Really Truly Trackerless BitTorrent

Charles P. Fry[1] **Michael K. Reiter**[2]

August 2006
CMU-CS-06-148

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]Electrical & Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA; `cfry@ece.cmu.edu`
[2]Electrical & Computer Engineering Department, and Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA; `reiter@cmu.edu`

This work was partially supported by NSF award number CCF-0424422

Keywords: peer-to-peer systems, expander graphs, random walks

Abstract

BitTorrent is a peer-to-peer protocol used for collaborative downloading. It allows peers to exchange blocks of the file they are downloading with each other, rather then obtaining them from a central server. Despite the decentralized nature of the protocol, BitTorrent has traditionally relied on a centralized tracker, which bootstraps the system by providing new clients with a random set of peers. The tracker has previously been perceived as a critical part of BitTorrent systems, and efforts to make it more fault-tolerant have focused on tracker replication and on tracker implementations using Distributed Hash Tables, which have been entitled "Trackerless BitTorrent". The tracker's sole responsibility, whether implemented in a centralized or distributed manner, is to allow peers to randomly select other peers to connect to, in an effort to construct a robust graph. We propose completely removing the tracker and replacing it with a set of distributed protocols based on random walks, accomplishing the work of the tracker without explicitly tracking every peer in the system. We use expansion as a measure by which to compare the quality of the graphs generated by distributed, trackerless algorithms to the graphs generated by a centralized tracker. We also explore the implications of various design decisions related to the random walks which are the key component of our proposal.

1 Introduction

BitTorrent is a distributed protocol by which multiple concurrent clients (*peers*) can download the same file by sharing blocks of that file with each other rather than obtaining it from a central location. In order to bootstrap the protocol, a new client must discover other peers who are currently downloading the file of interest. To accomplish this, a centralized *tracker* explicitly tracks all peers who are currently downloading a file. When contacted by a new client, the tracker adds that client's address to the list of current downloaders, collectively known as a *swarm*, and selects a random set of peers from the swarm to provide to the joining client. The client then contacts these peers, attempting to establish connections with them. Clients that are connected together are *neighbors*, and as long as they remain connected they regularly attempt to exchange blocks of the file they are downloading with each other.

Unfortunately, the distributed network formed by BitTorrent peers is far more robust and scalable than the centralized tracker. If a tracker goes offline, the remaining clients are able to continue exchanging blocks of the file being downloaded only until they lose their current neighbors, and it is impossible for new peers to join the swarm. To remedy this, we propose the removal of the tracker, replacing its functionality with a distributed algorithm based on random walks through the swarm. In our approach, no party keeps track of all members of the swarm, and peers need not notify others of their departure or continued presence. Rather, each peer simply maintains connections to a bounded set of neighbors, which it must already do for file block exchange. Each peer's neighbors are selected by walking randomly in the existing swarm, and can be refreshed similarly (e.g., if a peer loses neighbors due to departures or failures). These walks are conducted either by *entry points* or by joining nodes starting from peers provided by entry points, though we stress that an entry point does not track the swarm membership more than any other peer does. In fact, *any* peer can act as an entry point—possibly simultaneously with others, and possibly transiently—with no further information than it already has by participating in the swarm as a peer.

Within this general framework, we evaluate several different design alternatives and their impact on the swarm, with particular attention to how the random walk algorithm affects the swarm quality. The measure of "quality" that we employ is *vertex expansion*, i.e., the minimum ratio between the number of peers neighboring a set and the size of that set, for all sets up to a certain size, such as half of the swarm. Expansion, and the low diameter that it implies [10], have long been recognized as important properties in peer-to-peer networks [14, 17, 21, 23]. High vertex expansion implies that a network is robust, in that a partition is highly improbable [4]; it ensures that there are a relatively large number of communication paths leaving each set of nodes in the network, maximizing the aggregate bandwidth of the network; and it also allows rare blocks to rapidly expand through the swarm, minimizing the risk that they disappear from the network.

We conduct our evaluations using trace-driven simulations. The traces we utilize are a well-known 5-month RedHat tracker log [20] and a more recent 5-month log of the Debian tracker. Using these traces, we demonstrate several random walk algorithms that yield similar, and in some cases *better*, expansion than a centralized tracker, for the same degree constraints imposed in the default BitTorrent configuration. This suggests that while our approach has merit in eliminating fragility due to the use of a tracker, it also does so without impairing block exchange.

2 Related Work

Here we place our contributions in the context of related work on BitTorrent specifically (Section 2.1) and on the distributed construction of random graphs more generally (Section 2.2).

2.1 BitTorrent

The traditional centralized tracker has long been recognized as a single point of failure in BitTorrent systems. The first efforts to remove this single point of failure involved instantiating multiple tracker replicas, and defining the order in which they should be accessed [19]. A more successful strategy, at least in terms of client adoption, for removing the dependency on a single centralized tracker involved support for distributed trackers. While the concept was originally pioneered by unofficial clients, BitTorrent recently added official support for a distributed tracker, naming the resultant system “Trackerless BitTorrent” [12]. Under this model, each client becomes a “lightweight tracker,” using the Distributed Hash Table (*DHT*) Kademlia [26] to store the identities of all peers in the torrent.

While storing tracker information in a DHT removes the single point of failure present with a centralized tracker, the construction and maintenance of the DHT requires that in addition to the neighbors with which it communicates as part of the BitTorrent protocol, each peer must also maintain an orthogonal set of neighbors within the DHT, and pay the communication cost of maintaining the DHT in the face of high rates of churn [22]. This begs the question of whether the distributed tracker could be combined with the existing BitTorrent neighbor infrastructure, and even whether a tracker of any kind, whether centralized or distributed, is really necessary at all. Here we show that it is not.

While not supported by the official client, Peer Exchange [29] is part of many BitTorrent implementations. Introduced as a mechanism for reducing load on the tracker, Peer Exchange is a gossip protocol by which peers tell each other about other peers in the swarm. Although Peer Exchange fails to entirely eliminate the tracker, it does demonstrate that peers can viably discover new peers by sharing peer information with each other, rather than being completely dependent on the tracker.

2.2 Random graphs

The algorithmic techniques we employ here derive from work in the distributed construction of random graphs, the vast majority of which seeks to build random *regular* graphs; in particular, a random regular graph is a good expander with high probability [15]. Bourassa and Holt [8] proposed a protocol for the distributed construction of random regular graphs, for which a bound on mixing time (see Section 3) was proved by Cooper et al. [13]. However, when nodes depart ungracefully a broadcast must be made over the entire graph, seeking other nodes which are also under-connected; we avoid such large-scale broadcasts. Law and Siu [21] similarly construct random regular graphs, though at the cost of maintaining Hamilton cycles over the graph; this approach was subsequently optimized by Gkantsidis et al. [17] using insights due to Gillman [16]. Unfortunately the Hamilton cycle constructions are unable to deal with large numbers of nodes leaving the network, requiring periodic regeneration of the graph. Other works construct random regular graphs from other graphs through graph transformations, e.g., [25, 30]. While introducing new randomness via these approaches could benefit BitTorrent graphs, we focus on the quality of the graphs which are initially constructed.

While random regular graphs possess desirable properties with high probability, such as low diameter and high expansion, the BitTorrent tracker does not construct regular graphs, but rather bounded degree graphs. We follow this goal to more closely mimic the behavior of the tracker, and because irregular graph constructions tend to be simpler and less intrusive when new connections are added, as it is never necessary to destroy existing connections. They also are more tolerant of node departures, as they do not attempt to maintain the same number of neighbors at each node.

Within this space, Pandurangan et al. [28] proposed a partially distributed, bounded degree graph construction protocol which relies on a centralized cache of a constant number of nodes. While this model is directly applicable to the BitTorrent tracker, it does require a central server, and is thus insufficient for the trackerless case. Cooper et al. [14] propose an algorithm where each peer contributes cm tokens to the graph, which circulate in random walks until they are picked up, m at a time, by new nodes joining the network; the new node connects to the m peers that contributed the tokens it picked. The resultant graph is shown to maintain diameter logarithmic in the total number of nodes in the graph, and to be robust against the adversarial deletion of both edges and vertices. One disadvantage of this protocol is that the tokens in the graph must be constantly circulated in order to ensure that they are well-mixed. Moreover, the rate at which new nodes can join the system is limited, as they must wait while the existing tokens mix before they can use them, a potential problem in the face of flash crowds which BitTorrent can otherwise handle so gracefully [6].

Vishnumurthy and Francis [31] evaluate various types of random walks for use in random neighbor selection in the construction of peer-to-peer networks according to a desired degree distribution. Unbiased walks are unfit for constructing graphs with fixed degree distributions, and so Vishnumurthy and Francis explore various biased halting and biased forwarding heuristics through simulation. While BitTorrent graphs are irregular, they do not require a prescribed degree distribution, but rather impose a lower and upper bound on degree.

Arthur and Panigrahy [2] define a generative random graph model of BitTorrent graphs which they use to model and analyze block distribution. Their model, however, is incomplete in that it doesn't account for the neighbor replacement which may occur when nodes leave the graph. They also propose a slight modification to BitTorrent graph construction where initial neighbor selection is biased towards newer nodes, which ends up improving block exchange.

3 Preliminaries

In this section we introduce key concepts that underlie the remainder of our discussion. We represent a BitTorrent swarm at any point in time as a simple graph $G = (V, E)$ where $V = \{1, \ldots, n\}$ is the set of peers and $E \subseteq V \times V$ is the set of neighbor relations, i.e., $(i, j) \in E$ iff i and j are connected to one another. Note that the neighbor relation is symmetric. Let $\Gamma(i) = \{j \in V : (i, j) \in E\}$ be the neighbors of $i \in V$, and let $\Gamma(S) = \bigcup_{j \in S} \Gamma(j)$. Let $\mathsf{deg}_i = |\Gamma(i)|$.

As discussed in Section 1, our primary measure of quality of this graph is its *vertex expansion*, i.e.,

$$\min_{1 \le |S| \le n/2} \frac{|\Gamma(S)|}{|S|}$$

While measuring the expansion of a graph is co-NP-complete [7], there is a close relationship between a graph's expansion and the second smallest eigenvalue λ of the graph's Laplacian matrix. More specifically, a graph of maximum degree mdeg has vertex expansion of at least $\frac{2\lambda}{2\lambda+\mathsf{mdeg}}$ [1]. Whenever we claim to compute the expansion of a graph, it is this lower bound that we report.

A random walk algorithm is defined by an $n \times n$ transition probability matrix P where each row $\mathsf{P}_i \in [0,1]^n$ is a distribution (i.e., its components sum to one) and $\mathsf{P}_{ij} > 0$ only if $j \in \Gamma(i) \cup \{i\}$. A *step* of a random walk currently at node i samples a node j according to the distribution P_i and transitions to node j, making it the new current node. A random walk is then the sequence of nodes visited by repeating such steps, starting from a node chosen according to an initial distribution $\pi_0 \in [0,1]^n$. A *stationary distribution* for the random walk algorithm is a distribution $\pi \in [0,1]^n$ that satisfies $\pi = \pi\mathsf{P}$. For the random walk algorithms we consider here, the stationary distribution is unique (if it exists) with high probability (where the probability is taken over the selection of the graph G induced by our algorithms), and so we treat it as such here. When π exists, the *mixing time* is the number t of steps needed to "reach" the distribution π or, informally, the minimum t such that $\pi \approx \pi_0\mathsf{P}^t$ for all initial distributions π_0. For some of the random walk algorithms that we consider here, the mixing time is known to be $t = O(\log n)$. However, since the random walk algorithms that we study here do not utilize t as a parameter, the fact that the mixing time is not established for a particular algorithm does not preclude its use.

4 Trackerless BitTorrent

We propose a modification to the BitTorrent protocol which removes its dependence on the tracker. The tracker's sole responsibility is to provide peers with neighbors randomly selected from the entire swarm. This currently happens both when a node first joins the swarm, and thereafter whenever its neighbor count falls below a certain threshold, $\mathsf{minNeighbors}$. The tracker accomplishes its job by maintaining a global list of all peers currently in the swarm, and then selecting random sets of $\mathsf{sampleSize}$ peers for distribution as potential neighbor sets. Each peer will initiate connections to up to $\mathsf{maxInitiate}$ other peers, and then cache any extra peers that the tracker told it about. Peers accept connections from other peers as long as they have less than $\mathsf{maxNeighbors}$ neighbors. In the current official BitTorrent implementation, the values of these configuration variables are $\mathsf{minNeighbors} = 20$, $\mathsf{maxInitiate} = 40$, $\mathsf{sampleSize} = 50$, and $\mathsf{maxNeighbors} = 80$.

While maintaining a global list of peers allows for rapid random selection among them, it is by no means the only mechanism by which random nodes can be selected. An appealing alternative is the use of random walks, which have long been used to randomly select nodes from graphs [24]. Simply performing an unbiased random walk of sufficient length (the mixing time) and selecting the last node in the walk will select each node in the graph with probability proportional to its degree. In order to select each node from an irregular graph with uniform probability, bias can be added to the direction (biased forwarding) or length (biased halting) of the random walk [3, 31].

Such biased random walks could be used in BitTorrent both to select initial neighbors for joining nodes and to replace failed neighbors, removing any dependence on the tracker.

Most of the random walk algorithms that we consider require the walking node i to randomly select an element of $\Gamma(j)$ where $j \in \Gamma(i)$ according to some distribution. This could be implemented either by i querying j to obtain $\Gamma(j)$ and then choosing from $\Gamma(j)$ itself, or by asking j to select an element from $\Gamma(j)$ according to the proper distribution and return that element to i. If i must be able to select from $\Gamma(j)$ even after j has failed (e.g., to replace j by a neighbor of j), then it should obtain $\Gamma(j)$ upon connecting to j and ask that j keep it notified of any changes.

4.1 Entry Points

Although BitTorrent can function properly without a tracker of any kind, it is still necessary for new nodes to somehow discover other nodes in the swarm. We accomplish this through the use of entry points that help joining nodes to obtain an initial random set of neighbors. In purpose, entry points serve the same role as the tracker; the key distinction is in how they perform this role. Both centralized and decentralized trackers, as their name implies, explicitly keep track of every node currently in the swarm. Each entry point, on the other hand, is only aware of its neighbors in the swarm. Each node in the swarm can act as an entry point, helping joining nodes to discover new neighbors. While entry points are not strictly required to be members of the swarm, swarm membership does have the advantage of helping entry points remain connected to the swarm.

Because entry points can be implemented in a manner that strongly resembles a tracker, we emphasize some of the key differences between the two. The tracker holds a distinguished role which cannot be arbitrarily shared; even DHT tracker implementations rely on a small number of nodes to store the membership of the swarm. Our entry points, on the other hand, can be any arbitrary nodes inside or outside the swarm. No communication is required between multiple entry points; they can all operate independently and in parallel. Finally, entry points proactively explore the swarm and thus do not need to be notified of join and leave events, nor do nodes need to regularly announce their presence as required by the tracker.

4.2 Initial Neighbor Selection

There are various ways in which random walks can be used by entry points in order to provide joining nodes with an initial random neighbor set. The most primitive strategy would be for either the entry point or the joining node to perform a well-mixing random walk in order to select each new neighbor. Regardless of who performs the walks, this solution has the disadvantage of requiring that the walk be advanced by at least the mixing time (see Section 3), so that the last node of the walk is selected from the desired stationary distribution. Having entry points perform these walks allows new nodes to more rapidly join the swarm, as long as the join rate through any given entry point is not too high. The advantage of requiring newly joining nodes to perform their own random walks is that it reduces the amount of work that entry points are required to perform for each node that joins the swarm.

It is not actually necessary for the walk to be newly advanced by its full mixing time for each join. By applying the technique of Gillman [16], first introduced in a peer-to-peer context by

Gkantsidis et al. [17], entry points can maintain sampleSize perpetual random walks which are extended by only c steps for each selection of a random node.[1]

The use of perpetual random walks requires that the entry points themselves extend the random walks in order to avoid repeatedly distributing the same set of neighbors to every joining node. While it is not strictly required, we recommend that entry points perform their perpetual random walks on their own neighbor set as this removes the need to maintain a separate set of nodes for block exchange and for random walks. This also has the beneficial side-effect of automatically refreshing entry point's neighbors, ensuring that they are randomly spread throughout the entire graph. As a practical measure, entry points should not refresh their neighbors with whom they are actively exchanging blocks; this is not difficult to avoid as peers can have up to maxNeighbors neighbors, but only need perform sampleSize random walks.

4.3 Failed Neighbor Replacement

BitTorrent clients have traditionally relied on the tracker to provide additional randomly selected nodes whenever their neighbor count drops below minNeighbors (or even maxInitiate, though with less urgency). In a truly trackerless system peers cannot necessarily return to their original entry point to request additional neighbors, as entry points may not necessarily remain in the swarm during the entire lifetime of the peers which they help to enter the swarm. However, since any node can serve as an entry point, a peer in need of additional neighbors could acquire them from any other node in the swarm.

5 Trace-Driven Simulations

We compare graphs generated by our approach to those created by the centralized tracker through trace-driven simulation using actual logs generated by two distinct trackers, both covering periods of around five months. Using the tracker logs we determine when nodes joined and left the swarm, and use this information to drive our simulator, which connects and disconnects nodes according to the various algorithms which we examine. The resultant connectivity graph is periodically analyzed in order to compare the quality of the graphs generated by different algorithms.

The first log which we analyze is the RedHat tracker log first examined by Izal et al. [20], covering a period from April to August of 2003. This trace began with a flash crowd which rapidly grew to over 4000 simultaneous clients during the first 16 hours, and then dropped almost as rapidly over the next few days, ultimately stabilizing at around 100 simultaneous clients within two weeks. The second log which we analyze is from the current Debian tracker, covering a period from December 2005 to May 2006. As no new Debian release was made during that time, the number of simultaneous clients remains fairly constant, fluctuating around 2000.

We begin our tracker log analysis by extracting client arrival and departure times. More specifically, we determine from the log when the tracker learned of a new client joining the swarm, and

[1]Gkantsidis et al. [17] let $c = 1$ in the case of regular graphs, and similar results would be expected with biased forwarding on irregular graphs. The case of biased halting would obviously require larger values of c.

when the tracker assumed that a client had left. Under normal modes of operation, clients explicitly notify the tracker of their arrival and departure, however in practice we found that this was not always the case. When the log indicated interaction from a client which was not currently known to be in the swarm, this was recorded as a join, whether the client request claimed to be a join or a periodic announcement.

Determining when nodes left the swarm is more involved if they do not exit cleanly by notifying the tracker of their departure. Trackers tell clients how often they are expected to announce themselves, and if clients go for too long without doing so, then they are assumed to have left the swarm. In the official BitTorrent implementation, clients are asked to announce themselves every 30 minutes, and are removed from the tracker's list of swarm members if they are not heard from for 45 minutes. We thus recorded a client's unclean departure as taking place 45 minutes after its last interaction with the tracker, in cases where it did not exit cleanly. In practice, the node likely left at an earlier time, and the tracker incorrectly assumed that it was still part of the swarm. For the sake of our simulation, however, we pretend that nodes remained in the swarm until the moment when the tracker would have assumed they had left. In doing so we err in favor of the centralized tracker, who otherwise would distribute stale nodes as potential neighbors. Because random walks actively explore the swarm, they are far less likely to falsely believe that a departed node is still in the swarm.

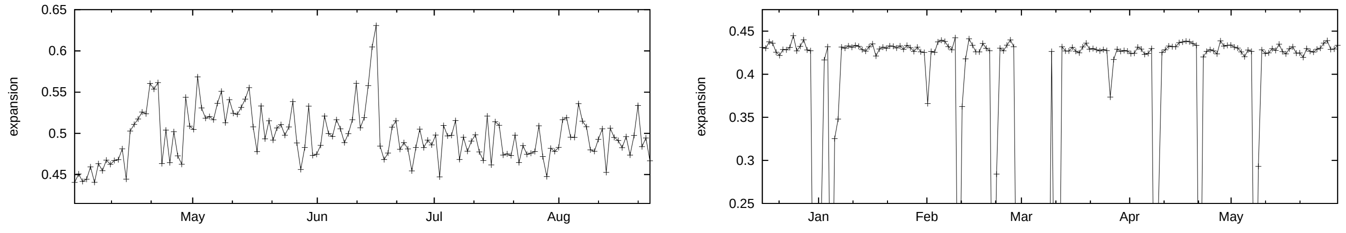
When using a tracker, peers request new neighbors from the tracker every five minutes if they have less than minNeighbors neighbors, and every half hour if they have less than maxInitiate neighbors. While such long waits between obtaining new neighbors is entirely unnecessary when using entry points, we again err in favor of the centralized tracker, imposing these delays between new neighbor requests across all algorithms that we study.

Just as the tracker can distribute nodes that are not alive, so can it suggest neighbors that are already full, and thus will be unwilling to accept new connections. The tracker accommodates this error by handing out groups of sampleSize neighbors even though only maxInitiate will ever be used at one time. While entry points could (and probably should) avoid distributing full nodes, our simulation currently selects the node that is walked to regardless of its degree, again erring in favor of the centralized tracker.

With the arrival and departure times of clients from actual tracker logs we are able to simulate the dynamic construction and evolution of the BitTorrent swarm under various algorithms. While the simulator has a notion of the times at which nodes join and leave, it completely processes each join and leave event, including the addition and removal of all associated neighbor connections, serially in the order in which they occur in the logs. Following the precedent of Vishnumurthy and Francis, we justify this because the random nature of the algorithms we consider make them impervious to the order and timing of events [31].

We thus process the tracker logs for each algorithm evaluated, recording the vertex expansion of a snapshot of the swarm taken at regular intervals. In the case of the tracker, whether centralized or distributed, new neighbors are randomly selected from a global swarm membership list. Entry points, on the other hand, use random walks to dynamically explore the swarm, randomly selecting peers encountered on their walks.

In order to minimize the number of random walk steps taken for each joining node, our entry



(a) RedHat (b) Debian

Figure 1: Centralized Tracker

points perform sampleSize perpetual random walks, each of which is extended by a single step to select new neighbors for every joining node. As a result, the neighbor sets of two successive nodes i and j that join through the same entry point will be connected, i.e., $\Gamma(j) \subseteq \Gamma(\Gamma(i))$. While this relationship between successive random walk samples is not expected to be problematic [16, 17], we perform our simulations with all nodes joining through a single entry point to ensure that we observe any negative impact this correlation could have on expansion. For the same reason, whenever a node needs additional neighbors it requests them from the single entry point.

5.1 Centralized Tracker

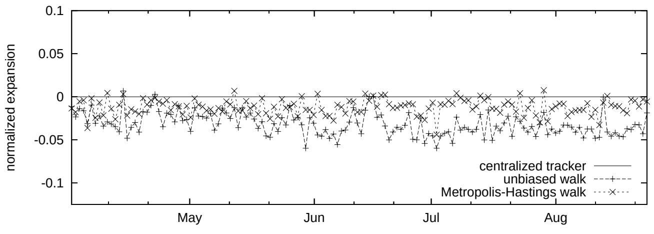
As we are evaluating the feasibility of replacing the centralized tracker, we begin by measuring the expansion over time of graphs generated by the centralized tracker. This will serve as a baseline to which other algorithms can be compared. Figure 1 shows the expansion of graph snapshots taken once per day over the lifetime of both the RedHat and Debian logs, when processed by our centralized tracker simulator.

As illustrated in Figure 1(a), the expansion of the RedHat torrent primarily oscillates between 0.45 and 0.55. During the rise and fall of the initial flash crowd, expansion hovers at the low end of this spectrum, rapidly rising as the total number of nodes decreases to 100.

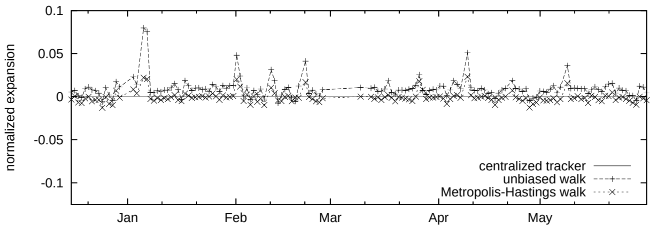
The steady-state expansion of the Debian torrent is far more regular, primarily varying between 0.43 and 0.44 as seen in Figure 1(b). The occasional periods of zero expansion reflect gaps in the tracker logs, which could be due either to tracker down time or to errors archiving the logs. Regardless of the original cause, our simulator deals with gaps by allowing all nodes in the swarm to expire, and then rebuilds the swarm graph from scratch once the log resumes. This is obviously pessimistic, as it is likely that even if the tracker was temporarily down that the swarm still stayed together and did not need to be reconstituted from scratch. However, it does create artificial flash crowds which allow us to observe the effect they have on the algorithms that we study.

5.2 Simple Random Walks

With this understanding of the expansion of the graphs generated by the centralized tracker, we now turn to an examination of the expansion of graphs generated by various random walk algorithms that could be employed by entry points. The first type of random walk which we examine uses the



(a) RedHat



(b) Debian

Figure 2: Normalized Simple Random Walks

classical Metropolis-Hastings algorithm [27, 18] which allows approximately uniform sampling from an irregular graph [9, 3], thus closely mimicking the behavior of the centralized tracker. In order to determine the advantage obtained by the bias of the Metropolis-Hastings walk, we also evaluate the performance of unbiased walks, that as a result select nodes with probability proportional to their degree.

The transition probabilities (see Section 3) for the unbiased random walk are:

$$\mathsf{P}^{\mathsf{unb}}_{ij} = \begin{cases} 1/\mathsf{deg}_i & \text{if } j \in \Gamma(i) \\ 0 & \text{otherwise} \end{cases}$$

The transition probabilities for the Metropolis-Hastings random walk are:

$$\mathsf{P}^{\mathsf{mh}}_{ij} = \begin{cases} \frac{1}{\max\{\mathsf{deg}_i, \mathsf{deg}_j\}} & \text{if } j \in \Gamma(i) \\ 1 - \sum_{k \in \Gamma(i)} \mathsf{P}^{\mathsf{mh}}_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Figure 2 shows the expansion from both the Metropolis-Hastings random walks and the unbiased random walks normalized against the expansion of the centralized tracker in order to underscore the relationship between these algorithms and the centralized tracker. In the case of the RedHat torrent, shown in Figure 2(a), both random walk algorithms perform slightly worse than the centralized tracker, with Metropolis-Hastings performing better than unbiased random walks. The Metropolis-Hastings algorithm is just a hair below the centralized tracker in the Debian torrent, shown in Figure 2(b), while the unbiased random walks outperform the centralized tracker at almost every point in time.

It is somewhat surprising that such good expansion is obtained when random nodes are selected by the use of unbiased random walks, as they select nodes with probability proportional to their degree. It is thus interesting to observe that while the uniform random sampling of the centralized tracker and the approximately uniform random sampling of the Metropolis-Hastings random walks both create graphs with good expansion properties, such uniformity in sampling is not a prerequisite for obtaining such results.

5.3 Degree-Biased Random Walks

The success of the unbiased random walks suggests the possibility that other types of walk bias which sample peers non-uniformly may also be reasonable candidates for replacing the centralized tracker. In their exploration of biased random walks, Vishnumurthy and Francis proposed walks where steps were taken with probability proportional to a node's outdegree (the number of connections that it initiated) and inversely proportional to a node's degree [31]. This is an appealing prospect, as it allows priority to be given to low degree nodes when establishing new neighbor connections. We also note that Cooper et al.'s token-based protocol circulates one token for each connection a node is willing to accept ($\mathsf{maxNeighbors}$), resulting in neighbor selections which are biased according to the residual degree of each node i, i.e., $\mathsf{maxNeighbors} - \mathsf{deg}_i$ [13]. Finally, Arthur and Panigrahy[2] bias new neighbor selection towards younger nodes to improve block exchange.

Our initial evaluation of degree-biased random walks selects neighbors to walk to with probability proportional to their residual degree, and then with probability inversely proportional to their degree. Note that as we don't distinguish between inbound and outbound connections, our inverse degree bias is slightly different than that of Vishnumurthy and Francis. We do, however, follow their model of applying the bias locally to each random walk step, which does not necessarily result in the global selection of nodes with probability exactly proportional to residual or inverse degree.

We define the residual degree for a node i as $\mathsf{rdeg}_i = \mathsf{maxNeighbors} - \mathsf{deg}_i$. The transition probabilities for the residual degree random walk are:

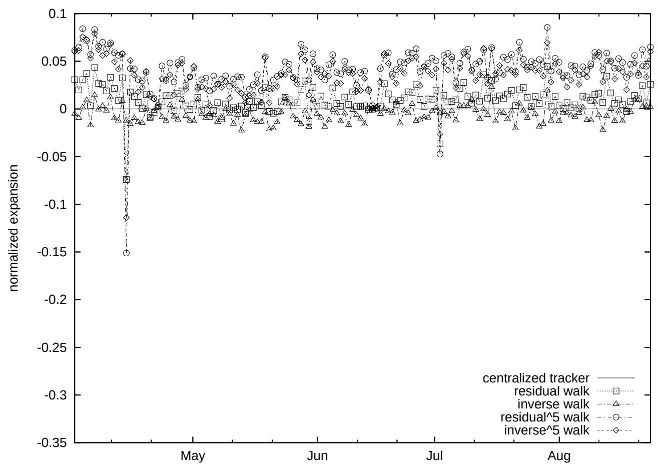
$$\mathsf{P}_{ij}^{\mathsf{res}} = \begin{cases} \frac{\mathsf{rdeg}_j}{\sum_{k \in \Gamma(i)} \mathsf{rdeg}_k} & \text{if } j \in \Gamma(i) \\ 0 & \text{otherwise} \end{cases}$$

The transition probabilities for the inverse degree random walk are:

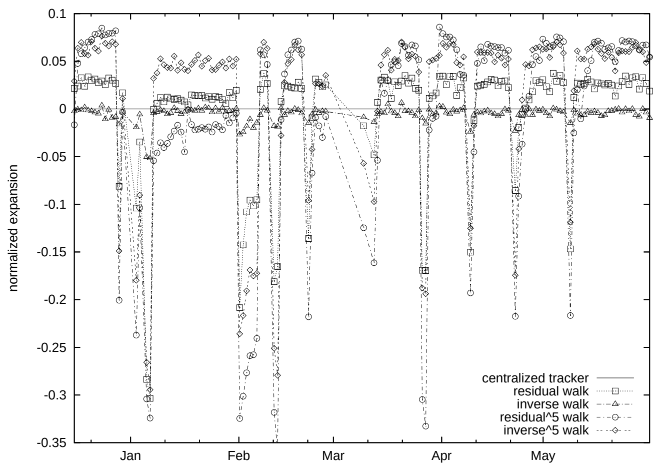
$$\mathsf{P}_{ij}^{\mathsf{inv}} = \begin{cases} \frac{\mathsf{minNeighbors}/\mathsf{deg}_j}{\sum_{k \in \Gamma(i)} \mathsf{minNeighbors}/\mathsf{deg}_k} & \text{if } j \in \Gamma(i) \\ 0 & \text{otherwise} \end{cases}$$

Figure 3 shows the normalized expansion which results from these walks (as well as some others which shall be discussed shortly). As seen in Figure 3(a), on the RedHat torrent the residual walk performs slightly better than the centralized tracker, and the inverse walk, slightly worse. The inverse walk is also subpar to the centralized tracker on the Debian torrent, shown in Figure 3(b), but in this case the residual walk yields expansion significantly higher than the centralized tracker most of the time.

There are, however, occasional drops in the normalized expansion for residual walks on the Debian torrent that were not seen for other algorithms. In fact, these drops all occur immediately after periods of zero expansion where there were no nodes in the swarm. In other words, the residual degree walk is slow to grow graphs with good expansion from scratch. This is best seen in Figure 4, which zooms in on the first two and a half days of the Debian torrent, showing non-normalized expansion for all algorithms measured hourly instead of daily. The residual walks initially create graphs with relatively low expansion, with expansion rising to its steady-state levels



(a) RedHat



(b) Debian

Figure 3: Normalized Degree-Biased Random Walks

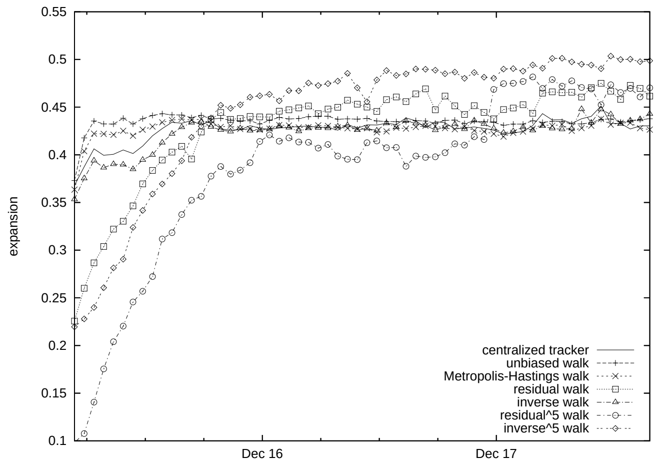


Figure 4: Debian Startup

over a slightly longer period of time than the centralized tracker. The likely reason for this behavior is that the older a node is, the more chances it has of being selected by a random walk, even if the walk is biased against high degree nodes. As a result, old nodes tend to have higher degree than new nodes, and degree-biased walks are more likely to select younger nodes. Vishnumurthy and Francis made a similar observation about their biased-forwarding random walks, and suggested periodically refreshing neighbors in order to introduce fresh randomness. In fact, they note that churn has the same impact on the graph as refreshing, which could partially explain why the low expansion initially experienced by some of our algorithms improves dramatically with the passage of time [31].

The success of the residual walk led us to experiment with other variations of degree-biased walks. We started by making the bias proportional to the square of the residual degree and the inverse degree, respectively. As this improved expansion over the simple residual and inverse degree, we also evaluated raising the residual and inverse degrees to powers greater than two. Due to space restrictions we only report on the results of power five, which we denote by $\mathsf{residual}^5$ and $\mathsf{inverse}^5$, as they generally tended to outperform the other powers.

The transition probabilities for the $\mathsf{residual}^5$ random walk are:

$$\mathsf{P}_{ij}^{\mathsf{res}^5} = \begin{cases} \frac{(\mathsf{rdeg}_j)^5}{\sum_{k \in \Gamma(i)} (\mathsf{rdeg}_k)^5} & \text{if } j \in \Gamma(i) \\ 0 & \text{otherwise} \end{cases}$$

The transition probabilities for the $\mathsf{inverse}^5$ random walk are:

$$\mathsf{P}_{ij}^{\mathsf{inv}^5} = \begin{cases} \frac{(\mathsf{minNeighbors}/\mathsf{deg}_j)^5}{\sum_{k \in \Gamma(i)} (\mathsf{minNeighbors}/\mathsf{deg}_k)^5} & \text{if } j \in \Gamma(i) \\ 0 & \text{otherwise} \end{cases}$$

These results are shown in Figure 3 and Figure 4 next to the results of the original (power one) residual and inverse walks. Both significantly outperform any other algorithm yet examined during most periods of stability, but take even longer to reach their steady-state behavior.

There is one anomaly in the expansion of graphs generated with $\mathsf{residual}^5$, which is best seen in Figure 4. While all of the other algorithms rise continually and then level off at their steady-state, $\mathsf{residual}^5$ rises (albeit slower than any other algorithm), levels off for an extended period of time at expansion levels inferior to the centralized tracker, and then unexpectedly jumps to levels significantly higher than the centralized tracker, on par with the $\mathsf{inverse}^5$ algorithm. At least that is what happens most of the time. However, during the month of January in the Debian trace, the $\mathsf{residual}^5$ algorithm never made the second jump, and remained at the lower steady-state until the next period of zero expansion. We leave the analysis of this anomaly to future study.

We hypothesize that the reason for which most of the degree-biased random walks yield higher expansion during their steady-state than other algorithms is because they more closely approach random regular graphs, which are known to have good expansion properties with high probability [15]. This hypothesis is supported by the fact that the maximum degree of graphs generated across all algorithms is strongly negatively correlated to expansion, with a correlation coefficient of -0.93 for graphs generated across all algorithms with the RedHat trace, and -0.73 for the Debian trace.

algorithm	median expansion	mean expansion	std dev expansion
residual^5 walk	0.5411	0.5400	0.0310
inverse^5 walk	0.5337	0.5334	0.0290
residual walk	0.5067	0.5077	0.0298
centralized tracker	0.4956	0.4977	0.0326
inverse walk	0.4884	0.4926	0.0323
Metropolis-Hastings walk	0.4846	0.4861	0.0345
unbiased walk	0.4558	0.4659	0.0357

Table 1: RedHat Expansion

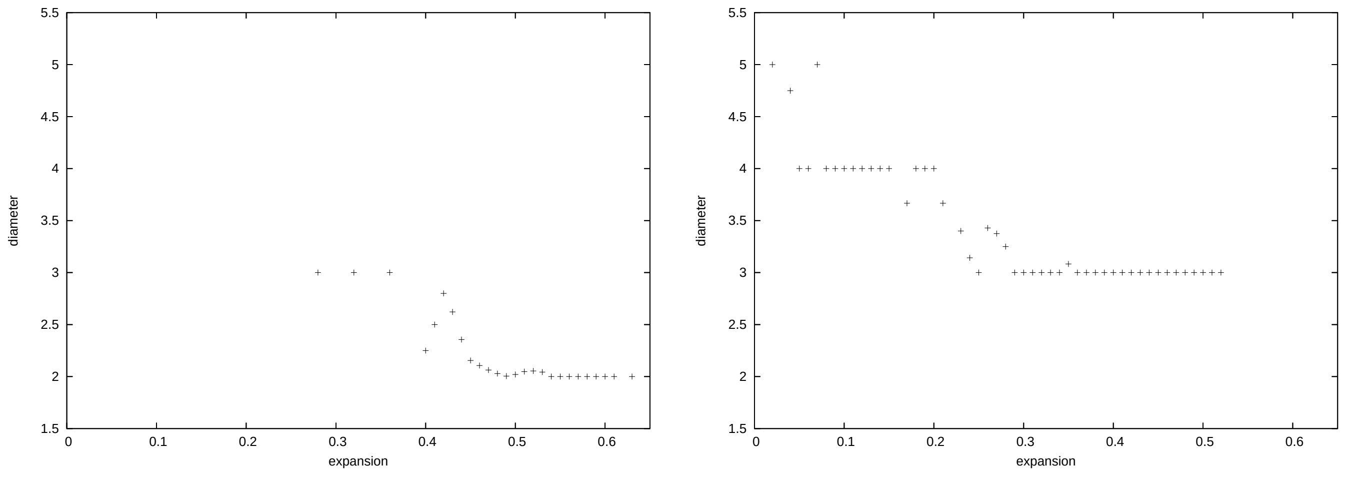
algorithm	median expansion	mean expansion	std dev expansion
inverse^5 walk	0.4816	0.4506	0.1136
residual^5 walk	0.4731	0.4248	0.1371
residual walk	0.4513	0.4325	0.0987
unbiased walk	0.4376	0.4430	0.0577
centralized tracker	0.4292	0.4333	0.0621
inverse walk	0.4259	0.4282	0.0651
Metropolis-Hastings walk	0.4272	0.4326	0.0605

Table 2: Debian Expansion

5.4 Discussion

We summarize the results of Section 5 in a single table for each of the torrent logs which we used to drive our simulations. Tables 1 and 2 show the median, mean, and standard deviation in expansion for the RedHat and the Debian torrents respectively. They are sorted by the median expansion, as this represents their steady state behavior. The slow ramp up of some of the algorithms is reflected in a higher standard deviation, and in a relatively low mean expansion compared to the median expansion for that algorithm.

All of our experiments involved multiple iterations, yet to our surprise we found that the expansion of the graphs generated by multiple iterations of the same algorithm only had negligible differences in their expansion (the average standard deviation in expansion between iterations was 0.0075 for the RedHat trace, and 0.0036 for the Debian trace). In other words a fixed series of node joins and departures determines with high precision the expansion of the constructed graph, at least in the case of the node selection algorithms which we study. We also experimented with extending each perpetual random walk by more than a single step for each new neighbor selection, but found that this did not have a noticeable effect on expansion, validating the perpetual random walks which we adopt from Gkantsidis et al. [17].



(a) RedHat (b) Debian

Figure 5: Diameter by Expansion

It is well known that expansion and diameter are two closely related graph parameters. Indeed, they can both be bound from above and below using the second smallest eigenvalue of the Laplacian [10]. In order to provide an intuition as to the relative import of observed deltas between measured levels of expansion, we show in Figure 5 the observed correlation between expansion and diameter in the graphs generated by the algorithms which we study. Each point on the plot shows the average diameter for all graphs of the given expansion, with expansion being truncated at two decimal places. Note that there are never more than two diameter values observed for a given expansion value, thus an average diameter of 4.75, which occurs for expansion values of .04 in the Debian graphs, indicates that $3/4$ of the graphs were of diameter five, and the remainder of diameter four.

Our first observation about Figure 5 is that while the algorithms we studied yield graphs with various levels of expansion, they all have the same diameter most of the time. Our second observation is that even graphs with low expansion still have relatively low diameters. We leave as future work an analysis of the negative implications of low expansion, especially when combined with relatively low degree, though we expect it to reveal itself in the form of graph fragility in the face of churn (e.g., fragmentation), inaccessible blocks, and decreased aggregate bandwidth.

While we have examined the initial graphs produced by various random walk algorithms, expansion could be improved in many cases by regularly refreshing neighbor sets [31]. This could be done by expiring neighbors after a fixed period of time, by randomly removing neighbors at a slow rate, or by performing regular random walks on one's own neighbor set (which happens automatically on entry points). A good refresh algorithm could likely make up for poor expansion resulting from a sub-par graph construction algorithm, even if it was executed only by a fraction of the peers in the swarm.

Finally, we would like to obtain the high levels of stable-state expansion enjoyed by the degree-biased walks without paying the currently high startup cost in poor expansion. Perhaps this could be accomplished by a mixed-algorithm walk, either combining multiple walks from several different algorithms, or varying the algorithm applied with each step taken, or both. Alternatively, some of the BitTorrent constants which we purposefully did not modify in our experiments could be altered to put tighter bounds on node degrees, another mechanism by which regular graphs could be more closely approximated.

6 Conclusion

BitTorrent has traditionally relied on a tracker, which explicitly tracks every peer in the swarm, to provide randomly selected neighbors both to joining nodes and to nodes with less than minNeighbors (or even less than maxInitiate) neighbors. We have proposed the replacement of the tracker by one or more entry points, where any peer in the swarm can serve as an entry point. Rather than tracking every peer, these entry points use multiple perpetual random walks to randomly select nodes. Because they don't maintain any global state, entry points can be arbitrarily replicated and replaced.

Entry points perform random node selection without tracking all nodes in the swarm by performing multiple perpetual random walks. The bias of these walks determines the distribution from which nodes are randomly sampled. Using trace-driven simulations, we compared the graphs generated by a centralized tracker to those generated using entry points with various types of biased random walks. We used vertex expansion as a means by which to quantify the quality of the graphs generated by each algorithm, and showed that random walks can be used to generate graphs with expansion properties very similar, and sometimes superior, to those found in graphs generated by a tracker.

In addition to the ability of entry points to completely replace the tracker, they can also serve a valuable role when used alongside existing trackers. For example, one or more entry points could be added to a swarm that was actively managed by a tracker, allowing future peers to join even if the tracker went offline. They can also be used by nodes to actively replace failed neighbors in a manner that preserves the swarm's expansion without relying on the tracker.

Acknowledgments

Thanks to David O'Hallaron for his Internet Services course which first got us thinking about the possibility of removing the tracker from BitTorrent. Thanks to Stan Bielski for his help with a class project where we initially explored trackerless BitTorrent. Thanks to the Debian Project for making their tracker logs available for analysis.

References

[1] N. Alon. Eigenvalues and expanders. Combinatorica 6(2):83–96, 1986.

[2] D. Arthur and R. Panigrahy. Analyzing BitTorrent and related peer-to-peer networks. ACM-SIAM Symposium on Discrete Algorithms (SODA), 2006.

[3] A. Awan, R. A. Ferreira, S. Jagannathan, and A. Grama. Distributed uniform sampling in unstructured peer-to-peer networks. Hawaii International Conference on System Sciences (HICSS), 2006.

[4] A. Bagchi A. Bhargava, A. Chaudhary, D. Eppstein, and C. Scheideler. The effect of faults on network expansion. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2004.

[5] A. Beygelzimer, R. Linsker, G. Grinstein, and I. Rish. Improving network robustness by edge modification. Physica A, 357(3–4), 593–612, November 2005.

[6] A. R.. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. IEEE Conference on Computer Communications (INFOCOM), 2006.

[7] M. Blum, R. M. Karp, O. Vornberger, C. H. Papadimitriou, and M. Yannakakis. The complexity of testing whether a graph is a superconcentrator. *Information Processing Letters*, 13(4/5):164–167, 1981.

[8] V. Bourassa and F. Holt. SWAN: Small-world wide area networks. International Conference on Advances in Infrastructure (SSGRR), 2003.

[9] S. Boyd, P. Diaconis, and L. Xiao. Fastest mixing Markov chain on a graph. *SIAM Review*, 46(4):667–689, 2004.

[10] F.R.K. Chung. Spectral graph theory. AMS Publications, 1997.

[11] B. Cohen. Incentives build robustness in BitTorrent. Workshop on Economics of Peer-to-Peer Systems (P2PECON), 2003.

[12] "Experimental draft: BitTorrent trackerless DHT protocol specifications v1.0." [Online]. Available: `http://www.bittorrent.org/Draft_DHT_protocol.html`

[13] C. Cooper, M. Dyer, and C. Greenhill. Sampling regular graphs and a peer-to-peer network. ACM-SIAM Symposium on Discrete Algorithms (SODA), 2005.

[14] C. Cooper, R. Klasing, and R. Radzik. A randomized algorithm for the joining protocol in dynamic distributed networks. In submission. September 2005.

[15] J. Friedman. On the second eigenvalue and random walks in random d-regular graphs. *Combinatorica*, 11(4):331–362, 1991.

[16] D. Gillman. A Chernoff bound for random walks on expander graphs. SIAM Journal on Computing, 27(4), 1203–1219, August 1998.

[17] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. IEEE Conference on Computer Communications (INFOCOM), 2004.

[18] W. Hastings. Monte carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[19] J. Hoffman. Multitracker metadata entry specification. [Online]. Available: `http://www.bittornado.com/docs/multitracker-spec.txt`

[20] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. Al Hamra, and L. Garcs-Erice. Dissecting BitTorrent: five months in a torrent's lifetime. Passive and Active Measurements Workshop (PAM), 2004.

[21] C. Law and K.-Y. Siu. Distributed construction of random expander networks. IEEE Conference on Computer Communications (INFOCOM), 2003.

[22] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. IEEE Conference on Computer Communications (INFOCOM), 2005.

[23] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), 2003.

[24] L. Lovasz. Random walks on graphs: a survey. *Combinatorics, Paul Erdos is Eighty*, 2:1–46, 1993.

[25] P. Mahlmann and C. Schindelhauer. Peer-to-peer networks based on random transformations of connected regular undirected graphs. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2005.

[26] P. Maymounkov and D. Mazieres. Kademlia: a peer-to-peer information system based on the XOR metric. International Workshop on Peer-to-Peer Systems (IPTPS), 2002.

[27] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculation by fast computing machines. *J. Chem. Physics*, 21:1087–1092, 1953.

[28] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter peer-to-peer networks. *IEEE Journal on Selected Areas in Communications*, 21(6), 995–1002, August 2003.

[29] "Peer exchange." [Online]. Available: `http://azureus.aelitis.com/wiki/index.php/Peer_Exchange`

[30] M. K. Reiter, A. Samar, and C. Wang. Distributed construction of a fault-tolerant network from a tree. IEEE Symposium on Reliable Distributed Systems (SRDS), 2005.

[31] V. Vishnumurthy and P .Francis. "On heterogeneous overlay construction and random node selection in unstructured P2P networks." IEEE Conference on Computer Communications (INFOCOM), 2006.