January 2012

An Improved Hybrid Peer-to-Peer System with Caching for Distributed Data Sharing

Arpana George

Dept. of CSE, Viswajyothi College of Engineering and Technology, arpanageorge@yahoo.in

Follow this and additional works at: <https://www.interscience.in/ijssan>

Part of the [Digital Communications and Networking Commons](#), and the [Electrical and Computer Engineering Commons](#)

Recommended Citation

George, Arpana (2012) "An Improved Hybrid Peer-to-Peer System with Caching for Distributed Data Sharing," *International Journal of Smart Sensor and Adhoc Network*: Vol. 1 : Iss. 3 , Article 13.

DOI: 10.47893/IJSSAN.2012.1062

Available at: <https://www.interscience.in/ijssan/vol1/iss3/13>

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Smart Sensor and Adhoc Network by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.

An Improved Hybrid Peer-to-Peer System with Caching for Distributed Data Sharing

Arpana George

Dept. of CSE, Viswajyothi College of Engineering and Technology
E-mail : arpanageorge@yahoo.in

Abstract - Peer-to-peer overlay networks are widely used in distributed systems. Based on whether a regular topology is maintained among peers, peer-to-peer networks can be divided into two categories: structured peer-to-peer networks in which peers are connected by a regular topology, and unstructured peer-to-peer networks in which the topology is arbitrary. Structured peer-to-peer networks usually can provide efficient and accurate services but need to spend a lot of effort in maintaining the regular topology. On the other hand, unstructured peer-to-peer networks are extremely resilient to the frequent peer joining and leaving but this is usually achieved at the expense of efficiency. The objective of this work is to design a hybrid peer-to-peer system for distributed data sharing which combines the advantages of both types of peer-to-peer networks and minimizes their disadvantages. Also a caching scheme is proposed for the hybrid peer-to-peer system to improve the system performance.

Keywords - Peer-to-peer systems, P2P, structured peer-to-peer, unstructured peer-to-peer, hybrid, overlay networks, caching.

I. INTRODUCTION

Research has shown that a large fraction of traffic in the Internet is occupied by peer-to-peer applications [2]. A peer-to-peer (P2P for short) network is a logical overlay network on top of a physical network. Each peer corresponds to a node in the peer-to-peer network and resides in a node (host) in the physical network. All peers are of equal roles. The links between peers are logical links, each of which corresponds to a physical path in the physical network. The physical path is determined by a routing algorithm and composed of one or more physical links. Logical links can be added to the peer-to-peer network arbitrarily as long as a corresponding physical path can be found, that is, the physical network is connected.

The flexibility of the overlay topology and the decentralized control of the peer-to-peer network make it suitable for distributed applications. For example, it can be used for distributed data (file) sharing, or for collaborative Web caching in which Web pages are cached in collaborative peers to reduce network delay for URL requests, or for application layer multicast in which peers are group members and the peer-to-peer overlay network is a multicast tree. It can also be used for distributed computing which utilizes the idle resources in the network for a huge computing task. Finally, it can be used to provide communication anonymity in which the sender's identity is concealed.

Based on whether a regular topology is maintained among peers, peer-to-peer networks can be divided into two categories: structured peer-to-peer networks in which peers are connected by a regular topology, and unstructured peer-to-peer networks in which the network topology is arbitrary. Structured peer-to-peer networks build a distributed hash table (DHT) on top of the overlay network. The hash table supports efficient data insertion and lookup. Given a key of the data item, the corresponding value of the data item can be inserted or found by transforming the key to a hash value by a hash function. The hash value is the index of the data item and all the hash values form the key space. In DHT, the key space is divided among peers. Each peer is responsible for one partition of the key space. Peers are connected by an overlay network through which the requests of data insertion and lookup are delivered. Structured peer-to-peer networks can provide efficient and accurate query service but need a lot of efforts to maintain the DHT, which makes it vulnerable to frequent peer joining and leaving, also known as churn[8][9]. Churn is a common phenomenon in peer-to-peer networks. Unstructured peer-to-peer networks organize peers into an arbitrary network topology, and use flooding or random walks to look up data items. Each peer receiving the flooding packets or random walk packets checks its own database for the data item queried. This approach does not impose any constraint

on the network topology. It can perform complex data lookup and support peer heterogeneity. Unstructured peer-to-peer networks are resilient to churn while they usually achieve this goal by sacrificing the data query efficiency and accuracy.

Hence, neither structured peer-to-peer networks nor unstructured peer-to-peer networks can provide efficient, flexible, and robust service alone. The motivation of this paper is to combine the two types of peer-to-peer networks and provide a hybrid solution which can offer efficiency and flexibility at the same time.

In this paper, two models for cooperative caching of P2P traffic is proposed. The first model enables cooperation among caches that belong to different autonomous systems (ASs), while the second considers cooperation among caches deployed within the same AS.

II. PREVIOUS WORKS

Many peer-to-peer networks have been proposed for different applications in the literature, see, for example, [1], [2]. In this scheme, we focus on peer-to-peer networks for efficient distributed data (file) sharing among peers.

The Content Addressable Network (CAN) [3] was proposed to provide a scalable indexing mechanism for file sharing over a large network. As a distributed infrastructure, CAN provides hash-table-like functionality over Internet-like scales. Both peers and data are hashed to a virtual d -dimensional Cartesian coordinate space. The entire space is partitioned to distinct zones such that each peer is in charge of one zone. Every peer maintains a routing table which holds the IP address of its neighbors in the coordinate space. The data is stored in and retrieved from the peer that owns the zone covering the data. CAN takes advantage of the ordering of the Cartesian coordinate space in the routing algorithm. Packets are forwarded along the straight line connecting the source and the destination in the Cartesian coordinate space. When a new peer joins the system, some existing zone will be split into two zones one of which is assigned to the new peer, and all the related peers need to update their neighbor lists. When a peer leaves the system, a neighboring peer will take over the zone by running a takeover algorithm, and all the related peers need to update their neighbor lists again.

Chord [4] organizes the peers into a circle which is called a chord ring, where each peer is assigned an ID. Peers are inserted into the ring in the order of their IDs. Each peer has two neighbors: successor and predecessor. When a peer joins the system, it first finds

the position to insert the new peer. Then, the successor pointers of both the new peer and an existing peer must be changed. The correctness of Chord relies on the fact that each peer is aware of its successor. To guarantee this, each peer maintains a successor list of size r which contains the peer's first r successors. Each data item also has an ID and is stored in a peer such that the ID of the data item is between the ID of the peer and its predecessor. Packets are forwarded along the circle. In order to accelerate the search, each peer maintains a finger table, where each finger points to a peer with a certain distance from the current peer. Compared to CAN, Chord is simpler as the key is hashed in a 1D space.

Gnutella [5] is a decentralized unstructured peer-to-peer network. The network is formed by peers joining the network following some loose rules. There is no constraint on the network topology. To look up a data item, a peer sends a flooding query request to all neighbors within a certain radius. As Gnutella has no requirement on the network topology and data placement, it is extremely resilient to peer joining and leaving the system frequently. However, flooding is not scalable and consumes a lot of network bandwidth.

Caching of the P2P traffic has recently been studied in a number of papers. The benefits of caching P2P traffic have been shown in [12] and [10]. Caching algorithms designed for P2P traffic have been proposed in [11]. In [11], two object replacement algorithms are suggested: Minimum Relative Size (MINRS) and Least Sent Byte (LSB). The first algorithm evicts the object which has the least cached fraction, and the second one evicts the object which served the least number of bytes from the cache.

III. THE HYBRID PEER-TO-PEER SYSTEM

A. System Overview

The new hybrid peer-to-peer system is composed of two parts: a core transit network and many stub networks, each of which is attached to a node in the core transit network. The core transit network, called t -network, is a structured peer-to-peer network which organizes peers into a ring similar to a chord ring. We call peers in the t -network t -peers. Each t -peer is assigned a peer ID (p_id), which is a positive integer. Peers are inserted to the ring in the order of their p_ids . Each t -peer maintains two pointers which point to its successor and predecessor, respectively. A finger table is also used to accelerate the search. A stub network, called s -network, is a Gnutella-style unstructured peer-to-peer network. We call the peers in an s -network s -peers except for the t -peer attached to this s -network. The topology of an s -network is arbitrarily formed. Each s -network is attached to a t -peer and this t -peer belongs

to both the t-network and the s-network. One thing to mention about the s-network is that the topology of an s-network is a tree instead of a mesh. Fig. 1 shows the overview of the proposed hybrid peer-to-peer system.

The basic idea behind the hybrid peer-to-peer system is that the t-network is used to provide efficient and accurate service while the s-network is used to provide approximate best-effort service to accommodate flexibility. Peers can join either t-network or s-network directly. An s-network is composed of peers that serve the data of some common properties. A data lookup is confined within an s-network if the queried data has the common properties served by the s-network. The lookup request is passed around the s-network through flooding or random walk. Although flooding may generate a lot of network traffic, it can greatly simplify peer joining and leaving process, which makes the system robust to churn. On the other hand, since the s-network contains only a small proportion of the total number of peers, flooding is confined within a small number of peers. When the queried data is served by another s-network, the data query request is first forwarded to the t-network through the t-peer in the s-network generating the request. Then, in the t-network, the request will be forwarded along the ring until it reaches the s-network serving the queried data. Finally, in the s-network, the request will be delivered to the s-peers by flooding again. The t-network links all the s-networks together and provides an efficient way to locate the desired s-network. The stableness of the t-network is critical to the system performance because all the communications between different s-networks are through the t-network. As a structured peer-to-peer network, the t-network is vulnerable to churn mainly because the t-network needs to recalculate the pointers in the finger tables whenever a t-peer joins or leaves. However, the hybrid system can effectively reduce the topology maintenance overhead caused by peer joining or leaving. This is because that, on one hand, a large portion of peers join the s-networks directly without disturbing the t-network; and on the other hand, an s-peer can be selected to substitute the leaving t-peer in the same s-network, i.e., the selected s-peer will become a t-peer. In this case, the total number of t-peers is unchanged. Therefore, there is no need to recalculate the pointers in the finger tables, and only a simple update is needed.

In this paper, the focus is on applying the hybrid peer-to-peer system to distributed data sharing. A data item is represented by a (key, value) pair. A key is a label or name of the data, such as a file name, while a value is the content associated with the key, such as a file. A peer uses operation store (key, value) to insert the data item into the system and operation lookup (key) to obtain the value of the data item. Before performing the store or lookup operations, a peer hashes the data key to

an integer d_id which is in the same range as p_id . As mentioned earlier, s-peers are grouped into different s-networks such that each s-network serves the data of some common properties. In the hybrid system, the p_ids of t-peers divide the range of the d_id into several segments. Each s-network is responsible for the data whose d_ids lie in the same segment. Both store and lookup operations try the local s-network first if the data item is served by the local s-network; otherwise, they turn to the t-network. Thus, such two-tier hierarchy structure can provide efficient lookup in the top tier while maintaining the flexibility in the bottom tier.

Fig. 1 : Overview of the proposed hybrid peer-to-peer system

B. Peer Join/Leave

1) T-Peer Join/Leave:

Each t-peer maintains two pointers that point to its successor and predecessor along the ring, respectively. After the joining peer obtains the arbitrary t-peer, it sends a join request containing its p_id to the t-peer. This join request will be forwarded along the ring until it reaches the t-peer such that the p_id of the joining peer is between the p_id of the t-peer and its successor. The t-peer will initiate a join process and the joining peer is inserted between the t-peer and its successor.

The p_id of a new peer is generated at the server. The server has several options to generate the p_id . One way is to generate the p_id by hashing the IP address of the new peer. Another way is to generate the p_id based on the location of the new peer. This can make the peers that are close to each other in the physical network also close to each other in the overlay network. Moreover, the server can generate a random p_id for the new peer. However, the p_id generation process does not guarantee the uniqueness of p_id . In the case of a conflict, the t-peer initiating the join process will generate a new p_id which lies in between the p_id of itself and its successor. The new p_id can be random or simply the midpoint for load balancing purpose.

After the join process completes, the segment of the d_id space represented by the successor has changed. The peers in the successor's corresponding s-network should

transfer part of its data load to the new peer, which is referred to as load transfer. The peers check the data items it stores and transfer all the data items whose *id* lie between the *p id* of the new peer and the predecessor of the new peer.

Sometimes, peers will leave the system without notice due to peer crash. We call it abruptly leaving. To handle abruptly leaving, peers send “HELLO” messages (also called heartbeat messages) to their neighbors periodically. Each peer maintains a timer for each of its neighbors. The time is reset on receiving a “HELLO” message from the corresponding neighbor. Timeout indicates peer crash. The disconnected *s*-peers will compete to replace the crashed *t*-peer by sending messages to the server. The server will pick an *s*-peer to be the new *t*-peer. The selection can be random or choosing the peer with the smallest IP address.

TABLE 1. Join/Leave Algorithm for *t*-Peers

```

Pre.join(n):
Pre.check(n.id);
Pre.successor=n;
n.predecessor=pre;
n.successor=suc;
Suc.predecessor=n;
Suc.loadtransfer(n.id);
n.leave():
If(s-network!=NULL)
    Pick a s-peer randomly;
    Set s-peer as the new t-peer;
    Transfer load from n to the new t-peer;
Else
    pre.successor=suc;
    suc.predecessor=pre;
    n.loaddump();
Pre.check(n.id):
If(n.id==id)
n.id=(id+suc.id)/2;
Suc.loadtransfer(n.id);
For each peer in the current s-network
    for each data in database
        If(data.id<=n.id)
            n.insert(data)
Suc.delete(data);
n.loaddump();
For each data in database
    Suc.insert(data);
n.delete(data);

```

2) *S*-Peer Join/Leave:

Each *s*-peer belongs to an *s*-network and maintains a list of its neighbors. A neighbor can be either an *s*-peer or a *t*-peer. After an *s*-peer acquires the IP address of the random peer, it adds the peer to its neighbor list. Then, it notifies the random peer to add itself to its neighbor list

and the join operation is completed. The *p id* of the *s*-peer is the same as its neighbor.

In a Gnutella-style peer-to-peer network, the data lookup is through flooding. The range of flooding is determined by the search radius, that is, the time-to-live (TTL) value of the packet. As the data is distributed around the network randomly, the search radius is critical to the probability of finding the desired data item. For the same topology and the same peer that initiates the search, the longer the search radius, the higher the probability of finding the desired data item, but the longer the latency required. Note that if we add some simple constraints on choosing the random peer when a new peer joins, we may shorten the network diameter, and thus, reduce the search radius without sacrificing the success ratio of finding the desired data item. Next, it will be discussed how to add such constraints. First, restrict the random peer to be picked to only *t*-peers. Thus, all *s*-peers in the same *s*-network are connected with one *t*-peer. As a result, the diameter of an *s*-network is at most two, and one data lookup can reach all the peers within two hops. The topology of the *s*-network is a star centered at a *t*-peer. Although the data lookup can achieve short latency in such an *s*-network, there is a notable disadvantage that the load is extremely unbalanced. The *t*-peer maintains a long neighbor list while the *s*-peer has a neighbor list with only one neighbor. Each data lookup request has to be forwarded through the *t*-peer. In order to alleviate this problem, we put another restriction on the degree of peers. When the degree of a peer reaches a threshold δ , it passes the join request to one of its neighbors randomly. The join request will be passed until it arrives at a peer whose degree is less than δ . In the simulation, we use this scheme for *s*-peer joining. The new *s*-peer searches from a *t*-peer along a random branch until it finds a peer with a degree less than δ . This peer is called the connect point (cp) of the new *s*-peer. Besides the neighbor list, each *s*-peer maintains two pointers that store the address of the *t*-peer of the *s*-network and its cp. The resulting topology of an *s*-network is a tree. Here, we use a tree instead of a mesh due to bandwidth efficiency consideration. A major drawback of an unstructured peer-to-peer network is that the flooded query messages occupy a lot of network bandwidth. In a mesh network, it is very likely that a peer receives the same query message multiple times from different neighbors. On the other hand, a tree structure guarantees that each peer receives the query message exactly once.

When an *s*-peer leaves the system, it should notify all its neighbors about the leaving. The neighbors then delete it from their neighbor lists. The neighbor whose cp is the leaving peer should rejoin the *s*-network by sending a join request to the *t*-peer again. The leaving *s*-

peer should also choose a neighbor to transfer the load to.

Again, we need to handle the abruptly leaving when peers crash without notice. To detect and recover from these errors, we still use the periodic “HELLO” messages. Each s-peer periodically broadcasts “HELLO” messages to all its neighbors.

C. Concurrent Join/Leave

Peer-to-peer networks are highly dynamic systems since peers usually are end hosts that are in charge of different individuals or groups. Concurrent joins and leaves are very common and can greatly degrade the performance if not handled carefully.

The concurrency handling in the s-network is simpler than that in the t-network. When an existing peer receives two join requests, it follows the First Come First Serve (FCFS) rule, that is, the second join request will be passed to the next neighbor if the degree of the peer reaches the limit after receiving the first join request. When two or more s-peers leave the system simultaneously, no special action is needed because the disconnected parts will rejoin the s-network, as described in the previous section.

For the t-network, concurrency handling is much more complicated as it involves three t-peers and the topology constraint is strict. The concurrent joins or leaves may lead to an incorrect topology or break the t-network apart if not handled carefully. For example, suppose a t-peer is inserting a new peer, say, x, between itself and its successor. After setting its successor pointer to the new peer, it receives another join request indicating that another new peer, say, y, should be inserted between x and its successor. Thus, the t-peer will pass the join request to its new successor x. However, the join operation of x is not completed, and the successor and predecessor pointers in x are not set correctly. Therefore, the join request of y will not be handled correctly.

The idea of the concurrency handling in the database system is adopted for the concurrent joins or leaves in the t-network. The join and leave requests are sequentialized such that the next request is not processed until the previous request finishes. For a join request, it follows a join triangle, as shown at the left of Fig. 3. When peer pre receives the join request of the new peer, it sets a mutex variable joining that indicates some peer is being inserted between peer pre and peer suc. Now peer pre will not accept any leave requests including that from itself. If a new join request comes before the previous request is completed, peer pre will insert the new request to a queue and process the request queue after the previous join request finishes. Then, peer pre sends a packet including the address of peer suc to

the new peer. The new peer sets its successor and predecessor pointers to suc and pre, respectively, and sends another packet to peer suc. After receiving the packet, peer suc updates its predecessor pointer and sends a packet back to peer pre. When peer pre receives this packet, it sets its successor pointer to the new peer and continues to process the next join request in the queue. If the queue is empty, it resets the mutex variable.

When a peer leaves the system, it follows the leave triangle which is shown on the right of Figure 2. When a peer is leaving, it also sets a mutex variable leaving. Now the peer will not accept any new join request (if the join request queue is not empty, the peer should process the join request first) and leaving request. Then, the leaving peer sends a packet to peer pre including the address of peer suc. Peer suc sets its successor pointer to peer suc and sends a packet to peer suc including the address of the leaving peer. After receiving the packet, peer suc will check whether the leaving peer included in the packet is what its predecessor pointer is pointing to. Only if they are the same peer, will the peer suc set its predecessor pointer to peer pre and send a packet to the leaving peer to notify the completion of the leaving operation.

Fig. 2 : Concurrent join/leave operation for t-peers

D. Data Insertion/Lookup

Data is generated and inserted to the system by peers. As mentioned earlier, each s-network is responsible for a range of ID space. The peer generating the data item first hashes the key into this space. If the d id lies in the range of the current s-network, the data item is inserted to its database and the data insertion is completed. If the d id does not lie in the range, the data item is sent to the t-peer of the current s-network. Then, it is forwarded along the t-network until it reaches the t-peer in charge of the ID range covering d id. Then, the data item is inserted into the database of the t-peer.

Note that although this data placement scheme is simple and easy to implement, the data load between different peers may be imbalanced. Each t-peer corresponds to an s-network. The data generated in all other s-networks except the one a t-peer is in will always be stored in the t-peer; therefore, the data load in

t-peers is much heavier than that in s-peers. To alleviate the load imbalance, we spread the data load to the neighbors of the t-peer, i.e., the s-peers connected to the t-peer. When a t-peer receives a data insertion request, it picks a random s-peer from its directly connected s-peers and itself, and then sends a data insertion request to the chosen peer. The random peer will perform the same random data load spreading until the data item is finally inserted to the system. In the performance evaluation section, we will implement both the original data placement scheme and the improved one and study their impacts on the probability density function of the number of data items per peer.

When a peer looks up a specified data item, it first obtains the d_id of the data item by hashing the data key. If the d_id lies in the current s-network, the peer floods lookup packets around the s-network and sets a timer for it. The timer will be reset if the peer receives the data item or expire, which indicates that the data item is not found. The peer may choose to increase the TTL value and the expiration duration of the timer and reflood the lookup packets. If the d_id does not lie in the current s-network, the peer sends a lookup request to the t-peer and also sets a timer for it. Similar to data insertion, the data lookup request is forwarded along the t-network until it arrives at a proper t-peer which will then flood data lookup packets around its s-network. Each peer receiving the lookup request will check its database for data item d_id . If the data item is found in its database, the peer will stop flooding and send the data item to the peer requesting the data item directly.

IV. MODELS FOR CACHING P2P TRAFFIC

A. Independent Proxy Caches

In independent caching, a cache is deployed near the gateway routers of ASs that choose to employ caching to reduce the burden of P2P traffic. See Fig. 3a, but note that caches in different ASs work independently from each other. In order to take full advantage of a deployed cache and to avoid modifying the source code of P2P client software, the cache should work in a transparent mode.

The primary goal of caching P2P traffic is to reduce the load on backbone links, and hence, reduce the operational costs of ISPs. To reflect this goal, we choose the byte hit rate as the main performance metric for evaluating caching systems for P2P traffic. The byte hit rate (BHR) is defined as the ratio of the number of bytes served from the cache to the total number of bytes transferred. Note that, unlike the case of caching Web traffic, the hit rate—defined as the ratio of the number of objects served from the cache to the total number of objects transferred—may not be well defined in the P2P case [6]. This is because requests in P2P systems are

typically issued for segments of objects, not for entire objects.

B. Cooperative Proxy Caches in Different ASs

The first model for cooperation considered in this paper is depicted in Figure 3a. In this model, caches deployed in different ASs cooperate with each other to serve requests from clients in their networks. The cooperating ASs may have a peering relationship to carry each other's traffic, or they can be located within the same geographical area such as a city where the bandwidth within the region is typically more abundant than the bandwidth on long-haul, intercity, links.

Caches cooperating with each other form what we call a cache group. The cooperation in the cache group works as follows: When a cache receives a request for an object that it does not store locally, it first finds out whether another cache in the cache group has the requested object. If any of them does have the object, the object is directly served to the requesting client. If otherwise, the request is forwarded to external sources. Communication and object lookup inside the cache group can be done in several ways. For example, a centralized directory can be used. The lookup process is straightforward in this case and it requires only two messages. However, the directory is a single point of failure and it requires frequent updates from participating caches. We adopt distributed lookup methods.

Fig. 3. The proposed two models for cooperative caching of P2P traffic. (a) Cooperation among caches in different ISPs. (b) Cooperation among caches within a large ISP.

C. Cooperative Proxy Caches within the Same AS

The second model for cooperation proposed in this paper is for caches deployed within the same AS, as shown in Fig. 3b. This model is suitable for a large ISP with multiple access exit points. The network of such ISPs is composed of multiple points of presence (POPs) interconnected with high-speed optical links. ISPs provide Internet access to their customers at POPs. The links inside an ISP are usually overprovisioned. ISPs are

attached to the Internet through inter-ISP links. Inter-ISP links are usually the bottlenecks of the Internet and where congestion occurs. In addition, the inter-ISP links are expensive because an ISP either pays another ISP for carrying its traffic (in a customer-provider relationship) or it needs to mutually carry the same amount of traffic from the other ISP (in a peer-to-peer relationship) [6]. Deploying cooperative caches in such large ISPs would save a huge amount of P2P traffic from going on the inter-ISP links, and thus, would reduce the costs incurred by ISPs, because the cost of the internal links (between caches) is much smaller than the cost of inter-ISP links [7]. Caching would also benefit clients because their traffic will traverse fewer inter-ISP links, which are more susceptible to overload and congestion.

V. CONCLUSIONS

In this paper, we have proposed a hybrid peer-to-peer system which combines both the structured peer-to-peer network and the unstructured peer-to-peer networks to form a two-tier hierarchy to provide efficient and flexible distributed data sharing service. The top tier is the t-network which is a structured ring-based peer-to-peer network providing efficient and accurate service. The bottom tier is composed of multiple unstructured s-networks which provide approximate best-effort service to accommodate flexibility. By assigning peers to the t-network or the s-network, the hybrid peer-to-peer system can utilize both the efficiency of the structured peer-to-peer network and the flexibility of the unstructured peer-to-peer network and achieve a good balance between them. Also in this paper, cooperative caching for P2P traffic is proposed. Two models for Cooperation are proposed: 1) among caches deployed in different ASs and 2) among caches deployed within a large AS. In both models, caches cooperate to save bandwidth on expensive WAN links. Considering the huge volume of the P2P traffic, even 1 percent improvement in byte hit rate accounts to saving in the order of terabytes of traffic on the expensive WAN links.

REFERENCES

- [1] E.K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Comm. Surveys and Tutorials*, vol. 7, no. 2 pp. 72-93, Mar. 2005.
- [2] S. Sen and J. Wang, "Analyzing Peer-to-Peer Traffic Across LargeNet works," *Proc. Internet Measurement Workshop*, Nov. 2002.
- [3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," *Proc. ACM SIGCOMM '01*, pp. 161-172, 2001.
- [4] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Trans. Networking*, vol. 11, no. 1, pp. 17-32, Feb. 2003.
- [5] Gnutella Development Forum, The Gnutella v0.6 Protocol, http://groups.yahoo.com/group/the_gdf/files/, 2009.
- [6] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, K. Claffy, and G. Riley, "AS Relationships: Inference and Validation," *ACM SIGCOMM Computer Comm. Rev.*, vol. 37, no. 1, pp. 31-40, Jan. 2007.
- [7] W. Norton, "The Evolution of the US Internet Peering Ecosystem," <http://www.nanog.org/mtg-0405/norton.html>, white paper, Nov. 2003.
- [8] S. Saroiu, K. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking Conf. MMCN*, Jan. 2002.
- [9] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan, "Measurement, Modeling and Analysis of a Peer-to-Peer File-Sharing Workload," *Proc. ACM Symp. Operating System Principles (SOSP)*, Dec. 2003.
- [10] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, "Should Internet Service Providers Fear Peer-Assisted Content Distribution?" *Proc. ACM Conf. Internet Measurement (IMC '05)*, pp. 63-76, Oct. 2005.
- [11] A. Wierzbicki, N. Leibowitz, M. Ripeanu, and R. Wozniak, "Cache Replacement Policies Revisited: The Case of P2P Traffic," *Proc. Int'l Workshop Global and Peer-to-Peer Computing (GP2P'04)*, pp. 182-189, Apr. 2004.
- [12] N. Leibowitz, A. Bergman, R. Ben-Shaul, and A. Shavit, "Are File Swapping Networks Cacheable? characterizing P2P Traffic," *Proc. Int'l Workshop Web Content Caching and Distribution (WCW'02)*, Aug. 2002.