nodes to which x is willing to send data objects concurrently. We assume nodes can estimate the capacities by themselves. In practice, node expected uptime should also been taken into consideration as in Gnutella protocol [14].

3.1 Node Join

When a node x joins, it first locates nearby Dnodes. If there is a nearby Dnode y which can still accept ordinary nodes, x attaches to y as a y's Onode. Otherwise, depending on x's capacity, x either joins as a Dnode at the upper layer or attaches to a randomly chosen Dnode as an Onode.

We leverage landmark clustering scheme [15] to generate proximity information. r nodes are picked up randomly from the Internet as landmark nodes. Node x measures the distance (i.e. delay) to these landmark nodes and obtains a landmark vector $< d_1^x, d_2^x, \cdots, d_r^x >$, where d_i^x is the distance from x to the ith landmark node. The intuition behind is that physically close nodes are likely to have similar or close landmark vectors, and thus close to each other in the landmark space. Then the m-dimension landmark vector is mapped to a 1-dimension landmark numbers using Hilbert curve, which is an example of space-filling curves [16]. We partition the landmark space into 2^{rb} smaller grids with equal size, where b is the order of Hilbert curve and controls the number of grids used to partition the landmark space. Then we fit a Hilbert curve on the landmark space to number each grid. The nodes whose landmark number falls into grid ln has the landmark number as ln. Since space-filling curves map an m-dimension point to a 1-dimension point without loss of proximity, closeness in landmark number indicates physical closeness. We denote the landmark number of node x as ln_x. In current design, the default values of r and b are 15 and 2, respectively.

Each Dnode y publishes its information on the upper layer by storing its landmark number ln_y and address on the successor of identifier ln_y. As in [13], we assume new nodes can learn the information of an existing Dnodes (denoted as y_0) in the upper layer by some external mechanism. New node x asks y_0 to issue a Chord query message with its landmark number ln_x as key on the DHT-based upper layer to locate nearby Dnodes of x. Dnodes whose landmark numbers fall into range $Q = \{q : |q - ln_x| < Q_x\}$ are recognized as the nearby Dnodes of x, where $Q_x = Q_0 * 2^{\frac{c-c_x}{2}}$ and c is the average node capacity, Q_0 is the basic search radius. Thus, the search radius decreases exponentially with the growth of node capacity. The intuition behind is that the bigger a node's capacity is, the higher probability it is a Dnodes. Note that since each Dnode maintains a continuous Chord identifier space and the information of Dnodes with close landmark numbers is stored closely in Chord ring, locating nearby Dnodes should be fast. Let R_x denote the set of nearby Dnodes located for x.

Node x selects a node $z \in R_x$ and sends z an attaching request message. If z is able to accept an Onode, it satisfies x's request. Otherwise, x selects another node and performs the same operation until one of the Dnode accepts it. We call z is able to accept an Onode if the number of Onodes having attached to it does not exceed the cluster capacity m_c. If none of the Dnode in R_x can accept x as an Onode, the subsequent operation depends on the x's capacity c_x: if c_x is big enough (i.e. c_x exceeds a threshold C_{thd}), then x joins the upper layer as a Dnode; otherwise, x asks the well-known Dnode y_0 to locate a random Dnode u which still can accept an Onode and attach to u.

Now we describe how to choose the threshold C_{thd}. A Dnode maintains the information of the Onodes attaching to it and several other Dnodes' information as Chord protocol specifying. The maintenance cost should not be ignored. In our current design, each Dnode dedicates one unit capacity for maintenance. And as mentioned before, *k-ary* implicit trees are used to transfer data objects among Dnodes. So, a Dnode's capacity should at least be k. As detailed in the next section, data objects are also delivered within the cluster using implicit tree structure. The implicit tree is rooted by the Dnode in the cluster. To expedite data dissemination, the Dnode should at least have two child nodes on the implicit tree within the cluster. In summary, the threshold is set as Equ. 1.

$$C_{thd} = k + 3 \tag{1}$$

To increase robust, each Onode has a backup Dnodes list of length b. Onode x's list consists of physically close Dnodes to x. When the current Dnode fails, x

tries to choose a node from the list to attach. The list is initialized by the nearby Dnodes located during the joining procedure. To ensure that the backup Dnodes are always the available ones in the upper layer, x checks the availability of their backup Dnodes periodically. If a half of its backup Dnodes are not available any more, x relocates the nearby Dnodes by issuing a Chord query message through its Dnode. The query message is similar to the join message in terms of locating nearby Dnodes.

Now we analyze the number of control messages used for a join operation. When a new node joins, to locate the close Dnodes, $O(\log n_d)$ Chord query messages are required, where n_d is the number of Dnodes. If the new node joins as a Dnode, another $O(\log^2 n_d)$ Chord join messages are required. Thus, when a new node joins, on average,

$$\#MSG_{join} = \log n_d + p_{dn} \cdot \log^2 n_d \tag{2}$$

messages are required, where p_{dn} is the probability that a new node joins as a Dnode.

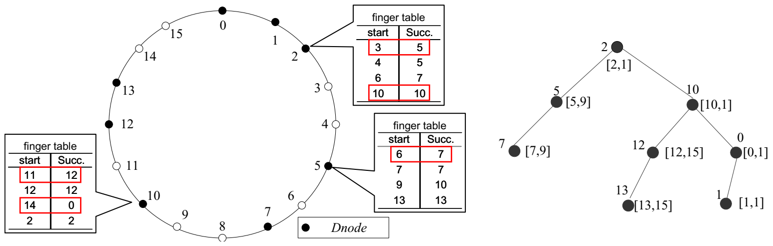
3.2 Node Departure and Failure

Each node has a buffer to store the data it received recently. When an Onode x leaves, it simply sends a *Leaving Message* to its corresponding Dnode. The Dnode then deletes the stored information for x. When a Dnode y leaves, it asks the Onodes attaching to it to switch their corresponding Dnode and then it leaves the upper layer based on the Chord protocol and unsubscribe its information (i.e. landmark number and address information) from the Chord ring. Each Onode z previously attaching to y selects a live node from its backup Dnode list to attach. If there is no node in the backup list can accept z, it rejoins. In any case, z records the sequence number of the last message received and fetches the messages it misses during above switch or rejoin process from other nodes' buffers actively.

Failure of an Onode can be detected by its corresponding Dnode through periodical message exchanging and this has little impact. When a Dnode fails, we resort to Chord protocol to recover the upper layer. Failure of a Dnode can also be detected by its Onodes through periodical message exchanging. Onodes switches their Dnodes as the departure procedure described in the previous paragraph.

A method to relieve the impacts of departure and failure of a Dnode is to set a backup Dnode for each cluster. The backup Dnode is selected from the Onodes within each cluster. The backup Dnode should at least have a capacity bigger than C_{thd} and it acts as an ordinary Onode when the current Dnode is alive. After the failure of Dnode, it takes over the role of the failed Dnode by joining the upper layer and forcing other Onodes to attach to it.

Because the departures or failures of Dnodes impose an larger impact on the hierarchical structure, node expected online time should be taken into account when selecting Dnodes.



finger table

start	Succ.
3	5
4	5
6	7
10	10

finger table

start	Succ.
6	7
7	7
9	10
13	13

finger table

start	Succ.
11	12
12	12
14	0
2	2

Fig. 2: An example of upper layer structure based on Chord ring and a 2-ary implicit tree initialized by Dnode 2.

4 Data Dissemination

When a new data object emerges on an Onode, the Onode delivers the object to its Dnode. When a Dnode y receives a new data object from its Onode or generates a data object by itself, it delivers the object to other Dnodes through a *k-ary* implicit tree based on node finger table which is defined and detailed in [13]. The finger table at node y contains the information of $O(\log n_d)$ Dnodes and the ith entry in the table is $1/2^{\log n_d - i}$ of the ring away from y, where n_d is the number of Dnodes and $0 \leq i < \log n_d$. Without loss of generality, we assume $2 \leq k \leq \log n_d$.

Initially, y holds the whole range of the Chord ring. We choose the first entry and the last $k-1$ entries in y's finger table as the children of y. y sends the data object to these k nodes concurrently. To ease of expression, we denote the node list constituted by these k nodes as $klist$ and assume the relative sequence of these k nodes in $klist$ is the same as the sequence in y's finger table. Now the ring is partitioned into k parts by these k Dnodes. The jth Dnode in $klist$ is responsible for the ring range which starts from it and ends with the predecessor of the $(j+1)$th node, where $1 \leq j < k$. The kth Dnode in $klist$ is responsible for the range which starts from it and ends with the predecessor of y. Each node z in $klist$ performs the same operations as y does: partitioning the ring range which it is responsible for and forwarding the data object. The only difference is that not all the nodes in the finger table at node z fall in the range which z is responsible for. In this case, only the entries falling in the range can be z's children in the implicit tree. We call these entries as *valid entries.* Note that the number of valid entries may be smaller than k. In this case, the number of z's children is not k, but is the number of valid entries. And if there is no entry in the finger table falls into the range, the partition operation stops because this indicates there is no Dnode in this range. With the continuous partition of the Chord ring, all the Dnodes receive the data object. Fig. 2 gives an example of upper layer structure and a *2-ary* implicit tree on it.

At the same time that z transfers data object to its child Dnodes, z also delivers the object to as many Onodes as it can. Let t $(0 \leq t \leq k)$ denote the number of z's child Dnodes in the upper layer's implicit tree and S_z denote the set of Onodes attaching to z. z randomly selects t' Onodes from S_z to transfer data, where $2 \leq t' = c_z - t - 1$. Let $S'_z \subset S_z$ denote the set of these t' Onodes and S''_z denote the set of Onodes that do not directly receive data object from z. Obviously, $S''_z = S_z - S'_z$. z divides S''_z into t' parts as evenly as possible and assigns them to the nodes in S'_z. After a node x in S'_z receives the data from z, it sends the data to as many Onodes (assigned by z) as it can. The operation is similar to the one performed by z except x can sends data to c_x nodes. For example, suppose 10 Onodes attach to Dnode z, which has capacity 10 and has 4 child Dnodes in the upper layer's implicit tree (i.e. $t = 4$). Then $t' = 5$. So, z sends the data message to 5 Onodes concurrently and also sends the address information of other 5 Onodes to them, one to each of them. These 5 nodes further delivery the message. This procedure naturally defines an implicit tree structure. Moreover, nodes within a cluster are always physically close to each other, so that the data delivery is fast.

Theorem 1. *A message from any source node can be delivered to all the nodes in $O(\log n)$ hops, where n is the number of nodes.*

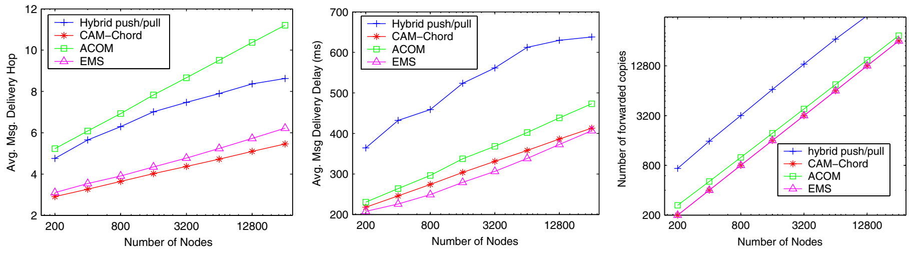
Due to space limitation, the proof detail is omitted. The intuition behind is straightforward: the message containing the data object is delivered on the upper layer and within the clusters along implicit trees. More details can be found in [19].

A node x may fail after receiving the data object but before forwarding it to child nodes on the implicit trees, which would cause descendant nodes of x miss the object. To avoid this, Dnode y periodically exchanges a message summary with its predecessor. If y finds that a message that it has not received, it obtains the message from its predecessor. Onodes also periodically exchanges a message summary with its Dnode. If an Onode finds that a message that it missed, it requests the Dnode to send a copy of the message. The Dnode sends the message directly to the Onode or designates another Onode to send the message. The message summary is piggybacked on the heartbeat message.

The basic Chord protocol does not account for node proximity. The study in [17] has pointed out that, instead of choosing the ith entry in y's finger table to be $(y + 2^i)$, a proximity optimization of Chord allows the ith entry to be any node within the range of $[y+2^i, y+2^{i+1}]$, which does not affect the complexities. We leverages this optimization in our scheme to improve the proximity effect. When a Dnode y joins the upper layer (which is Chord based), it iteratively checks whether there is a nearby Dnode in R_y falls in the range $[y+2^i, y+2^{i+1}]$. If there is one, it chooses this nearby Dnode as the ith entry. Recall that R_y is the set of nearby Dnodes located for y during y's join process.

5 Performance Evaluation

We develop an event-driven simulator to evaluate the performance of our scheme. We also compare our scheme with hybrid push/pull [3], CAM [4] and ACOM [5]. The metrics we mainly focus on are as following.



(a) Avg. hops (b) Avg. delay (c) Avg. traffic volume

Fig. 3: Comparison of different schemes while varying system size

M1. *Average length of delivery paths*: the average number of overlay hops it takes a message to reach a node;

M2. *Average delivery delay*: the average time it takes a message to reach a node. It is determined by the product of the path length and the logical link latency;

M3. *Average traffic volume*: the average number of copies that are forwarded by all nodes for delivering one message;

M4. *Average maintenance overhead*: average number of control messages used for each node join.

If not specified, the system consists of 10,000 nodes. When the average node capacity is c, the distribution of node capacity is *Zipf-like*. The capacity of a node is $c/2$, c, $2c$ and $4c$ with probability of 50 percent, 25 percent, 12.5 percent and 6.25 percent, respectively. The default value of c is 8. The default degree of implicit tree on the upper layer is 6, or $k = 6$. The basic search radius for locating nearby Dnodes is set to 50, or $Q_0 = 50$.

The underlying physical topology is generated by GT-ITM[18]. This topology has about 2,500 nodes: 4 transit domains each with 4 transit routers, 5 stub domains attached to each transit router, and 30 routers in each stub domain on average. *Member nodes are attached to randomly chosen stub routers.* Different latencies are assigned to the edges according to edge types: the distance of the edge between a node to the router it attached is 1 millisecond; the distance of intra-domain edge is 2 milliseconds; the distance of the edge between transit and stub domain is 10 milliseconds; and the distance of inter-transit edge is 50 milliseconds.

In the simulations, nodes join first and then 500 randomly chosen nodes start to send data messages. We perform the simulations for 5 times and report the average results.

5.1 Comparisons of Different Schemes

In this set of experiments, we comprehensively compare the performance of different schemes.

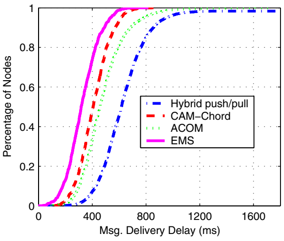


Fig. 4: CDF of avg. delivery delay

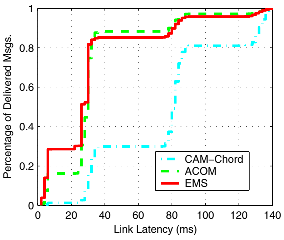


Fig. 5: CDF of delivered messages

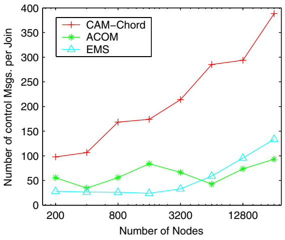


Fig. 6: Control messages per join

Fig. 3 shows the impact of system scale on the performance. The x-axis is in logarithm. In all schemes, the average length of delivery paths grows logarithmically with system scale. Our scheme EMS and CAM-Chord have comparable length of delivery paths and they outperform ACOM and hybrid push/pull. The reason why average length of delivery paths is so high in hybrid push/pull scheme is that a large part of the messages are duplicate messages. As to average delivery delay, EMS outperforms CAM-Chord and ACOM outperforms hybrid push/pull scheme for that both EMS and ACOM take the proximity information into account greatly. As to traffic volume, because EMS and CAM-Chord uses tree structures to delivery data messages, they bring no duplicate messages. In comparison, hybrid push/pull scheme and ACOM use about 75 percent and 25 percent more messages, respectively.

To better understand the average delivery delay of different schemes, we study the cumulative distribution of delivery delay of one message, which is shown in Fig. 4. Two observations are notable. First, not all the nodes can be reached in hybrid push/pull scheme. Second, EMS outperforms other schemes. Specially, within 400ms, 73.4 percent, 47.7 percent and 33 percent nodes can be reached in EMS, CAM-Chord and ACOM, respectively.

Fig. 5 illustrates cumulative distribution of delivered data messages. Intuitively, more messages are delivered within shorter distances indicates less cost and faster convergence. Fig. 5 shows that EMS delivers about 51 percent of total messages within 26ms and about 87 percent within 76ms, while CAM-Chord only delivers about 27 percent within 76ms and ACOM delivers about 18 percent of total messages within 26ms and about 88 percent within 76ms. The results implies that EMS effectively delivers data messages between physically close nodes.

Fig. 6 compares the maintenance overhead of different schemes in terms of number of control messages per join. Since CAM-Chord has many more neighbors ($O(c \log_c n)$) per node than EMS ($O(\log n_d)$) and ACOM ($O(c)$), its overhead is much higher than that of EMS and ACOM. Recall that n_d is the number of Dnodes in EMS. Another observation is that EMS and ACOM have comparable maintenance overhead in terms of the cost per join.

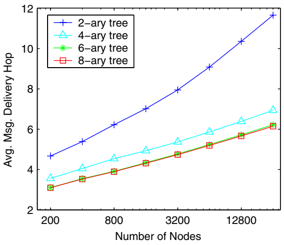


Fig. 7: Avg. path length while varying k

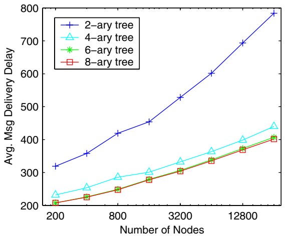


Fig. 8: Avg. delay while varying k

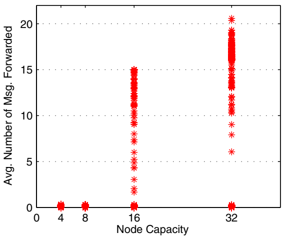


Fig. 9: Load distribution

5.2 Performance of EMS

In this set of experiments, we evaluate the performance of EMS while varying the degree (k) of implicit tree on the upper layer and show node load distribution in terms of the average number of messages it forwards.

Fig. 7 and Fig. 8 show the impact of the implicit tree degree (k) of the upper layer on the average path length and delivery delay, respectively. As expected, the path length and delivery delay decrease with the growth of the degree. However, the extent of decreasing is very small when increasing the degree from 6 to 8. This is explained by the fact that the Chord ring range at a node x can be partitioned into v_x parts at most, where v_x is the distinct nodes in x's finger table and is $O(\log n_d)$ on average. Recall that n_d is the number of the number of Dnodes. In our experiment, n_d is always smaller than 1/10 to 1/5 of the total number of nodes. Therefore, when the degree of implicit tree is 8 (always bigger than $O(\log n_d)$), the actual degree is about $O(\log n_d)$, not 8.

Fig. 9 shows the load distribution in terms of the average number of messages it forwards. Since the nodes with capacity 16 or 32 are on the upper layer with high probability, they contribute more than others. Nodes with capacities 8 contribute as smaller as nodes with capacities 4. This is because they are all in the lower layers.

6 Conclusion

In this paper, we propose EMS, an efficient multi-source data dissemination in P2P networks. Nodes are organized into a two-layer structure: the upper layer is DHT-based and consists of powerful and stable nodes, ordinary nodes attache to physically close nodes at the upper layer. The data objects are delivered through implicit trees so that no control message is required to build the tree structures. The average length of delivery paths is $O(\log n)$, where n is the number of nodes in the P2P network. Compared with existing schemes, EMS is proximity-aware and has small maintenance overhead and small traffic volumes.

As a future work, we will evaluate the performance with different capacity profiles.

Acknowledgments

This work is supported by National Basic Research Program of China with grant No.2007CB310702, by National Natural Science Foundation of China with grant No.60403031 and 90604015, and by France Telecom R&D.

References

1. S. El-Ansary, L. O. Alima, P. Brand, S. Haridi. Efficient Broadcast in Structured P2P Networks, *Proc. of IPTPS*, 2003
2. S. Saroiu, P.K. Gummadi, and S.D. Gribble, A Measurement Strudy of Peer-to-Peer File Sharing Systems, *Proc. of MMCN*, 2002
3. A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems, *Proc. of ICDCS*, 2003.
4. Z. Zhang, S. Chen, Y. Ling, R. Chow, Resilient Capacity-Aware Multicast Based on Overlay Networks, *Proc. of ICDCS*, 2005
5. S. Chen, B. Shi, S. Chen. ACOM: Any-source Capacity-constrained Overlay Multicast in Non-DHT P2P Networks, *IEEE Trans. on Parallel and Distributed Systems*, vol. 18, no. 9, pp. 1188-1201, Sept., 2007
6. X. Zhang, J. Liu, B. Li and T.-S. P. Yum. DONet/CoolStreaming:A Data-Driven Overlay Network for Live Media Streaming, *Proc. of INFOCOM*, 2005.
7. V. Venkataraman, K. Yoshida, P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-based Peer to Peer Multicast, *Proc. of ICNP*, 2006.
8. S. Banerjee, B. Bhattacharjee, C. Kommareddy. Scalable Application Layer Multicast, *Proc. of SIGCOMM*, 2002.
9. Y.-h. Chu, S. G. Rao, S. Seshan, H. Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communication (JSAC)*, vol. 20, 2002.
10. H. Shen and C. Xu, Hash-based Proximity Clustering for Load Balancing in Heterogeneous DHT Networks, *proc. of IPDPS*, 2006.
11. F. Wang, Y. Xiong, and J. Liu, mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast, *Proc. of ICDCS*, 2007
12. M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, Splitstream: High-bandwidth Multicast in Cooperative Environments,*Proc. of SOSP*, 2003.
13. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *Proc. of SIGCOMM*, 2001
14. Gnutella Protocol Specification v0.6. *http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html*
15. Z. Xu, C. Tang, and Z. Zhang. Building Topology-Aware Overlays using Global Soft-State. *Proc. of ICDSC*, 2003.
16. T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier, Space Filling Curves and Their Use in Geometric Data Structures, *Theoretical Computer Science*, 1997.
17. K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, The Impact of DHT Routing Geometry on Resilience and Proximity, *Proc. of SIGCOMM*, 2003.
18. E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to Model an Internetwork, *Proc. of INFOCOM*, 1996.
19. Z. Li, G. Xie, Z. Li, Efficient Multi-source Data Dissemination in Peer-to-Peer Networks, *Technical Report*, Nov. 2007. available on request.