Herding BitTorrent Traffic away from Expensive ISP Links

UMass CMPSCI Technical Repor UM-CS-2008-036

Bruno Ribeiro*, Daniel Figueiredo+, Don Towsley*

*Computer Science Department
University of Massachusetts
Amherst, MA, 01003
{ribeiro,towsley}@cs.umass.edu

+Computer and Systems Engineering Department
UFRJ/COPPE
Rio de Janeiro, RJ, Brazil
daniel@land.ufrj.br

Abstract

The steady increase in access link capacity of broadband subscribers coupled with the significant increase in traffic, partially due to popular peer-to-peer (P2P) file sharing applications, has had an adverse economic impact on last-mile Internet Service Providers (ISPs). Reducing the cost associated with P2P traffic has become a major issue for ISPs worldwide and has triggered several proposals. However, previous works advocate for changes to P2P software or require application-specific packet identification and manipulation. This paper presents a distinct approach. We leverage on the observation that traffic costs can be reduced by diverting traffic away from expensive links and into cheaper links. We focus on BitTorrent (BT) traffic and propose a change to the ISP's access router packet schedulers in order to divert traffic away from links labeled as expensive by the ISP. Our proposed packet scheduler is stateless. Our approach requires no change to applications, is blind to packet payload, and only requires changes to access routers of ISPs that desire to reduce their BT traffic costs. Our simulation and experimental results indicate that our approach can successfully herd BT traffic towards cheaper links (up to 50% reduction), without sacrificing user-level performance.

1. Introduction

The last decade has witnessed a steady increase in the access link capacities of broadband subscribers, which are mainly home users. Despite its initial commercial success, broadband access has recently had an adverse economic impact on Internet Service Providers (ISPs), mainly due to the significant increase in traffic traversing their networks [4, 14, 20]. Recent studies indicate that large fractions of such traffic are generated by peer-to-peer (P2P) file sharing applications, such as BitTorrent and Gnutella [17]. Since for most ISPs traffic costs are proportional to traffic volume, the cost of P2P traffic can be reduced using one (or both) of the following approaches: *(i)* reduce the amount of traffic generated by P2P applications; *(ii)* divert traffic generated by P2P applications into cheaper links. In a competitive market, where ISPs share a subscriber population, ISPs prefer cost reduction mechanisms that do not degrade user-level performance. For such a reason, several proposals for diverting P2P traffic without degrading user-level performance have appeared in the literature [1, 3, 5, 12, 18, 19, 22].

Unfortunately, these mechanisms are generally inadequate for several reasons. First, the diversity in the implementation of a given P2P protocol, as well as the increasing use of payload encryption, makes it hard, if not impossible, to implement mechanisms that are application dependent (e.g., payload-based application identification). Second, the vast amount of illegal content present in P2P applications discourages large ISPs from adopting solutions that require interacting with the applications (e.g., content caching or tracker modifications). Third, changing the behavior (i.e., source code) of P2P applications is not an immediate option available to ISPs, as most applications are developed by third parties. Moreover, even if P2P applications were to be modified, it is not clear that such modifications would be aligned with the individual interests of every ISP.

In contrast, we take a fresh look at the problem of reducing the cost of P2P traffic in a given ISP, focusing specifically on BitTorrent traffic. We propose a novel approach to divert traffic that requires no changes to the application nor resorts to any application-dependent packet classification. Our proposal is practical and can be readily deployed by ISPs, as we next describe.

Reducing the economic cost of traffic in an ISP can be a simple matter of transferring traffic load between ISP links. Figure 1 illustrates this approach. In this example we focus on the cost of serving traffic of a user of ISP A. Assume that forwarding or receiving traffic through two of the three ISPs connected to A is expensive (the two links marked with

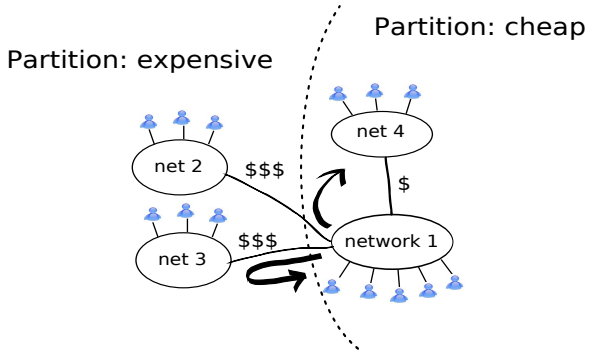


Figure 1: Reduce the ISP's operating costs by diverting traffic into the cheap partition.

\$\$\$). However, there is an ISP that charges A much less, making the link cheap. Note that it may also be cheap for the ISP to use its own network. Thus, in order to reduce costs ISP A should try to change its traffic matrix, i.e., induce applications to exchange traffic with cheaper hosts whenever possible.

One idea to transparently induce applications to use cheaper hosts is to treat traffic differently. For example, traffic exchanged with cheap hosts can be provided *some* priority over traffic exchanged with expensive hosts. A common mechanism to differentiate traffic are class-based packet scheduling policies. Under the right conditions and using an adequate scheduler, such differentiation might suffice to induce applications to exchange more traffic with cheap hosts and less with expensive hosts. This is the essence of our approach.

In a nutshell, our solution is a simple class-based packet scheduling discipline deployed at the subscriber's access link. In particular, the scheduler we propose operates only on the link from the subscriber to the ISP (i.e., uplink) and no change is required in the reverse direction (i.e., downlink). Our solution targets mostly BitTorrent traffic, since the characteristics of this application, as we soon describe, provide the necessary conditions for our approach to work effectively. In what follows, and in the remainder of this paper, we simplify our exposition by restricting our discussion to two classes of links: cheap and expensive. However, our approach can be extended to accommodate more cost classes.

The proposed scheduler, in brief, works as follows. Packets that arrive to the uplink of a user are classified into one of two classes: *expensive* or *cheap*. This classification is trivial since an ISP knows if a given packet will traverse expensive links on its path to its destination, based on the packet's destination IP address. The scheduler maintains two separate finite capacity queues, one for each class, and mixes the FIFO and strict priority disciplines as follows. It schedules W consecutive packets according to their global order of arrival (FIFO) and then the highest priority packet available (strict priority). This pattern is repeated indefinitely.

Recall that the desired effect is for expensive flows to decrease their sending rates, while for cheap flows to increase theirs, thus, reducing the costs to the ISP. However, can the proposed scheduler effectively divert traffic? Moreover, will user-level performance, such as download completion times, be preserved? In this paper, we show that when considering BitTorrent applications the answer to both these questions is yes. We have conducted an extensive evaluation of our proposal using simulations and world-wide scale PlanetLab experiments. All our results indicate that BitTorrent traffic on expensive links can be significantly reduced (e.g., up to 50% reduction) without increasing download completion time.

It is important to note that conventional class-based schedulers, such as strict priority or weighed fair queueing, are not appropriate for our purposes, as we soon discuss. For example, the greedy approach taken by the strict priority scheduler, where the cheap class is always served ahead of the expensive class, can backfire and increase ISP expenditure when users are running BitTorrent.

The remainder of this paper is organized as follows. In Section 2 we formally describe the problem we investigate, stating our assumptions. Section 3 describes the proposed scheduler and discusses why other schedulers are not appropriate. In Section 4 presents the evaluation through simulation and experimental results, respectively. Section 5 presents the related work. Finally, Section 6 concludes the paper.

2. Problem Statement and Solution

The recent clash between users of P2P file sharing applications and ISPs has its roots in a business dilemma. While ISPs that provide Internet access to users enjoy the revenue generated by broadband subscribers, they prefer that such users not fully utilize the contracted bandwidth, since users generally pay a flat rate, independent of the amount of traffic exchanged. However, P2P applications tend to exchange large amounts of data (e.g., movie downloads), leading to a high utilization of the contracted bandwidth. Such behavior, if widely adopted by broadband users, can have a significant impact on the operational costs of ISPs.

The operational costs borne by an ISP can be divided into fixed costs and variable costs. Among the variable costs, is the cost associated with sending traffic to or receiving traffic from its neighboring ISPs. Network links between ISPs are regulated by financial contracts, which can vary in their pricing and are usually volume dependent (e.g., monthly cost is proportional to the amount of traffic). In particular, an ISP

that has multiple neighbors is likely to have different costs associated with these links. Thus, a way to reduce costs is to avoid using expensive links, and use cheap links when possible. But should an ISP always do this? More fundamentally, why maintain the expensive links in the first place?

In a competitive market, where multiple ISPs share a potential subscriber population, user performance matters. A given ISP needs to provide rich connectivity to its users, many times establishing multiple links to other ISPs in order to efficiently cover the entire IP address space (e.g., AS peering links). In particular, users paying for broadband access that receive poor performance are likely to switch to another ISP (possibly the reason why most ISPs tolerate P2P file sharing applications). Thus, ISPs would prefer not to adopt practices that reduce costs at the expense of user performance. This is a serious concern for ISPs, as it jeopardizes their main source of revenue namely, broadband subscribers.

Therefore, we search for an approach that can reduce costs by diverting traffic away from expensive links and into cheaper links, while preserving user-level performance. Moreover, we search for a solution that is both practical, namely that an ISP alone can deploy, and effective. Therefore, such a solution should not be application-dependent, such as requiring packet-based application identification, nor require changes to applications, something not under the control of the ISP.

2.1 Scheduling: a promising approach

Given the tight constraints presented above, it might seem that there is little hope for an effective mechanism to reduce the cost of P2P traffic. However, an ISP has control of a fundamental part of the end-to-end path, namely, the packet scheduling policy at the access link of their subscribers. Thus, our hope is to leverage this scheduling policy in order to construct a mechanism that will reduce the costs of an ISP by diverting traffic away from expensive links, without degrading user-level performance. But can this lead to an effective solution? And if so, under what conditions?

A few observations about BitTorrent, and more broadly modern P2P file sharing applications, will help us understand why and when scheduling-based solutions can be effective. First, a key observation is that the same data (e.g., data block) is available at several locations, in hosts possibly behind cheap and expensive links. Moreover, retrieving information from a cheap host is in many cases equivalent (or better), in terms of user-level performance, to retrieving it from an expensive host. Therefore, it is possible to reduce costs without degrading user-level performance, by simply changing the host from which information is retrieved.

Second, due to their high efficiency, P2P file sharing applications tend to saturate the uplink of a user, generating prolonged busy periods and occasional packet drops (i.e., high link utilization). Thus, scheduling can play an important role in controlling traffic, such as deciding the order at which packets are served. Note that if no queue was ever generated then work-conserving scheduling policies would have no impact.

Third, most P2P file sharing applications simultaneously connect to multiple hosts to exchange information, giving rise to multiple simultaneous packet flows. Thus, a class-based scheduling policy can have an impact on which hosts the application connects to.

Fourth, BitTorrent operates an incentive mechanism, such that the download bandwidth obtained by the application is an increasing function of the upload bandwidth. This means that the download bandwidth of BitTorrent can be indirectly controlled through its upload bandwidth. This indicates that a schedule in the uplink might also impact the downlink without any changes to the downlink scheduler. In fact, Piatek et. al provide evidence that the download bandwidth in BitTorrent is indeed a convex function of the upload bandwidth [16].

Last, but fundamentally important, P2P applications use the TCP protocol to exchange data. Recall that the congestion control mechanism of TCP reacts to packet losses and delay, either by increasing or decreasing its sending rate. Thus, flows experiencing a lower packet loss and delay are likely to have a higher throughput. This would allow a class-based scheduler to impose differences on the throughput of different TCP flows. For example, cheap flows can be made to experience lower packet losses and delays than expensive flows, thus being more likely to increase their sending rate. In fact, this issue is closely related to the framework of *congestion pricing* [13].

Given the observations above, at least intuitively it seems that a class-based scheduling policy controlling the uplink can be used to divert traffic away from expensive links without degrading user-level performance. In the next section we describe the proposed scheduler in detail.

3. The vFIFO+PRI scheduler

In this section we first describe in detail the proposed class-based packet scheduler, illustrated in Figure 2. In what follows, we also discuss why conventional class-based schedulers, such as weighted fair queueing and strict priority are not an appropriate solution for our problem.

The first task of the class-based scheduler is to classify arriving packets into one of its classes. In this paper, we assume there are only two possible classes, namely cheap (high priority) and expensive (low priority). However, this can be readily extended, if needed. From the ISP perspective a packet is classified as *expensive* if it will traverse expen-

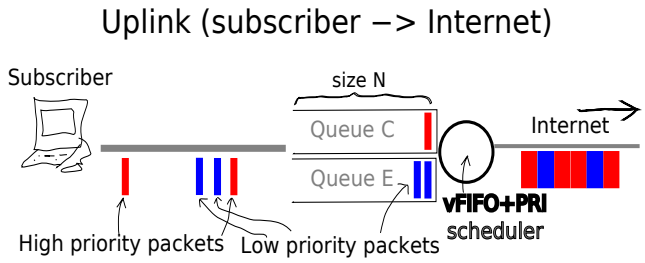


Figure 2: Schematic view of the proposed scheduler

Algorithm 1: Enqueue a new packet

input : packet pkt, two physical queues Q_C and Q_E, individual queue capacity N, and virtual FIFO queue F
output: Updated queues F and Q_C or Q_E

```
/* pkt.class ∈ {C, E}                                   */
i ← pkt.class;
if |Q_i| < N then
    pushTail(Q_i, pkt);
    if # of class i packets in F < N then
        pushTail(F, i);
    end
end
```

sive links on its path to its destination; all remaining packets are classified as *cheap*. For the ISP this effectively divides the Internet (i.e., the IP address space) into two partitions. For example, ISP A of Figure 1 partitions the Internet into *cheap* and *expensive*. Note that this classification is trivial, as ISPs know if a particular packet arriving from the subscriber will be forwarded through expensive links (based on the destination IP address of the packet). This classification can be applied to flows[1], since all packets in a given flow receive the same classification.

After being classified, a packet is placed in its corresponding queue. In particular, the scheduler maintains two finite capacity queues, Q_C and Q_E, where cheap and expensive packets, respectively, are stored. In order to maintain the global ordering of arrival, the scheduler maintains a virtual finite capacity queue F, where packet classes are stored. Note that queue F stores just the packet class (and nothing about the packet) as this is sufficient to establish the global ordering among the arrivals of the different classes. The scheduler serves using a global FIFO discipline and a strict priority discipline, alternating between these two disciplines. In particular, for every W packets served according to their global order of arrival, one packet is served according to the strict priority discipline. This procedure is repeated indefinitely. Note that the virtual FIFO queue F defines which *class* is going to be served next by the FIFO scheduler. When a packet is served by the priority scheduler it is **not** removed from the virtual FIFO queue (this measure is taken to reduce bursts of packets in the expensive class). The FIFO scheduler serves the next non-empty queue that is first in the virtual FIFO queue. We refer to the proposed scheduler as vFIFO+PRI(W), with W being the its sole parameter.

[1]A flow is a sequence of packets relatively close together in time with identical source and destination IP addresses, source and destination port numbers, and protocol number.

Algorithm 2: Dequeue a packet

input : Two physical queues Q_C and Q_E, virtual FIFO queue F, scheduler weight W, and counter c
output: Packet to be served, updated queues Q_C and Q_E, updated queue F, and updated counter c

```
pkt ← null;
if |Q_C| + |Q_E| > 0 then
    /* Virtual FIFO                                     */
    if c < W then
        repeat
            i ← popHead(F);
        until |Q_i| > 0 ;
        pkt ← popHead(Q_i);
        c ← c + 1;
    end
    /* Priority                                         */
    else
        if |Q_C| > 0 then
            i ← C;
        else
            i ← E;
        pkt ← popHead(Q_i);
        c ← 0;
    end
end
return pkt;
```

The pseudo-code of the algorithm to enqueue a packet as well as the algorithm to dequeue (serve) a packet are given in Algorithms 1 and 2, respectively. These algorithms fully describe the proposed scheduling policy. Note that the scheduler is simple to implement and fast to execute, and more importantly, it has an adjustable priority given by the parameter W. When W is zero, the scheduler behaves as a strict priority scheduler. As W increases, the scheduler behaves more and more like the classical FIFO scheduler. We next describe why previous class-based schedulers are not appropriate for our purposes.

3.1 WFQ scheduler: an unfit approach

Consider the traditional weighted fair queue (WFQ) scheduler operating with two classes, namely a cheap and a expensive class. This scheduler will unequally divide the bandwidth capacity of the link between the two classes. Thus, the cheap class will receive a fraction $\omega > 1/2$ of the link's bandwidth capacity, since it should have some priority over the expensive class. The expensive class will receive the rest of the link's capacity. Unfortunately, this discipline does not appropriately differentiate the classes at the flow level. The problem occurs because the WFQ discipline only guarantees that the link's bandwidth capacity will be divided among the classes according to ω. However, it makes no guarantees as to the relative bandwidth that individual flows will receive.

For example, consider a scenario where $\omega = 2/3$ and there are four flows traversing the link. However, assume that three of these flows belong to the cheap class, and that both queues are non-empty (cheap and expensive classes have packets waiting). Thus, each cheap flow receives 2/9 of the bandwidth capacity while the expensive flow receives 1/3. Thus, the expensive flow will receives more bandwidth than the cheap flows, inverting the desired effect! Clearly, WFQ is unfit for our purposes.

3.2 Strict priority scheduler: potential drawbacks

We now consider the traditional strict priority scheduler operating with the cheap and expensive classes. Thus, packets in each queue are served in FIFO order, however, the expensive class is only served when the cheap queue is empty. Recall that this scheduler is obtained by setting $W = 0$ in the proposed scheduler.

Under strict priority, the packet loss probability of the cheap class will be minimal, usually around zero, while the expensive class will have to bear the entire link cost, being exposed to high packet loss probabilities and large delays. This large difference between the two classes would potentially divert more traffic away from the expensive links than any other work conserving split. Since the goal of the ISP is to reduce its financial costs by moving traffic away from expensive links, the strict priority scheduler seems potentially optimal from the perspective of the ISP. Unfortunately strict priority can starve expensive class packets, possibly impairing other subscriber applications. Another undesired effect of the strict priority scheduler may happen under certain scenarios where it may *increase* ISP expenditures, as described next.

Consider the pricing scheme where ISPs are billed according to their largest traffic bursts. These schemes consider a charging horizon (e.g., one month) that contains the traffic volume of N five-minute time periods. Under maximum rate charging, the ISP is charged based on the largest of these N recorded volumes, regardless of all volumes in other time periods. Under 95-th percentile charging, the ISP is charged according to the $(N/20)$-th largest traffic volume. Refer to [15] for more details on these pricing schemes.

When considering BitTorrent applications, a strict priority scheduler can sometimes generate a maximum traffic burst higher than a less strict priority scheme, increasing the cost to the ISP. We discuss this phenomenon, which we refer to as the *partition problem*, in the next section.

3.3 The partition problem

In this section, we intuitively present the partition problem through the following example. Consider a swarm (i.e., set of peers interested in disseminating and retrieving a given file) formed by few seeders (i.e., peers that have the entire file and behave like servers for some time) and a large number of homogeneous leechers (i.e., same bandwidth peers that need the content) that join the system almost at the same time (i.e., a flash crowd). Suppose seeders serve all data blocks a certain number of times and then depart from the system. Moreover, suppose that the set of leechers is strictly partitioned, such that leechers prefer to serve other leechers in their own class. After all seeders have departed the system, there might be a non-negligible probability that no leecher within a given partition has a copy of one (or more) particular data block. As time progresses, leechers within a partition tend to exchange data blocks among themselves. And because all leechers have arrived at about the same time and are homogeneous, leechers in a partition finish exchanging their blocks almost simultaneously. At this point, nearly all leechers in a given partition will request the missing data blocks to peers outside the partition, potentially generating a large burst of traffic between partitions. Moreover, this burst of traffic can be larger than the bursts generated when no partitions are used.

The above phenomenon can occur because strict partitioning the set of peers can lead to a synchronization of the demand for particular data blocks. This synchronization is more easily observed when peers arrive in batches to a partition that has missing data blocks. In the appendix, we provide a simple model to capture the likelihood of having partitions with missing blocks.

Note that partitioning the set of peers occurs whenever a strict preference mechanism is used for peer selection. For example, a strict priority scheduler, which prefers a set of peers over the others, will partition the system. Another example is strictly exploiting network locality, where peers within the ISP are always preferred for exchanging data.

3.4 Deployment issues

In this section we discuss a few issues related to the deployment of the proposed scheduler in a given ISP. Our so-

lution can be deployed both when the access link uses the cable modem or the ADSL technology. In the cable modem setting, ISPs today already deploy a per subscriber rate limitation mechanism at their access router (i.e., their end of the access link). This is needed to ensure that a given subscriber receives just the contracted bandwidth, and not more. Our solution can be readily deployed in this setting, simply coupling the proposed vFIFO+PRI scheduler with the rate limitation mechanism.

In the ADSL setting, the proposed scheduler could be placed inside the user's modem, scheduling packets to be transmitted on the access link. Another option is for the ISP to increase the line speed of the access link and then deploy a rate limitation mechanism at their end of the link. This would ensure that the bottleneck is inside the ISP and would require no change to the user modem. In this case, the proposed scheduler could be deployed in conjunction with the rate limitation mechanism.

Finally, recall that the proposed scheduler operates only on the uplink, such that download-intensive applications, such as Web browsers and Email, would remain mainly unaffected. However, to further minimize the impact of the proposed scheduler on these applications, ACK packets traversing the uplink can be treated differently by the scheduler, for example, completely bypassing it.

4. Experiments

In this section we evaluate our approach by showing that it can effectively reduce traffic costs when compared to the traditional FIFO scheduler. Traffic costs can be defined using three metrics, which reflect the three most common traffic-based charging schemes [15]: the overall average, the maximum, and the 95-th percentile rates, all taken over 5-minute interval bins. Please refer to Section 3.2 for more details on these charging schemes. For example, under the 95-th percentile charging scheme, a reduction in the 95-th percentile rate of a link results in a cost reduction for the link. Thus, our measure of interest is the reduction in traffic between partitions using these three metrics. We perform experiments using the NS-2 [23] simulator and the PlanetLab [6] network.

Unless stated otherwise we use the following scenario: We consider a single BitTorrent swarm. Subscribers participate in the same swarm and have capacities taken from Table 1. These subscribers are randomly split between partitions A and B, which guarantees that end-to-end network performance is independent of partition memberships. Partition A is typically small. We restrict the deployment of our vFIFO+PRI scheduler to peers in partition A only. All other links, including upload links of peers in partition B, use regular FIFO schedulers. This is an important characteristic of our experiment: **only subscribers in partition A implement our vFIFO+PRI scheduler**. All peers in partition A see peers in partitions A and B as "cheap" and "expensive", respectively. Note that such scenario tests our approach under unfavorable conditions as partition A is typically small and peers in B do not implement the vFIFO+PRI scheduler.

Capacity	Probability	Downlink	Uplink
High	0.1	10Mbps	5Mbps
	0.15	3Mbps	1Mbps
Low	0.525	1.5Mbps	384Kbps
	0.225	784Kbps	128Kbps

Table 1: Bandwidth used in PlanetLab experiments. This bandwidth distribution is closely related to the values reported in [2].

Our results show that our approach can reduce traffic between partitions by up to 50% (both in the simulation and in our PlanetLab experiments). Our experiments also indicate that our approach does not increase BitTorrent download completion times. The results of our simulations are presented in Section 4.1, followed by the results of our PlanetLab experiments in Section 4.2.

4.1 Simulation

In this section we show the results of our simulations. Our simulator is a modified version of the NS-2 simulator [23] that implements two extra modules: our proposed scheduler and the BitTorrent client developed by Eger et at. [9]. All simulations focus on the following basic scenario: 100 peers split into two partitions (A and B). Figure 3 shows the topology of our simulation. Partition A and B peers are assigned to networks A and B, respectively. The link between networks A and B is over-provisioned to guarantee that the end-to-end performance between pairs of peers is virtually independent of their partition assignments. Downlink and uplink capacities are chosen according to the distribution shown in Table 1 and kept fixed across experiments. The swarm file consists of 430 MB (megabytes) divided into 1720 blocks of 256 kilobytes each. Upon finishing its download a peer becomes a seeder and serves the data for a time given by an exponential random variable with average of one minute. Each experiment is the result of 30 runs where two runs differ only in the times in which peers decide to join the swarm.

Single seeder with flash crowd (FC).

Our first experiment simulates a flash crowd where there is a single seeder in partition B and all remaining 99 peers arrive randomly between zero and 2 minutes. In this experiment we compare schedulers FIFO, vFIFO+PRI(4), vFIFO+PRI(2), and strict priority. The notation vFIFO+PRI(W)

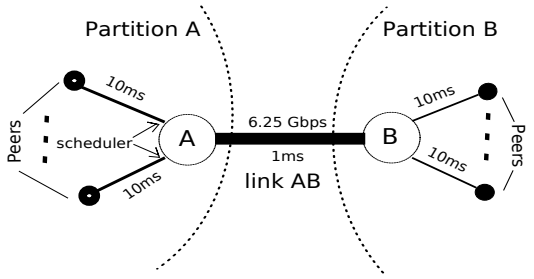


Figure 3: Simulated topology

refers to the proposed scheduler with parameter W, where W is the number of packets served according to the FIFO discipline for every packet served according to strict priority discipline.

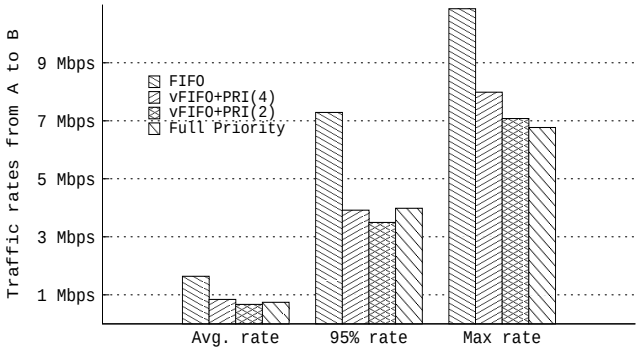
Figure 4 shows the results averaged over 30 runs. We can observe that the vFIFO+PRI(2) scheduler is able to reduce the average traffic rate between partitions A and B by nearly 50% in both directions. Thus, despite being implemented only at peers in partition A, our scheduler effectively triggers BitTorrent's reciprocation mechanism (a.k.a. tit-for-tat) to reduce traffic from B to A. We also observe a significant reduction in the maximum and 95-th percentile rates on the traffic from A to B, while the reverse direction shows a much smaller reduction. Note that the vFIFO+PRI(2) scheduler provides almost the same reduction in traffic as the strict priority scheduler with the advantage of being less likely to starve low priority flows. Finally, we observe that the vFIFO+PRI(4) scheduler is less effective than the vFIFO+-PRI(2) scheduler when it comes to reducing the AB traffic.

Lower peer arrival rate with 50% as seeders.

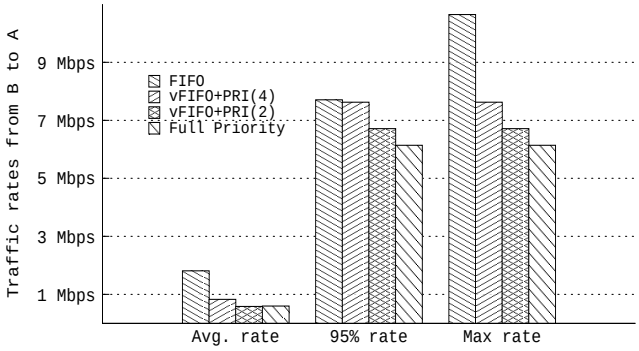
In our second experiment we simulate peers arriving randomly between zero and 10 minutes. Different from the previous experiment (where there is only one seeder in partition B) half of the peers in partitions A and B are seeders (10 and 40, respectively). The results in Figure 5 show the average over 30 runs. We can see that the vFIFO+PRI scheduler achieves even larger reductions (more than 50%) in inter-partition traffic when compared to our previous experiment. Note that here the vFIFO+PRI(4) scheduler reduces the AB traffic almost as much as the vFIFO+PRI(2) scheduler.

BitTorrent performance.

Here we assess the average user-level performance of our experiments. Figure 6 shows the average fraction of peers that finish within 6 hours (solid bars) with the respective standard deviation (lines). Table 2 shows the average download completion times of peers that finish within 6 hours. These results show that our approach does not have a negative impact over BitTorrent performance. In fact, the proposed scheduler is sometimes able to improve BitTorrent's performance. Note that the strict priority scheduler can induce slightly longer downloads when compared to the vFIFO+PRI(4) scheduler. This behavior is likely due to the *partition problem* described in Section 3.3 but further investigation is required. In the next section we provide the results of our PlanetLab experiments.



(a) Traffic at A→B



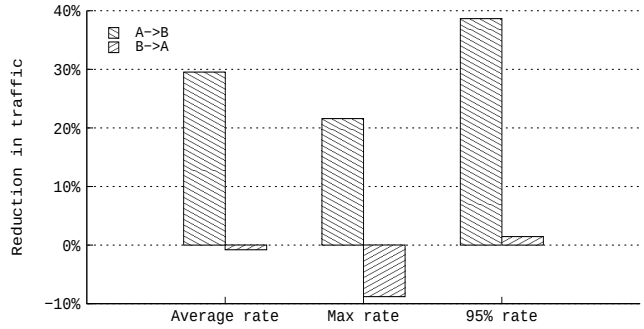
(b) Traffic at B→A

Figure 4: Single seeder and 99 leechers. Leechers arrive randomly between zero and 2 minutes. Graphs show the overall average, 95-th percentile, and maximum traffic rates in the AB link. Figure 5(a) shows the direction A→B and Figure 5(b) the direction B→A.

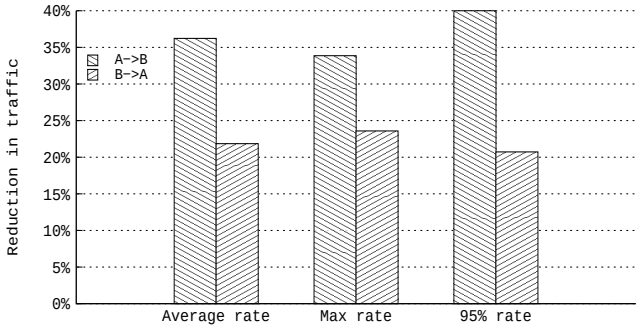
4.2 PlanetLab experiments

In our Internet experiments we use the PlanetLab network and implement a user-level scheduler using PlanetLab's TAP/TUN support. PlanetLab nodes are carefully chosen among the most unloaded and reliable servers over five continents. The number of PlanetLab nodes participating in our experiments ranges from 82 to 123. Upload and download links are artificially rate limited with speeds taken from a distribution based on the access link experiments [2]. Nodes are located mainly in North America and Europe (with a small fraction of them in other continents).

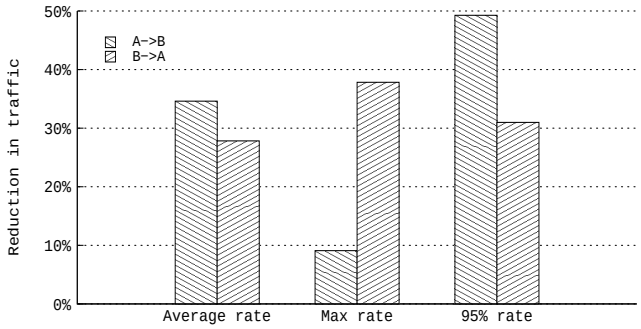
These experiments emulate a swarm that starts with one



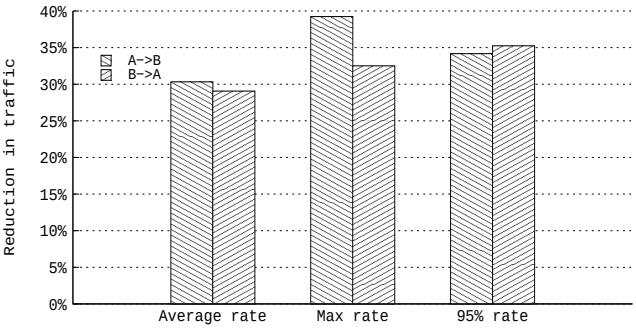
(a) Run 1, Azureus client, large buffer, 20% peers in partition A



(b) Run 2, Bittornado client, small buffer, 20% peers in partition A



(c) Run 3, Azureus client, large buffer, 20% peers in partition A



(d) Run 3, Azureus client, large buffer, 50% peers in partition A

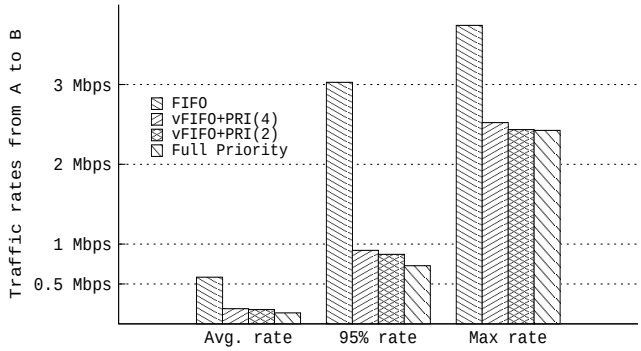
Figure 7: PlanetLab experimental results. In this experiment we replace the FIFO scheduler with a vFIFO+PRI(2) scheduler and plot the reduction in traffic in the link AB (bars show directions A→B and B→A). Our experiments indicate that a vFIFO+PRI(2) scheduler can effectively reduce the BitTorrent traffic between partitions in most scenarios.

Experiment	Scheduler	Avg. download time (hours)
Single seeder w/FC	FIFO	3:18
	vFIFO+PRI(4)	2:25
	vFIFO+PRI(2)	2:24
	Priority	2:44
50% seeders w/o FC	FIFO	2:20
	vFIFO+PRI(4)	2:18
	vFIFO+PRI(2)	2:22
	Priority	2:22

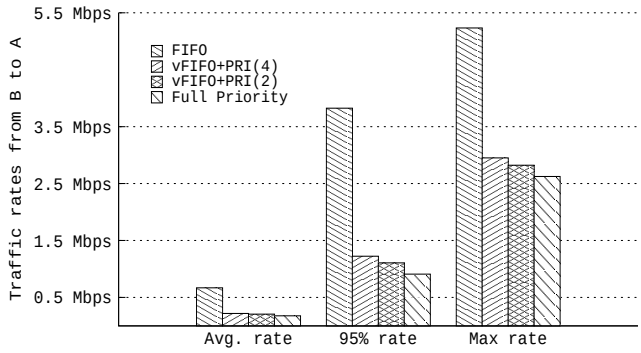
Table 2: Average download completion times of peers that finish within 6 hours.

seeder. Leechers join the swarm randomly between zero and 30 minutes. PlanetLab nodes are instrumented with a user-level packet scheduler that manages buffers using a simple drop-tail strategy; i.e., incoming packets are dropped when the buffer is full. We implement two packet scheduling disciplines: FIFO and vFIFO+PRI. PlanetLab nodes are divided into *high* (between 19 and 32 nodes) and *low* (between 62 and 91 nodes) bandwidth nodes. The starting seeder has high bandwidth, 10Mbps (megabits/s) downlink and 5Mbps uplink. High and low bandwidth nodes have bandwidth as described in Table 1. Unless stated otherwise the queue size is set to 1.5 times the bandwidth value, i.e. maximum queue delay of 1.5 seconds, to reflect real access link queue sizes [8]. A leecher becomes a seeder after completing the file download but leaves the swarm after a time randomly distributed between zero and 2 hours. The starting seeder is located inside network B. The torrent file size is again 430 MB. We perform our experiments with two of the most popular BitTorrent clients, Azureus [24] and BitTornado [25], both with their default configuration (e.g. maximum of 4 simultaneous uploads).

These experiment were performed between December 2007



(a) Traffic at A→B



(b) Traffic at B→A

Figure 5: 50 seeders and 50 leechers. Leechers arrive randomly between zero and 10 minutes. Graphs show the overall average, 95-th percentile, and maximum traffic rates in the AB link. Figure 5(a) shows the direction A→B and Figure 5(b) the direction B→A.

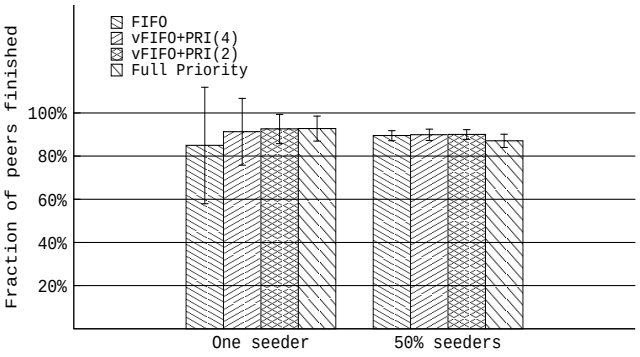


Figure 6: Fraction of peers that finish the download within 6 hours. This graph shows the average (solid bars) with the standard deviation (lines).

and May 2008. We define an experiment *run* to be an experiment where the same capacities and peer starting times were used. In what follows we present the results of four experiments taken from three different runs. But before we present our results we believe that our PlanetLab experiments require a disclaimer. Our experiments are sensitive to resource (e.g. CPU, network) variability, i.e., comparing two runs requires the same resources availability for both runs. PlanetLab is a shared worldwide experimental network and as such suffers from a good amount of resource variability. Nevertheless, we believe that our PlanetLab experiments can be used to indicate potential benefits of our approach on a real network.

Run 1: 20% peers in A, Azureus client.

In the first experiment we place 14 leechers inside partition A and 71 leechers inside partition B. In this experiment Azureus [24] acts as the BitTorrent client. The results shown in Figure 7(a) indicates that our approach is able to significantly reduce (up to 38%) the traffic flowing from partition A to partition B. The reverse traffic, from B to A, suffers only an 8% increase over its maximum rate. This shows that in this experiment our scheduler failed to get leechers in partition B to reciprocate the reduction in traffic from A to B.

Run 2: 20% peers in A, smaller queue size, Bittornado client.

In the second experiment we reduce queue sizes (as to make the maximum delay 500 milliseconds) and use Bittornado [25] as our BitTorrent client. There are 23 leechers inside partition A and 100 leechers inside partition B. The results shown in Figure 7(b) indicate that the vFIFO+PRI(2) scheduler can significantly reduce (up to 40%) the traffic exchange between partitions (in both directions).

Run 3: 20% peers in A, Azureus client.

In this third experiment we place 17 leechers inside partition A and 67 leechers inside partition B. As in *run 1* we use Azureus [24] as our BitTorrent client. The results in Figure 7(c) show a greater reduction in traffic over the link AB when compared to *run 1*. The reduction in the 95-th percentile in the A→B direction was close to 50%.

Run 3: 50% peers in A, Azureus client.

In this last experiment we place 41 leechers inside partition A and 41 leechers inside partition B. Capacities and peer starting times are the same as in the previous experiment. The results shown in Figure 7(d) indicate a good reduction (up to 40%) in traffic in the link AB. Our results also show that the reduction in traffic in the direction A→B is similar to the reduction in traffic in the direction B→A.

BitTorrent performance.

In the above experiments, both schedulers achieve roughly the same download completion times. Figure 8 shows the

Experiment	Scheduler	Avg. download time (hours)
Run 1	FIFO	2:57
	vFIFO+PRI(2)	3:07
Run 2	FIFO	3:11
	vFIFO+PRI(2)	2:19
Run 3 20% in A	FIFO	3:01
	vFIFO+PRI(2)	2:49
Run 3 60% in A	FIFO	2:46
	vFIFO+PRI(2)	2:40

Table 3: Average download times for each experiment.

fraction of peers that finish the download within 6 hours and Table 3 shows the average download times for each experiment. As with the simulation results, the experimental results also indicate that the vFIFO+PRI(2) scheduler is able to significantly reduce traffic between partitions without hurting BitTorrent's performance. In particular, we again observe that vFIFO+PRI(2) can reduce the average download completion time, up to 25% (on Run 2).

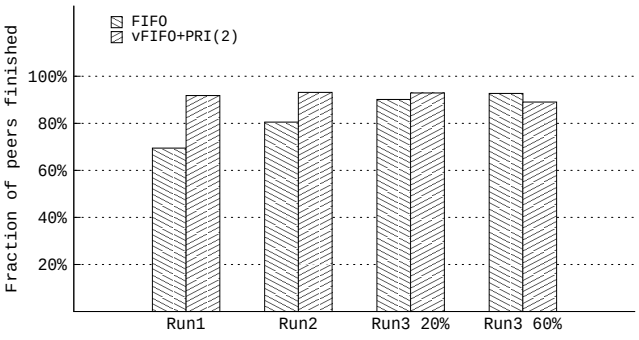


Figure 8: PlanetLab experiments: Fraction of peers that finish the download within 6 hours.

4.3 Experiment conclusions

The experiments above show that our approach reduce traffic between partitions by approximately 30% in most cases according to our PlanetLab experiment and by approximately 50% according to our simulations. The experiments also indicate that our approach does not increase BitTorrent download completion times.

5. Related Work

The increasingly higher volumes of peer-to-peer (P2P) traffic in the Internet has prompted the search for mechanisms to mitigate its cost on ISPs, both in the industry as well as in the academia. A concrete and effective approach towards reducing the cost of P2P traffic is caching, since it reduces the volume of traffic the ISP has to exchange [11, 18, 19]. In an early study, Gummadi et. al have shown that P2P traffic can be highly cacheable due to the skewed popularity of the content [11]. More recently, Saleh and Hefeeda proposed a specific caching algorithm for P2P traffic that was shown to be efficient even when cache sizes are relatively small [18]. Shen et. al have proposed a framework to leverage the existing web proxy cache infrastructure in order to cache P2P traffic and showed that this can significantly reduce network traffic [19]. Despite being fairly effective in reducing traffic, caching P2P content is not trivial and remains controversial. Implementing caching requires either changes to the P2P application (to become aware of the cache) or actively interfering with the P2P protocol (to serve download requests directly from the cache). Moreover, due to the vast amount of copyrighted content downloaded by P2P users, caching could raise delicate legal copyright violation issues.

Another concrete direction to reduce the amount of P2P traffic entering/leaving the ISP network (thus, reducing costs) is the exploitation of locality [1, 3, 11, 12]. In particular, Gummadi et. al showed that network locality could be exploited in P2P file sharing applications [11]. More recently, Karagiannis et. al have evaluated simple locality-aware solutions using real BitTorrent logs which have shown to significantly alleviate P2P traffic [12]. Bindal et. al propose an algorithm based on biased neighbor selection in order to explicitly exploit locality within the ISP [3]. Aggarwal et. al have proposed an oracle-based framework to assist P2P applications in the exploitation of locality [1], while Choffnes and Bustamante have shown how to exploit the Content Delivery Network (CDN) infrastructure to act as an oracle [5] for BitTorrent. Finally, Xie et. al have recently proposed an infrastructure to facilitate the interaction between ISPs and P2P applications [22]. Despite having shown to be effective, the proposals for exploiting locality or creating infrastructure within the ISP all require modifying the P2P application, something that is not under the ISP's control.

Some ISPs have decided to take active measures in order to reduce the amount of P2P traffic in their networks. Approaches like selective dropping and rate limiting are indeed being used by ready-to-install solutions [7]. However, such approaches also require identifying P2P applications within the aggregate traffic, which is non-trivial and application dependent. Finally, although effective, such approaches clearly degrade user-level performance, and are therefore avoided by some ISPs.

Finally, mathematical models have recently been proposed to better understand this problem [10, 21]. In particular, Garetto et. al propose an economic and performance-based framework to study the dilemma between ISPs and P2P file sharing users [10]. Their study establishes conditions under

which the ISP can operate a lucrative business when users run P2P applications.

6. Conclusion

In this paper we leverage on the observation that traffic costs to an ISP can be reduced by diverting traffic away from expensive links. We propose a novel class-based scheduler that should operate on the uplink of broadband subscribers of a given ISP. Our solution is innovative in the sense that it requires no changes to applications nor any application-dependent packet classification. The proposed scheduler can be readily deployed by an ISP that wishes to reduce its BitTorrent traffic costs. Through simulations and Internet experiments, we have shown that the proposed scheduler can significantly reduce BitTorrent traffic on expensive links, without degrading user-level performance.

We believe the approach proposed in this paper, namely using a class-based scheduler to differentiate traffic, is a powerful mechanism that can be more broadly applied to reduce traffic costs. This is the subject of our ongoing investigations.

7. References

[1] Vinay Aggarwal, Anja Feldmann, and Christian Scheideler. Can ISPs and P2P users cooperate for improved performance? *ACM SIGCOMM Computer Communication Review*, 37(3):29–40, 2007.

[2] Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan. Analyzing and improving a BitTorrent networks' performance mechanisms. In *Proceedings of the IEEE INFOCOM conference*, 2006.

[3] Ruchir Bindal, Pei Cao, William Chan, Jan Medved, George Suwala, Tony Bates, and Amy Zhang. Improving traffic locality in BitTorrent via biased neighbor selection. In *Proceedings of the 26th IEEE ICDCS*, page 66, 2006.

[4] Kenjiro Cho, Kensuke Fukuda, Hiroshi Esaki, and Akira Kato. The impact and implications of the growth in residential user-to-user traffic. In *Proc. of the ACM SIGCOMM*, 2006.

[5] David R. Choffnes and Fabin E. Bustamante. Taming the torrent: A practical approach to reducing cross-ISP traffic in P2P systems. In *Proc. of the ACM SIGCOMM*, 2008.

[6] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):00–00, July 2003.

[7] Sandvine co. Meeting the challenge of todays evasive P2P traffic. Technical report, Waterloo, Canada, 2004. An Industry Whitepaper.

[8] Marcel Dischinger, Andreas Haeberlen, P. Krishna Gummadi, and Stefan Saroiu. Characterizing residential broadband networks. In *ACM Internet Measurement Comference*, pages 43–56, 2007.

[9] Kolja Eger, Tobias Hoßfeld, Andreas Binzenhöfer, and Gerald Kunzmann. Efficient simulation of large-scale P2P networks: Packet-level vs. flow-level simulations. In *2nd Workshop on the Use of P2P, GRID and Agents for the Development of Content Networks (UPGRADE-CN'07) in conjunction with the 16th IEEE HPDC*, Monterey Bay, California, USA, June 2007.

[10] Michele Garetto, Daniel Figueiredo, Rossano Gaeta, and Matteo Sereno. A modeling framework to understand the tussle between ISPs and peer-to-peer file-sharing users. *Performance Evaluation*, 64(9–12):819–837, 2007.

[11] P.K. 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. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[12] Thomas Karagiannis, Pablo Rodriguez, and Konstantina Papagiannaki. Should internet service providers fear peer-assisted content distribution? In *ACM Internet Measurement Conference (IMC)*, pages 66–76. USENIX Association, 2005.

[13] Frank Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, January 1997.

[14] Kevin J. O'Brien. Who will pay as the internet grows? International Herald Tribune, Jun 2008. URL: `http://www.iht.com/articles/2008/06/08/technology/neutral09.php`.

[15] Andrew Odlyzko. Internet pricing and the history of communications. *Computer Networks*, 36(5-6):493–517, August 2001.

[16] Michael Piatek, Tomas Isdal, Thomas E. Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent? In *NSDI*, 2007.

[17] Louis Plissonneau, Jean-Laurent Costeux, and Patrick Brown. Analysis of peer-to-peer traffic on adsl. In *Passive and Active Network Measurement Workshop*, pages 69–82, 2005.

[18] O. Saleh and M. Hefeeda. Modeling and caching of peer-to-peer traffic. In *IEEE International Conference on Network Protocols (ICNP)*, 2006.

[19] Guobin Shen, Ye Wang, Yongqiang Xiong, Ben Y. Zhao, and Zhi-Li Zhang. HPTP: Relieving the tension between ISPs and P2P. In *International workshop on Peer-To-Peer Systems (IPTPS)*, 2007.

[20] Brad Stone. Comcast adjusts way it manages internet traffic. The New York Times, Mar 2008. URL: `http://www.nytimes.com/2008/03/28/technology/28comcast.html`.

[21] J.H. Wang, D.M. Chiu, and J.C. Lui. Modeling the peering and routing tussle between ISPs and P2P applications. In *IEEE International Workshop on Quality of Service (IWQoS)*, pages 51–59, 2006.

[22] Haiyong Xie, Y. Richard Yang, Arvind Krishnamurthy, Yanbin Liu, and Abraham Silberschatz. P4P: provider portal for applications. In *Proc. of the ACM SIGCOMM*, 2008.

[23] `http://www.isi.edu/nsnam/ns/`.

[24] `http://azureus.sourceforge.net/`.

[25] `http://www.bittornado.com/`.

APPENDIX

A simple partitioning model

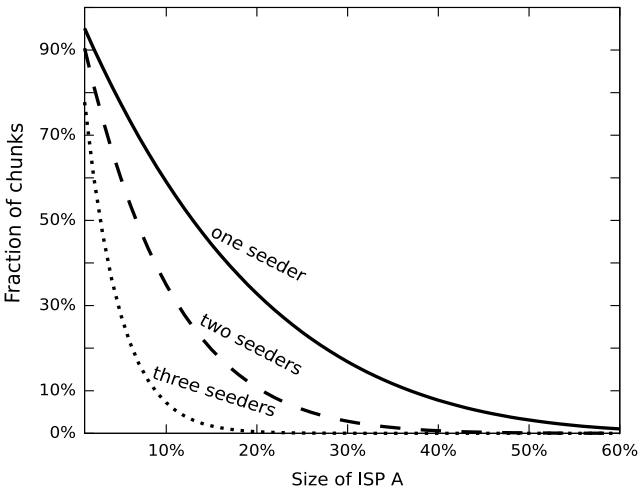


Figure 9: Fraction of blocks not present in any peer in a given partition as a function of the fraction of peers in the partition.

Consider a swarm (i.e., a set of peers interested in disseminating and retrieving a file) formed by a few seeders (i.e., peers that have the entire file and behave like servers for some time) and a large number of leechers (i.e., peers that are downloading the file) that join the system almost at the same time (i.e., a flash crowd). Let s be the number of seeders in the swarm and k be the number of times that each block is served by a seeder. We assume that once a seeder has served each block k times, it leaves the system. Moreover, we also assume that seeders serve blocks uniformly at random (i.e., the probability that a particular block is served by a particular seeder to a particular leecher is the same for all blocks, seeders and leechers). Let m be the total number of blocks that form the data. Finally, assume the set of leechers is partitioned and let α be the fraction of leechers that are in partition A. Let p denote the probability that no leecher in partition A receives a copy of a particular block from the seeders. Thus, we have $p = (1-\alpha)^{sk}$, since sk copies of a block are disseminated by the seeders. Finally, let X denote the number of blocks that are not present in partition A. Thus, X follows a binomial distribution with parameters m and p, and is given by

$$P[X = j] = \binom{m}{j} p^j (1-p)^{m-j} \ , \quad j = 0, 1, \ldots, m.$$

Note that the expected number of blocks not present in partition A is given by $E[X] = m(1-\alpha)^{sk}$, which may not be small when m (# of blocks) is large and s (# of seeders) is small. Finally, we can calculate the expected fraction of blocks that are not present in partition A, which is given by

$$f_b = \frac{E[X]}{m} = (1-\alpha)^{sk}.$$

Note that this fraction does not depend on the number of blocks m. Figure 9 illustrates this fraction as a function of α for various values of s, assuming $k = 5$ (a commonly used value in BitTorrent-like applications). Notice that f_b strongly depends on the number of initial seeders. However, for cases where α is not too large, a significant fraction of the blocks cannot be found within the partition after all initial seeders have departed from the system. This analysis indicates that the partition problem can indeed occur. We further investigate this issue by evaluating the following scenario using a BitTorrent implementation [9] for the NS-2 simulator [23].