Bandwidth Allocation and Peer Selection Methods in BitTorrent Systems

Pasieka Manuel
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València , España
mapa17@posgrado.upv.es

September 6, 2012

Abstract

The BitTorrent protocol is one of the most successful Peer-to-Peer systems and has received immense attention from researchers as well as the industry. In order to improve its performance a series of protocol modifications have been published and investigated. Focus of this work was set a subset of these improvements that intend to optimize the Bandwidth Scheduling as well as Peer selection. To evaluate some of the most promising protocol modifications they have been implemented inside a Peer Overlay Simulator and tested extensively against the unmodified BitTorrent protocol. This work is therefore presenting a series of protocol modifications, a newly developed Peer Overlay Simulator that is publicly available and results that shows under which circumstances the modificated protocol outperforms the traditional BitTorrent protocol.

Keywords: BitTorrent, Peer selection, Bandwidth Allocation, Simulator

Introduction

This Thesis discusses the peer selection and bandwidth scheduling method used in the BitTorrent protocol, as well as proposals in the literature on how to adapt those in order to improve download performance. After analysing the different proposals a conclusion is presented that captures the core ideas of possible improvements. To evaluate these proposals a simulation comparing peers running the original BitTorrent protocol and a modified Protocol is performed.

The BitTorrent protocol (hence forth BT) was chosen because of its enormous success and widespread use, as the BT Protocol is here to stay and makes up more than 90% of all p2p traffic world wide ([26, 7]). Therefore, the studies of its inner working and the reasons for its success are more than beneficial to the studies of distributed systems.

The topics of peer selection and bandwidth allocation have been chosen for their intrinsic property to peer to peer systems and a wide range of systems to which the knowledge, gained and deepened through process of creating this Thesis, can be applied.

The rest of the document is structured as follows. In chapter (1) the current BT protocol with a historical background and an emphasis on its algorithms of peer selection and bandwidth scheduling is presented. In chapter (2) important publications in this field are discussed. Chapter (3) represents the protocol evaluated improvements and describes the simulation performed. In chapter (5) results of the simulation are presented and discussed. The document ends with chapter (5.8) containing conclusions and references to further work.

Contents

Chapter 1

BitTorrent Protocol

In this chapter, the BitTorrent Protocol and relevant parts to this Thesis are represented. Section 1.1 gives an overview of the BT protocol, followed by section 1.2 with a short historical evolution of BT and section 1.3 that discusses in detail peer selection and bandwidth allocation.

1.1 Overview

BitTorrent is a partially distributed peer-to-peer file transfer application system of which the main steps of operation are the following:

- A user with the desire to download a specific digital content has to provide a corresponding meta info file (*Torrent* file) that has to be acquired by means not described by the BT system.
- This torrent file contains directions to one or more servers (called *Tracker*) that act as central servers to coordinate peers cooperating (what is called *swarm*) in downloading the same content.
- From this tracker, the client receives a list of peers that have the requested data or are downloading the same content at the same time themselves and have parts of the content to share.
- In a peer-to-peer like fashion, the client then connects to these peers directly and exchanges data in order to acquire a complete copy of the requested content.

- After completing the download a client can either choose to remain in the system and act as a *seeder* (that is a peer who only provides data but does not request them itself) to support other peers by spreading the content, or leaving the system.

This very simplified view of the BitTorrent system is ought to be enough for the scope of this work as its focus is set not on the BT protocol as a whole but on specific parts of it, discussed later in this chapter. The interested reader is encouraged to look elsewhere for a more in depth description and explanation of the terminology ([16, 6, 5]).

1.2 Short history on the BitTorrent protocol and client

The BitTorrent [16] protocol although extremely successful [26] and implemented by various projects, has no official specification. Instead there is an unofficial protocol specification [6] that reflects the "common" knowledge of the research community, that was created and is maintained as a wiki.

The original protocol was designed by Bram Cohen and implemented by him in the first BitTorrent client released in 2001. This BitTorrent client which is often referenced in the literature as **BitTorrent Mainline** or *vanilla BitTorrent* was undergoing major changes with each version released. The enormous success of the Mainline client and BitTorrent as a protocol was the basis for the company BitTorrent, Inc. (created by Cohen in 2004) that continues the development and extension of the BitTorrent client, now called **μTorrent**.

Although there are alternative BitTorrent clients [13],[10] available, their market share is low compared to μTorrent [12]. From a researchers point of view, most noteworthy is the fact that all BitTorrent protocol changes introduced into μTorrent have been adopted by all other clients, giving it a leading role not only with regards to popularity in the user base but also as a trendsetter in the evolution of the BitTorrent protocol itself.

On the contrary as one might expect new BitTorrent clients are developed constantly, as the core BitTorrent engine (Azureus [3]) of the popular Vuze

BitTorrent client is distributed under GPL and often used as a basis for the creation of new clients. These clients tend to specialize on certain key modifications of the BitTorrent protocol to follow a specific goal. The OneSwarm [8] client for examples tries to protect user privacy; the client Tribler [11] is a completely decentralized BitTorrent system, lacking Trackers; BitMate [4] on the other hand is specialised to be used by low bandwidth clients.

1.2.1 Evolution of the BitTorrent protocol

The BitTorrent protocol itself was going through several changes since its first use in the BitTorrent Mainline client of 2001.

Starting with Cohen's own description of BitTorrent [16], much of discussion and research, have been devoted in the following years to improve BitTorrent. As a consequence of this the protocol used in the BitTorrent Mainline client was adapted.

Because of the lack of an official protocol specification, as mentioned before, it is difficult to exactly pin down changes or in which release they were introduced into the Mainline client. Besides this uncertainty, the following can be said.

The most important changes are due to the adoption of the new seeding algorithm called **Super Seeding** [1], the improvement of the Optimistic unchoking (OU) algorithm [2] and the extension of the BitTorrent protocol to

[1] Super Seeding: To improve the spreading of new data in the network, the super seeding algorithm is applied by a node (seeder) have a full copy of the file to distribute. Instead of applying the normal seeding protocol which would satisfy request of specific blocks from other nodes, the seeder decides which blocks to distribute. The seeder will continue to distribute the piece until it gets confirmation of at least one other node distributing the same piece and then will change to the next piece. By doing so the time it takes to replicate a full copy of the file is reduced and the overall system performance enhanced.

[2] Optimistic un-chocking (OU) has two important but different roles in the BTP. On one hand, it is used as a node discovery protocol and on the other hand as an altruistic act that supports the boot strapping of new nodes. As in its later role, new nodes that get selected by OU receive data from a node without having to provide data of their own. This way they have data to share and can participate in the BT System. In the first implementation of the protocol, OU would in some cases change the node so rapidly that no whole piece of data, but only fragments of a piece would be transferred. New peers would then have to take a long time to acquire a complete piece and because of that would

include a Distributed Hash Table.

As of 2006 to the day of this writing the BitTorrent core protocol stayed stable and further releases of the μ**Torrent** client concentrated on improving the client itself or adding extensions to the protocol that are of no interest to this work.

1.3 Peer selection and Bandwidth Scheduling

In the following section the Tit-for-tat **(TFT)** and **Optimistic un-choking (OU)** algorithms which are responsible for peer selection and bandwidth scheduling are discussed in detail. This is done to prepare the reader and ensure an extensive understanding about the current state of the BitTorrent Protocol as well as the possibility to better follow the discussions on proposed improvements and their impact on the BT system. In addition and complementary to this section, [23] is found to be an excellent discussion on the topics at hand.

1.3.1 Overview

After a client connects to the tracker, it receives a random generated list of 50-80 peers which are part of the swarm. It is up to the client to decide which peers to contact to and cooperate with. The first thing the client will do is connect to about 40 of these peers and evaluate if they offer any interesting pieces of the file to be downloaded and if so the peer makes its interest known to the other peer (un-choking). If two peers have mutual interest in each others files, they will start to exchange data over a discrete time period (called a slot) and than reevaluate whenever to continue with the same peer in the next slot or choose another peering partner. At normal times the client has four of these slots in place. Three of them will be using the Tit-for-tat (TFT) algorithm and the fourth one will use the Optimistic un-choking algorithm to evaluate which peer to select for cooperation.

wait a long time to participate in the system. OU was optimized, extending the time of an OU slot, by making sure that inside of one OU slot, at least one whole piece would be transferred.

1.3.2 Choke/un-choke

In order to improve the performance of the TCP/IP connections, BT clients keep an open connection to all peers they interact with. The state of a connection between two peers then depends on two properties (choked and interested) for each end of the connection.

- **choked**: meaning that no request will be handled. Effectively closing the connection on the application level between the two peers, but keeping the connection on the network layer open.
- **interested**: Whether or not the remote peer has pieces of data that the client is interested in.

For each connection to another peer the client keeps track of the following states:

- **am_choking**: this client is choking the remote peer (default 1)
- **am_interested**: this client is interested in pieces offered by the remote peer (default 0)
- **peer_choking**: remote peer is choking this client (default 1)
- **peer_interested**: remote peer has interest in pieces offered by this client (default 0)

A peer can choke or un-choke another at any time. This could for example happen if a peer receives pieces it was waiting for by another peer and therefore loses interest or has found a better peer to exchange data with.

As soon as both sides of the connection un-choke one another and at least one has interest, the previously requested pieces of data will be transferred.

As mentioned earlier a client will have four slots open at any time whereby three are applying the TFT algorithm and the fourth uses the OU algorithm. The upload bandwidth of the client is separated in four parts of the same size and is used in each slot separately. The TFT algorithm will be running 10 seconds on a slot and than restart where the OU algorithm is only executed every 30 seconds, effectively running 9 TFT and 1 OU slot in a time period of 30 seconds.

1.3.3 Tit-for-Tat Algorithm

The Tit-for-Tat algorithm is a distributed peer selection algorithm that rates peers by their contribution (in upload bandwidth) they provide to the client. Resulting in the BitTorrent protocol with three TFT slots, that the three peers that have uploaded most data during the previous slot will be chosen in the next TFT slot. Ranking peers by their upload capacity is effectively hindering FreeRiders[3] and is making sure the client will finish the download as fast as possible.

1.3.4 Free Riders

FreeRiders are peers that contribute very little or no resources to other peers. Previous peer-to-peer applications like Gnutella or Napster had big performance losses as more than 70% of their clients where effectively behaving like FreeRiders. The TFT algorithm is found to be a working countermeasure against FreeRiders [23] and is one of the reasons BitTorrent is as successful as it is. Peers that provide little or no upload bandwidth are rated so low by the TFT algorithm compared to cooperating peers that they are highly unlikely to be chosen by TFT, leaving FreeRiders only the chance to download from seeders or be chosen by OU.

1.3.5 Optimistic un-choking Algorithm

In order to find newer peers that may have better upload performance, one of the active slots in the BT protocol is scheduled using the Optimistic un-choking algorithm. In contrary to the TFT algorithm, the OU algorithm chooses one peer by random[4]. This **Peer discovery** (more will be said later) aims to please other peers by providing upload bandwidth to them and in return be rated higher during the next execution of the TFT algorithm. The intention of this is the following. The peer (called local peer) connects to another peer (called remote peer) and at the beginning has no possibility to rate the quality/upload capacity of each other because they never exchanged data. This changes as the local peer selects the remote peer by OU for an active slot and contributes. In return the remote peer on its next TFT slot

[3]A more detailed discussion will follow later.

[4]Skipping seeders completely but can being able to selecting peers that, for the moment at least, are not interested.

allocation will rate the local peer. If the local peer receives a high enough rating, the remote peer will allocate it as a TFT slot and return data. Now the local peer is able to rate the remote peer (as it receives contribution by it) and if the contribution is high enough, will allocate it as an active TFT slot as well. After this "handshake" like process both peers are either interested in each other and schedule each other for TFT slots or not and will continue with their current TFT allocations and next OU allocations. It is noteworthy that in order for a peer to be scheduled in a TFT slot it does not need to remain in the OU slot for a whole OU Period of thirty seconds, but can be scheduled at any TFT allocation algorithm.

1.3.6 Peer discovery and Bootstrapping

The OU algorithm has two important functions in the BitTorrent protocol. It is on one hand used as peer discovery algorithm by testing the upload performance of new nodes which then can be used in TFT slots, and on the other hand provides new peers (during what is called the Bootstrapping phase) that do not yet have any data to share (hindering them to be chosen by the TFT algorithm as they have no way to provide data and therefore cannot cooperate, leaving them with a bad TFT rating and a wasted upload bandwidth) with the first piece of the file to download. This is supported by making OU slots three times as long as TFT slots and by doing so giving new clients a higher chance to receive a whole piece of data as they are selected for an OU slot. The OU slot length is therefore a parameter with which it is possible to tune how effective OU will be as method for peer discovery (using a short slot length and therefore traverse more peers) or be used to support new comers (by an increased slot length).

Chapter 2

Literature

In the following chapter, publications about peer selection and bandwidth scheduling methods in BitTorrent systems are reviewed. The chapter is organized in an introduction that gives an overview about publication in this field, followed by a selection of articles that discuss protocol improvements and ends with a conclusion on general concepts underlying the proposed modifications.

2.1 Introduction

The BitTorrent system is discussed and evaluated extensively in the literature and has received much attention in the time period between 2003 and 2006 ([16, 25, 20, 1]), concentrating on the BitTorrent protocol as it was implemented in the Mainline client [23].

As of 2006 a series of articles ([24], [22] and [19]) started to emerge that put their focus on adapting the BT protocol by modifying the peer and bandwidth allocation algorithms. These articles have been selected to be discussed in detail as they have either received a high response and interest of the research community or present complementary concepts on how to improve the BT system performance.

In the articles ([15, 18]) which will be discussed as well, peer selection is applied to other interesting ends. The focus of these works lies on methods of applying peer selection in order to improve network performance by reducing

traffic traversing different autonomous systems (AS) like Internet Service Provides (ISPs).

2.2 Survey on Methods to optimizing network usage

As BitTorrent systems are responsible for the majority of Internet traffic worldwide ([26, 7]) suspicions have been raised if ISPs will not cut costs by filtering and traffic shaping of BitTorrent traffic and by doing so degenerate system performance. In response to this a few publications have investigated the possibilities of reducing network traffic between autonomous systems (AS) in order to make sure that possible manipulation of data streams by ISPs will not reduce the download performance of BitTorrent systems and reduce costs for ISPs.

Two different solutions ([15, 18]) to this problem are presented and discussed.

2.2.1 Taming the Torrent

[15] is found to contain an excellent discussion on the topic; where the authors have developed a system that works on top of public available content distribution networks (CDNs) that are used to approximate the network distance between peers and generate a metric that can then be used by peer selection methods to promote neighbours in order to reduce inter AS network traffic.

The authors follow the assumption that CDNs are located all over the world in high numbers and that they have an excellent map of the Internet in order to provide replicas of content servers near to the consumer. These CDNs work by dynamically forwarding clients via DNS redirection or URL rewriting to its closest content replica. As CDNs work in cooperation with different ISPs they have a good insight in which the client (depending on its public ip address) is best served by which content replica. Policy considerations aside the most important factor on which to define distance is network latency. Therefore it is assumed that if two peers are forwarded to the same replica are also close to each other. Taking into account that an

ISP that is eager to reduce traffic crossing its borders will most likely prefer to serve local peers (peer that reside in its own domain) by local content servers; clustering peers based on the content server they are redirected to, effectively groups peers that are in the same domain.

Evaluation and Results

The authors have developed a BitTorrent extension to the Azureus client that modifies the peer selection algorithm in order to prefer peers close to oneself on the basis described earlier. As of the popularity of Azureus clients they have received data from more than 120 000 clients, providing them with real-world feedback about their system.

The collected data was used to compare download performance of clients applying the bias peer selection to clients using the normal (unbiased) peer selection. Results show that choosing a peer from the same AS can be beneficial because of network properties like lower package drop rate and lower latency, but only in rare cases the average transfer rate depends on the number of AS hop counts[1].

2.2.2 A more complex Model

In [18] the authors follow a different approach, presenting a system that optimizes network usage by constructing an extensive model of the network structure and its transport properties on itself instead relying on an underlying CDN. This model is constantly updated and used to calculate virtual costs for each network operation. By applying cost minimizing functions that assign a high cost to traffic crossing the boards of different AS, it is attempted to optimize the network performance.

The described cost calculations are done not by the BT clients themselves but by modified BitTorrent trackers (called P2P Redirectores) that receive client requests for a peer list and return a filtered subset of swarm members. The peers returned are elected on the basis of the requesting peer and the cost traffic between them would produce. In order to do so the P2P Redirector

[1] AS Hop count is the number of different AS a packet going from one peer to the other has to cross.

needs in part information [2] that is only known to the ISP of a network. The system therefore requires to have Redirectors in each AS and the cooperation of the local domain authority.

The cost model includes network costs like bandwidth usage, transport delay, package loss and the costs to traffic crossing AS borders, as well as costs generated at the endpoints of the connections (on the peer side) because of bandwidth usage.

Evaluation and Results

The proposed system is simulated and compared to previous work done by the same group as well as to another system that optimizes P2P network traffic. The scenarios or the inner workings of the simulation are not presented but the system is shown to outperform its competitors.

2.2.3 Discussion

The first article [15] presented at first glance appears to be an elegant solution to the problem but leaves some elements of doubt with respect to the assumptions made by the authors. The inner working of CDN are not easily observable and the Content Servers (CS) are administrated by third party companies that may be bound to provide some kind of service quality to the ISP but in the end will try to reduce their own costs and expenses. This results in a system in which, due to load balancing or cost optimization, users are not served by the local CS but by someone else. Even more difficult is to evaluate how many CSs resign in a single AS and if their density is equally distributed. In a case where a single ISP has hundreds of CSs, it is very likely that peers on different ends of the same domain will be redirected to different CSs. It can then be easily possible that a peer from another ISP is closer (in a metric of matching CDs) to this peer than to one in the same domain on the other side of the network.

The second article [18] leaves many questions unanswered and partially lacks proof for its claims, but even if this is left aside its main weak points are the complexity of its model and the dependency on the ISPs. Besides the

[2] Like membership information of an ip.

many network properties the model depends on, which need constant observation and testing, the network status between two peers is approximated by evaluating the path between the two corresponding P2P Redirectors responsible for the peer. This may benefit scalability as the system grows in complexity with the number of P2P Redirectors and not with the number of peers, but loses accuracy as well and the weight of this loss is not discussed in the article. In addition to the technical challenges of this solution there are undiscussed topics of trust and controllability. The P2P Redirectors depend on crucial information to be provided by the local ISP, but the question arises as to how they can be trusted with the data provided if they have not been trusted in the first place to let data travel freely and not filter it for their benefits. With this in mind it seems that this is less a bug but a feature of the system, which supports central control of data flow and censorship.

2.2.4 Conclusion

The presented works succeed in using peer selection in order to effectively reduce inter ISP traffic but fail to improve download performance. As presented in the results [15], in none of the cases was a major difference in download time observed, and in the cases where this happened, it was traced back to strong filtering on the side of very few ISPs. Current studies [7] support the suggestion that ISP almost never interfere with BitTorrent traffic. This may partially be because the percentage of Internet traffic produced by BitTorrent application is dropping and traffic caused by alternative systems, like video streaming, that are more easily controlled, are growing.

It was therefore decided during the writing of this Thesis that improving download performance by reducing inter ISP network traffic is not a path worth following and as a consequence will not be discussed in more detail from here on.

2.3 Survey on Bandwidth scheduling Algorithms

The articles [24, 22, 19] presented in the following section will apply bandwidth allocation methods in order to increase download performance.

In [24] the authors present a selfish BitTorrent client which tries to establish as much connections as possible to utilize its download bandwidth as opposed to [22] in which a client tries to establish at best a single connection to another trusted client to improve not its own performance but that of the whole system. In [19] the authors present a system to manage the ratio of OU and TFT slots in order to have a higher peer discovery during the bootstrap and endgame phase without changing the number of peers to cooperate with.

2.3.1 The more the better

Article [24] probably is one of the most controversial of its kind as it breaks selfishly with the concept of cooperating peers and asks the questions how to maximize the performance of a single peer by abusing the good will and altruistic acts of other peers.

The authors find high capacity nodes that on one hand need a great amount of time to find peers matching their bandwidth and because of that "waste" upload bandwidth to peers that return little during that time, and on the other hand restrict themselves to too few connections which results in a not fully utilized download bandwidth as a source of altruism that can be used.

On these results the authors build their own BitTorrent Client called *BitTyrant* that follows three basic concepts

- *Maximize reciprocation bandwidth per connection*: meaning to select those peers from the known peer list that upload most
- *Maximize number of reciprocating peers*: don't limit the client to only four peers to connect to, but download from as many as needed to fully utilize the download bandwidth
- *Deviate from equal split*: upload to each cooperating peer as little as possible so that it continues to reciprocate and use the excess of upload bandwidth to connect to even more peers

These concepts are implemented in BitTyrant by modifying the TFT algorithm to not equally split the upload bandwidth to three (with OU,

four) cooperating peers, but rather connect to as many peers as possible and to individually upload as little as needed to keep the peer to reciprocate. Further the bandwidth provided to each peering partner is not a discrete fraction of the total upload bandwidth as in the normal TFT protocol, but it is adapted continually, with the goal of minimizing invested upload. In addition the protocol is not time based (meaning up/download slots being allocated for a specific time) but event based. Nothing is said about the number of OU slots created, but it can be assumed that there is no limit either, and as long as unused upload bandwidth is available, new peers are discovered.

Evaluation and Results

The created client is evaluated in comparison with a standard Azureus client on a real-world BitTorrent swarm and two PlanetLab [9] simulations. In one of the simulations the clients behaviour is analysed depending on its upload capacity in a swarm of normal BitTorrent clients and in the second simulation a swarm of nodes all using BitTyrant is created to evaluate the results of a wide spread usage of *BitTyrrant*.

In comparison to the standard BT client in a real-world swarm, *BitTyrrant* can almost always reduce download time and in 25% of the cases is even twice as fast in completing a download than its counterpart. A similar reduction in download time can be observed during the PlanetLab simulations, where all peers run a *BitTyrrant* client as well a more stable behaviour regarding the variation in download time during different trails of the simulation and in upload capacity usage.

In cases when *BitTyrrant* cannot improve performance, the authors suspect this to be due to either the missing of high capacity peers and their altruistic contributions or fresh torrents where the number of available peers is low because of low data availability.

Lesson to learn

Modifying the normal BT bandwidth allocation algorithm in order to minimize upload bandwidth for each connection, and having a dynamic number of active connections seem to produce the following main effects.

For one, the improvement in download performance as presented in the article appears to be due to increased number of reciprocal peers and their combined upload, suggesting that it is beneficial to have a dynamic number of active peers with which to cooperate than to rely on only the most promising four as done in the standard TFT protocol. Second the authors found a dynamic number of OU slots to reduce the Bootstrap time of new peers and therefore reduce overall download time. This is similar to the first effect suggesting that not limiting the number of OU slots can increase peer discovery and in the long run download performance.

Discussion

That *BitTyrrant* is not free of problems and less than optimal behaviour can at best be seen in the pure *BitTyrrant* simulations where all peers run a *BitTyrrant* client.

In those scenarios the high capacity bandwidth peers play an important role for peers with a low bandwidth connection as they contribute a lot more than other peers while searching for the optimal peering partner. Where though a high bandwidth capacity peer running *BitTyrrant* splits its upload into many smaller fragments than a normal BT client would do, and by doing so effectively acts as a low bandwidth peer, and therefore not provide those benefits.

A purely selfish *BitTyrrant* peer only cares about using its OU slots for peer discovery, and will try to reduce the altruistic aspect of OU which is responsible for providing highly needed data to new peers during their Bootstrap phase. A BitTyrrant client therefore reduces its own Bootstrapping phase but extends that of others. This results in an overall extended Bootstrap phase in situations where all clients behave selfishly.

Difficult to evaluate but a possible savvier additional downside of the *BitTyrrant* system is the inefficient usage of the TCP/IP protocol by connection to a high number of peers which increases protocol overhead as well as the potentially fast switching between peers that may not give enough time to the network protocol to fully ramp up and make use of the available bandwidth. This would get worse in a scenario that is not discussed in the article but that effectively outers itself in a race to the bottom where every client reduces the upload bandwidth per connection to lower and lower values while connecting to more and more peers.

To counter these effects, it makes sense on one hand to have a maximum number of peers to actively cooperate and on the other hand to make the UO slots long and frequently enough to help new comers during their Bootstrap phase.

2.3.2 A good friend is better than a thousand colleagues

A completely different concept for improving the download performance over the standard BitTorrent protocol, as compared to the previous section 2.3.1, is explored in [22] where the authors design a system of trusted peers that try to minimize the number of cooperating nodes and try to find a single optimal partner with whom to exchange data.

The designed system follows two main concepts

- *The uplink should be partitioned minimally*: meaning that uplink bandwidth should be focused on as few active peers as possible.
- *The allocation should be constrained minimally*: not partitioning the upload bandwidth symmetrically between all cooperating peers, but give preference to high upload capacity nodes which themselves can spread the data fast to others.

The modified uplink allocation algorithm designed by the authors is called *BitMax* and has at its core element a peer rating function decides, which peer to upload which fragment of the upload bandwidth. *BitMax* comes in two flavours (MAX1 and MAX2) where in the first case peers are solely ranked on their bandwidth capacity and the amount of data they receive by other nodes, and in the second case where nodes that are of high interest to their neighbours (because of the pieces they have to offer) are preferred even if they lack a high capacity bandwidth. These rating for each peer is then used in a Knapsack algorithm to greedy find the best fit of peer rating to upload usage which maximises the overall network utility.

This is in contrast to the normal TFT/OU scheduling algorithm used in BitTorrent that heavily relies on the tit-for-tat concept to promote peers that cooperate and penalizes FreeRiders, and an even bigger step away from the egoistic *BitTyrrant* client.

With the *BitMax* algorithm the authors develop a system that neglects the desire of a single peer to increase its download performance for the benefit of the system as a whole and promote a scenario in that all peers work together to increase the overall system performance through which the download performance of every single peer is improved.

In order for the system to work properly, the authors have hard constraints on the trust that peers invest in each other and on the homogeneity of the clients. In an optimal situation, the authors would use swarms of BitMax only peers and a closed client that offers no way to manipulate its behaviour to the user. This is partially needed because during the ranking of its peering partners a node needs vital information on the network capacity and performance of its neighbours that they have to provide and the peer itself has no way to validate in order to adequately maximize network utility.

Evaluation and Results

The performance of *BitMax* is evaluated in the peer-to-peer simulator GPS [27] with three different network scenarios. The most simple scenario contains nodes that are restricted in regard to their upload capacity only (Case U), followed by an additional restriction of download capacity (Case UD) and the last and most restricted scenario (Case UDN) in which traffic shaping by ISPs is simulating by restricting the maximum network usage between groups of peers that are of different autonomous systems (AS).

In all cases of the simulation *BitMax* outperforms the normal BitTorrent client and often, by a factor of two, is reducing the download time. The authors find this to be the result of a better upload bandwidth utilization compared to the normal BitTorrent client that may be performing well in the *steady state* phase of a Torrent download where a client has a wide range of peers to choose from with which to cooperate and is not restricted by data availability, but is performing sub optimal in cases of new torrents where data availability is low.

Lesson to learn

Although completely different in their concepts and implementation, the two clients, *BitMax* and *BitTyrrant*, appear to achieve similar improvement on

download performance over the normal BitTorrent client. It is interesting that in case of *BitMax*, the source of its performance increase is claimed to be the focusing of upload bandwidth on as few nodes as possible as in *BitTyrrant* where doing the opposite[3] seems to achieve the same results by using as many peers as possible.

A possible explanation for this can be the following. In the beginning of the life time of a Torrent, the low data availability and clustering effects result in environment where focusing upload bandwidth to few peers promotes the faster spread of rare pieces and by doing so, reduces the waste of bandwidth due to nodes not having enough data to share. After enough copies of the file are available in the network, the restriction to only peer with four nodes at the same time may be sub-optimal and letting this restriction fall, as is done explicitly in *BitTyrrant* and is a possibility[4] in *BitMax*, may be the cause of a reduced download time.

Discussion

Considering the goal of the Algorithm to focus its upload bandwidth to as few peers as possible, some of the results, showing the number of un-choked peers for example, list a rather surprising high number of peers the client tries to cooperate with, but unfortunately, nothing is mentioned regarding how many of those connections are actually used at the same time to exchange data. In the UDN simulation case the authors conclude that it may be beneficial to extend the maximum number of active nodes to more than 4, but again this is not discussed in more detail.

Concerning trust dependencies of the system, it may well be that there are environments where the vast majority of nodes can be trusted to cooperate and to have only the good of the system in mind, but this is difficult to apply and expect of the majority of BitTorrent system on the Internet and in operation as of this writing. A peer to peer system having such high demands on trust between nodes is often an easy target for malicious peers and FreeRiders that have only their own benefits in mind. *BitMax* is therefore

[3] As described in section 2.3.1 the BitTyrrant client establishes a high number of connections to active peers to improve its download performance.

[4] As mentioned in 2.3.2 the authors suggest that it may be appropriate to not limit the maximum number of active peers to four in order to achieve a better upload bandwidth utilization.

more appropriate to be used in private networks, like in companies or Internet communities with restricted access.

2.3.3 Balance of Powers

In this [19] work, the authors are not concerned with the number of active peers a node is cooperating with, but with the management of the ratio between TFT and OU slots. The authors find one reason for a no-optimal utilization of the upload Bandwidth utilization during the Bootstrap and the endgame phase of life cycle of a BT client. The reason for this is presented to be a static quota-based peer selection strategy that uses three TFT and one UO slots regardless of the need for a faster peer discovery during these phases.

The authors therefore propose a system that uses a dynamic quota-based peer selection strategy that classifies known peers into distinct sets and adapts the ratio between TFT and OU slots depending on the size of these sets. The system distinguishes between *high return* peers that a client had cooperated successfully in the near past, *urgent* peers that hold needed pieces of data but with whom no data was exchanged yet and peers on which nothing is known of the data they hold. The rest (peers that constitute of uninteresting peers) are categorised as *normal* peers.

During the download of the first and the last piece, the number of *urgent* to *high return* is high and because of that more upload slots are set to use the OU than to TFT algorithm. In addition, the system can be tuned by adjusting the parameters α and β although as can be seen from the results of the experimental evaluation their impact seems of no great importance.

Evaluation and Results

The algorithm is evaluated in an undescribed and unknown simulator with an initial configuration of 1000 peers and one seeder. Peers are configured to remain in the swarm for a short period of time after completing the download acting as seeders. The network configuration provides for heterogeneity in node connection bandwidth by separating the peers in three distinct classes with different upload and download capacities.

The focus of the simulation is set to the time needed to share the first and

the last piece of a torrent. Four BitTorrent clients are compared, whereby one is the normal static quota based BitTorrent client and the other three use a dynamic quota with slightly different α and β parameters.

The authors claim a reduction of download time for the first and the last block of about 59-63% and for all blocks of 62-63% over a normal BT client.

Lesson to learn

This work shows the possibility of improving the BT system by not only being flexible on the number of peers to cooperate with (in TFT and OU slots) but as well on the type of slot (TFT or OU) which to allocate to each connection. It would be therefore interesting to evaluate a system that dynamically adjusts the number of active peers as well as the ratio between TFT and OU slots.

2.4 Conclusion

Reflecting on the presented articles and the concepts extracted out of them, the following ideas appear interesting to evaluate and determine their use in improving the download performance of future BitTorrent systems.

- *Variable number of active peers*: meaning that instead of limiting the number of active peers to four the system should try to adapt to the situation allowing for more or less active peers, which even improves download performance.
- *Dynamic quota ratio between TFT and OU Slots*: depending on the data availability and its pool of possible peering partners, a client should dedicate more or less in peer discovery using optimistic unchoking more than one peer at the time.
- *Asymmetric bandwidth allocation*: clients should try to get the best peering partner for their bandwidth and neighbourhood configuration. Meaning that in a situation in which reciprocation of another peer is achieved by providing less upload bandwidth than available, the client shall do so, and invest the excess of bandwidth in creating connections to other peers or to discover new peers using OU.

- *Tend to work with less peers and limit bandwidth fragmentation*: meaning that a client should try to find a few good/suitable peers with which to saturate its download bandwidth and if this is not possible, shall limit the number of active clients to a maximum value in order to not over fragment its upload bandwidth and in consequence reduce TCP/IP performance and download performance.

Chapter 3

Proposed modifications

In the following chapter a series of modifications to the normal BitTorrent protocol [6] as discussed in chapter 2 and summarized in section 2.4 are presented.

This chapter is structured as follows. In section 3.1 a modification adapting the number of TFT and OU slots is explained, followed by section 3.2 that explains how the variable slow number effects peer discovery and how Bandwidth allocation is adapted, ending with section 3.3 that explains the modifications to the TFT algorithm.

3.1 Variable number of TFT and OU Slots

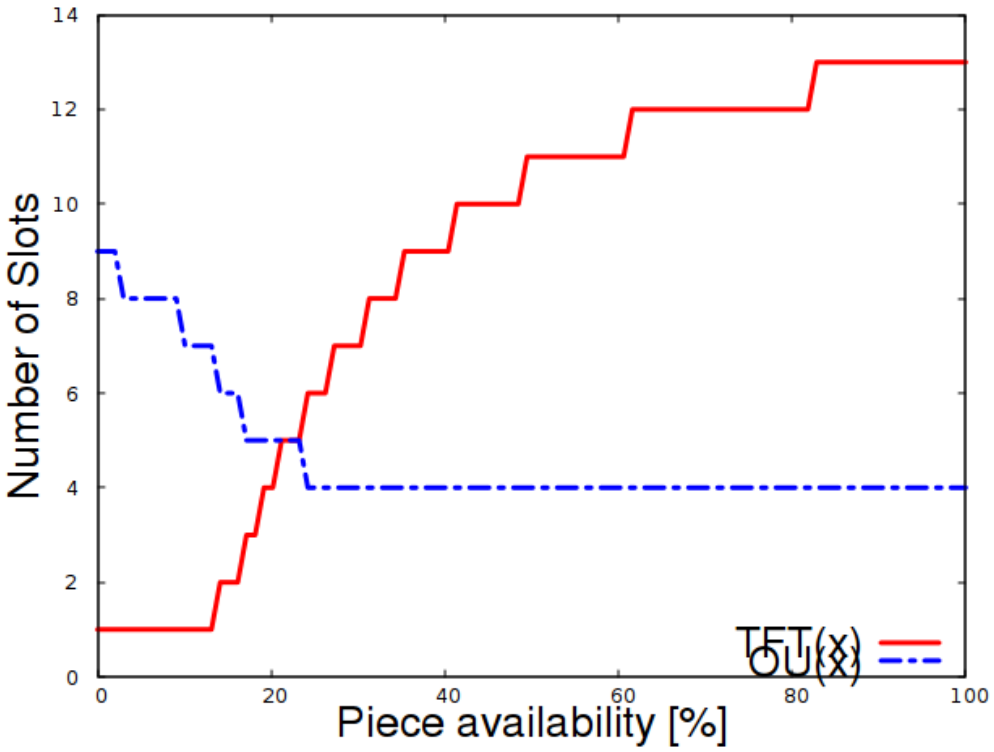
The normal BitTorrent protocol is modified by making the number of OU and TFT slots variable and to specify the bandwidth capacity for each connection individually. Whereby the number of OU and TFT slots are calculated based on the local piece availability. The concrete formulas are shown in 3.1 , 3.2 and where found by empirical studies. A plot showing the course of the functions is shown in 3.1

$$maxTFTSlots = \lfloor \arctan(0.05 \ * (x - 10)) * 10) \rfloor \tag{3.1}$$

$$nOutSlots = \lfloor (8 - ((\arctan(0.15 * \ x - 2) * 0.5) + 0.5) * 4) + 1 \rfloor \tag{3.2}$$

Where equation 3.1 is used to calculate the maximum number of TFT slots, and equation 3.2 the number of OU Slots. In addition the “maxTFT-

Figure 3.1: TFT and OU Slots



Slot" is controlled to at least have a value of 1 and "nOutSlot" is addition divided by a factor of {1, 2, 4} (in a round robin like fashion every 30 ticks a new divider is set) as the piece availability rises above 15%.

The number of TFT and OU slots is calculated with the following ideas in mind. During the bootstrap phase or the beginning of the lifetime of a torrent (when the piece availability is low) a high number of OU slots are allocated to support new clients and rapidly spread the first pieces. As enough pieces are available, the self interest of a client shall be enforced and more TFT slots should be used. It is important to note that "maxTFTSlot" is only defining an upper limit on the number of TFT slots to allocate. The actual number of TFT slots to use is set by the modified TFT algorithm as described in section 3.3.

3.2 Enhanced Peer Discovery and Bandwidth controlling

The bandwidth allocated of TFT and OU Slots depends on the ratio between the total number of OU slots and the maximum number of TFT slots. As seen in figure 3.1 this means that at a low piece availability rate 90% of bandwidth is dedicated to OU slots and slowly drops (about 50% with a piece availability of 20% and then slowly comes closer to about 25% as the piece availability rises above 70%).

As described in section 3.1 the number of OU slots with a local piece availability of more then 15% is set to variate between the values 1, 2, 4. What this means is that for calculating the bandwidth shared between TFT and OU, the number of OU slots is held constant at 4, but the actual number of peers scheduled for OU slots variates to be either 1, 2 or 4. This is done so that the peer discovery process (for which OU is partially used) is finding peers with different costs of reciprocation. In the case that only one peer is scheduled by the OU algorithm, the whole portion of OU bandwidth is used to please this peer, whereby if four peers are scheduled, each of them only receives a fourth of the total OU bandwidth. This makes it possible to find peers and get them to reciprocate for a lower dedication of upload bandwidth than with the normal BitTorrent protocol, which always uses one fourth of its upload bandwidth.

3.3 Greedy TFT allocation

The TFT algorithm is modified to make use of the peers discovered by the OU algorithm with different upload bandwidth by applying greedy knapsack algorithm that rates each peer as a ratio between their provided download bandwidth and the upload bandwidth that was needed to make the reciprocate. The goal of the algorithm is to saturate the download bandwidth of a client with providing as little upload bandwidth as possible. This is normally not achieved because in all simulation scenarios, the download bandwidth of a peer is always by multiple factors larger than its upload bandwidth. What the knapsack algorithm then achieves is the maximization of the download rate of the client with the upload bandwidth available.

Chapter 4

Simulation and Evaluation

In this chapter a series of practical proposals to improve BT download performance are presented and their evaluation in a newly developed simulator are described.

The chapter is organized in the following way. In section 4.1 an overview of possible methods to evaluate a BT client are discussed. In section 4.1.2 the evaluation using an Simulator as well as available simulators is presented, following section 4.2 where a new Simulator, developed during this Thesis is presented.

4.1 Methods of evaluation

In the following section the benefits and disadvantages of evaluation using real-world implementations and simulations as well as the experience gained in processing these methods are described.

4.1.1 Real-world clients and client extensions

There exists a possibility of implementing a complete client, which has the benefit of real-world results, that can be evaluated directly, but comes with a high price of development effort and difficulties of practical testing. Specially peer to peer applications are difficult to evaluate as they normally depend on a high number of peers running the application. To evaluate the system in a realistic environment where peers behave very different to each other (

because of available resources like bandwidth or data but also to local policy like up time and willing to cooperate) can be a lengthy endeavour. To ease this process, special frameworks like [9] have been developed in which researchers can run their applications in an artificial constructed environment, but doing so on one hand remains a heavy task and on the other hand may distort results because the constructed environment may not behave as the target group. In cases like [4] [24] [11], the developers therefore have chosen to implement a real-world peer to peer application on top of a BitTorrent core [3]. This has the benefit of receiving real-world results in a natural testing environment where the clients can use the underlying peer to peer network; as long as they remain compatible with the core protocol; and still can introduce new features and modifications.

This path was pursued first for the BT protocol modifications described later, but was then dropped as the BT core application [3] was found to be public available source code but is completely undocumented. The work that would therefore be needed to implement the modifications proposed was evaluated to be out of reach for this thesis.

4.1.2 Simulation

An alternative to implementing a complete client is the use of simulation in which only parts of the client (in an optimal scenario only the proposed modifications) are implemented and the behaviour of the client is simulated in a carefully designed test environment. This has in general a series of advantages like a reduced development effort, more control over the testing environment and therefore the possibility to evaluate the system under different assumptions, as well as a complete oversight of the peer to peer network which is not possible in a real-world example where only parts of the network are visible. This comes at a cost and like all simulations that by definition simplify the process they simulate, and that is precision. Depending on the care and focus taken during the development of the simulation, the results acquired can at best be used to predict the behaviour of an application in the real-world. Another deficit of simulation depend on the completeness and capabilities of the simulation framework applied, but can be the loss of focus as there may be the need to extend or modify the simulation framework itself in order to implement the desired modifications. Therefore time and development effort is diverted from the main goal. As long as the simulation

is not proven to be correct [1] results acquired in this way are still left in doubt and speculation.

Therefore simulations are often used as a first step to guide the development of a system that later is evaluated by implementing a real world client.

As described earlier because of the time limitations of this Thesis the proposed modifications have not been implemented in a real-world client, but the first step in this direction was taken by implementing them inside of a simulator.

A series of simulators for BitTorrent system [27] [21] [17] [2] exist and continue to be developed as of this writing. A comprehensive survey of available simulators was performed at the beginning of this work, but unfortunately found all available system not fit for the task at hand. A short summary of the findings is presented here as follows.

The General Purpose Simulator for P2P networks [27] seem have been deserted by its creators and is lacking support in all dimensions. It was evaluated due the lack of documentation and active development of the project this makes no basis for further studies.

Similarly, this applies to [2], that lacks active development but at least has partially documented source code.

The projects [21] and [17] both are active projects and as they work on top of the common OMNet++ Simulation framework receive a decent amount of attention by developers and researchers. The reasons for not choosing one of them are the following. For one neither has a extensive source code documentation, making it necessary to first get an in depth understanding of OMNet++ and then interpret the provided extensions. Further and even worse, both work on top of the INET extension, which is responsible for simulating realistic network characteristics, down to simulation TCP/IP package traffic. This may gain precision, but demands a clear view of the data network on which the BT system should be simulated. Trying to describe a real-world peer to peer network structure with hundreds of peers interconnected by different hierarchies of routers and network infrastructure

[1] correct is here to be understood as the correlations between results obtained by simulation and results obtained by real-world experiments. Always considering the doubt in precision of results and the limited scope of properties evaluated.

is out of question and definitely not the goal of this Thesis. Using simplified network models that behave more like a LAN than the Internet raise the question why to use such an extended model like the INET in the first place, and secondly if simulating BitTorrent on a LAN like environment will not generate results that are not applicable to an Internet environment. Besides these conceptual doubts, both systems were unsuited for at least one of the modification proposed, as it is needed to control the bandwidth that is reserved for each active connection, and bandwidth control is not possible with the INET extension and therefore not possible to be simulated.

Concluding this survey, it was decided to implement a new Simulator (described in section 4.2) that is less general and more optimized to evaluate the modifications proposed. As it turns out during the process of writing this Thesis writing a new simulator has in addition the major benefit that it supports the automatic generation of statistics in a detail and focus not seen in any other simulator.

4.2 Eruliaf

In this section the Peer to Peer BitTorrent simulator [14] developed during this Thesis is presented. At first an overview of major features is presented, followed by discussion and details about the implementation and main components and ending the chapter with an overview of the methods provided by the simulator to gather statistics of the simulation.

4.2.1 Introduction

The simulator is implementing the following major features that will be discussed later in more detail:

- Discrete event based peer to peer simulator
- Overlay peer to peer network simulation
- Implement the BitTorrent protocol (Seeder,Tracker, Peers - missing end game algorithm)
- Simplified network model (upload/download Bandwidth with uniform noise)

- easy specification of simulation properties (number of peers, rate of new peers joining/leaving swarm, torrent size, number of pieces, network bandwidth)
- Framework to launch multiple simulations in parallel (utilizing multi-core system)
- automatic generation of statistics and summaries

Eruliaf is a discrete event peer to peer overlay simulator for the BitTorrent system implemented in Python3 for the console (without graphical user interface).

The simulation has implemented as described by the BitTorrent protocol [6] without the end game algorithm. Every simulation contains a single Tracker, a single initial seeder and multiple peers.

Communication is simulated on a peer overlay network only, meaning that no network simulation is performed. A peer is able to connect to any other peer directly (fully connected graph) but only knows a subset of available nodes (its neighbourhood). The interconnection between peers is defined by upload/download bandwidth only (neglecting latency, jitter or dramatic network changes), but slight variations in connection quality between peers is reflected by adding a uniform random noise of $\pm 10\%$ to bandwidth capacity.

A complete simulation contains a scenario file, specifying simulation details and a simulation file specifying which scenario to execute and where to store temporary files as well as simulations results. At the end of a simulation the generated simulation data is used to generate statistical diagrams summarised in a single pdf file.

4.2.2 Simulation and Scenario files

End-users are interacting with the simulator using the simulation and scenario files only. Scenario files configure a single instance of a simulation where simulation files are used by the simulation framework to run multiple scenarios. When using the simulation framework only a simulation file has to be provided by the user. The framework will then automatically generate scenario files and pass them to the simulator.

A normal simulation is therefore a three phase process. First the simulation framework parses the simulation file, secondly generates different

scenario files and executes the simulation as them as arguments, and in the third phase generates statistics based on the results of each scenario.

Scenario files

In appendix A.1 a simple example scenario file is presented. The file contains three categories (General, Peer and PeerC1) with each containing different key values pairs. All values must be set in order to simulate a scenario, but some of them will be overwritten (as described later) by the simulation framework. Table 4.1 contains a description of all parameters contained in the scenario file

Parameters specifying a value range (e.g UploadRateMin/Max or MaxSleep) are passed to the python3 random number generator that produces numbers uniform distributed in the specifying range.

The categories Peer and PeerC1 share the same type of parameters as they configure the two types of peers implemented in the simulator. Peer is defining the parameters for the original BitTorrent protocol and PeerC1 for the client containing the proposed protocol modifications.

Simulation Files

The simulation file as the scenario file is structured by categories containing peers of keys and values. Appendix A.2 shows a typical Simulation configuration file used by the Simulation Framework to launch 4 iterations with 4 parallel threads running the dynamic_50p_50p1_1000MB.cfg scenario. A complete list of parameters with description is shown in table 4.2.

4.2.3 Generated Statistics and evaluation

A R script using ggplot2 is executed by the Simulation Framework at the end of each scenario simulation to generate statistics. For each scenario simulation a statistic summary will be created that contains all statistic diagrams in a single pdf file. In addition, an ECDF (Empirical Cumulative Distribution Function) and histogram plot of the download time for each peer is generated. This plot contains results of all scenario simulations performed

Table 4.1: Scenario parameters

General	(General scenario parameters)
TorrentSize:	Torrent Size in Bytes
PieceSize:	Piece Size in Bytes (number of Pieces should be between 500 and 2000, higher values need more processing time)
SimEnd:	Specifies the length of the simulation in ticks
SeederUpload:	Upload bandwidth of the seeder in Bytes / second
SeederDownload:	Download bandwidth of the seeder in Bytes / second
logFile:	path to the logfile created by the simulation (*will be overwritten by SimulationFramework*)
logLevel:	defines the level of detail captures in the log file (possible values are DEBUG, INFO, WARN, NONE) (*will be overwritten by SimulationFramework*)
Peer	(Original BitTorrent client)
nInitialPeers:	Number of peers at simulation start
SpawnRate:	Probability to spawn new peer at each tick (in %)
LeaveRate:	Probability to peer leaving swarm after completing download at each tick (in %)
MaxSleep:	Newly created peers will sleep up to MaxSleep ticks after joining a swarm
UploadRateMin:	Minimum upload bandwidth (Bits / second)
UploadRateMax:	Maximum upload bandwidth (Bits / second)
DownloadRateMin:	Minimum download bandwidth (Bits / second)
DownloadRateMax:	Maximum download bandwidth (Bits / second)
PeerC1	(Modified BitTorrent client)
...	*same parameters as the Peer category*

Table 4.2: Scenario parameters

General	(General simulation parameters)
nIterations:	number of Iterations to execute the same scenario
nThreads:	number of simulation to run in parallel
iterationPrefix:	defines a simulation prefix. Will be set as file name prefix to all files generated by this simulation.
scenario:	path to the scenario file
runDirectory:	directory in which log files, csv, and temp files are stored.
randSeedBase:	passed to the python random initialization and incremented by one with each iteration
statsScript:	path to the R script used after simulating a scenario to generate statistic data.
logLevel:	log level for this simulation (possible values are DEBUG, INFO, WARN, NONE)
statsSummaryDir:	path to where statistic summaries for this simulation should be stored

during the same simulation. The plots are generated out of a csv file produced by each scenario simulation containing a tuple of values for each peer in each tick.

Each diagram compares the behaviour of an unmodified BitTorrent client (curve labeled BT) to a client containing the proposed protocol modifications (curve labeled BT_ext). Plots contain data averaged over all peers of the same time, not including the initial seeder.

A set of different diagrams are generated automatically, containing plots of upload/download bandwidth usage, neighbourhood size, number of TFT/OU Slots, download time, number of downloading and completing peers in each tick, and a few more (A.1) contains an example plot).

4.2.4 Implementation Details

In the following section, a description of the innerworking of the most important simulator components is given. Starting with an explanation of the Tick and phase system, followed by the Peer/PeerC1 node, the connection component and finishing with the observer class.

Tick and Phase system

The simulator is operating in what is called "ticks" what correspond to a discrete time frame (e.g. a second) in simulation time. In each tick a series of phases is executed that are used by peers to register callback functions and that then are called on each tick. Table 4.2.4 shows a list of all phases with their corresponding description.

Peer nodes

The behaviour of a peer is simulated by the Peer and PeerC1 class that correspond to a peer client following the normal BitTorrent protocol and a client following the modified BitTorrent protocol.

Peers are spawned by what is called the "PeerFactory" that is executed at the beginning of each tick and evaluates if new peers are to be spawned and is configured by the scenario configuration file.

INIT:	A newly spawned peer is sleeping by default. The init phase checks if the peer is to be woken up.
UPDATE_LOCAL:	First periodic executed phase; Checks if the peer has completed the file download and updates piece list locally.
UPDATE_GLOBAL:	Updates all connection of the peer depending on the piece list updates in each neighbouring peer done in the previous phase.
LOGIC:	Executes the peer logic responsible for slot allocation using TFT and OU
FILETRANSFER:	simulates the data transfer done in the current time frame
CONCLUSION:	used to keep track on the bandwidth usage of the current tick and prints log messages summarizing the current tick
STATISTICS:	Internal phase used to gather information on each peer and to generate entry in simulation log (csv file)
SIMULATION_END:	executed once at the end of the simulation for the simulator internal cleanup processes

Table 4.3: Simulation phases

After spawning, the peer remains in a sleep state which duration is defined on peer creation (by the PeerFactory). This is done because as the simulator is processing discrete time periods, without a sleep period all peers spawn in the same tick would have a semi synchronized behaviour that leads to artificial effects not present in a real Peer to Peer system. At wake up, a peer connects to the tracker (there is only one tracker per simulation that is indicated in the torrent file passed to the peer by the PeerFactory) and receives a list of random chosen peers part of the swarm. The client then connects to these peers using a connection class object for each peer and follows the normal BitTorrent protocol.

The connection class is interconnecting peers, giving them information on the pieces offered by the other side, sets connection attributes like interest and choke status. The peer logical defines the maximum bandwidth usage available to each connection and keeps track of the actual used bandwidth at the end of each tick.

In addition, each peer has a torrent object which is used to keep track on the pieces downloaded already and which are still to be downloaded.

Observer

In order to generate statistics on the process of the simulation, a global observer is implemented, which is called in each tick during the "SIMULATION_END" phase in order to gather information about each peer. For each peer an entry to the simulation log is added in csv format that contains information about the current state of the peer. It includes information like the type of peer, its spawn tick, the current number of neighbours, the number of used OU and TFT slots, maximum upload/download bandwidth, currently used upload/download bandwidth and some additional bits of information that will be used by the statistic script to generate statistical diagrams.

Chapter 5

Results

In this chapter, the results of a series of different simulations are presented and the performance of the modified BitTorrent client is evaluated.

The chapter is structured in the following way. Section 5.1 will explain the different scenarios simulated, section 5.2 will present the results of each simulation and section 5.7 will compare and evaluate the two clients based on the simulation results.

5.1 Simulation scenarios

The different simulations performed are variations of three main parameters (churn, swarm size and torrent size).

The first parameter defines if the swarm is dynamic in size, hence if new peers join the swarm during the simulation and peers that have completed the download remain for a while as seeders in the swarm, or if it is a static system in which all peers are present, at the start of the simulation and peers leave the swarm as soon as they complete their download. Where the former method (dynamic system) appears to be closer to a real-world swarm behaviour, the later (static system) is often found in the literature and therefore both shall be examined here.

The second interesting simulation factor is the swarm size where two kinds of swarm sizes shall be evaluated. Small swarms with only 100 peers that could be found with private trackers, and big swarms with about 500 peers that represent public trackers.

At last the file size of the torrent is changed, differentiating between the exchange of small files of 10MB, medium files of 350MB and large files of 1000MB.

Table 5.1 gives an overview with description about the different simulations performed and table 5.1 lists all Scenario Parameters.

For each file size	small, medium and large
Dynamic private Tracker:	Dynamic system where constantly new peers join the swarm but the swarm size is around 100 peers
Dynamic public Tracker:	Dynamic system with joining peers but a much larger swarm size of around 500 peers
Static private Tracker:	Static system with one seeder and 100 peers. No new peers join the swarm and completed peers leave immediately.
Static public Tracker:	Static system with one seeder and 500 peers. No new peers join the swarm and completed peers leave immediately.

Table 5.1: Simulation Scenarios

5.1.1 Swarm Size

It was found that the swarm size (i.e. the number of peers and seeders in a swarm) has an important influence on the download performance. In general it was found that as the swarm size increases, the individual download time increases as well. This makes sense as each peers join the swarm is in a way consuming download bandwidth, and offering upload bandwidth. As the peers consume more than they provide, the download of each individual peer suffers degeneration. This effect is more complex though because as seeders remain in a swarm, they consume nothing and only provide. Therefore the download performance of a general downloading peer depends on the swarm size as well as the ratio of seeders to leechers. In the static simulations where the *SpawnRate* is set to 0, the swarm size is easily controlled by the number of initial peers and the *LeaveRate* (which in all static simulations is set to 1.0 and therefore makes all peers leave the swarm as soon as they complete

	Scenarios											
	Static						Dynamic					
	Private Tracker			Public Tracker			Private Tracker			Public Tracker		
	small	medium	large	small	medium	large	small	medium	large	small	medium	large
Parameters												
TorrentSize (MB):	10	350	1000	10	350	1000	10	350	1000	10	350	1000
PieceSize:(kB)	10	500		10	500		10	500		10	500	
SimEnd (sec):	1800	2500		3000			2000	3000	4000	2000	2500	5000
SeederUpload:	640kB	100Mb		640kB	100Mb		640kB	100Mb		640kB	100Mb	
Peer & PeerC1												
MaxSleep:	30											
UploadRateMin :	8kB	1Mb		8kB	1Mb		8kB	1Mb		8kB	1Mb	
UploadRateMax:	32kB	5Mb		32kB	5Mb		32kB	5Mb		32kB	5Mb	
DownloadRateMin:	32kB	10Mb		32kB	10Mb		32kB	10Mb		32kB	10Mb	
DownloadRateMax:	128kB	50Mb		128kB	50Mb		128kB	50Mb		128kB	50Mb	
LeaveRate:	1.0						0.003	0.005	0.0015	0.01		
Peer												
nInitialPeers:	100, 90, 50, 10, 0			500, 450, 250, 50, 0			30, 27, 15, 3, 0			200, 180, 100, 20, 0		
SpawnRate:	0.0						0.12[a]	0.1[a]	0.035[a]	0.7[a]	0.5[a]	0.15[a]
PeerC1												
nInitialPeers:	0, 10, 50, 90, 100			0, 50, 250, 450, 500			0, 3, 15, 27, 30			0, 20, 100, 180, 200		
SpwanRate:	0.0						-[a]					

Table 5.2: Overview of scenarios and parameters

[a]The SpawnRate is proportional to the ratio of peer and peerC1 nodes. The values referenced here are taken for pure swarms only. Meaning that for a scenario where the peers types are mixed 1 to 1, the SpawnRate of each type is half of the value stated in the table.

their download). For the dynamic simulations the swarm size is not that easily to predict. The *SpawnRate* is set to values higher than 0.0 (as in all dynamic cases) and depending on the performance of the algorithms, as well as a random factor, the swarm size variates.

For these reasons the format in that the ratio and number of peers is described for each simulation (for example 50p_50pc1 says there will be about 50 peers running one and 50 peers running the other BT algorithm in the swarm) is not perfectly reflecting the number of peers of each kind. The *Peer Count* plot that is generated for each simulation and can be found in the appendix lists in more detail the actual number of peers for each simulation and how they variate during the progress of the simulation.

5.2 Simulation results

In the following section, the important results of each of the simulation scenarios described in the previous section are presented. As it would take too much space to present all data gathered during each simulation, only the most relevant data is presented. This includes for each simulation the ECDF plot (which makes for the most important single metric of the download performance of the algorithm), the peer status plot (titled *Total and completed peers*), upload usage, average number of OU and TFT slots,

Each simulation was run with a different ratio of peers running the normal BitTorrent protocol and peers running the modified BitTorrent protocol. The number of peers of each kind is specified in the title of every graphic, where the suffix *p* stands for an unmodified and the suffix *pc1* for a modified client. If not specified differently, each simulation was executed for (100% peers, 0% modified peers) , (90p , 10pc1), (50p, 50pc1), (10p,90pc1), (0p,100pc1). By doing so it is possible to estimate the behaviour of systems with different popularity of the client.

Multiple scenario instances are run for single simulations (if not mentioned differently four instances have been executed) differing in the initialisation of the random number generator. As for size constraints it would be impossible to include the simulation results for each instance of each sce-

nario [1] it was chosen to only include the results of the first instance, where the ECDF plot is different as the results of all instances can easily be fused into a single plot and therefore has been done.

To further make it easier to compare the algorithm performance over different simulations, it has been decided to move all results but the ECDF plots into the appendix. The ECDF plot is therefore the first and main metric to compare the system performance.

[1] it is to remember that each simulation, from there are nine, five variations are simulated with different ratio of normal and unmodified peers, and on top of this four instances each. Making for 9*5*4 = 180 instances. The interested reader is therefore encouraged to get in contact with the author in order to receive a copy of all the simulation results.

5.3 Static private Tracker

In this section the protocols are compared in a scenario of a small private tracker with about 100 peers and a static setup where all peers are present at the start of the simulation and leave the swarm as soon as they have completed their download.

All parameters for this scenario can be found in table 5.1. The ECDF plots are shown at Figure 5.1 (small File size), Figure 5.7 (medium File size) and Figure 5.13 (large File size). Additional plots can be found in appendix at B.1.1 , B.1.2 and B.1.3.

The overall picture shows a very similar behavior of both algorithms, with peers running the unmodified BitTorrent protocol slightly downloading faster on average than peers using the modified protocol.

A note worthy difference can be seen at the borders of the ECDF plots for the fastest and the slowest peers. Peers running the modified protocol use more OU slots at the beginning of the simulation and therefore altruistically spread the important first pieces to other peers that have no way to cooperate in TFT slots unless they have something to share. This is visible in most of the scenarios, but most clearly in the pure swarms (e.g 100p and 100pc1). As a result the fastest peers (e.g those that complete the download first) running the modified protocol tend to take longer to complete their download than peers running the unmodified protocol, but as a consequence support slower peers (e.g those peers that complete their download as one of the last) by supplying them with the important first pieces and by staying longer in the swarm. As can be seen in the plots B.1.1 showing the local piece availability of the swarm, the modified protocol tends to keep the divergence in local piece availability smaller than the unmodified protocol so that all peers in the swarm accumulate pieces at a more or less similar rate.

The importance of this effect clearly depends on the total download time and therefore is a function of the torrent size. This can be seen when comparing scenarios of different file size (like 10p10MB, 350p10MB and 1000p10MB). As the total download time increases, the effect of higher usage of OU slots and altruistic spread of first pieces, as well as the longer download time of fast peers, gets less weight and the results of the effect

decrease. This can be seen as the download time of the first and the last peers of both kind are almost the same.

Figure 5.1: **static,10MB ECDF**

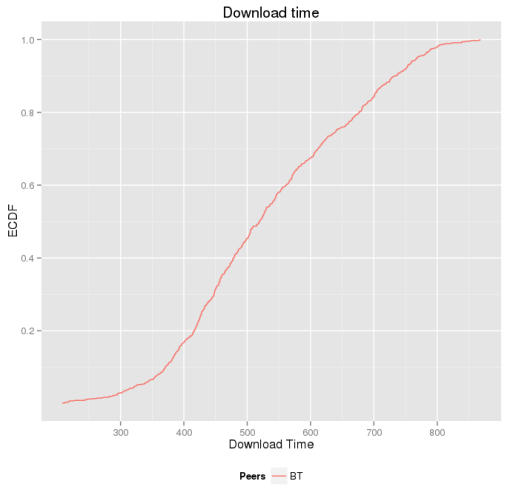


Figure 5.2: static,100p,0pc1,10MB

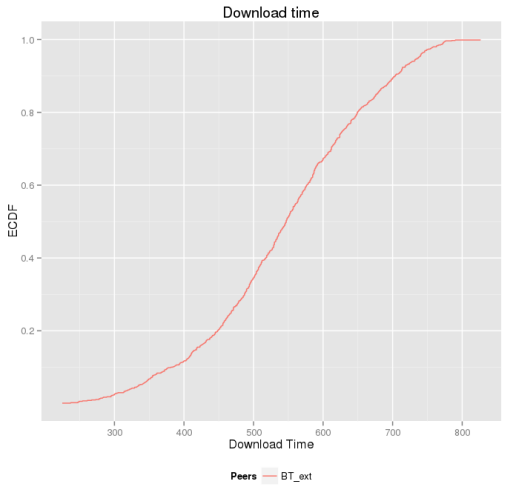


Figure 5.3: static,0p,100pc1,10MB

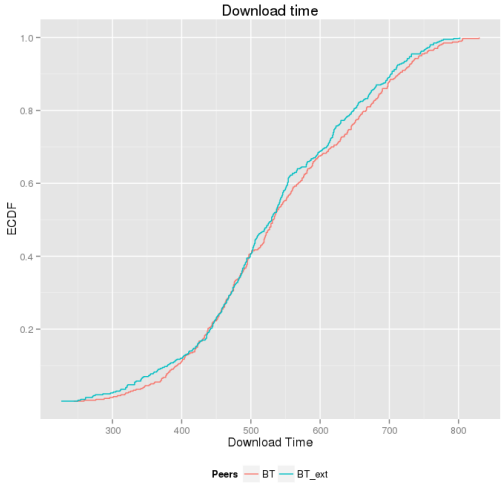


Figure 5.4: static,50p,50pc1,10MB

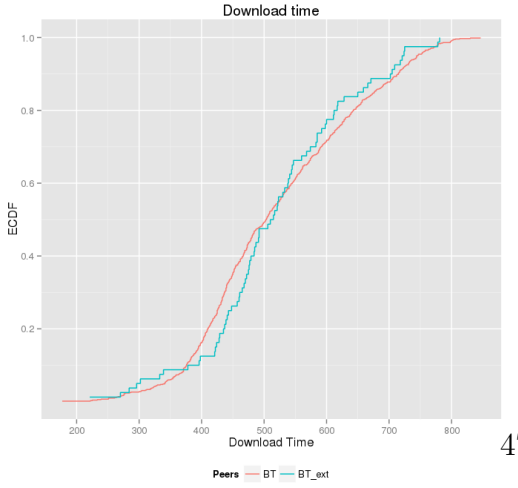


Figure 5.5: static,90p,10pc1,10MB

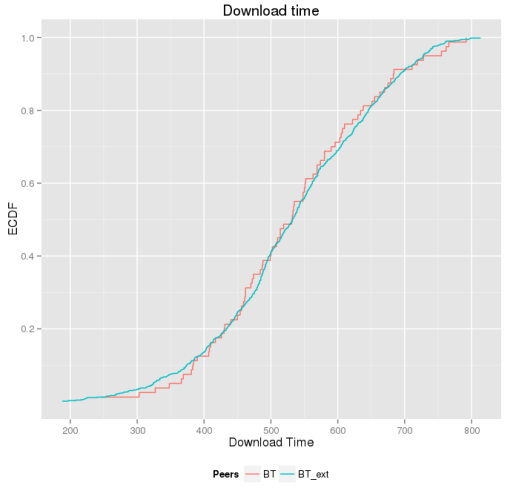


Figure 5.6: static,10p,90pc1,10MB

Figure 5.7: **static,350MB ECDF**

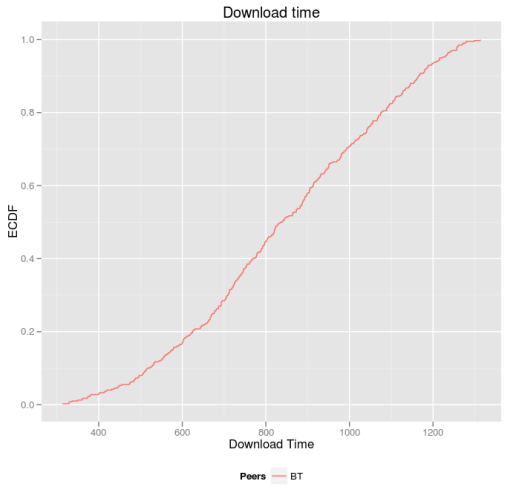


Figure 5.8: static,100p,0pc1,350MB

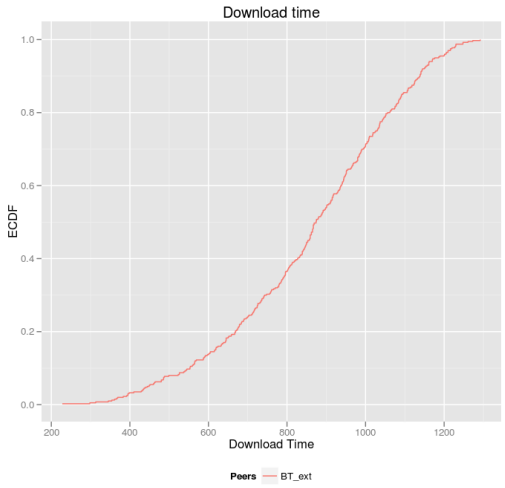


Figure 5.9: static,0p,100pc1,350MB

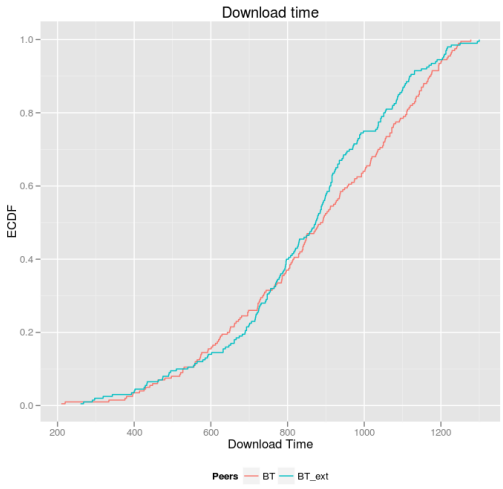


Figure 5.10: static,50p,50pc1,350MB

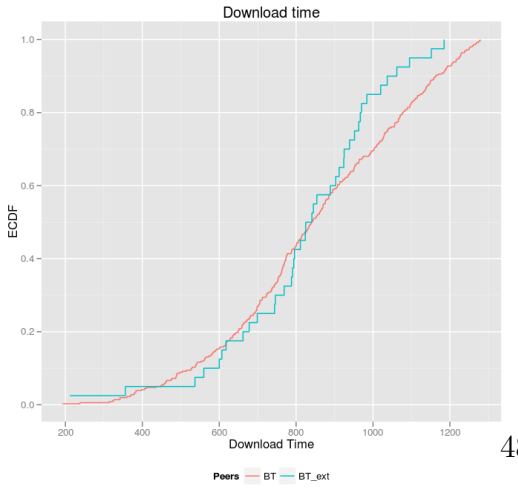


Figure 5.11: static,90p,10pc1,350MB

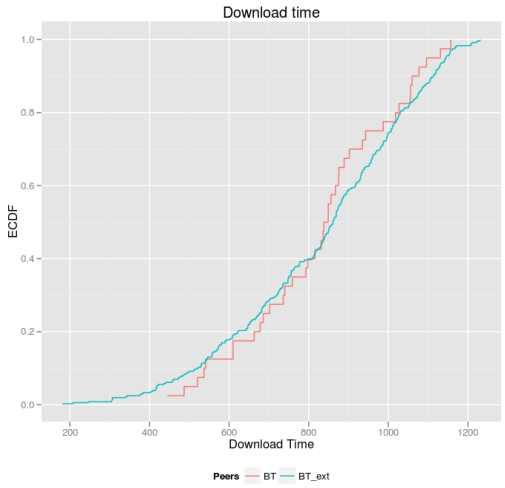


Figure 5.12: static,10p,90pc1,350MB

Figure 5.13: **static,1000MB ECDF**

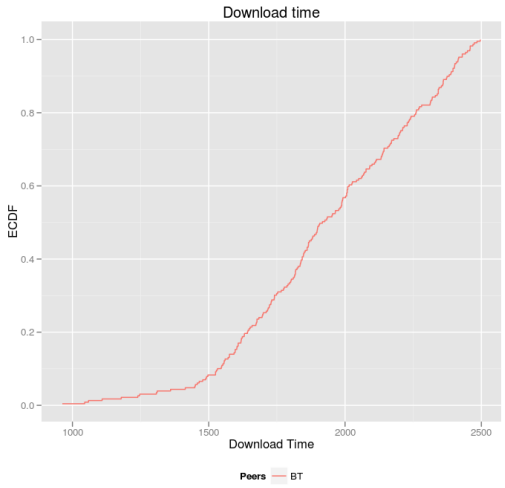


Figure 5.14: static,100p,0pc1,1000MB

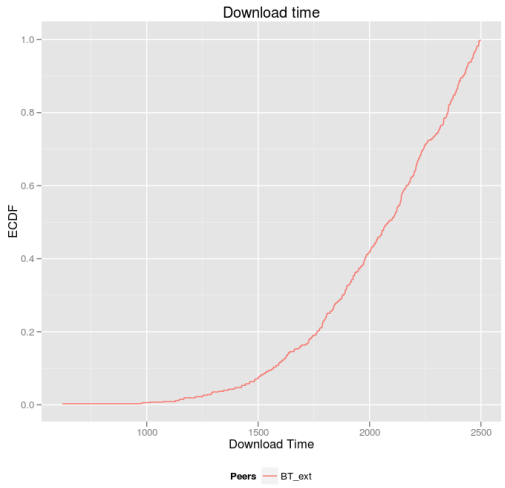


Figure 5.15: static,0p,100pc1,1000MB

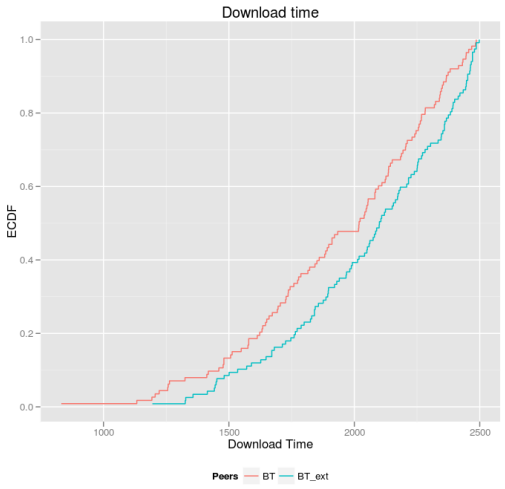


Figure 5.16: static,50p,50pc1,1000MB

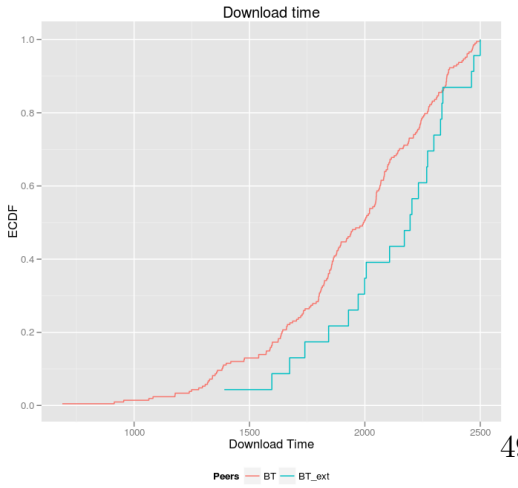


Figure 5.17: static,90p,10pc1,1000MB

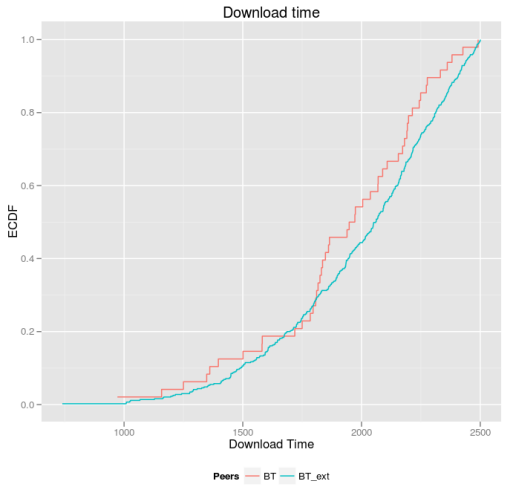


Figure 5.18: static,10p,90pc1,1000MB

5.4 Static public Tracker

In this section the protocols are compared in a scenario with a public tracker that manages a swarm of about 500 peers in a static system where no new peers appear after the simulation starts.

All parameters for this scenario can be found in table 5.1. The ECDF plots are shown at Figure 5.19 (small File size), Figure 5.25 (medium File size) and Figure 5.31 (large File size). Additional plots can be found in appendix at C.1.1 , C.1.2 and C.1.3.

Similar to the private Tracker scenario 5.3, both protocols show comparable good results. Peers running the unmodified protocol still have a slight benefit over peers that implement the modified protocol, and the effect that increases download time for fast peers and reduces download time for slow peers (as mentioned in the previous section) remains.

Because of the bigger swarm size (500 peers over 100 in the previous scenario) it is interesting to compare the behaviour of each protocol in a pure scenario (like the 500p or 500pc1) scenarios to a mixed scenario (like the 250p250pc1) in order to understand how the protocols interact with each other and if there is some kind of bias or special effect when mixing peers of the two kind.

It is consistent through out all scenario that the behaviour of both protocols remains almost exactly the same in pure swarms (where there are only peers of the same kind) as in mixed swarms. This can be most easily seen in the download time (e.g. ECDF plots) where the distribution of download time between a pure scenario and a mixed ones changes only slightly. It does not matter if the ratio between peer types is big or if the number of types of each peer is equal. Both protocols show a very stable and settled behaviour. There does not seem to be imbalance or exploiting effect taking place which would cause one of both types to have an important benefit of the other and therefore reduce its own download time to the costs of the other.

The reader is reminded though that in this simulation all peers behave as described by the protocol specifications (e.g with good intentions). Previous work [20] has shown that TFT is an effective method to penalize FreeRiders and therefore make BitTorrent resilient against this form of misbehaving

nodes, but because of the time limitations on this work, the effects caused by peers with bad intentions where on the modified BT protocol where not simulated. It can be assumed though that FreeRiders in a swarm of peers with the modified BT protocol would have a benefit over their counter parts in a swarm of peers running the unmodified protocol, as the former relies more on OU and therefore can be easier abused by not cooperative peers. Similar applies to strategical peers like BitTyrant [24], although it is difficult to say if they have it easier or equally difficult to make abuse peers running the modified as the unmodified protocol.

Figure 5.19: **static,10MB ECDF**

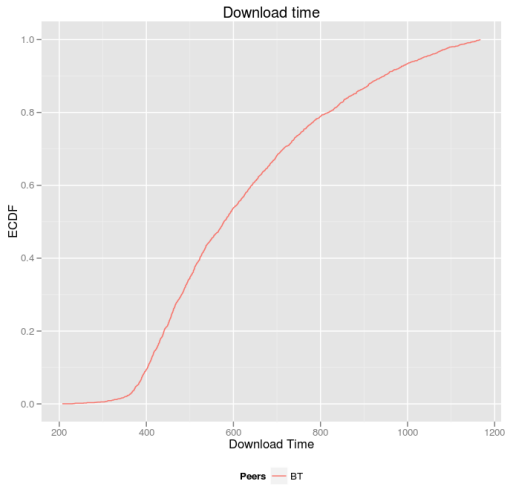


Figure 5.20: static,500p,0pc1,10MB

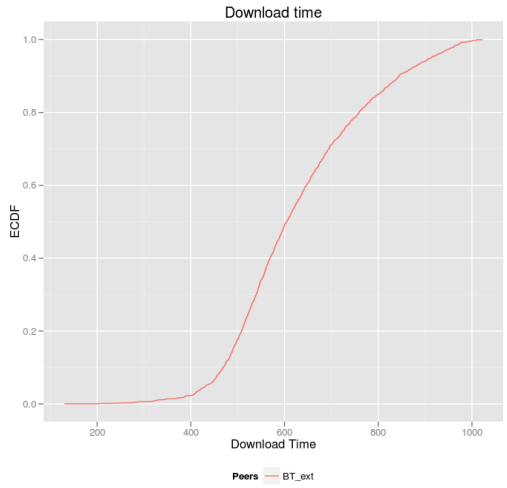


Figure 5.21: static,0p,500pc1,10MB

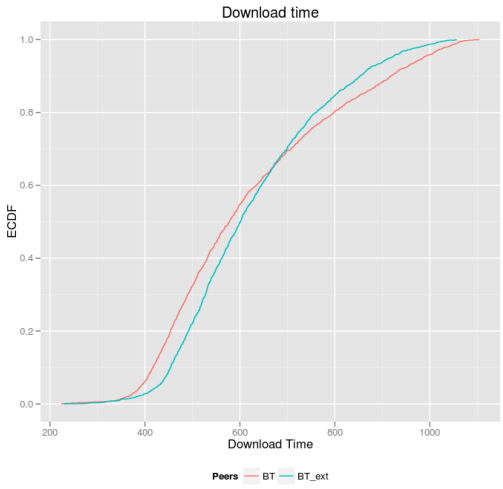


Figure 5.22: static,250p,250pc1,10MB

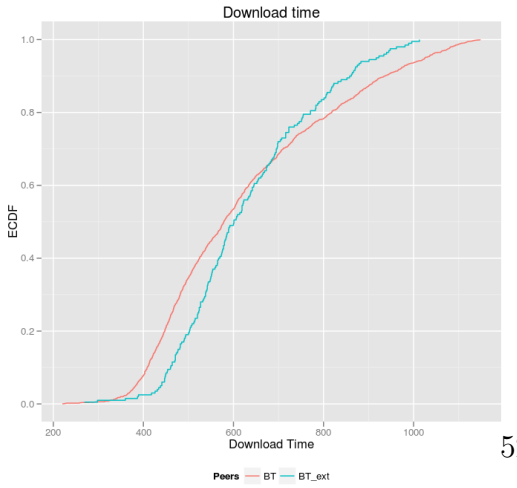


Figure 5.23: static,450p,50pc1,10MB

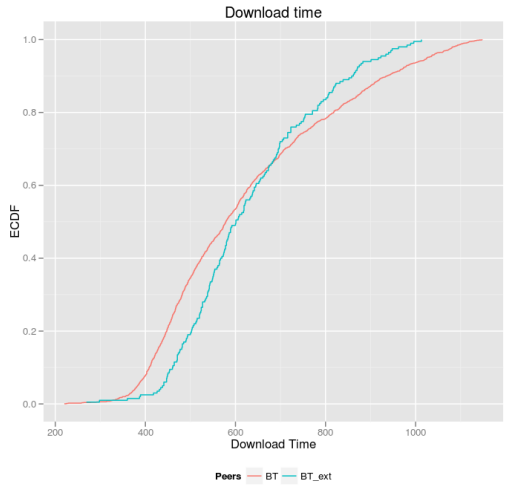


Figure 5.24: static,50p,450pc1,10MB

Figure 5.25: **static,350MB ECDF**

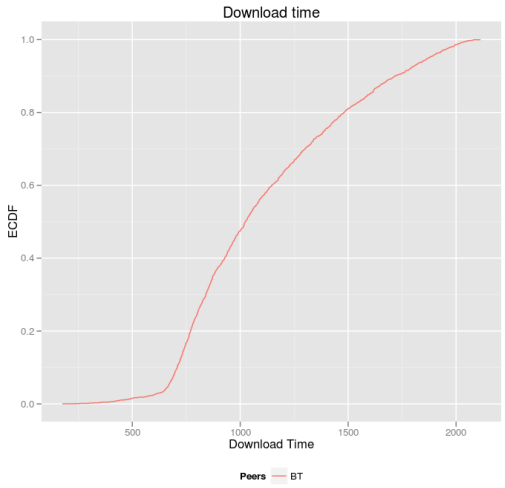


Figure 5.26: static,500p,0pc1,350MB

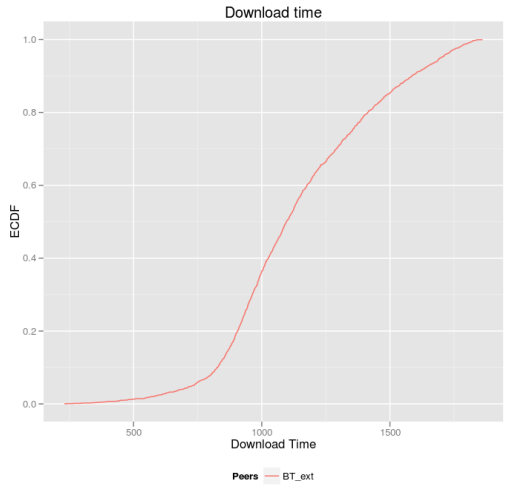


Figure 5.27: static,0p,500pc1,350MB

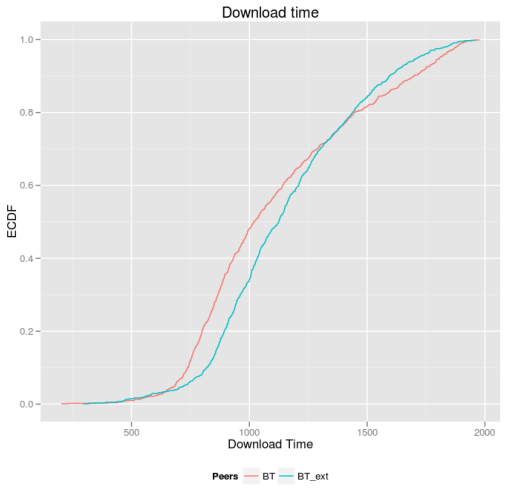


Figure 5.28: static,250p,250pc1,350MB

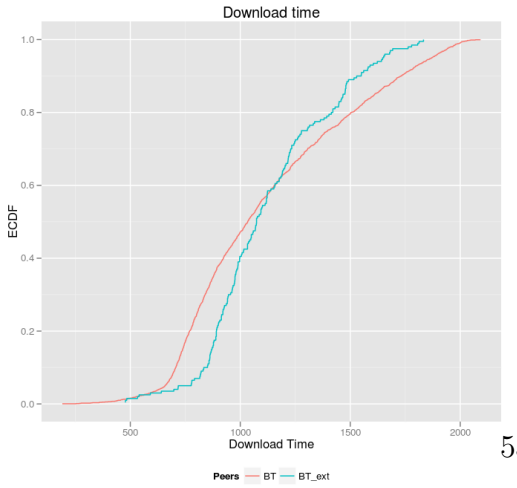


Figure 5.29: static,450p,50pc1,350MB

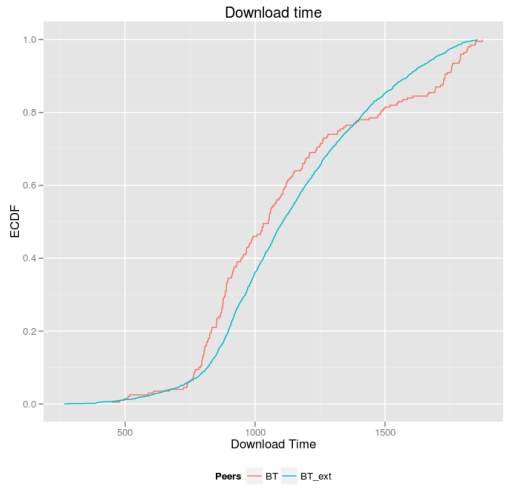


Figure 5.30: static,50p,450pc1,350MB

Figure 5.31: **static,1000MB ECDF**

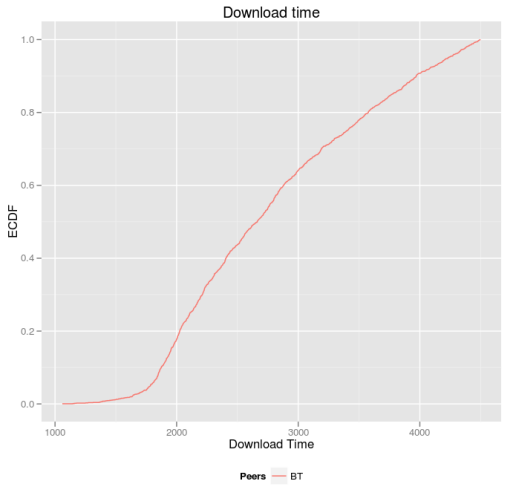


Figure 5.32: static,500p,0pc1,1000MB

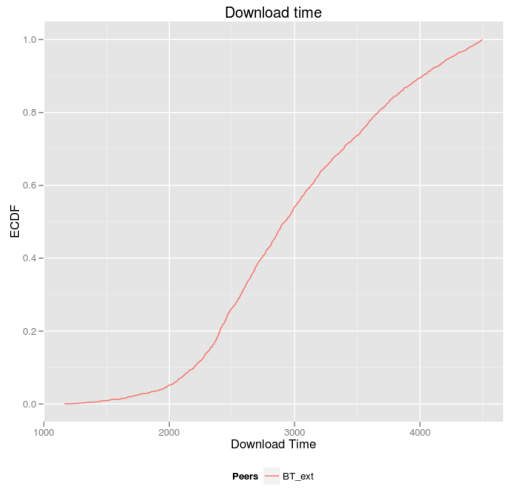


Figure 5.33: static,0p,500pc1,1000MB

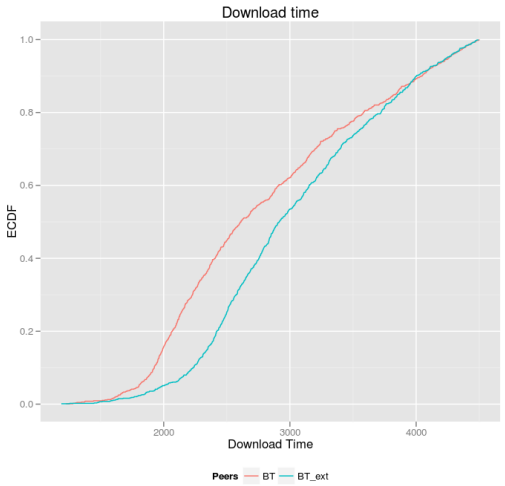


Figure 5.34: static,250p,250pc1,1000MB

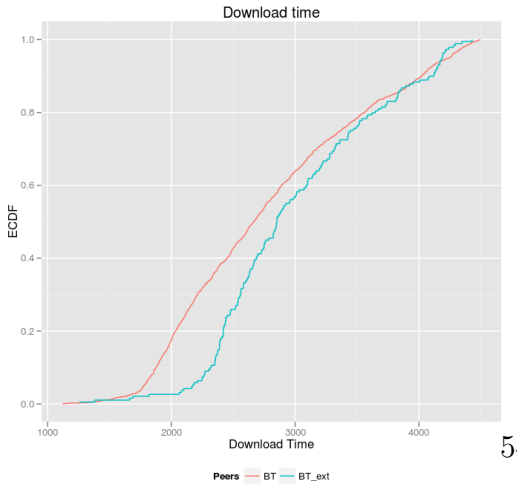


Figure 5.35: static,450p,50pc1,1000MB

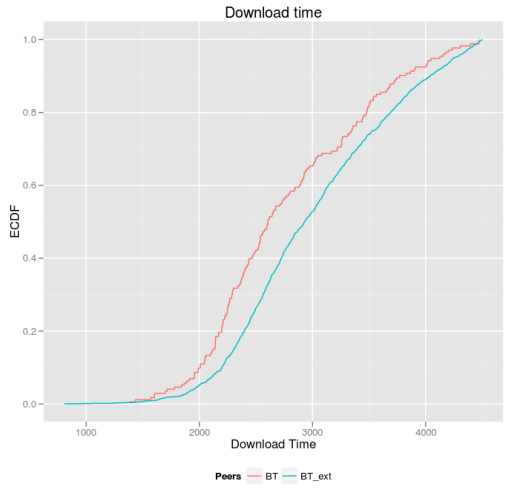


Figure 5.36: static,50p,450pc1,1000MB

5.5 Dynamic private Tracker

In this section the protocols are compared in a scenario of a small private tracker with about 100 peers and a dynamic setup where new peers are spawned with a probability defined by the simulation parameter *SpawnRate*. After completing their download each round a peer evaluated if it is to remain in the swarm acting as a seeder, or leave, governed by the simulation parameter *LeaveRate*.

All parameters for this scenario can be found in table 5.1. The ECDF plots are shown in Figure 5.37 (small File size), Figure 5.43 (medium File size) and Figure 5.49 (large File size). Additional plots can be found in appendix at D.1.1 , D.1.2 and D.1.3.

It is very interesting to see the difference between the previously discussed static simulations and the following two dynamic simulations. As was mentioned in the static private tracker scenario 5.3 there was found to be only a minor difference between the modified and the traditional BT protocol for small swarms. This is a strong contrast to the results acquired for a swarm of similar size but a dynamic peer spawning behaviour. In the ECDF plots presented, it is clearly visible that the difference in download performance is strongest for short download times (e.g. the 10MB torrents) and declines with longer download times (e.g the 350MB and 1000MB torrents).

In addition there is a strong contrast to the results achieved in mixed scenarios in the static scenario as described in 5.4. Where in the static scenario both algorithms perform independently, it is clear that in the corresponding dynamic scenario, the modified BT protocol is improving tremendously with corresponding deficit of peers running the traditional protocol. This is most visible in the short simulations (e.g 10MB Torrents) where a small number of peers using the modified BT protocol (e.g. 90p10pc1) not only clearly outperform their counterparts, but even making for the shortest overall download time. As the number of peers running the modified protocol increases, the difference between peers running different protocol declines, but as well as the absolute download time. The difference between both kind of peers continues to decline with longer download times (e.g. 350MB and 1000MB Torrents) but is consistent.

Although the modified BT protocol in these scenarios seems to perform better than the traditional protocol it is to remember that this is only true for the mixed scenarios. Besides the dynamic,90p,10pc1,10MB scenario the average peer download time of swarms consisting only of peers running the traditional protocol (e.g 100p10MB, 100p350MB and 100p1000MB) give the best results.

Figure 5.37: **dynamic,10MB ECDF**

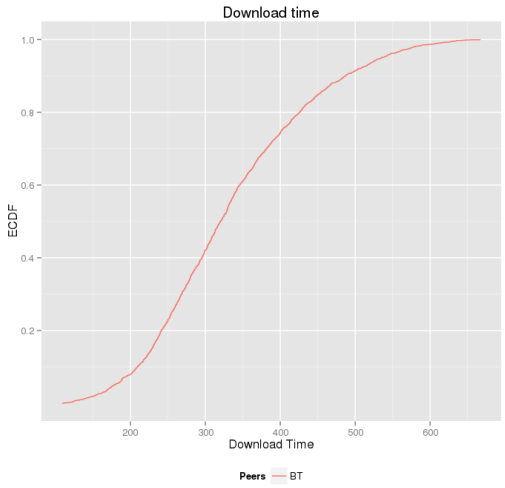


Figure 5.38: dynamic,100p,0pc1,10MB

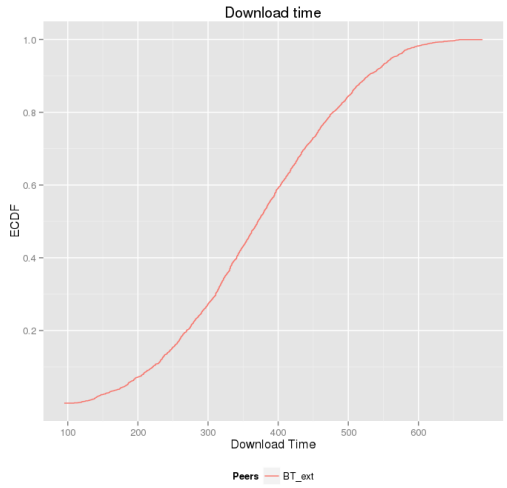


Figure 5.39: dynamic,0p,100pc1,10MB

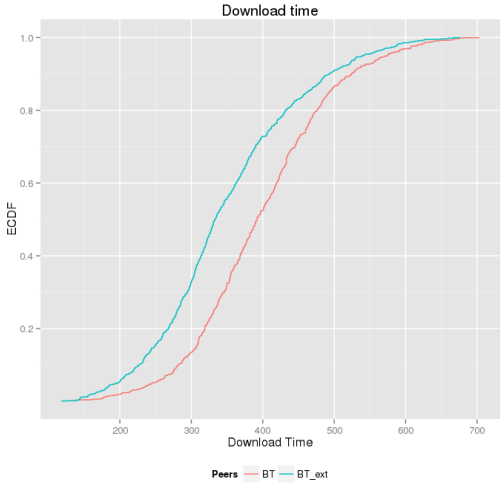


Figure 5.40: dynamic,50p,50pc1,10MB

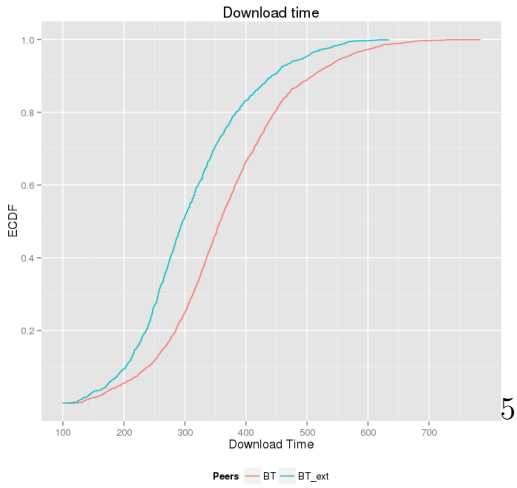


Figure 5.41: dynamic,90p,10pc1,10MB

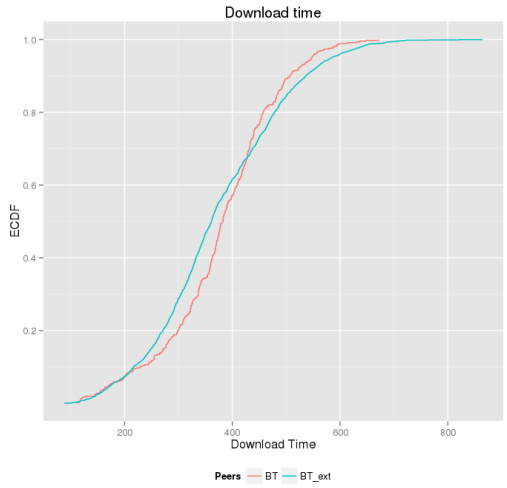


Figure 5.42: dynamic,10p,90pc1,10MB

Figure 5.43: textbfdynamic,350MB ECDF

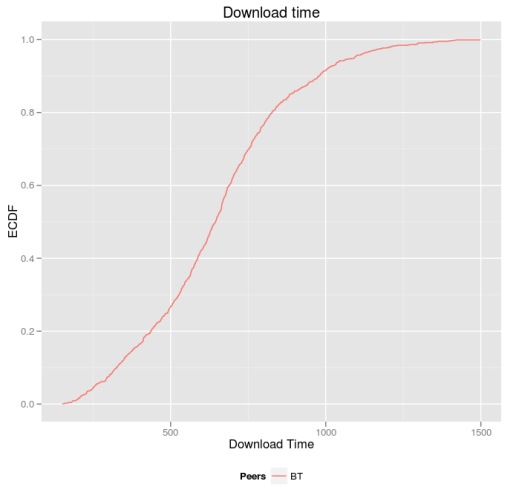


Figure 5.44: dynamic,100p,0pc1,350MB

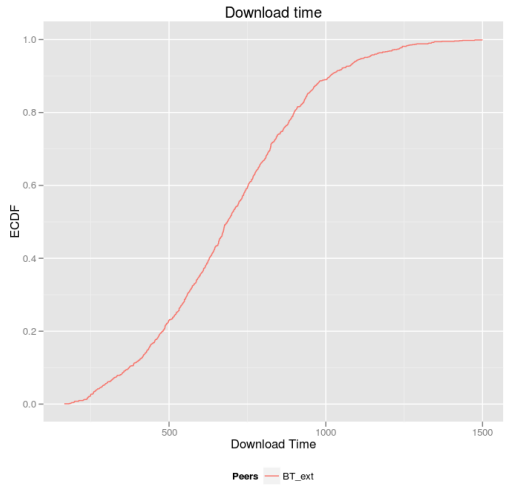


Figure 5.45: dynamic,0p,100pc1,350MB

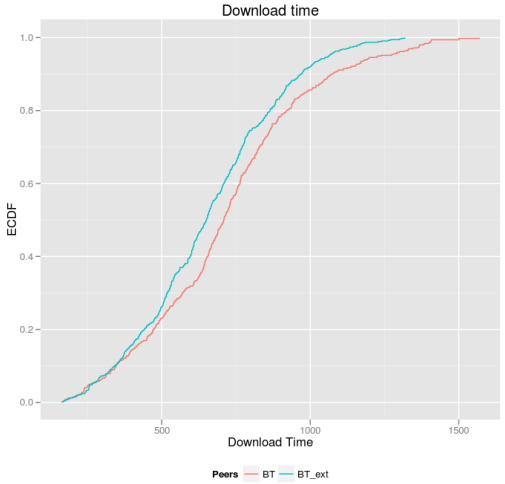


Figure 5.46: dynamic,50p,50pc1,350MB

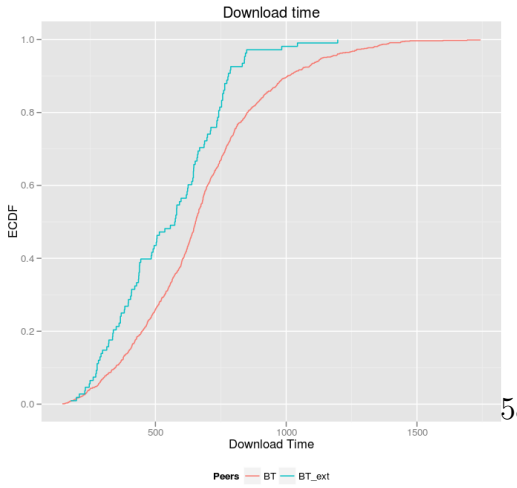


Figure 5.47: dynamic,90p,10pc1,350MB

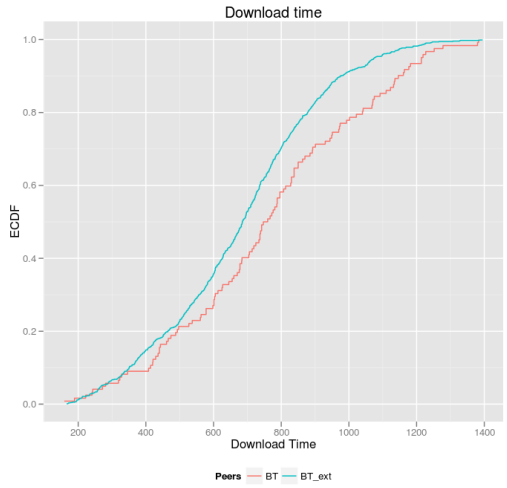


Figure 5.48: dynamic,10p,90pc1,350MB

Figure 5.49: **dynamic,1000MB ECDF**

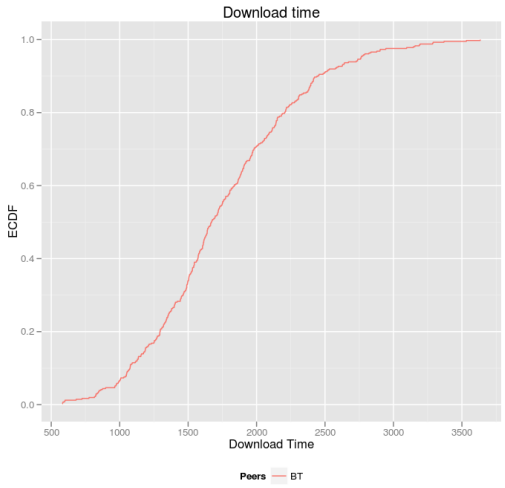


Figure 5.50: dynamic,100p,0pc1,1000MB

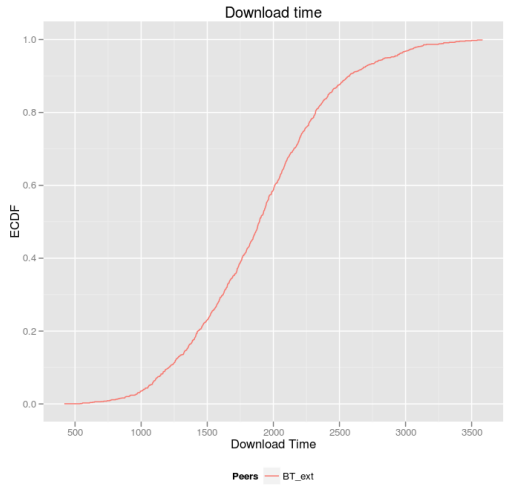


Figure 5.51: dynamic,0p,100pc1,1000MB

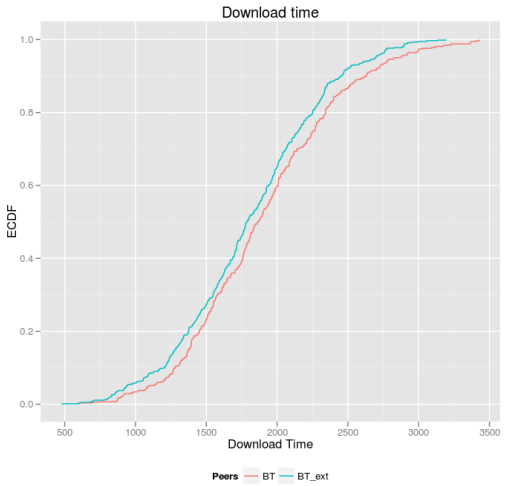


Figure 5.52: dynamic,50p,50pc1,1000MB

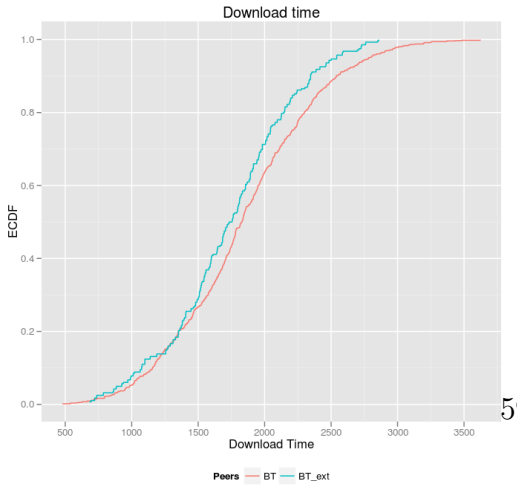


Figure 5.53: dynamic,90p,10pc1,1000MB

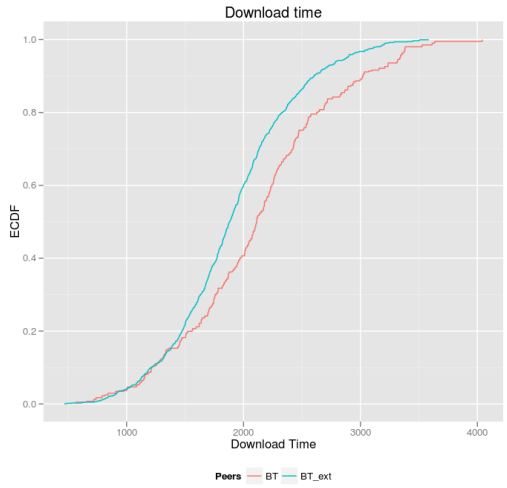


Figure 5.54: dynamic,10p,90pc1,1000MB

5.6 Dynamic public Tracker

In this section a final and most realistic scenario is simulated. The swarm consists of about 500 peers that join and after completion leave the swarm depending on the simulation parameters SpawnRate and *LeaveRate*.

All parameters for this scenario can be found in table 5.1. The ECDF plots are shown at Figure 5.55 (small File size), Figure 5.61 (medium File size) and Figure 5.67 (large File size). Additional plots can be found in appendix at E.1.1 , E.1.2 and E.1.3.

Two consistent effects that already have been described in the previous scenarios dominate this simulation.

For one the difference in average download time of the two algorithms decline with increasing total download time (e.g. the 350MB and 1000MB Torrents) and secondly, the modified BT protocol remains to reduce the average download time of the slowest peers and increase the average download time of the fastest peers in a swarm.

As described in the previous scenario 5.5, peers running the modified protocol benefit more of a mixed swarm than peers running the traditional protocol. What was not visible in the previous scenario was that this effect seems to decline with an increasing absolute download time. Where in the 10MB Torrent scenario the modified protocol clearly outperforms the traditional one, the situation is more complex in the 350MB and 1000MB Torrent simulation. What remains valid is the fact that the download time for the slowest peers is reduced through the modified protocol, but for peers with a medium download capacity in some scenarios have a lower average download time using the traditional protocol. This for example can be seen in the mixed 350MB file scenario (i.e Figure E.52) where both algorithms have a similar performance for the fastest and the slowest peers, but in the medium capacity peers (15% - 83%) can complete their download faster with the traditional protocol. For the simulation on big files (i.e Figure E.82) on the other hand a smaller range of peers have a benefit of using the traditional protocol and the slowest peers clearly are benefiting from the modification applied.

In addition both algorithms tend to converge on the average download time of their fastest peers in mixed swarms. Meaning that depending on the

ratio of modified and unmodified peers, the average download time tends to equal the value of the corresponding pure swarm simulation.

Figure 5.55: **dynamic,10MB ECDF**

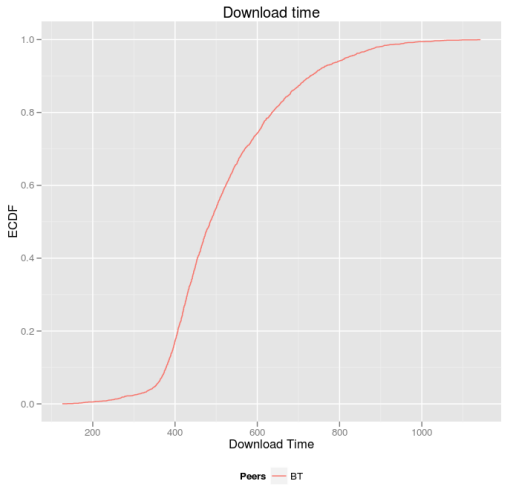


Figure 5.56: dynamic,500p,0pc1,10MB

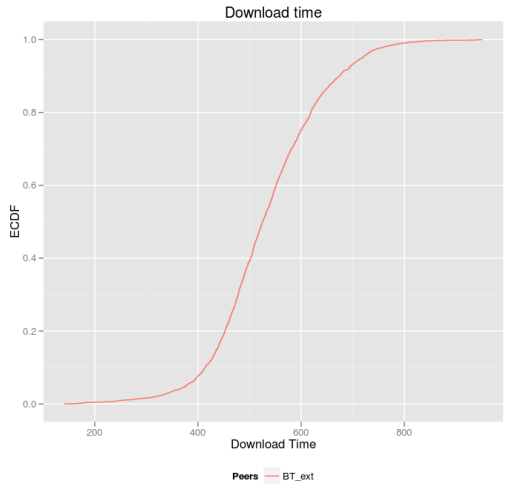


Figure 5.57: dynamic,0p,500pc1,10MB

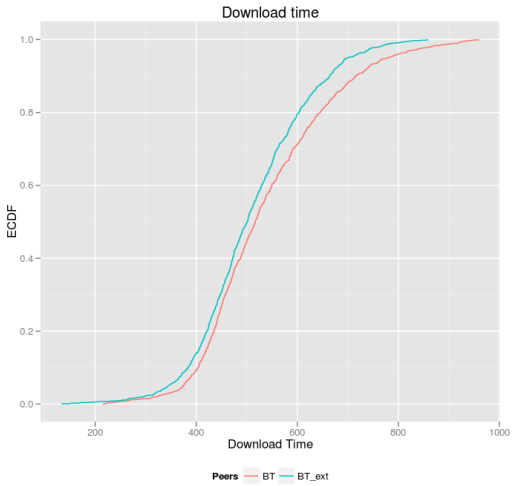


Figure 5.58: dynamic,250p,250pc1,10MB

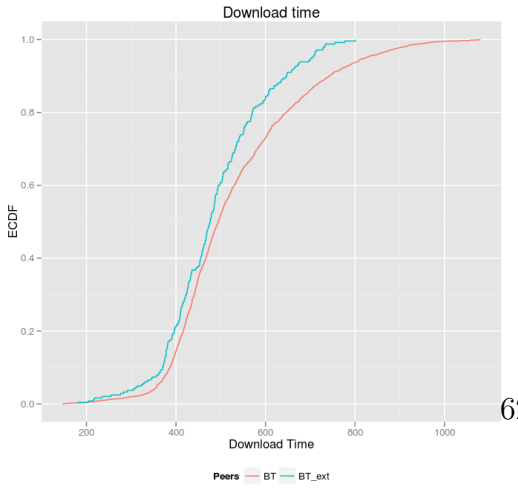


Figure 5.59: dynamic,450p,50pc1,10MB

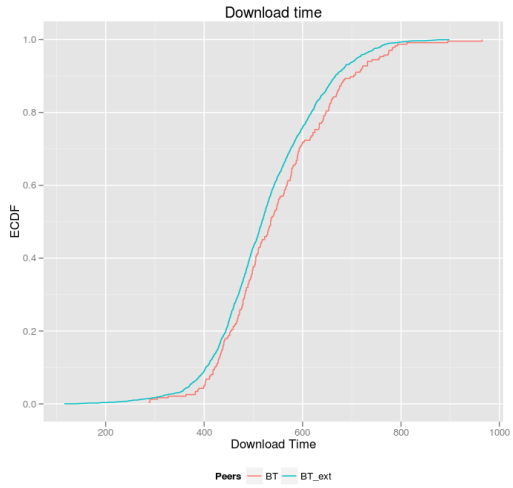


Figure 5.60: dynamic,50p,450pc1,10MB

Figure 5.61: **dynamic,350MB ECDF**

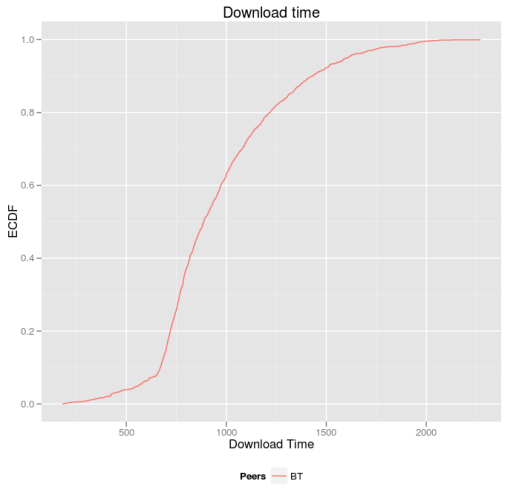


Figure 5.62: dynamic,500p,0pc1,350MB

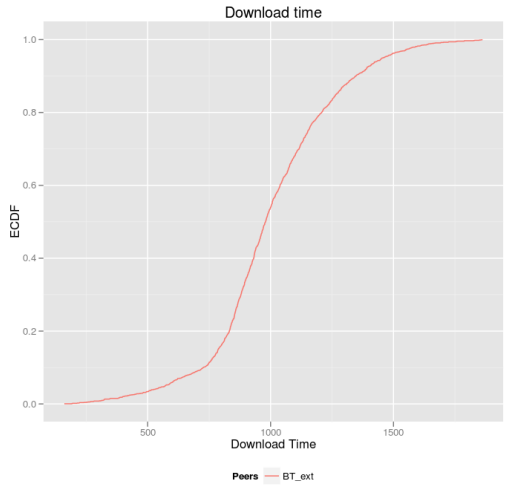


Figure 5.63: dynamic,0p,500pc1,350MB

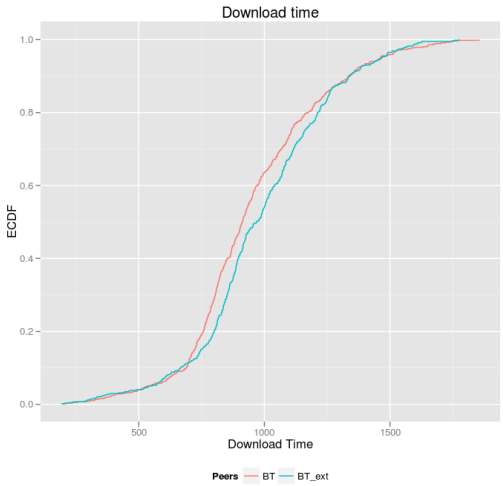


Figure 5.64: dynamic,250p,250pc1,350MB

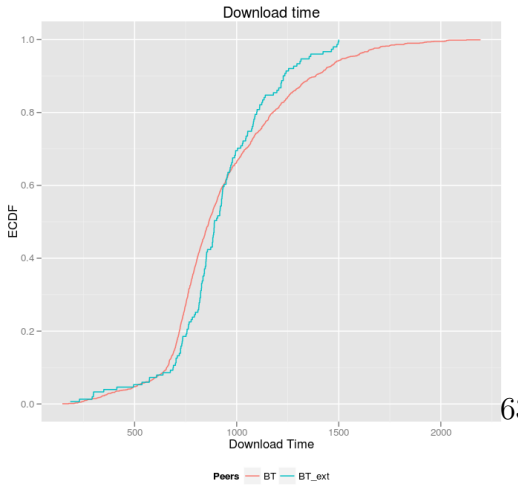


Figure 5.65: dynamic,450p,50pc1,350MB

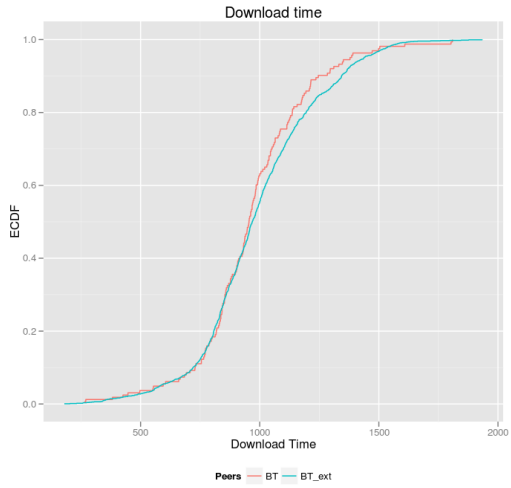


Figure 5.66: dynamic,50p,450pc1,350MB

Figure 5.67: **dynamic,1000MB ECDF**

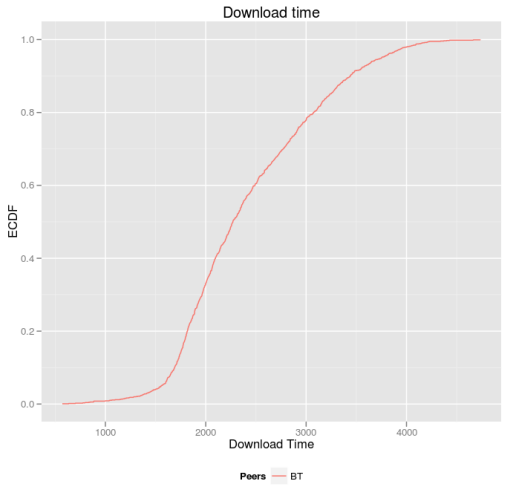


Figure 5.68: dynamic,500p,0pc1,1000MB

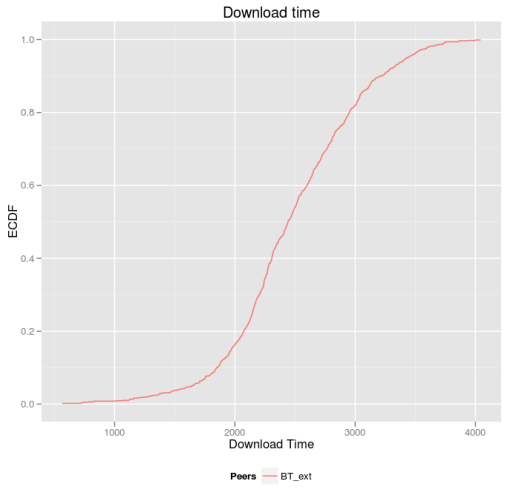


Figure 5.69: dynamic,0p,500pc1,1000MB

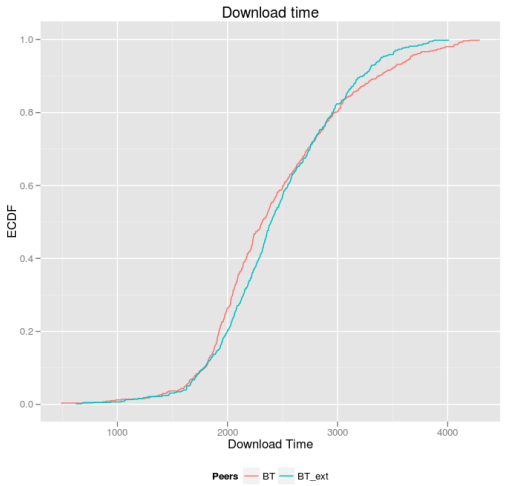


Figure 5.70: dynamic,250p,250pc1,1000MB

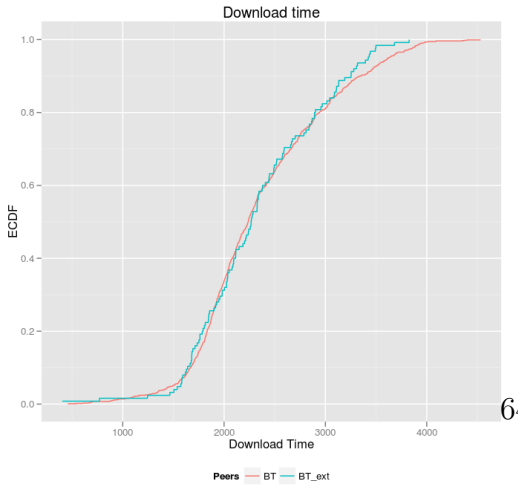


Figure 5.71: dynamic,450p,50pc1,1000MB

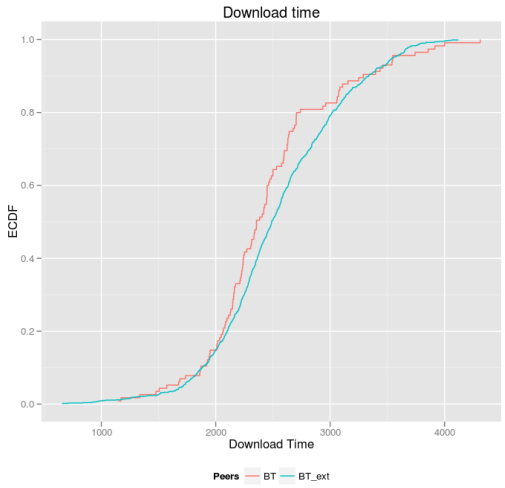


Figure 5.72: dynamic,50p,450pc1,1000MB

5.7 Over all comparison

In this section a summary of the results found during the simulation of all four type of scenarios is given. This includes a conclusion about the difference in performance for both algorithms, as well as what have been found important simulation parameters.

5.7.1 Simulated protocols

Over all simulations it was found that the average download time of peers running the different protocols is performing quite similar, where for short download times the difference is bigger and as the download time increases the difference declines. The biggest difference can be found in the dynamic scenario with small files and a private tracker 5.37 in which the modified protocol clearly outperforms the traditional. The modifications analysed therefore make sense to be adapted in an environment in which a small number of peers share small files, as can be found in wireless ad-hoc networks or similar scenarios.

In addition a second consistent effect is that the modified protocol is reducing the variation in download time over all peers in the swarm. Increasing the average download time for the fastest and reducing it for the slower peers in a swarm. This can be used as a positive effect in an environment in which soft quality of service is of importance, and a maximum download time is tried to be guarantied. An additional scenario could be the distribution of content in a system that can only precede to a following system sate when all peers have completely received the shared data and reducing the download time or the slowest peers increases the performance for the whole system.

5.7.2 Simulation itself

Besides effects that have been found in the behaviour of the both algorithms, two main parameters or properties of Peer-to-Peer simulations themselves have been found that have not been described or analysed in the literature that makes for the basis of this work.

As mentioned earlier 5.1.1 one of the most important parameters of the simulations has been found to be the swarm size; the number of peers as

well as the ratio of seeders and leechers, in the swarm. Where in the static scenarios this can be controlled, the swarm size in the dynamic scenarios is a result of system performance, churn and a random factor. It was found that in general the average download time of each peer increases as the swarm size grows, but it is not clear how this increase is governed by the swarm size and more importantly which effect the number or ratio of seeders have.

A second important property of the simulations is the churn rate. In most of the literature used for this work, simulations of what here is called static swarms was performed, where all peers have been present in the simulation from the start and after completing the download immediately leave the swarm. The results received by the dynamic simulations have been found to be quite different than their static counterparts and the magnitude of the effects discussed in the previous section differ as well.

5.8 Further Research

Drawing from the overall conclusion of the previous section two important fields of further research appear.

The churn rate of the system was modeled in a very basic matter through only two scalar parameters (e.g. *SpwanRate* and *LeaveRate*). As this was found to be an important property of the simulations it makes sense to investigate more sophisticated and realistic mechanism to be added to the simulator. One can easily image a system where the *SpawnRate* is dynamic and changes throughout the simulation to for example simulate a initial rush into the swarm, followed by a continuous drop of popularity of the torrent and therefore less and less new peers joining the swarm.

The second interesting field of further research can be the effect of swarm size and seeder ratio on the average download time. This is tightly coupled to the previous suggestions as the number of seeders in the current simulation is only governed by the *LeaveRate* which is constant. Making this value dynamic as well as the number of initial seeders would make it possible to simulate a fast range of different scenarios and possible exposes a deeper insight in the working of Peer-to-Peer systems.

Conclusion

In this work, a series of in the literature presented BitTorrent protocol improvements have fused and evaluated inside a simulator against the traditional protocol. The simulator used was specially developed for this work and is publicly available. The achieved results show the modified protocol to be superior in scenarios where small files are shared among few peers and to balance the average download time among peers with different bandwidth limitations. In addition future research topics haven be exposed that suggest to investigate the effect of swarm size and churn rate on the average download rate of peers.

Acknowledgement

This work was coordinated and overseen by Francesc Daniel Muñoz-Escoí. The author would like to thank him in addition to Arvid Norberg and Ivana Dzakula for their support.

Bibliography

[1] *Dissecting BitTorrent: Five Months In Torrent's Lifetime*, 2004.

[2] Bit torrent simulator. http://research.microsoft.com/en-us/downloads/20d68689-9a8d-44c0-80cd-66dfa4b0504b/, 2005.

[3] Azureus. http://sourceforge.net/projects/azureus/, 2011.

[4] Bitmate. http://www.dritte.org/bitmate.html, 2011.

[5] Bittorrent protocol on wikipedia. http://en.wikipedia.org/wiki/BitTorrent_(protocol), 2011.

[6] Bittorrent protocol specification v1.0. http://wiki.theory.org/BitTorrentSpecification, 2011.

[7] Internet observatory - real-time internet statistics. http://www.internetobservatory.net/, 2011.

[8] Oneswarm. http://www.oneswarm.org, 2011.

[9] A open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org, 2011.

[10] Transmission. http://www.transmissionbt.com/, 2011.

[11] Tribler. http://www.tribler.org, 2011.

[12] utorrent keeps bittorrent lead, bitcomet fades away. http://torrentfreak.com/utorrent-keeps-bittorrent-lead-bitcomet-fades-away-110916/, 2011.

[13] Vuze - bittorrent client. `http://www.vuze.com`, 2011.

[14] Eurliaf - a bittorrent like overlay peer simulator. `https://github.com/mapa17/Eruliaf`, 2012.

[15] David R. Choffnes and Fabián E. Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. In *SIGCOMM*, pages 363–374, 2008.

[16] B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

[17] Pedro Evangelista, Marcelo Amaral, Charles Miers, Walter Goya, Marcos A. Simplício Jr., Tereza Cristina M. B. Carvalho, and Victor Souza. Ebitsim: An enhanced bittorrent simulation using omnet++ 4. In *MASCOTS*, pages 437–440, 2011.

[18] Tao Guo, Xu Zhou, Hui Tang, and Zexu Wu. A peer selection algorithm with consideration of both network topology information and node capability in p2p network. In *Proceedings of the 2010 International Conference on Intelligent Computation Technology and Automation - Volume 01*, ICICTA '10, pages 293–298, Washington, DC, USA, 2010. IEEE Computer Society.

[19] Kun Huang, Li'e Wang, Dafang Zhang, and Yongwei Liu. Optimizing the bittorrent performance using an adaptive peer selection strategy. *Future Generation Comp. Syst.*, 24(7):621–630, 2008.

[20] Seung Jun and Mustaque Ahamad. Incentives in bittorrent induce free riding. In *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, P2PECON '05, pages 116–121, New York, NY, USA, 2005. ACM.

[21] Konstantinos V. Katsaros, Vasileios P. Kemerlis, Charilaos Stais, and George Xylomenos. A bittorrent module for the omnet++ simulator. In *MASCOTS*, pages 1–10, 2009.

[22] Nikolaos Laoutaris, Damiano Carra, and Pietro Michiardi. Uplink allocation beyond choke/unchoke: or how to divide and conquer best. In *CoNEXT*, page 18, 2008.

[23] Arnaud Legout, Guillaume Urvoy-Keller, and Pietro Michiardi. Rarest first and choke algorithms are enough. In *Internet Measurement Conference*, pages 203–216, 2006.

[24] Michael Piatek, Tomas Isdal, Thomas E. Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent? (awarded best student paper). In *NSDI*, 2007.

[25] Dongyu Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 367–378, New York, NY, USA, 2004. ACM.

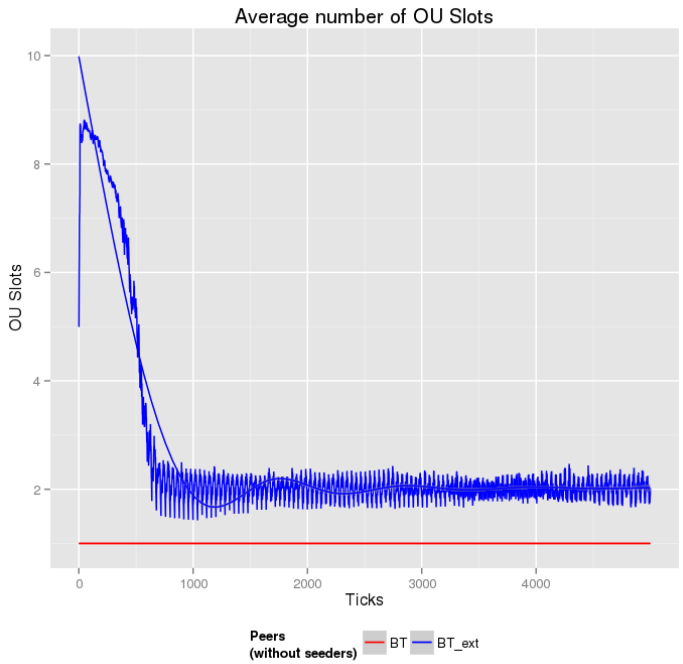
[26] Hendrik Schulze and Klaus Mochalski. Internet study 2008/2009. *Africa*, pages 1–13, 2009.

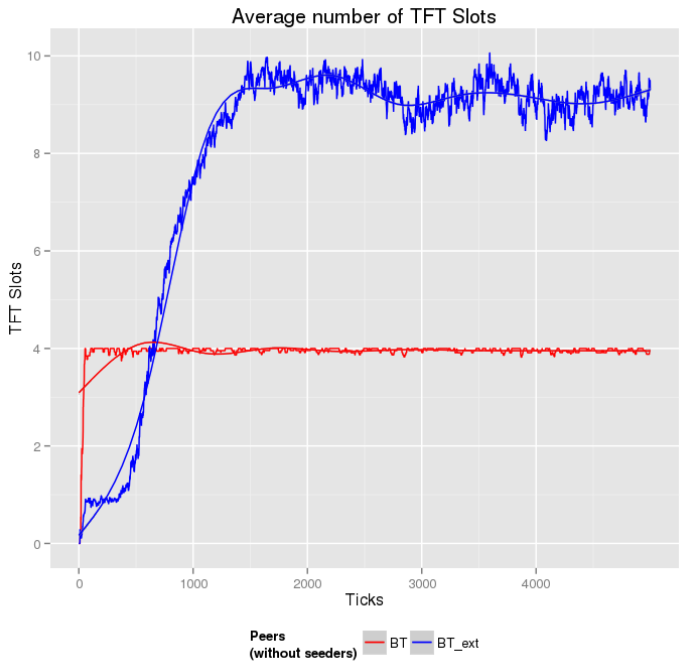
[27] Weishuai Yang and Nael Abu-Ghazaleh. Gps: A general peer-to-peer simulator and its use for modeling bittorrent. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 425–434, Washington, DC, USA, 2005. IEEE Computer Society.

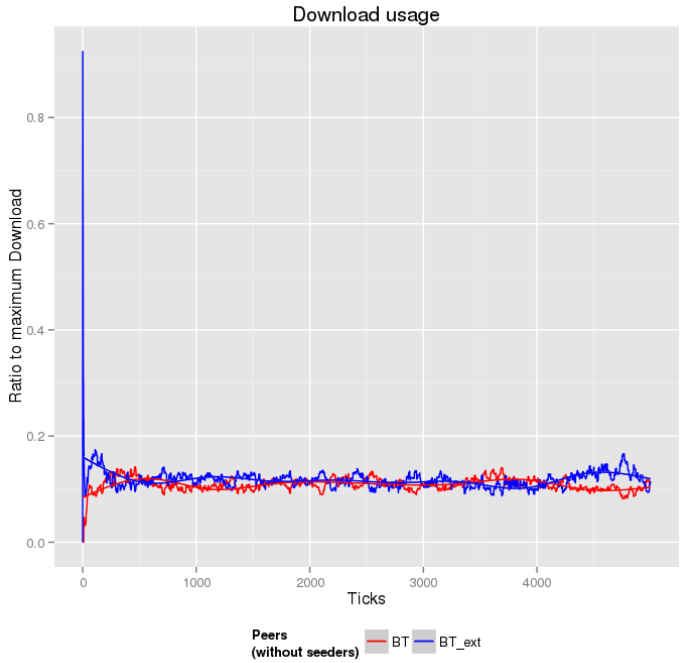
Appendix A

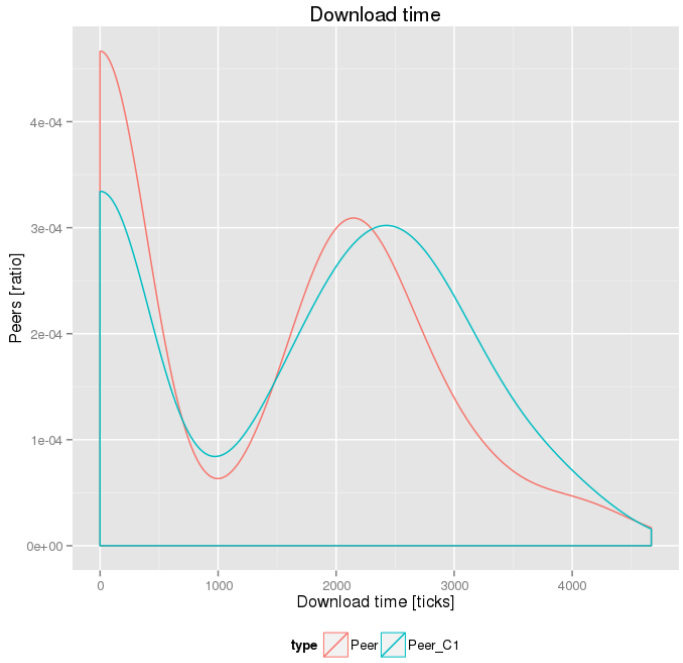
A.1 Statistic Summary example

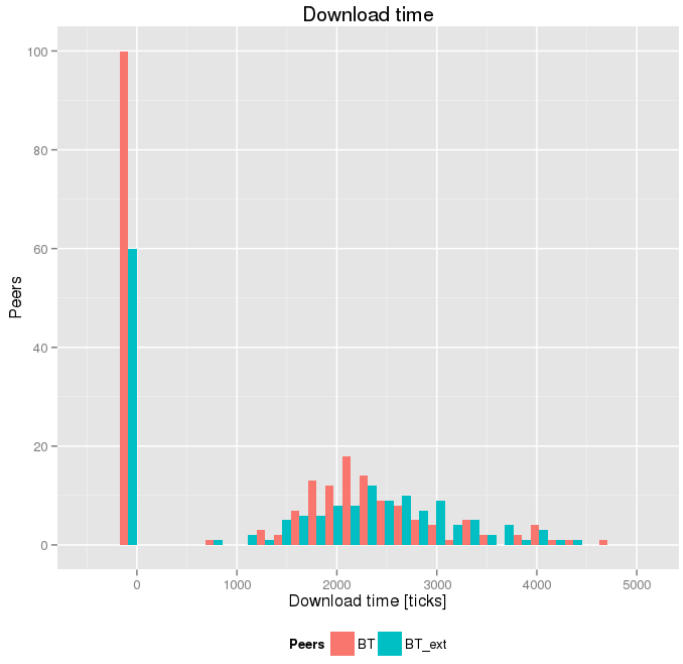
The following is a plot of the statistic summary pdf tile generated as described in section 4.2.3.

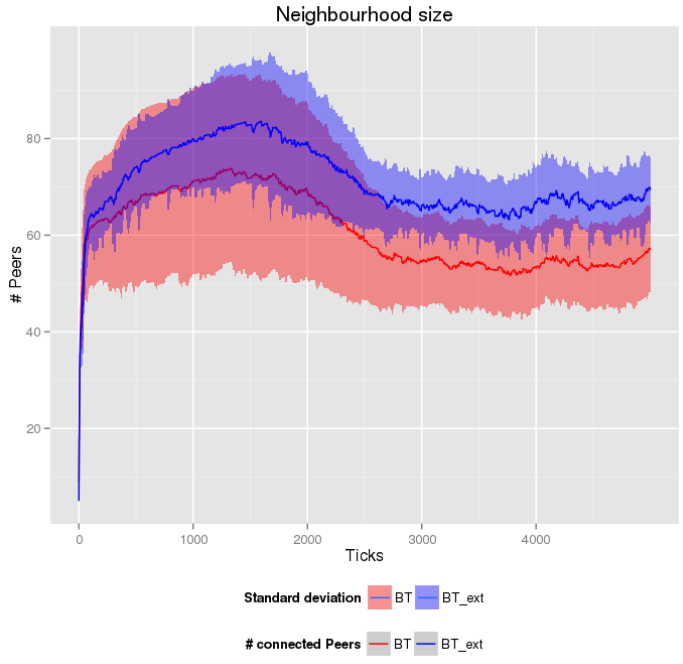

Average number of OU Slots
OU Slots
Ticks
Peers
(without seeders)
BT
BT_ext

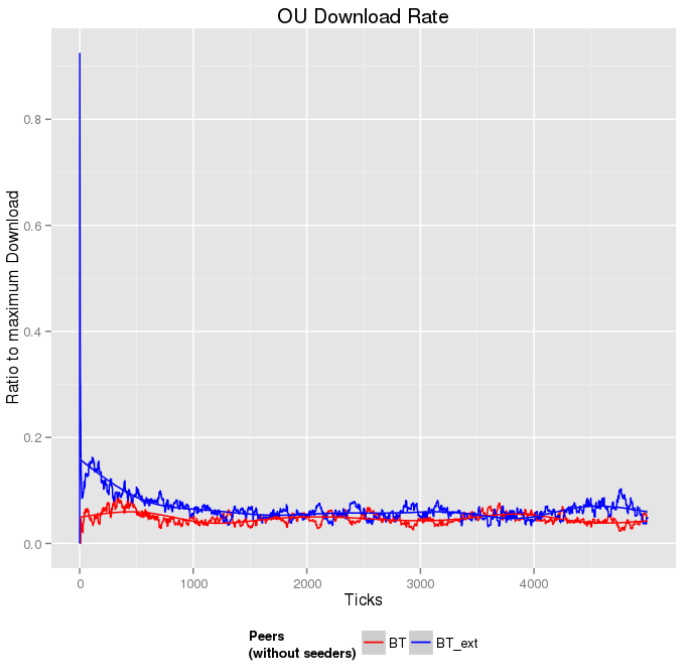

Average number of TFT Slots
TFT Slots
Ticks
Peers
(without seeders)
BT
BT_ext

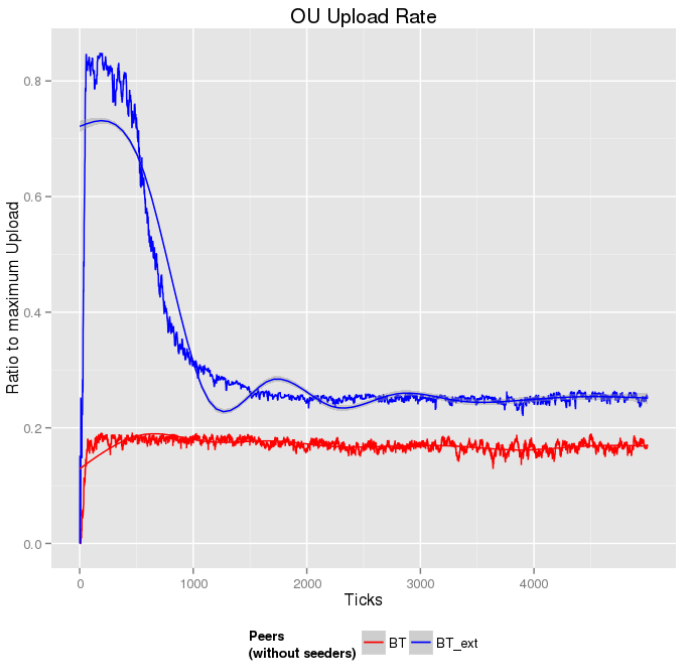

Download usage
Ratio to maximum Download
Ticks
Peers
(without seeders)
BT
BT_ext

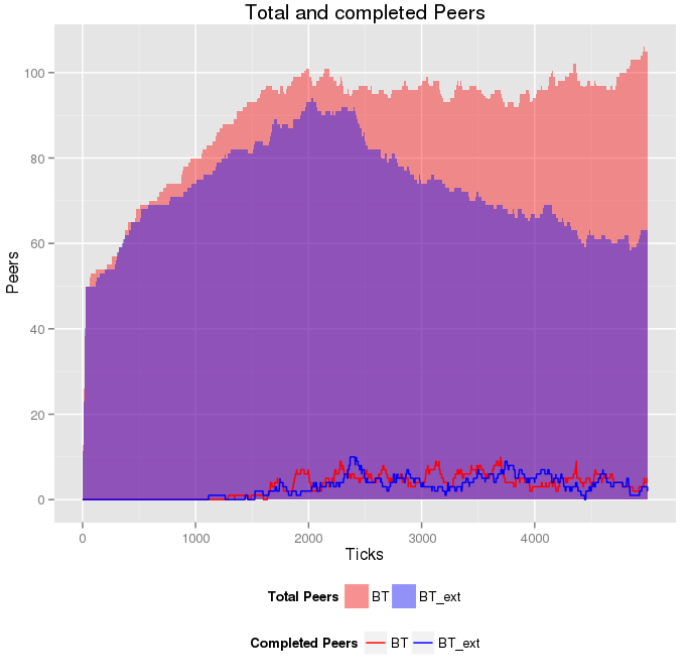

Download time
Peers [ratio]
Download time [ticks]
type
Peer
Peer_C1


Download time
Peers
Download time [ticks]
Peers
BT
BT_ext

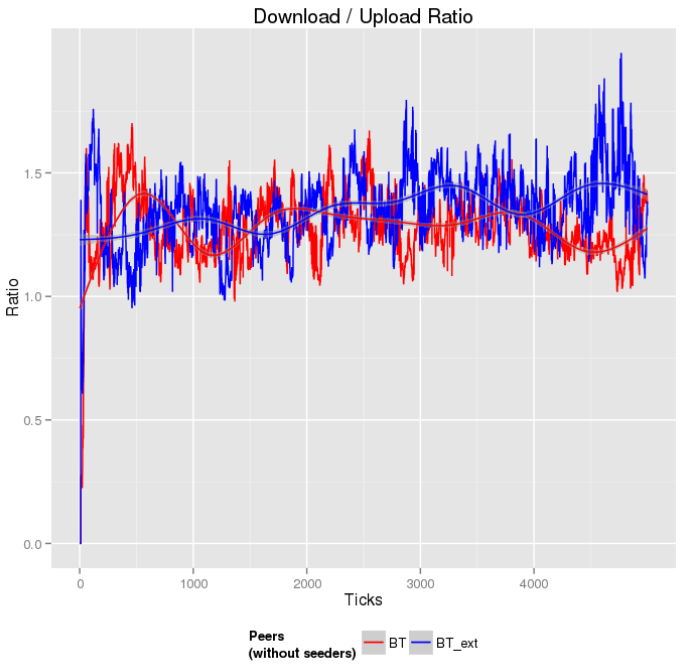

Neighbourhood size
Peers
Ticks
Standard deviation
BT
BT_ext
connected Peers
BT
BT_ext

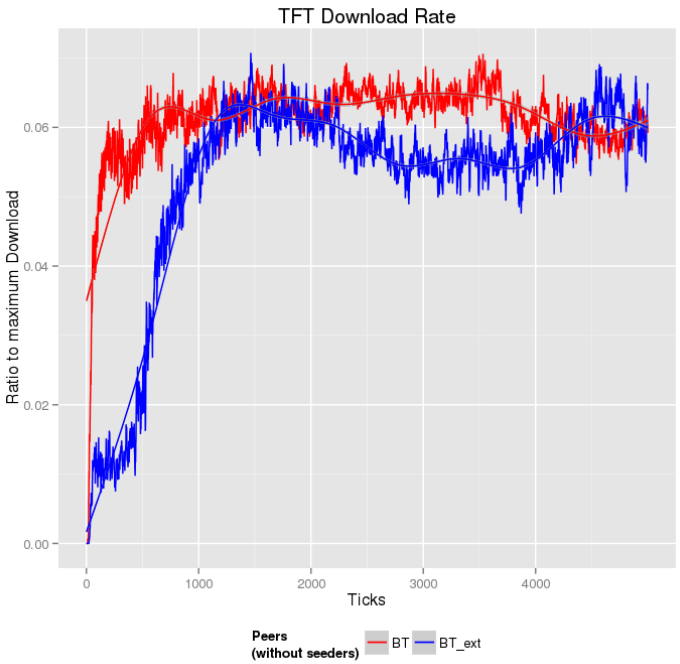

OU Download Rate
Ratio to maximum Download
0.0
0.2
0.4
0.6
0.8
0
1000
2000
3000
4000
Ticks
Peers
(without seeders)
BT
BT_ext

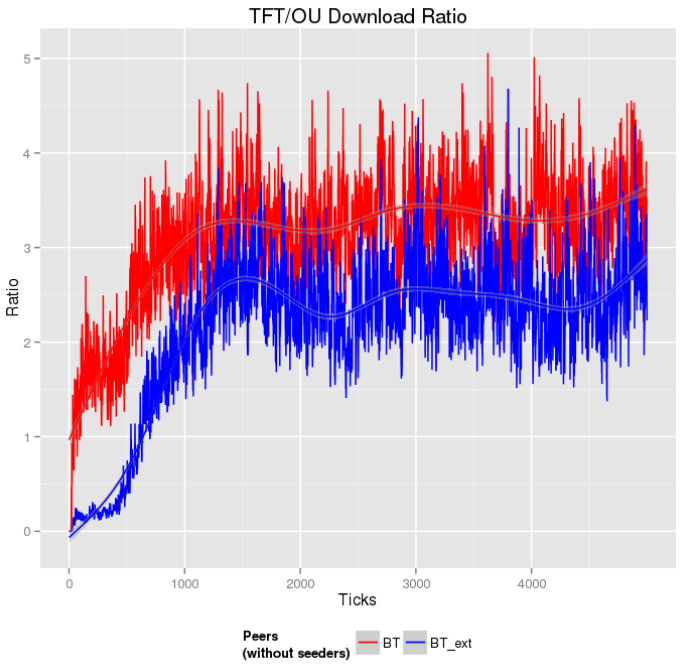

OU Upload Rate
Ratio to maximum Upload
0.0
0.2
0.4
0.6
0.8
0
1000
2000
3000
4000
Ticks
Peers
(without seeders)
BT
BT_ext

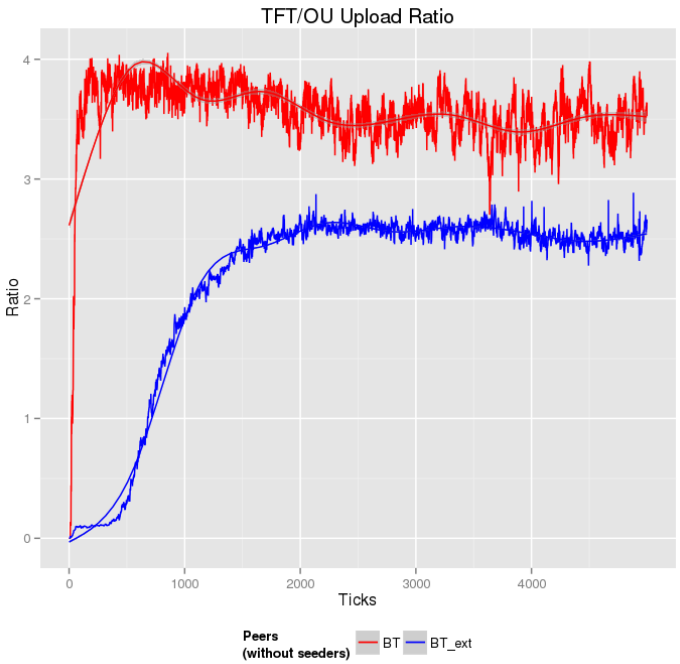

Total and completed Peers
Peers
0
20
40
60
80
100
0
1000
2000
3000
4000
Ticks
Total Peers
BT
BT_ext
Completed Peers
BT
BT_ext

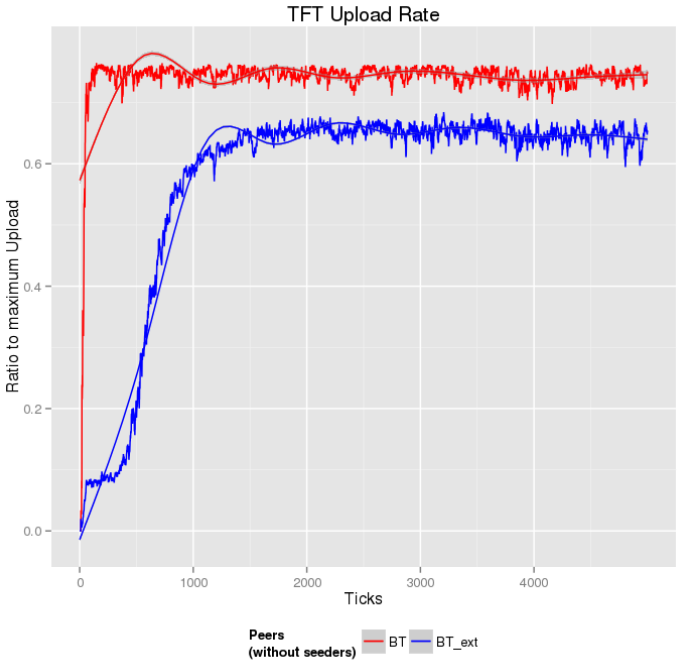

Local Piece availability
Pieces
0
500
1000
1500
0
1000
2000
3000
4000
Ticks
Number of Pieces
BT
BT_ext
Standard deviation
BT
BT_ext

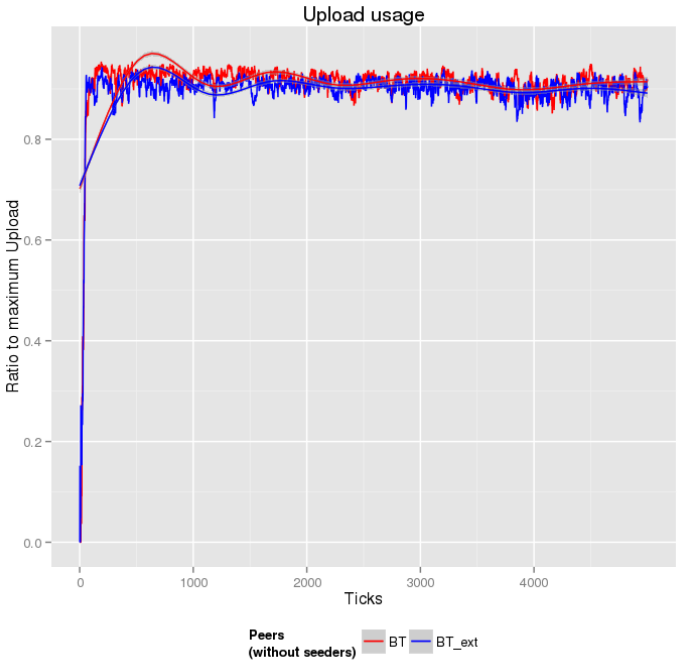

Download / Upload Ratio
Ratio
0.0
0.5
1.0
1.5
0
1000
2000
3000
4000
Ticks
Peers
(without seeders)
BT
BT_ext

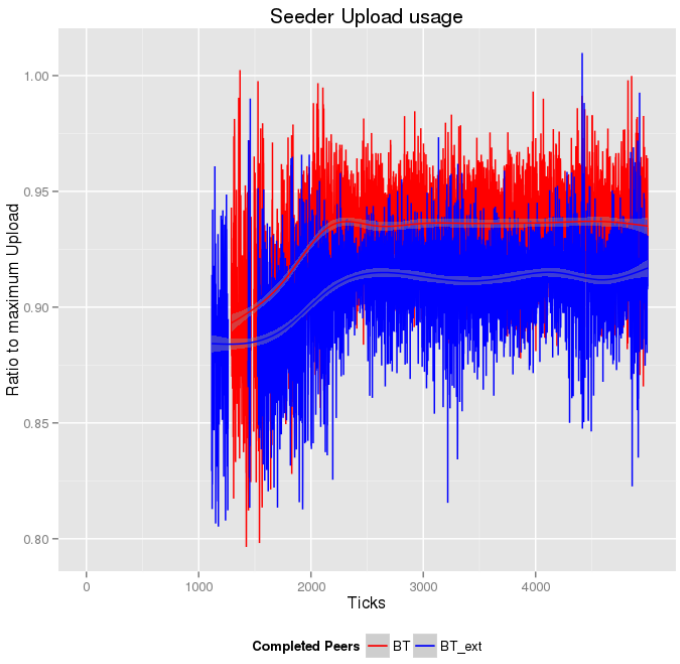

TFT Download Rate
Ratio to maximum Download
0.00
0.02
0.04
0.06
0
1000
2000
3000
4000
Ticks
Peers
(without seeders)
BT
BT_ext


TFT/OU Download Ratio
Ratio
Ticks
0
1000
2000
3000
4000
0
1
2
3
4
5
Peers
(without seeders)
BT
BT_ext


TFT/OU Upload Ratio
Ratio
Ticks
0
1000
2000
3000
4000
0
1
2
3
4
Peers
(without seeders)
BT
BT_ext


TFT Upload Rate
Ratio to maximum Upload
Ticks
0
1000
2000
3000
4000
0.0
0.2
0.4
0.6
Peers
(without seeders)
BT
BT_ext


Upload usage
Ratio to maximum Upload
Ticks
0
1000
2000
3000
4000
0.0
0.2
0.4
0.6
0.8
Peers
(without seeders)
BT
BT_ext


Seeder Upload usage
Ratio to maximum Upload
Ticks
0
1000
2000
3000
4000
0.80
0.85
0.90
0.95
1.00
Completed Peers
BT
BT_ext

Listing A.1: "Scenario example File"

```
#Scenario: Little PeerC1 Net

[General]
#1024*1024*10 ... 10MB
TorrentSize = 10485760
#10Kb -> 1024 pieces
PieceSize = 10240
SimEnd = 1800
#5Mb up / 1 kb down
SeederUpload = 655360
SeederDownload = 1024

#Parameters will be changed during simulation
logFile = ./run/100.log
#logCfg = ./log.conf
logLevel = INFO
statsFile = ./run/100.csv
randSeed = 100

[Peer]
nInitialPeers = 15
SpwanRate = 0.025
LeaveRate = 0.005
MaxSleep = 30

#64-256kb Up, 256-1024kb Down
#DirectUpload min 320 ticks, max 1280 ticks
#DirectDownload min 80 ticks, max 320
UploadRateMin = 8192
UploadRateMax = 32768
DownloadRateMin = 32768
DownloadRateMax = 131072

[PeerC1]
nInitialPeers = 15
SpwanRate = 0.025
LeaveRate = 0.005
MaxSleep = 30

#64-256kb Up, 256-1024kb Down
#DirectUpload min 320 ticks, max 1280 ticks
#DirectDownload min 80 ticks, max 320
UploadRateMin = 8192
UploadRateMax = 32768
DownloadRateMin = 32768
DownloadRateMax = 131072
```

Listing A.2: "Simulation example File"

```
[General]
nIterations = 4
nThreads = 4
iterationPrefix = run
scenario = ./configs/scenarios/dynamic_50p_50pc1_1000MB.cfg
runDirectory = ./run/dynamic_50p_50pc1_1000MB
randSeedBase = 0
statsScript = ./R/createStatistics.py
rScript = ./R/Statistics.R
statsOutput = ./run/dynamic_50p_50pc1_1000MB/statsData
logCfg = ./log.conf
logLevel = INFO
statsSummaryDir = ./run/results/dynamic_50p_50pc1_1000MB
```

Appendix B

B.1 Static Private Tracker - Simulation Results

B.1.1 Static private Tracker with small files

B.1.2 Static private Tracker with medium files

B.1.3 Static private Tracker with large files

Figure B.1: **static,10MB Peer Count**

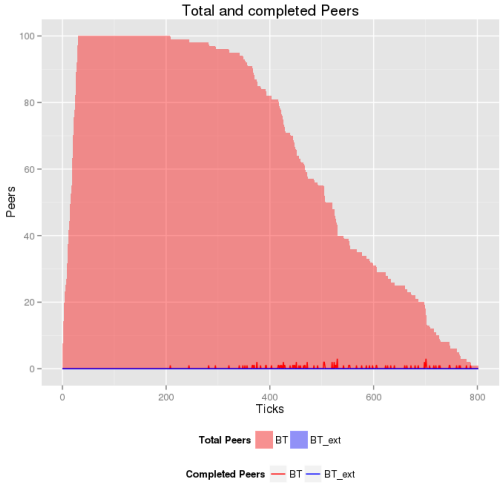


Figure B.2: static,100p,0pc1,10MB

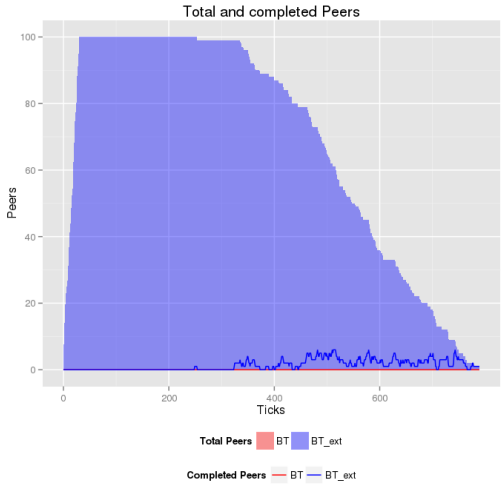


Figure B.3: static,0p,100pc1,10MB

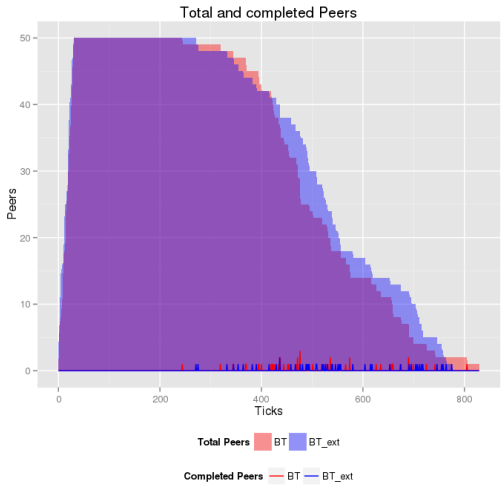


Figure B.4: static,50p,50pc1,10MB

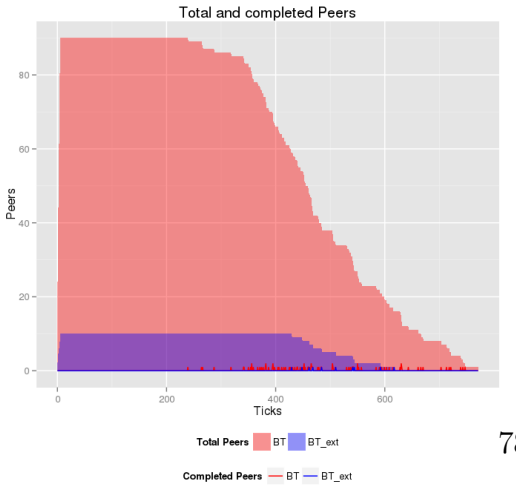


Figure B.5: static,90p,10pc1,10MB

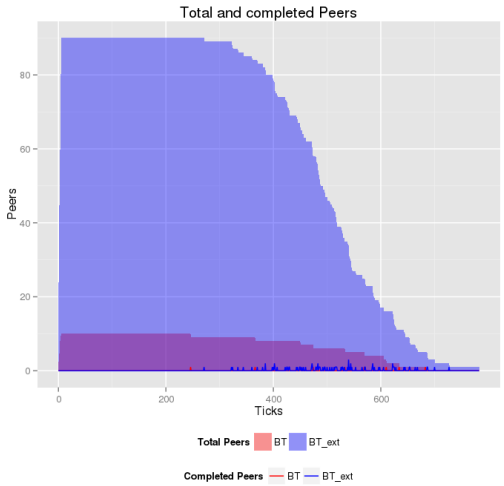


Figure B.6: static,10p,90pc1,10MB

Figure B.7: **static,10MB Average number of OU Slots**

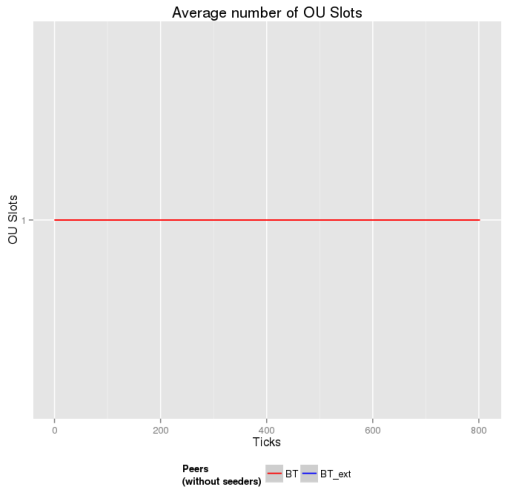


Figure B.8: static,100p,0pc1,10MB

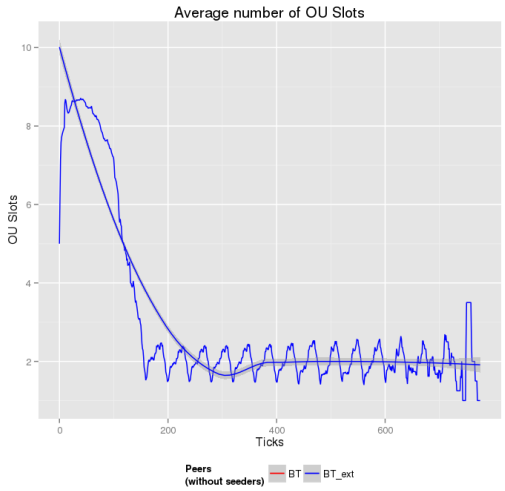


Figure B.9: static,0p,100pc1,10MB

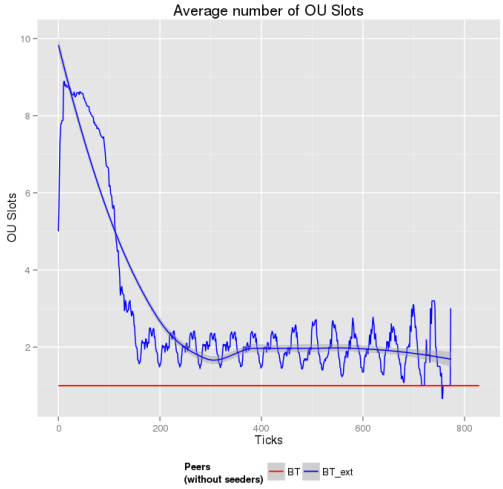


Figure B.10: static,50p,50pc1,10MB

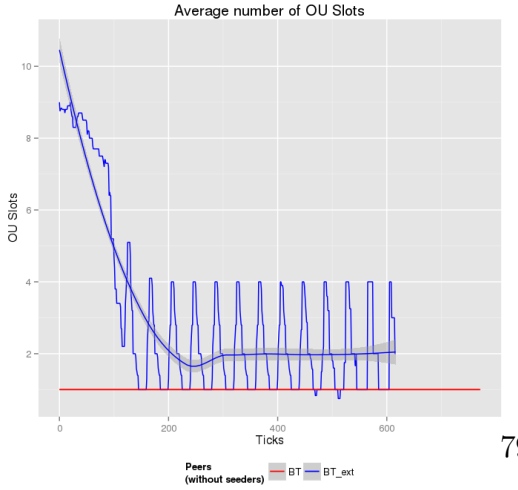


Figure B.11: static,90p,10pc1,10MB

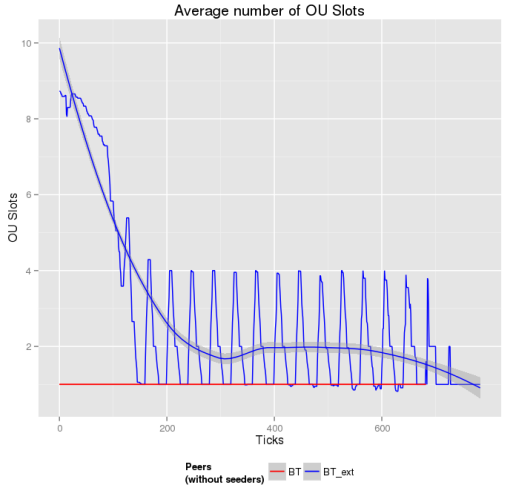


Figure B.12: static,10p,90pc1,10MB

Figure B.13: **static,10MB Average number of TFT Slots**

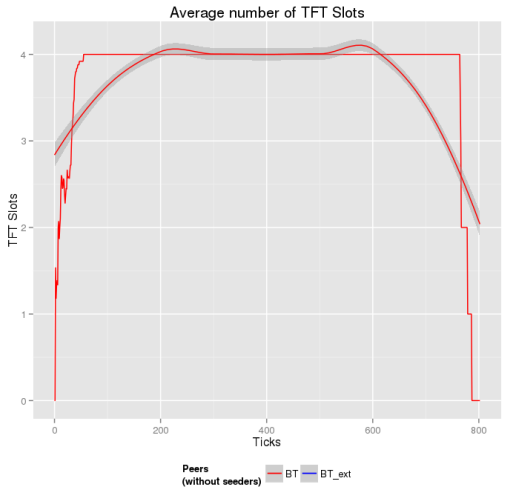


Figure B.14: static,100p,0pc1,10MB

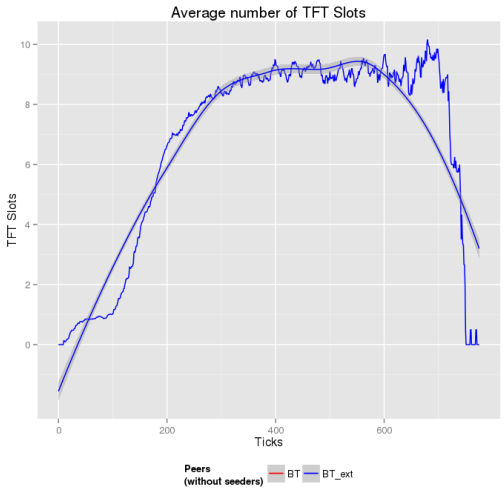


Figure B.15: static,0p,100pc1,10MB

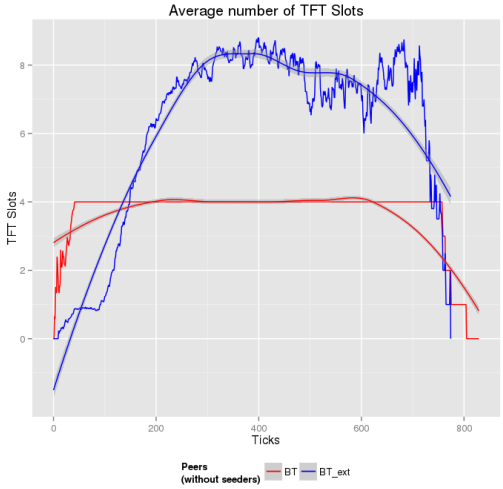


Figure B.16: static,50p,50pc1,10MB

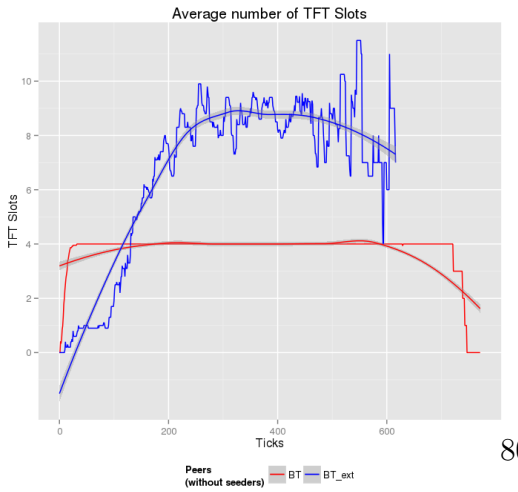


Figure B.17: static,90p,10pc1,10MB

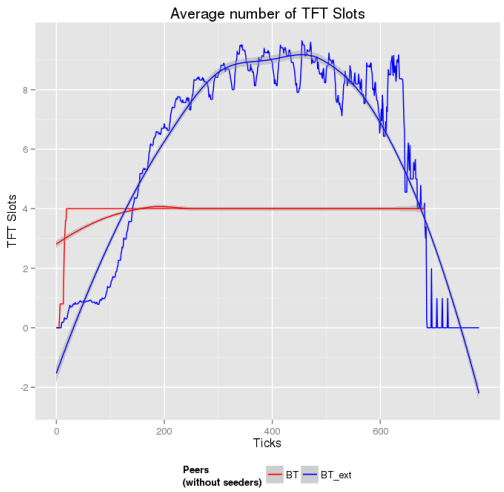


Figure B.18: static,10p,90pc1,10MB

Figure B.19: **static,10MB Scaled upload rate**

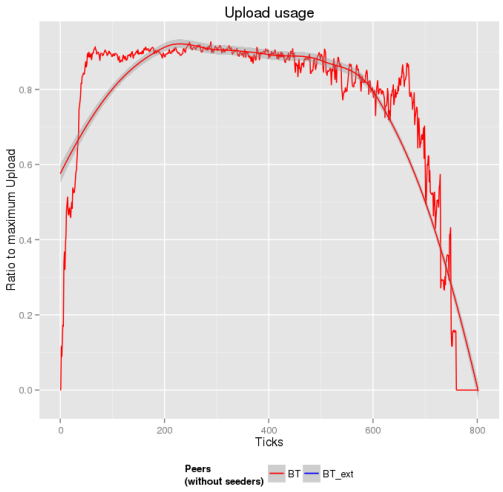


Figure B.20: static,100p,0pc1,10MB

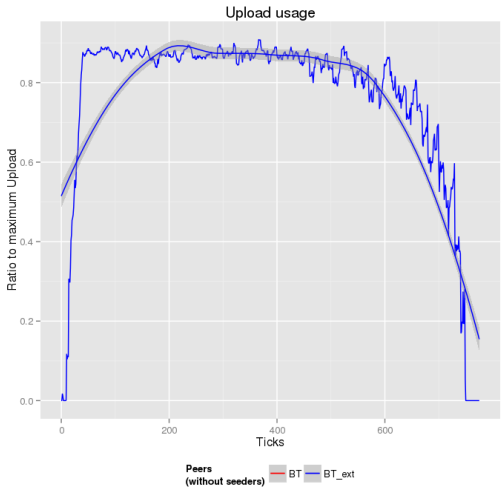


Figure B.21: static,0p,100pc1,10MB

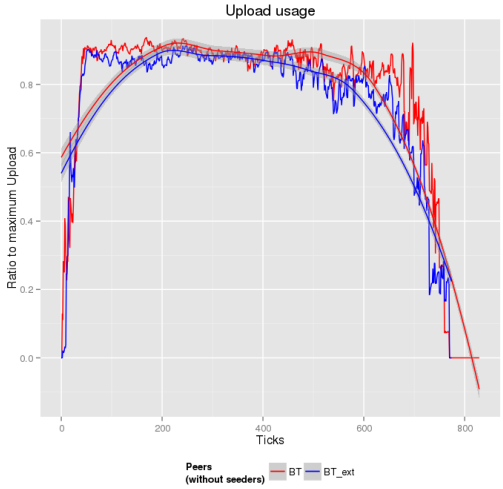


Figure B.22: static,50p,50pc1,10MB

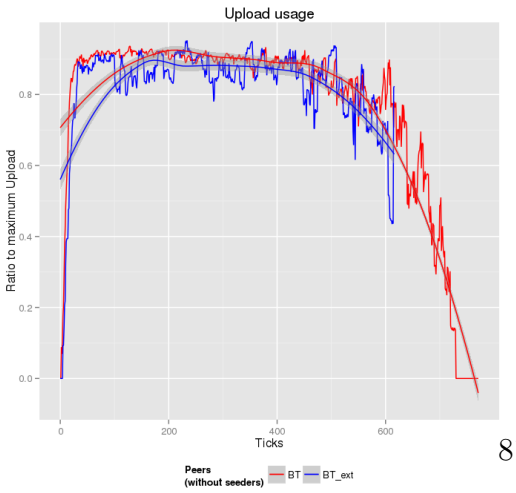


Figure B.23: static,90p,10pc1,10MB

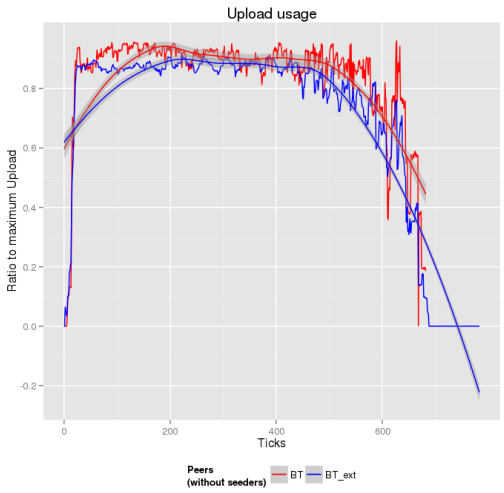


Figure B.24: static,10p,90pc1,10MB

Figure B.25: **static,10MB Scaled download rate**

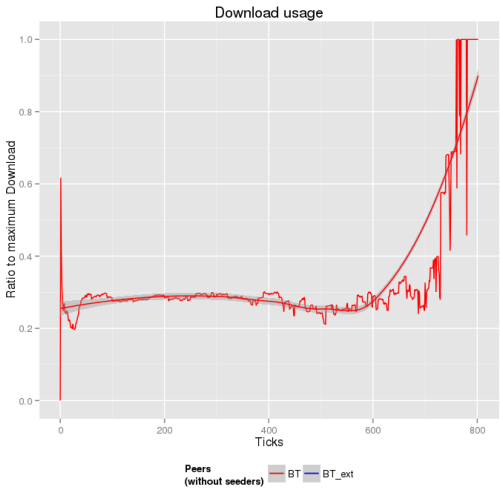


Figure B.26: static,100p,0pc1,10MB

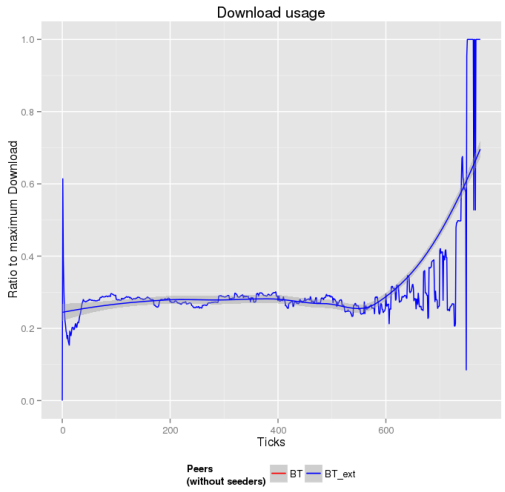


Figure B.27: static,0p,100pc1,10MB

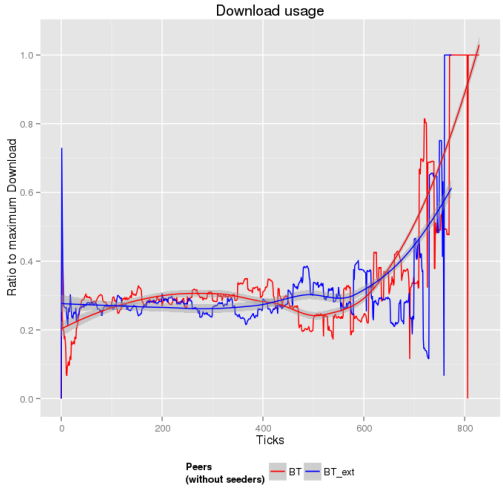


Figure B.28: static,50p,50pc1,10MB

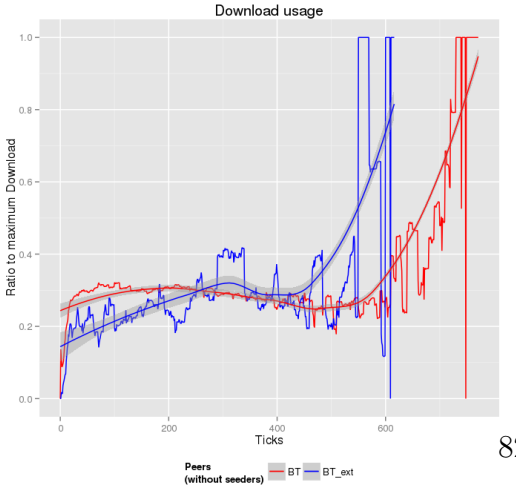


Figure B.29: static,90p,10pc1,10MB

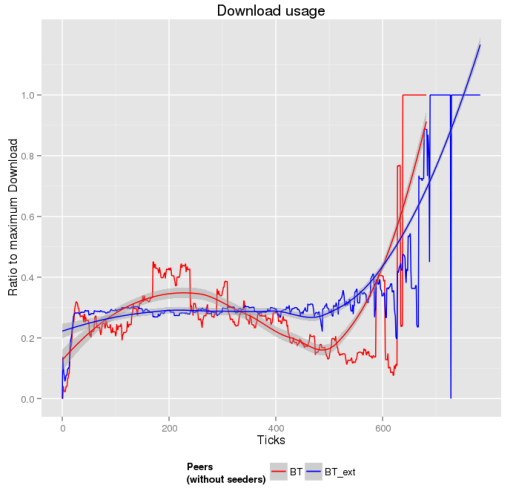


Figure B.30: static,10p,90pc1,10MB

Figure B.31: **static,10MB Peer Count**

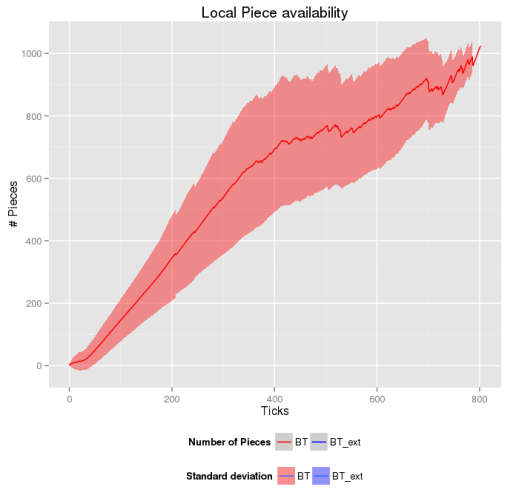


Figure B.32: static,100p,0pc1,10MB

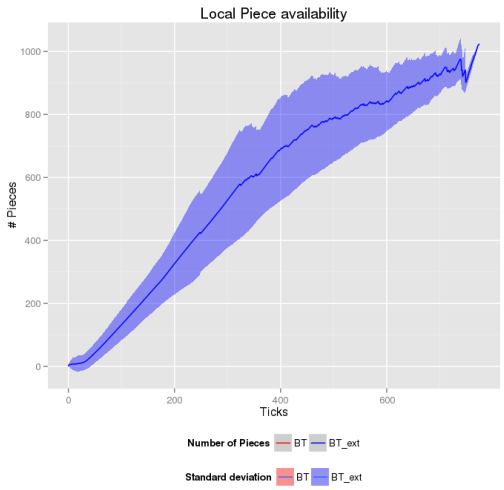


Figure B.33: static,0p,100pc1,10MB

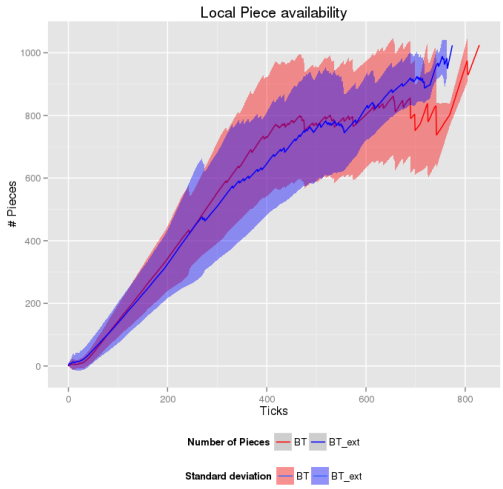


Figure B.34: static,50p,50pc1,10MB

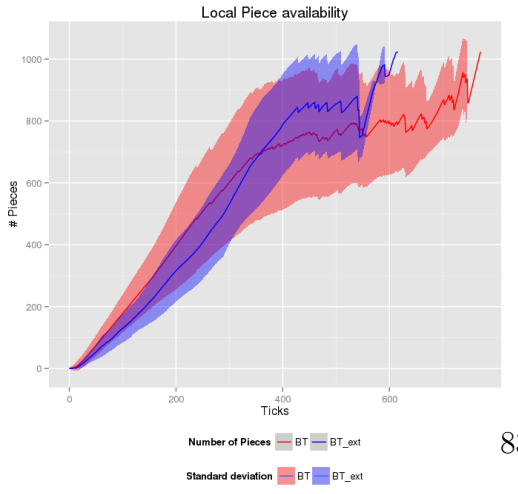


Figure B.35: static,90p,10pc1,10MB

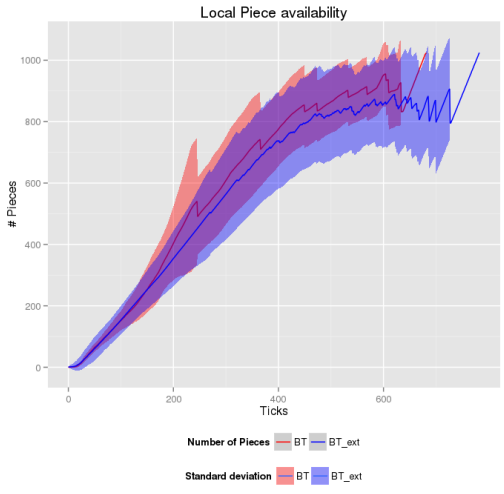


Figure B.36: static,10p,90pc1,10MB

Figure B.37: **static,350MB Peer Count**

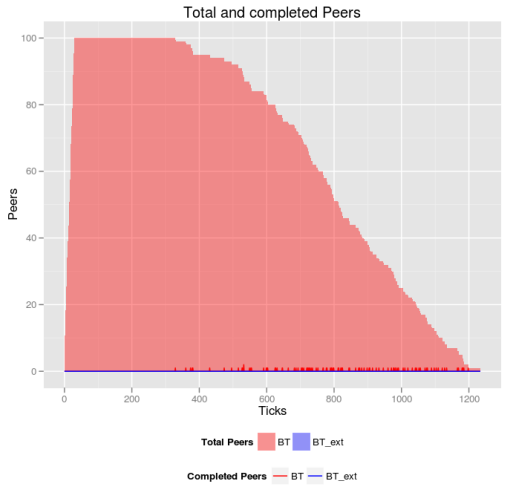


Figure B.38: static,100p,0pc1,350MB

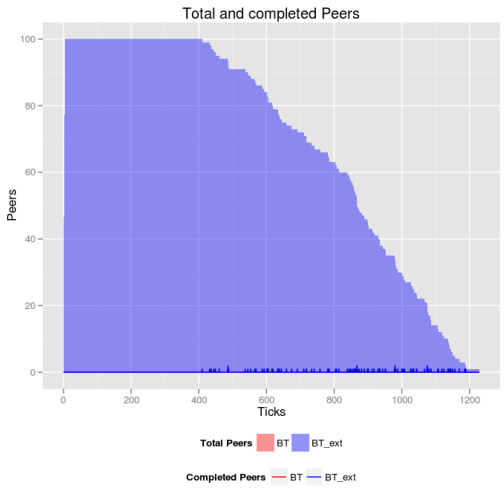


Figure B.39: static,0p,100pc1,350MB

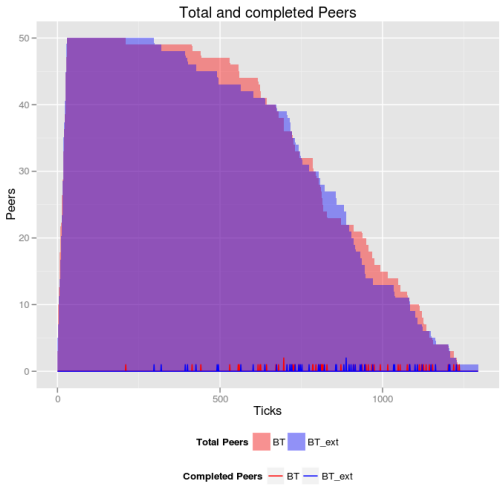


Figure B.40: static,50p,50pc1,350MB

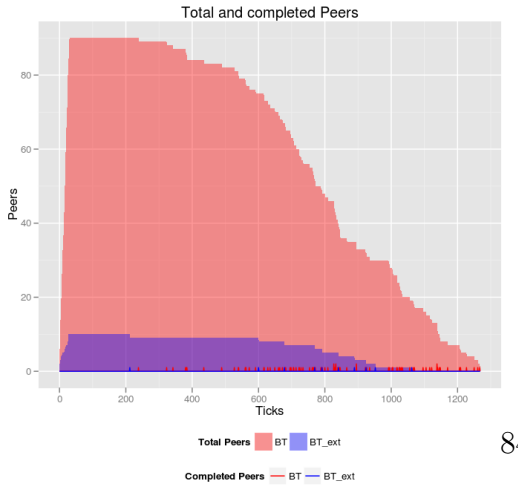


Figure B.41: static,90p,10pc1,350MB

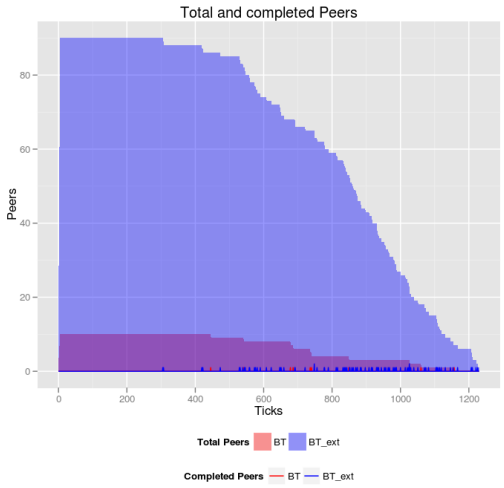


Figure B.42: static,10p,90pc1,350MB

Figure B.43: **static,350MB Average number of OU Slots**

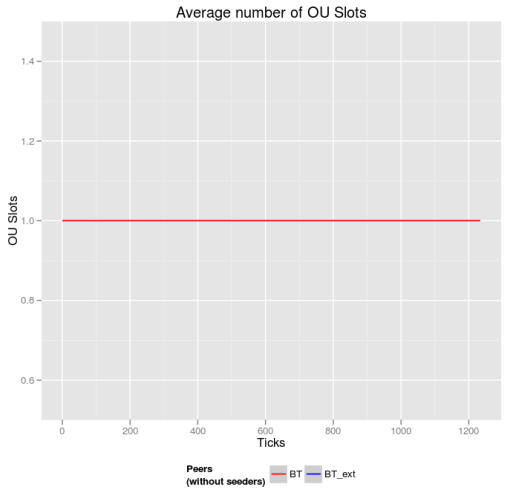


Figure B.44: static,100p,0pc1,350MB

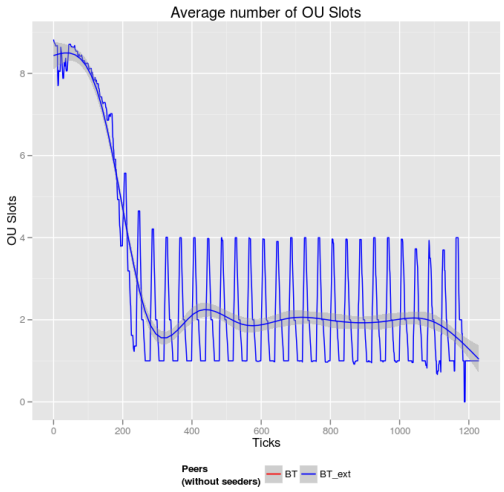


Figure B.45: static,0p,100pc1,350MB

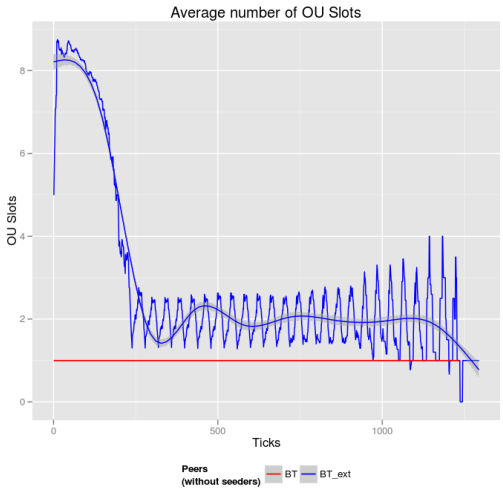


Figure B.46: static,50p,50pc1,350MB

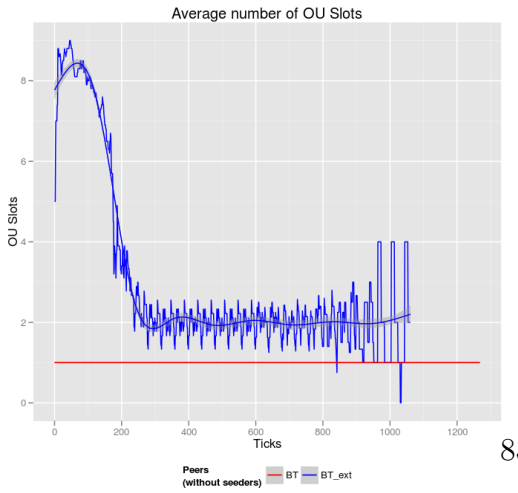


Figure B.47: static,90p,10pc1,350MB

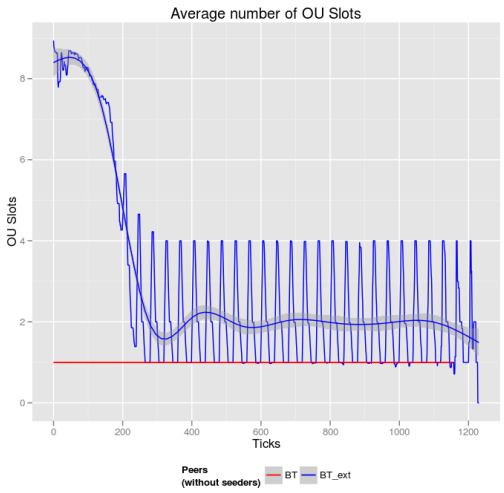


Figure B.48: static,10p,90pc1,350MB

Figure B.49: **static,350MB Average number of TFT Slots**

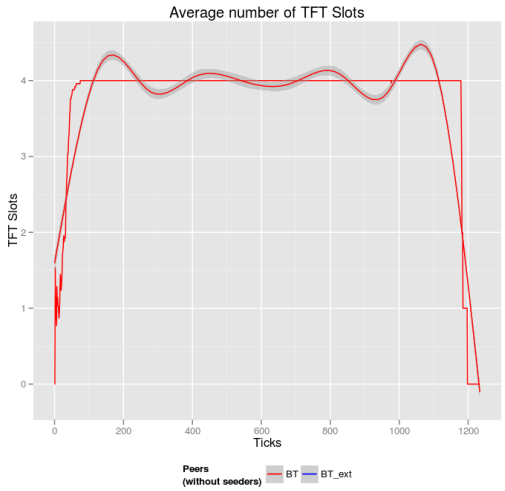


Figure B.50: static,100p,0pc1,350MB

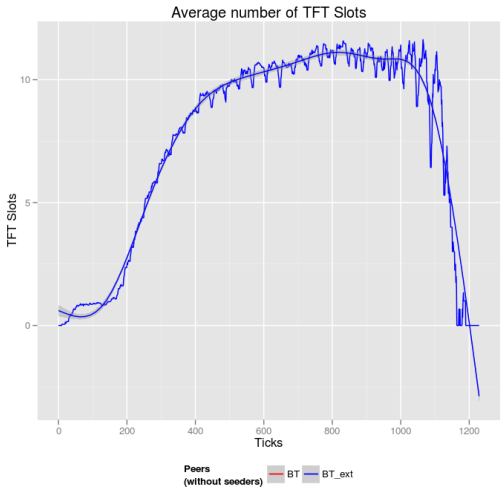


Figure B.51: static,0p,100pc1,350MB

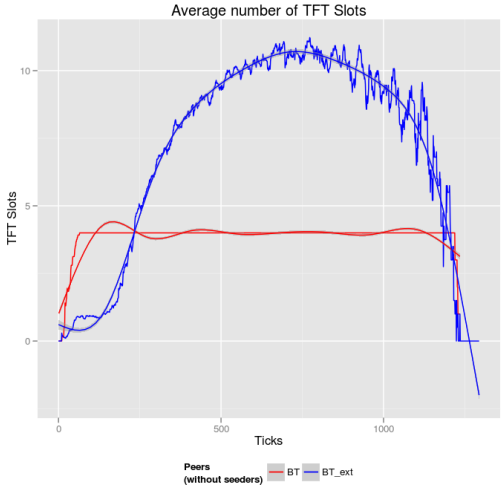


Figure B.52: static,50p,50pc1,350MB

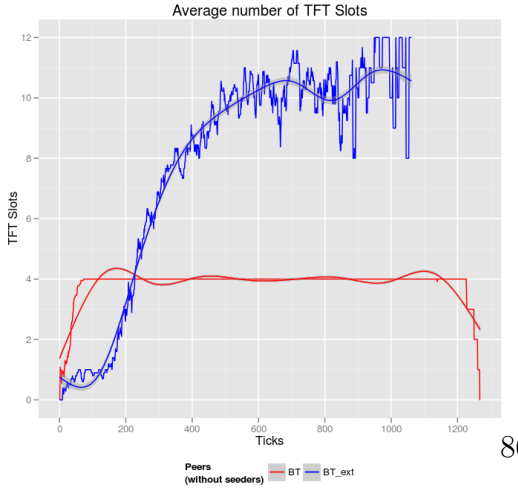


Figure B.53: static,90p,10pc1,350MB

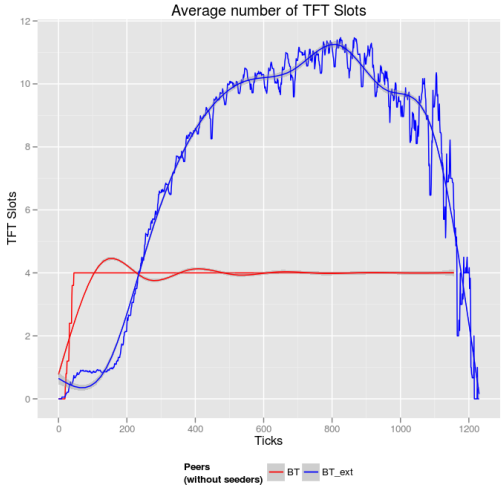


Figure B.54: static,10p,90pc1,350MB

Figure B.55: **static,350MB Scaled upload rate**

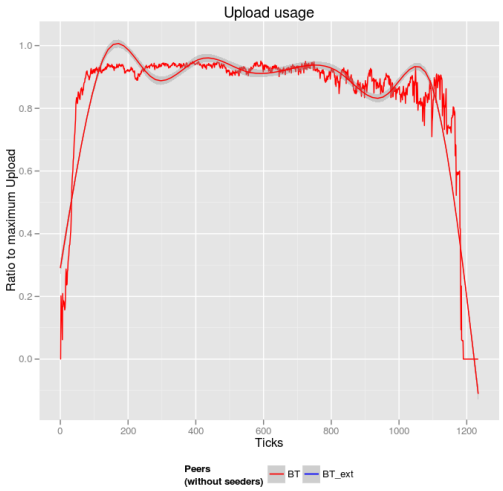


Figure B.56: static,100p,0pc1,350MB

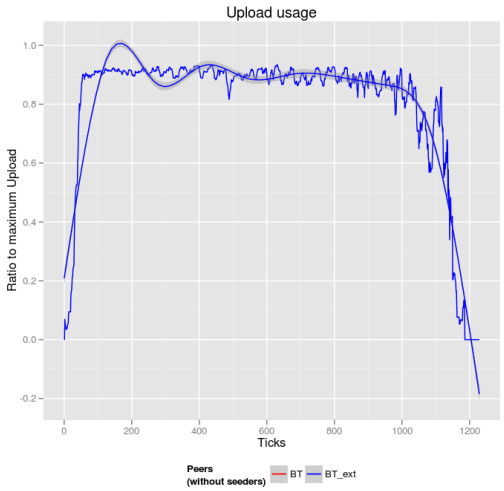


Figure B.57: static,0p,100pc1,350MB

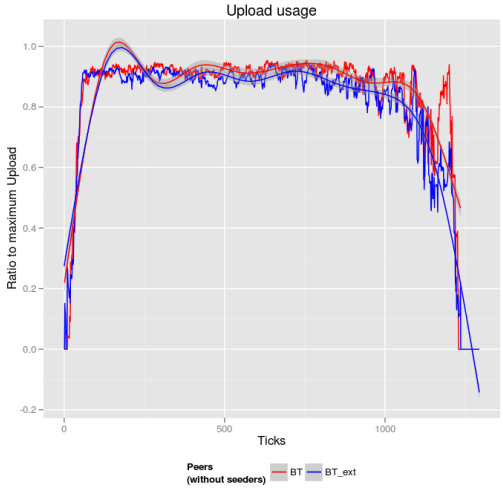


Figure B.58: static,50p,50pc1,350MB

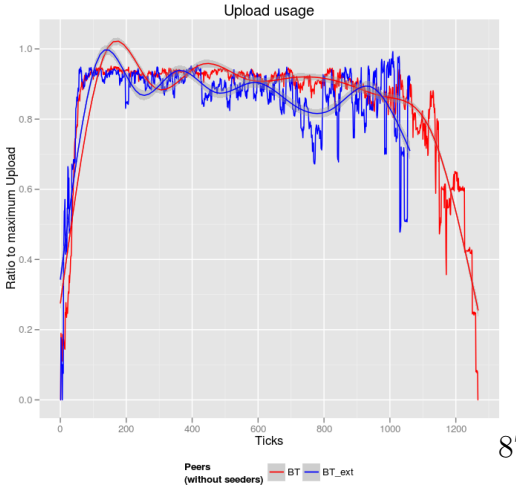


Figure B.59: static,90p,10pc1,350MB

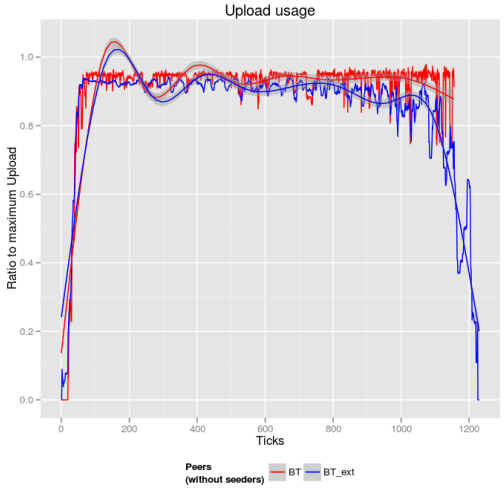


Figure B.60: static,10p,90pc1,350MB

Figure B.61: **static,350MB Scaled download rate**

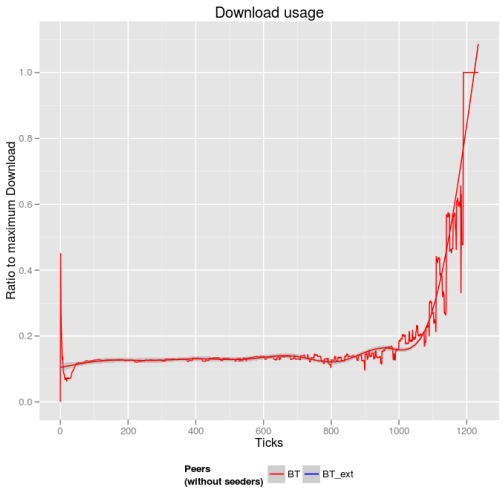


Figure B.62: static,100p,0pc1,350MB

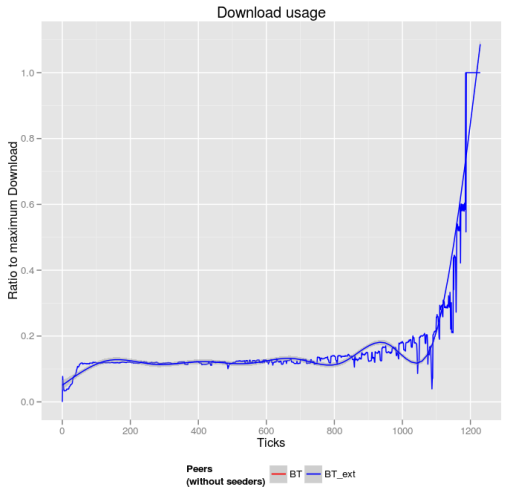


Figure B.63: static,0p,100pc1,350MB

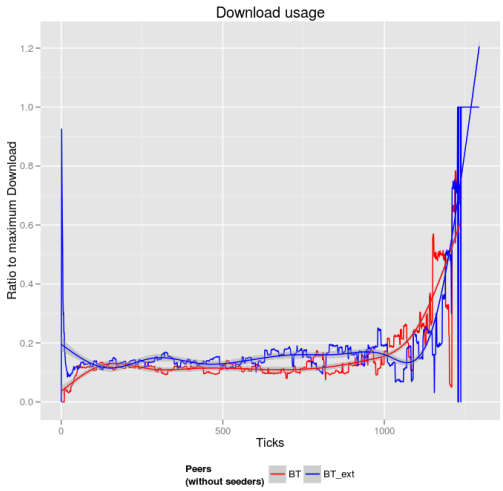


Figure B.64: static,50p,50pc1,350MB

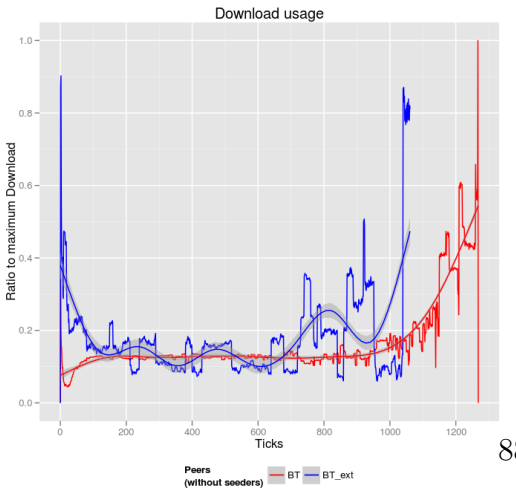


Figure B.65: static,90p,10pc1,350MB

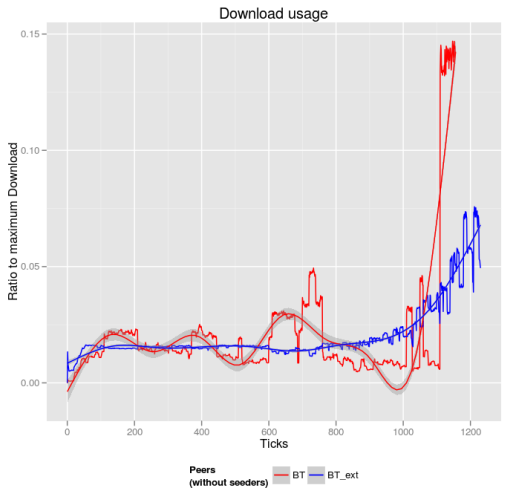


Figure B.66: static,10p,90pc1,350MB

Figure B.67: **static,350MB Peer Count**

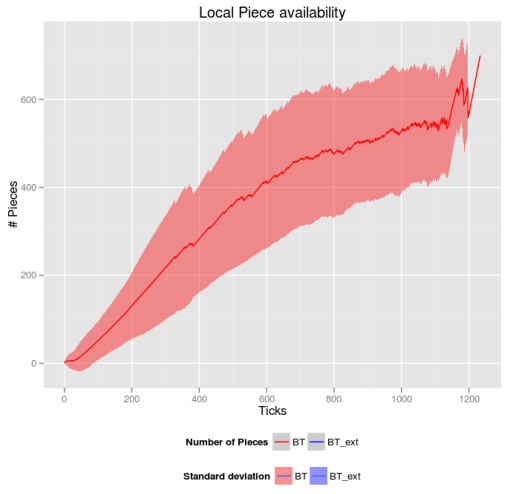


Figure B.68: static,100p,0pc1,350MB

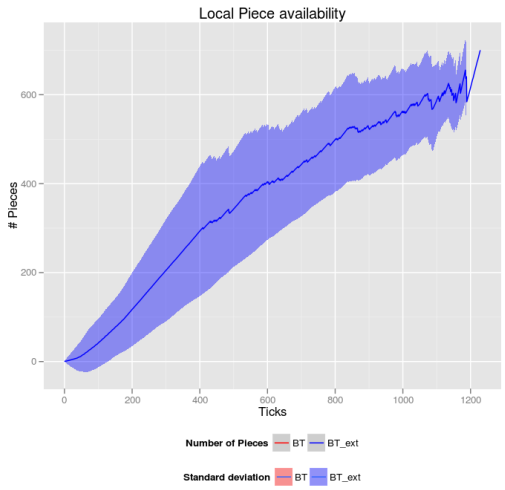


Figure B.69: static,0p,100pc1,350MB

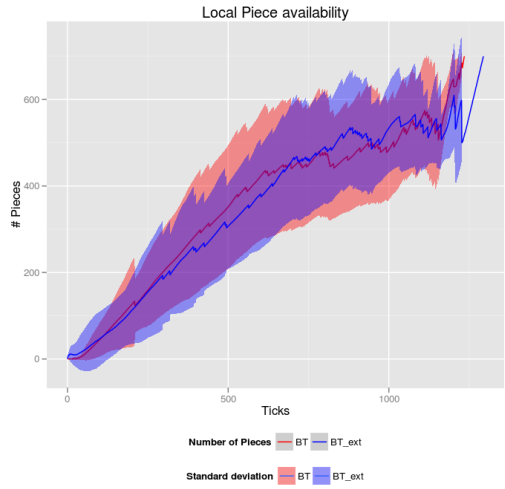


Figure B.70: static,50p,50pc1,350MB

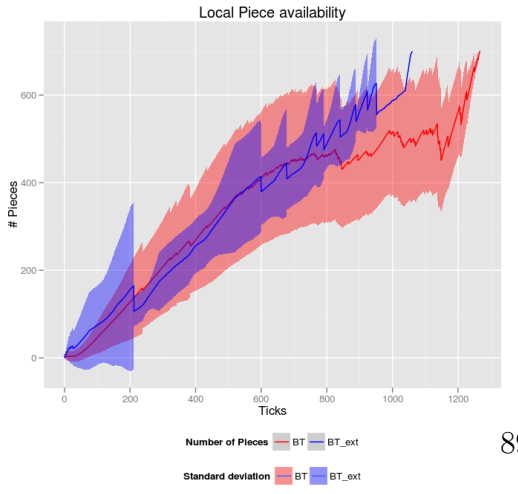


Figure B.71: static,90p,10pc1,350MB

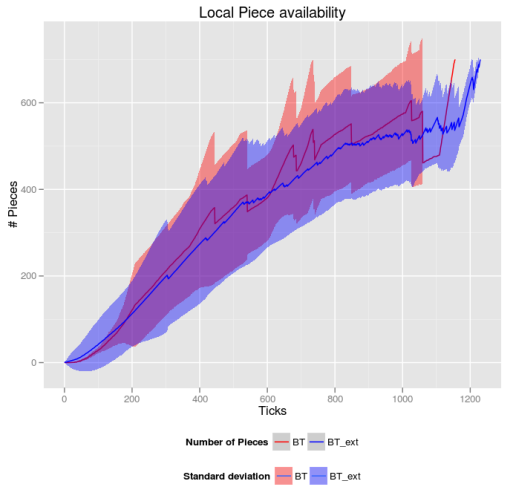


Figure B.72: static,10p,90pc1,350MB

Figure B.73: **static,1000MB Peer Count**

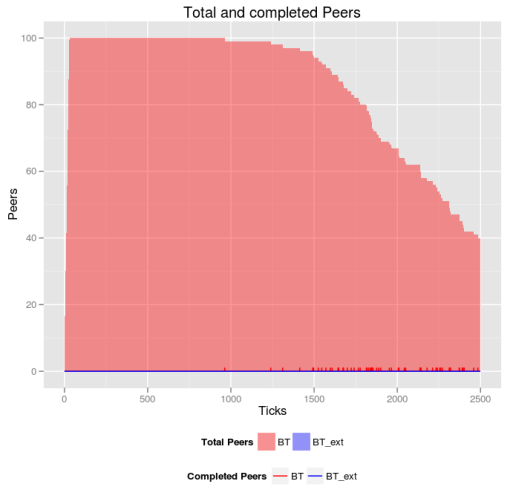


Figure B.74: static,100p,0pc1,1000MB

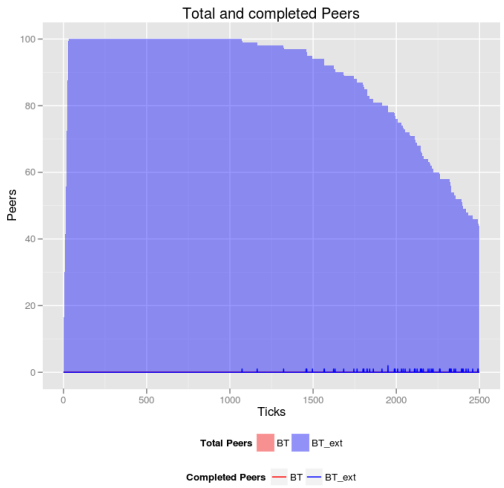


Figure B.75: static,0p,100pc1,1000MB

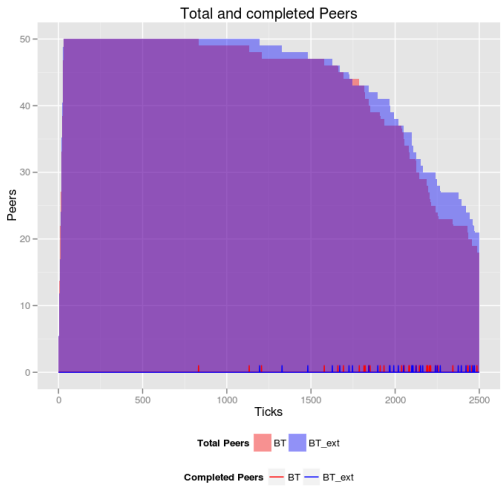


Figure B.76: static,50p,50pc1,1000MB

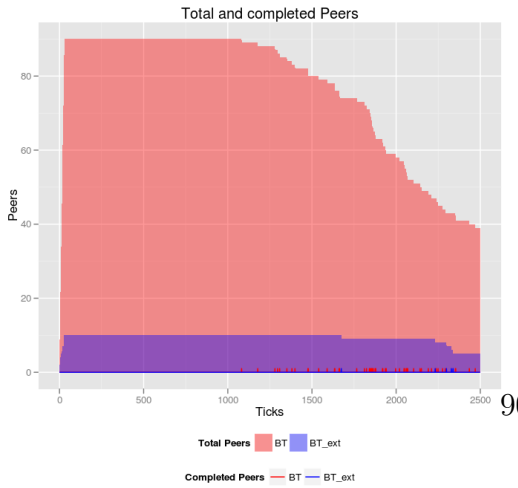


Figure B.77: static,90p,10pc1,1000MB

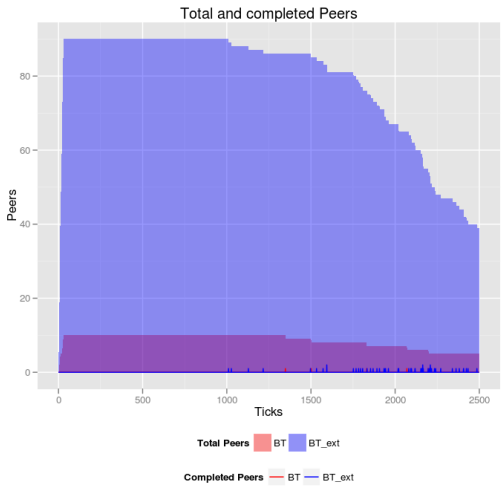


Figure B.78: static,10p,90pc1,1000MB

Figure B.79: **static,1000MB Average number of OU Slots**

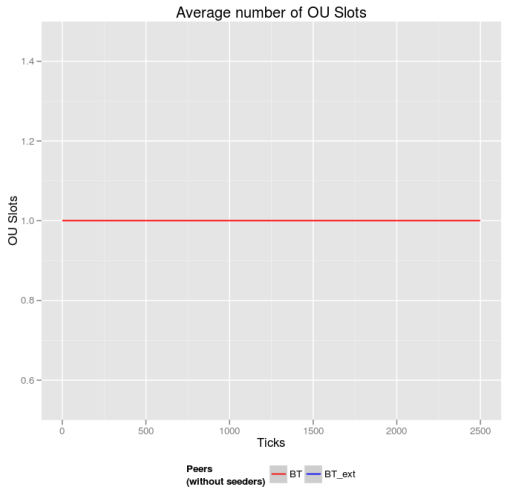


Figure B.80: static,100p,0pc1,1000MB

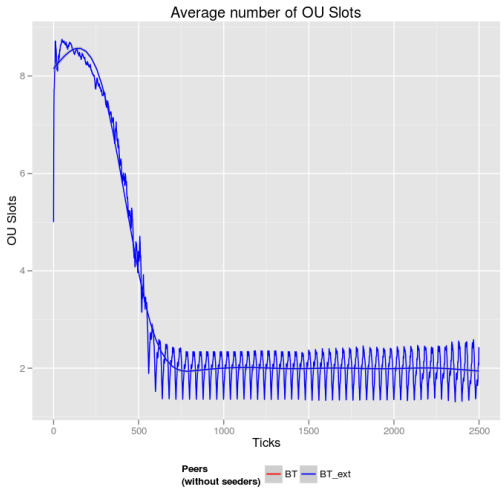


Figure B.81: static,0p,100pc1,1000MB

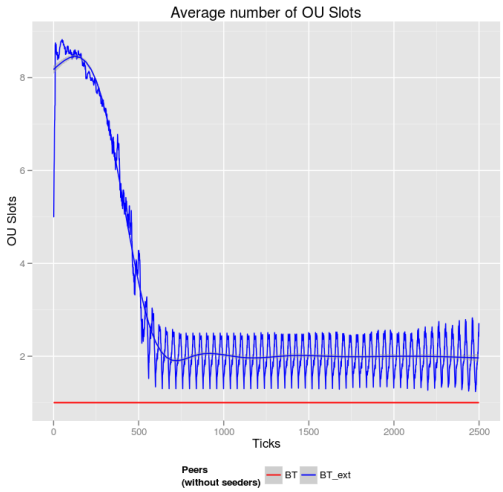


Figure B.82: static,50p,50pc1,1000MB

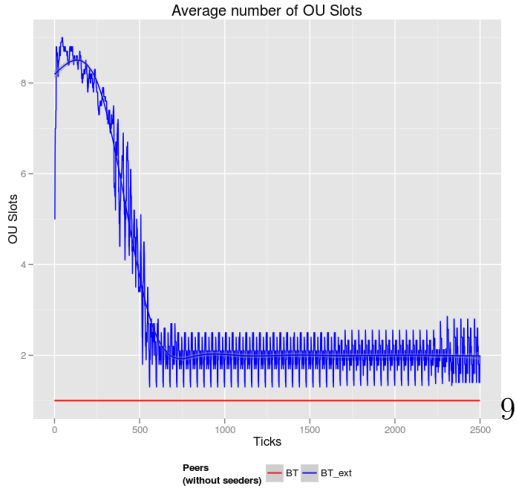


Figure B.83: static,90p,10pc1,1000MB

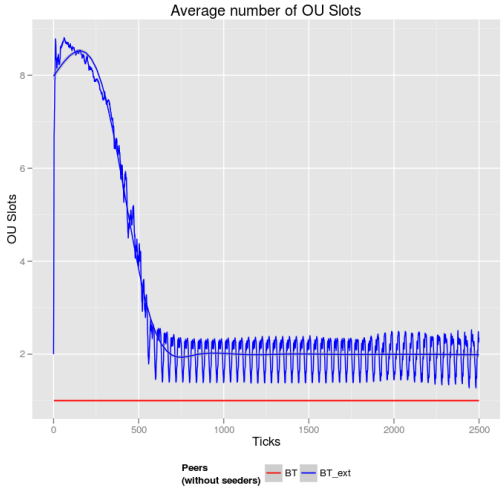


Figure B.84: static,10p,90pc1,1000MB

Figure B.85: **static,1000MB Average number of TFT Slots**

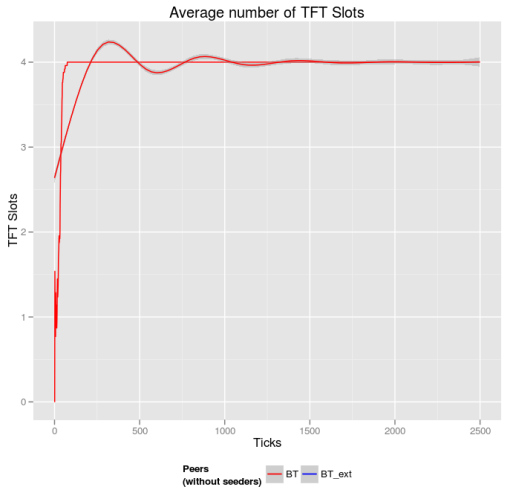


Figure B.86: static,100p,0pc1,1000MB

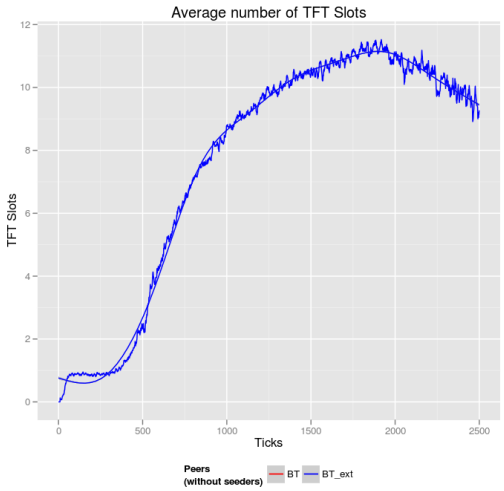


Figure B.87: static,0p,100pc1,1000MB

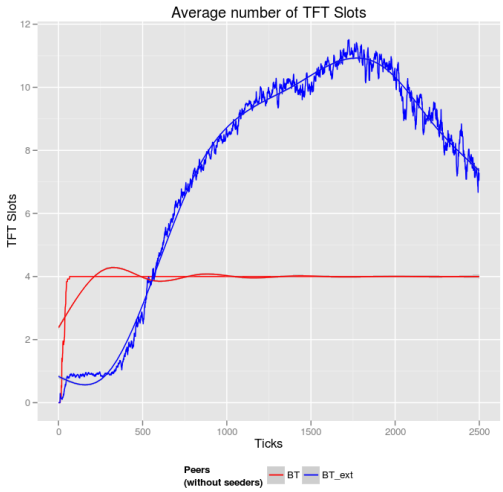


Figure B.88: static,50p,50pc1,1000MB

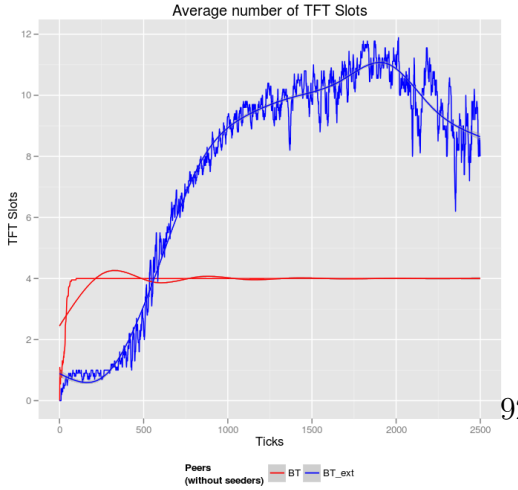


Figure B.89: static,90p,10pc1,1000MB

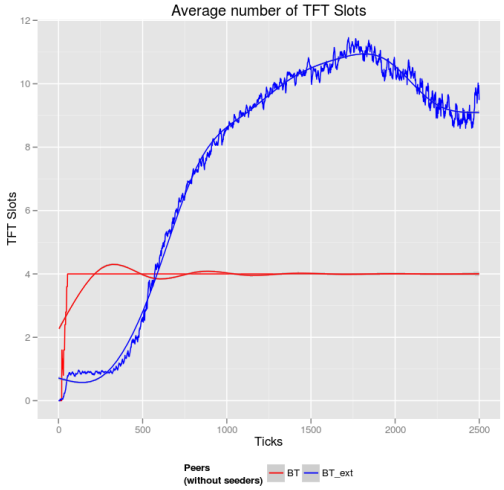


Figure B.90: static,10p,90pc1,1000MB

Figure B.91: **static,1000MB Scaled upload rate**

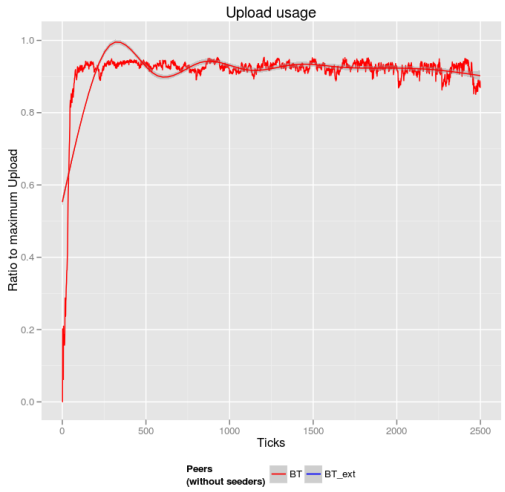


Figure B.92: static,100p,0pc1,1000MB

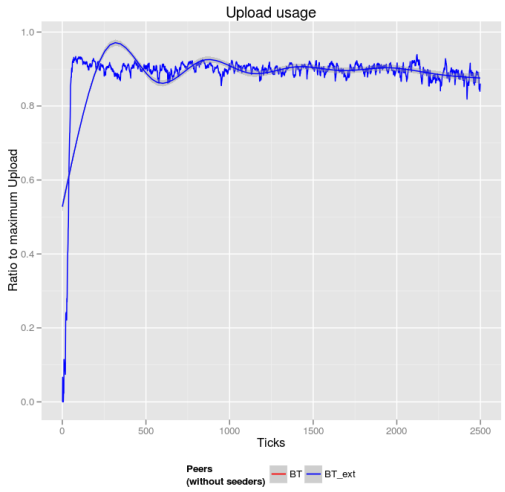


Figure B.93: static,0p,100pc1,1000MB

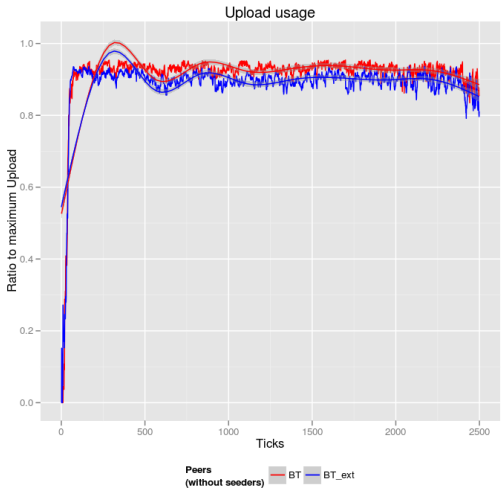


Figure B.94: static,50p,50pc1,1000MB

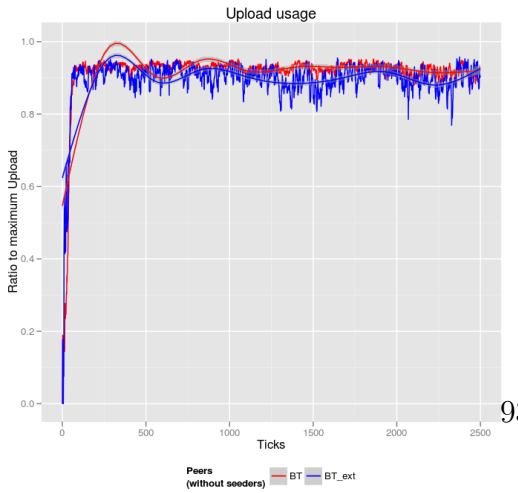


Figure B.95: static,90p,10pc1,1000MB

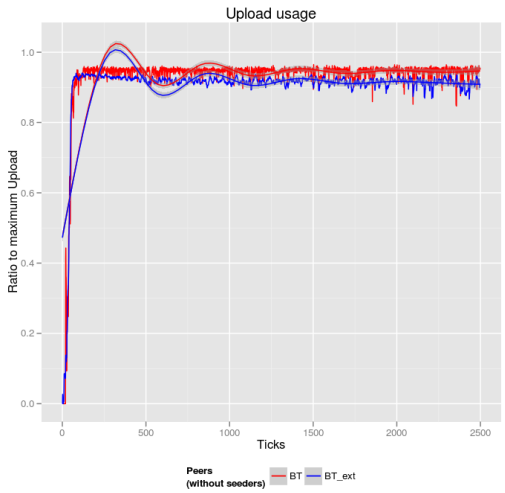


Figure B.96: static,10p,90pc1,1000MB

Figure B.97: **static,1000MB Scaled download rate**

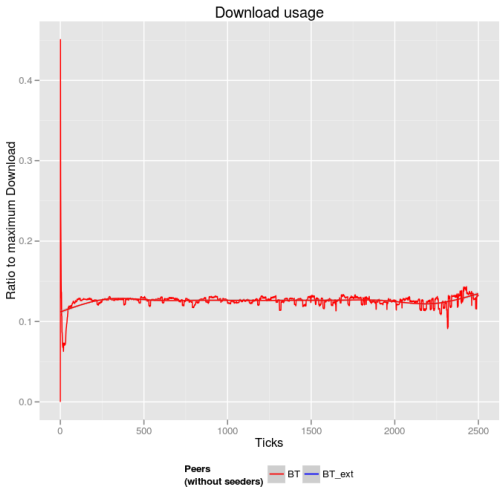


Figure B.98: static,100p,0pc1,1000MB

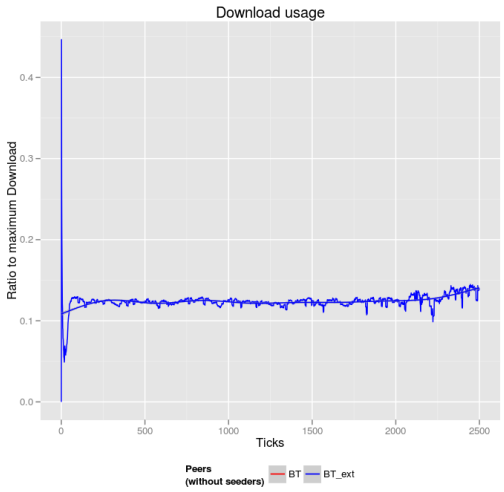


Figure B.99: static,0p,100pc1,1000MB

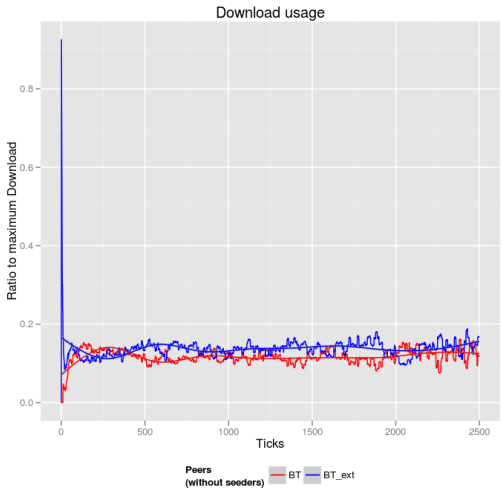


Figure B.100: static,50p,50pc1,1000MB

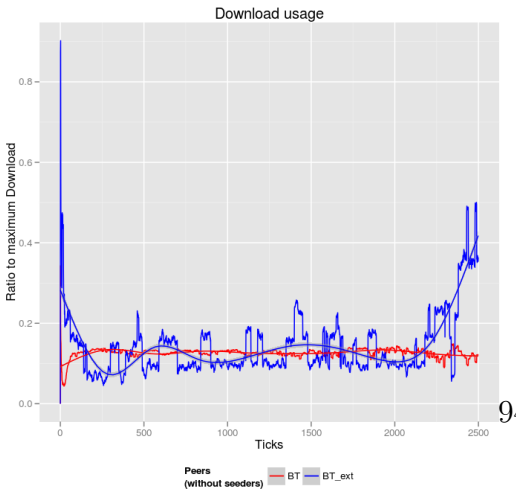


Figure B.101: static,90p,10pc1,1000MB

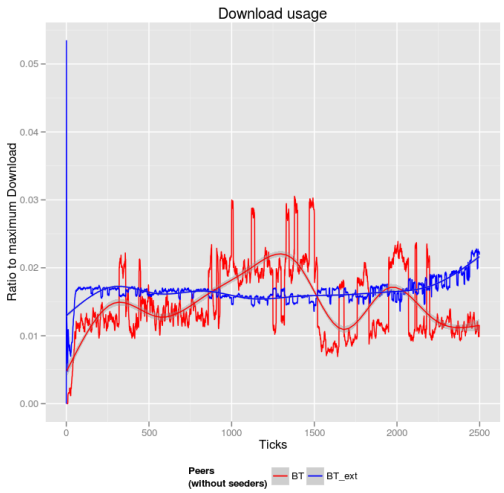


Figure B.102: static,10p,90pc1,1000MB

Figure B.103: **static,1000MB Peer Count**

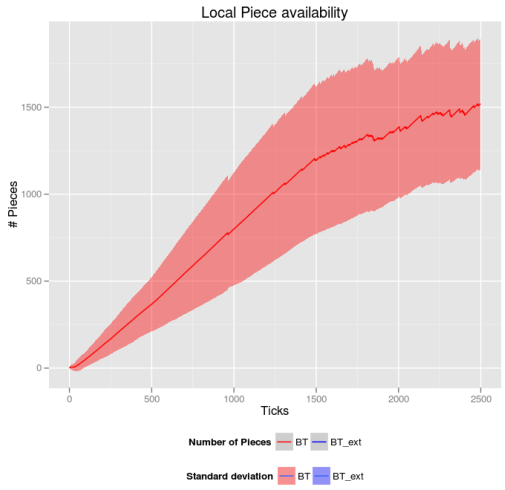


Figure B.104: static,100p,0pc1,1000MB

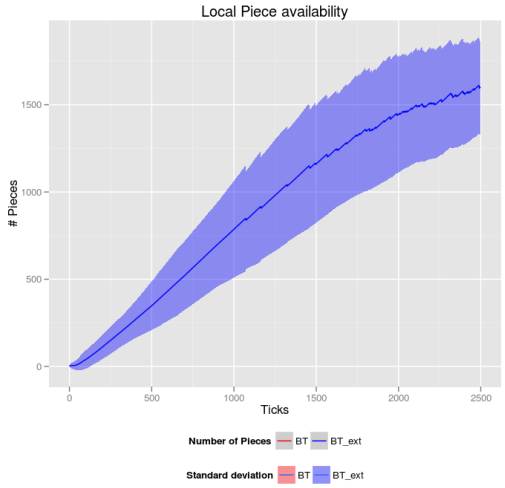


Figure B.105: static,0p,100pc1,1000MB

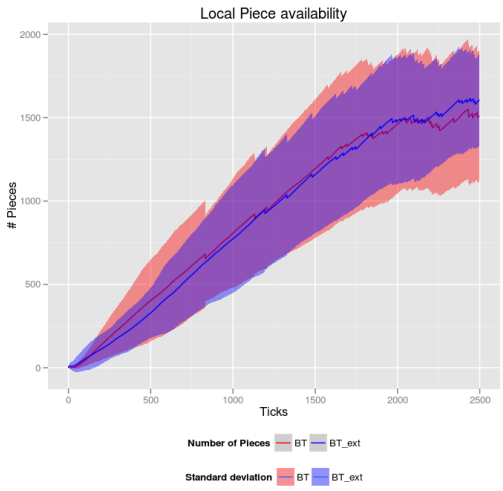


Figure B.106: static,50p,50pc1,1000MB

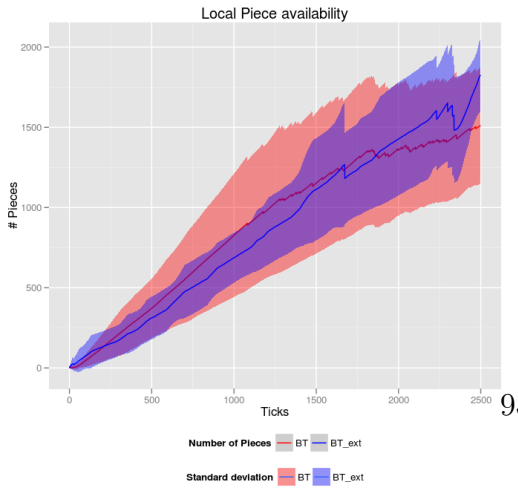


Figure B.107: static,90p,10pc1,1000MB

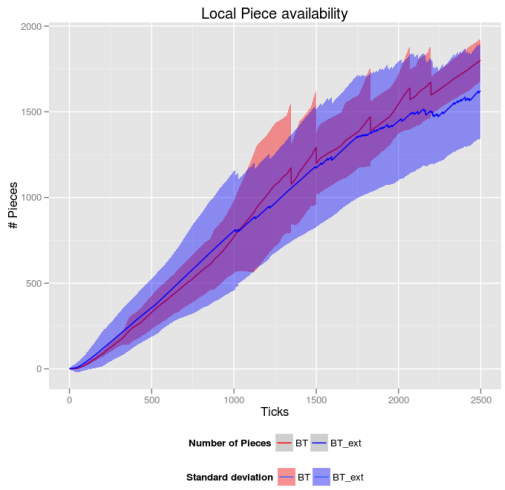


Figure B.108: static,10p,90pc1,1000MB

Appendix C

C.1 Static Public Tracker - Simulation Results

C.1.1 Static private Tracker with small files

C.1.2 Static private Tracker with medium files

C.1.3 Static private Tracker with large files

Figure C.1: **static,10MB Peer Count**

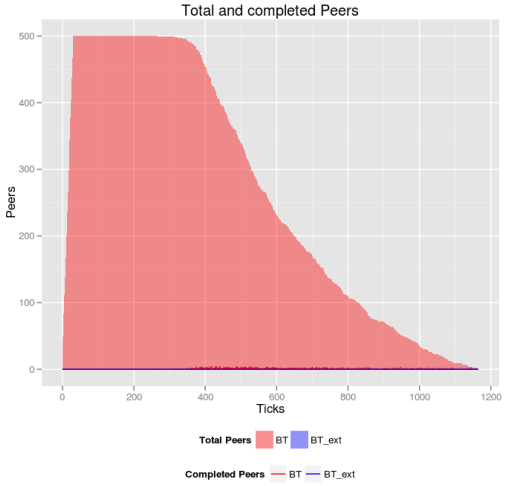


Figure C.2: static,500p,0pc1,10MB

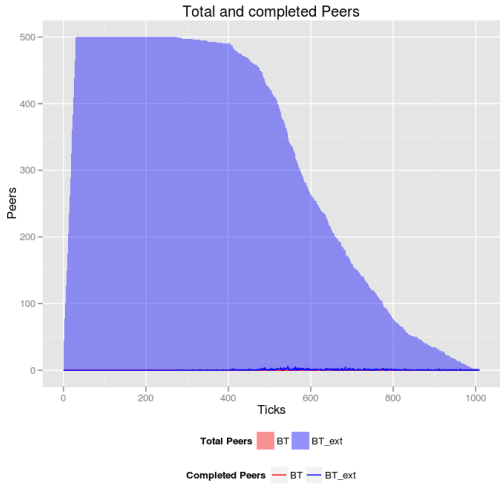


Figure C.3: static,0p,500pc1,10MB

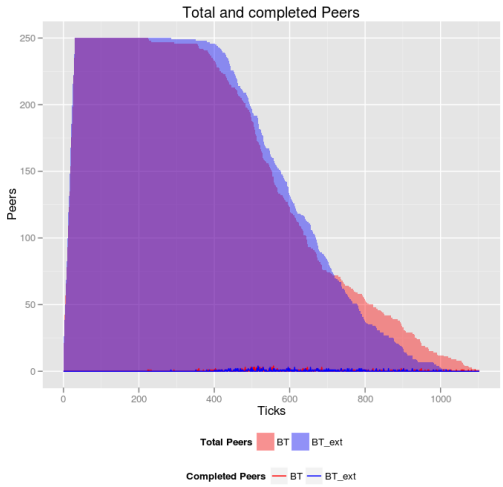


Figure C.4: static,250p,250pc1,10MB

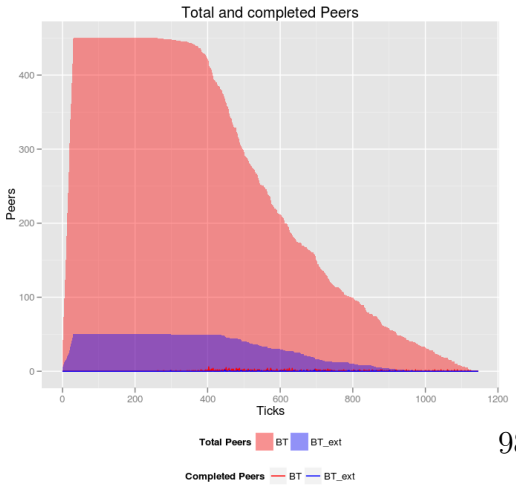


Figure C.5: static,450p,50pc1,10MB

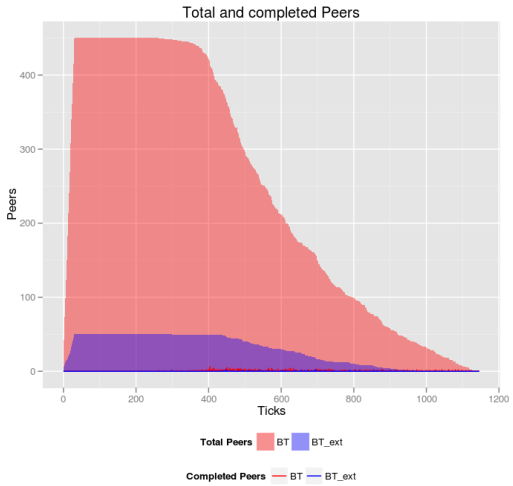


Figure C.6: static,50p,450pc1,10MB

Figure C.7: **static,10MB Average number of OU Slots**

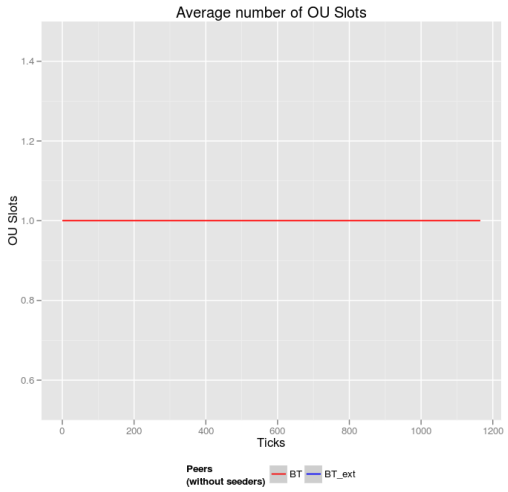


Figure C.8: static,500p,0pc1,10MB

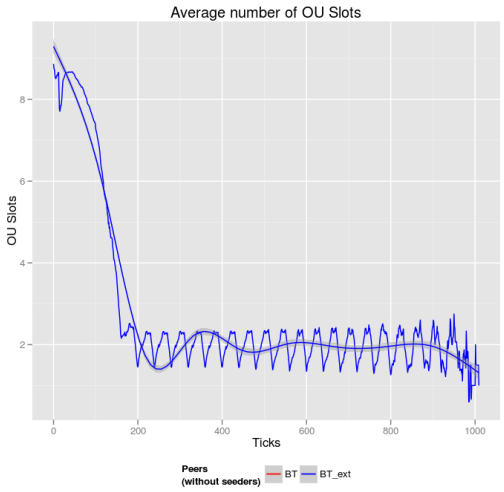


Figure C.9: static,0p,500pc1,10MB

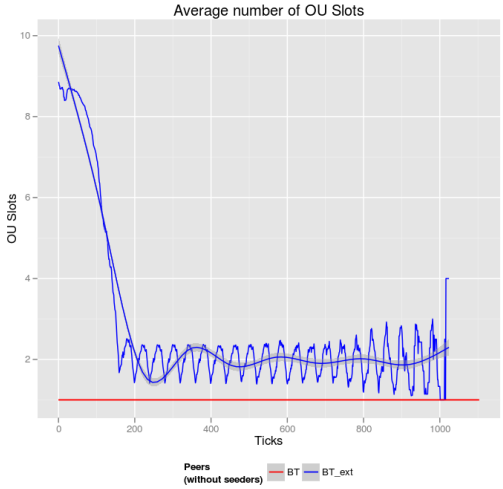


Figure C.10: static,250p,250pc1,10MB

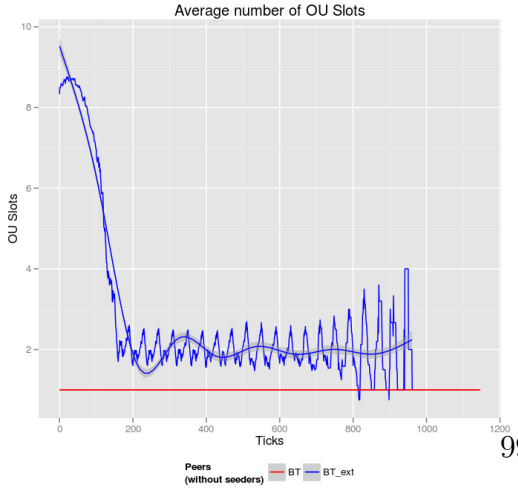


Figure C.11: static,450p,50pc1,10MB

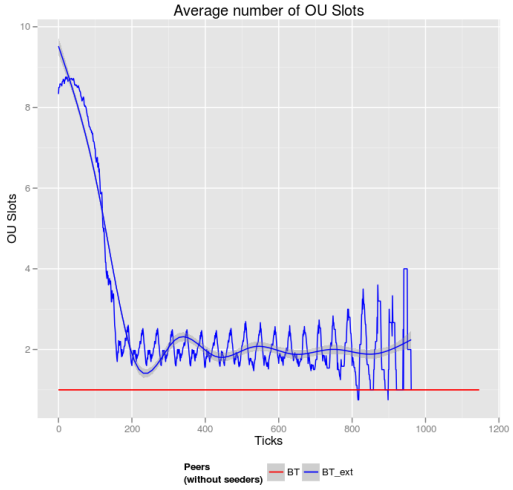


Figure C.12: static,50p,450pc1,10MB

Figure C.13: **static,10MB Average number of TFT Slots**

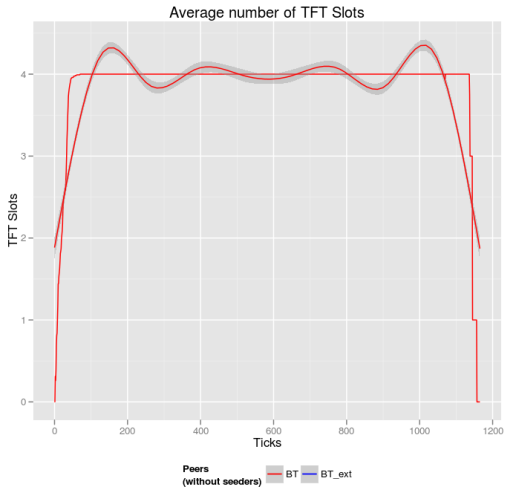


Figure C.14: static,500p,0pc1,10MB

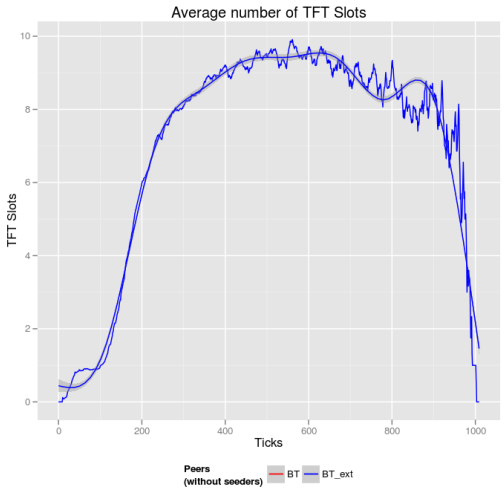


Figure C.15: static,0p,500pc1,10MB

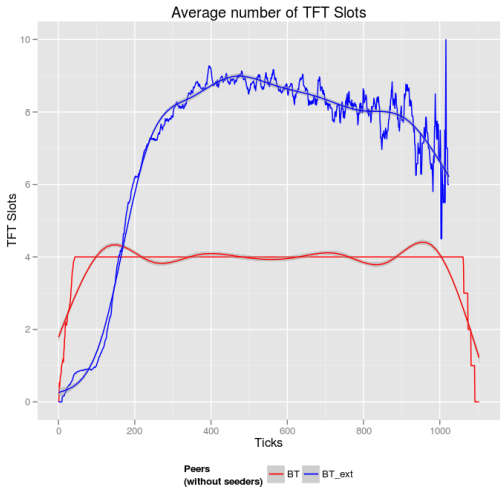


Figure C.16: static,250p,250pc1,10MB

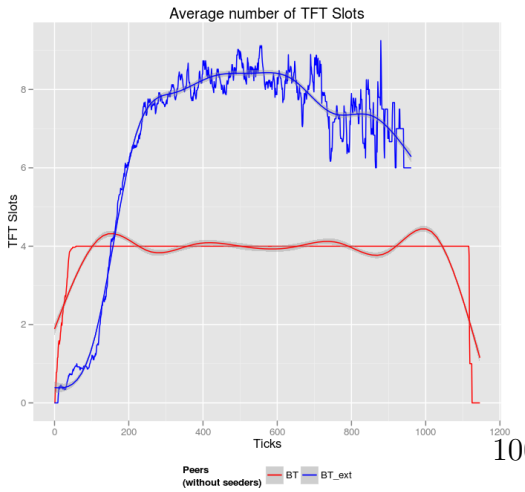


Figure C.17: static,450p,50pc1,10MB

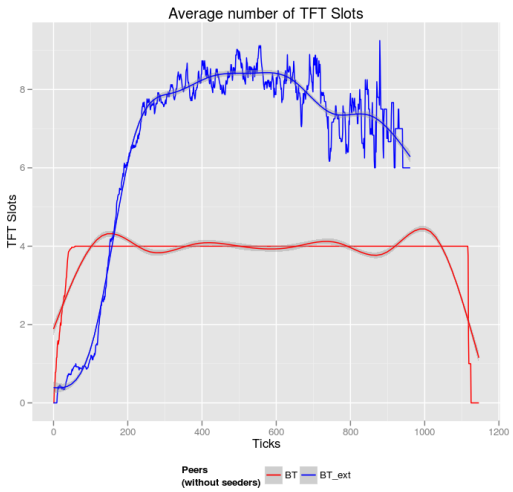


Figure C.18: static,50p,450pc1,10MB

Figure C.19: **static,10MB Scaled upload rate**

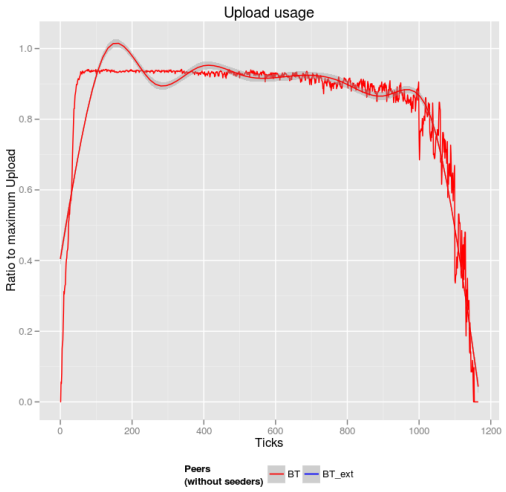


Figure C.20: static,500p,0pc1,10MB

Figure C.21: static,0p,500pc1,10MB

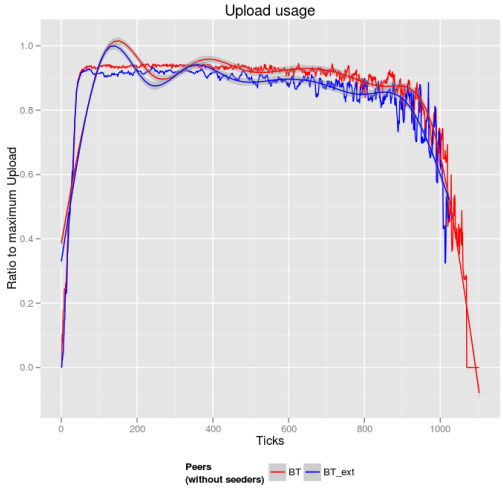


Figure C.22: static,250p,250pc1,10MB

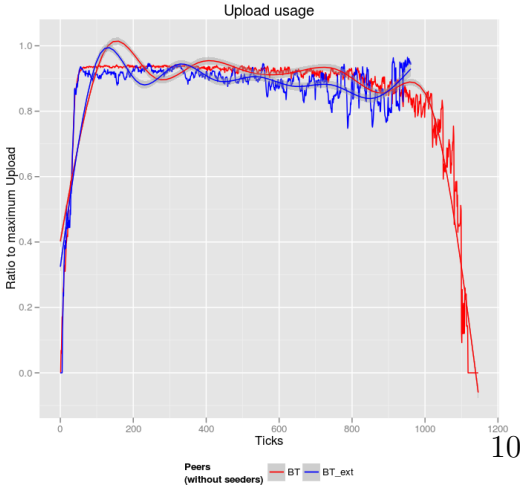


Figure C.23: static,450p,50pc1,10MB

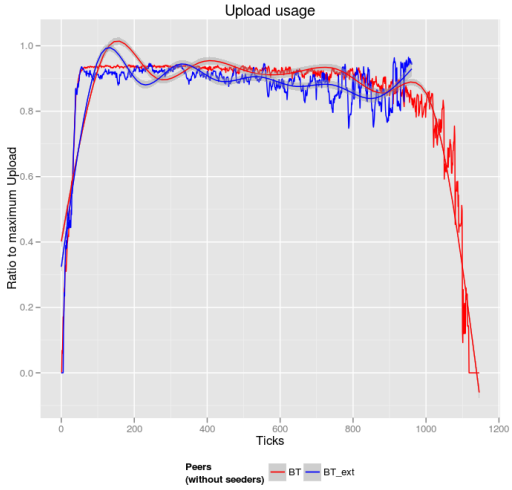


Figure C.24: static,50p,450pc1,10MB

Figure C.25: **static,10MB Scaled download rate**

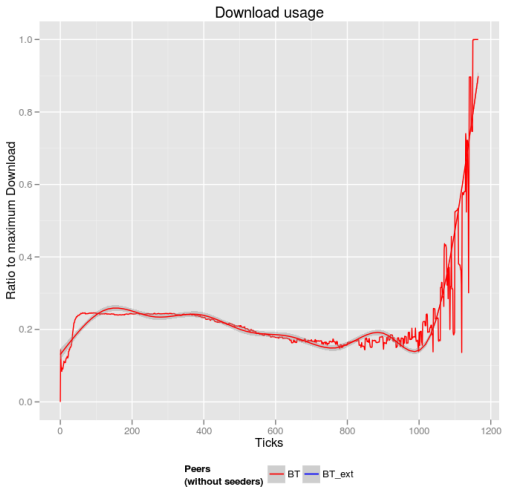


Figure C.26: static,500p,0pc1,10MB

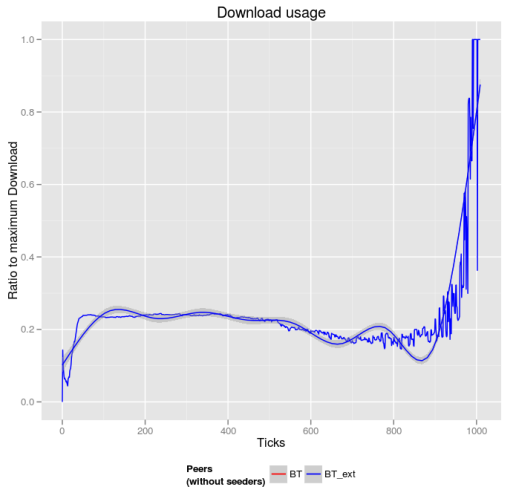


Figure C.27: static,0p,500pc1,10MB

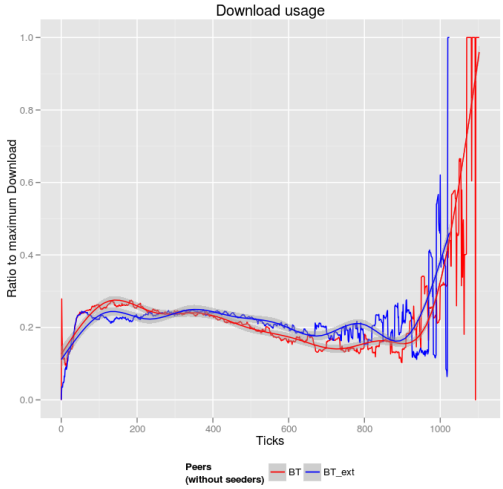


Figure C.28: static,250p,250pc1,10MB

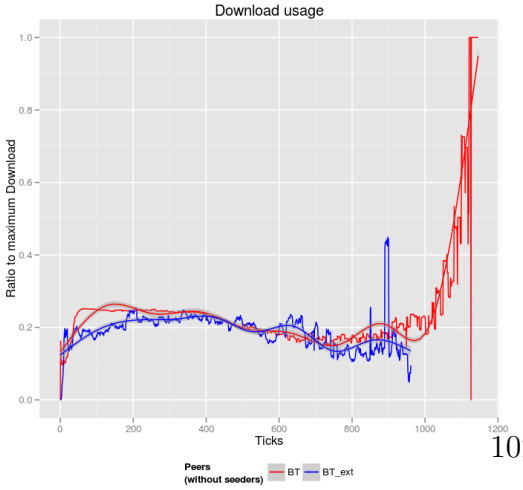


Figure C.29: static,450p,50pc1,10MB

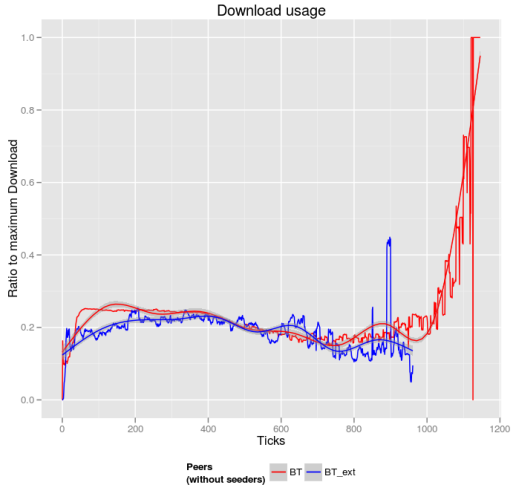


Figure C.30: static,50p,450pc1,10MB

Figure C.31: **static,350MB Peer Count**

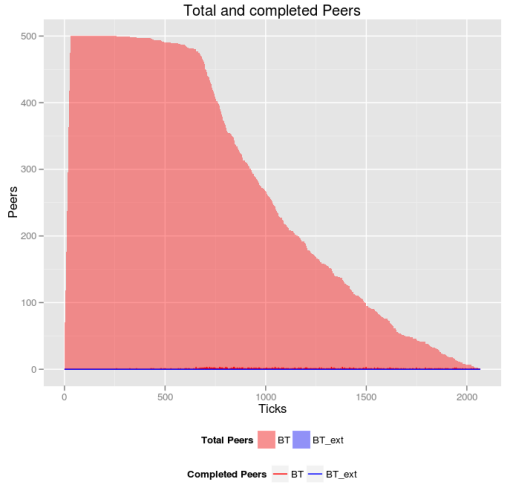


Figure C.32: static,500p,0pc1,350MB

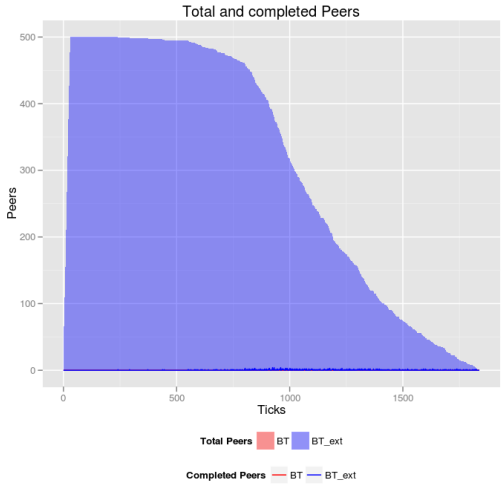


Figure C.33: static,0p,500pc1,350MB

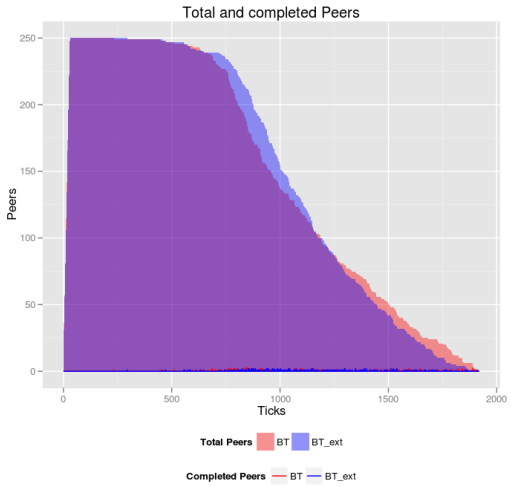


Figure C.34: static,250p,250pc1,350MB

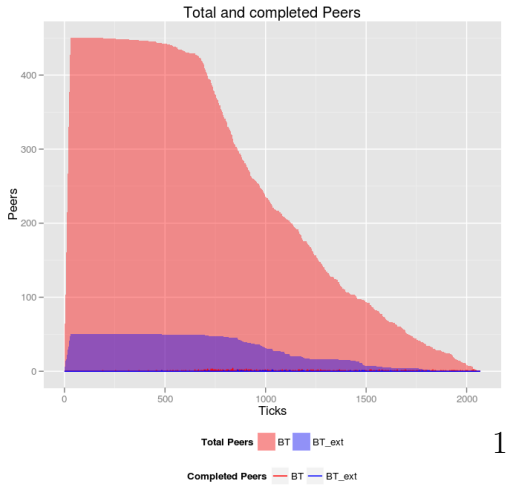


Figure C.35: static,450p,50pc1,350MB

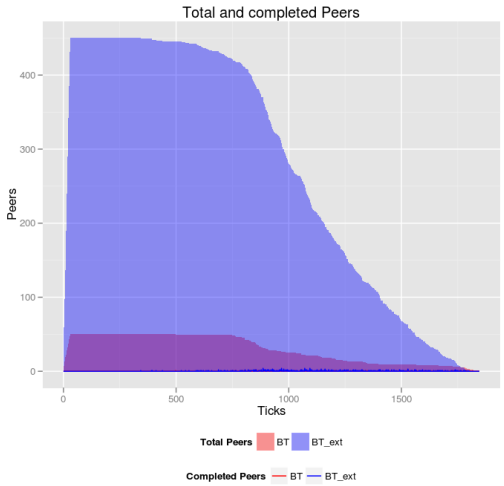


Figure C.36: static,50p,450pc1,350MB

Figure C.37: **static,350MB Average number of OU Slots**

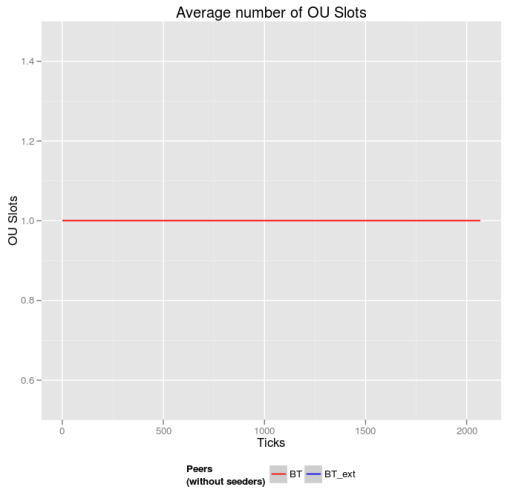


Figure C.38: static,500p,0pc1,350MB

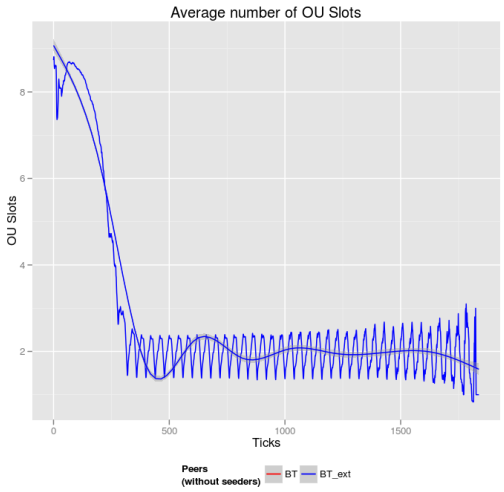


Figure C.39: static,0p,500pc1,350MB

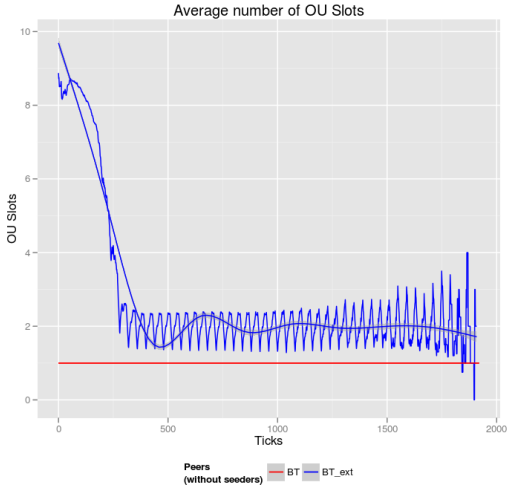


Figure C.40: static,250p,250pc1,350MB

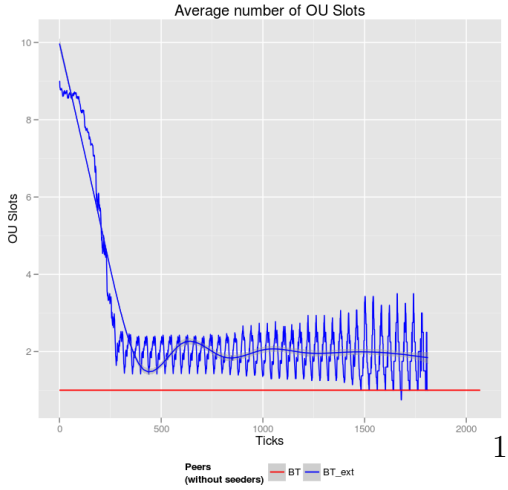


Figure C.41: static,450p,50pc1,350MB

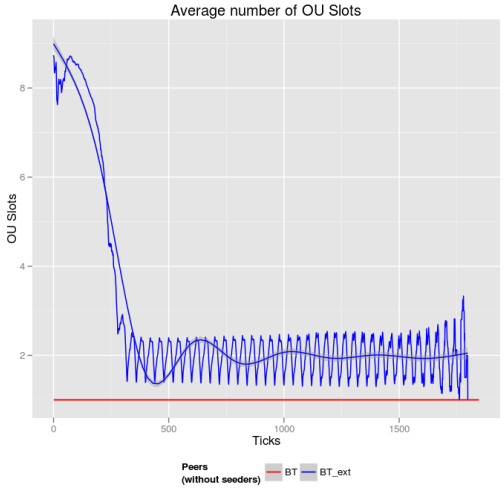


Figure C.42: static,50p,450pc1,350MB

Figure C.43: **static,350MB Average number of TFT Slots**

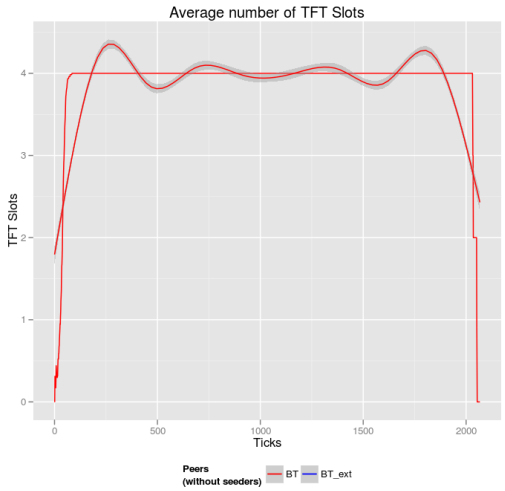


Figure C.44: static,500p,0pc1,350MB

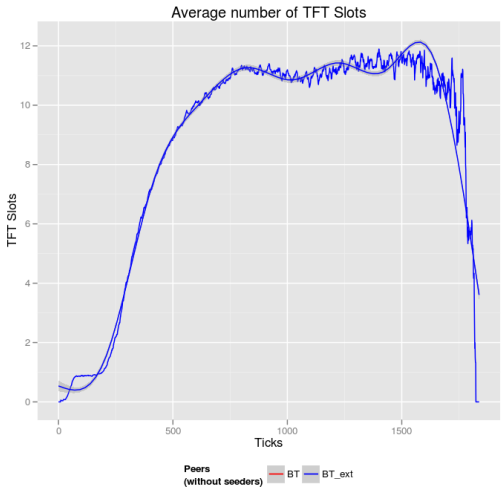


Figure C.45: static,0p,500pc1,350MB

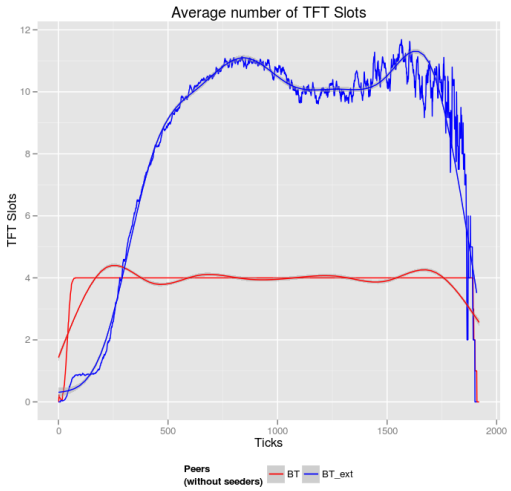


Figure C.46: static,250p,250pc1,350MB

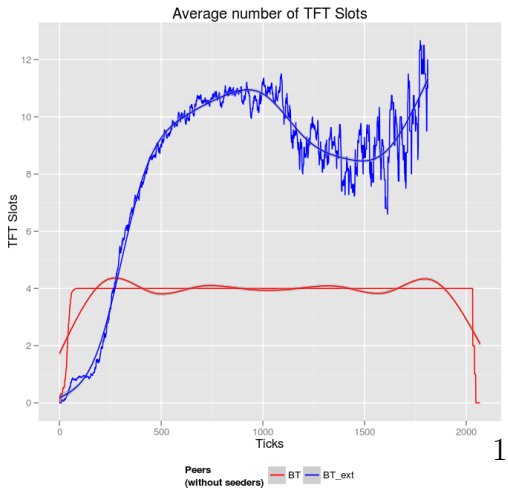


Figure C.47: static,450p,50pc1,350MB

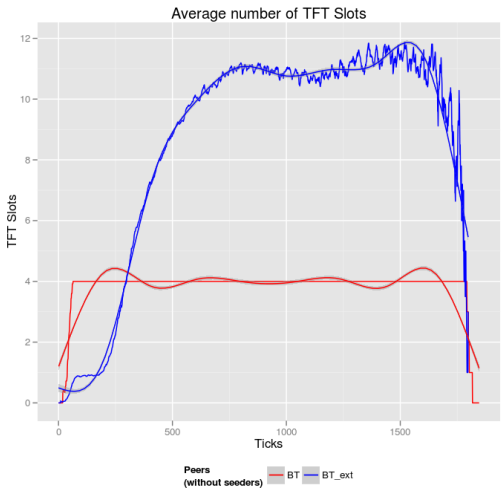


Figure C.48: static,50p,450pc1,350MB

Figure C.49: **static,350MB Scaled upload rate**

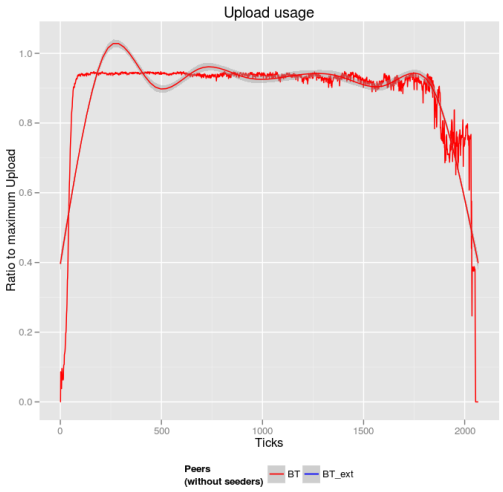


Figure C.50: static,500p,0pc1,350MB

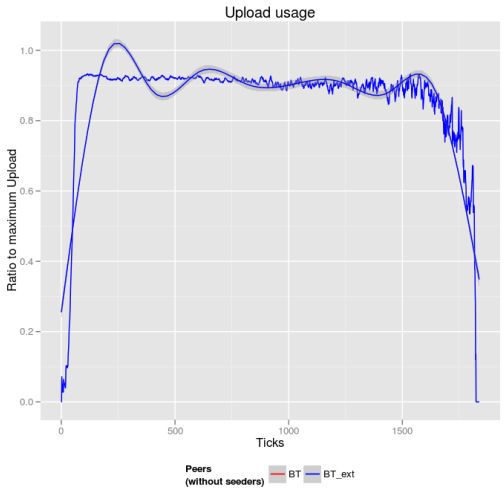


Figure C.51: static,0p,500pc1,350MB

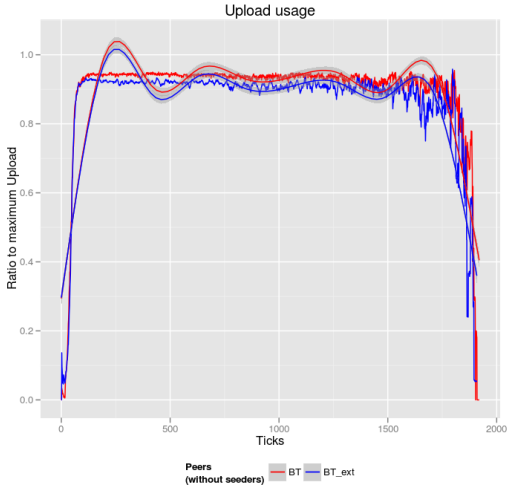


Figure C.52: static,250p,250pc1,350MB

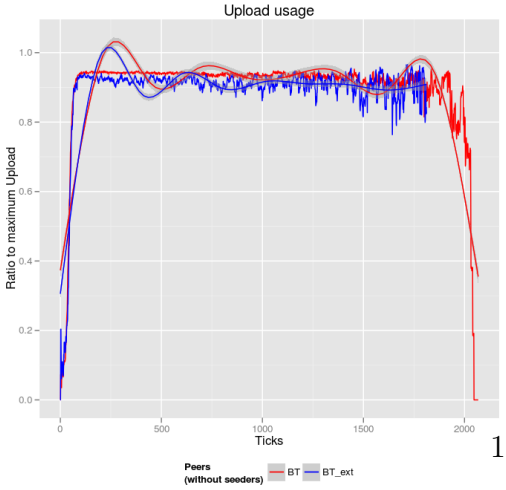


Figure C.53: static,450p,50pc1,350MB

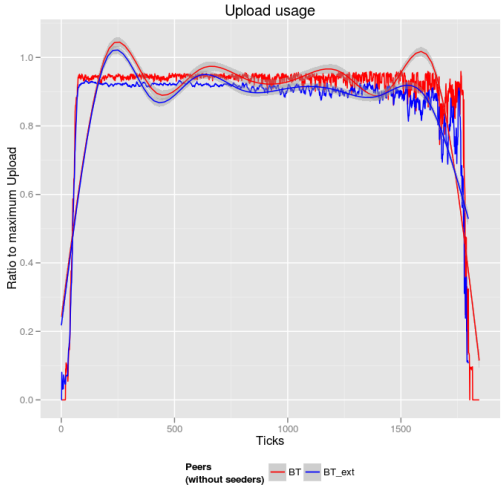


Figure C.54: static,50p,450pc1,350MB

Figure C.55: **static,350MB Scaled download rate**

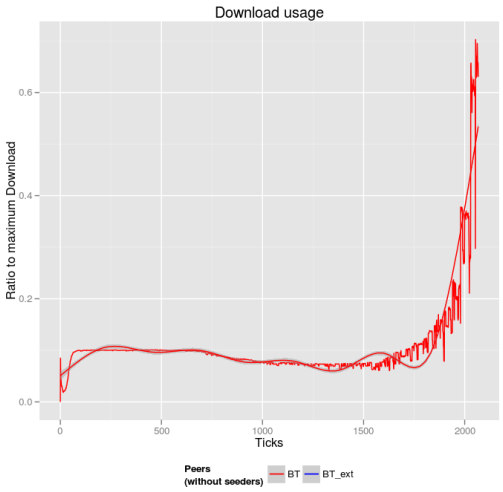


Figure C.56: static,500p,0pc1,350MB

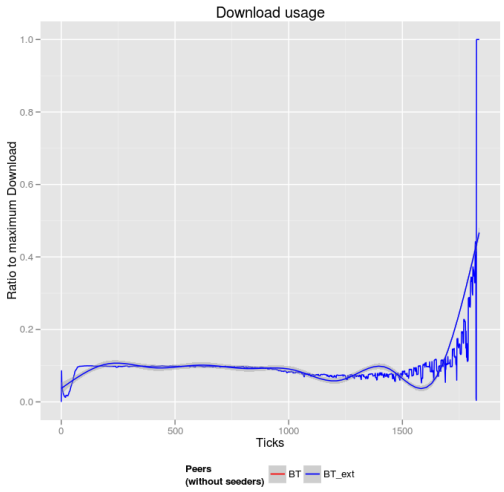


Figure C.57: static,0p,500pc1,350MB

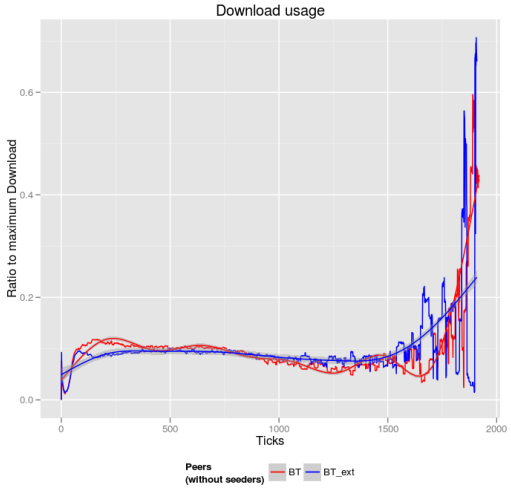


Figure C.58: static,250p,250pc1,350MB

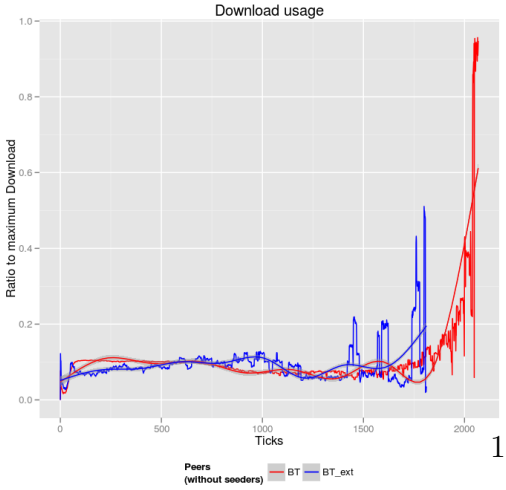


Figure C.59: static,450p,50pc1,350MB

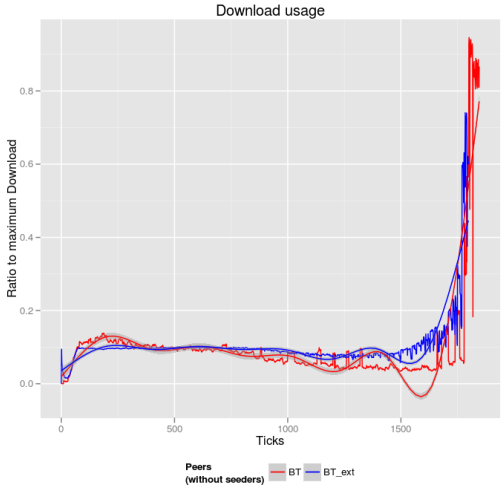


Figure C.60: static,50p,450pc1,350MB

Figure C.61: **static,1000MB Peer Count**

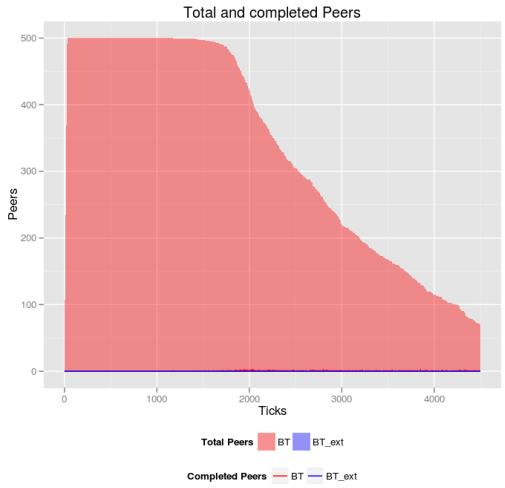


Figure C.62: static,500p,0pc1,1000MB

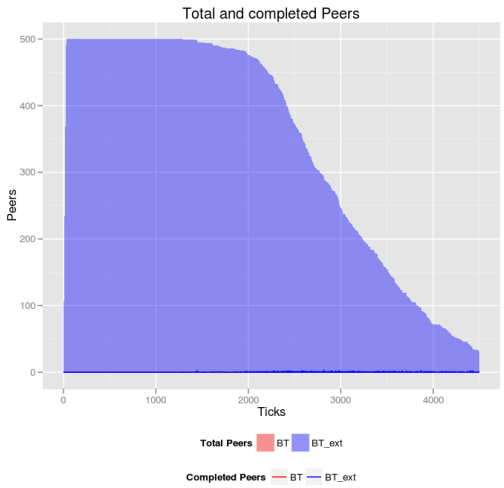


Figure C.63: static,0p,500pc1,1000MB

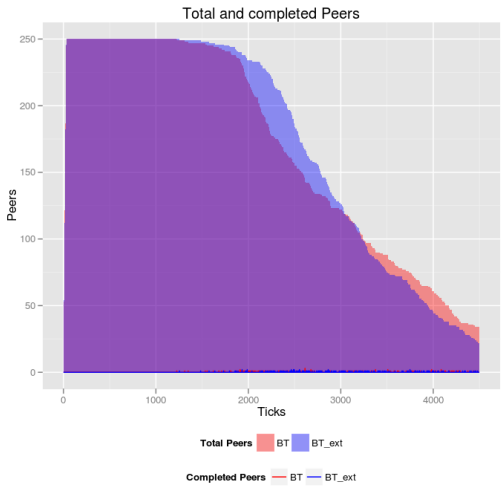


Figure C.64: static,250p,250pc1,1000MB

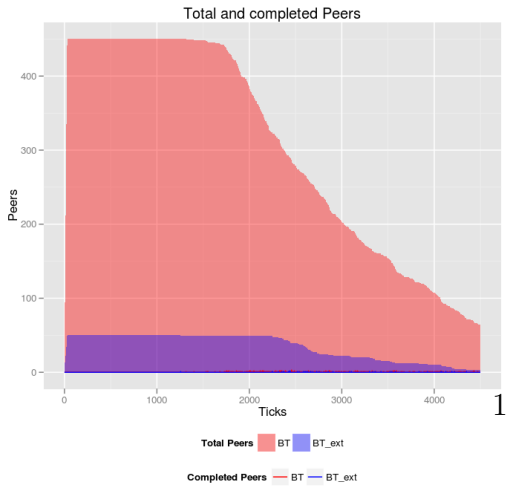


Figure C.65: static,450p,50pc1,1000MB

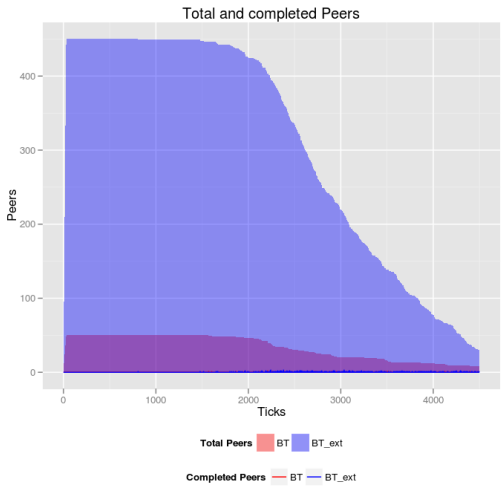


Figure C.66: static,50p,450pc1,1000MB

Figure C.67: **static,1000MB Average number of OU Slots**

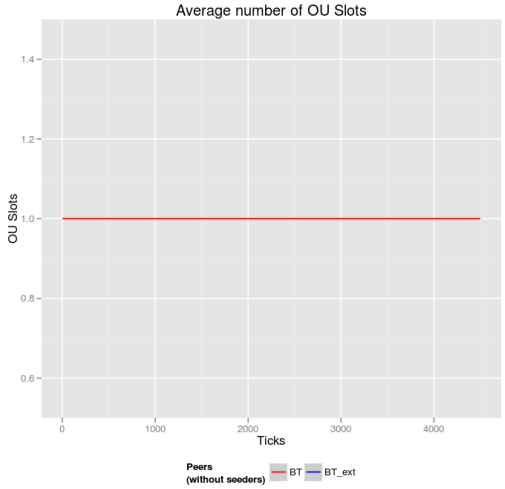


Figure C.68: static,500p,0pc1,1000MB

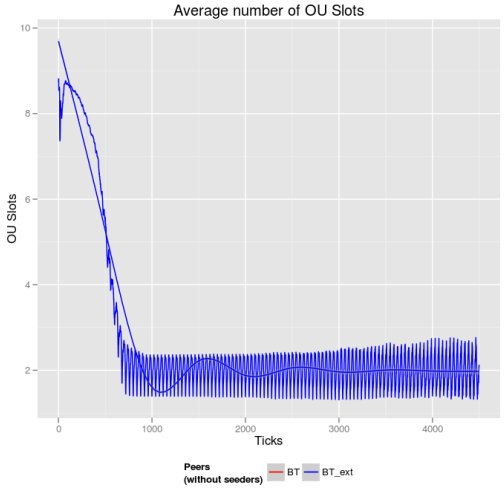


Figure C.69: static,0p,500pc1,1000MB

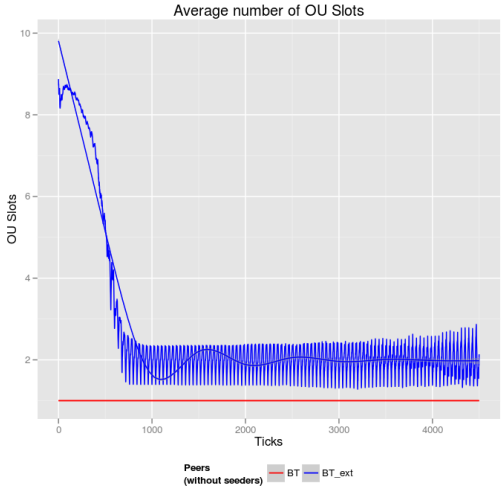


Figure C.70: static,250p,250pc1,1000MB

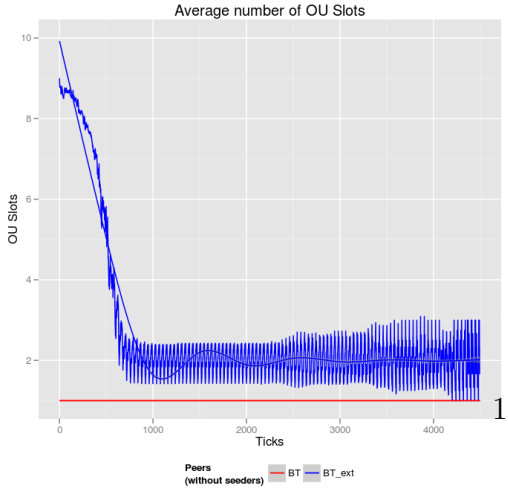


Figure C.71: static,450p,50pc1,1000MB

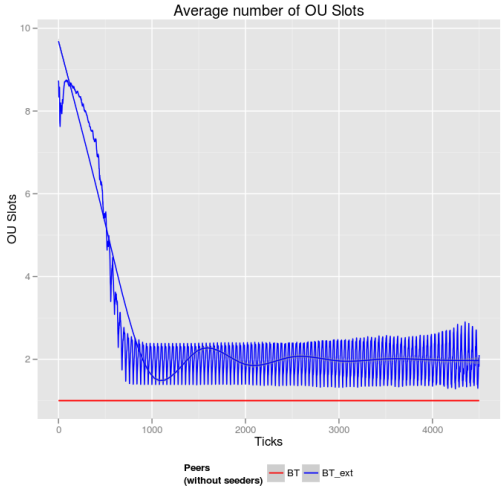


Figure C.72: static,50p,450pc1,1000MB

Figure C.73: **static,1000MB Average number of TFT Slots**

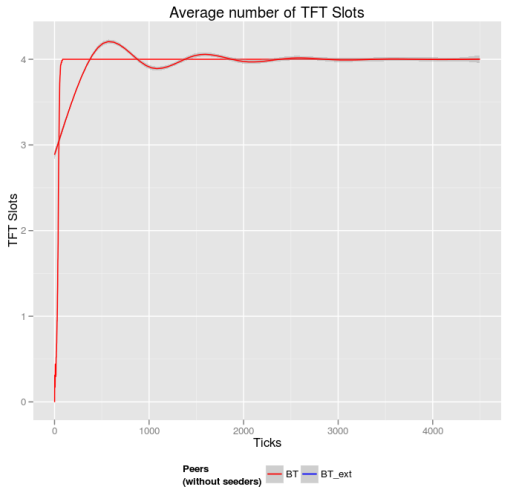


Figure C.74: static,500p,0pc1,1000MB

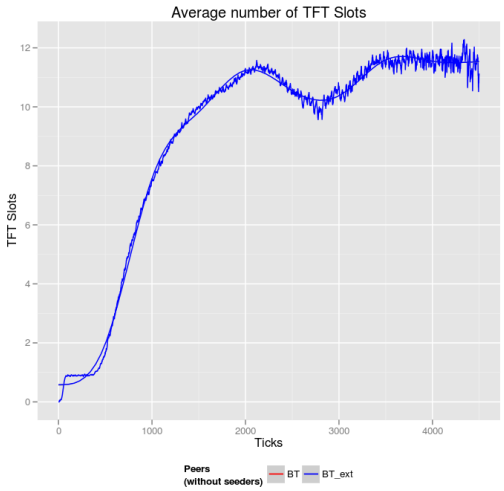


Figure C.75: static,0p,500pc1,1000MB

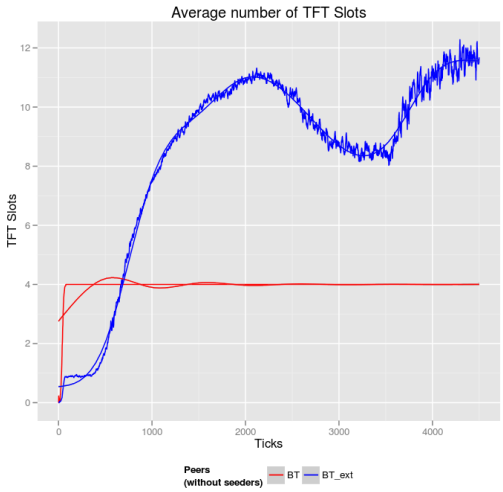


Figure C.76: static,250p,250pc1,1000MB

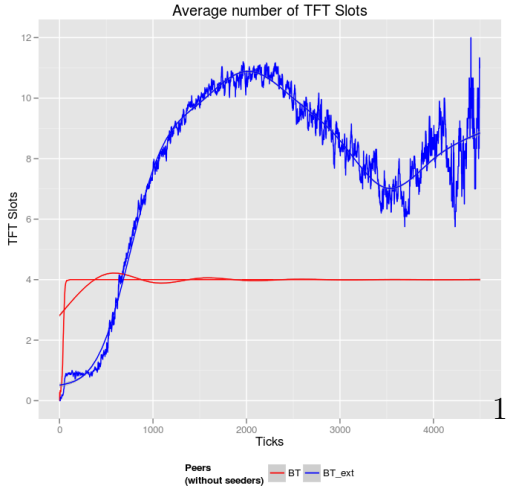


Figure C.77: static,450p,50pc1,1000MB

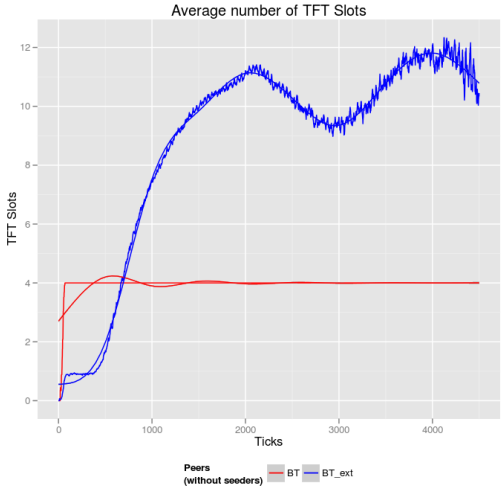


Figure C.78: static,50p,450pc1,1000MB

Figure C.79: **static,1000MB Scaled upload rate**

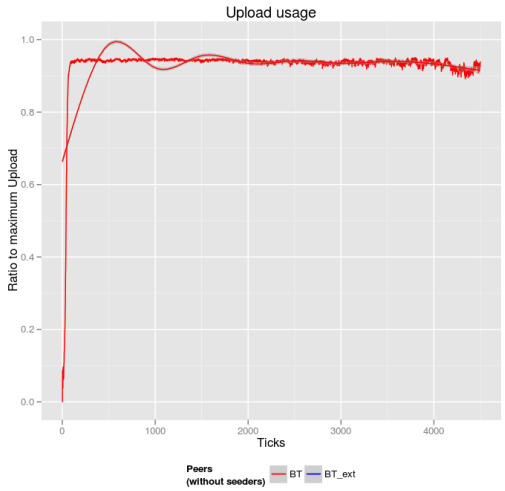


Figure C.80: static,500p,0pc1,1000MB

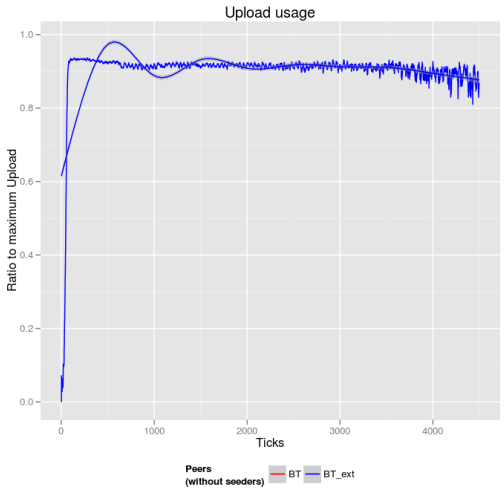


Figure C.81: static,0p,500pc1,1000MB

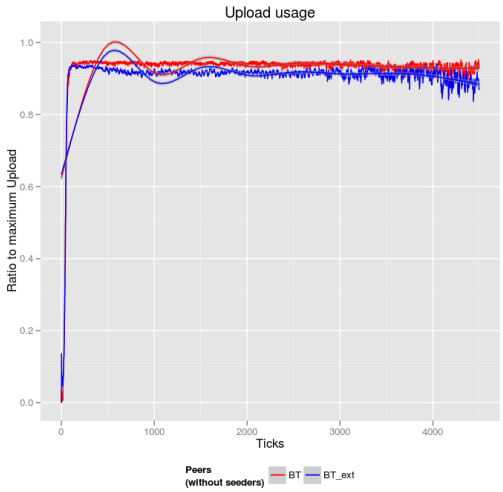


Figure C.82: static,250p,250pc1,1000MB

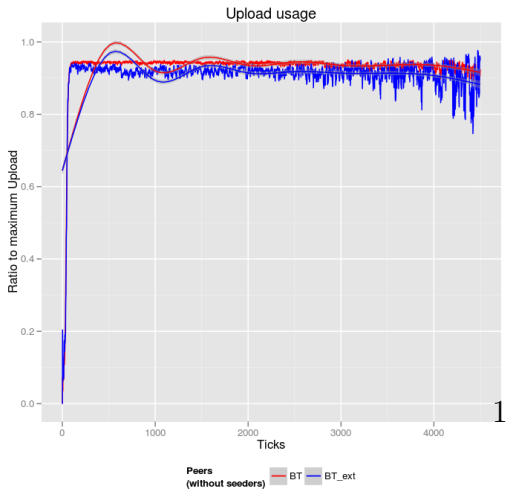


Figure C.83: static,450p,50pc1,1000MB

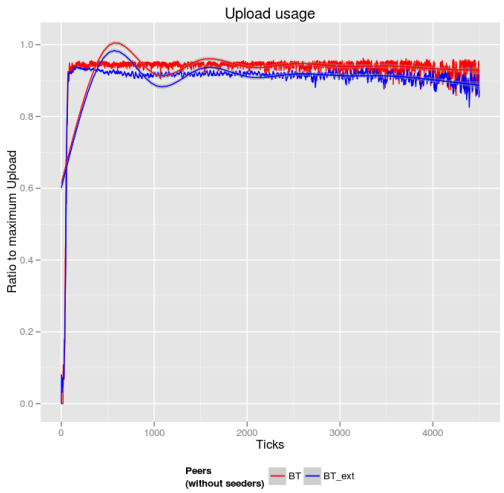


Figure C.84: static,50p,450pc1,1000MB

Figure C.85: **static,1000MB Scaled download rate**

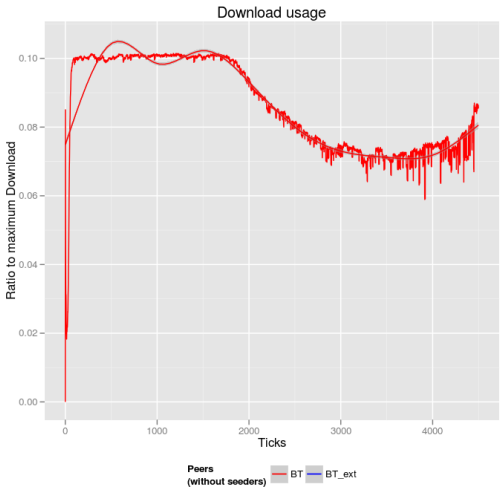


Figure C.86: static,500p,0pc1,1000MB

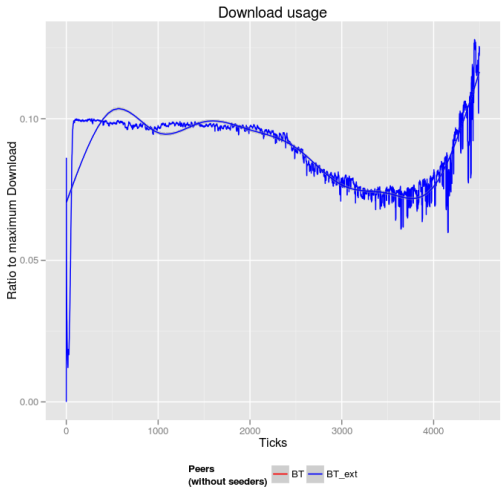


Figure C.87: static,0p,500pc1,1000MB

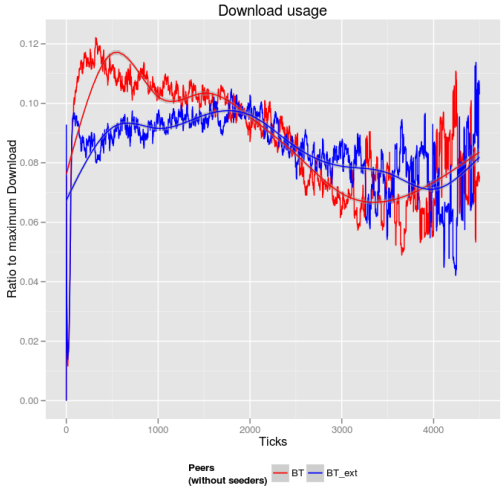


Figure C.88: static,250p,250pc1,1000MB

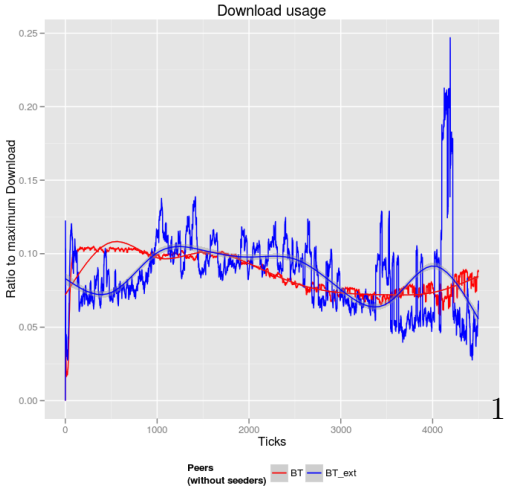


Figure C.89: static,450p,50pc1,1000MB

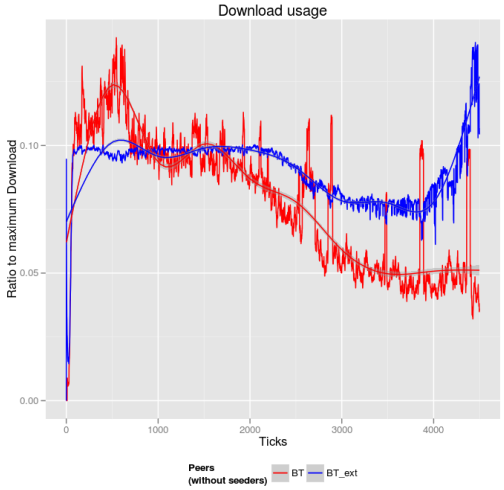


Figure C.90: static,50p,450pc1,1000MB

Appendix D

D.1 Dynamic Private Tracker - Simulation Results

D.1.1 Dynamic private Tracker with small files

D.1.2 Dynamic private Tracker with medium files

D.1.3 Dynamic private Tracker with large files

Figure D.1: **dynamic,10MB Peer Count**

Figure D.2: dynamic,100p,0pc1,10MB

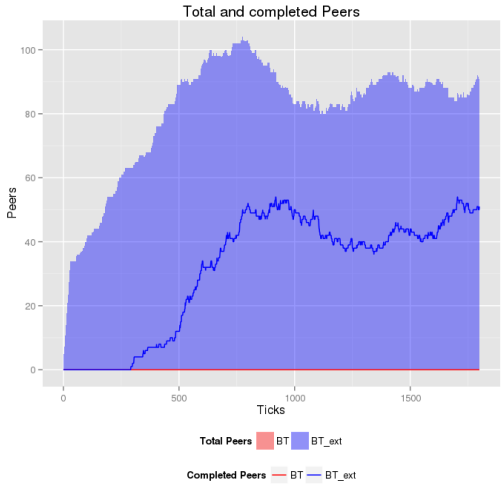


Figure D.3: dynamic,0p,100pc1,10MB

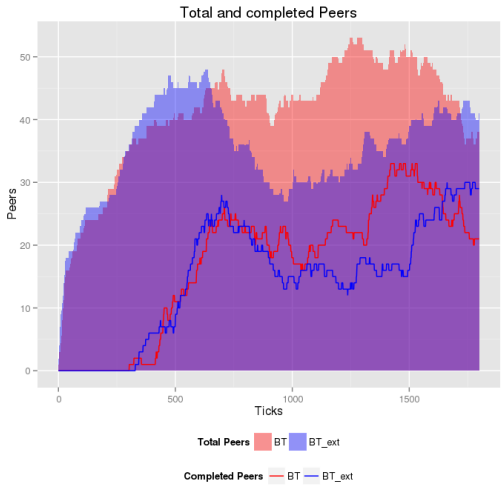


Figure D.4: dynamic,50p,50pc1,10MB

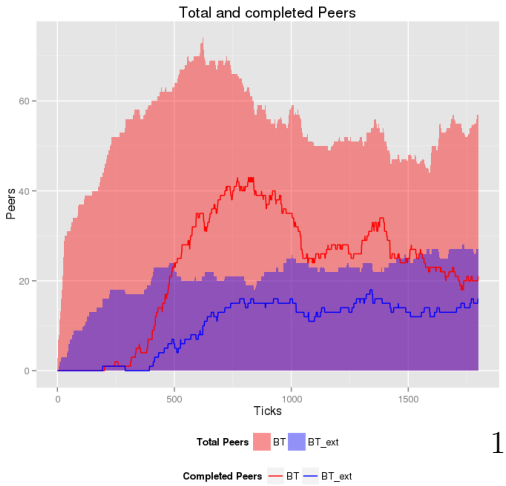


Figure D.5: dynamic,90p,10pc1,10MB

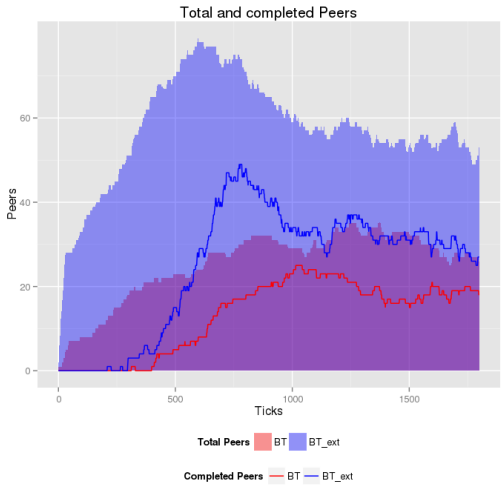


Figure D.6: dynamic,10p,90pc1,10MB

Figure D.7: **dynamic,10MB Average number of OU Slots**

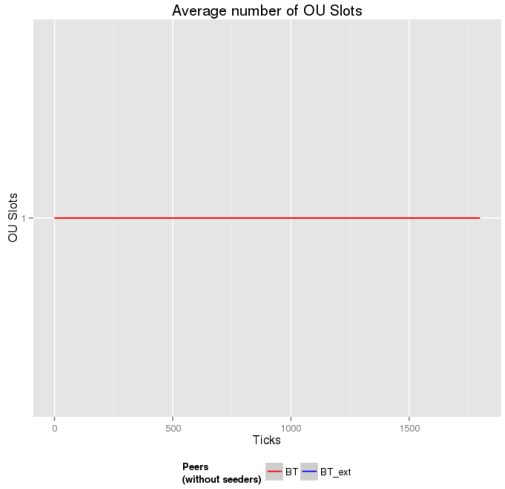


Figure D.8: dynamic,100p,0pc1,10MB

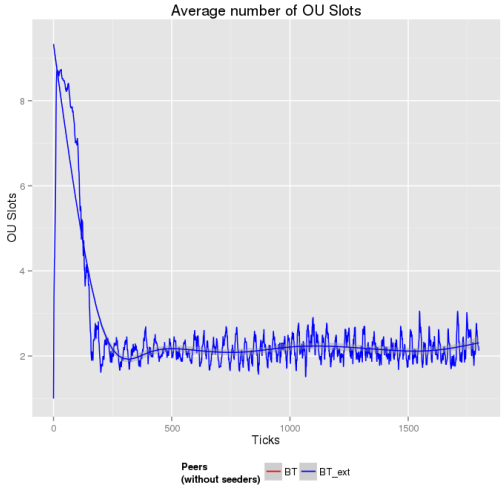


Figure D.9: dynamic,0p,100pc1,10MB

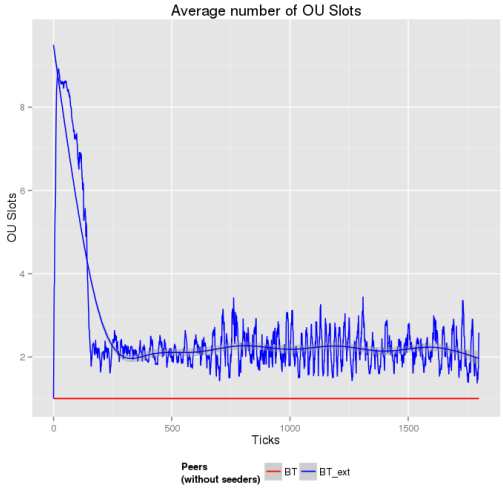


Figure D.10: dynamic,50p,50pc1,10MB

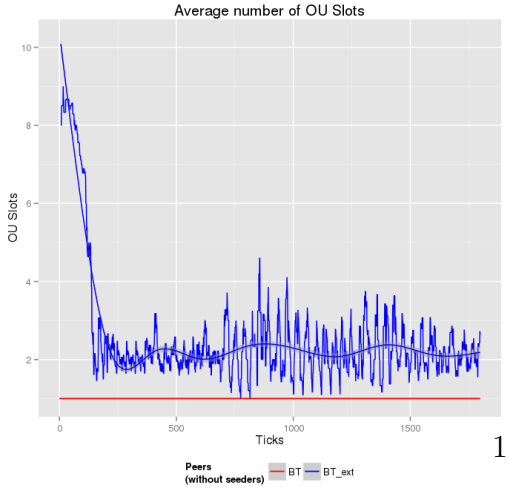


Figure D.11: dynamic,90p,10pc1,10MB

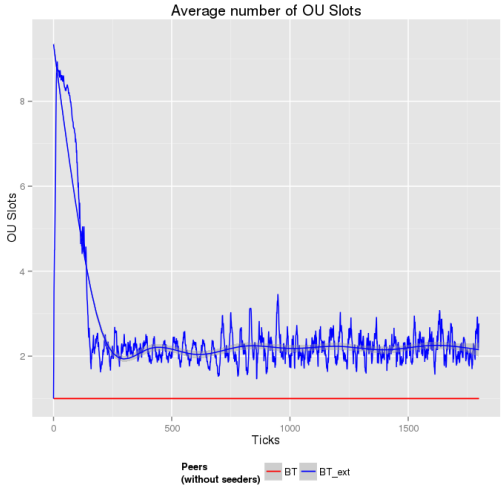


Figure D.12: dynamic,10p,90pc1,10MB

Figure D.13: **dynamic,10MB Average number of TFT Slots**

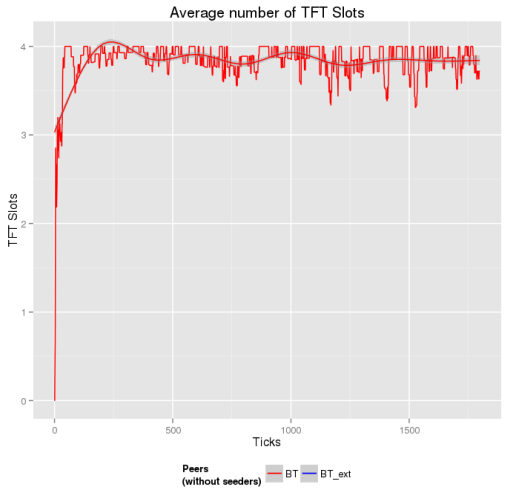


Figure D.14: dynamic,100p,0pc1,10MB

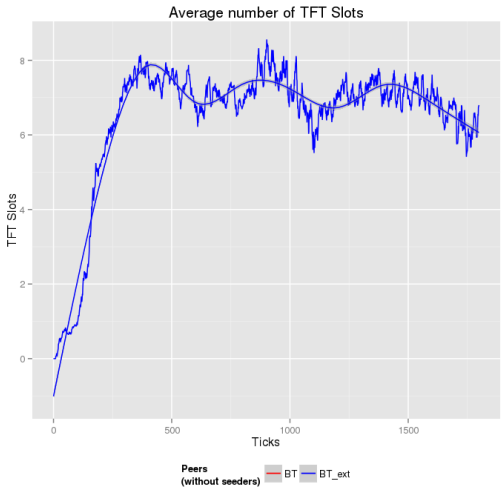


Figure D.15: dynamic,0p,100pc1,10MB

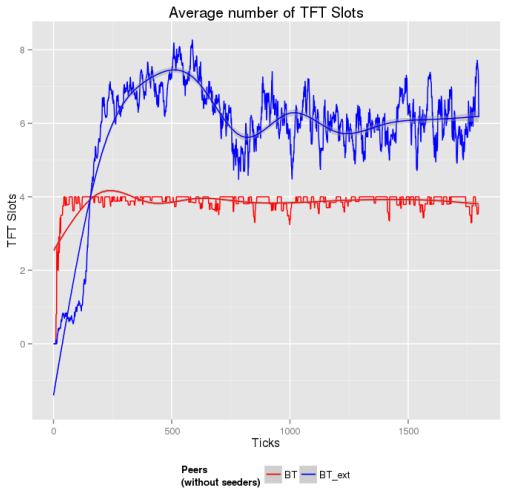


Figure D.16: dynamic,50p,50pc1,10MB

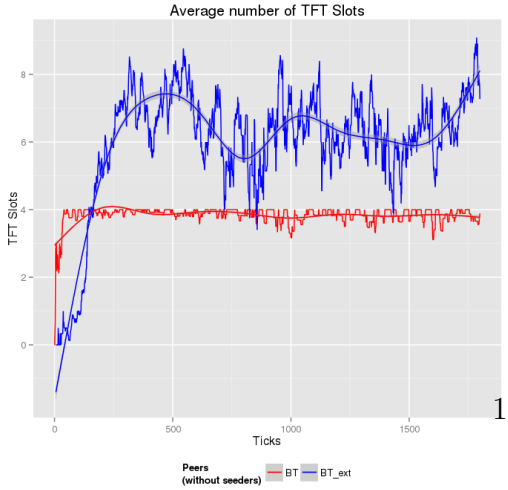


Figure D.17: dynamic,90p,10pc1,10MB

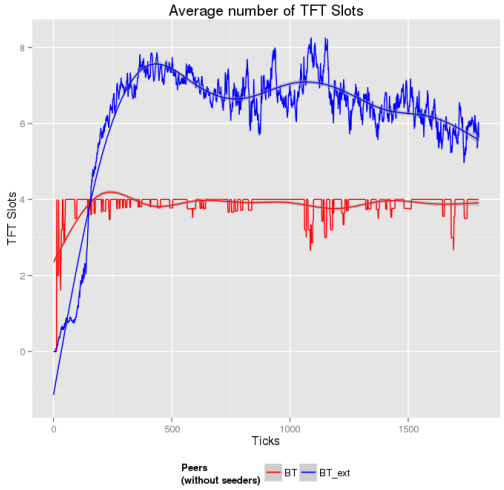


Figure D.18: dynamic,10p,90pc1,10MB

Figure D.19: **dynamic,10MB Scaled upload rate**

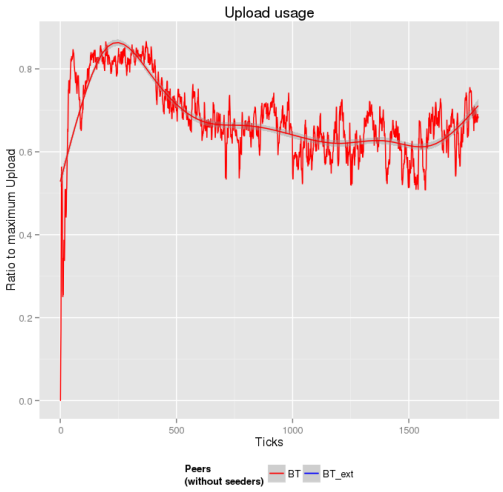


Figure D.20: dynamic,100p,0pc1,10MB

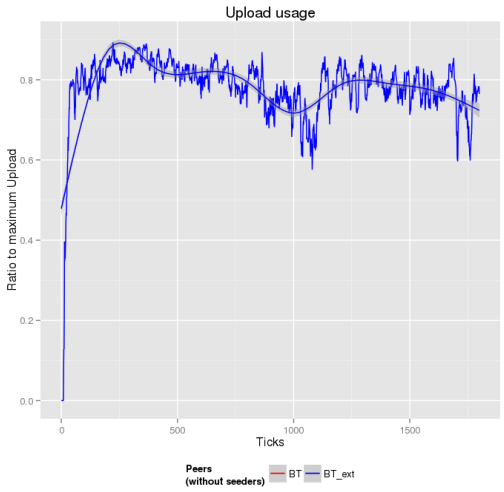


Figure D.21: dynamic,0p,100pc1,10MB

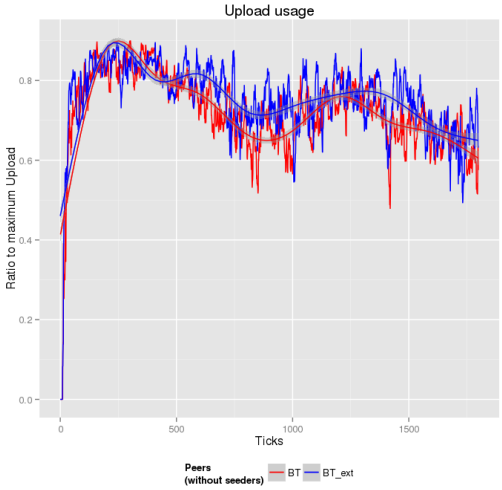


Figure D.22: dynamic,50p,50pc1,10MB

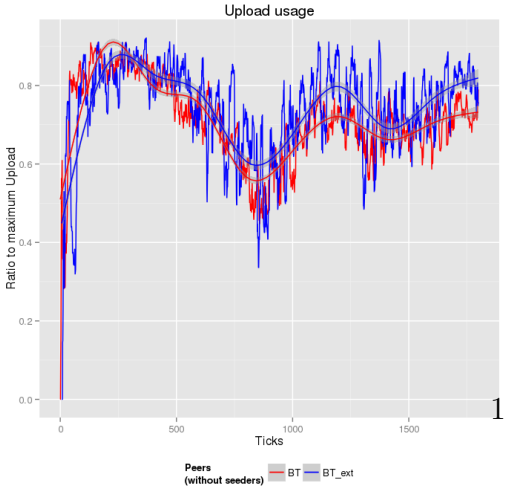


Figure D.23: dynamic,90p,10pc1,10MB

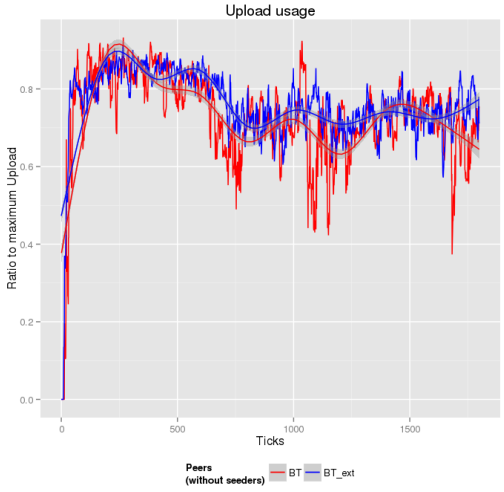


Figure D.24: dynamic,10p,90pc1,10MB

Figure D.25: **dynamic,10MB Scaled download rate**

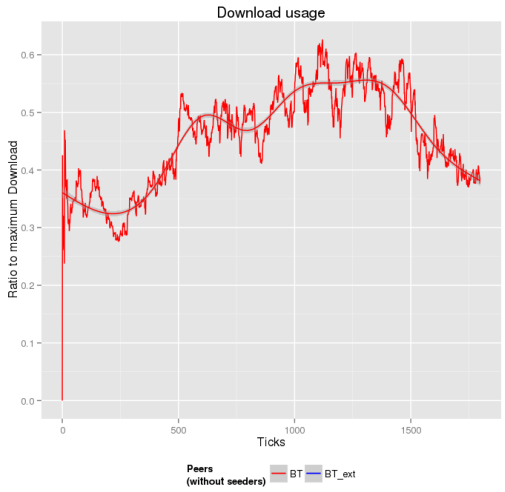


Figure D.26: dynamic,100p,0pc1,10MB

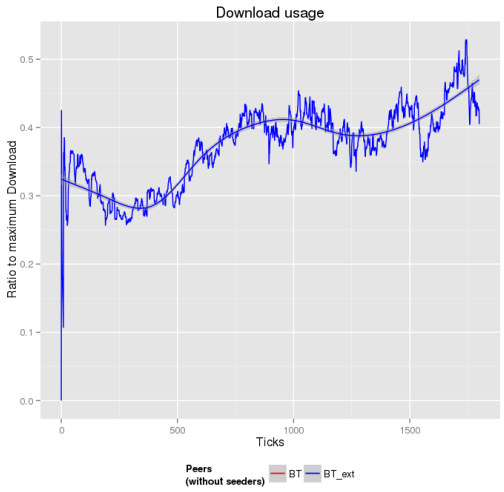


Figure D.27: dynamic,0p,100pc1,10MB

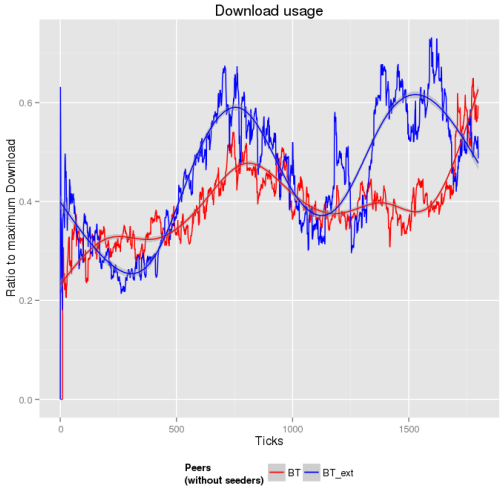


Figure D.28: dynamic,50p,50pc1,10MB

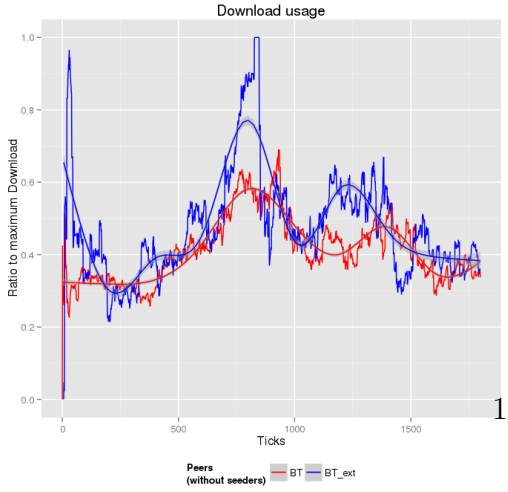


Figure D.29: dynamic,90p,10pc1,10MB

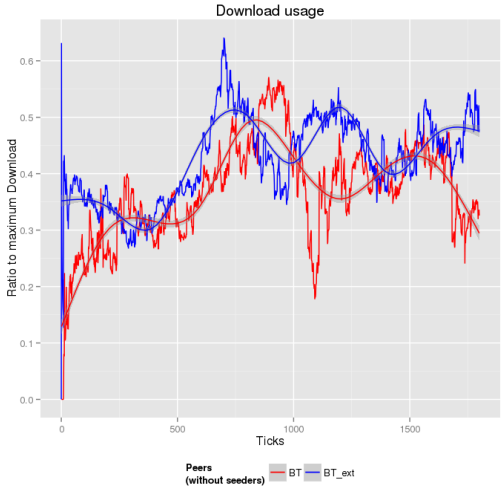


Figure D.30: dynamic,10p,90pc1,10MB

Figure D.31: **dynamic,350MB Peer Count**

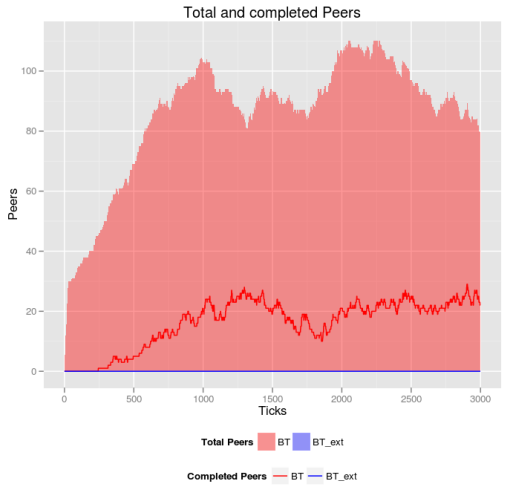


Figure D.32: dynamic,100p,0pc1,350MB

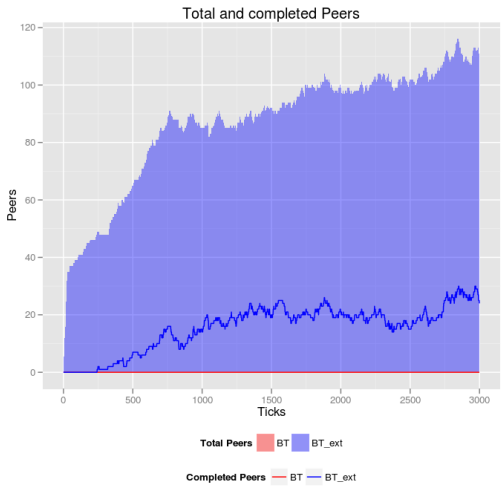


Figure D.33: dynamic,0p,100pc1,350MB

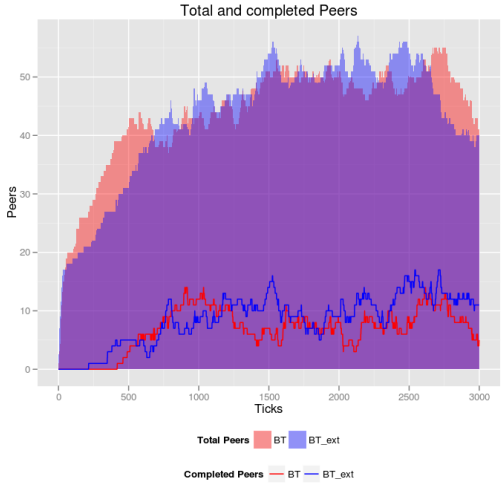


Figure D.34: dynamic,50p,50pc1,350MB

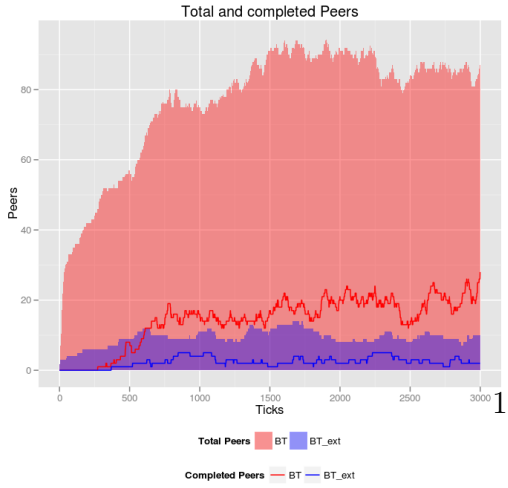


Figure D.35: dynamic,90p,10pc1,350MB

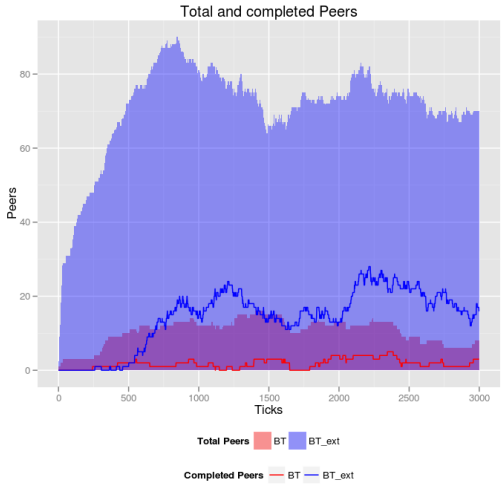


Figure D.36: dynamic,10p,90pc1,350MB

Figure D.37: **dynamic,350MB Average number of OU Slots**

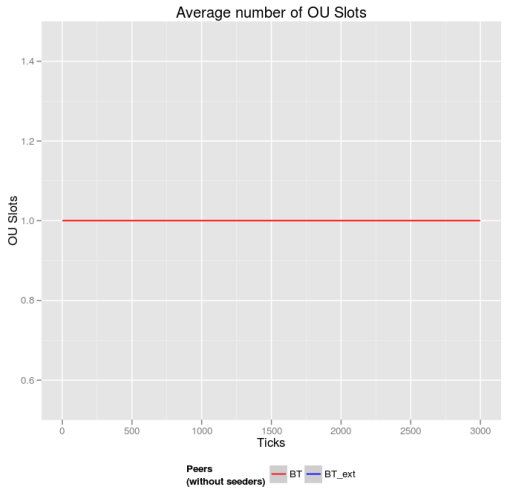


Figure D.38: dynamic,100p,0pc1,350MB

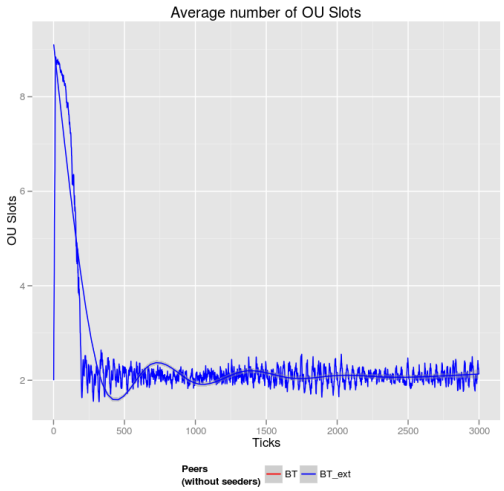


Figure D.39: dynamic,0p,100pc1,350MB

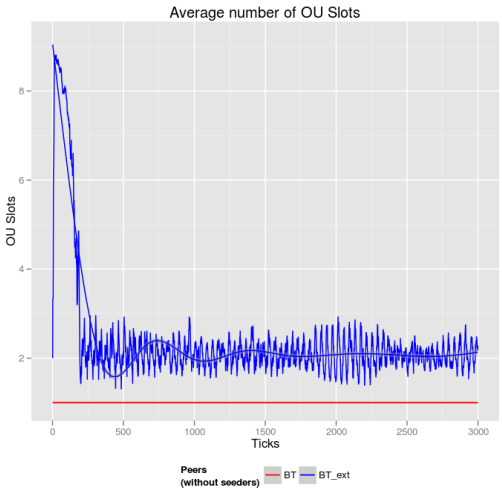


Figure D.40: dynamic,50p,50pc1,350MB

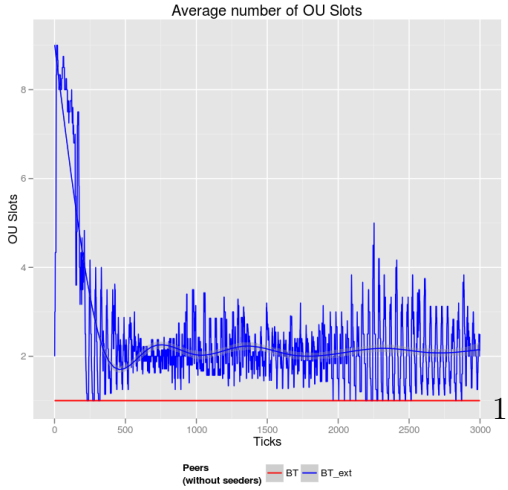


Figure D.41: dynamic,90p,10pc1,350MB

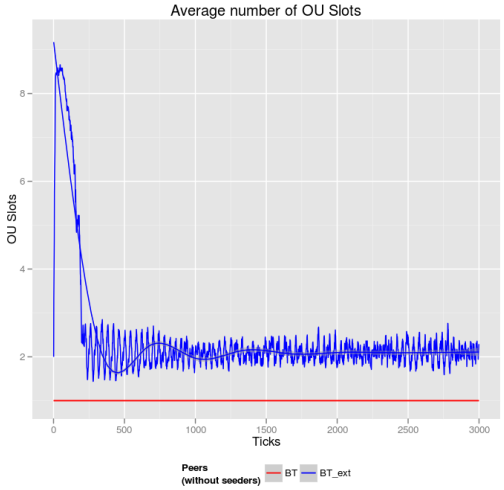


Figure D.42: dynamic,10p,90pc1,350MB

Figure D.43: **dynamic,350MB Average number of TFT Slots**

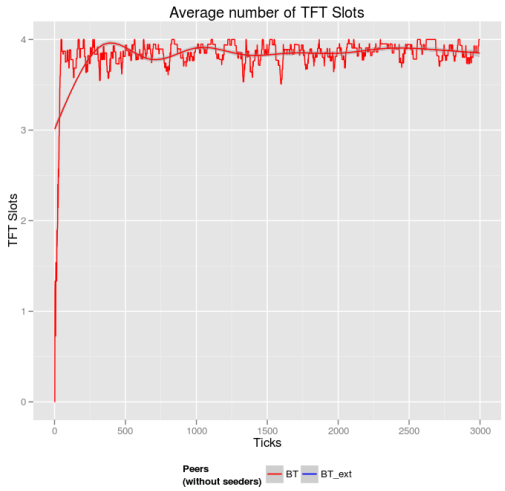


Figure D.44: dynamic,100p,0pc1,350MB

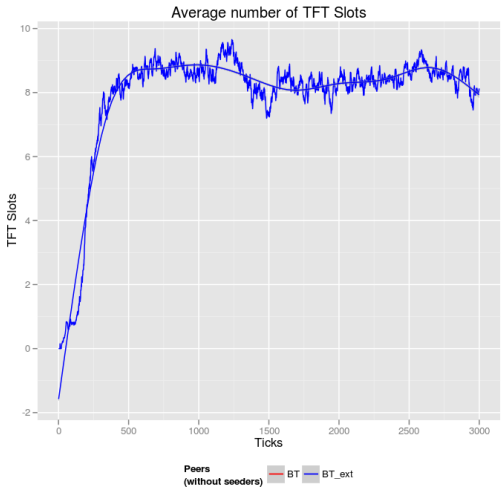


Figure D.45: dynamic,0p,100pc1,350MB

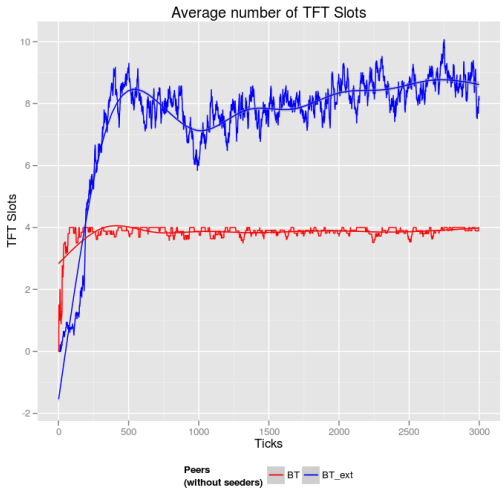


Figure D.46: dynamic,50p,50pc1,350MB

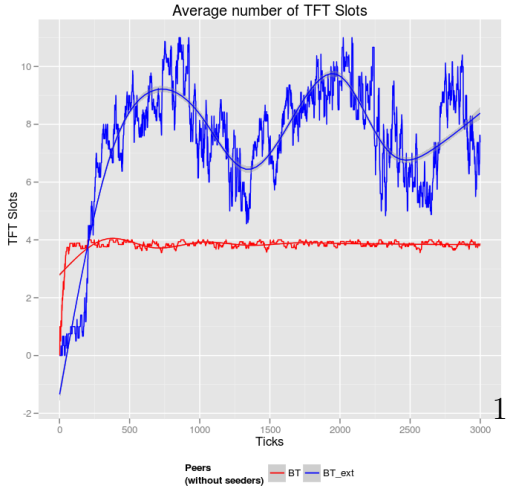


Figure D.47: dynamic,90p,10pc1,350MB

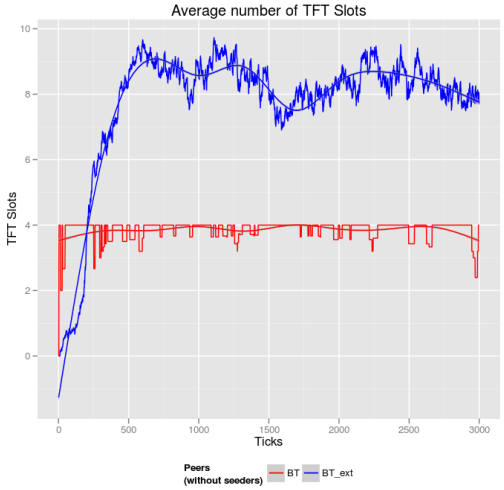


Figure D.48: dynamic,10p,90pc1,350MB

Figure D.49: **dynamic,350MB Scaled upload rate**

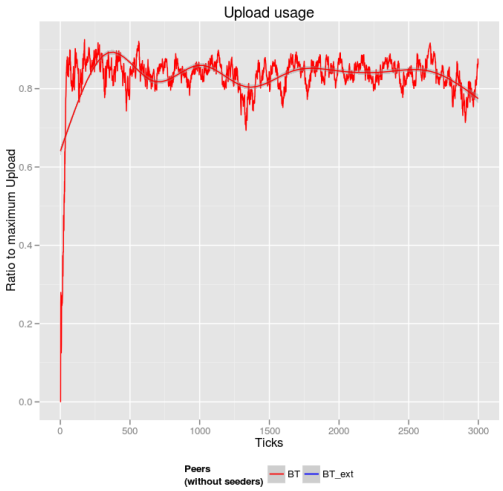


Figure D.50: dynamic,100p,0pc1,350MB

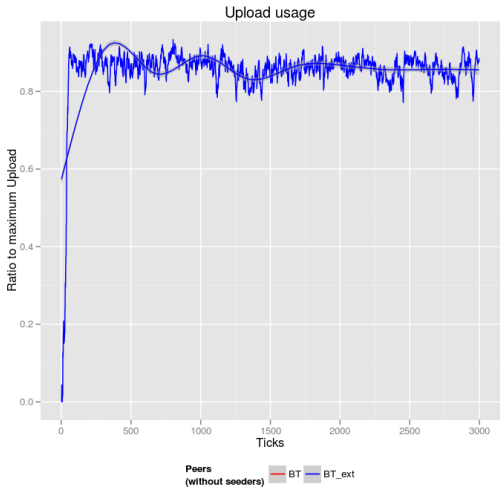


Figure D.51: dynamic,0p,100pc1,350MB

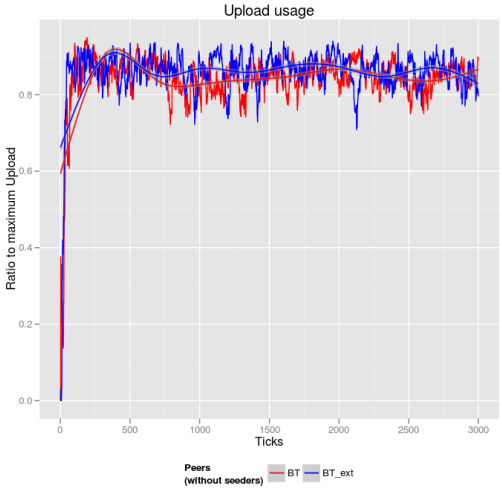


Figure D.52: dynamic,50p,50pc1,350MB

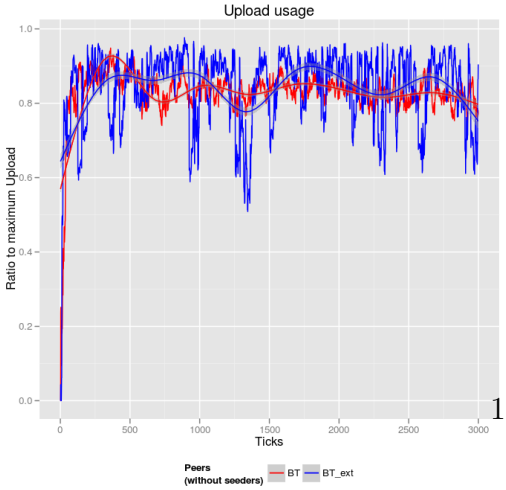


Figure D.53: dynamic,90p,10pc1,350MB

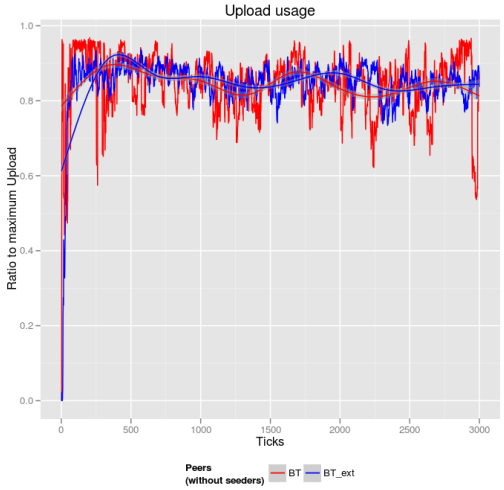


Figure D.54: dynamic,10p,90pc1,350MB

Figure D.55: **dynamic,350MB Scaled download rate**

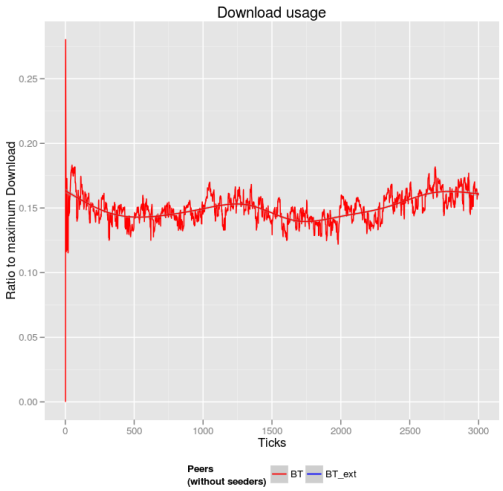


Figure D.56: dynamic,100p,0pc1,350MB

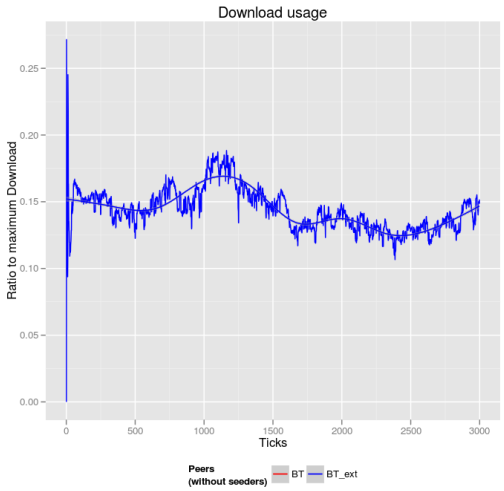


Figure D.57: dynamic,0p,100pc1,350MB

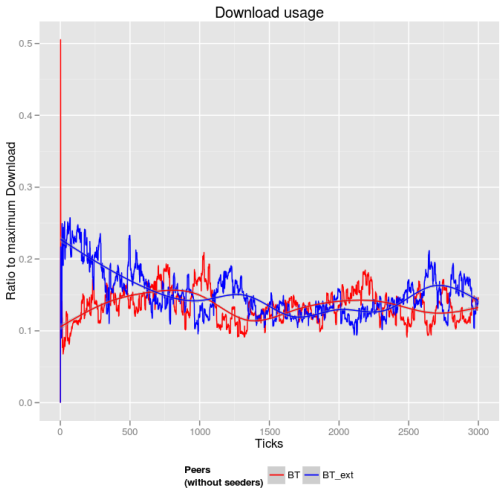


Figure D.58: dynamic,50p,50pc1,350MB

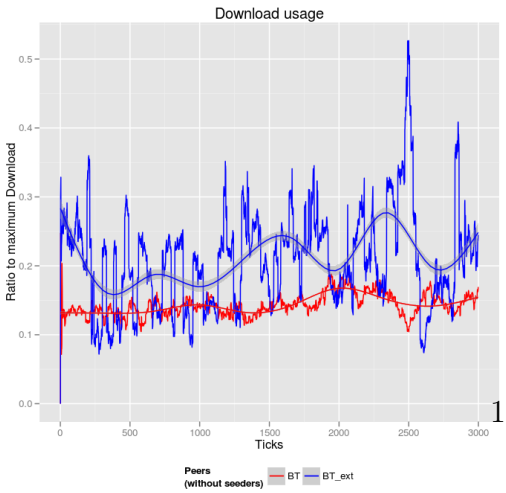


Figure D.59: dynamic,90p,10pc1,350MB

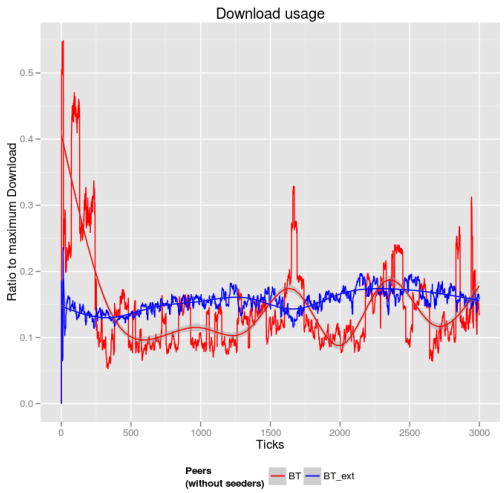


Figure D.60: dynamic,10p,90pc1,350MB

Figure D.61: **dynamic,1000MB Peer Count**

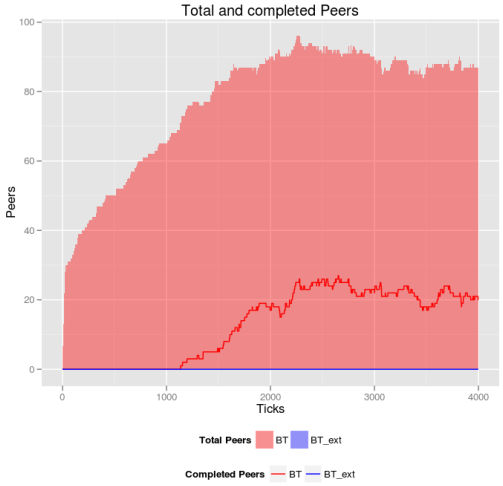


Figure D.62: dynamic,100p,0pc1,1000MB

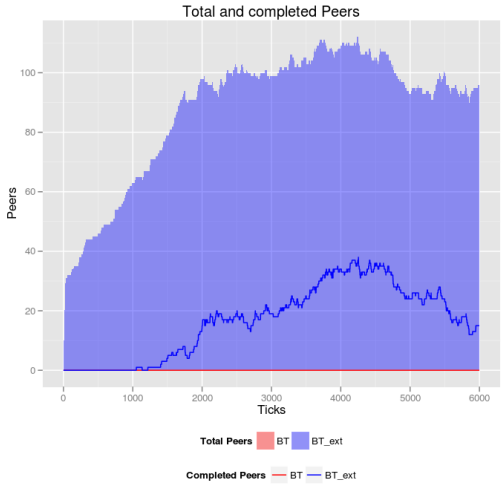


Figure D.63: dynamic,0p,100pc1,1000MB

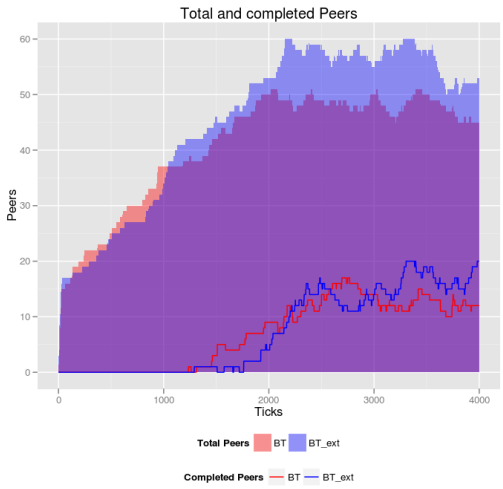


Figure D.64: dynamic,50p,50pc1,1000MB

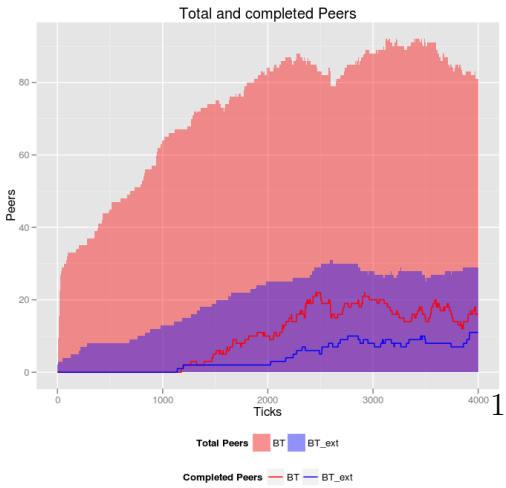


Figure D.65: dynamic,90p,10pc1,1000MB

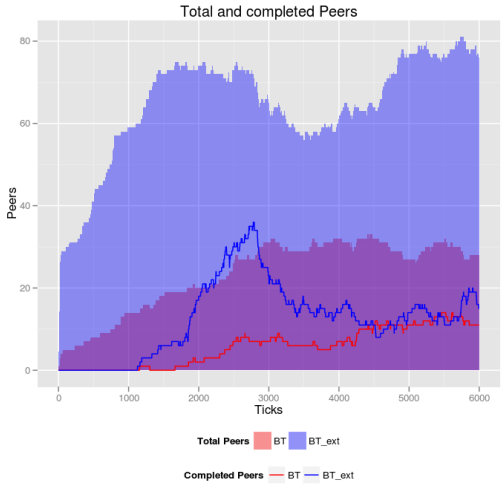


Figure D.66: dynamic,10p,90pc1,1000MB

Figure D.67: **dynamic,1000MB Average number of OU Slots**

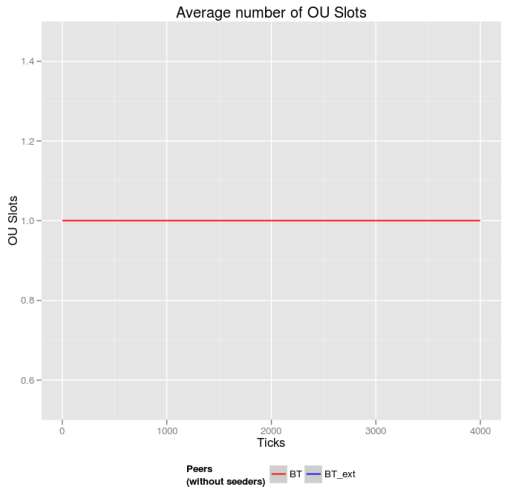


Figure D.68: dynamic,100p,0pc1,1000MB

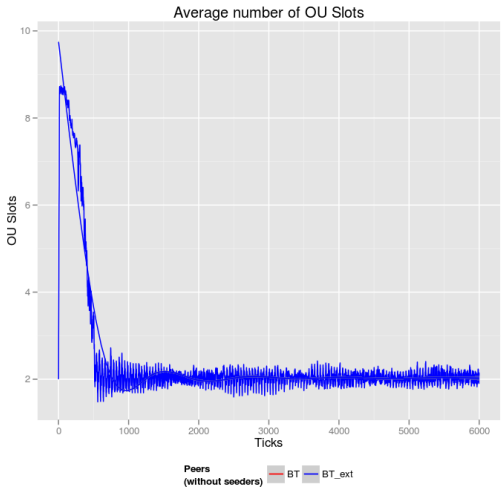


Figure D.69: dynamic,0p,100pc1,1000MB

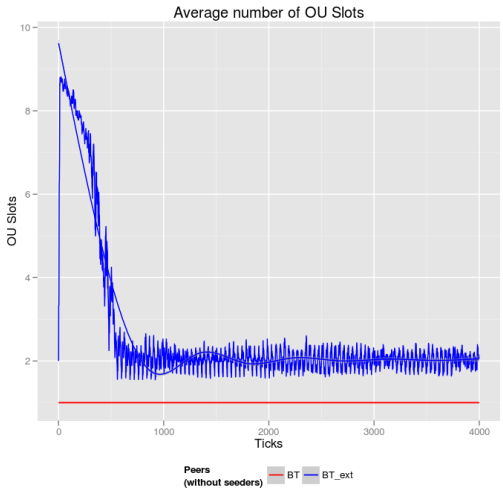


Figure D.70: dynamic,50p,50pc1,1000MB

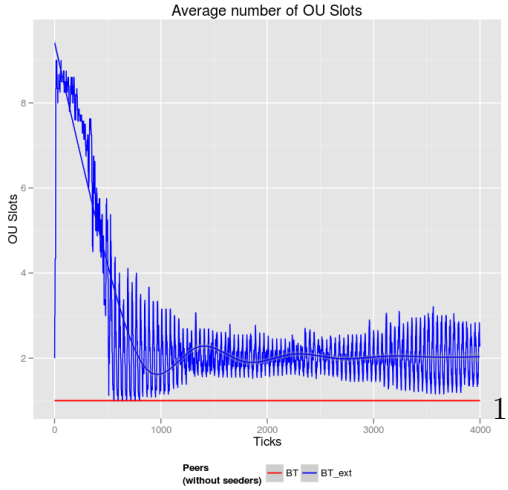


Figure D.71: dynamic,90p,10pc1,1000MB

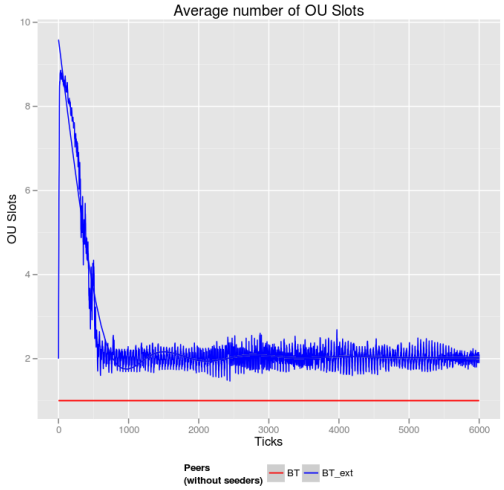


Figure D.72: dynamic,10p,90pc1,1000MB

Figure D.73: **dynamic,1000MB Average number of TFT Slots**

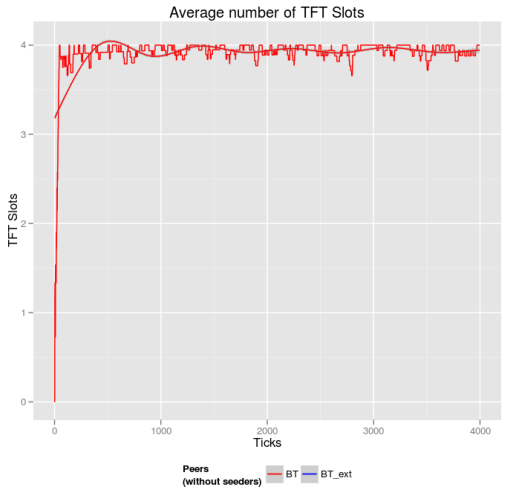


Figure D.74: dynamic,100p,0pc1,1000MB

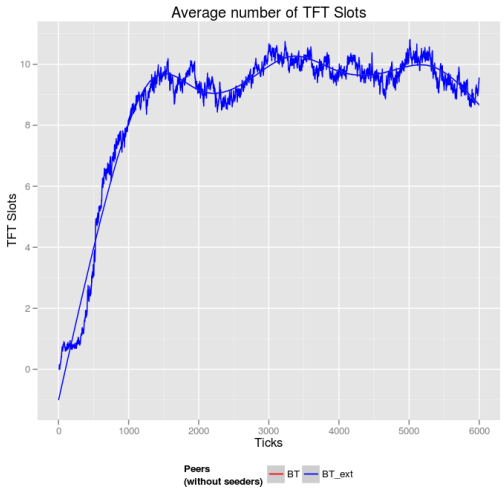


Figure D.75: dynamic,0p,100pc1,1000MB

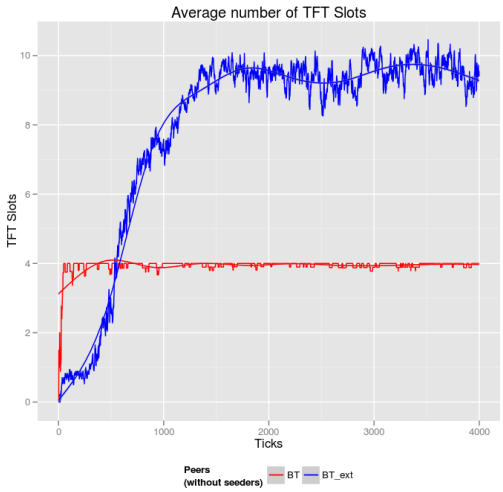


Figure D.76: dynamic,50p,50pc1,1000MB

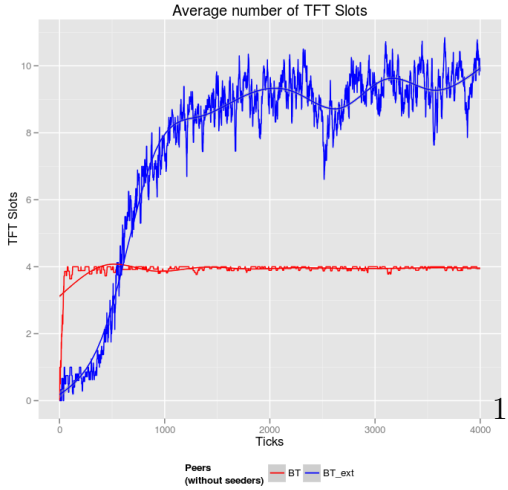


Figure D.77: dynamic,90p,10pc1,1000MB

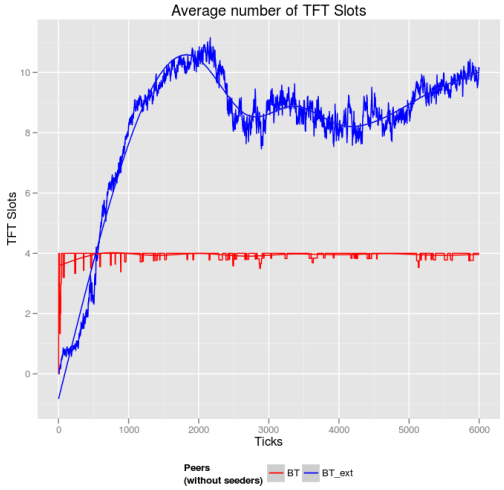


Figure D.78: dynamic,10p,90pc1,1000MB

Figure D.79: **dynamic,1000MB Scaled upload rate**

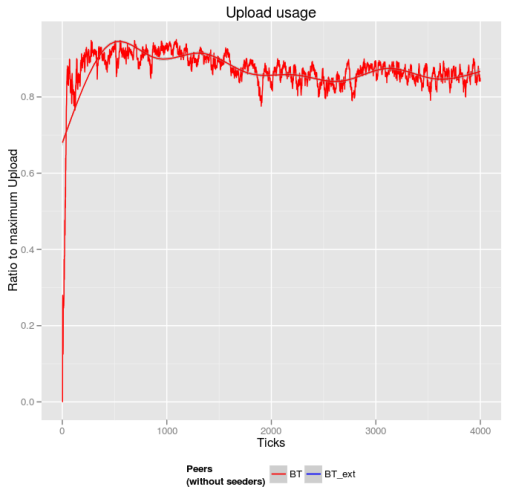


Figure D.80: dynamic,100p,0pc1,1000MB

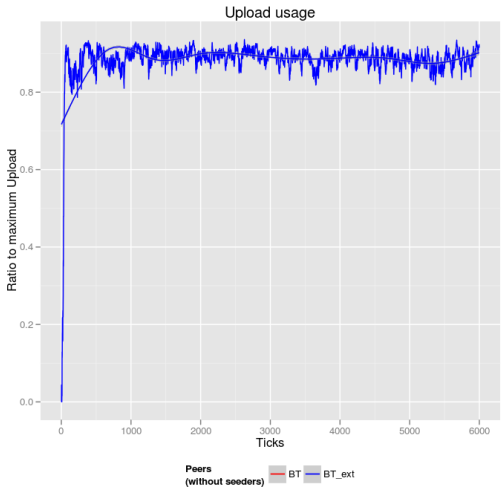


Figure D.81: dynamic,0p,100pc1,1000MB

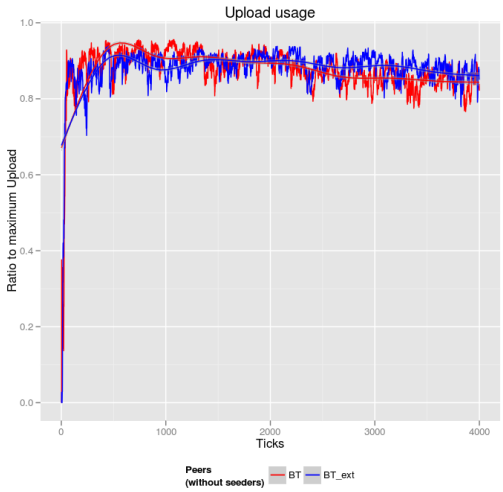


Figure D.82: dynamic,50p,50pc1,1000MB

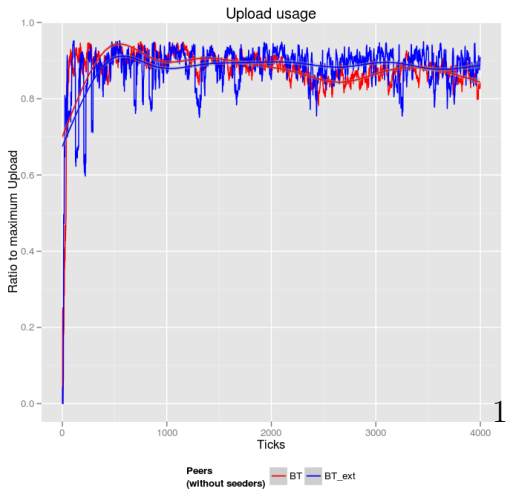


Figure D.83: dynamic,90p,10pc1,1000MB

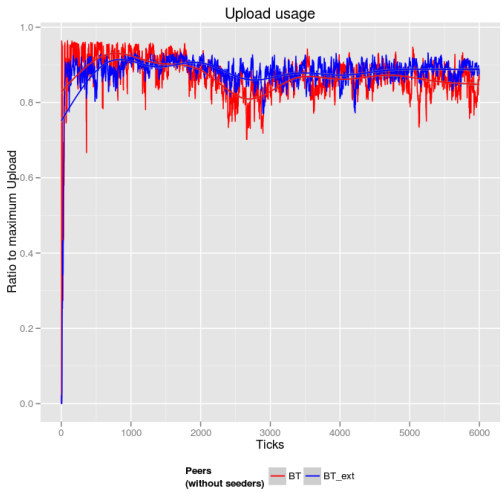


Figure D.84: dynamic,10p,90pc1,1000MB

Figure D.85: **dynamic,1000MB Scaled download rate**

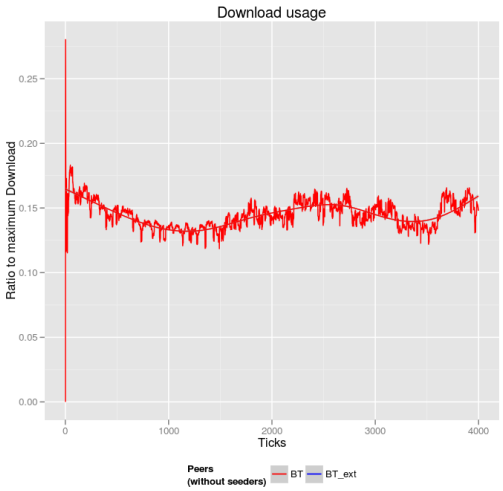


Figure D.86: dynamic,100p,0pc1,1000MB

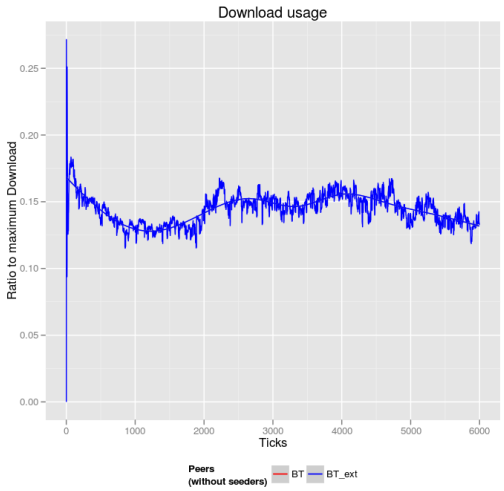


Figure D.87: dynamic,0p,100pc1,1000MB

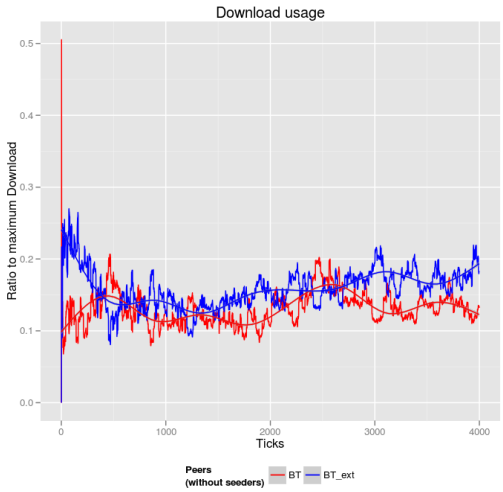


Figure D.88: dynamic,50p,50pc1,1000MB

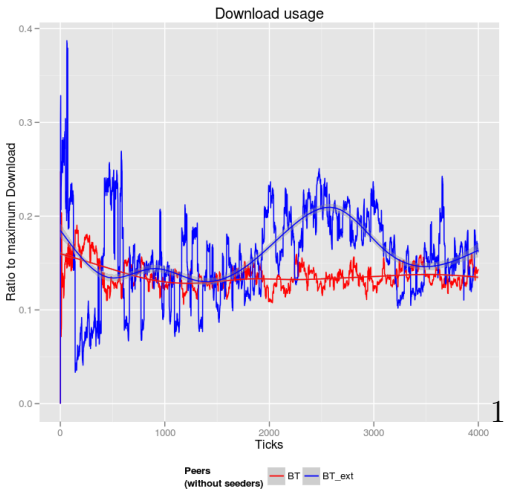


Figure D.89: dynamic,90p,10pc1,1000MB

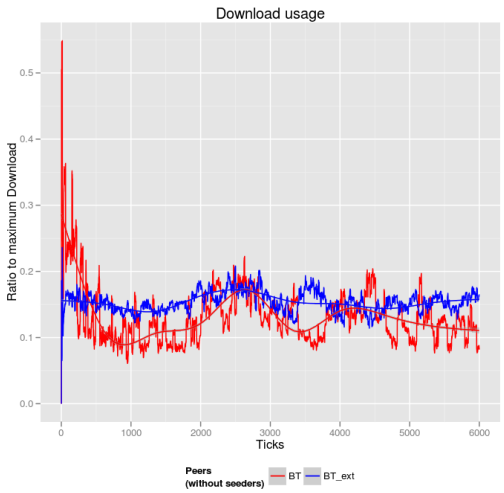


Figure D.90: dynamic,10p,90pc1,1000MB

Appendix E

E.1 Dynamic Public Tracker - Simulation Results

E.1.1 Dynamic private Tracker with small files

E.1.2 Dynamic private Tracker with medium files

E.1.3 Dynamic private Tracker with large files

Figure E.1: **dynamic,10MB Peer Count**

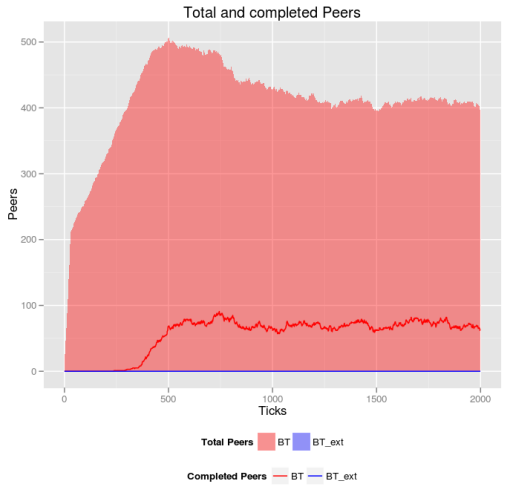


Figure E.2: dynamic,500p,0pc1,10MB

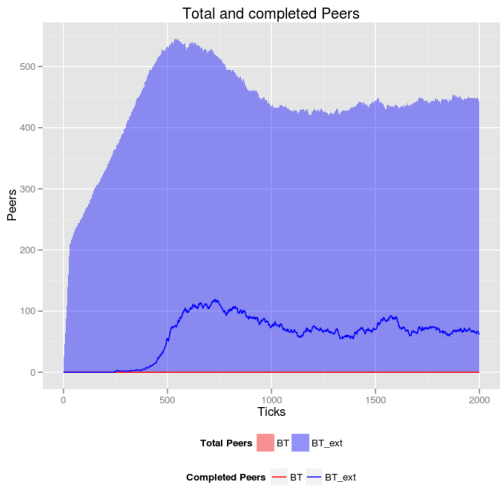


Figure E.3: dynamic,0p,500pc1,10MB

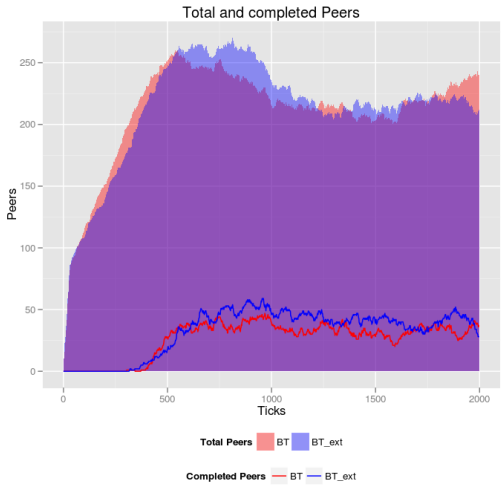


Figure E.4: dynamic,250p,250pc1,10MB

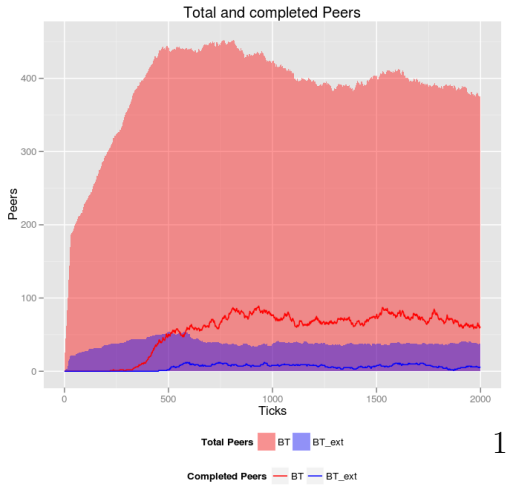


Figure E.5: dynamic,450p,50pc1,10MB

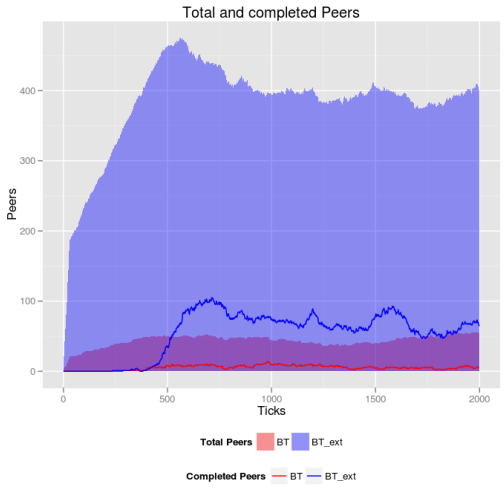


Figure E.6: dynamic,50p,450pc1,10MB

Figure E.7: **dynamic,10MB Average number of OU Slots**

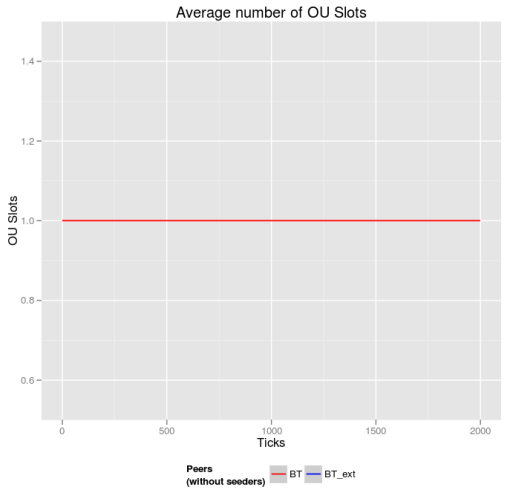


Figure E.8: dynamic,500p,0pc1,10MB

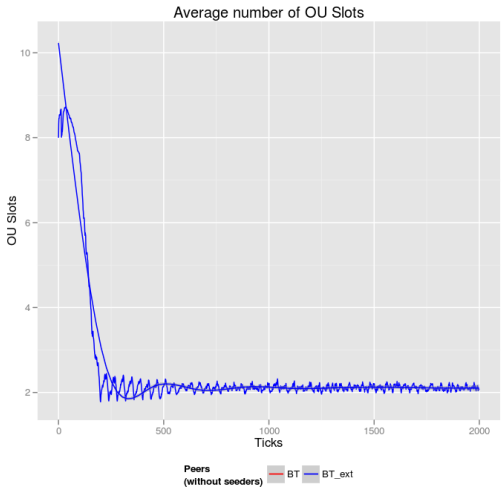


Figure E.9: dynamic,0p,500pc1,10MB

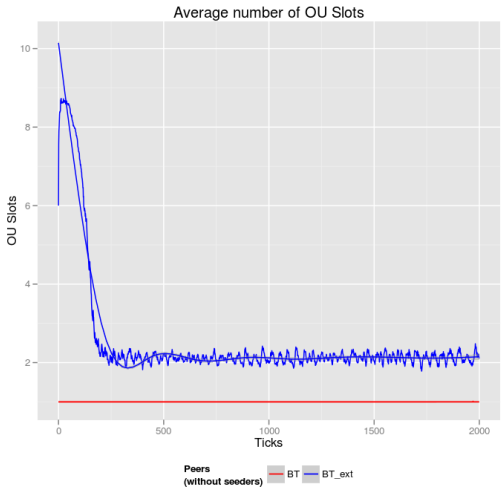


Figure E.10: dynamic,250p,250pc1,10MB

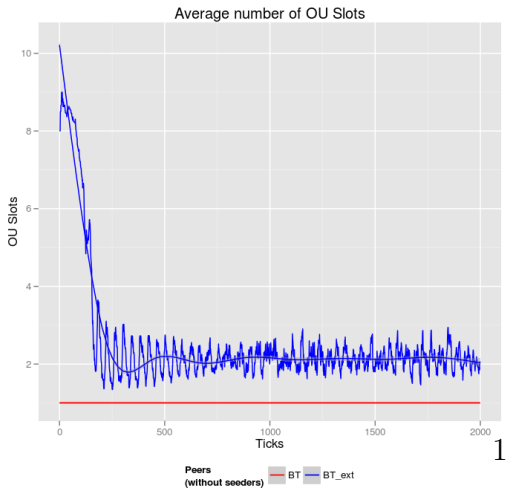


Figure E.11: dynamic,450p,50pc1,10MB

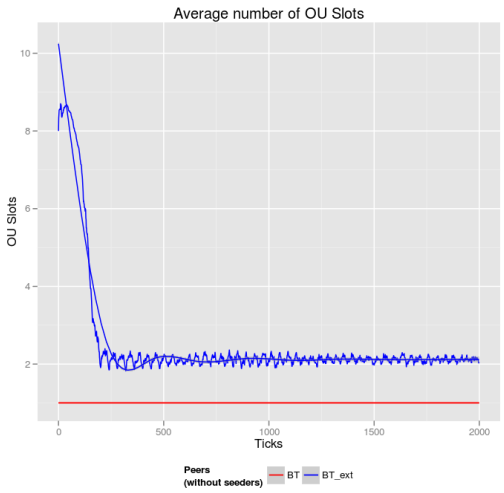


Figure E.12: dynamic,50p,450pc1,10MB

Figure E.13: **dynamic,10MB Average number of TFT Slots**

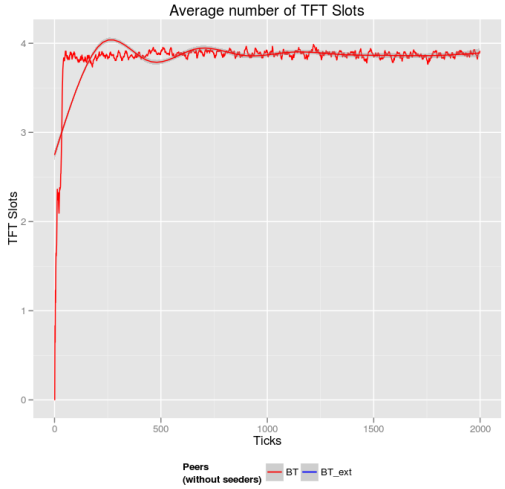


Figure E.14: dynamic,500p,0pc1,10MB

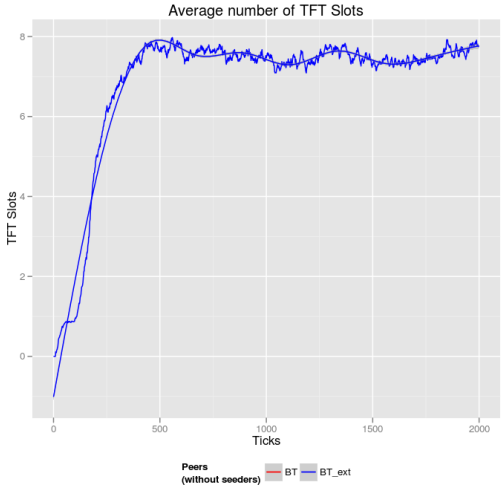


Figure E.15: dynamic,0p,500pc1,10MB

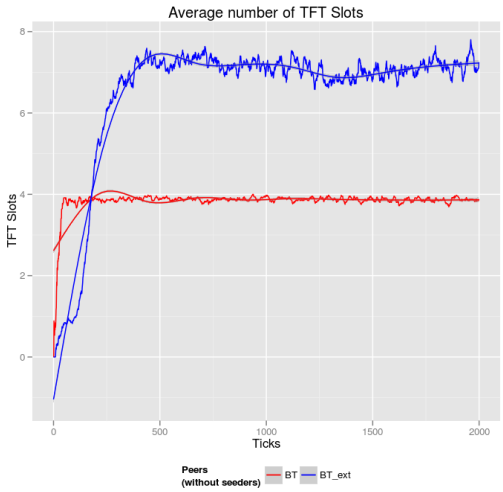


Figure E.16: dynamic,250p,250pc1,10MB

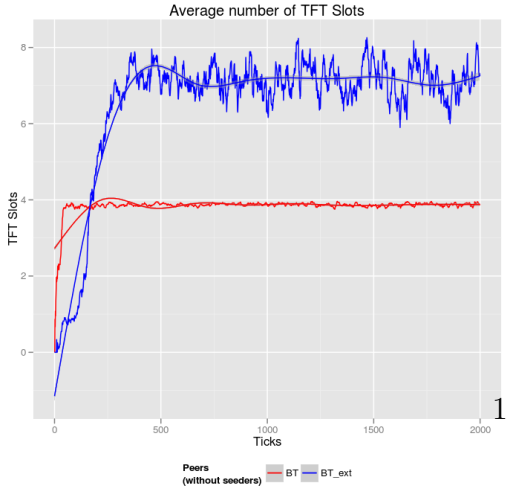


Figure E.17: dynamic,450p,50pc1,10MB

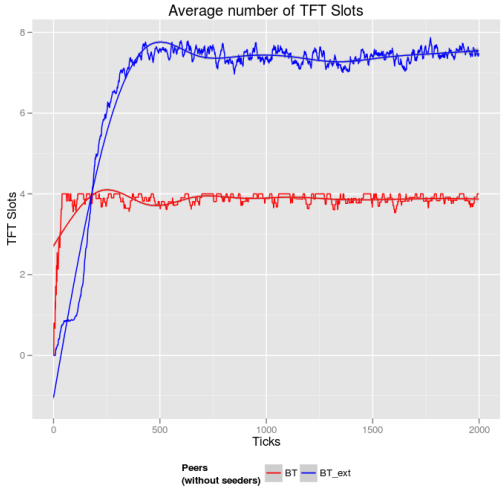


Figure E.18: dynamic,50p,450pc1,10MB

Figure E.19: **dynamic,10MB Scaled upload rate**

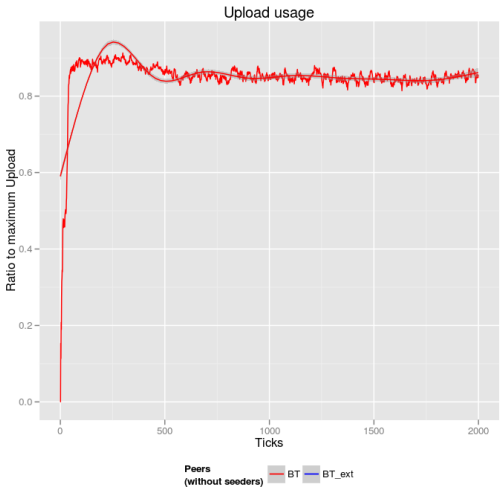


Figure E.20: dynamic,500p,0pc1,10MB

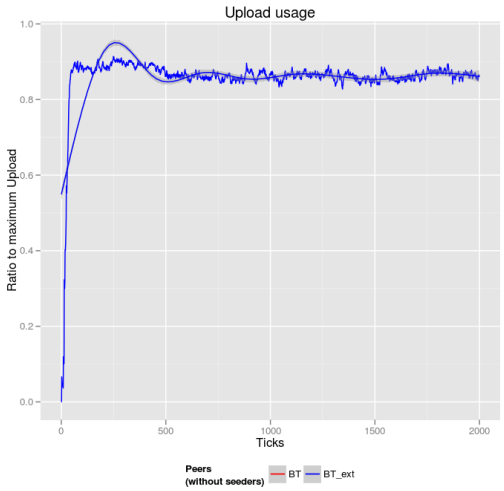


Figure E.21: dynamic,0p,500pc1,10MB

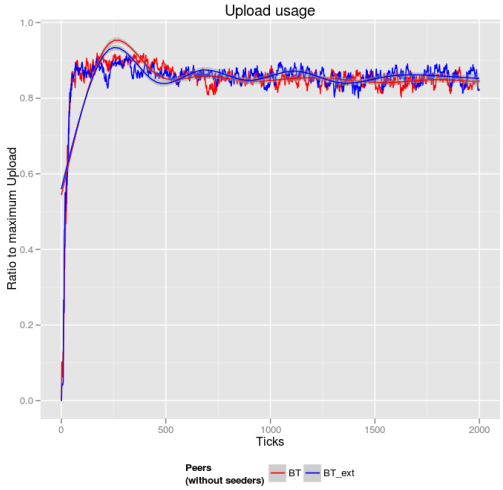


Figure E.22: dynamic,250p,250pc1,10MB

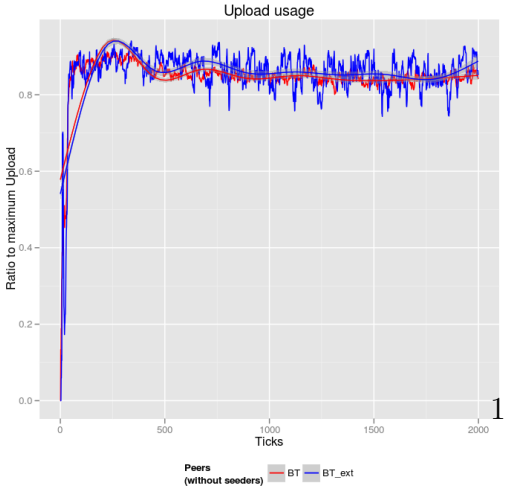


Figure E.23: dynamic,450p,50pc1,10MB

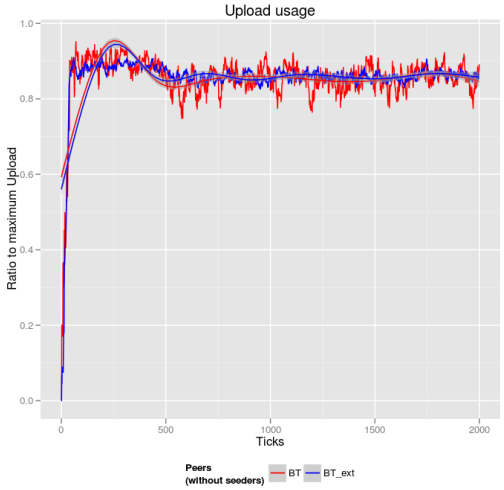


Figure E.24: dynamic,50p,450pc1,10MB

Figure E.25: **dynamic,10MB Scaled download rate**

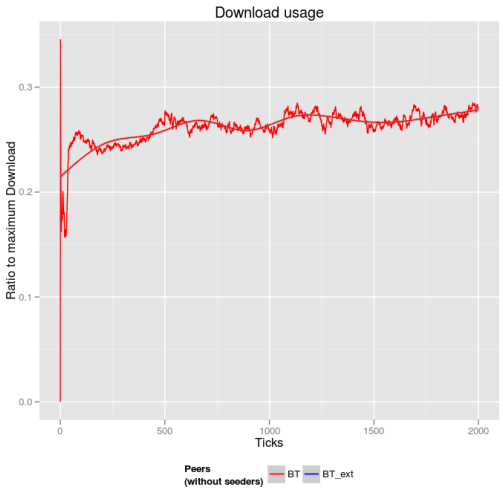


Figure E.26: dynamic,500p,0pc1,10MB

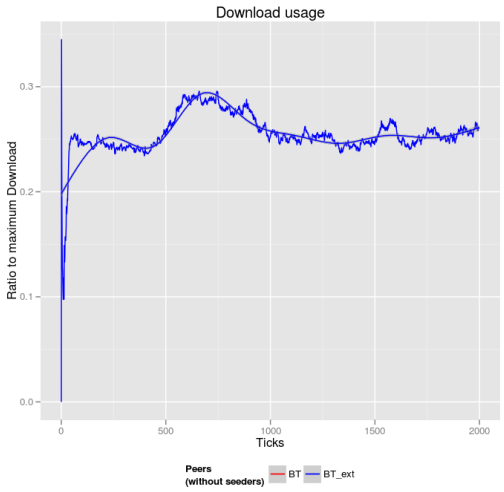


Figure E.27: dynamic,0p,500pc1,10MB

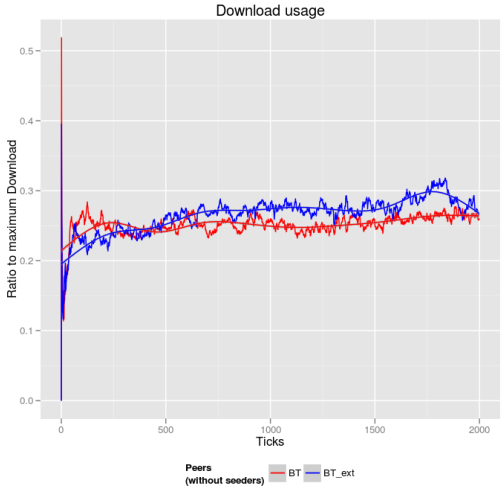


Figure E.28: dynamic,250p,250pc1,10MB

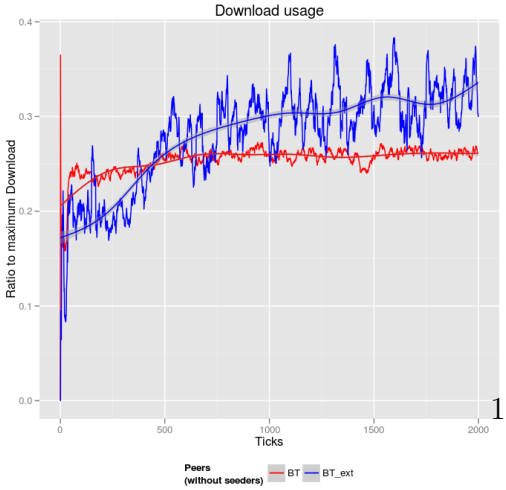


Figure E.29: dynamic,450p,50pc1,10MB

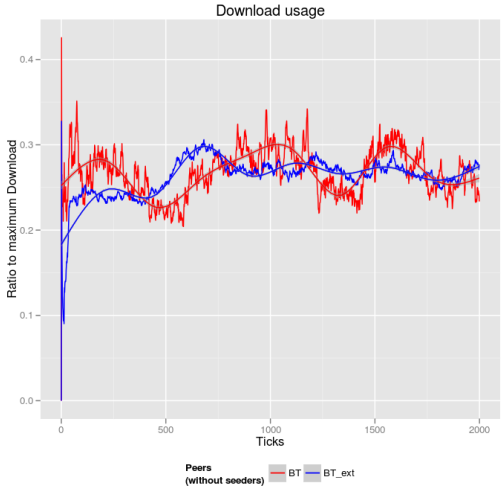


Figure E.30: dynamic,50p,450pc1,10MB

Figure E.31: **dynamic,350MB Peer Count**

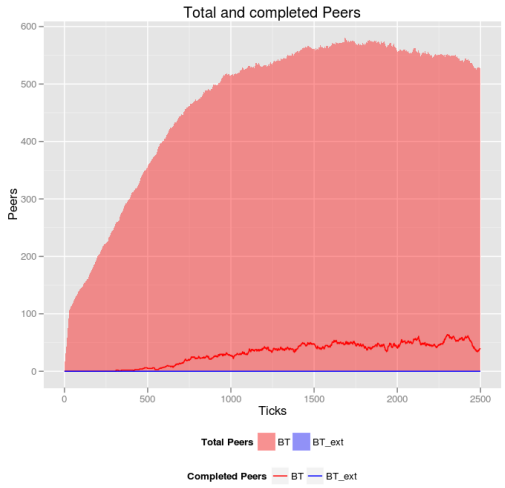


Figure E.32: dynamic,500p,0pc1,350MB

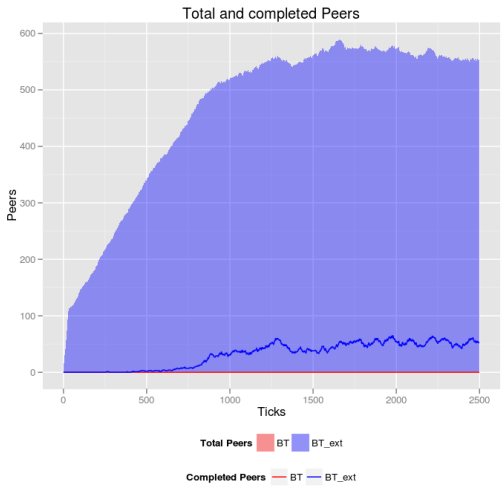


Figure E.33: dynamic,0p,500pc1,350MB

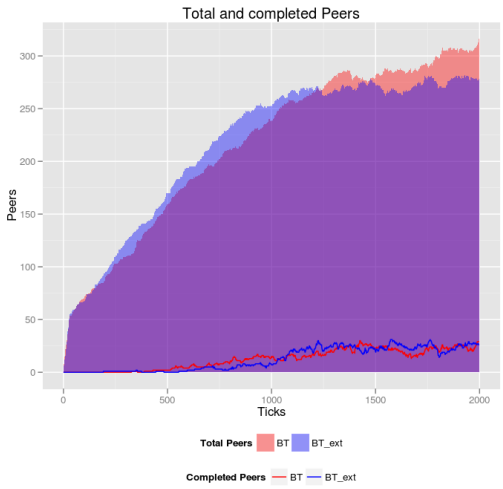


Figure E.34: dynamic,250p,250pc1,350MB

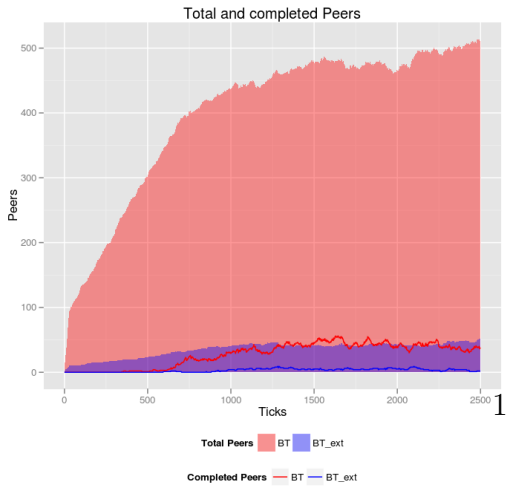


Figure E.35: dynamic,450p,50pc1,350MB

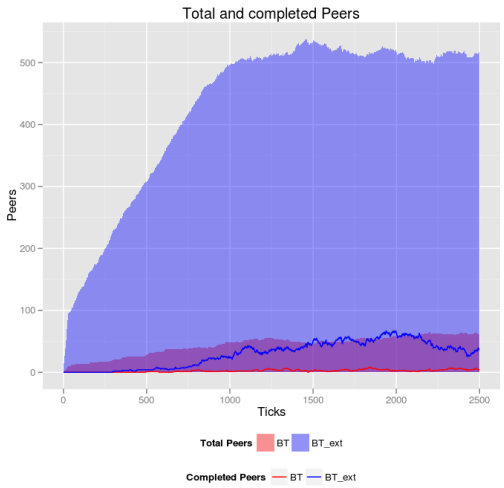


Figure E.36: dynamic,50p,450pc1,350MB

Figure E.37: **dynamic,350MB Average number of OU Slots**

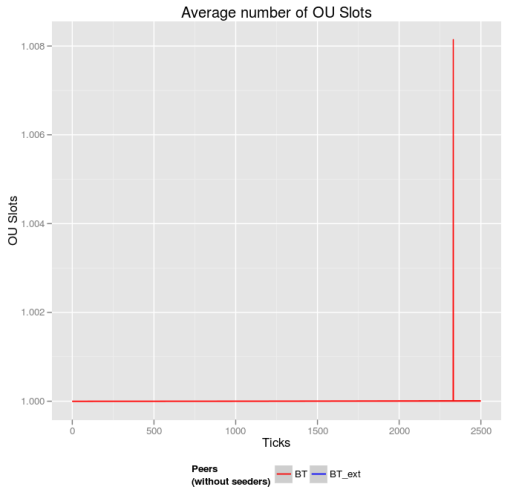


Figure E.38: dynamic,500p,0pc1,350MB

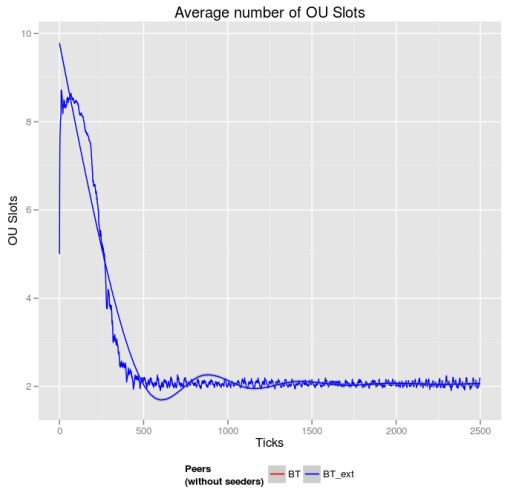


Figure E.39: dynamic,0p,500pc1,350MB

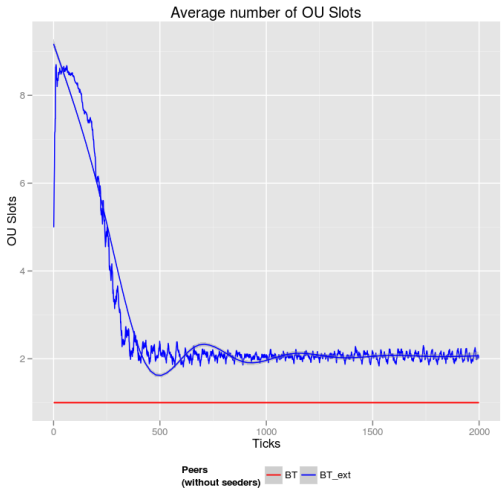


Figure E.40: dynamic,250p,250pc1,350MB

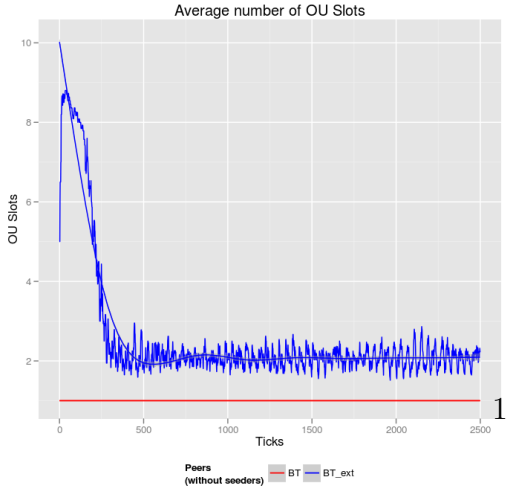


Figure E.41: dynamic,450p,50pc1,350MB

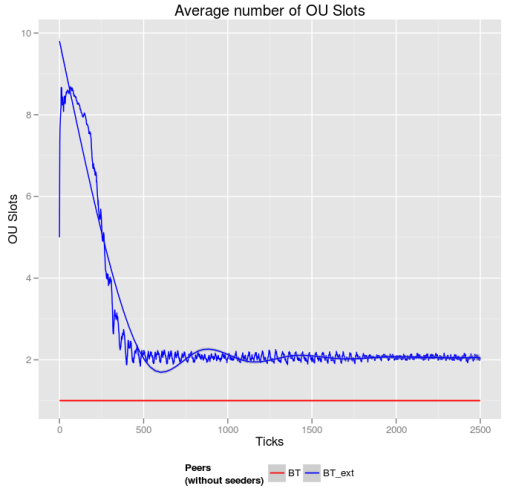


Figure E.42: dynamic,50p,450pc1,350MB

Figure E.43: **dynamic,350MB Average number of TFT Slots**

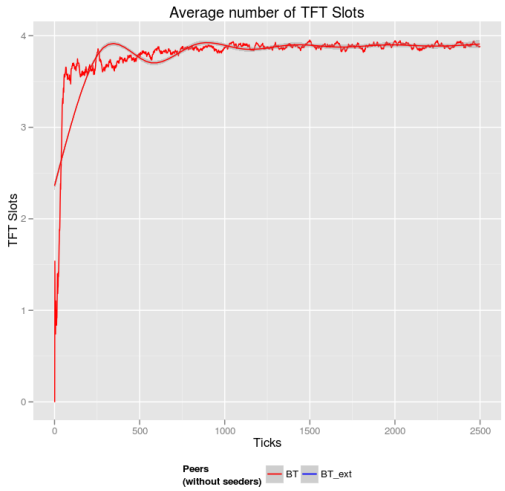


Figure E.44: dynamic,500p,0pc1,350MB

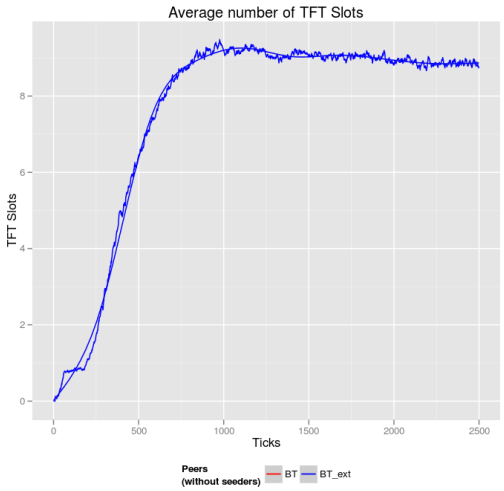


Figure E.45: dynamic,0p,500pc1,350MB

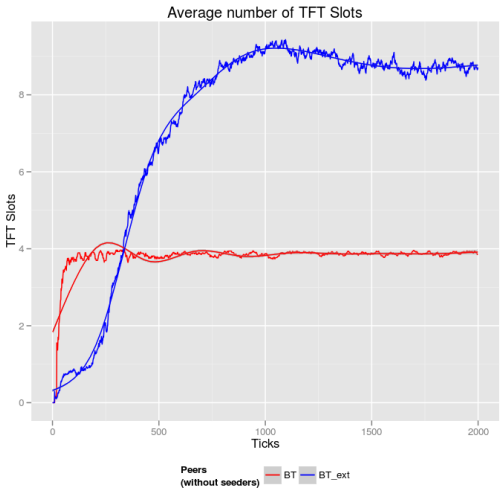


Figure E.46: dynamic,250p,250pc1,350MB

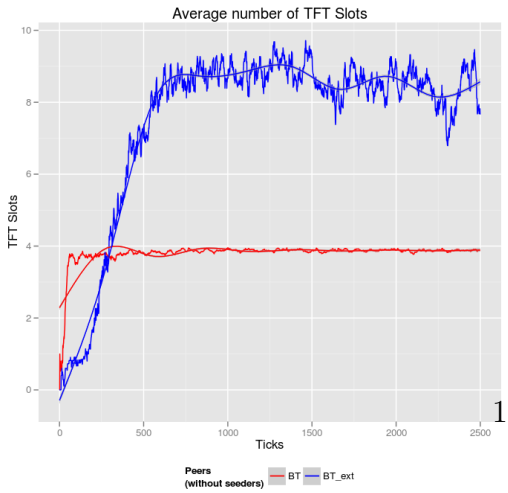


Figure E.47: dynamic,450p,50pc1,350MB

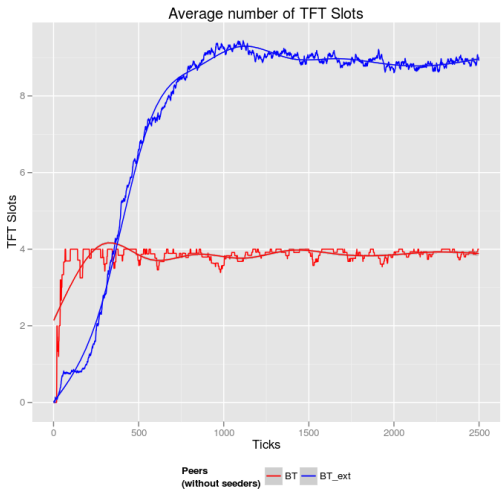


Figure E.48: dynamic,50p,450pc1,350MB

Figure E.49: **dynamic,350MB Scaled upload rate**

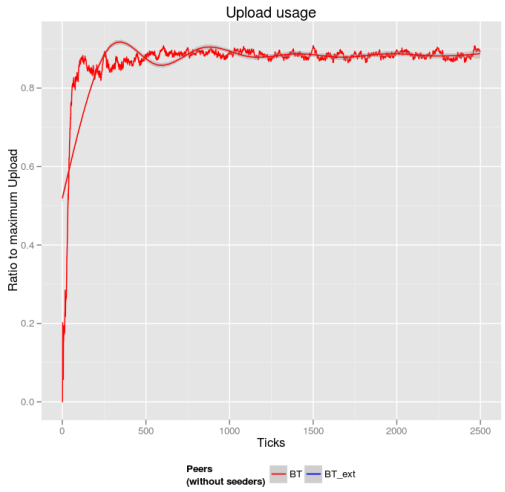


Figure E.50: dynamic,500p,0pc1,350MB

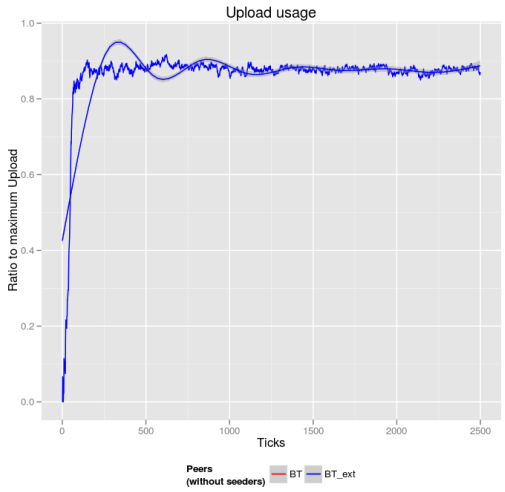


Figure E.51: dynamic,0p,500pc1,350MB

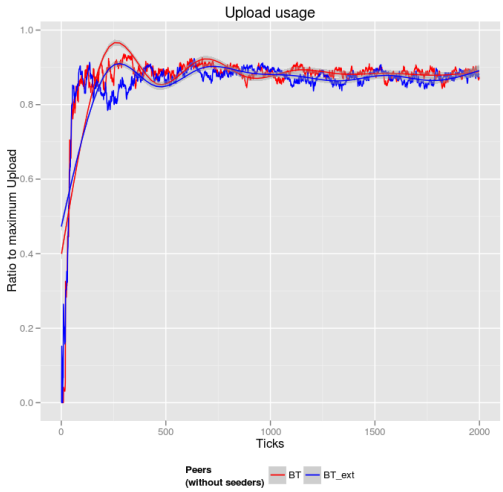


Figure E.52: dynamic,250p,250pc1,350MB

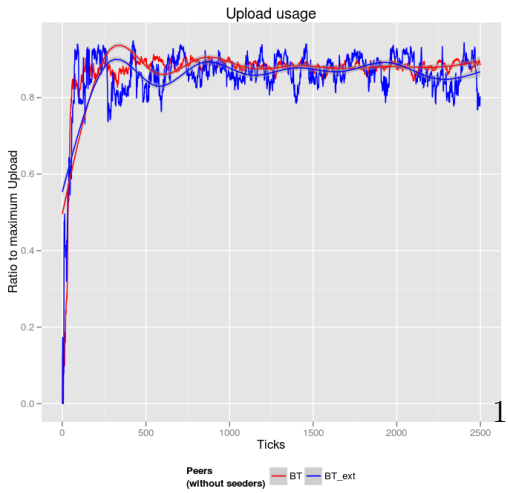


Figure E.53: dynamic,450p,50pc1,350MB

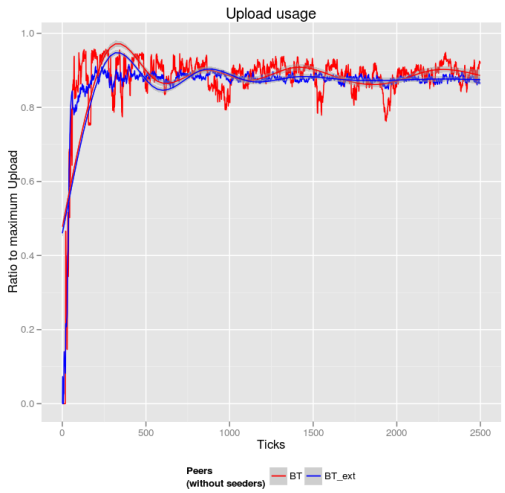


Figure E.54: dynamic,50p,450pc1,350MB

Figure E.55: **dynamic,350MB Scaled download rate**

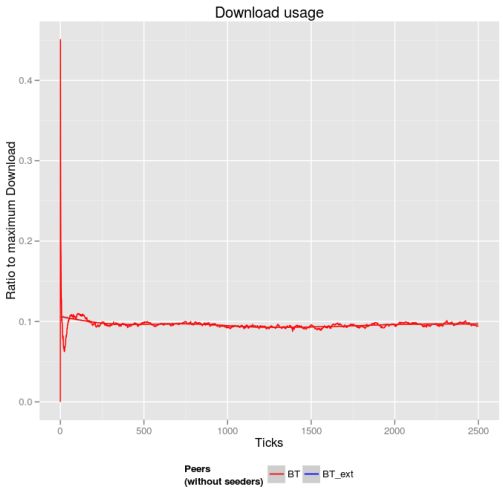


Figure E.56: dynamic,500p,0pc1,350MB

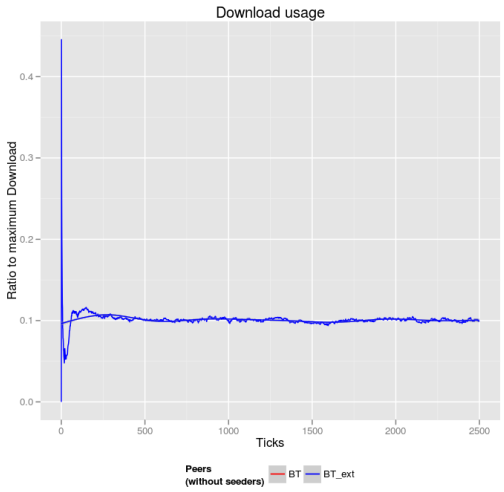


Figure E.57: dynamic,0p,500pc1,350MB

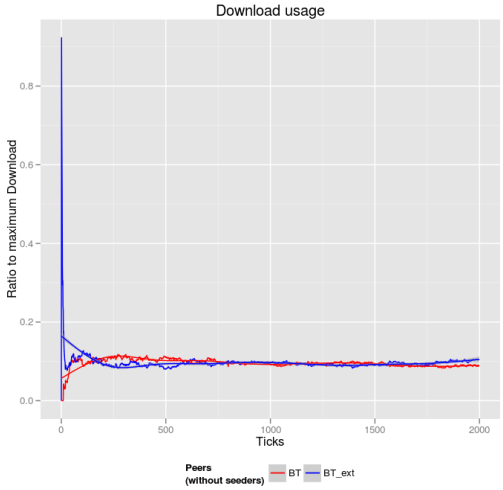


Figure E.58: dynamic,250p,250pc1,350MB

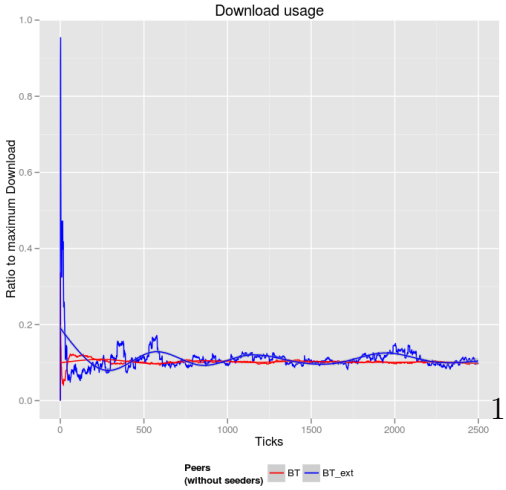


Figure E.59: dynamic,450p,50pc1,350MB

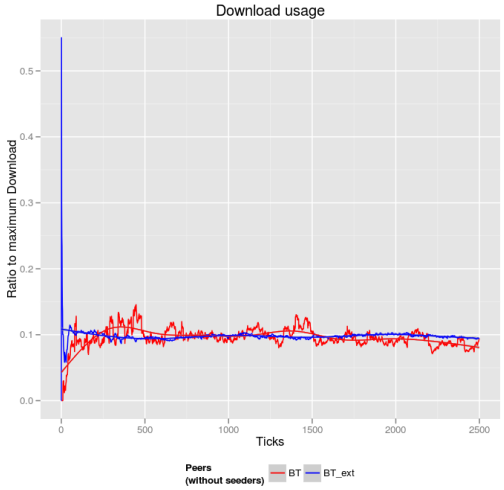


Figure E.60: dynamic,50p,450pc1,350MB

Figure E.61: **dynamic,1000MB Peer Count**

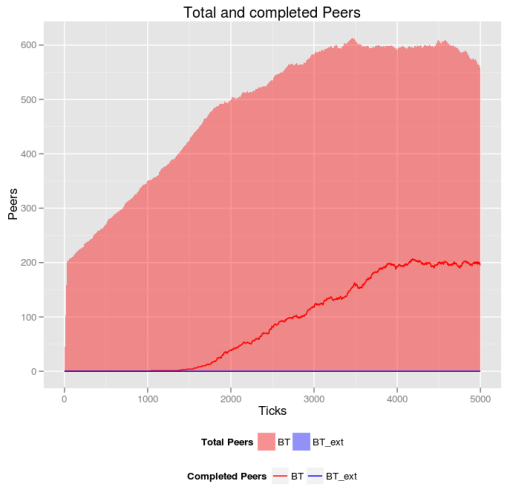


Figure E.62: dynamic,500p,0pc1,1000MB

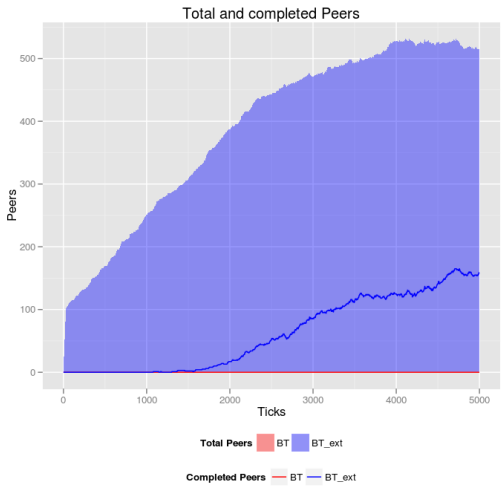


Figure E.63: dynamic,0p,500pc1,1000MB

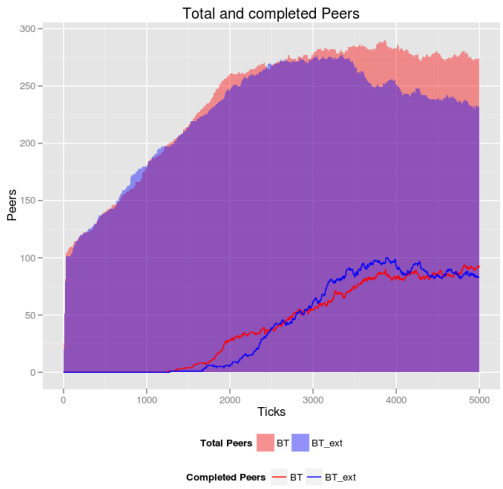


Figure E.64: dynamic,250p,250pc1,1000MB

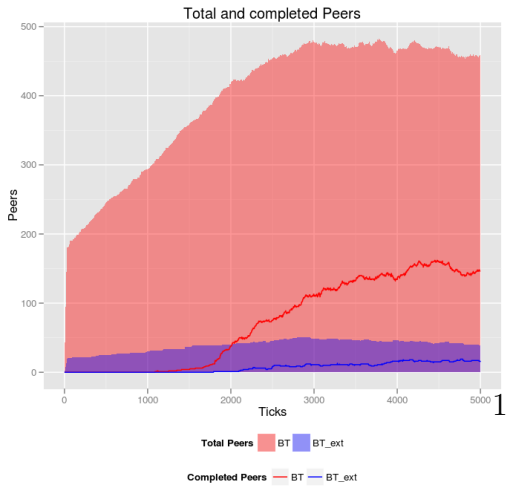


Figure E.65: dynamic,450p,50pc1,1000MB

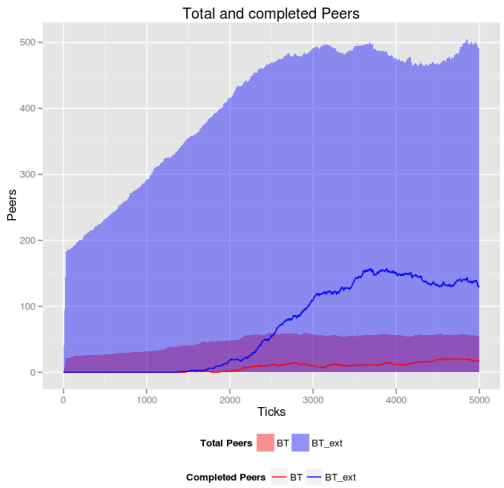


Figure E.66: dynamic,50p,450pc1,1000MB

Figure E.67: **dynamic,1000MB Average number of OU Slots**

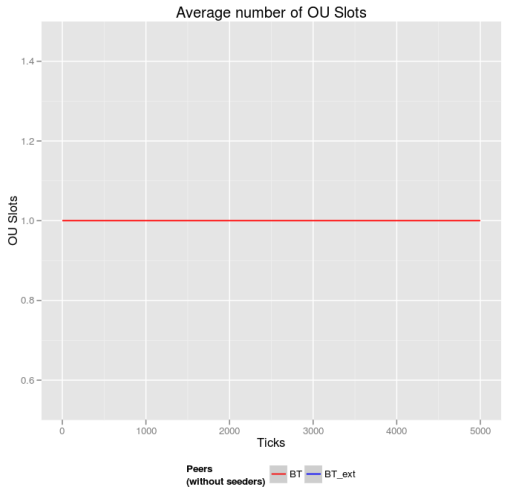


Figure E.68: dynamic,500p,0pc1,1000MB

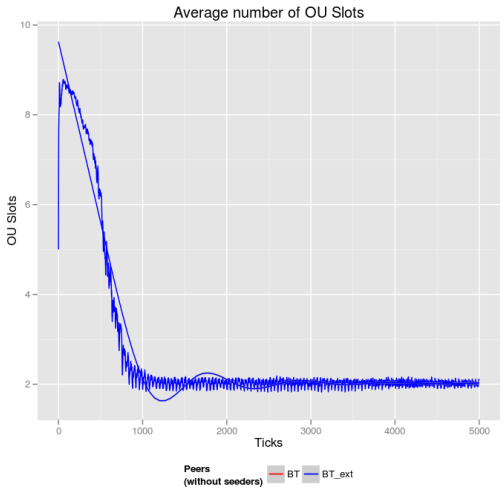


Figure E.69: dynamic,0p,500pc1,1000MB

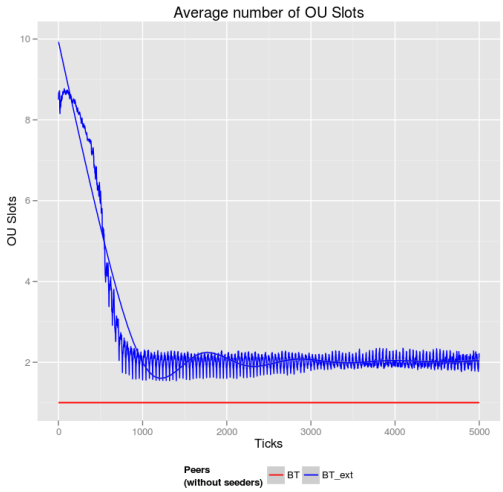


Figure E.70: dynamic,250p,250pc1,1000MB

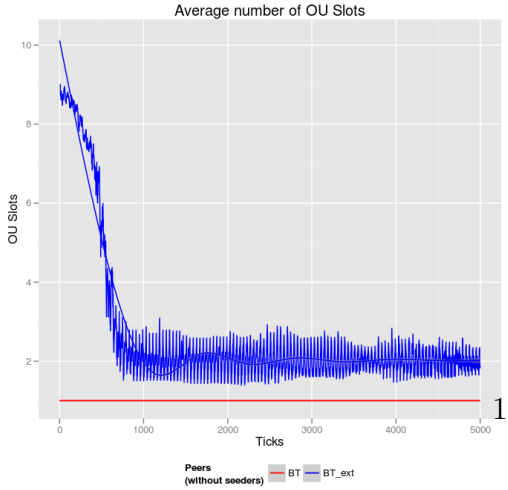


Figure E.71: dynamic,450p,50pc1,1000MB

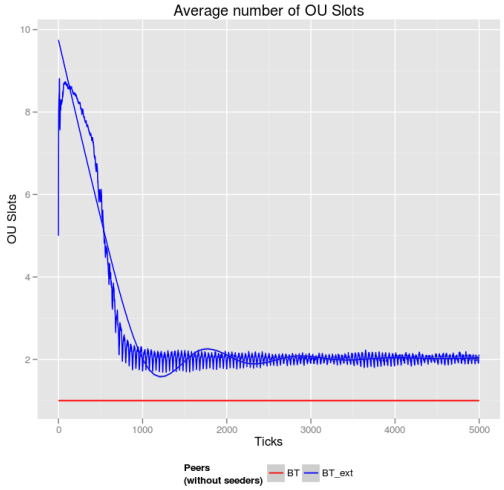


Figure E.72: dynamic,50p,450pc1,1000MB

Figure E.73: **dynamic,1000MB Average number of TFT Slots**

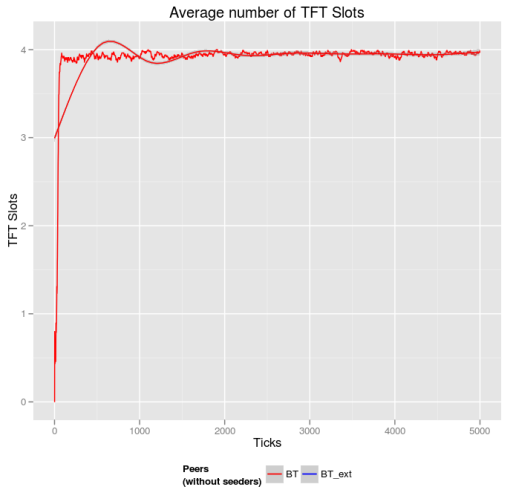


Figure E.74: dynamic,500p,0pc1,1000MB

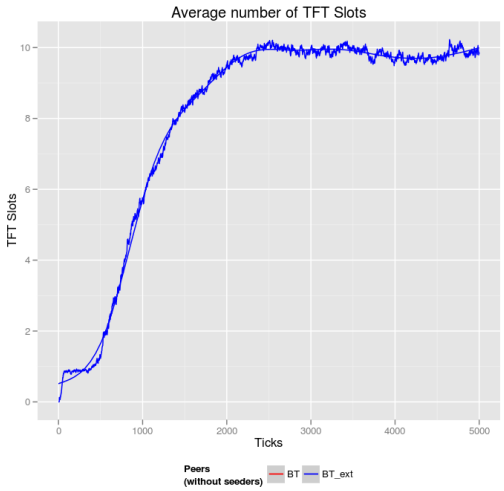


Figure E.75: dynamic,0p,500pc1,1000MB

Figure E.76: dynamic,250p,250pc1,1000MB

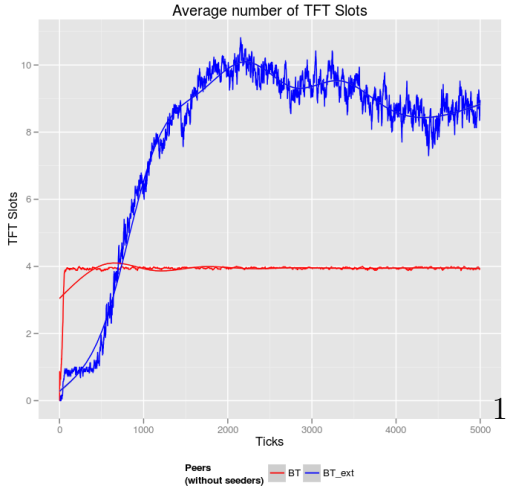


Figure E.77: dynamic,450p,50pc1,1000MB

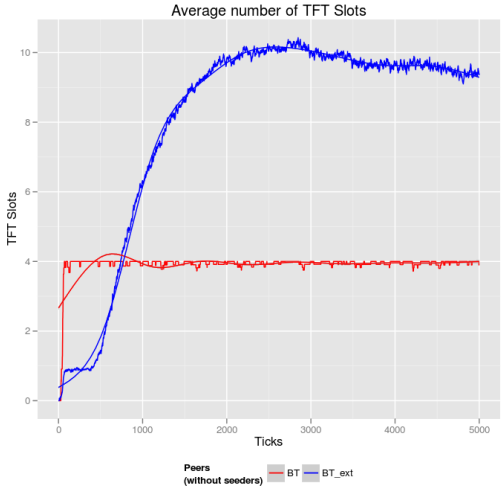


Figure E.78: dynamic,50p,450pc1,1000MB

Figure E.79: **dynamic,1000MB Scaled upload rate**

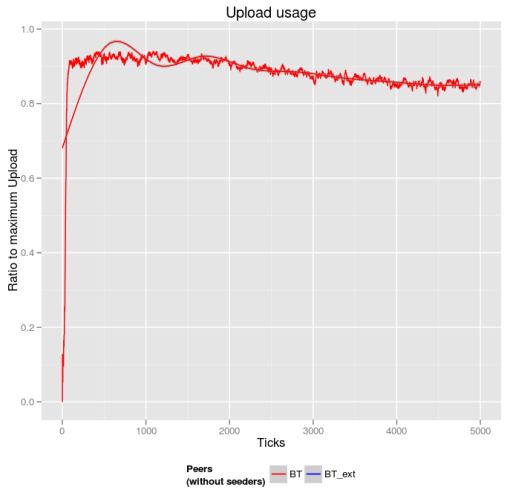


Figure E.80: dynamic,500p,0pc1,1000MB

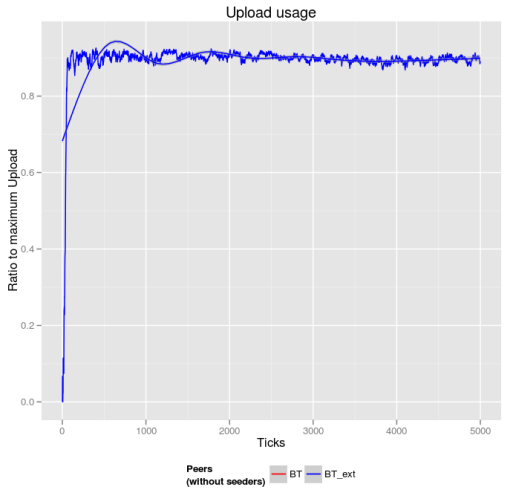


Figure E.81: dynamic,0p,500pc1,1000MB

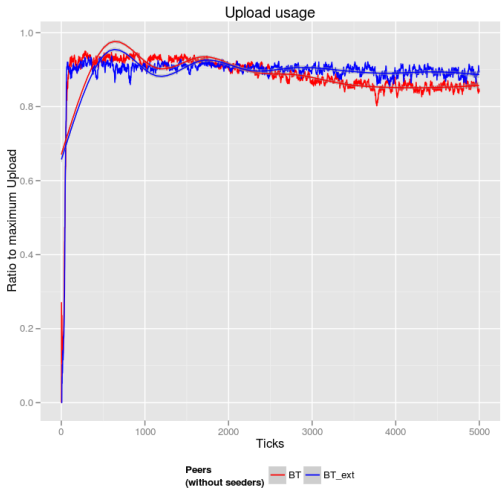


Figure E.82: dynamic,250p,250pc1,1000MB

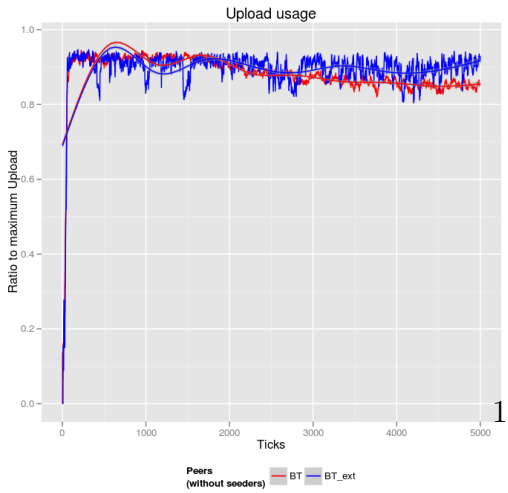


Figure E.83: dynamic,450p,50pc1,1000MB

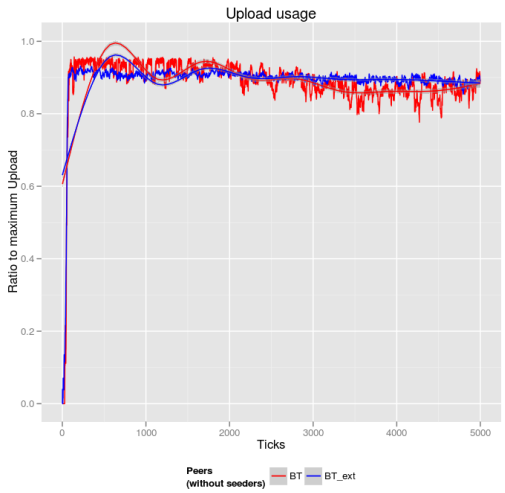


Figure E.84: dynamic,50p,450pc1,1000MB

Figure E.85: **dynamic,1000MB Scaled download rate**

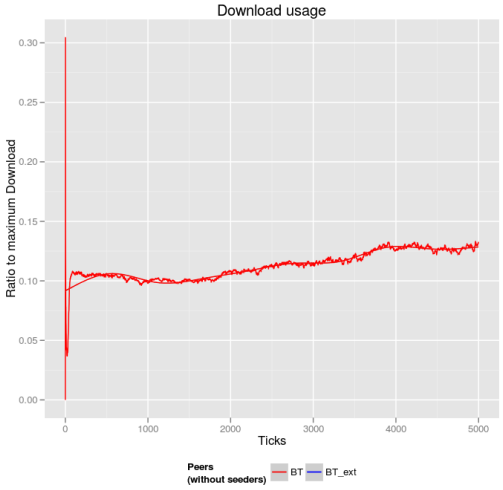


Figure E.86: dynamic,500p,0pc1,1000MB

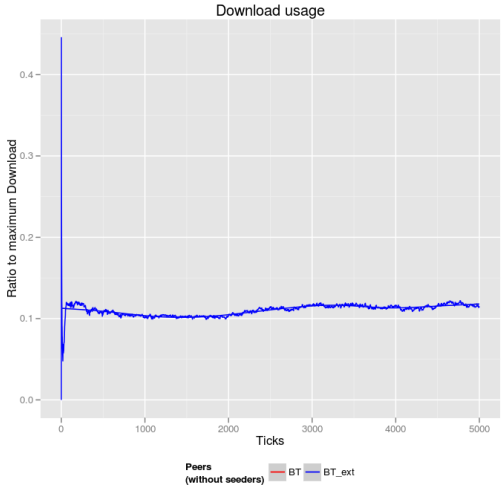


Figure E.87: dynamic,0p,500pc1,1000MB

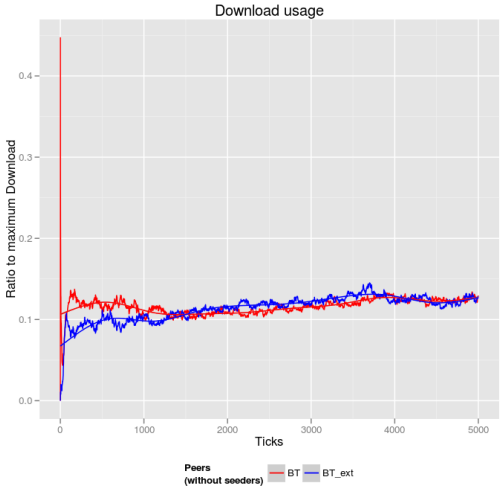


Figure E.88: dynamic,250p,250pc1,1000MB

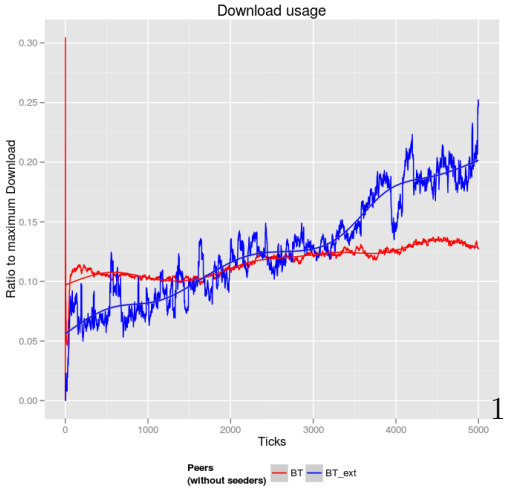


Figure E.89: dynamic,450p,50pc1,1000MB

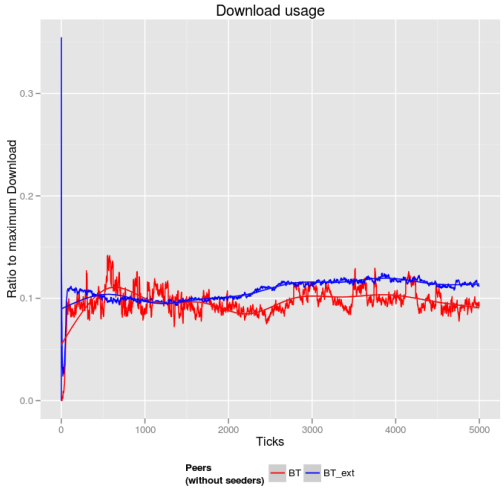


Figure E.90: dynamic,50p,450pc1,1000MB