Performance Issues of P2P File Sharing Over Asymmetric and Wireless Networks

Yao-Nan Lien
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 affect system and network performance: data transmission paths are highly redundant wasting a lot of backbone bandwidth, the download throughput in a node may be limited by the the upward bandwidth of other 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. These problems are further complicated by the voluntary nature of P2P: peer nodes are fairly unstable. For similar reasons, when such P2P file sharing paradigm is moving to wireless networks, it may suffer from even worse performance degradation due to many reasons such as lower link reliability, lower bandwidth, and the impairment of TCP protocol.

This paper analyzes these problems mainly from topology and protocol viewpoints and proposes some solution approaches to alleviate some of these problems.

Key Words:

P2P and TCP.

1. Introduction

P2P file sharing is a very popular network application. There are many successful systems such as Napster, Kazza, Gnutella, Freenet, and BitTorrent [1,5,9]. By the degree of centralization, these systems can be roughly classified into Pure Decentralized, Partially Decentralized, and Hybrid Decentralized [14]. By the network structure, they can be classified into Unstructured, Structured, and Loosely Structured systems [8]. *BitTorrent*, 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 (*original seeder*) waiting for retrieving by some peer nodes. Each peer node may retrieve these file fragments either from the original seeded or from other peer nodes that have already retrieved some fragments. Each peer node may also serve as a redistribution node to share out the retrieved fragments.

In reality, many users are using various software tools that are compliant with BitTorrent protocol to cheaply spread files around the Internet. If a user wants to share a file to others, it will serve as the *original seeder* (or *original downloader*). A BitTorrent-compatible server software, called *tracker*, breaks the file into many smaller fragments, then publishes a small file called *torrent* to some web site. If a user wants to download a copy of the file, rather than downloading the actual file, instead the user would download the torrent onto his/her computer. When the torrent is opened by a BitTorrent-compatible client software, called *end user downloader*, the downloader searches for other downloaders that have downloaded the same torrent, and try 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 node combines all retrieved fragments back into a single file that is identical to the original file. As a common courtesy, a downloader needs to voluntarily stay online becoming a seeder to share out his/her downloaded copy to other downloaders for some certain length of time. What a tracker does is giving each request a fragment, then introducing all peer nodes 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 over each other severely. As a consequence, it may waste a lot of backbone bandwidth. Although structured network model can organize the participating peer nodes into a less redundant network topology and thus can relieve this problem, current solutions are all file based such that they 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 higher bandwidth. 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 such that the failure rates of download operations are fairly high.

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 [12]. 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. These shortcomings severely impair the performance of networks and the P2P file sharing itself.

When this kind of P2P file sharing model is applied to a wireless network, similar network performance problems may occur. It may suffer from even worse performance degradation due to the poor TCP performance over unreliable wireless links.

This paper is set to analyze these problems mainly from network topology and TCP protocol viewpoints as well as to propose some solution approaches to alleviate the problems.

2. Influence of Fragment Topology

For simplicity, we assume only a single file is to be shared. Assuming each peer user is accessing a fragment either from the original downloader or from another end user downloader, the download-upload relationship among all peer nodes forms a *fragment tree*. All fragment trees share the same root node, which is the original downloader. It is sufficient to analyze the performance of BitTorrent operation model based on the topology of a single fragment tree. Thus, our analysis is based on a single fragment tree. Note that it is not necessary for all downloaders of the same fragment tree to be alive simultaneously. As long as there is at least one downloader that has the complete fragment and is willing to share it out, the fragment itself is available for retrieving.

2.1 Long Physical Paths

Each link in a fragment tree, named *f-link* for simplicity, is really a path of any length on the Internet. Unfortunately, BitTorrent operation model does not force downloaders to take path length into account such that the physical topology of a fragment tree may contain many redundant path segments. As a result, P2P file sharing unnecessarily generates too much Internet traffic, and together they devour one third of backbone bandwidth today.

Using structured-network approach to construct an overlay network and then having all peer nodes to download the desired files from designated neighboring nodes seems an attractive solution [6,15]. However, since peer nodes in BitTorrent paradigm join and leave the file sharing network arbitrarily, and the locations of file fragments are hectically determined in real time, it is impractical to use such a pre-planning approach.

Nevertheless, a straightforward solution is to have every downloader select the "nearest neighbor" to download the fragment [11]. To identify the "nearest neighbor" of a node, we need to estimate the physical distances between each other based on some measurement, such as throughput or packet transfer latency.

2.2 Width of Fragment Trees

Bushy Tree

The average width of a fragment tree may have a significant impact on the performance. A bushy tree may cause a downloader uploading too many file sharing streams to other downloaders and suffering from severe FUB and BoA problems.

Slim Tree

In a slim fragment tree, each downloader needs to offer fewer uploading streams for others to download. Thus, in a peer node, each uploading stream may be allocated with a larger share of the upward bandwidth. Furthermore, the upward channel may be less congested. The FUB and BoA problems associated with a bushy fragment tree can be alleviated.

On the other hand, a slim fragment tree may cause some problems too. First, a newly joined downloader may be forced to retrieve the fragment from a remote downloader rather than a neighboring downloader. As a result, the average length of access paths may be longer. Furthermore, it would take much more time to search a downloader that has the desired fragment and has available "quota" for file sharing.

The two influence factors associated with the width of the fragment tree seems contradict to each other. Good P2P file sharing program designers must strive for the balance between the two factors. It is an interesting research issue to find out the balance points under various conditions and objectives.

2.3. Distributed Minimum Spanning Tree for Fragment Tree

When the backbone bandwidth is a precious resource such as that in an Ad Hoc Wireless LAN, it is necessary to reduce (or minimize) the total physical length of f-links.

To reduce the length of a f-link, as mentioned earlier, a downloader can estimate the physical distances to all other downloaders based on some measurement, and then select the best one. This greedy solution may not be the best solution. Furthermore, when some downloaders are out of service for any reason, a reconfiguration procedure needs to be initiated to reconstruct the fragment tree. The configuration of the fragment tree can be modeled as a variation of the conventional *Distributed Minimum Spanning Tree* with a constraint on the maximum number of adjacent nodes [7]. Finding a good solution is an interesting research topic.

3. Influence of TCP Protocol

A network application that demands a reliable data transfer would probably choose TCP to "transport" data [12]. This section will discuss the performance problem caused by the impairment of TCP.

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. The sender breaks the file or the message that is to be sent into packets and transmits them in sequence to the receiver. The sender will keep track of packet delivery and retransmit the packets that are lost. When the receiver receives some packets successfully, it sends acknowledge packets back to the sender. If the sender doesn't receive the acknowledge packet within some certain time limit, the packets corresponding to the missing acknowledge is considered lost and will be retransmitted by the sender.

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 transmitting data rate in a trial-and-error fashion. To prevent the network from overly congested, TCP takes a packet loss as a signal 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 by trial-and-error is not very efficient. Many improvement mechanisms have been proposed and implemented to enhance TCP performance under various conditions [2,3,10].

Current TCP is designed to take packet loss as a signal of network congestion. It works well for regular networks. However, it may not work well in other network environments, such as unreliable wireless networks or BitTorrent over asymmetric networks, where acknowledge packets may be lost or delayed due to some causes other than network congestion. In these cases, TCP will unnecessarily initiate undesired congestion control to reduce data rate when it detects the occurrence of packet loss. This problem will be illustrated in the rest of this section.

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 acknowledge packets back. To make TCP perform well, neither channel can be congested.

In an asymmetric network such as ADSL, one of the two-way channels has smaller bandwidth than the other. Theoretically, the channel that has larger bandwidth must be able to carry a bigger traffic flow up to its maximum bandwidth. In reality, its actual throughput may be limited by the congestion occurred on the other channel. The congestion occurred on the other channel may block the delivery of acknowledge packets and trigger the congestion control mechanism at the sender side unnecessarily.

Unfortunately, BitTorrent compatible P2P file sharing over asymmetric networks is facing exactly such a problem. Most downloaders are sharing out their own fragments while downloading fragments from others. The upward channels may be congested by the sharing out traffic. Many acknowledge packets will be held at the receiver side and be treated as lost packets when the timers at the sender side are expired. Once the sender detects the occurrence of severe packet loss, its congestion control mechanism will automatically reduce the transmission rate to a minimum level accordingly. In Section 4, some approaches will be discussed to solve this problem.

TCP Problems on Unreliable Networks

In some environments, such as an unreliable wireless network, many packets may be lost due to high noise on the communication channels. Similar to the problem associated with asymmetric networks, the sender of a TCP connection will trigger a congestion control mechanism to reduce the transmission data rate although the causes of these two situations are different.

Many researches are trying to solve this problem [2,15]. Most of them use some kind of proxy mechanism to buffer packets for receivers that are attached to the network through a noisy channel. The TCP connection looks to the sender like a reliable network. Therefore, unnecessary congestion control will not be triggered. In reality, implementing these mechanisms may not be practical because they require assistant and support from proxy nodes. The simplicity of original TCP will be destroyed.

When BitTorrent P2P file sharing is moving to a wireless network, upward channels are both congested and noisy, the TCP performance will severely deteriorate. It is not easy to solve this problem though.

4. Approaches to Improve Transport Protocol

There may be some approaches to overcome the problems mentioned in the previous section. The first approach is to use UDP instead of TCP to transport data. The second approach is to modify TCP to accommodate to these special situations. They will be discussed in this section.

4.1. 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, since UDP has neither the capability to recover lost packets nor the capability to determine the appropriate transmitting data rate, some enhancements are needed to overcome these problems.

Lost Packet Recovery

Lost packet recovery mechanism, also referred to as *data recovery* for generality, can be implemented at the application level. Data granularity can be set either at fragment level or at packet level. Fragment level data recovery is to throw the entire fragment away when any packet is found lost and then to ask the same sender of a different peer node to retransmit the desired fragment. This method is easy to implement but may waste too much bandwidth. Thus, it is only applicable when either the network is reliable or fragments are small.

On the other hand, packet level data recovery is more efficient but more tedious to implement. First, receivers must keep track of packet loss. Secondly, senders must be able to break the fragment into packets at the application level and repack the packets that are to be retransmitted into a new message for retransmission. Moreover, this approach will violate the layer structure of network protocols. In reality, application level software doesn't know the packetization details at the transport layer such that it is not easy to extract the desired packets out of the original fragment. Some virtual packetization mechanism will have to be built into the application itself. Further researches are needed for both approaches.

Data Rate Determination

To prevent the sender from congesting the network, 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. In many real world applications, users have to choose an appropriate data rate explicitly. One simple way to enhance UDP is to send probing packets to the receiver periodically to measure the effective network bandwidth and then adjust the data rate accordingly. Researches are undergoing to design appropriate probing procedures.

4.2. TCP Approaches

Since UDP doesn't retransmit lost packets, it is obviously not applicable for file sharing in an unreliable wireless network. Thus, TCP is more appropriate in such situations. There are some approaches to modify TCP to accommodate to such special network environments. However, they are not designed for BitTorrent paradigm though. In other words, a new TCP protocol specially designed for BitTorrent environment may be able to achieve a better performance.

Longer Lost Packet Timer

For asymmetric networks where acknowledge packets are held at the receiver side because of congestion occurred on the upward channel, a simple technique is to set a larger waiting time for acknowledgement. Since many acknowledge packets are not lost but delayed, a larger waiting time would be able to prevent sender from triggering congestion control procedure. Nevertheless, this simple technique only works on regular asymmetric networks (e.g. wired ADSL), but not for unreliable networks where packet loss rates are high.

Estimate Data Rate

One possible source of inefficiency is several versions of TCP protocol is that they take a trial-and-error fashion to determine the appropriate data rate and to perform congestion control. Unfortunately, this trial-and-error approach counts on the occurrence of congestion to adjust its data date. One possible way to improve the efficiency of TCP is to estimate the effective network bandwidth first, then to determine the appropriate data rate accordingly. Another way is to use different indicators that can detects potential congestions before they actually occurs. Some researches are undergoing in our research team.

Improve Congestion Control Policy

Congestion control mechanism, which includes triggering condition and rate control policy, must be improved too. Although packet loss is still an important indicator of network congestion, it shouldn't be the only indicator in many special network environments. Furthermore, AIMD may not be the best rate control policy any more. A good TCP protocol suite must have good solutions for both issues. Further researches are needed to obtain concrete results.

5. Summary

In this paper, we analyze the performance problems of BitTorrent based P2P file sharing operation models over asymmetric networks and wireless 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 also propose some solution approaches that can alleviate these problems. Many issues are yet to be researched. Since TCP protocol is a dominant transport protocol that are used my 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. L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1465-1480, 1995.

4. I. Clarke, T. W. Hong, O. Sanberg, and B. Wiley, "Protecting Free Expression Online with Freenet", *IEEE Internet Computing*, Vol. 6, No. 1, Jan.-Feb. 2002, pp. 40-49.

5. I. Clarke, O. Sandberg, and B. Wiley, "Freenet: A Distributed Anonymous Information Storage and Retrieval System", *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, California, June 2000.

6. 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.

7. Yao-Nan Lien, "A New Node-Join-Tree Distributed Algorithm for Minimum Weight Spanning Trees", *Proc. of Eighth IEEE International Conference on Distributed Computing Systems*, June, 1988, pp. 334-340.

8. 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.

9. R. Matei, A. Iamnitchi, and P. Foster, "Mapping the Gnutella Network", *IEEE Internet Computing*, Vol. 6, Issue 1, Jan.-Feb. 2002, pp. 50-57.

10. J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas", *Proc. of IEEE INFOCOM*, pp. 1556-1563, Mar. 1999.

11. C. G. Plaxton, R. Rajaraman, and A. H. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment", *Proceedings of ACM SPAA*, ACM, June 1997.

12. J. Postel, "Transmission Control Protocol", IETF RFC 793, Sep. 1981.

13. 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.

14. 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.

15. G. Xylomenos, G.C. Polyzos, P. Mahonen, M. Saaranen, "TCP Performance Issues over Wireless Links", *IEEE Communications*, Apr. 2001.

16. 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.