A UDP Based Protocol for Distributed P2P File Sharing

Yao-Nan Lien and Hong-Qi Xu
Computer Science Department
National Chengchi University
Taipei, Taiwan
lien@cs.nccu.edu.tw

Abstract **-- Some Peer-to-Peer (P2P) file sharing operation models over asymmetric networks have several shortcomings that may have a significant impact to the system and network performance: data transmission paths are highly redundant wasting a lot of backbone bandwidth; the download throughput in a peer node may be limited by the upward bandwidth of other peer nodes; TCP performance is deteriorated due to the blocking of acknowledge packets on the upward channel. These shortcomings severely impair the efficiency of P2P file sharing as well as network performance. We designed a UDP based transport mechanism at the application level to alleviate parts of problems caused by TCP.**

Key Words: P2P and TCP.

I. INTRODUCTION

P2P file sharing is a very popular network application. There are many successful systems such as Napster, Kazza, Gnutella, Freenet, and BitTorrent [1,2,6]. By the degree of centralization, these systems can be roughly classified into Pure Decentralized, Partially Decentralized, and Hybrid Decentralized [9]. By the network structure, they can be classified into Unstructured, Structured, and Loosely Structured systems [5]. *BitTorren* (BT), which is originated by Bram Cohen [1], has become a bandwidth glutton, devouring more than one third of the Internet's bandwidth. Its operation can be classified as a Partially Decentralized Unstructured-network operation model. Because BitTorrent is becoming a dominant technology, this paper will take the BitTorrent compatible P2P file sharing model as a typical model to study. Nevertheless, some problems are not unique to the BitTorrent and many techniques can be applied to other paradigms as well.

Without loss of generality, we assume the following abstract operation model. The file to be shared is broken into many smaller fragments and is stored in a seed node (*initial seeder*) waiting for retrieving by other peers. Each peer may retrieve these file fragments either from the initial seeder or from other peers that have already retrieved some fragments. Each peer may also serve as a redistribution node to share its retrieved fragments with other peers.

In reality, Internet users are using various software tools that are compliant with BitTorrent protocol to cheaply spread files around the Internet. To share a file to other peers, a user server serves as the *initial seeder* to launch a BT-compatible server software, called *tracker*, to break the file into many smaller fragments and store the meta data into a small file called *torrent*, and then to publish it to some web site. To download the file, a user downloads the torrent onto his/her computer first. A BT-compatible client software, called *downloader*, opens the torrent to obtain the contained meta data, then searches for other downloaders that have downloaded the same torrent, and tries one by one to download the needed fragments from those downloaders. Each downloader will share the fragments it has downloaded successfully even before it finishes downloading all fragments. Through BitTorrent protocol, all peer users share their downloaded fragments to each other. The downloader in each peer combines all retrieved fragments back into a copy of the original file. As a common courtesy, a downloader needs to voluntarily stay online becoming a seeder to share the downloaded copy to other peers for some certain length of time. What a tracker does is giving each request a fragment, then introducing all peers to one another so that they can download file fragments from each other.

Unstructured P2P file sharing networks do not take network topology into consideration, the paths of data transmission may overlay each other severely. As a consequence, it may waste a lot of backbone bandwidth. Although existing structured network model can organize the participating peers into a less redundant network topology and thus can relieve this problem, they are all file based and may not be applicable for fragment-based model such as BitTorrent.

Furthermore, when most peer nodes are attached to the Internet via asymmetric access networks such as ADSL, there will be some performance problems. First, within a node, the download throughput is often smaller than the upload throughput, even though the former usually has much larger bandwidth capacity. Secondly, the performance of TCP based network applications will be interfered by the P2P file sharing operation. These problems are further complicated by the voluntary nature of P2P that peer nodes may be fairly unstable making downloading operations fairly unstable.

We can easily identify at least two possible causes that may contribute to the problems mentioned above. First, since a file fragment is often retrieved by more than one node, all such file sharing streams must share the narrow upward bandwidth of the node that is sharing out the file fragment. Therefore, each stream is only allocated with a small bandwidth. Secondly, when an upward channel is congested by the file sharing load, the performance TCP will deteriorate due to the blocking of acknowledge packets on the upward channel [7]. For simplicity, the first problem is referred to as the *Fractional Upward Bandwidth* (FUB) problem and the second one as the *Blockage of Acknowledge* (BoA) problem [4]. These shortcomings severely impair the performance of networks and the P2P file sharing itself.

The problems mentioned above were analyzed with some solution approaches proposed in paper [4]. This paper addresses the performance issues caused by TCP.

II. IMPACT OF TRANSPORT PROTOCOLS

A network application that demands a reliable data transfer would probably choose TCP to "transport" data [7]. This section will discuss the performance problems caused by the impairment of TCP as well as solution approaches.

A. Overview of TCP Protocol

TCP is a transport protocol that can guarantee the delivery of packets and is built-in with a congestion control mechanism. TCP software resides at the both ends (sender and receiver) of a connection. The basic version can perform well without any support from the network elements at IP layer. To transmit a message (or a file), the sender breaks the message into packets and transmits them in sequence to the receiver. The receiver acknowledges the received packets by sending acknowledge packets (ACK) back to the sender. If the sender receives a packet loss signal (three duplicated ACKs) for a particular packet or doesn't receive the ACK by a certain time period, the packet is considered lost and must be retransmitted. The process of transmitting data from the beginning to the end is a *session*.

When TCP is invoked to transport a file or a message, the sender doesn't know the appropriate data rate it should take to transmit data. Therefore, it takes some calculated steps, such as *AIMD* (Additive Increasing Multiplicative Decreasing) policy, to adjust a sending window, which has a direct effect of adjusting the data rate, in a trial-and-error fashion. To prevent the network from overly congested, TCP takes packet losses as signals of network congestion and adjusts the data rate accordingly. AIMD policy adjusts data rate much slower in increasing phases and much faster in decreasing phases. Adjusting transmission data rate in this trial-and-error fashion is not very efficient. Many improvement mechanisms have been proposed and implemented to enhance TCP performance under various conditions [4].

Controlling congestion by taking packet losses as the signals of network congestion performs well for regular networks. However, it may not work well in some network environments, such as BitTorrent over asymmetric networks, in which acknowledge packets sent by a TCP receiver may be lost or delayed frequently in the congested upward channel of the receiver. In these cases, TCP will unnecessarily initiate undesirable congestion control to reduce data rate when it detects the occurrences of packet loss. This problem will be illustrated in the rest of this section.

B. TCP Problems on Asymmetric Networks

The performance of TCP depends on an uncongested two-way communications, one channel for sending data packets, the other for sending ACKs back. To make TCP perform well, neither channel can be congested.

In an asymmetric network such as ADSL, the upward channel (from user's viewpoint) has a smaller bandwidth than the downward channel. Ideally, the downward channel must be able to carry a bigger downloading traffic up to its maximum bandwidth. In reality, its actual throughput may be influenced by the congestions occurred on the upward channel. The congestion occurred on the upward channel may block the delivery of ACKs and trigger the congestion control mechanism at the sender side unnecessarily.

Unfortunately, BT-compatible P2P file sharing over asymmetric networks is facing exactly such a problem. Most downloaders are sharing out their own fragments while downloading the needed fragments from other peer

nodes. The upward channels may be congested by the sharing out traffic and many ACKs for TCP will be held at the receiver side. Consequently, the corresponding data packets are considered lost when the timers at the sender side expire. Once the sender detects the occurrence of severe packet loss, its congestion control mechanism will automatically reduce the data rate to the minimum level accordingly. This is one of the major reasons why BT users are not satisfied with the downloading speed. In next section, some approaches will be discussed to solve this problem.

C. Approaches to Improve Transport Protocol

The first approach to overcome the problems mentioned above is to design a new TCP protocol for BitTorrent environment [4]. The second approach is to use UDP instead of TCP to transport data.

C.1 TCP Approaches

Some possible techniques are (1) use longer lost packet timers to accommodate delayed ACKs and thus to prevent the sender from triggering congestion control procedure; (2) improve the accuracy of bandwidth estimation to prevent senders from running too fast; (3) use more delicate information to detect congestion (not to take packet loss as the sole congestion signal). Further researches are needed to implement this approach [4].

C.2 UDP Approaches

The advantage of using UDP is obvious: the download throughput at the receiver side will not be affected by the congestion occurred on the upward channel. However, this approach has two problems yet to be solved: (1) data integrity assurance (i.e. to recover the lost packets) and (2) data rate determination (i.e. flow control and congestion control). Because the sender of a transport session has no knowledge about the end-to-end network bandwidth between the source and the destination, the determination of data rate becomes one of the trickiest tasks to a transport protocol. A number of TCP flavors were developed to cope with this problem. Most of them use a complicated mechanism to control the size of sending window, which has an equivalent effect of automatic data rate adjustment. On the other hand, the sender of a UDP session blindly transmits data at a fixed data rate, which is set by the application program without any adaptation to the network status. Overflowed packets are dropped somewhere in the congested network nodes.

III. DESIGN OF UDP BASED APPROACH

We solve the two problems mentioned in the previous section by embedding extra mechanisms at the application level. Data integrity assurance is done by a mechanism similar to TCP. That is, the receiver acknowledges for the received data and the sender retransmits the lost data. On the other hand, the data rate determination is done by decomposing a UDP session into a number of smaller UDP sessions, measuring the available network bandwidth by inserting probing packets before each session, and then transmitting the succeeding packets using the measured bandwidth. In a nutshell, we designed a special TCP protocol on the top of UDP dedicated for BT-like file sharing systems.

A. Granularity of Retransmitted Data Segment

A *retransmitted data segment* (*RDS*) is a group of data packets, which will be retransmitted entirely if any packet in the group is lost in transmission. The first granularity option is the packet level RDS. This option may not be practical because it requires a tedious packet management mechanism implemented at the application level. It may also incur a significant acknowledge overhead if real time acknowledgement is needed. The second option is the file (fragment) level RDS. If there is any packet lost, the entire file fragment must be retransmitted. The retransmission cost of this option is too high, but it doesn't require any packet management mechanism at the application level. To reduce retransmission overhead, we took the third option to design an *Incremental Retransmission* (IR) mechanism that doesn't require to retransmit the entire file fragment if a transmission session fails.

B. Incremental Retransmission

The incremental retransmission is done as follows:

1. If a RDS is found unrecoverable, the receiver sends a signal to the sender to request for retransmission. This signal also carries necessary information to indicate the missing RDSs.
2. Upon receiving a retransmission request, the sender terminates the current transmission, repackages the remaining file fragment, then reinitiates the transmission.

This retransmission mechanism is simple but may not be precise. In other words, it may retransmit more data than necessary. However, it is already able to reduce the retransmission overhead from the file level down to the RDS level. More delicate retransmission mechanisms such as those used in TCP are yet to be evaluated for a possible reuse.

C. Packet Level Error Correcting Mechanism

To reduce retransmission overhead, we also designed an error correcting mechanism that partitions the data stream into groups of size *n*, called *self-protect packet set* (SPPS), and embeds a *parity packet* into each group. As depicted in Fig. 1, the value of each bit in a parity packet is the parity value of the same bit position in the same group. (In fact, a SPPS is a good candidate for a RDS. Furthermore, we use *n* instead of *n+1* to denote the size of SPPS for simplicity).

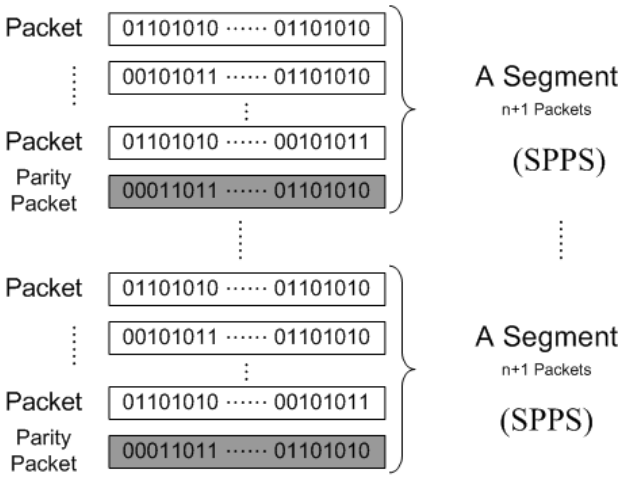


Fig. 1 Packet Level Error Correction

Assuming packets are numbered in sequence and the receiver keeps track of received packet sequence, the receiver can recover one lost packet for each SPPS. When more than one packet in a SPPS is lost, the entire file (or the rest of the file if IR is used) must be retransmitted. The size of SPPS has a significant impact to the retransmission efficiency. The higher the value, the smaller the number of parity packets, but higher the retransmission probability. The determination of SPPS size is dependent on the reliability of data transmission. This will be discussed in next section.

D. SPPS Size Determination

We assume to use file (fragment) level RDS for simplicity. A file fragment consists of *m* packets. Each segment has *n* packets. Thus each fragment has $\lceil m/n \rceil$ segments. Each of these segments is associated with a parity packet to form a SPPS. Assuming the probability that a packet may be lost in the transmission is γ, it is straightforward to calculate the probability that a file fragment must be retransmitted is

$$p = 1 - \left(bin(0 \mid n+1, \gamma) + bin(1 \mid n+1, \gamma) \right) \quad (1)$$

Assuming the granularity of RDS is one SPPS, the total extra overhead is then:

$$Penalty \quad (n \mid m, \gamma) = \left\lceil \frac{m}{n} \right\rceil \left[1 + \frac{(n+1)p}{(1-p)^2} \right] \quad (2)$$

The optimal SPPS size can then be easily obtained by taking a derivative on (2).

E. Adaptive UDP mechanism

To prevent a data sender from congesting the network, a transport protocol must choose an appropriate data rate to transmit data. As described in the previous section, TCP is able to adjust transmitting data rate by trail-and-error to match to the network bandwidth. However, UDP can't determine the appropriate data rate by itself. It depends on external instructions to set its data rate. An easy way to solve this problem is to send probing packets to the receiver periodically to measure the effective network bandwidth and then adjust the data rate accordingly. This technique is similar to TCP Vegas, which measures the round trip packet delay time and adjusts sending window size accordingly. In our algorithm, a fragment downloading session is decomposed into a number of smaller UDP sessions and then several probing packets are sent before each UDP session to measure the available network bandwidth. The succeeding UDP session then uses the measured bandwidth to transmit succeeding data. Receivers estimate the network bandwidth based on the measured dispersion of inter-packet elapse time [3], and then send the information back to the senders. The bandwidth estimation formula is as follows:

$$\text{bandwidth} = \alpha \text{ (packet size)/(avg. packet dispersion time)} \quad (3)$$

In (3), α is the *network bandwidth coefficient* representing individual network characteristics and can only be determined empirically.

IV. PERFORMANCE EVALUATION

In this section, the best size of SPPS is analyzed, the network bandwidth coefficient is experimentally studied, and our UDP-based BT protocol is evaluated against that with TCP-Reno and with TCP-Vegas using experimental approach.

A. Size of SPPS

In this analysis, we calculate the optimal size of SPPS given the parameters of fragment size and network error

rate. The fragment size is set at 250 packets and each packet consists of 1000 bytes. The packet error rate is set from 0.005 to 0.02. Assuming the entire fragment is retransmitted if a SPPS is found unrecoverable. Penalty is defined as the overhead for retransmission. The penalty as a function of SPPS size is shown in Fig. 2. The optimal size of SPPS as a function of packet loss rate is shown in Fig. 3. As we can see from Fig. 2 and 3, the optimal size of SPPS is 21 at 0.005 packet loss rate. From Fig. 3 we can see that the lower the packet loss rate, the higher the optimal size of SPPS.

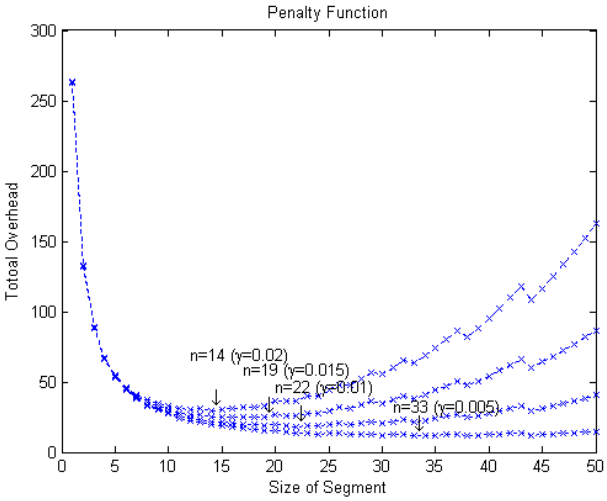


Fig. 2. Penalty Analysis

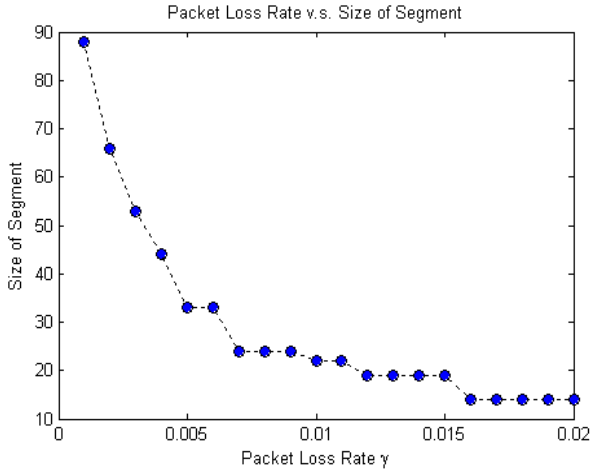


Fig. 3. SPPS size Analysis

B. Determination of Network Bandwidth Coefficient

To use (3) to probe the network bandwidth, we need to know the value of Network Bandwidth Coefficient (α). It is an empirical value depending on the characteristics of specific networks. We use the topology shown in Fig. 4 to investigate the appropriate value of α. Testing traffic is sent from X_1 to Y_1. Interference traffics are crossing the intermediary routers to simulate the change of available bandwidth.

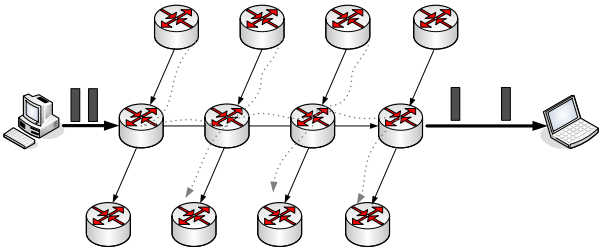


Fig. 4. Topology for Network Bandwidth Coefficient Experiments

The number of intermediary routers in five different topologies is 1,3,5,7, and 9 respectively. For each topology, the traffics are set from 1 to 10 Mbps. The value of α is calculated using (4).

$$\alpha = avg\left(\frac{Available\_Bandwidth}{Packet\_Size/Average\_Dispersion\_Time}\right) \quad (4)$$

The average α value among all five topologies is 0.929. The difference between the maximum and minimum is smaller than 0.1. Therefore, it is a good value for most networks.

C. Evaluation of UDP-Based Approach

This section evaluates the performance of BT with our UDP-based approach against BT with TCP Reno or with TCP Vegas using NS-2 network simulator. The simulated topology is shown in Fig. 5. The bandwidth at the core network is set at 1 Gbps such that the core network will never be congested. Six downloaders (Y_1, X_1, X_2, ..., X_5) are located at the both ends of a core network path. R_1 and R_2 are access routers. Y_1 downlaods a file (fragment) from every other downloader (session 1-5) and uploads a file to each of them (session 6-10). Each downloader accesses the core network using an asymmetric bidirectional link. Each access link has a 2-8 Mbps downward channel and a 56 kbps upward channel. Under this topology, the link from Y_1 to R_1 will most likely be congested by the upward streams and, as a consequence, the streams from Y_1 to X_1-X_5 (session 6-10) will be affected by the BoA problem. Parts of results are shown in Fig. 6.

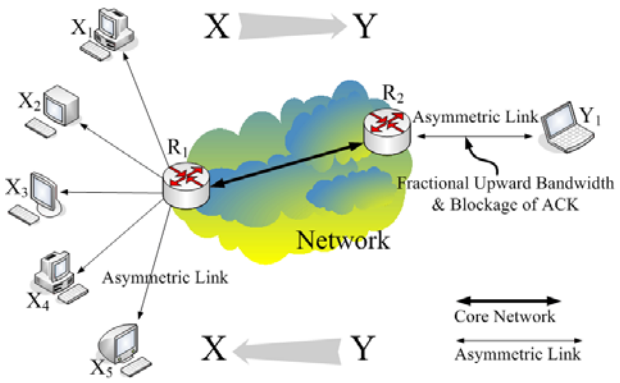
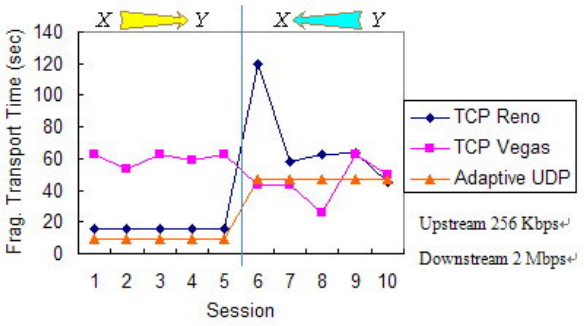
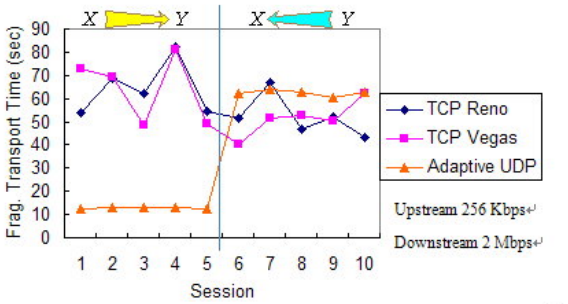


Fig. 5. Topology for Performance Evaluation



(a)



(b)

Fig. 6. Performance of BT with Three Protocols
(a) packet loss rate = 0.1%, (b) packet loss rate=10%

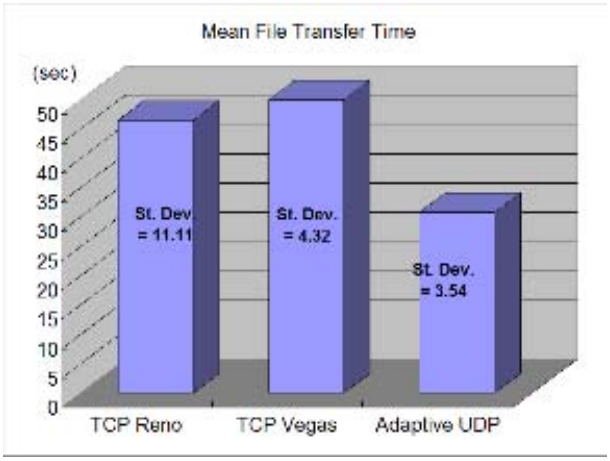


Fig. 7. Average Time to Transport 1/4 Mbytes of Data

The average elapse time to transfer a file of 1/4 M bytes by BT with TCP Reno, TCP Vegas, and our protocol are 46.5 sec., 50 sec. and 30.8 sec. respectively. From these experiments we can see that our algorithm is much better than the other two, especially when the packet loss rate is high. This result is consistent with our analysis. TCP Reno is a very aggressive transport protocol and is highly dependent on the acknowledge packets to "clock" its operation. The blockage of acknowledge packet will cause excessive number of time out and, as a consequence, reduce its sending window size and data rate. TCP Vegas is very conservative and may not be able to fully utilize the network bandwidth. Besides, our packet error correcting mechanism may make a significant contribution too.

V. CONCLUDING REMARKS

In this paper, we analyze the performance problems of BitTorrent based P2P file sharing operation model over asymmetric networks from two viewpoints: network topology and impairment of TCP protocol. Some shortcomings may affect system and network performance: data transmission paths are highly redundant wasting a lot of backbone bandwidth, download throughput is only a fraction of upward bandwidth; TCP performance is deteriorated due to the blocking of acknowledge packets on the upward channel. We propose a solution approach that use UDP for data transportation to alleviate BOA problem. The simulation results show that it can really improve the throughput. Since TCP protocol is a dominant transport protocol that are used by many network applications, the expected research results will be applicable to other P2P file sharing models as well as other network applications.

REFERENCES

1. http://bittorrent.com.
2. Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan and Randy H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", *Proc. ACM SIGCOMM'96*, Aug. 1996.
3. P. Heleher, B. Bhattacharjee, and B. Silaghi, "Are Vitrualized Overlay Networks Too Much of a Good Thing?", *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, MIT Faculty Club, Cambridge, MA, USA, Mar. 2002.
4. Asymmetric and Wireless Networks ", *Proceedings of The First International Workshop on Mobility in Peer-to-peer Systems (MPPS05)*, June. 6-9, 2005, pp. 850-855.
5. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-peer Networks", *Proceedings of the 16th Annual ACM International Conference on Supercomputing*, 2002.
6. R. Matei, A. Iamnitchi, and P. Foster, "Mapping the Gnutella Network", *IEEE Internet Computing*, Vol. 6, Issue 1, Jan.-Feb. 2002, pp. 50-57.
7. J. Postel, "Transmission Control Protocol", IETF RFC 793, Sep. 1981.
8. S. Saroiu, K. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-peer File Sharing Systems", *Proceedings of Multimedia Conferencing and Networking*, San Jose, Jan. 2002.
9. S. Saroiu, P. K. Gummadi, and S. D. Gribble, "Exploring the Design Space of Distributed Peer-to-peer Systems: Comparing the web, Triad and Chord/cfs", *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, MIT Faculty Club, Cambridge, MA, USA, Mar. 2002.
10. B. Y. Zhao, Ling Huang, J. Stribling, S. C. Rhea, A. D. Joseph, J. D. Kubiatowicz, "Tapestry: a Resilient Global-Scale Overlay for Service Deployment", *IEEE Journal on Selected Areas in Communications*, Vol. 22, Issue 1, Jan. 2004.