Reducing ISP Cost by Caching of P2P Traffic

Course Project for EE695B
Sabyasachi Roy and Zheng Zhang
Instructor: Prof. Sanjay Rao
Purdue University
West Lafayette, IN 47907
{roy0, zhang97}@purdue.edu

Abstract

The advent of peer-to-peer applications has led to a significant growth of network traffic and thereby a significant concern among ISPs about the growing operating costs. However, the traffic generated by peer-to-peer applications such as BitTorrent has redundancy, which can be exploited by caching to reduce the network traffic. In this project, we focused on BitTorrent and proposed to reduce the ISP's cost by caching BitTorrent traffic on the ISP's gateway. Because the cache has limited storage capacity, the cached data should be replaced when the cache fills up. The local-rarest-first block selection strategy employed by BitTorrent clients has impact on the choice of the replacement policy. In this project, we studied by simulation the performance of three proposed replacement policies as well as two classical replacement policies. Our simulation results showed that our coarse rank-based replacement policy performs best among five policies under typical network topology and BitTorrent user behavior pattern.

1. Introduction

The advent of peer-to-peer applications has led to a significant growth of network traffic and thereby a significant concern among ISPs about the growing operating costs. For example, it has been reported recently [3, 4] that BitTorrent [5] constitutes 20% of the traffic that goes through the Internet and 60% of the traffic seen by the ISPs. However, the traffic generated by BitTorrent, as seen by the gateway of the ISP, has redundancy [6], which implies the potential of mitigating the costs incurred. The cause for such redundancy is that BitTorrent clients choose their neighbors randomly, disregarding the network cost of the transfer paths. The consequence is that multiple copies of the same data may traverse the same physical link and high cost is incurred at ISP's side. In this project, we aim to reduce the redundancy of BitTorrent traffic and thus reduce the ISP cost.

There have been many solutions proposed to the problem of reducing ISP costs. Such solutions mostly focus on approaches such as bandwidth limiting, biased neighbor selection [2], gateway peers [6] and caching [4]. Each solution has its pros and cons. Bandwidth limiting reduces ISP costs at the expense of the rate of content transfer. Biased neighbor selection schemes cluster peers in a way that the neighbors are within the same ISP. Although this can reduce ISP costs in a transparent manner, it is not clear if the availability of content and rate of content transfer will be affected or not. Gateway peers have the potential to reduce the ISP costs significantly but they can be a source of bottleneck and hence not scalable. Moreover there might be thousands of files being shared and it is not practical for the gateway peer to store and serve each and every file. The solution that we espouse is based on caching of content at the ISP's gateway. Such a solution has the efficiency of a gateway-peer based solution without being a source of bottleneck.

In this project, we propose to reduce the ISP's cost by caching BitTorrent traffic on the ISP's gateway. The reason for an ISP to deploy cache is that caching has help the ISP to reduce the volume of traffic generated by BitTorrent and thus reduce its own operating cost. The high-level view of the cache is as follows. The cache the conversation between each BitTorrent client inside the ISP and each BitTorrent client outside the ISP, but appears transparently to the clients on both ends. It caches the data that arrives from outside the ISP. Later requests from inside the ISP for the same data would be served by the cache directly. This serves two purposes. Firstly, the same requests by different users within the ISP would not need to go outside the ISP, which creates

cost savings for the ISP. Secondly, BitTorrent performance is limited by the uploading bandwidth [1] of the client access link. The fact that a cache node has uploading bandwidth much more than that of a common peer, can shift the bottleneck to the downloading bandwidth of the clients, which is usually larger (around 5 times) than the their uploading bandwidth. We seek to leverage this observation in order to improve user-perceived performance besides reducing cost.

Because the cache has limited storage capacity, replacement policy comes into play when the cache fills up. As in BitTorrent the request for data is made on file block basis and the cached data also consists of file blocks instead of entire files, the replacement policy is more flexible if the decisions are made on block granularity than file granularity. The local-rarest-first strategy, employed by BitTorrent clients in deciding the file block to request from other peers, results in a different request patterns of BitTorrent traffic from that of traditional web traffic. In this project, this observation motivated us to mainly focus on the study of its impact on the choice of replacement policies. We examined this impact by evaluating three proposed replacement policies as well as two classical replacement policies, LRU and random.

The three proposed replacement policies, namely *Coarse Rank-Based* (CRB), *Fine Rank-Based* (FRB) and *Most Replication Based* (MRB), use the number of replicas of a block to estimate the probability of this file block being requested. The coarse rank-based policy is broadly based on the fact that the blocks requested by clients are governed by the number of replicas of the blocks in the local perspective of a BitTorrent client. Fine rank-based policy is similar to coarse rank-based policy, but it makes replacement decisions based on a finer ranking metric. Most replication based policy is in essence opposite to CRB but still reasonable.

The replacement policies were evaluated by simulation. Our discrete-time simulator captures most of the BitTorrent client functionalities such as tit-for-tat choking/unchoking, optimistic unchoking, local-rarest-first block selection, random neighboring, and anti-snub, while abstract away some details irrelevant to our study. The network topology used in the evaluation consists of 140 nodes in 14 ISPs, with the bandwidth of the nodes generated from a realistic distribution. Our simulation results showed that our coarse rank-based replacement policy performs best among the five evaluated policies.

The rest of this report is organized as follows. Section 2 presents assumptions we made and an overview of the cache. Section 3 discusses the replacement policy. Section 4 describes the simulation methodology. Section 5 presents the simulation results. Finally, Section 6 discusses the challenges and future directions.

2. Overview of ISP caching

The cache intercepts the conversation between each BitTorrent client inside the ISP and each BitTorrent client outside the ISP, but appears transparently to the clients on both ends. We assume all the traffic between inside the ISP and outside the ISP goes through a gateway, where the cache is deployed. The gateway monitors all outgoing traffic and incoming traffic, and can identify BitTorrent traffic as follows. When a BitTorrent client starts downloading, it contacts the tracker to bootstrap. This is a http request and can be identified by examing the message header. The ip address and port number the BitTorrent client is listening at are also contained in the request and thus can be obtained. The gateway then sets up state for this ip address and port number. Later, the BitTorrent client will use this port to establish TCP connections with each of its neighbors to exchange data blocks. The gateway once sees TCP connection handshakes destinated to or originated from this port, would intercept the TCP connection, and establish another TCP connection from itself to the destination but spoof the ip address and port of the source. Any data sent through the intercepted TCP connection is monitered by the gateway, manipulated if necessary and then forwarded by the gateway to the destination. In this way, the gateway is able to cache the data without breaking the transparency on the clients on both ends. The BitTorrent data that arrives from outside the ISP is cached. Later requests from inside the ISP for the same data would not be forwarded by the gateway to the destination outside ISP. Instead the ISP would be served the request directly from the cache, and pretend to be the destination peer.

In this project, our focus is the replacement policy of the cache. An implementation of the cache described above is not the scope of this project, but it is feasible.

3. Replacement Policies

In this section we present the several replacement policies that we studied. Some of them are well-known replacement policies. Besides, we propose three new ones and also explain the intuition behind such policies.

3.1. Well-Known Replacement Policies

Below we describe the already existing replacement policies that have been using in various fields of com-

puter science.

Belady's replacement policy Belady's replacement algorithm is an optimal replacement algorithm given the knowledge of future. It serves as a benchmark for all other replacement policies. The difference between the Belady's algorithm and any other replacement algorithm defines the scope of improvement that is theoretically possible. For determining the performance of Belady's algorithm we filter out the information about accesses made by the different BitTorrent clients. We run the Belady's algorithm on these accesses offline to estimate the upper limit of cache performance. Note that it is only an estimate because the access pattern itself is affected by the cache replacement policy.

Least Recently Used (LRU) replacement policy LRU is arguably the most common replacement policy, used under different contexts. LRU essentially keeps the most recently obtained/used block of data at the expense of the data block that has not been used for the longest amount of time. Temporal locality of data accesses is the basic premise on which LRU is based. Such a premise is particularly true in hardware caches and buffer caches. Hence, LRU performs remarkably well under such scenarios. However, it is not straightforward to think of any reason why LRU should perform well under BitTorrent based system. One possible reason for LRU to work well can be the following. BitTorrent clients request blocks that they deem as the rarest in the network. In such a case, many clients can request the same block that is the rarest until it becomes relatively popular. However, this can happen under two cases. First, there are blocks that are much less available than other blocks, i.e., the block distribution is skewed to begin with. This might not be true as this the scenario that BitTorrent tries to avoid by its rarest-first strategy. Second, all the clients have a global knowledge of availability of the blocks because local knowledge may lead to totally different blocks being determined as rarest. Our results show that under certain circumstances, LRU does perform very well for caching of BitTorrent blocks. We will explain the details in Section 5.

Most Recently Used (MRU) replacement policy MRU is a replacement policy that performs particularly well in case the access pattern of the blocks is looping. This essentially means that if the time-gap between two accesses to the same block is constant, MRU can perform well. Since BitTorrent tries to keep all the blocks equally available in the system, there is a fair chance that all the blocks will be accessed in a way such that the time-gap between two accesses to the same blocks is relatively constant. This argument follows from symmetry.

Random replacement policy Random replacement policy can be used as a benchmark. Any sophisticated replacement policy should be compared against the random policy to gauge its effectiveness. If there is not much difference between random replacement policy and some other replacement policy, it implies that it is not worthwhile to pursue a more fancy replacement policy. Moreover, if there is not a significant difference between the performance of Belady's and random replacement policies, it means that there is not much scope of improvement.

3.2. New Replacement Policies

Below we propose new replacement policies specific to BitTorrent caching and the rationale behind their design.

Coarse Rank-Based replacement policy (CRB) The Coarse Rank-Based policy is broadly based on the fact that the blocks requested by clients are governed by the number of replicas of the blocks in the local perspective of a BitTorrent client. Lower the number of replicas of a block, higher is the chance that the block will be requested. In our implementation, we calculate the ranks of each block in the cache in the following manner. For each block we calculate the number of replicas that are present in the neighborhood of a node within an ISP. Then we take the minimum of all the values over all the nodes within the ISP, which gives us the rank of that particular block. The block with the highest rank is evicted. The reason we take the minimum over all nodes is as follows. The caching is done at the gateway of an ISP, all the requests made by the nodes within a gateway are intercepted by the cache and these requests determine the contents of the cache at any time. So we are concerned with keeping the blocks that are most likely to be requested again in the recent future by *any* of the nodes within an ISP. Hence, we take the minimum of the ranks over all the nodes. The results are shown in Section 5.

Fine Rank-Based replacement policy (FRB) The CRB might not be accurate because the rank information is based on the local knowledge of a node within its neighborhood. The limitation of local knowledge cannot be gotten rid of because of the way BitTorrent is designed. But there is an additional source of

inaccuracy, i.e., the rank information is calculated per block only. In reality, the probability that a block is accessed by a node A in the future depends on the rank of a particular block present (in terms of the number of replicas present) with one of the neighbors B who has unchoked A. Thus to make the replacement policies more fine grained, we propose the Fine Rank-Based replacement policy. In this policy, we calculate the ranks of all the blocks that are not present in a node A and present in node B, where B is a node that has currently unchoked A. We order all such blocks based on this rank. There there is a separate ordering for each node pair of the form (A, B). Such an ordering gives another level of ranking among the different blocks. We take the minimum of such a rank for a block over all nodes like B. Then we take the minimum of the values obtained by the previous operation over all nodes like A that are within the same ISP. This value gives us the final rank of a block. The block with the maximum rank is evicted. Note that in practice, the FRB is not necessarily better than the CRB scheme because of two reasons. First, the former considers only the nodes that have unchoked another node. However, such a set of nodes keeps changing and cannot be predicted. So the decisions made in the past may no longer be valid in the future. Second, it is also computationally very costly, as now information has to be kept for each node pair and data block. As we observed in our simulations, the computational cost is way too high to have any practical value.

Most Replication Based replacement policy (MRB) The most replication based replacement policy is in essence exactly opposite to the CRB replacement policy. The reason why such a policy may work well is as follows. The CRB policy is inaccurate because it does not have knowledge about the nodes that are going to unchoke it in the future. Since the blocks that are accessed by a node depends on nodes who unchoke it and the blocks present with them, it is also unlikely that a block that is very rare in a neighborhood is present a node that is going to unchoke (or has already unchoked) in the most recent future. However if the number of replicas is fairly large in a neighborhood, it is likely that most of the nodes that are going to unchoke another node in the future have that block and hence the chances that such a block will be re-requested can be fairly high.

4. Simulation Methodology

In this section, we present the simulation methodology. Testbed or network emulation tools like Modelnet [8] have a large demand of infrastructure if the experiment involves hundreds of nodes. Because our evaluation involves a large population of BitTorrent peers, we decided to build a discrete-time simulator. Our simulator captures most of the BitTorrent client functionalities such as tit-for-tat choking/unchoking, optimistic unchoking, local-rarest-first block selection, random neighboring, and anti-snub, while abstract away some details irrelevant to our study. For the purpose of verifying the simulator, we also repeated the similar experiments and reached the same conclusions that were established in previous BitTorrent studies.

4.1. Simulator Design

In designing the simulator, we attempted to implement functionalities as faithfully to the BitTorrent specification as possible in order to perform realistic simulations, but in the meanwhile, we abstracted away some irrelevant details to reduce the engineering complexity and simulation runtime cost. The code is about 4000 lines in C++ language. In the following, we present our design decisions.

The simulator has two components, a BitTorrent peer component and a network component. The simulation is driven by events generated by BitTorrent peer component. There are two types of events, control events and data transfer events. Control events are generated by one peer to pass control information to another peer. Examples are "have" event, "timer" event, "choke" event and "unchoke" event. These events incur zero delay, i.e. the delay of virtual time is zero from the creation of the event to the handling of the event. This is because we assume in reality the packets related to these events are small in size and thus the transferring delay is small. Data transfer events are also generated by one peer, but their purpose is to transfer file block to another peer. The creation of a data transfer event indicates the beginning of a data transfer, and the handling of the event indicates the completion of the data transfer. In contrast to control events, data transfer events incurs non-zero delay, because we assume the data of a file block is relatively large in size, compared to the data for control events. The delay is dictated by the network component which models a network. The network component assumes a simple network model and calculates the throughput of a data transfer as the bandwidth divided by the number of connections. The delay of a data transfer event is thereafter determined by the throughput the transfer gets. In modeling the network, we did not use ns2 as network simulator because modeling packet-level transfer is time-consuming and redundant in our

Parameters	Values
Number of blocks of the torrent	1000
Block size	256KB
Number of neighbors per peer	10
Maximum number of unchokes per peer	1
Rechoke timer interval	10s
Optimistic unchoke timer interval	30s

Table 1. The various BitTorrent parameters used in the simulations

Fraction of nodes	Up. BW. (KBps)	Dn. BW. (KBps)
0.20	16	98
0.40	48	125
0.25	125	375
0.15	625	1250

Table 2. Upload and download bandwidth distribution.

experiment.

The BitTorrent peer component is responsible for handling various control events and data transfer events. A particular design decision to speed up the simulation is that all peers share a global "timer" event, which periodically arrives to trigger the rechoking and optimistic unchoking. When this event arrives, all peers simultaneously perform rechoking. In reality, every peer has a rechoke timer and rechoking takes places asynchronously. This simplification reduces the events to be processed, without altering the functionalities of BitTorrent.

4.2. Verification of the Simulator

For the purpose of verifying the simulator, we repeated the similar experiments and reached the same conclusions that were established in previous BitTorrent studies. In the following, we present the results.

Table 1 lists the BitTorrent parameters used in the simulation. The number of neighbors, number of unchokes, rechoke timer interval, and optimistic unchoke timer interval parameters are set according to the official BitTorrent client. The number of blocks and block size parameters are the settings of the typical torrent. The peer bandwidth is drawn from a distribution shown in Table 2. The seed typically has high bandwidth and hence its bandwidth is drawn from the last row.

Constant arrival In the first experiment, a stream of leechers arrives with a predetermined time interval and leave the system upon download completion. Figure 1(A) shows how the number of peers in the system

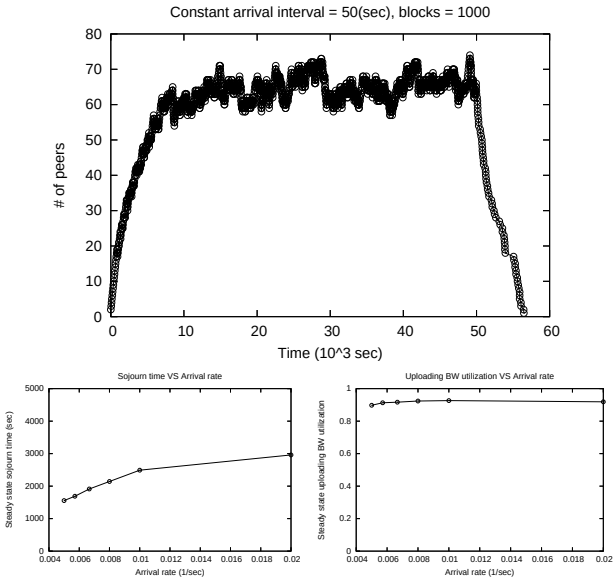

Figure 1. (A). The number of peers in the system reaches a steady state. Arrival interval = 50 second; (B). The average sojourn time in steady state under various arrival intervals; (C). The average uploading bandwidth utilization in steady state under various arrival intervals

evolves over time. Although initially the system is overloaded and the number of peers increases dramatically, at some point, this number no longer increases and the system reach a steady state. This means the BitTorrent system has the ability to adjust itself to prevent from crash. However, only the existence of steady state does not give us any hint on performance. So we calculate the average sojourn times of the peers in the steady state for various arrival intervals 50, 100, 125, 150, 175 and 200 seconds and show them in Figure 1(B). As we can see in the figure, the average time that a peer has to spend in the system does not degrade much as the arrival rate increases. This observation means BitTorrent scales well to the arrival rate of the peers. Besides scalability, we wonder if BitTorrent is efficient in using the bandwidth. So we calculate the average uploading bandwidth utilization during the steady state and show them in Figure 1(C). As we can see, the utilization is around 90% in all circumstances. This means that BitTorrent keeps all the peers busying uploading and is almost optimal in utilizing uploading bandwidth.

Flash crowd arrival In the previous experiment, we concluded that BitTorrent has good performance in steady state no matter how much the peer arrival rate is. The case of flash crowd is more demanding than

Parameters	Values
Cache size	100 blocks
Number of seeds	1
Number of leechers	139
Number of ISPs	14

Table 3. The network and cache parameters used in the simulations

Policy	Hit Rates (%) under various scenarios		
	Homo.	Heter.	Homo. Low BW.
Random	14.8	13.6	24.8
LRU	22.3	36.9	72.2
CRB	41.7	52.3	71.8
MRB	24.2	23.2	71.8

Table 4. The average hit rates of different cache policies under various scenarios over all the caches.

constant arrival because the system does not have time to react by absorbing enough peers into the system in steady state to match the capacity with the load. We also studied the performance of BitTorrent under flash crowd arrival. Due to space limit, the results are not shown here. What we found is that while the uploading bandwidth utilization is still high, the average downloading time is not as good as the case of steady state in constant interval arrival. However, it still increases sub-linearly as the crowd size increases.

The observations we made in the two experiments are consistent with the conclusion in [1] and [7]. This adds to our confidence in the simulator.

5. Results

In this section we present the results for the different cache policies in terms of average hit rate. We also explain why certain policies behave in a particular way they do.

5.1. Simulation Settings

Table 3 shows the network and cache settings that we used for our simulations, in addition to the settings of BitTorrent listed in Table 1. We followed the scenarios in [2] for our simulations. However, we scaled it down to reduce the running time. We scale down the various parameters proportionally. For example, [2] has 700 nodes in total whereas we have only 140. Thus we scale down by a factor of 5. Accordingly, we scale down the rest of the parameters such as the number of unchokes, the number of neighbors etc. also by a factor of 5. We believe that scaling down should not change the qualitative behavior of the system. We use a flash-crowd model as suggested by [1].

We conduct two types of simulations. The first type of simulations contain nodes with homogeneous upload and download bandwidths. Such simulations are relatively unrealistic, nevertheless they help us isolate the effects of heterogeneity of the network on the cache performance and understand the behavior of cache replacement policies better. The second type of simulations consist of nodes with heterogeneous upload and download bandwidths, which is closer to reality. We use a bandwidth distribution as suggested in [1]. Table 2 gives the distribution of the upload and download bandwidths under a heterogeneous case. For a homogeneous case we use 100KBps and 500KBps as the upload and download bandwidths, respectively.

5.2. Ideal Caching

In this section we assume ideal caching. By ideal caching we mean that the caching happens at the block level and so does the transfer of data among peers. Consequently, there is no difference between the unit of transfer and the unit of caching. In reality this is not the case, as will become clear in Section 5.2. However, we do consider ideal caching in order to estimate the potential of the different caching scheme and to study their behavior without the influence of other factors.

Homogeneous bandwidth First, we explain the results obtained under the homogeneous case. The second column of Table 4 gives the hit rates for the LRU, CRB and MRB policies. We also conducted experiments with the rest of the policies such as MRU, FRB etc., but we do not show all the results because they did not give reasonable performance. Moreover, FRB was computationally infeasible for even a small number of nodes rendering it impractical. The second column of Table 4, shows that CRB gives the best hit rates followed by LRU and MRB. There are two important things to note from such an outcome. First, LRU scheme does not perform very well, which indicates that probably there is not much temporal locality that can be exploited. The reason for pointing this out will be clear later in this section. Second, CRB which is purely rank based proves to be much better than and MRB which tries to improve chances that a block that is in the cache will be found in a node from which an unchoke is sent. This implies that the rarest-first block selection is the single most important factor that affects the efficiency of the cache replacement policy.

Heterogeneous bandwidth The third column of Table 4 shows the hit rates for the different replacement

policies under a heterogeneous environment. The relative performances of all the replacement algorithms remain the same. However, the interesting thing to note here is all the replacement policies perform much better than under homogeneous environment. Below we describe the reasons for such a finding. Note that under the homogeneous case, on average, the upload and download bandwidths are higher that under the heterogeneous case. This observation is related to the differences observed in the cache performance. As the upload/download bandwidth goes up, the time taken to download a block reduces. Hence the distribution of the different blocks in the system changes faster. In other words, the dynamics of the blocks distribution higher. Since MRB and CRB are both based on distribution of the blocks in the system, the decisions made by such policies at the current time are less likely to be true in the future. In effect, higher dynamics in the system means more mis-predictions by such replacement policies and hence degraded performance. To confirm this, we run all the simulations under a homogeneous case with much lower upload and download bandwidths (5 times lower). The fourth column of Table 4 shows the hits rates. As can be observed, the hits rates go up significantly.

The disparity in the results under the heterogeneous and homogeneous case can be explained on the basis of the theory of dynamics of block distribution for CRB and MRB. However, LRU is not dependent on the block distribution then it should be immune to such parameter changes, but it is not. This suggests that there are some additional factors and parameters that affect the temporal locality of the access pattern. It turns out that for LRU the dynamics of the block distribution affects its performance in the following way. What LRU does is to retain a block most recently accessed by a node within an ISP at the expense of another block that was not accessed for the longest time. Usually, when a node completes the download of a block, it notifies all its neighbors about that block. As a results, all its neighbors that do not have that block express interest in that block. Depending on the unchoking policy and the limit on the number of simultaneous unchokes that can be given, a subset of the neighbors are allowed to download the block. Under the case where the block distribution is relatively static, the time between the event a block is downloaded by a node A within an ISP I, and the event that it is again requested by another node B within the same ISP I, is relatively small as compared to the timescales at which the block distribution changes in the network. Thus, there is a high chance of the same block being requested more than once by two nodes within the same ISP. Note that other factors such as the number of unchokes, decreasing the rechoke period etc., also determine the effectiveness of LRU. For example, increasing the number of unchokes makes it more likely to get an unchoke from a node, which in turn makes it more likely that two nodes within the same ISP will get unchokes from the same node within a short period of time, which makes it more likely that two such nodes will ask for the same block within that period of time, eventually leading to a cache hit. Similarly, decreasing the rechoke period also improves the chances that two nodes will get an unchoke from the same node within small period of time thus increasing the hit rate.

In summary, the performance of the different cache replacement policies depends on scenarios under which they are operating. Low dynamics in block distribution improves the cache performance of all the replacement schemes. CRB scheme that is based on the number of replicas in the network is significantly better than MRB which is solely based on the probability of a block being present in a node that is going to send an unchoke, which shows that the rarest-first selection strategy is the single most important factor determining the cache performance.

5.3. Practical Caching

In this section, we compare the different caching schemes with practical considerations in mind. In a real BitTorrent network, the unit of data transfer is not a file block, but a subblock. A subblock is typically much smaller than a file block. Moreover, the subblocks within a block can be uploaded/downloaded in any order. Thus our method of caching at the block level may not be practical. The reason is as follows. Since the transfer units are subblocks, the cache system has to now keep state of the different blocks which is incomplete, i.e, a subset of the subblocks within the block have arrived at the cache and the rest have not. Potentially, the cache might have to keep state for all the blocks in the cache, which defeats the entire purpose of caching. Ideally we would like to have state only for a fixed amount of blocks. Now the problem of caching becomes different. Initially, we were assuming that whenever the last remaining subblock arrives, the cache has the whole block ready. However, now we need to make sure that all the subblocks within a block have arrived and the cache has state for a particular block too. There are few points of view that can be considered. First, we can do caching at the subblock level. This is not a feasible option as all the protocol messages about the availability of blocks are at the block level. Second, we know that a peer requests for a

Policy	Hit Rate (%)
Random	14.3
LRU	28.5
CRB	52.3
MRB	23.8

Table 5. The average hit rates of different cache policies under heterogeneous bandwidth distribution over all the caches with realistic caching.

block that is incomplete before it starts downloading a complete block. This might suggest that at any given time there should not be too many incomplete blocks and hence, there might not be any need to keep state for all the blocks in the file. Unfortunately, our simulations suggest otherwise. The simulations show that at any given time the number of incomplete blocks seen by a cache grows indefinitely. Thus, this option is also rendered futile.

Given the above problems, the only alternative left is to incorporate caching before we decide to keep state for any block. In other words, we apply the caching algorithm when a block is requested from within an ISP for the first time. The caching algorithm decides whether to keep state for the new block or not and which block to discard. Note that the block being discarded may be complete or may be incomplete. We used this approach and got favorable results. As Table 5 shows, the hit rate for CRB remains similar to the case with idealistic caching.

5.4. Proactive Dissemination

Caching enhances the performance of the BitTorrent network, besides reducing traffic. The gains achieved by using caching can be further improved by proactively disseminating the content present in the cache. Such a proactive method is based on a simple hack that has the potential to increase the temporal locality of the requests made to a block and hence increase the hit rates significantly. In brief, such a proactive scheme can be achieved as follows. The cache can start dummy BitTorrent clients. These dummy BitTorrent clients can start up connections with the peers within the ISP and advertise blocks present in the cache to those peers and unchoke them for free. On getting unchoked, the normal peers can start requesting blocks from the dummy clients. Since the blocks requested by the normal peers can only be the blocks hosted by the cache at the current point of time, the hit rate of the cache should go up.

Unfortunately, by the time the report is written, we have not yet completed the implementation of this scheme in the simulator, and are not able to assess it.

6. Challenges and Future Directions

More simulations show that there is still a gap of around 20% between Belady's algorithm and the CRB policy. Hence, there is scope for further improvement. However, due to the distributed nature of BitTorrent and the impossibility of future prediction, this might be the best that one can do. As a part of further study, finding efficient and better replacement policies is one of our concern. Even if it is not feasible to design a policy that is comparable to Belady's scheme, it would be nice if we could come up with some analysis of proof of what cache performance on average can be achieved.

Another challenge that we face is a real implementation such a caching scheme. A real implementation would require BitTorrent traffic detection which might not be easy. If time permits we would like to implement a small prototype of the cache and see how much it helps.

There are two more problem related to our approach of improving BitTorrent performance and reducing ISP's traffic costs. First, the cache based scheme will not be able to reduce ISP costs due to outgoing traffic unlike the biased neighbor selection scheme which achieves lower traffic in both directions. Second, BitTorrent uses a tit-for-tat policy while choosing a peer to upload to. In our scheme since the cache may be the one uploading on behalf of a peer, the tit-for-tat policy will no longer be valid and hence the transparency is compromised.

References

[1] Ashwin Bharambe, Cormac Herley and Venkat Padmanabhan, "Analyzing and Improving a BitTorrent Network's Performance Mechanisms", *IEEE INFOCOM 2006, Barcelona, Spain.*

[2] Ruchir Bindal, Pei Cao, William Chan, Jan Medved, George Suwala, Tony Bates, Amy Zhang, "Improving Traffic Locality in BitTorrent via Biased Neighbor Selection", *ICDCS 2006.*

[3] EcontentMag.com, "Chasing the user: The revenue streams of 2006", December 2005 Issue.

[4] Cachelogic - advanced solutions for peer-to-peer networks, http://www.cachelogic.com/

[5] Bram Cohen, "Incentives Build Robustness in BitTorrent", *Workshop on Economics of Peer-to-Peer Systems, June, 2003*

[6] Thomas Karagiannis, Pablo Rodriguez, Konstantina Papagiannaki, "Should Internet Service Providers Fear Peer-Assisted Content Distribution? " *IMC 2005.*

[7] Dongyu Qiu, R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks", *SIGCOMM 2004*

[8] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, and David Becker, "Scalability and Accuracy in a Large-Scale Network Emulator", *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI), December 2002*