Reducing Free Riding in Peer-to-Peer Systems with Supervised Teaming

Boudewijn Schoon

Delft University of Technology

Reducing Free Riding in Peer-to-Peer Systems with Supervised Teaming

Master thesis in Computer Science

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Boudewijn Schoon
✉ peer-to-peer@frayja.com

April 18, 2008

Author
Boudewijn Schoon

Title
Reducing Free Riding in Peer-to-Peer Systems with Supervised Teaming

MSc presentation
May 6, 2008

Graduation Committee

Prof. dr. ir. H. J. Sips (chair)	Delft University of Technology
Dr. ir. J. Pouwelse	Delft University of Technology
Ir. dr. D. H. J. Epema	Delft University of Technology
Dr. ir. D. de Ridder	Delft University of Technology

Abstract

Peer-to-peer technology has produced thriving communities in which peers contribute bandwidth to each other. However, when free riding occurs, these communities will not be able to sustain themselves without incentives to enforce this contribution. This thesis presents Supervised Teaming, a peer-to-peer transfer protocol that gives uploading peers, called supervisors, control over a team of downloading peers, called team members. The team members are given an incentive to transfer data among each other, effectively reducing the upload cost of the supervisors to one piece of data while still duplicating this piece to every team member. Using transport efficiency, time efficiency, and sharing ratio as performance metrics, we prove that Supervised Teaming performs equally to BitTorrent under best-case scenarios and several factors better, depending on the chosen team size, under flash crowds and free riding scenarios. Furthermore, we have implemented a peer-to-peer client that can use both the BitTorrent protocol and a simplified version of the Supervised Teaming protocol. The experiments we have performed with this client verify that Supervised Teaming performs as expected.

Preface

In this thesis a peer-to-peer protocol is discussed that attempts to provide protection against threats that can occur in modern peer-to-peer communities. The research and development reported in this thesis have been conducted in the context of the I-Share project of Delft University of Technology and other institutes, which focuses on the sharing of resources in virtual communities for storage, communications, and processing of multimedia data.

When I started my Master thesis, one and a half year ago, I found the subject of free riding and other peer-to-peer system abuse very interesting. At that time no work was being done exclusively on this subject in Delft, making it a good choice for me to focus on. During my research assignment, where I catalogued problems and solutions in peer-to-peer systems, I was already formulating criteria that are required to create a secure peer-to-peer protocol. Therefore, it was only a small step to continue on this course with my Master thesis and develop a peer-to-peer protocol that would be less vulnerable to free riding and other common problems.

I am grateful to the people of the Parallel and Distributed Systems Group, for their on- and off-topic discussions and their company in general. Specifically, I would like to thank dr. ir. Johan Pouwelse for his advise and numerous reviews during the time that I spent on research, implementation, and evaluation. Furthermore, I would like to thank ir. dr. Dick Epema for his advice during the final months. Finally, I am very grateful to my friends and family, in particular my parents, Cor and Hanneke Schoon, for their support and gentle nudges when needed; Silvian de Jager, for his patience; Pascal Vree, for his down-to-earth advice; and Michel Hale for his time and effort during the final weeks.

Boudewijn Schoon

Delft, The Netherlands
April 18, 2008

Contents

Chapter 1

Introduction

Take the first step, and your mind will mobilize all its forces to your aid.
But the first essential is that you begin. Once the battle is started,
all that is within and without you will come to your assistance.
- Robert Collier

Currently, a lot of people have come into contact with peer-to-peer systems. BitTorrent [7], developed by Bram Cohen, is easily the most used protocol at this time. Studies [15, 19] show that in the year 2004, BitTorrent was responsible for roughly 30% of all internet traffic. Since this time the popularity of BitTorrent has only increased. Traditionally, peer-to-peer systems have been used to give a large community an easy way to access files. These files can range from music tracks of a few megabytes to DVD copies of several gigabytes.

Looking at the history of peer-to-peer systems we can see that a lot has changed: the number of users has increased, the amount of traffic has grown by 8% per month since 1997 [1], and the type of data has shifted from mostly music to the larger video files. However, these changes did not require the development of new peer-to-peer systems. It is the attitude of users towards peer-to-peer systems that causes problems that force peer-to-peer systems to either adapt or go extinct. This thesis will focus on a solution to the following three problems: *flash crowds*, where large numbers of peers join in a short amount of time; *free riding*, where peers reduce their bandwidth cost as much as possible; and *initial risk*, which is a form of free riding where generosity towards new peers is exploited. We present the Supervised Teaming protocol and prove, using both theory and practical experiments, that it is capable of handling these problems in a scalable and low-risk way. However, first these three problems will be discussed in detail.

1.1 Problem description

There are many methods to exploit a peer-to-peer system and we believe, as peer-to-peer communities continue to increase in size, that the current problems will also increase. Among these problems we believe that flash crowds, free riding and initial risk are good representatives of what peer-to-peer systems are currently facing. Therefore, we will use these three problems to assess the effectiveness of both the BitTorrent and the Supervised Teaming protocol.

Flash crowd. Peer-to-peer systems ideally contain thousands of peers sharing millions of files. However, these files usually originate from a single peer called the initial seeder. When a peer wants to introduce a new file, this file must be transferred to the community as quickly as possible to increase the chances of other peers retrieving this data fast and reliably. When the file is very popular, the number of peers who are trying to retrieve data can grow very large in a short amount of time, which is known as the flash crowd problem.

Free riding. Our second problem only recently became a problem. At the dawn of peer-to-peer technology the applications connecting the peers could be trusted due to the novelty of peer-to-peer systems. This made it possible to have all peers behave altruistically. However, this has changed. Modern peer-to-peer systems are either open source or reverse engineered, allowing the development of alternative peer-to-peer clients. These clients will only follow the prescribed protocol when they are given sufficient incentives. In other words: we have to assume that peers, in a modern peer-to-peer system, behave selfishly.

A selfish peer that is downloading more than her 'fair share' is said to be free riding. This definition is as broad as it is vague. However, the general idea is as follows: in a peer-to-peer community, each peer should contribute something back to the system in order to keep it going. When resources of a community are used beyond what it can sustain, the *tragedy of the commons* [14] occurs. There are numerous ways that a peer can free ride, perhaps the client allows its upload rate to be set to a low value, or fake information can be introduced in the system. The options vary depending on the peer-to-peer system that is used.

No system to date had been able to completely eliminate free riding, nor do we believe that such a system will ever emerge. In a peer-to-peer system there will always be peers who upload more—and consequently, other peers will upload less—than their fair share. A peer who introduces a file will only be able to upload and the last peer will not be able to upload simply because there is no one left who is interested. Another complication originates from the topology of the internet itself, when two peers are behind a Network Address Translation (NAT) or firewall any communication between them is difficult and in some cases even impossible.

Initial risk. Our third problem involves a specific form of free riding. Several peer-to-peer systems use incentives to convince peers to help share a resource. Tit-for-tat and counting the number of bytes received and sent between peers, all come down to the simple: 'You scratch my back and I'll scratch yours' principle. However, this concept has a drawback: who scratches first? Or in other words: who will take the initial risk.

When two peers encounter each other for the first time, one will have to take some initial risk. Because the number of peers in a peer-to-peer system can be very large, it is likely that a lot of unknown peers are encountered. And each encounter results in at least some initial risk. This problem is even larger in peer-to-peer systems where no user identification is required; in these cases the *whitewashing* [9] attack is an easy way for a free rider to continuously take advantage of initial risk.

The challenge of this thesis is to design a peer-to-peer system that must be able to cope with the above three problems. Our protocol must be efficient to those who are willing to share and able to ensure that all peers are uploading at least some data. Our solution, called Supervised Teaming, is a protocol that ensures this with the cooperation of peers in small teams. These teams consist of *team members* who are peers that are interested in downloading a specific piece. Furthermore, the team members follow the instructions of the *supervisor*, who is the peer that has the piece that the team members are going to download. There is always one supervisor and one or more team members. The *team size* is the number of peers who are cooperating to download a specific piece, therefore, the team size does not include the supervisor. Each team member receives blocks of data from the supervisor and must forward those blocks to the other team members. An incentive for this forward is provided in the form of a reward that contains the offset of the data block. Without this offset, the block is useless. And without forwarding, a peer will never receive the reward.

Throughout this thesis we will use transport efficiency, time efficiency, and sharing ratio to compare the protocols that are discussed. The *transport efficiency* is the ratio of the useful data to all the used bandwidth, expressed as a percentage, where useful data is the data that is intended to be received by the downloading peer and, therefore, does not include the overhead from additional messages or message headers. The *time efficiency* is the ratio of the minimal transfer time to the actual transfer time, expressed as a percentage, where the minimal transfer time is the time it takes to transfer the data that is actually part of the file. The *sharing ratio* is the ratio of the upload cost to the download cost. However, because an initial seeder has no or very low download cost, we define the download cost for an initial seeder to be the size of the file that is shared. In short, the transport efficiency and the time efficiency indicate how efficiently a protocol handles message overhead and latency, whereas the sharing ratio indicates the distribution of cost between peers in a swarm.

1.2 Thesis outline

This thesis is organized as follows. In Chapter 2, the BitTorrent protocol is introduced. Because the BitTorrent protocol is one of the leading peer-to-peer protocols at this time, it offers a valuable reference point against which we can measure the performance of our protocol. In this chapter we will present a short history and a detailed discussion of the BitTorrent protocol. This is followed with an analysis where the transport efficiency, the time efficiency, and the sharing ratio are presented. We conclude this chapter by assessing to what extent the BitTorrent protocol offers protection against the three problems that are posed in the previous section.

In Chapter 3 the Light Supervised Teaming protocol is introduced, which is an implementation of the Supervised Teaming protocol where a supervisor has authority over a teams consisting of two team members. Having two team members allows simplifications in the protocol that made it possible to implement this protocol within the time that we wanted to spend on the verification of our theoretical results. In this chapter we will present a detailed discussion and analysis of the Light Supervised Teaming protocol, for which we use the transport efficiency, the time efficiency, and the sharing ration. We also assess to what extent the three problems, that are posed in the previous section, are solved by Light Supervised Teaming.

In Chapter 4 the Extended Supervised Teaming protocol is introduced. In contrast to the Light Supervised Teaming protocol, the Extended Supervised Teaming protocol is not restricted to a team size of two. Instead a supervisor can create a team with any number of team members and can add and remove team members when required. How this influences the protocol is discussed in the protocol description and performance analysis, where we calculate the transport efficiency, the time efficiency, and the sharing ratio. Finally, we discuss how the Extended Supervised Teaming protocol addresses the problems of flash crowd, free riding, and initial risk.

In Chapter 5 we will present several experiments we have performed using our peer-to-peer client, BitTorrent, and Light Supervised Teaming implementations. This chapter will show the similarity between theory and practice in addition to experiments that show the difference between BitTorrent and Light Supervised Teaming in flash crowd, free riding, and initial risk situations.

Finally, we present our conclusions in Chapter 6.

Chapter 2

BitTorrent

Why is it that, as we grow older, we are so reluctant to change?
It is not so much that new ideas are painful, for they are not.
It is that old ideas are seldom entirely false, but have truth, great truth in them.
The justification for conservatism is the desire to preserve the truths and standards of the past;
its dangers, of which we are seldom aware, is that in preserving those values, we may miss the infinitely greater riches that lie in the future.
- Dale E. Turner

The BitTorrent protocol plays an important role in this thesis. First of all, our Supervised Teaming protocol uses a similar torrent, piece, and block structure. Secondly, BitTorrent is currently the dominant peer-to-peer system in the world [15, 19] and is therefore a good candidate with which our own Supervised Teaming approach can be compared.

Section 2.1 gives a short history of the BitTorrent protocol. Section 2.2 explains how the protocol works. Section 2.3 discusses the three performance metrics that are used, throughout this thesis, to evaluate the different protocols. Section 2.4 discusses how efficient the protocol is in terms of the transport efficiency, the time efficiency, and the sharing ratio. And finally in Section 2.5 we assess how the BitTorrent protocol handles flash crowd, free riding, and initial risk problems.

2.1 History

After the introduction of BitTorrent in 2004 by Bram Cohen, this peer-to-peer system has become one of the most popular protocols for mass file sharing on the internet. One of the reasons for the popularity is the good scaling that the protocol provides. With a traditional client/server approach the cost for a server

increases linearly with the population, making the server the bottleneck of the system. However, with the BitTorrent protocol the collective upload bandwidth increases with the population, removing this bottleneck.

BitTorrent is peer-to-peer system that gives users an incentive to provide bandwidth to each other. This incentive is based on a strategy called *tit-for-tat*. In 1981 Axelrod [3] organized a tournament involving several players who could choose whether they wanted to cooperate with each other or not. The results from this tournament, presented in the Evolution of Cooperation, indicated that tit-for-tat was both the best and one of the simplest submitted strategies to solve this, what we now call, *Prisoners Dilemma* [8]. The tit-for-tat strategy had two basic rules: cooperate on the first round and continue to do whatever the other player did in the preceding round.

Axelrod concluded that the success of tit-for-tat could be traced back to three rules: never be the first to stop cooperating, retaliate after the other player does not cooperate, and forgive when the other player starts cooperating again. Tit-for-tat remained one of the most effective strategies to solve the Prisoner Dilemma until cooperation between a select group of players was used to boost the score of a single player [21]. By sacrificing the other players in the group, one player could be more effective than a player using tit-for-tat. This cooperation is often seen in attempts to bypass trust systems and is called *collusion* [8].

2.2 Protocol description

The general idea of BitTorrent is, in short, break a file into pieces and let the different peers download each piece separately. Once a peer has downloaded a piece, she can start contributing to the collective bandwidth of the swarm by uploading that piece to other peers. This lets the available upload bandwidth scale with the number of available peers, preventing bottlenecks and problems with flash crowds. Choosing which piece to download first is important to increase the chance that, during the lifetime of the swarm, no pieces are lost. Different *piece picking policies* [7] can be used depending on how much the peer has downloaded so far and how much of the data is available.

Once peers are connected, they continually inform each other of the pieces that they have. Using this knowledge peers can ask other peers for bytes from a piece that they have available. Transferring a single piece between peers is divided into two different phases. First is the information phase, and second the transfer phase. We will continue with a detailed description of these phases. Flow diagrams for the seeder and leecher are given in Figures 2.1 and 2.2, respectively. A detailed description of the messages that are used, including size and payload, are given in Appendix A.1.

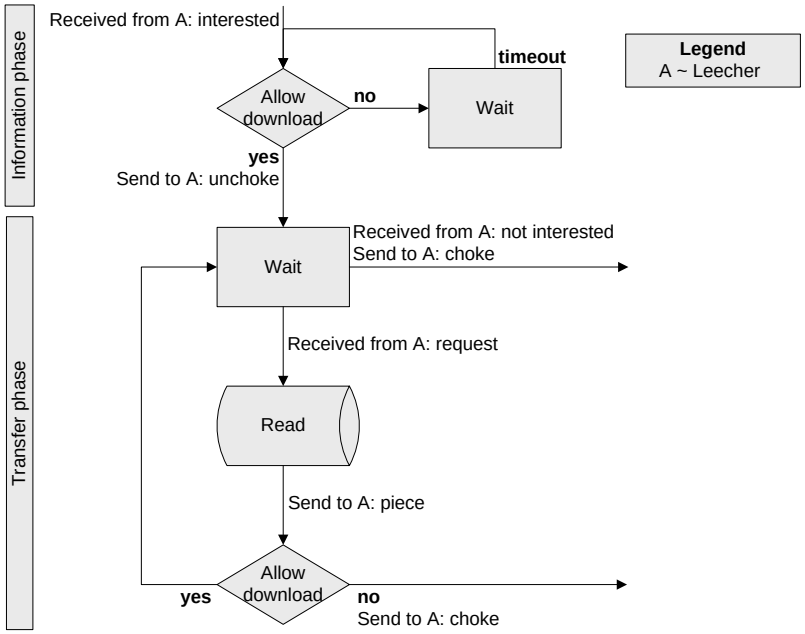


Figure 2.1: *The flow diagram for seeders.*

Information phase

BitTorrent uses a *pulling* strategy, meaning that a peer needs to request transfers from other peers. However, before data can be requested, a peer needs to know what data is available on the other peers. This knowledge is gathered using `HAVE` and `BITFIELD` messages.

When a connection is first established between peers the `BITFIELD` message may be sent. One such message will contain a bit for each piece in the torrent and each bit indicates whether the peer has this piece by either setting or unsetting this bit. While a connection exists between peers, the `HAVE` message is sent to indicate the availability of newly acquired pieces at a peer.

When peer A receives a `BITFIELD` or a `HAVE` message that indicates that peer B has a piece that A still needs, then A can send an `INTERESTED` message to B to indicate that A would like to start downloading from B. From this point on A will be called the leecher and B will be called the seeder.

When a seeder receives an `INTERESTED` message, she can decide whether or not to allow a download. This can be based on current bandwidth usage, or part of a strategy that finds better peers to cooperate with. When the seeder decides that she can provide the leecher with some data she sends an `UNCHOKE` message back. Otherwise no action is taken.

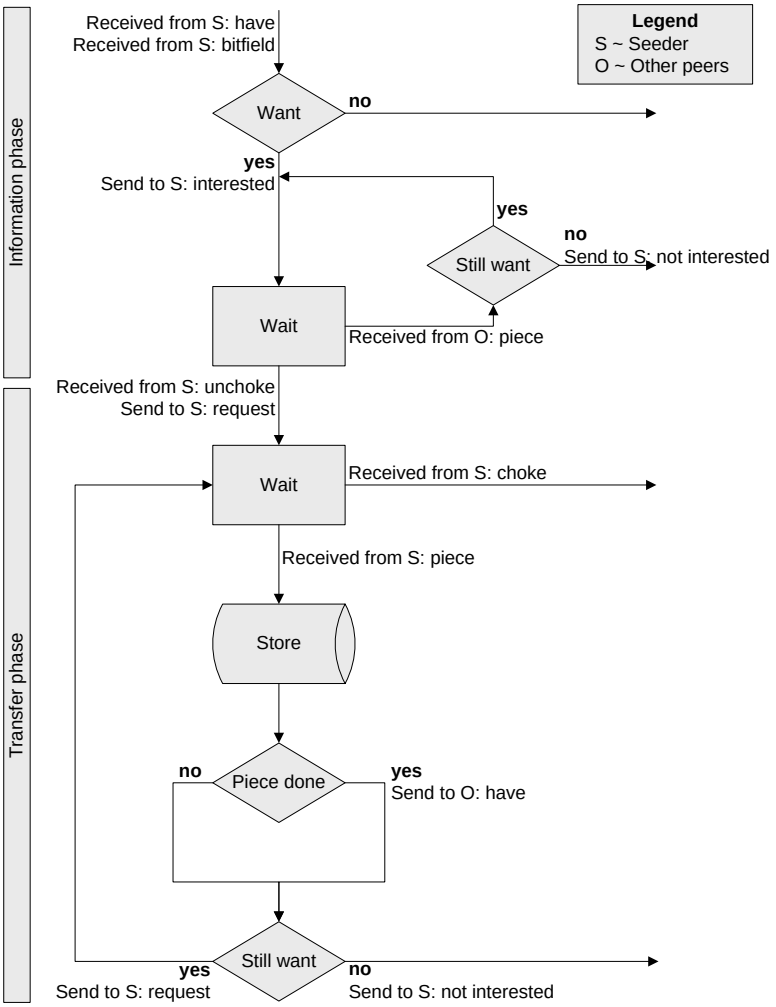


Figure 2.2: *The flow diagram for leechers.*

Transfer phase

Once a leecher receives an `UNCHOKE` message the transfer phase begins. This `UNCHOKE` message indicates that the seeder is willing to upload data to the leecher, all that remains is for the leecher to tell the seeder what data she requires.

The leecher sends one or more `REQUEST` messages to the seeder and the seeder sends `PIECE` messages back containing the requested data. This continues until either the leecher sends a `NOT INTERESTED` message, indicating that she no longer wants any data from the seeder; or until the seeder sends a `CHOKE` message, telling the leecher that no further data will be uploaded.

A leecher usually has several outstanding requests at a seeder because this allows the seeder to provide a continuous stream of data, reducing latency and therefore increasing the time efficiency of the protocol. This is further discussed in Section 2.4.2.

2.3 Performance metrics

Throughout this thesis we will use the transport efficiency, the time efficiency, and the sharing ratio to evaluate and compare the performance of the BitTorrent protocol, the Light Supervised Teaming protocol, and the Extended Supervised Teaming protocol. What these performance metrics are is discussed in the following three sections.

2.3.1 Transport efficiency

The transport efficiency is the ratio of the useful data to the used bandwidth, expressed as a percentage, where the useful data is every byte that is intended to be received by the downloading peer or peers. The transport efficiency is calculated for the transfer of a single piece of data from a single source-peer to one or more destination peers. Using the transport efficiency we can see how efficiently the available bandwidth is used.

For the BitTorrent protocol the useful data is every byte that is part of the shared file. The used bandwidth is every byte that is transferred. Because there are exactly two peers involved in the transfer of a piece of data, the seeder and the leecher, the useful data and the used bandwidth are the same for both these peers. Therefore the transport efficiency for the seeder is the same as that of the leecher when the BitTorrent protocol is used.

For the Supervised Teaming protocol, where two or more peers are involved in the transfer of a piece of data, the definition of useful data is more complex. The useful data to a team member is every byte that is part of the shared file which is received. However, the useful data to a supervisor is every byte that is part of the shared file which is received by, possibly, several team members. Therefore, when

a 1MB piece is transfered from a supervisor to four team members, the useful data is 1MB and 4MB for each of the team members and the supervisor, respectively.

2.3.2 Time efficiency

The time efficiency is the ratio of minimal transfer time to the actual transfer time, expressed as a percentage, where the minimal transfer time is the theoretical minimum amount of time that the transfer will take given a certain latency and bandwidth. The time efficiency is calculated for the transfer of a single piece of data from a single source-peer to one of more destination peers. Using the time efficiency we can see the affect of the connection speed and latency.

2.3.3 Sharing ratio

The sharing ratio is the ratio of the number of uploaded bytes to the number of downloaded bytes. However, while the transport and time efficiency refer to the efficiency for the transfer of a single piece, the sharing ratio is calculated for the distribution of an entire file in an entire swarm consisting of multiple peers. Using the sharing ratio we can see how the bandwidth cost is distributed between seeders, leechers, and free riders.

Predicting the efficiency for an entire swarm is difficult. The piece picking policy, the number of concurrent up and downloads, different bandwidth and latency between different peers, connectability, and many other factors affect the sharing ratio. Therefore, we will simplify the sharing ratio by not including any overhead from additional messages or message headers. Neither will the latency be included in the sharing ratio calculation.

Sharing ratio will be based on the size of the file that is shared, the number of peers and the amount of available upload bandwidth that these peers have available. Furthermore, because no overhead is present, we can assume that both the leechers and the free riders download no more than the file that is shared. For a similar reason, the seeders will download nothing at all. Therefore, when calculating the sharing ratio for seeders, we define the number of downloaded bytes to be the size of the file that is shared.

2.4 Performance analysis

To get a reference point for the performance of the Supervised Teaming protocol, which is the main focus of this thesis, we compare it against the performance of the popular BitTorrent protocol. Therefore, this section will present an analysis of the performance of the BitTorrent protocol. The comparisons between Supervised Teaming and BitTorrent will be presented in Sections 3.2 and 4.2 for the Light Supervised Teaming and Extended Supervised Teaming protocols, respectively.

Symbol	Meaning	Unit
B	Size of block	kB
F	Size of torrent or file	kB*
L	Latency	ms
M	Number of team members	
NF	Number of free riders	
NL	Number of leechers	
NS	Number of initial seeders	
P	Size of piece	kB*
RF	Sharing ratio of free riders	
RL	Sharing ratio of leechers	
RS	Sharing ratio of initial seeders	
SF	Upload speed of free riders	kB/s
SL	Upload speed of leechers	kB/s
SS	Upload speed of initial seeders	kB/s
T	Time	s

*Given in MB when otherwise specified

Table 2.1: *The symbols used throughout this thesis.*

The equations and parameters used throughout this thesis use the notation presented in Table 2.1. Furthermore, we have performed several experiments using our own peer-to-peer client and protocol implementations to verify that the results from the theories presented in this thesis correspond with an actual file transfer. These experiments are presented and discussed in Chapter 5.

2.4.1 Transport efficiency

During the transfer of a file, any number of connection failures or deviations from the protocol can occur. To simplify we will not take into account any such failures or deviations and assume that the transfer takes place as described in the protocol description in Section 2.2.

The transport efficiency depends on four factors: connection failures, deviations from the protocol, size of pieces, and size of blocks. Since we will not take the first two factors into account, we are left with the size of both the pieces and the blocks. The best efficiency can be attained by having large pieces and blocks because this will reduce the overhead of request and message headers. Therefore, from a purely transport efficiency point of view, the best piece and block size would equal the size of the entire file that is shared. However, this would remove any possibility for trading during transfer, and since this is the strongest aspect of a peer-to-peer system this is not desired.

Because the .torrent file is hosted at central servers, and these servers have limited available bandwidth, it is preferred to have relatively small .torrent files. Because

Message	Seeder Upload (bytes)	Seeder Download (bytes)	Leecher Upload (bytes)	Leecher Download (bytes)
Information phase				
`HAVE`	9	0	0	9
`INTERESTED`	0	9	9	0
`UNCHOKE`	5	0	0	5
Transfer phase (x128)				
`REQUEST`	0	2,176	2,176	0
`PIECE`	4,195,968	0	0	4,195,968
Total				
	4,195,982	2,185	2,185	4,195,982
Efficiency				
Total bandwidth	4,198,167		4,198,167	
Overhead	3,863		3,863	
Transport efficiency	**99.91%**		**99.91%**	

Table 2.2: *The transport efficiency. [P=4MB, B=32kB]*

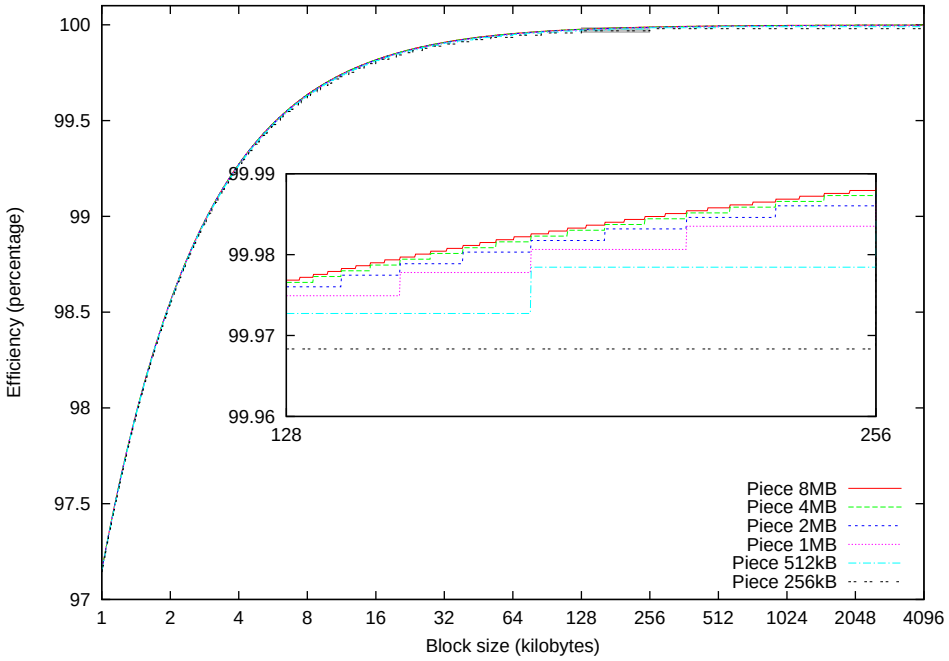


Figure 2.3: *The transport efficiency.*

the size of the .torrent file is directly dependent on the number of pieces in a torrent, the piece size is therefore constrained by the practical maximum size of a .torrent file. While there is no general agreement on a specific piece size, the most commonly encountered piece sizes, for torrents of several gigabytes, are 1MB, 2MB, and 4MB.

There are no general agreements for the block size either. However, there are two limits: the length of the piece and the maximum message length. The length of the piece might, from a technical point of view, be used as a block size. The maximum message length, which is over 4GB, is obviously not used. In the end it comes down to what the BitTorrent clients choose to allow. Commonly encountered block sizes are 8kB, 16kB, and 32kB.

An efficiency computation for the transfer of a single 4MB piece from a seeder to a leecher using 32kB blocks can be seen in Table 2.2. The table shows the various messages that are sent during transfer and their respective sizes including the five byte message headers. When we look at the bottom of the table we can see that the resulting efficiency is a little under 100% for both the seeder and the leecher, which means that the BitTorrent protocol adds little overhead to the 4MB that the seeder allowed to upload and what the leecher wanted to download.

The efficiency result that is shown in Table 2.2 can be repeated for many different piece and block size combinations. By collecting the results of several combinations we can create an overview of the transport efficiency for the BitTorrent protocol under different circumstances. This overview is presented in Figure 2.3. This figure shows that changing the piece size hardly changes the efficiency at all, changing the block size, on the other hand, does change the efficiency. When choosing a piece and block size it should be taken into account that a big block size increases the risk of free riding, and a small block size increases the overhead.

To explain the somewhat surprising similarity between the different piece sizes in Figure 2.3 we will look at two cases in detail. In the first case we transfer a 8MB piece by sending 4096 blocks of 2kB. For the second case we transfer a 265kB piece by sending 128 blocks of 2kB. It is clear that the first case gives more overhead, however, this overhead is relatively small compared to the piece size. Therefore, this has only a small effect on the efficiency. For this reason the first case is slightly more efficient at 98.56% than case two at 98.55%. Such differences are too small, except for the area that is enlarged, to show in the figure.

2.4.2 Time efficiency

The time efficiency depends on six factors: connection failures, deviations from the protocol, size of pieces, size of blocks, latency of the connection, and bandwidth of the connection. As explained in the previous section, we will not take into account any connection failures or deviations from the protocol. The piece and block size affects the time efficiency in the same way as described in the previous

Message	Latency time (milliseconds)	Transfer time (milliseconds)	
Information phase			
`HAVE`	200	0.29	
`INTERESTED`	200	0.29	
`UNCHOKE`	200	0.16	
Transfer phase (x128)			
`REQUEST`	200	70.83	
`PIECE`	200	136587.50	
Total			
	1000	136659.08	
Efficiency			
Total in seconds	1.00	136.66	**137.66**
Overhead in seconds	0.80	0.13	**0.93**
Time efficiency			**99.33%**

Table 2.3: *The time efficiency. [P=4MB, B=32kB, L=200ms, SS=SL=30kB/s]*

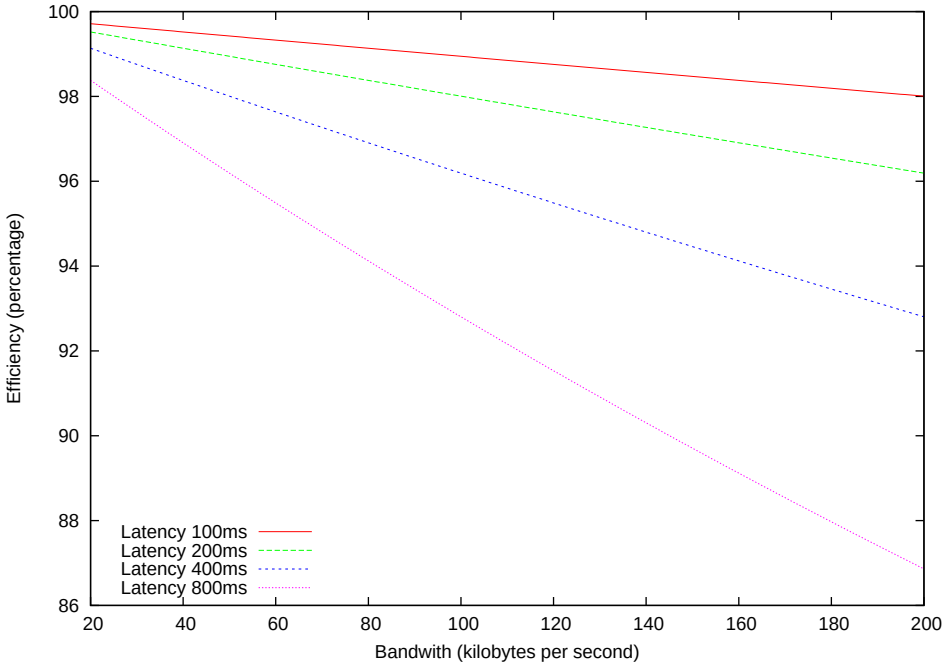


Figure 2.4: *The time efficiency. [P=4MB, B=32kB]*

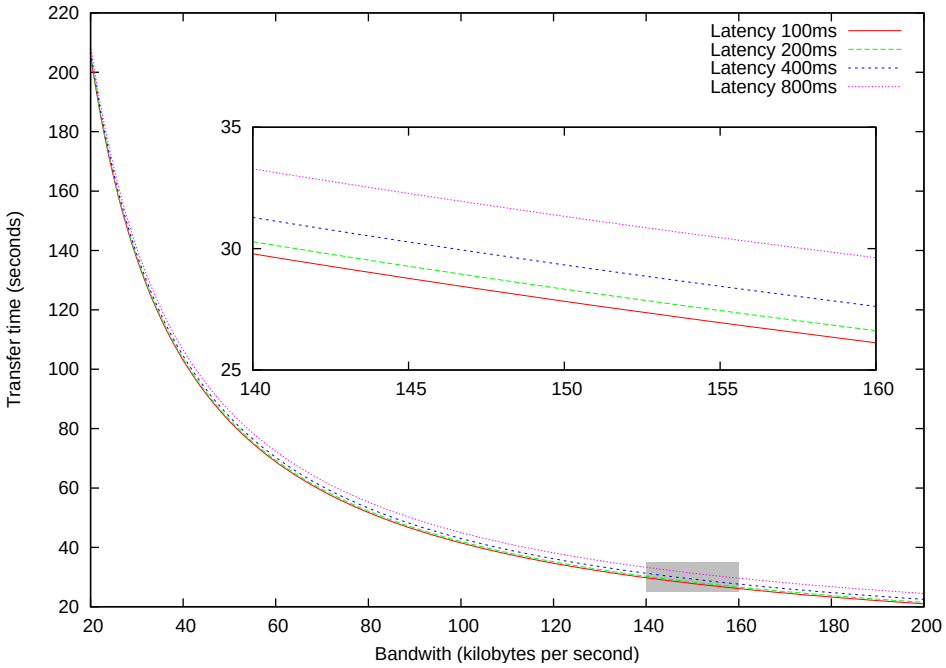


Figure 2.5: *The transfer time. [P=4MB, B=32kB]*

section. Furthermore, we will assume that the transfer takes place as described in Section 2.2.

The latency of a connection is the time that it takes between sending the first byte of a message and receiving this first byte at the other end of the connection. Having a low latency will decrease this delay thus increasing the efficiency. The bandwidth of a connection indicates the number of bytes that can be sent through the connection in a certain amount of time. Having a high bandwidth will allow more data to travel from one peer to the other in less time, thus increasing the efficiency. While not all peers have the same up and download bandwidth, we will simplify our analysis and use the same latency and bandwidth for both communicating peers.

An efficiency prediction for the transfer of a single 4MB piece in 32kB blocks with a 200ms latency and using a bandwidth of 30kB/s can be seen in Table 2.3. The table shows the various messages that are sent during transfer and the corresponding latency and transfer time delays. When we assume that the optimal transfer of 4MB of data at 30kB/s takes 136 seconds, then we can calculate the efficiency for the actual transfer. The resulting efficiency of 99.33% is shown at the bottom of the table. Having such a high efficiency means that the BitTorrent protocol adds little additional delay to the transfer.

One of the reasons for this high efficiency is that a peer will always attempt to

queue several requests. This allows the seeder to send a continuous stream of data, eliminating any additional delay due to latency. Not using a request queue would result in a latency time increase of 2×200 for each block. This would result in over 50 seconds additional delay. Thus, by having a request queue the latency hardly affects the time efficiency of the BitTorrent protocol.

The efficiency result that is shown in Table 2.3 can be generated for many different piece, block, latency and bandwidth combinations. By generating the efficiency values for several combinations we can plot an overview of the time efficiency for the BitTorrent protocol under different circumstances. This overview is presented in Figure 2.4. The four lines, indicating different latency values, clearly show a difference in efficiency. As could be expected, better efficiency can be achieved with a lower latency. However, as the bandwidth increases, the efficiency decreases. Even though this is somewhat counterintuitive it can be explained simply because as the bandwidth increases, the norm—the optimal transfer time that can only be achieved in theory—increases even more because no protocol overhead is used in this optimal value.

A more intuitive way of looking at the time efficiency is how much time it takes to transfer a piece with a certain block, latency, and bandwidth. These values are generated and plotted in Figure 2.5. In contrast to the time efficiency, where a higher bandwidth seemed to cause a negative effect, the transfer time reacts positive as the bandwidth increases. In this figure we can see that the latency does not have much affect on the transfer time.

2.4.3 Sharing ratio

The sharing ratio, for the initial seeder, leechers, and free riders in a swarm, can be expressed by the following equations which maximize the usage of the available bandwidth:

$$T = \frac{F \times (NL + NF)}{SS \times NS + SL \times NL} \tag{2.1}$$

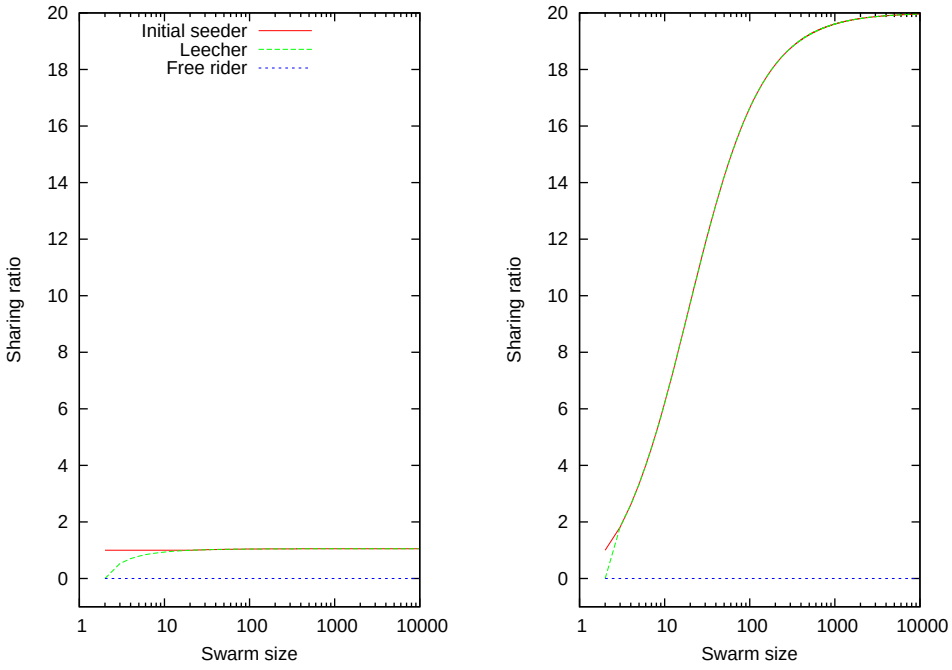
$$RS = \frac{SS \times T}{F} \tag{2.2}$$

$$RL = \frac{SL \times T}{F} \tag{2.3}$$

$$RF = \frac{0}{F} = 0 \tag{2.4}$$

However, these equations do not take into account that the initial seeder is required to upload the file at least once. Therefore, these equations are only valid when $SS \times NS \times T \geq F$. The equations that generate the swarm efficiency figures in this thesis are slightly different to take this into account.

To clarify the strength and weakness of BitTorrent we show two extreme cases where a single initial seeder has the task of uploading a file to a swarm. In the



(a) Best-case scenario with 5% free riders.

(b) Worst-case scenario with 95% free riders.

Figure 2.6: *The sharing ratio. [F=100MB, NS=1, SS=SL=SF=128kB/s]*

first case 5% of the swarm consists of free riders while in the second case 95% of the swarm consists of free riders. For each case we calculate the sharing ratios, using the above equations, with a varying swarm size. The resulting graphs are shown in Figure 2.6. These figures show the swarm size, which includes the single initial seeder, on the horizontal axis. The vertical axis shows the corresponding sharing ratios for the initial seeder, the leechers, and the free riders. These figures were generated for a 100MB file and an upload bandwidth of 128kB/s for the initial seeder, the leechers, and the free riders.

The strength of BitTorrent, or any peer-to-peer system for that matter, is shown in Figure 2.6(a). From this figure we see that, as the swarm size increases, the sharing ratio for the initial seeder and the leechers remains more or less equal to one. In other words, the bandwidth cost is spread more or less equally between the peers, with the exception of the few free riders who do not upload anything at all.

Unfortunately, the sharing ratio for a swarm that consists for 95% of free riders requires a much higher sharing ratio from the initial seeder and leechers. This worst-case scenario is shown in Figure 2.6(b). That the sharing ratio, for the initial seeder and the leechers, approaches 20 is caused by the available leechers. A swarm consisting entirely of free riders would require the initial seeder to upload everything and hence have a sharing ratio of $1{:}\infty$, which is similar to what is seen in a traditional client/server approach.

2.5 Concerning the problem statement

The previous chapter stated the three problems that we will address in this thesis. This section will describe how BitTorrent deals with these problems.

Flash crowd. A simple solution to the flash crowd problem, or in other words, dealing with large numbers of leechers with relatively few seeders, is the *super seeding* [4, 5, 24] strategy. This strategy ensures two things: first, the seeder does not choke a leecher until that has downloaded a partial piece. This ensures that the leecher is able to upload a completed piece to other peers. Second, the seeder prevents premature serving of duplicate pieces to maximize the diversity of the available pieces. The super seeding strategy has proven itself for giving a noticeable improvement to the bandwidth utilization of seeders.

Free riding. BitTorrent has proven itself to be resilient against free riding, even though there are several strategies that free riders can use [17, 18]. As long as there are enough altruistic peers, both the cooperating and the free riding peers, will be able to complete their download, keeping the swarm healthy [13]. However, as more free riding approaches are incorporated into user friendly BitTorrent clients, the number of free riders will increase.

One such user friendly free riding application is RatioMaster [2]. This program uses a HTTP proxy to spoof the statistics that are sent to the tracker, making it possible to fool a private tracker into believing that the user is a great benefit to the community. Another strategy to start free riding is to limit the upload bandwidth—an option that all well known peer-to-peer clients now provide—coupled with increasingly selfish and aggressive implementations, also limits the amount of bandwidth that a free rider will contribute to the peer-to-peer community.

It is our belief that the BitTorrent protocol will not be able to cope with a high number of free riders in the network, resulting in more independent swarm failures. Currently most peers behave as the protocol desires. However, the protocol does not have any options to make it harder for free riders and because of this lack of control the BitTorrent protocol will eventually collapse under a growing number of selfish users.

Initial risk. One of the reasons that free riding can be achieved in a BitTorrent swarm is the initial risk. A new peer can only upload a piece once it is completely downloaded, therefore, an uploading peer risks an entire piece worth of bandwidth before the new peer is able to provide her upload bandwidth to the community.

One might argue that this initial risk can be reduced by decreasing the size of the pieces, while this is true, this has several disadvantages: smaller pieces will require a larger `BITFIELD` message, more `HAVE` messages to be sent between peers, and the .torrent file will grow larger because of the additional hash values that have to

be stored. However, even small pieces will not remove the initial risk because the BitTorrent protocol does not require a peer to upload any data. While the tit-for-tat policy suggests that uploading will increase the download performance, a peer is not guaranteed that this will actually be the case. While the optimistic unchoke is designed to find peers where this will be the case, this also gives free riders opportunities to receive free data.

Chapter 3

Light Supervised Teaming

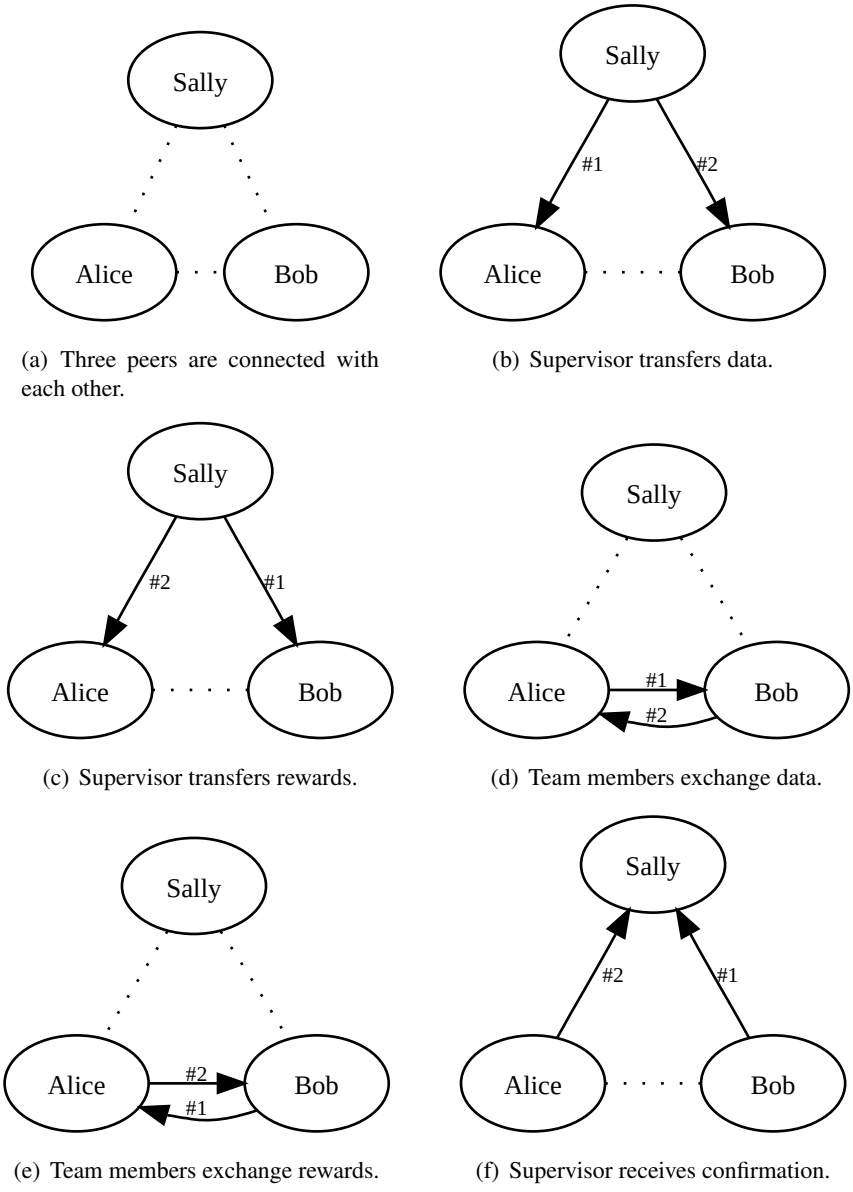
When the solution is simple, God is answering.
- Albert Einstein

In order to verify the performance of the Supervised Teaming protocol, we have implemented a peer-to-peer client that is able to use both BitTorrent and Supervised Teaming. However, we decided to distinguish between a *light* and an *extended* version of the Supervised Teaming protocol. The difference between these versions is that the Light Supervised Teaming protocol, which is discussed in this chapter, is restricted to a team size of two, whereas the Extended Supervised Teaming protocol, which is discussed in the following chapter, is able to handle teams of any size. The experiments that are performed with the BitTorrent and Light Supervised Teaming protocols are presented in Chapter 5.

Section 3.1 explains how the Light Supervised Teaming protocol works and which messages are used to communicate between the peers. Section 3.2 discusses how efficient the protocol is in terms of the transport efficiency, the time efficiency, and the sharing ratio. And finally in Section 3.3 we assess how the Light Supervised Teaming protocol handles flash crowd, free riding, and initial risk problems.

3.1 Protocol description

The problems that are presented in Section 1.1, flash crowds, free riding, and initial risk, can be partially solved by increasing the upload efficiency and by reducing risk. These two goals, *efficient uploading* and *risk reduction*, form the basis for Supervised Teaming.



(a) Three peers are connected with each other.

(b) Supervisor transfers data.

(c) Supervisor transfers rewards.

(d) Team members exchange data.

(e) Team members exchange rewards.

(f) Supervisor receives confirmation.

Figure 3.1: *The strategy of the Light Supervised Teaming protocol.*

3.1.1 Protocol design

Reducing the risk that a peer takes while cooperating with other peers in the community is achieved in two ways. The first is a tried and proven approach, namely, to cut the data into smaller blocks and upload one block at a time. When a peer feels—for any reason—that she is being cheated, she stops uploading. For the second approach we have to realize that, after receiving data, a peer is usually not required—and can certainly not be forced—to upload any data in return. Supervised Teaming will reduce this risk by making sure that a peer is unable to use a received block of data without a special key. This key, in the form of a reward, provides an incentive to forward a block of data.

The second aspect of Supervised Teaming is that the transport efficiency for altruistic peers is increased. This increased efficiency must logically come at the expense of the downloading peers. However, this requires a strong incentive or otherwise the downloading peers will not cooperate. Traditionally only two peers cooperate, up and download, with each other. However, this is not always possible. Not all peers have data that they can exchange and seeders have no interest in downloading any data at all. To solve this dilemma, Supervised Teaming allows more than two peers to cooperate with each other.

To incorporate these two aspects, the Supervised Teaming protocol will use small groups, consisting of a supervisor and a team with two team members, who are committed to the distribution of a single piece. The supervisor, who is required to have completely downloaded this piece, is given authority over the team. Furthermore, the team members, who are attempting to download this piece, must be willing to contribute some of their upload bandwidth.

Once a team is created and all the team members are connected to the supervisor and each other, the supervisor divides the piece into smaller blocks and starts transferring them to the team members. However, the supervisor does not reveal the offsets of the blocks. By giving the offsets to the opposite team member, who will only reveal this information after receiving this block in a forward, an incentive is created to forward the blocks between the team members. The supervisor will only send the next block when it is confirmed that the previous block has been forwarded. This is repeated until the entire piece is distributed among the team members. This strategy is illustrated in Figure 3.1 with the transfer and forward of two blocks from the supervisor Sally to the team members Alice and Bob.

Using this strategy, the supervisor has to upload every block only once while these blocks are received by two team members. This effectively doubles the transport efficiency for the supervisor. Furthermore, because the forward of every block must be confirmed, the initial risk is reduced to the size of a single block. Because a team member is unlikely to benefit from downloading a block without actually forwarding it, we can assume that the initial risk is actually much lower.

It should be noted that any peer can act as a supervisor for any piece that she has completely downloaded. Furthermore, a peer can be a team members in any

number of teams at the same time.

3.1.2 Message flow

A description of the messages that are used to accomplish the above strategy is given in this section. Figures 3.2 and 3.3 show the message flows for the supervisor and the team members, respectively. A detailed description of the messages that are used, including size and payload, are given in Appendix A.2.

The Light Supervised Teaming protocol consists of three distinct phases. First the information phase, second the setup phase, and third the transfer phase. We will now discuss these three phases.

Information phase

Light Supervised Teaming uses a *pushing* strategy, meaning that when a peer is willing to share a certain piece, she will contact other peers to see if they are willing to receive it. In order to do this, each peer must know what pieces the other peers are interested in.

For example, when Sally has completed a piece she can contact two other peers who are interested in this piece and ask them if they want to be in a team with her as supervisor and them as team members.

To know which peers are interested in which pieces, the messages `TEAM INTERESTED`, `TEAM NOT INTERESTED`, and `TEAM INTERESTED FIELD` are used.

Setup phase

The setup phase starts when a potential supervisor has discovered two potential team members. Each of these potential team members is contacted using `TEAM REQUEST` messages. The supervisor waits until either both team members have given a reply, or until a timeout has occurred.

For the team members the setup phase starts when they receive a `TEAM REQUEST` message. Because all team members need to be connected to each other, the team members will not reply until these connections are established. Depending on the connectability of the team members, the team request is accepted or declined by sending a `TEAM REPLY` message back to the potential supervisor.

When the potential supervisor receives accepting messages from both team members, she can start to prepare the piece for transfer. The piece is divided into blocks, and each block is given a unique and random identifier. The blocks are then randomly and evenly assigned to the team members. When the number of blocks is uneven, one of the team members is chosen randomly and is assigned one additional block.

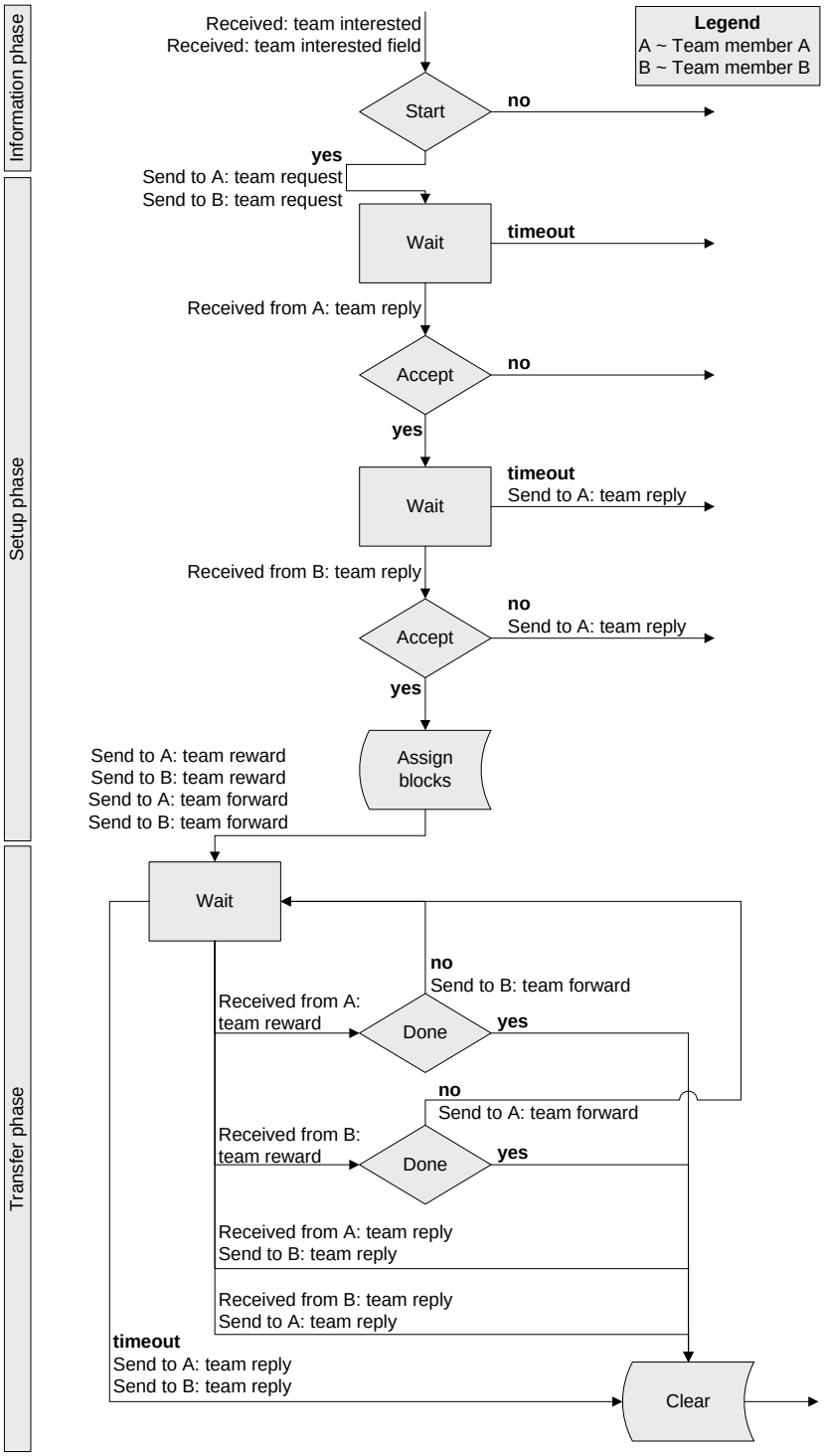


Figure 3.2: *The flow diagram for supervisors.*

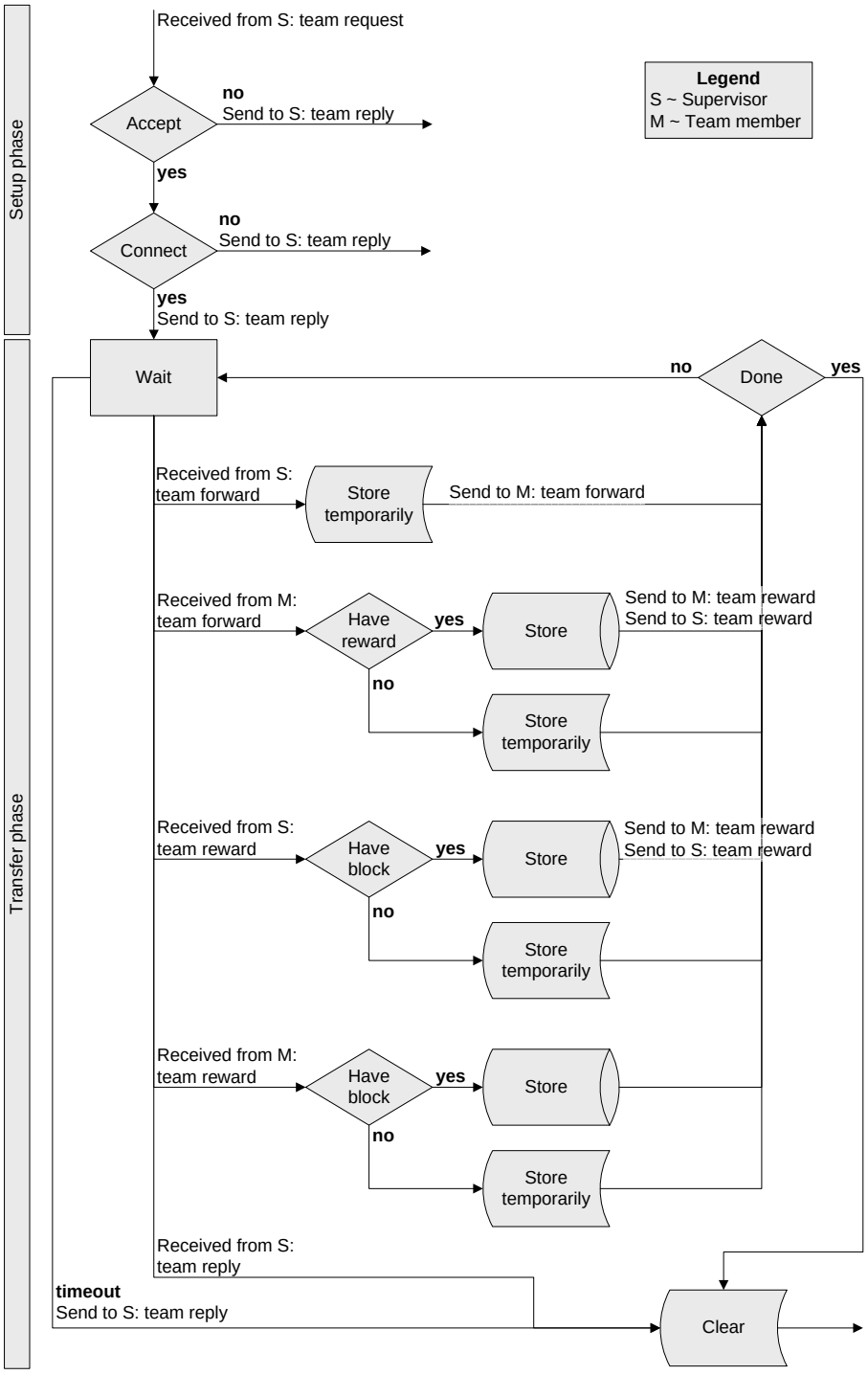


Figure 3.3: *The flow diagram for team members.*

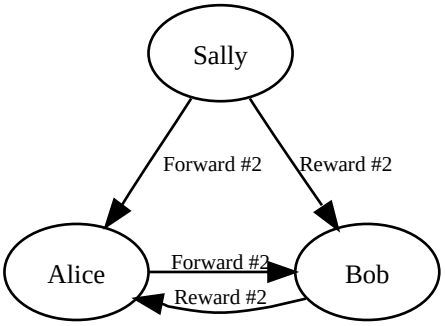


Figure 3.4: *The message flow between a supervisor and two team members.*

In the last step of the setup phase, the supervisor sends a `TEAM REWARD` and `TEAM FORWARD` message to each team member. The reward message provides the offset for one or more blocks, and the forward message contains the data for a single block. Each team member is sent the rewards for the blocks that are assigned to the other team member. For example, supervisor Sally has assigned block #2 to team member Alice. Thus Sally will send the data for block #2 to Alice, and the reward for this block to Bob who is the other team member; see Figure 3.4.

Transfer phase

The transfer phase consists of a simple algorithm that repeats until either all the blocks have been sent, or something goes wrong. The start of this phase is initiated with the first `TEAM FORWARD` messages that the supervisor sends to the team members.

A `TEAM FORWARD` message, because it consists of only an identifier and a stream of bytes, is in itself useless to a team member. The `TEAM REWARD` message with the same identifier contains the byte offset for that block, so only after receiving both the `TEAM FORWARD` and the `TEAM REWARD` messages, can a team member store the data.

Therefore, upon receiving a `TEAM FORWARD` message from the supervisor, this message is forwarded to the other team member. When this message is received, the associated `TEAM REWARD` is sent back. Furthermore, a confirmation message, in the form of a `TEAM REWARD` is also sent to the supervisor to indicate that the data was properly forwarded. This provides the team members with an incentive to forward the data because otherwise they will not receive the reward.

When the supervisor receives the confirmation message, she can send the next `TEAM FORWARD` message containing a new block. This process repeats itself until all the blocks have been transfered.

At any time during the transfer phase a timeout or disconnect can occur. When this occurs at a team member, the supervisor is notified with a `TEAM REPLY`

message. When this occurs at the supervisor, the remaining team members are notified with a `TEAM REPLY` message indicating that the team will be disbanded.

3.2 Performance analysis

The Light Supervised Teaming protocol is designed to be efficient for uploading to other peers, and to reduce risk when transferring data to other, possibly unknown, peers. To achieve this, several control messages are used. In this section we will determine how the messages and their payload affect the efficiency of the Light Supervised Teaming protocol. As with the BitTorrent protocol, we will express the performance using the transport efficiency, the time efficiency, and the sharing ratio which are defined in section 2.3.

3.2.1 Transport efficiency

The setup for the transport efficiency calculations are the same as previously for the BitTorrent protocol in Section 2.4.1. Thus we are dependent on connection failures, deviations from the protocol, the size of pieces, and the size of blocks; and we will not take into account the connection failures and deviations from the message flow as it is presented in Section 3.1.2.

In contrast to the BitTorrent protocol, where both up and downloading peers share the same transport efficiency, the Light Supervised Teaming protocol results in different efficiencies for supervisors and team members. It is even possible that both team members have different transport efficiencies. Such a difference occurs when a piece is divided in an uneven number of blocks or when, because of block alignment, one block has a different size. In either of these cases it is impossible to evenly divide the forwarding load between the team members, therefore, we distinguish between a lucky and an unlucky team member.

Figure 3.5 shows the transport efficiency for the lucky and unlucky team members, and the average between these two for the transfer of a 4MB piece with different block sizes. Fortunately, the supervisor can almost always select the block size in such a way that it aligns with the piece size. The only likely circumstance when a difference between lucky and unlucky team members occurs is with the last piece in the torrent. Therefore we simplify our analysis by using the average transport efficiency of the lucky and unlucky team members.

An efficiency prediction for the transfer of a single 4MB piece from a supervisor to two team members using 32kB blocks can be seen in Table 3.1. This table shows the various messages that are sent during the transfer and their respective sizes including the five byte message headers. When we look at the bottom of the table we can see that the resulting efficiency for the supervisor is a little under 200% which indicates that the benefit—two peers have received 4MB worth of data—for the supervisor is almost twice as high as her costs.

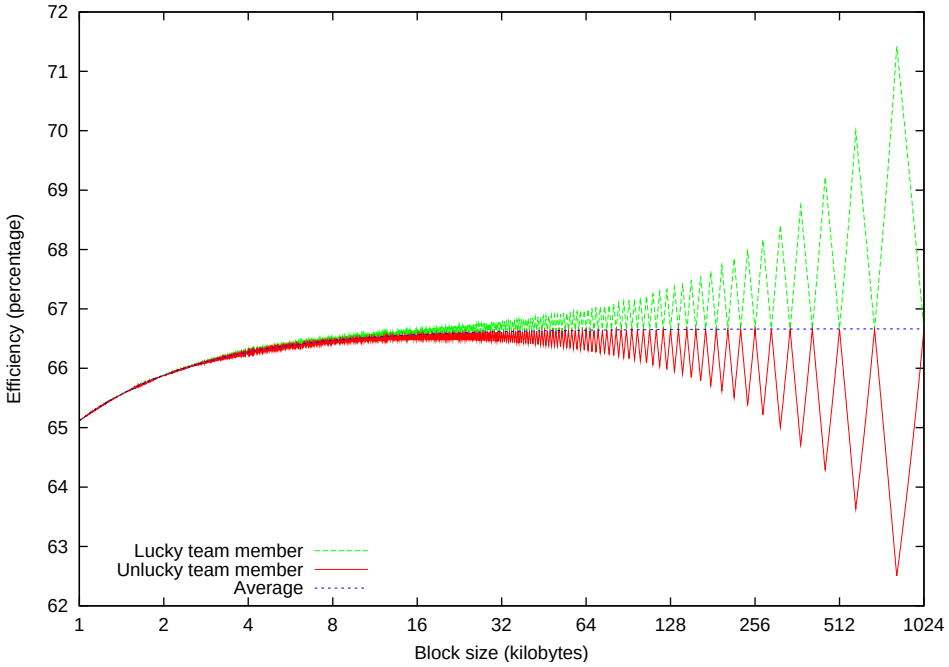


Figure 3.5: *The transport efficiency for lucky and unlucky team members. [P=4MB]*

Message	Supervisor Upload (bytes)	Supervisor Download (bytes)	Average team member Upload (bytes)	Average team member Download (bytes)
Information phase				
`TEAM INTERESTED`	0	18	9	0
Setup phase				
`TEAM REQUEST`	78	0	0	39
`TEAM REPLY`	0	20	10	0
`TEAM REWARD`	658	0	0	329
Transfer phase (x64)				
`TEAM FORWARD`	4,195,584	0	2,097,792	4,195,584
`TEAM REWARD`	0	1,280	1,536	896
Total				
	4,196,320	1,318	2,099,347	4,196,848
Efficiency				
Total bandwidth	4,197,638		6,296,195	
Overhead	-4,190,970		2,101,891	
Transport efficiency	**199.84%**		**66.62%**	

Table 3.1: *The transport efficiency. [P=4MB, B=32kB]*

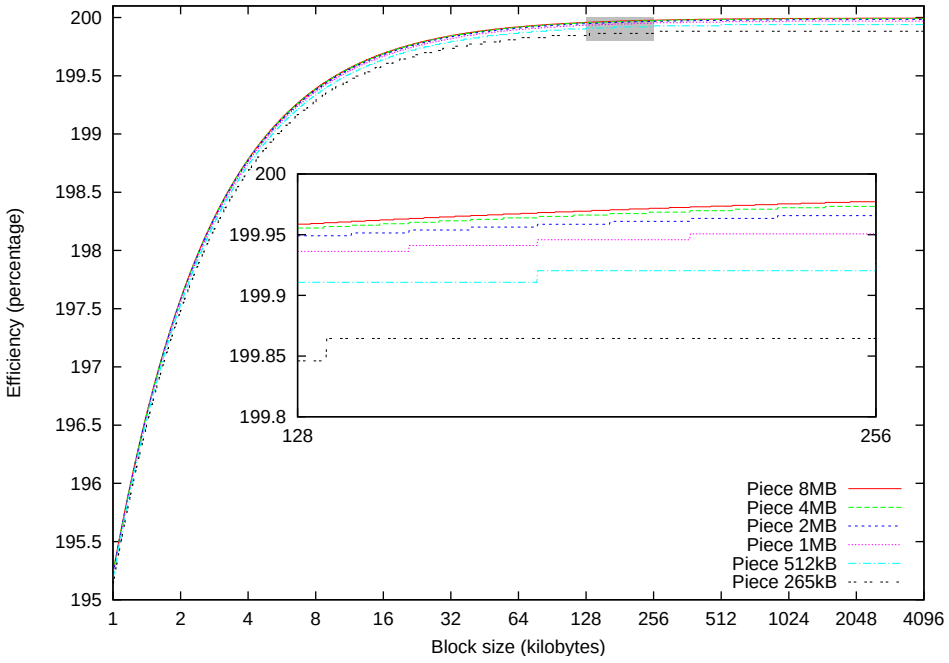


Figure 3.6: *The transport efficiency for supervisors.*

The table further shows that the efficiency for the team members is calculated at roughly 66%. With the 66% efficiency, the Light Supervised Teaming protocol compares badly with the nearly 100% efficiency that the BitTorrent protocol provides. This is the result of forcing the team members to upload while they download, increasing their own bandwidth costs to reduce the bandwidth cost of the supervisor.

The efficiency result that is shown in Table 3.1 can be generated for many more piece and block size combinations. By collecting the results of several combinations we can generate an overview of the transport efficiency for the Light Supervised Teaming protocol under different circumstances. These overviews are shown in Figures 3.6 and 3.7 for the supervisor and team members, respectively. In these figures we can see that the piece size has little or no effect on the transport efficiency. We can also see that when we increase the block size, the transport efficiency is also increased. Unfortunately we can not increase the block size indefinitely, we are always limited by the piece size, and the number of blocks that would result. Having only a few blocks will increase the chance that a peer can guess the reward, instead of having to upload her share to earn it.

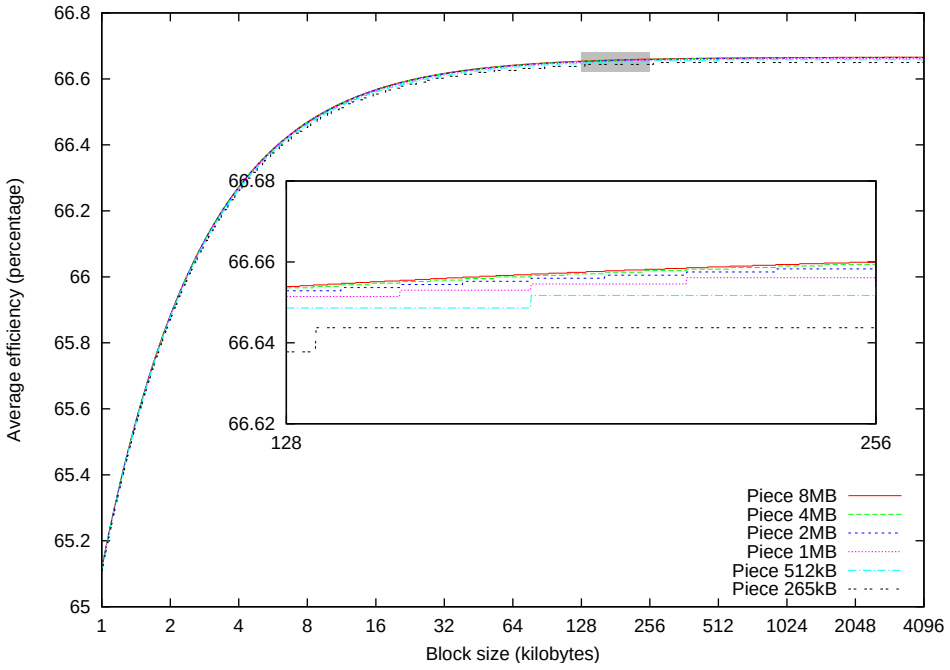


Figure 3.7: *The transport efficiency for team members.*

3.2.2 Time efficiency

The setup for the time efficiency calculations are the same as with the BitTorrent protocol in Section 2.4.2. Thus we are dependent on connection failures, deviations from the protocol, size of pieces, size of blocks, latency of the connection, and bandwidth of the connection; and we will not take into account the connection failures and deviations from the message flow as it is presented in Section 3.1.2.

Table 3.2 shows the messages with the respective latency and bandwidth time that is required when transferring a 4MB piece in 32kB blocks, having a 200ms latency and a bandwidth of 30kB/s. To clarify, we will explain the difference between the two `TEAM FORWARD` messages. The first `TEAM FORWARD` in the table accounts for the transfer from the supervisor to the team members. This transfer takes longer because two transfers are occurring at the same time. The second `TEAM FORWARD` accounts for the exchange between team members. This exchange is much faster because the team members are uploading to each other, allowing both their upload bandwidth to be used.

Assuming that the optimal theoretical transfer of 4MB to two peers takes 136 seconds at 30kB/s—this optimal transfer can be achieved by having the seeder transfer to the first leecher, and having the first leecher transfer to the second leecher as each byte is received—then we can calculate the efficiency for the transfer with

Message	Latency time (milliseconds)	Transfer time (milliseconds)	
Information phase			
`TEAM INTERESTED`	200	0.29	
Setup phase			
`TEAM REQUEST`	200	2.54	
`TEAM REPLY`	200	0.33	
`TEAM REWARD`	200	21.42	
Transfer phase (x64)			
`TEAM FORWARD`	12800	136575.00	
`TEAM FORWARD`	12800	68287.50	
`TEAM REWARD`	12800	20.83	
Total			
	39200	204907.91	
Efficiency			
Total in seconds	39.20	204.91	**244.11**
Overhead in seconds	38.80	68.37	**107.17**
Time efficiency			**56.10%**

Table 3.2: *The time efficiency. [P=4MB, B=32kB, L=200ms, SS=SL=30kB/s]*

the Light Supervised Teaming protocol. The resulting efficiency of 56.10% is shown at the bottom of the table. The decrease in efficiency, compared to the BitTorrent protocol, is caused by latency from additional control messages and the forward to the other team member.

When we generate the time efficiency values for several piece, block, latency, and bandwidth combinations we get the overview presented in Figure 3.8. Similar to the results for BitTorrent the efficiency decreases as the bandwidth increases. However, we will focus on the more intuitive transfer time shown in Figure 3.9 where the transfer of a 4MB piece in 32kB blocks is presented.

While the latency has almost no effect on the BitTorrent protocol, see Figure 2.5, it does have a significant effect on the Light Supervised Teaming protocol. The risk reducing strategy of Light Supervised Teaming dictates that the supervisor must wait with the transfer of the next block until the appropriate confirmation message has been received. While this does reduce the risk that is taken, it is the cause of the additional latency delay. However, because a peer is allowed to be a team member in several teams at the same time, the drop in bandwidth usage should be filled with the bandwidth for another team.

3.2.3 Sharing ratio

The sharing ratio, which is presented in this section, is used for the comparison between the BitTorrent protocol and the Light Supervised Teaming protocol.

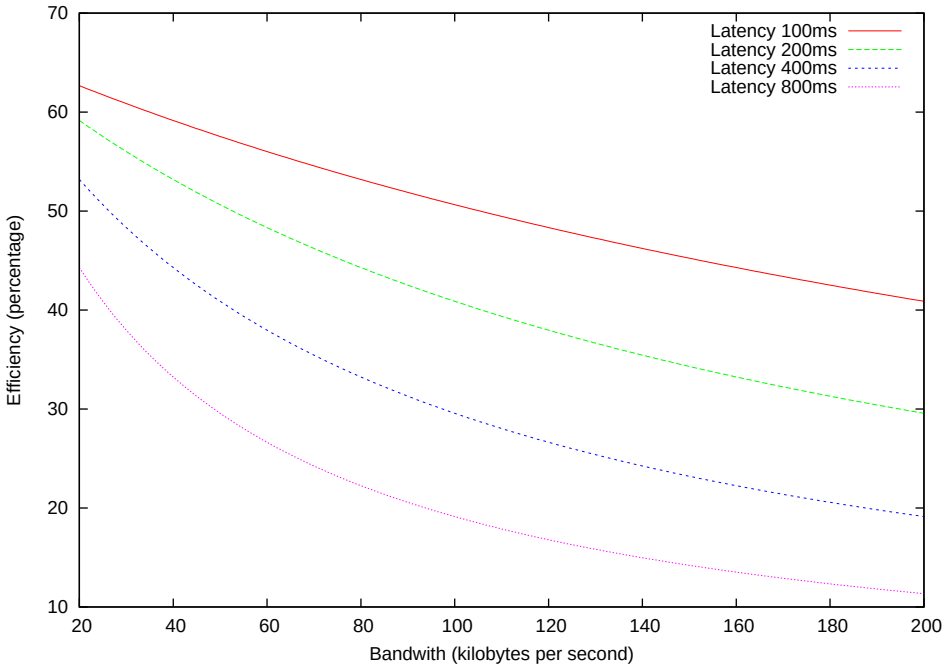


Figure 3.8: *The time efficiency. [P=4MB, B=32kB]*

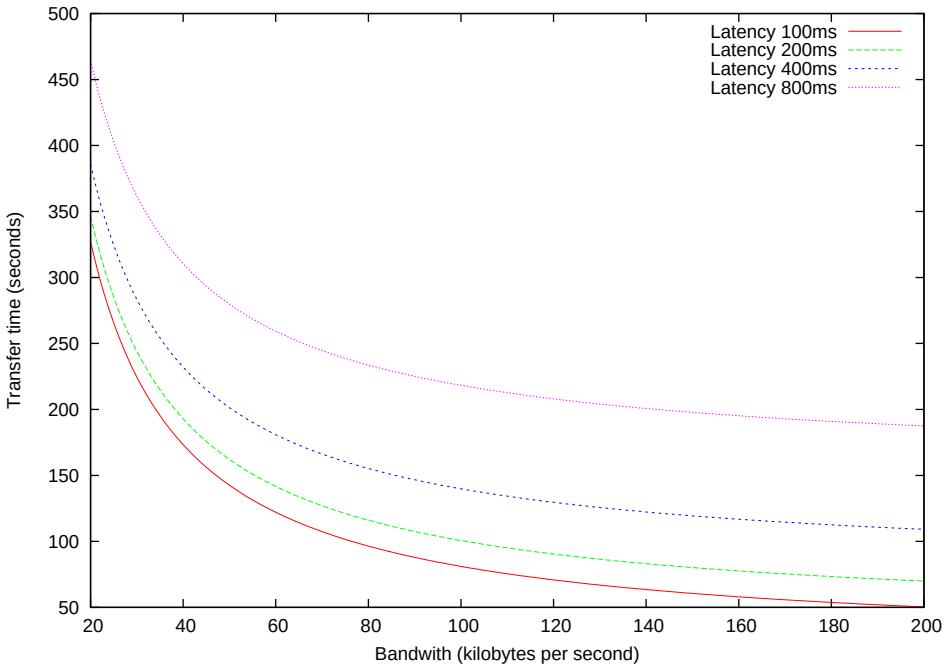
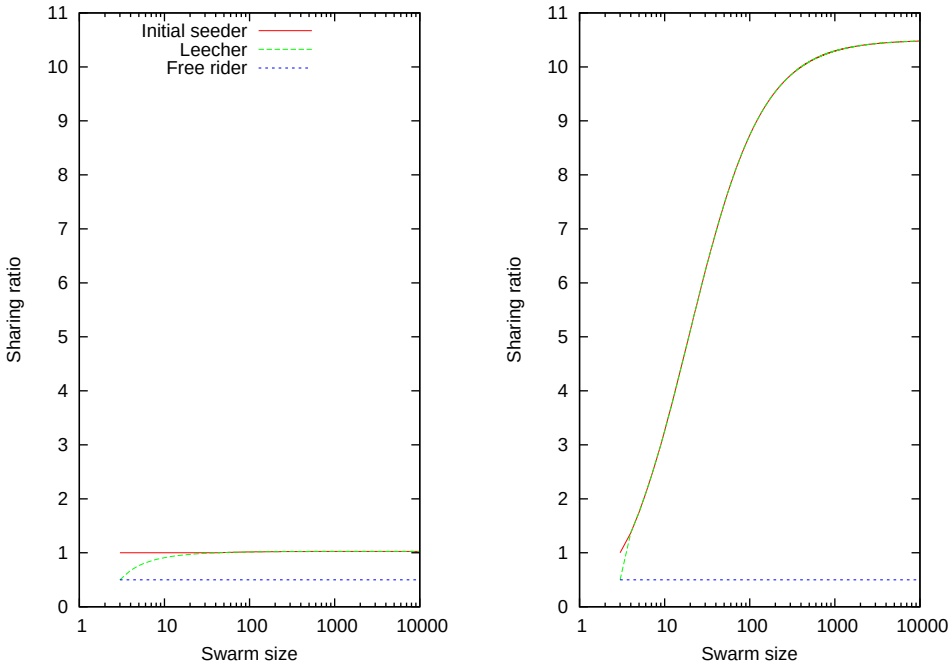


Figure 3.9: *The transfer time. [P=4MB, B=32kB]*



(a) Best-case scenario with 5% free riders.

(b) Worst-case scenario with 95% free riders.

Figure 3.10: *The sharing ratio. [F=100MB, NS=1, SS=SL=SF=128kB/s]*

Furthermore, it is used as a baseline for the experiments that are done in Chapter 5. Because of the simplifications, which are discussed in section 2.3.3, the swarm efficiency can be expressed by the following equations:

$$T = \frac{F \times (NL + NF) - 0.5 \times F \times NF}{SS \times NS + SL \times NL} \tag{3.1}$$

$$RS = \frac{SS \times T}{F} \tag{3.2}$$

$$RL = \frac{SL \times T}{F} \tag{3.3}$$

$$RF = \frac{0.5 \times F}{F} \tag{3.4}$$

However, these equations do not take into account that the initial seeder is required to upload the entire file at least once. Therefore, these equations are only valid when $SS \times NS \times T \geq F$. The figures that are presented in this section are generated with slightly different equations that take this into account.

The strong aspects of peer-to-peer networks—that they scale with the number of peers that participate in the network—is also present in the Light Supervised Teaming protocol. Figure 3.10(a) shows that the sharing ratio for the initial seeder

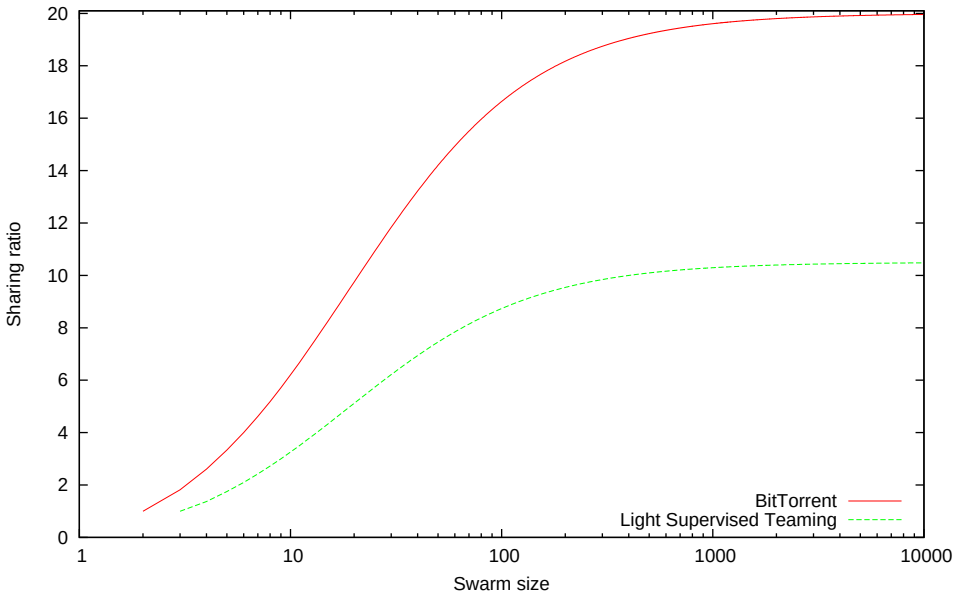


Figure 3.11: *Comparing the sharing ratio for the initial seeder in a worst-case scenario for Light Supervised Teaming and BitTorrent. [F=100MB, NS=1, SS=SF=128kB/s]*

and the leechers stays around 1:1. Because these figures were generated using the same parameters as those for BitTorrent in Figure 2.6, we can compare them to each other. Therefore, we can conclude that for the best-case scenario, where only 5% of the swarm consists of free riders, the Light Supervised Teaming protocol performs equally well as BitTorrent as the swarm size increases.

Figure 3.10(b), which shows the worst-case scenario where 95% of the swarm consists of free riders, shows that the sharing ratio for the initial seeder and the leechers approaches 1:10.5 as the swarm size increases. While this sharing ratio is high, it is almost half that of the sharing ratio for BitTorrent under similar circumstances. Therefore, in a worst-case scenario, the Light Supervised Teaming protocol performs roughly twice as good than the BitTorrent protocol.

This increased performance is caused by the relatively small amount of data that each free rider is forced to upload. This causes each free rider to have a sharing ratio of 1:0.5 and allows the initial seeder to save a considerable amount of bandwidth. Figure 3.11 shows the sharing ratios that a single initial seeder can expect in the worst-case scenario when 95% of the swarm consists of free riders. From this figure it is clear that Light Supervised Teaming performs twice as good than BitTorrent.

3.3 Concerning the problem statement

In Section 1.1 we presented the problems that we want to address in this thesis. This section presents the solutions that the Light Supervised Teaming protocol brings to these three problems.

Flash crowd. The problem with flash crowds can be solved by transferring all the data from the initial seeder into the community as quickly and as evenly as possible. Because the Light Supervised Teaming protocol has a high transport efficiency—from the supervisors point of view—and transfers the data to more than one peer at the same cost, it is ideally suited to solve the flash crowd problem.

As stated before, at the cost of uploading a single piece once, the data is duplicated to two peers. Effectively doubling the upload for the supervisor, thereby doubling the chance that either of the two peers will act as a supervisor for that piece themselves. Unfortunately the time efficiency of the Light Supervised Teaming protocol is lower than that of the BitTorrent protocol. However, this loss can be minimized by acting as a supervisor for more pieces at the same time and/or acting as a team member in more then one team at the same time.

Furthermore, the super seeding strategy that BitTorrent uses to allow seeders to more evenly distribute the data is inherent to the Supervised Teaming protocol, as it uses a pushing strategy instead of a pulling strategy.

Free riding. The problem of free riding is not solved. solving free riding is, in our opinion, impossible [23] with the current internet infrastructure. However, enforcing two team members to cooperate with each other will ensure that even free riders have a minimal sharing ratio of 1:0.5, whereas with the BitTorrent protocol, a free rider could manage a sharing ratio of 1:0.

Similar to BitTorrent, a peer has no direct benefit from being a seeder, or in the case of Supervised Teaming a supervisor. Therefore, a peer that acts like one is considered to behave altruistically. However, peers can maintain a credit count and use this to prefer peers in their teams that have acquired the most credits. This credit count can either be based on a global—and thus insecure—value or a local—and thus incomplete—value. We believe that a perfect peer-to-peer system is impossible, enforcing a 1:0.5 sharing ratio is a step forward. The protocol presented in the following chapter will allow us to further increase this sharing ratio.

Initial risk. Uploading in a peer-to-peer system is usually done because the uploader hopes to get something in return. However, because of inherent limitations to the internet, it is impossible to ensure that any bandwidth is returned at all. Some peer-to-peer systems try to solve this with certified credits, or with a globally maintained history, and others will only look at personal interaction with fellow

peers. In the end, each technique can only hope to minimize the risk that is taken when helping someone else. This is also the case with Light Supervised Teaming.

When all peers follow the protocol correctly a piece is transferred. In this case the supervisor will get nothing except perhaps, if a team member remembers, something in the future. However, when one of the team members stops forwarding there will be the risk of bandwidth loss. The supervisor risks losing two blocks worth of bandwidth—one for each team member—and each team member risks losing one block worth of bandwidth.

Thus, from the supervisors point of view it will be rewarding to have each peer remember who has been her supervisor, so they can attempt to get them as their team member in the future. This allows for the same tit-for-tat strategy that is used in BitTorrent. With the exception that not one, but two peers can remember the supervisor. Doubling the supervisors chance of being invited back as a team member.

Chapter 4

Extended Supervised Teaming

Aim at perfection in everything, though in most things it is unattainable. However, they who aim at it, and persevere, will come much nearer to it than those whose laziness and despondency make them give it up as unattainable.
- Lord Chesterfield

One of the most important aspects of the Supervised Teaming protocol is the ability to increase the transport efficiency for altruistic peers. The previous chapter achieved roughly 200% efficiency for such peers by having them supervise teams consisting of two team members. In this chapter we will present the Extended Supervised Teaming protocol where larger teams are allowed, thereby increasing the transport efficiency further.

Section 4.1 discusses the challenges that the Extended Supervised Teaming protocol faces because of the increased team size. Section 4.2 discusses how efficient the protocol is in terms of the transport efficiency, the time efficiency, and the sharing ratio. And finally in Section 4.3 we assess how the Extended Supervised Teaming protocol handles flash crowd, free riding, and initial risk problems.

4.1 Protocol description

In the design of the Light Supervised Teaming protocol we deliberately limited the team size to two. This simplification allowed us to simply disband the team whenever something went wrong. However, the Extended Supervised Teaming protocol allows larger teams, and disbanding an entire team because of a problem with a single team member is no longer an option. This section will present several issues that Extended Supervised Teaming will therefore have to deal with, and will describe how it affects the protocol.

4.1.1 NAT and firewall traversal

The Supervised Teaming protocol requires that the supervisor and all the team members are connected with each other. In terms of connectability, this means that only the supervisor or one of the team members can have a Network Address Translation (NAT) or firewalled connection. How many peers in a peer-to-peer system are unreachable depends on the social makeup of the swarm, measurements [6, 16, 20, 25] indicate that 25% to 76% of the peers are connectable.

Having non connectable peers is disastrous for the Supervised Teaming protocol. However, there are several options available that alleviate this problem:

- Use one of the many NAT and firewall traversal techniques [10, 12].
- Instead of communicating with TCP sockets use UDP messages. Several techniques are available that allow UDP messages to pass through a NAT or firewall [10, 12, 22].
- The IPv6 protocol [11], that should become the standard internet communication protocol, promises to solve NAT traversal problems by supplying each individual computer with a unique address.
- Reducing the team size will increase the chance for an unconnectable peer to find a team which has no unconnectable supervisor or team members so far.

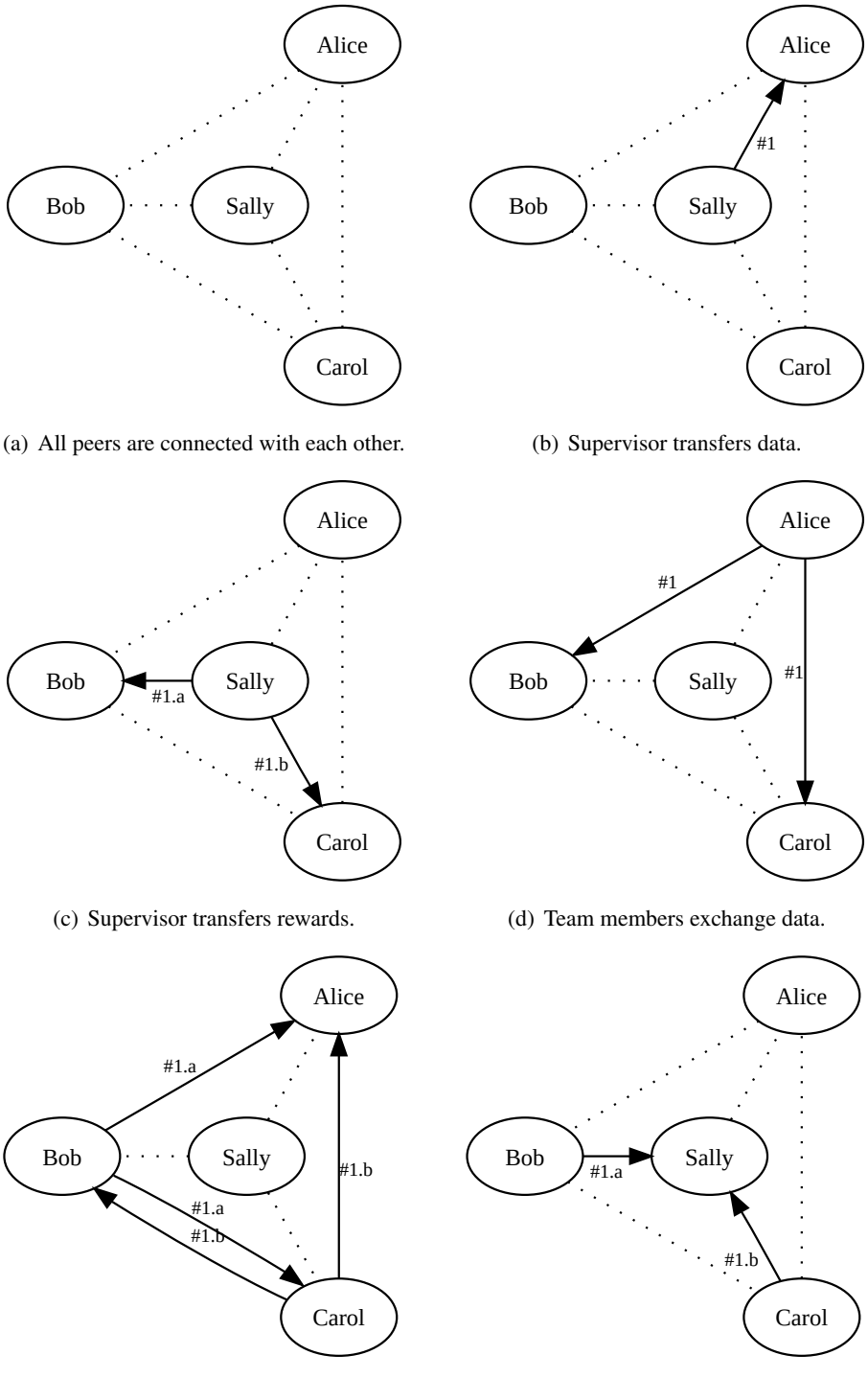
While NAT and firewall traversal may be the biggest drawback of Supervised Teaming we believe that the technical aspects that are involved in solving NAT and firewall traversal are out of the scope of this thesis.

4.1.2 Forwarding incentive

The forwarding incentive that is used in Light Supervised Teaming—giving one team member the reward, and the other the data—will not function with larger teams. In Extended Supervised Teaming a team member requires an incentive to forward the data to all the other team members. Therefore, instead of one reward, each team member, except the one that is forwarding, will have part of the reward. Similar to the Light Supervised Teaming protocol, we choose to use the data offset as the reward, dividing this reward between the team members can be accomplished by giving each a value that, when added together, results in the correct offset. For example, dividing a reward for offset 1024 between 3 peers can be done by having one reward set to 1280, the second set to 768, and the third to -1024.

The supervisor should not be predictable when dividing a block offset into rewards. For example, dividing the offset uniformly, where 1024 becomes 341, 341, and 342, makes it possible to guess the offset using only a single reward. However, this thesis will not discuss the benefits of different dividing techniques.

Figure 4.1 presents the different steps that are part of the Extended Supervised Teaming protocol. To reduce the number of links in this figure, only the forward



(a) All peers are connected with each other.

(b) Supervisor transfers data.

(c) Supervisor transfers rewards.

(d) Team members exchange data.

(e) Team members exchange rewards.

(f) Supervisor receives confirmation.

Figure 4.1: *The strategy of the Extended Supervised Teaming protocol.*

from a single team member, called Alice, is displayed. However, it should be noted that Bob and Carol also receive a block of data from Sally and that they are expected to forward their data to the other team members as well.

Giving each team members part of the reward ensures that team members have an incentive to forward the data to all the other team members. However, it does require that additional `TEAM REWARD` messages are sent during transfer. How this affects the efficiency of the Extended Supervised Teaming protocol is discussed in Section 4.2.

4.1.3 Team size

The efficiency of the Supervised Teaming protocol increases with the team size. However, there are certain limitations to the team size. In an old swarm there will be relatively few new peers, the older peers will have already downloaded pieces that this new peer still requires. However, because there are no or only a few other peers that also require these highly available pieces, forming a large team will not be possible. Therefore, smaller teams—perhaps even a team size of one—should be allowed. Incidentally, when the team size is one, the supervisor will be uploading to a single team member which is effectively the same as using the BitTorrent protocol.

The optimal team size depends largely on the progress of all the peers in the swarm. Creating a large team in a recently created swarm will be both efficient and easy because the availability of pieces is low and we can therefore expect peers to put more effort into their downloads. However, as the swarm ages the availability will increase and the possibility to find team members that require the same pieces decreases. We will not give an exact value for the optimal team size, however, we believe that it is possible for peers to estimate this value based on the `TEAM INTERESTED` messages that are send between peers in a swarm.

In a swarm the altruistic peers will choose to supervise a team, thereby giving them the ability to decide how much effort the team members must put into obtaining their data. This gives uploading peers the ability to decide the effort that is required from downloading peers. A peer who is not willing to put this effort into her download can wait until more peers have acquired that data and the effort that they demand from their team members decreases.

4.1.4 Incomplete transfer

Until now we have assumed that a peer has either downloaded nothing or everything from a piece. Obviously this is not always the case. To allow the Extended Supervised Teaming protocol to efficiently handle this situation the team members must inform the supervisor of their download progress when they join the team. In certain situations a supervisor can use this information to reduce bandwidth usage.

For example, if a team member Alice informs supervisor Sally that she already has block #42, then Sally can assign the forward of block #42 to Alice, and refrain from uploading this block herself. If the supervisor is lucky enough she might find herself in a team where she does not have to upload any significant amount of data at all, the role of the supervisor then changes to simple delegation. In this case, the supervisor has to instruct which team members have to forward which blocks between each other. Because this can not be accomplished with regular `TEAM FORWARD` message, the `TEAM DELIVER` message is introduced. Instead of a block-id/data pair, this message contains a block-id/offset pair.

When one or more of the team members already have some parts of the piece, it is unnecessary for the other team members to forward their assigned blocks to these team members. To make this possible, the supervisor must inform the forwarding peer to which team members the block should be forwarded. Consequently, the supervisor should also take into account that only the peers that still require a block will actually receive part of the reward. To inform a forwarding peer of her targets a *delivery* bitfield is added to the `TEAM FORWARD` and `TEAM DELIVER` messages. This bitfield will contain one bit for each team member, where a set and unset will indicate a required forward or no forward, respectively.

The number of bytes that is used for this bitfield is directly influenced by the maximum size of a team. Therefore, the number of bytes that is assigned to this delivery bitfield depends on the team size. To allow each team member to determine this, the maximum team size is defined during the setup phase. Using this maximum, each team member knows how large the delivery bitfield will be.

4.1.5 Transfer speed requirements

Because a supervisor waits with the forward of new blocks until all the confirmation messages have been received, the team will progress at the speed of the slowest team member. However, these confirmation messages also allow the supervisor to estimate how much time it took a team member to forward her block, thereby indicating which team member is the so called *weakest link*.

To avoid a team member from slowing down a team, the Extended Supervised Teaming protocol must allow the supervisor to remove a team member while the remaining team members can continue with as little disruption as possible. Extended Supervised Teaming is able to achieve this by having the supervisor reallocate the blocks from the removed team member to the remaining team members.

Because team members can leave during the existence of a team, it becomes inadvisable to transfer all of the allocated rewards during the setup phase. Instead the rewards should be sent in separate `TEAM REWARD` messages as the transfer progresses, this will leave the removed peer with little or no information concerning the remaining blocks.

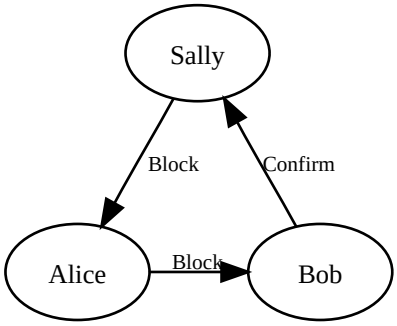


Figure 4.2: *Delaying a confirmation message.*

However, finding a weakest link is not an easy task because the estimated transfer speed depends on third party information, namely the team members who send the confirmation messages. This is shown in Figure 4.2 where Sally, our supervisor, sends a block to Alice. Alice forwards this block to Bob who sends the confirmation message back to Sally. The estimation that Sally makes can be influenced by both Alice and Bob. Alice can influence the speed of her forward by allocating more or less bandwidth to it, and Bob can influence the estimation by sending the confirmation message too soon or too late. Some of these four cases, when applied to a large team, will present Sally with an inaccurate estimation. Unfortunately it is impossible for Sally to blame either Alice or Bob because she in unaware of what happens between these two team members. When Bob is delaying confirmation messages this might even implicate himself as the delaying factor. However, this is also the case when Alice decides to decrease the transfer speed for the forward to Bob. Fortunately the knowledge of team member performance grows with each transferred and confirmed block, this can be used to slowly build trust between peers.

While transferring the entire torrent, a peer is likely to encounter many peers with different up and download speeds. These peers will also be active in several different teams. Furthermore, because each team progresses at the speed of its slowest member, it is unlikely that a team will have a high transfer speed. Therefore, it is likely that the overall transfer speed will not depend on the speed of a single team but on the number of teams in which a peer can participate. To allow peers to decide whether to accept or decline a request to join a team, the request should state the minimal bandwidth that should be available to join a team. Should the supervisor estimate that a peer falls below this requirement, the team member should be removed from the team.

However, participating in several teams at the same time, and with different teams having different transfer speed requirements, may be problematic with a traditional transfer protocol where a single message is sent through a connection at a time. For example, Alice and Bob are both part of the same two teams, illustrated in

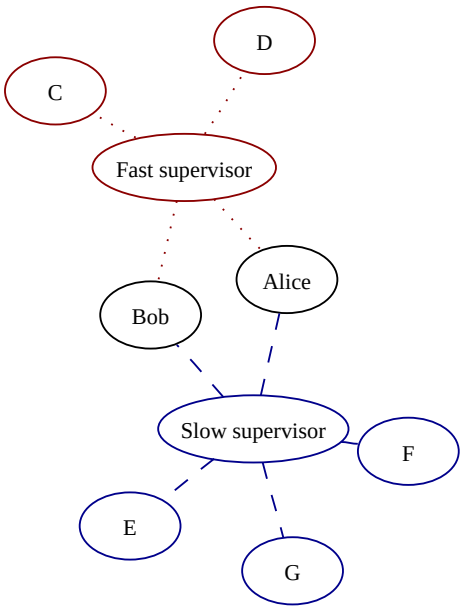


Figure 4.3: *Teams with different transfer speed requirements.*

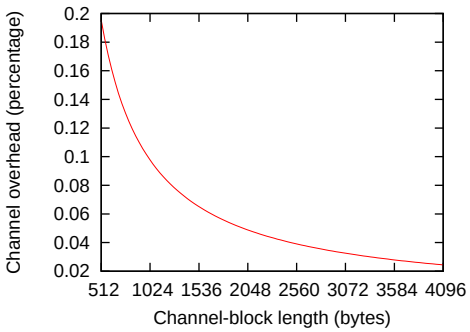


Figure 4.4: *The additional transport overhead by switching channels every n bytes.*

Figure 4.3. The first team requires a transfer speed of 4kB/s and the second team requires 1kB/s. Because Alice has an available upload bandwidth of 5kB/s she decided that she can join both teams. Using her available upload bandwidth it takes Alice 6.5 seconds to transfer a 32kB block to Bob. The supervisor for the fast team expects a forward to take 8 seconds and the supervisor for the slow team expects a forward to take 32 seconds. If the message for the slow team is sent along the connection first, this will arrive well within the 32 second limit. The second message, containing the forward for the fast team, will arrive another 6.4 seconds later, effectively after 12.8 seconds. Given that it was promised to take no more than 8 seconds, it was delivered 4.8 seconds behind schedule.

This problem can be solved by allowing both messages to be transferred through

the connection in parallel. However, this requires a change in the transfer protocol. We propose to divide each connection into 256 channels that can be used more or less simultaneously. Each message is split into channel-blocks that are n bytes long. Each channel-block is preceded with a single byte indicating one of the 256 channels. Now channel-blocks from different channels can follow each other, allowing semi-parallel communication. The order of the channel-blocks is based on the required transfer speed of the messages that the channel belongs to. For the example in the previous paragraph, where the one team is four times faster than the other, the density of the channel-blocks for the fast message are four times as high as the slow message. The resulting overhead that channels have on the communication depends on n and follow the formula: $O = 100/n$. Figure 4.4 shows that the resulting overhead is small.

4.1.6 Endgame

The BitTorrent protocol uses a so called endgame [7] strategy to increase the download speed in the final stage of the transfer. Using this strategy the remaining blocks are requested from several peers at the same time. This strategy can also be applied with Supervised Teaming, however, the cost is higher because the peer will need to be part of more than one team for the same piece.

Aside from this, the Supervised Teaming protocol has a second endgame problem. While the BitTorrent protocol uses a pulling mechanism where request messages are sent that indicate exactly what is required, the Supervised Teaming protocol uses a pushing mechanism where one peer asks other peers to join a team for a specific piece. However, the effectiveness of this pushing mechanism goes down as the availability of a piece goes up because many supervisors are requesting peers to join their teams. The following two examples show that the chance to form a team decreases as the availability of pieces increases.

In our first example we have a swarm with 1000 peers. In this swarm there is one seeder while the remaining 999 peers are leechers. When the seeder tries to create a team with two team members this will not be a problem because none of 999 leechers in the swarm are currently in a team and will all accept the invitation to join. In this case the chance to form a team is 100%.

In our second example we again have a swarm with 1000 peers. However, in this swarm there are 998 seeders who are trying to create a team with two team members. Now the two leechers will each receive 998 requests to join a specific seeder in her team. The chance that both the remaining peers select the same seeder is $1/998 \times 100 = 0.1002\%$.

This problem can be solved on two sides: at the potential supervisors and at the potential team members. Solving this problem on the side of potential team members can be accomplished by letting the potential team members discuss among each other to which `TEAM REQUEST` they prefer to respond positively. Unfortunately, this requires additional messages to be sent, further increasing

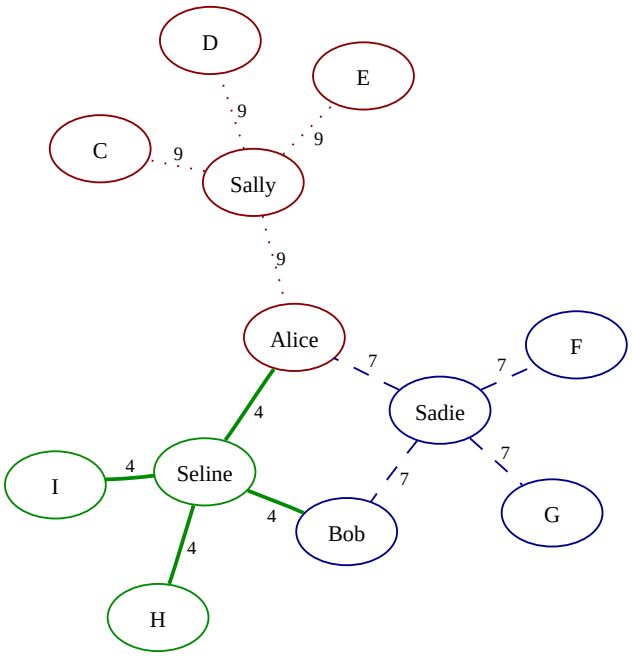


Figure 4.5: *Choosing teams using eager values.*

complexity and communication overhead. Therefore, we will focus on the supervisor side solution.

The potential supervisors can reduce the problem by coordinating the number of `TEAM REQUEST` messages that are sent to potential team members. One way to achieve this is to give supervisors a preference when creating a team. For example, if Sally has both piece #2 and #3 and both pieces have the same availability she can choose either piece #2 or #3 randomly or based on some mathematical function that uses her IP address or peer id.

Additionally a difference between supervisors can be created by including some value in the `TEAM REQUEST` message. This value can indicate how *eager* a peer is to supervise that piece. Peers that receive `TEAM REQUEST` messages from multiple peers can simply select the supervisor with the highest eager value. Note that this eager value should include, at least some, randomness to avoid peers in a similar situation to constantly have the same eager value. An example of this is given in Figure 4.5 where the three supervisors Sally, Sadie, and Seline attempt to create a team using eager values of 9, 7, and 5, respectively. Alice can choose between three offers and chooses Sally because she gave the highest eager value. Seline, with the lowest eager value, might not start her team because only two out of four peers replied positively to her request.

4.2 Performance analysis

This section will present the performance analysis for the Extended Supervised Teaming protocol. Similar to the analysis for the BitTorrent and Light Supervised Teaming protocols, we distinguish between transport efficiency, time efficiency, and sharing ratio.

Several assumptions are made for these performance analyses. The `TEAM MEMBER` message will send the IP address of the team members as an IPv4 address. None of the team members will have any part of the piece that they are downloading, therefore the `bitfield` in the `TEAM REPLY` message that is sent in response to the `TEAM MEMBER` message will be empty. We will use a 5 byte message header and we will not use the channels that are proposed in Section 4.1.5. Furthermore, instead of 32kB blocks—which were used for the performance analyses for both the BitTorrent and the Light Supervised Teaming protocols—we will use 16kB blocks to ensure that each team member will still have several blocks to trade, even with the increased team size.

4.2.1 Transport efficiency

The transport efficiency is calculated in the same way as in the previous two chapters. However, the Extended Supervised Teaming protocol has one additional variable that affects the efficiency of the protocol, namely the team size. And because the Extended Supervised Teaming protocol allows a team size that is higher then two we are not limited to one lucky and one unlucky team member, as is the case with the Light Supervised Teaming protocol. For example, transferring a 4MB piece to a size three team, in 14kB blocks will require roughly 292.6 blocks. This means that one *lucky* team member is required to forward 97 blocks, while the two other *unlucky* team members have to forward 98 blocks. However, one block is only 8kB large. Therefore, instead of forwarding 98 blocks, one unlucky team member is required to forward 97 14kB blocks and one 8kB block. We call this team member *less lucky*.

Similarly to the previous two chapters we calculate the transport efficiency under different piece, block, and team sizes to produce Figure 4.6. In this figure the efficiencies for lucky, less lucky, and unlucky team members is given for the transfer of a 4MB piece to a size three team. The figure shows that, as the block size increases, the difference in efficiency also increases. Having a smaller block size will result in more and smaller blocks that the team members are required to forward. This reduces the difference between lucky, less lucky, and unlucky team members. To keep the difference between the team members as small as possible the choice of block size—depending on the piece size and the team size—is very important. In the best-case the supervisor is able to both align the block to the piece size and evenly distribute the blocks among the team members. However, this may not always be possible. In the remainder of this chapter we will use the average

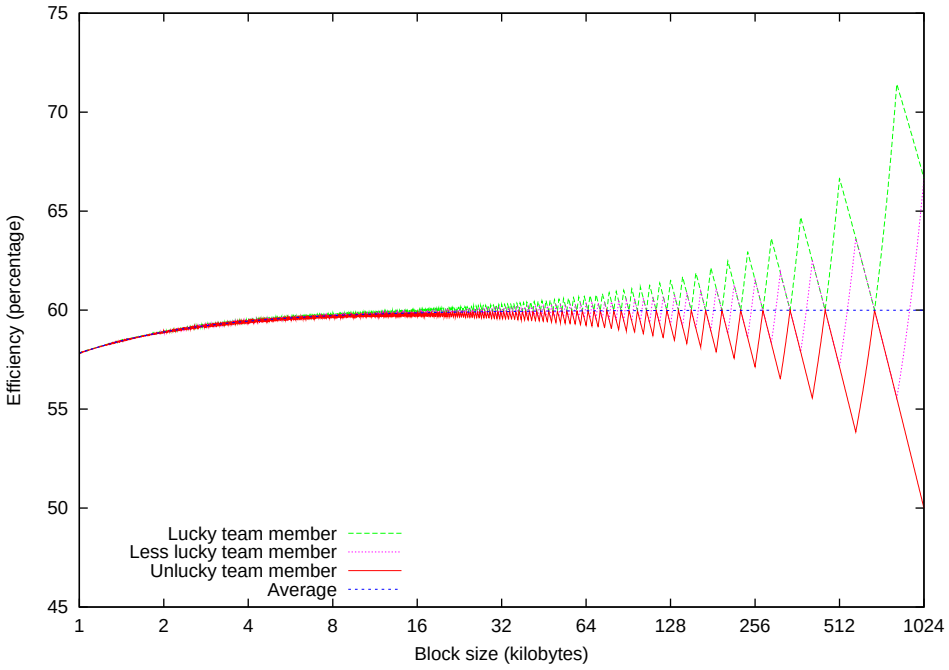
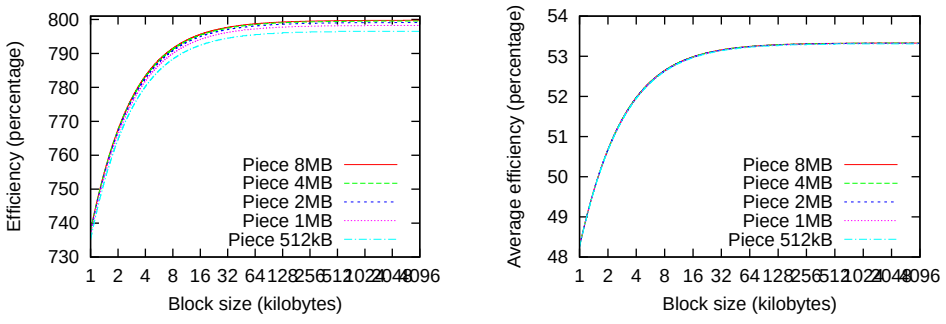


Figure 4.6: *The transport efficiency for lucky, less lucky, and unlucky team members. [P=4MB, M=3]*



(a) Supervisor. (b) Team member.

Figure 4.7: *The transport efficiency. [M=8]*

Message	Supervisor Upload (bytes)	Supervisor Download (bytes)	Average team member Upload (bytes)	Average team member Download (bytes)
Information phase				
`TEAM INTERESTED`	0	72	9	0
Setup phase				
`TEAM REQUEST`	168	0	0	21
`TEAM REPLY`	0	80	10	0
`TEAM MEMBER`	1,744	0	0	218
`TEAM REPLY`	0	80	10	0
`TEAM REWARD`	1,352	0	0	169
Transfer phase (x32)				
`TEAM FORWARD`	4,197,120	0	3,672,480	4,197,120
`TEAM REWARD`	0	17,920	24,192	21,952
Total				
	4,200,384	18,152	3,696,701	4,219,480
Efficiency				
Total bandwidth	4,218,536		7,916,181	
Overhead	-29,335,896		3,721,877	
Transport efficiency	**795.40%**		**52.98%**	

Table 4.1: *The transport efficiency. [P=4MB, B=16kB, M=8]*

efficiency instead of separate values for the lucky, less lucky, and unlucky team members.

To illustrate how we calculate the transport efficiency, we present Table 4.1 where the required bandwidth for the distribution of a 4MB file in 16kB blocks from a supervisor to a size eight team is given. this figure shows that the supervisor has, with 795.40%, a very high efficiency. This is caused by the difference between the benefit—eight peers receive a piece—and the cost—one piece is uploaded—is very high. The team members, with 52.98%, do most of the work. Their total bandwidth is almost twice the piece size.

The previous two chapters showed that the piece size barely affects the transport efficiency of either the BitTorrent or the Light Supervised Teaming protocol. This also holds for the Extended Supervised Teaming protocol. This is shown in Figures 4.7(a) and 4.7(b) where the efficiencies for the different piece sizes are shown for a size eight team and a varying block size. As with BitTorrent and Light Supervised Teaming the efficiencies overlap for the most part. Because this has already been discussed in the previous chapters we will not go into this any further, instead we will focus on the effect that the team size has on the transport efficiency.

We perform our efficiency calculation on the transfer of a 4MB piece. Because the Extended Supervised Teaming protocol is based on sharing between team members, it is essential that there are enough blocks. Therefore, in contrast to

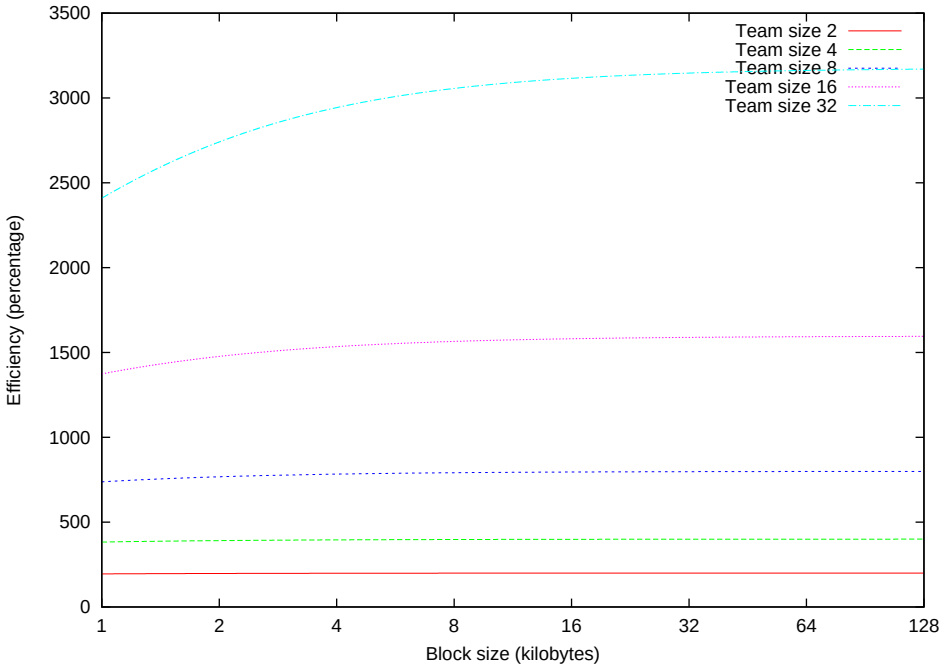


Figure 4.8: *The transport efficiency for supervisors. [P=4MB]*

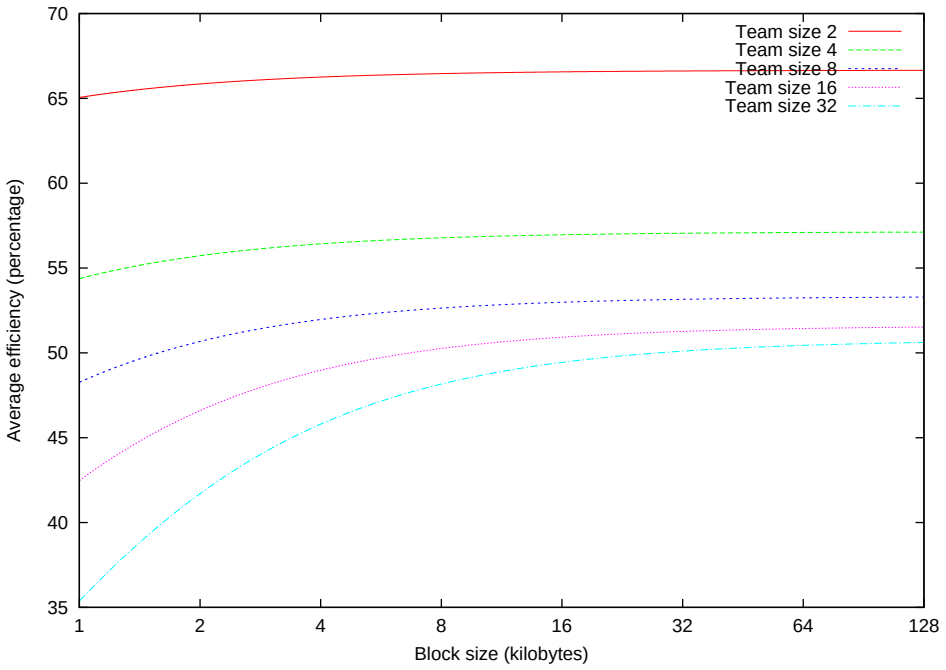


Figure 4.9: *The transport efficiency for team members. [P=4MB]*

Message	Latency time (milliseconds)	Transfer time (milliseconds)	
Information phase			
`TEAM INTERESTED`	200	0.29	
Setup phase			
`TEAM REQUEST`	200	5.47	
`TEAM REPLY`	200	0.33	
`TEAM MEMBER`	200	56.77	
`TEAM REPLY`	200	0.33	
`TEAM REWARD`	200	44.01	
Transfer phase (x32)			
`TEAM FORWARD`	6400	136625.00	
`TEAM FORWARD`	6400	17078.12	
`TEAM REWARD`	6400	72.92	
Total			
	20400	153883.24	
Efficiency			
Total in seconds	20.40	153.88	**174.28**
Overhead in seconds	18.80	17.35	**36.15**
Time efficiency			**79.26%**

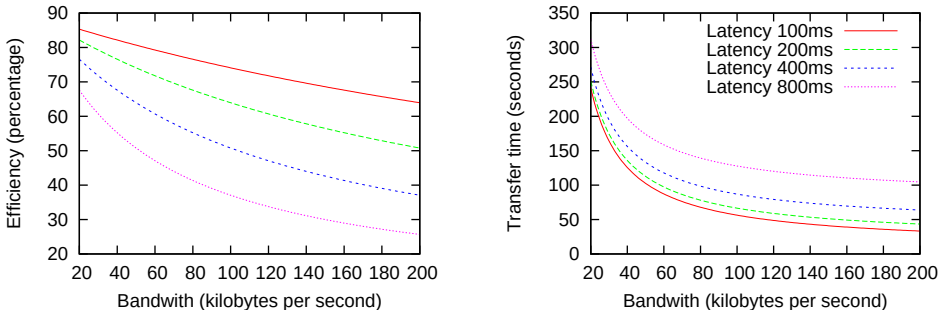
Table 4.2: *The time efficiency. [P=4MB, B=16kB, L=200ms, S=30kB/s, M=8]*

the previous two chapters where we let the block size grow to 4MB, we choose to limit the block size to 128kB. With 32 team members this gives each team member a single block. While this should not be used during an actual transfer, it shows the limits of the efficiency. Figures 4.8 and 4.9 show the transport efficiencies, using different team sizes, for the supervisor and team members, respectively.

From these figures we see that the efficiency for the supervisor grows linearly with the size of the team. In the efficiency for the team members two characteristics can be seen. For small blocks the efficiency is relatively low. This is primarily caused by the overhead of the many `TEAM REWARD` messages that are sent between the team members. As the block size increases, the number of blocks and the overhead decreases, resulting in a higher efficiency. At the larger block sizes, where this overhead is relatively low, we see the second characteristic, which is that the efficiency has a lower bound of 50%. Having a transport efficiency of 50% is the same as uploading the same amount as you download, or in other words, having a sharing ratio of 1:1.

4.2.2 Time efficiency

The time efficiency and how it is calculated for the transfer of a 4MB piece in 16kB blocks with a latency of 200ms, a bandwidth of 30kB/s to a size eight team



(a) Time efficiency.

(b) Transfer time.

Figure 4.10: *The time efficiency and the transfer time. [P=4MB, B=16kB, M=8]*

is given in Table 4.2. When we compare this table with Table 3.2, where the time efficiency for the Light Supervised Teaming protocol is presented, we see that the Extended Supervised Teaming protocol is almost 70 seconds faster. This is caused by the second `TEAM FORWARD` entry in the table. This entry indicates the time that it takes the team members to forward their assigned blocks to each other, having more team members provides more bandwidth that can be used in parallel, resulting in a faster distribution of the data.

The effect that the bandwidth and the latency have on the time efficiency and the resulting transfer time is presented in Figures 4.10(a) and 4.10(b), respectively. Compared to Light Supervised Teaming the Extended Supervised Teaming protocol has a higher time efficiency and therefore a better transfer time. However, the effect that the latency and the bandwidth have on these values is the same as with the Light Supervised Teaming protocol. Therefore, we will not go into this subject again.

We calculate the time that the transfer of a single 4MB piece takes when transferred using 16kB blocks and 200ms latency. The resulting transfer time, for several different team sizes, is presented in Figure 4.11. The figure shows that having more peers in the team will decrease the transfer time. When Figure 4.11 is compared with Figure 2.5, which shows the transfer time for the BitTorrent protocol, it shows that, as the size of the team increases, the transfer time of the Extended Supervised Teaming protocol approaches that of the BitTorrent protocol. A single team is effectively behaving as a small BitTorrent swarm. With each peer—or team member—the collective bandwidth of the swarm—or team—is increased. However, the lifetime of a team, and the authority that the supervisor has over the team members reduces risk and increases the control of seeders when compared to the BitTorrent protocol.

A fair comparison, on piece level, between BitTorrent and Extended Supervised Teaming is difficult because of the different rates at which the two protocols duplicate data. While with BitTorrent, a piece is duplicated from one peer to another, the Extended Supervised Teaming protocol duplicates a piece from one

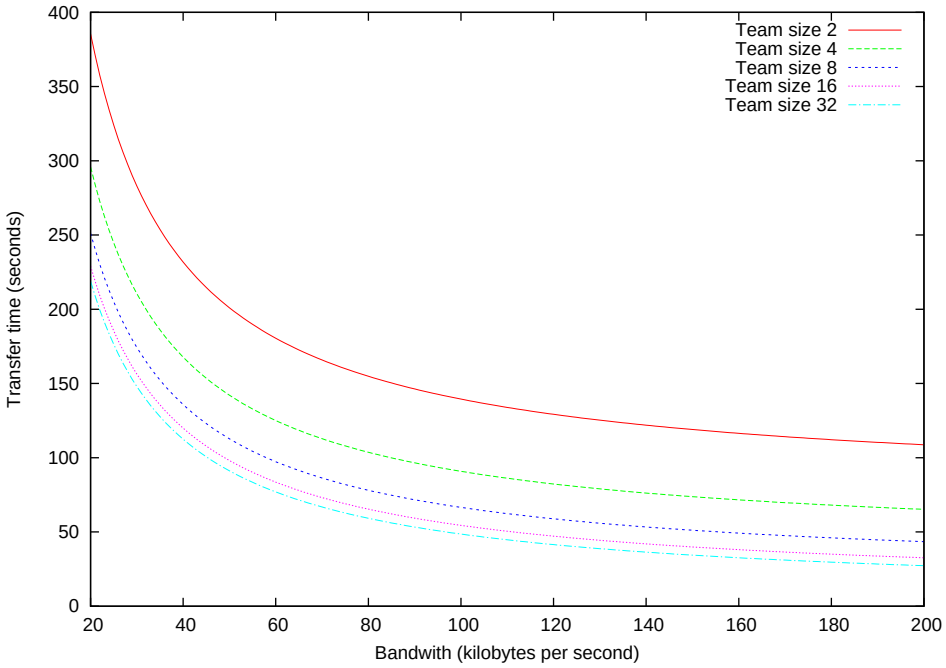


Figure 4.11: *The transfer time. [P=4MB, B=16kB, L=200ms]*

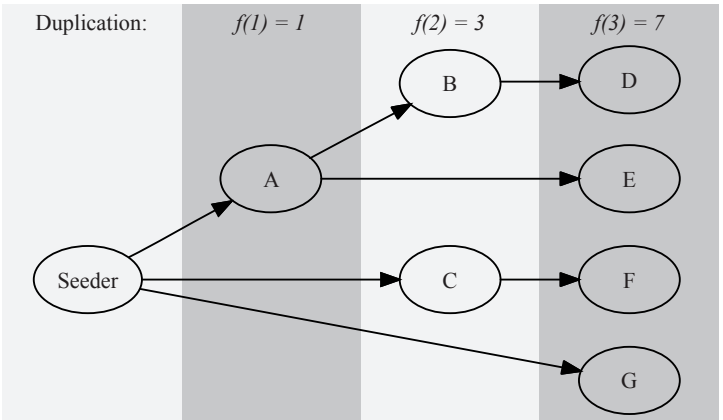


Figure 4.12: *Duplication of data using the BitTorrent protocol.*

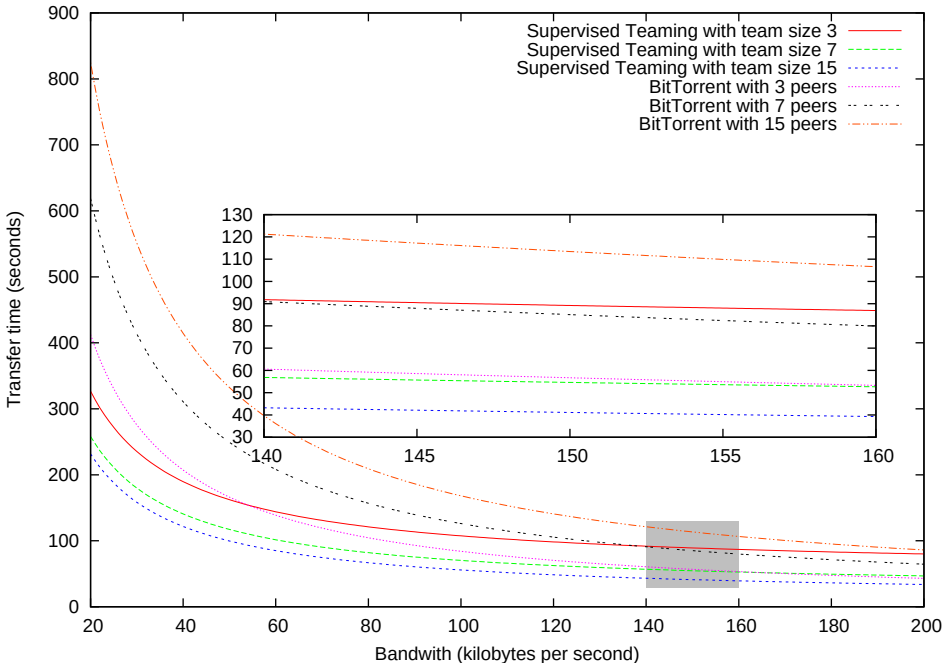
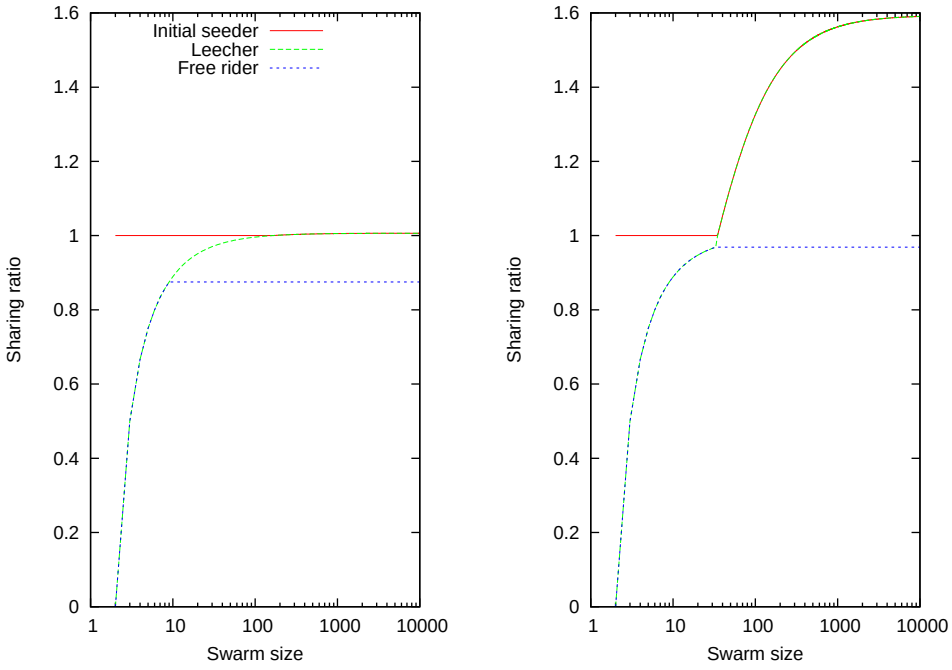


Figure 4.13: *The transfer time of Extended Supervised Teaming compared with that of BitTorrent. [P=4MB, B=16kB, L=200ms]*

supervisor to two or more team members. The rate at which the BitTorrent protocol duplicates data follows equation:

$$f(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2f(n-1) + 1 & \text{if } n > 1 \end{cases} \tag{4.1}$$

In this equation n is the number of rounds. The resulting speed of duplication is displayed graphically in Figure 4.12. From Equation 4.1 follows that, for a fair comparison between BitTorrent and Extended Supervised Teaming, we need to use 3, 7, 15, 31, etc. leechers. The Extended Supervised Teaming protocol can achieve this by using a team size of 3, 7, 15, 31, etc. The associated transfer times for these team sizes and peer counts are presented in Figure 4.13. This figure shows that, for a swarm with 7 or 15 peers, the Extended Supervised Teaming protocol duplicates the data faster than the BitTorrent protocol. Even with a swarm of 3 peers, the Extended Supervised Teaming protocol is faster at a bandwidth that is lower than 55kB/s. As explained in Section 3.2.2, this is caused by the parallel transfers that are part of the Extended Supervised Teaming protocol.



(a) Best-case scenario with 5% free riders. *[M=8]*

(b) Worst-case scenario with 95% free riders. *[M=32]*

Figure 4.14: *The sharing ratio. [F=100MB, NS=1, SS=SL=SF=128kB/s]*

4.2.3 Sharing ratio

The following equations are used to generate the sharing ratio for the Extended Supervised Teaming protocol. Similar to the equations used to generate the sharing ratio for BitTorrent and Light Supervised Teaming, these equations are only valid when $SS \times NS \times T \geq F$. The equations are as follows:

$$T = \frac{F \times (NL + NF) - (F/M) \times (M-1) \times NF}{SS \times NS + SL \times NL} \tag{4.2}$$

$$RS = \frac{SS \times T}{F} \tag{4.3}$$

$$RL = \frac{SL \times T}{F} \tag{4.4}$$

$$RF = \frac{(F/M) \times (M-1)}{F} \tag{4.5}$$

The sharing ratios that are presented in Figure 4.14 are generated using the same values as the swarm efficiencies for BitTorrent and Light Supervised Teaming. In the best-case scenario 5% of the swarm consists of free riders. In the worst-case scenario 95% of the swarm consists of free riders. The figures are generated using

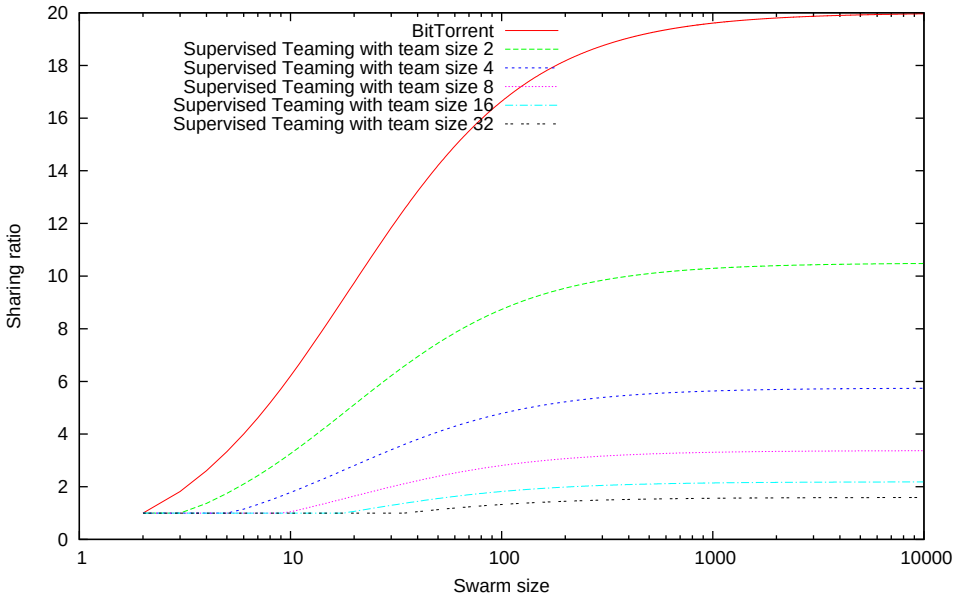


Figure 4.15: *Comparing the sharing ratio for the initial seeder in a worst-case scenario for Extended Supervised Teaming and BitTorrent. [F=100MB, NS=1, SS=SL=SF=128kB/s, M=8]*

a team size of 8 in the best-case scenario, and a team size of 32 in the worst-case scenario.

Figure 4.14(a), where the best-case scenario is shown, is very similar to those of the BitTorrent and Light Supervised Teaming protocols. The only difference that can be seen are with the smaller swarms. Therefore, the Extended Supervised Teaming protocol handles the best-case scenario just as efficiently.

The worst-case scenario, which is displayed in figure 4.14(b), shows that the free riders have a sharing ratio that is almost 1:1. This reduces the number of bytes that the initial seeder and the remaining leechers are required to upload, effectively giving them a 1:1.6 sharing ratio. When we compare this sharing ratio with the 1:20 that results from using BitTorrent, then we can clearly see the advantage of using Supervised Teaming in a worst-case scenario.

To further illustrated the difference between BitTorrent and Extended Supervised Teaming, we show the sharing ratios for an initial seeder in a worst-case scenario in Figure 4.15. In this figure we see that changing the team size allows great control over the resulting sharing ratio for the supervisors. Furthermore, we see that the Supervised Teaming protocol clearly results in a better sharing ratio that the BitTorrent protocol.

4.3 Concerning the problem statement

The first chapter in this thesis presents three problems that should at least partially be solved by using the Supervised Teaming protocol. In this section we will again look at these problems and describe how they are handled and whether they remain a problem.

Flash crowd. The best way for a peer-to-peer system to handle a large number of peers in a short amount of time is to use the bandwidth from as many peers as possible. Unfortunately, not all peers are able and willing to provide their bandwidth, furthermore, each peer in the swarm is likely to have a different bandwidth. Because it is very hard, if not impossible, to determine which peers are able and willing to upload, it is unavoidable that bandwidth is 'wasted' on peers that are only downloading.

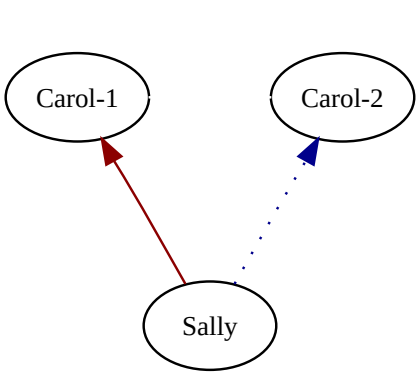
The method that the Extended Supervised Teaming protocol uses to increase the transport efficiency of seeding or supervising peers, achieves two things: the data is duplicated to more than one peer at the same bandwidth costs, and downloading peers are guaranteed to provide part of their bandwidth to the community.

While this will not solve the problem of flash crowds, it does reduce it by several factors. Because this factor depends on the team size, it is possible to change the factor as required. In case of flash crowds, where a few seeding peers are required to provide data to a relatively large number of downloading peers, a larger team size can be chosen to optimally use the few available seeders and altruistic peers. As the swarm matures, and more seeding or supervising peers become available, the team size can be reduced to compensate for the difficulty of finding team members.

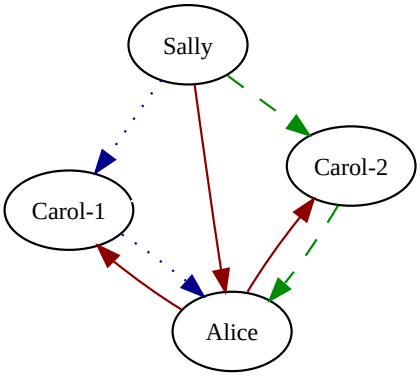
Free riding. Using Extended Supervised Teaming a supervisor can ensure that team members achieve a certain sharing ratio. For example, transferring a 4MB piece in 16kB blocks with team sizes: 2, 4, 8, 16, or 32 will ensure that the team members have sharing ratio 1:0.5, 1:0.75, 1:0.88, 1:0.94, or 1:0.97, respectively. Because we can hardly consider a peer with a sharing ratio of 1:0.88 or more to be a free rider, we feel that free riding has been solved.

However, by involving more than two peers in the transfer process, we have introduced the possibility of collusion [8]. Collusion occurs when two or more peers cooperate to gain an unfair advantage over other peers in the community. Unfortunately, collusion can be used to mislead the Supervised Teaming protocol. We distinguish between the following four forms of collusion:

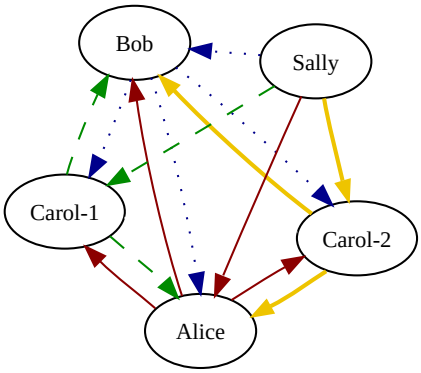
- Collusion by a single user owning several team members.
  When all the team members are at the same physical location, the cost of forwarding the data between them is more or less nothing. However, this collusion will only benefit the colluder if she owns all team members, see



(a) Successful collusion.



(b) Unsuccessful collusion.



(c) Unsuccessful collusion.

Figure 4.16: *Collusion by a single user.*

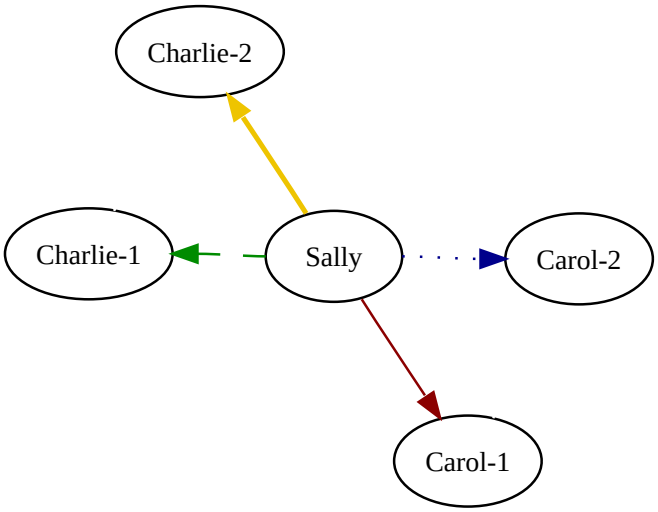


Figure 4.17: *Collusion by two users.*

Figure 4.16(a). This collusion will no longer work when a regular peer is introduced in the team. Figure 4.16(b) shows Alice—a regular peer—and Carol—the colluder—who are in a team. In this case Carol is still required to forward one block of data from each colluding peer that she owns. Furthermore, Alice forwards the same block to every colluding peer, wasting bandwidth. In this case there is no benefit for the colluder.

When a colluder is part of a team with two or more non colluding peers, shown in Figure 4.16(c), the colluder is required to upload even more than the non colluding peers. Therefore, collusion by a single user is only useful when the colluder is able to own all the team members. The supervisor can make this difficult by selecting a team that includes at least two different IP addresses. We believe that this collusion technique is unlikely to occur.

- Collusion by several users owning several team members.
  When several users are willing to collude together the possibility of a team consisting entirely of colluders will increase. However, such a team will not benefit the colluders when they are required to actually forward data to each other.

  For example, Figure 4.17 shows several peers owned by Carol and Charlie forming a team with Sally the supervisor. Sally starts sending blocks of data to the colluding team members, however, instead of forwarding the data, the colluding peers only send token messages to each other. After receiving these token messages, a confirmation message is sent to Sally. This way the colluders are only required to transfer small control messages while the supervisor supplies the data. The colluders can join several teams, and indicate which blocks they have, to eventually download the entire piece without uploading any blocks themselves.

  This strategy has the same drawbacks as the previous strategy, namely: colluders still have to upload to non colluding team members. Unfortunately, the only protection against this strategy is the chance that this may occur. When enough users use this strategy, the Supervised Teaming protocol is reduced to a classic server/client architecture.

- Collusion by influencing team member selection.
  The previous two collusion strategies show how important it is for colluders to influence the supervisor in her choice of team members. One of the few sources of information that influence this choice is the performance of previously known team members. A peer that has been part of many failed teams is less likely to accept invitations from someone with whom a team previously failed then from someone with whom past interaction went smoothly.

  A colluder can increase her success rate simply by following the protocol and forwarding blocks. While the cost of this is high, it will result in a

higher chance to be invited to teams. If the colluder owns several peers with a high success rate, the chance that her peers are selected for the same team, or a team with other colluders, also increases. Once a high success rate is achieved, the collusion attack can take place, without the supervisor realizing what is happening. Again, the chance to be invited to a team with fellow colluders depends on the size of the swarm and the percentage of colluders.

A less expensive method to increase the chance to be invited to a team containing only colluders, is to decrease the success ratio of teams that contain non colluding team members. This can be accomplished in two ways: 1) delay or do not send a confirmation message, 2) delay or do not send a forward message. The supervisor will not be able to blame any specific team member, and is therefore forced to blame everyone. Blame can be shifted in favor of the colluders when several colluders agree to drop the confirmation messages for a specific team member. A colluding peer that becomes useless because it has been involved in too many failed teams can be replaced with a new and often free peer, this is known as whitewashing [9].

- Collusion by influencing the team size.
  Because the size of the team is directly responsible for the number of blocks that a team member is required to upload, a free rider will try to reduce the team size. This can be achieved by making it look like other team members are not following the protocol. The two methods to achieve this have been discussed in the previous collusion strategy.

  A supervisor can counter this attack by refusing to supervise a team that is smaller than a certain number of team members. This minimum team size is communicated to each team member when they are invited to join the team. A team where too many members have left or were removed for any reason, should be disbanded, resulting in no benefit to the colluder.

Unfortunately it is impossible to completely solve these forms of collusion. However, the effectiveness of these collusion attacks will decrease dramatically when there are cooperating peers in the team. A supervisor that suspects collusion can increase the team size to increase the chance that non colluding peers are invited in a team. Other than this, the effectiveness of collusion depends on the number of colluders that are present in the swarm.

Initial risk. The initial risk that peers take is equal to the number of bytes that they give freely before they expect some bytes in return. The risk that the Extended Supervised Teaming protocol takes is equal to the size of the blocks that are transferred. A supervisor will risk, at most, one block for each member in the team. A team member will risk one block for each other member in the team. Initial risk can be reduced by using smaller blocks, however, this has the disadvantage of less efficiency, as described in Section 4.2.1.

It is possible for a free rider to take advantage of initial risk. When a free rider joins a team, she will receive one block from the supervisor and one block from each team member. Instead of forwarding her data, the free rider can leave the team. By repeating this, the free rider will eventually gather all the blocks from the piece. Because the free rider has the hash value of the completed piece, each combination of the blocks can be attempted. This brute force attack, while very inefficient, will allow the free rider to save upload bandwidth. However, the risk of downloading duplicated blocks is very high.

A free rider that, when she joins a team, indicates that only one or two blocks are still required, will not have to use an expensive brute force attack to find the locations of received blocks. In this case the supervisor is responsible for scheduling these blocks behind one or more blocks that the potential free rider is responsible for forwarding. This will remove any advantage that the free rider may have.

Chapter 5

Experiments and evaluation

Theory: when we know everything, but nothing works.
Practice: when everything works, but no one knows why.
We incorporate both theory and practice:
Nothing works, and no one knows why.
- Unknown

To prove that Supervised Teaming is not only a theoretical solution, we have implemented a peer-to-peer client that is capable of using both the BitTorrent and the Light Supervised Teaming protocol. This new implementation gives us complete knowledge of the source code and, furthermore, we have the ability to extract information about connection statistics and performance during our experiments. Our client is written in the programming language Python, which is the same language that Bram Cohen used to write his original BitTorrent client. Unfortunately it took far more time to implement this peer-to-peer client than we anticipated. We have spent more than 50% of our time working on it. However, we are satisfied with the resulting application and have used it to run the experiments that are discussed in this chapter.

Section 5.1 shows the similarity between the results from our implementation and the theory in Chapters 2 and 3. Section 5.2 gives the settings that are used for the experiments that are performed. Section 5.3 evaluates several experiments with varying numbers of leechers. Section 5.4 evaluates several experiments with varying numbers of free riders. And finally Section 5.5 evaluates an experiment where protocol violation is used.

	Theory (1 piece)	Experiment (10 pieces)	Difference (10 pieces)
Bytes uploaded by seeder	4,195,982	41,959,760	-60
Bytes downloaded by seeder	2,185	21,923	73
Byte efficiency for seeder	99.91%	99.91%	0%
Bytes uploaded by leecher	2,185	21,923	73
Bytes downloaded by leecher	4,195,982	41,959,760	-60
Byte efficiency for leecher	99.91%	99.91%	0%
	Theory (1 piece)	Experiment (10 pieces)	Difference (10 pieces)
Time in seconds	137.66	1,370.00	-6.60
Time efficiency	99.33%	99.67%	0.34%

Table 5.1: *BitTorrent: theory versus practice. [P=4MB, B=32kB, L=200ms, SS=SL=30kB/s]*

5.1 Verify the setup environment

Ideally theory and practice give the same results, however, this is usually not the case. This section will verify how closely a small experiment with the peer-to-peer client will match with the theory of the BitTorrent and Light Supervised Teaming protocols that have been presented in Sections 2.4 and 3.2 respectively.

In order to make this comparison between theory and practice possible, we extended the peer-to-peer client with a flexible statistics gathering module. This module, called *progress*, connects with a special progress server and continuously transfers all statistics that are generated by the client. Each statistics message contains a timestamp, key, and value. Using these statistics it is possible to, for example, piece together the exact number of data transfers between different peers, how fast these transfers were going, and how much a peer has transferred in a specific time period.

5.1.1 Verify setup: BitTorrent protocol

To verify the predicted transport and time efficiency of the BitTorrent protocol, we have run an experiment with two peers. We let the experiment mimic the settings that are used to generate the transport efficiency and time efficiency presented in Tables 2.2 and 2.3, respectively. The results from theory and practice should now correspond with each other.

After running the experiment and analyzing the statistics from the progress server, we find that a total of 41,981,683 bytes are required to transfer the 40MB from the seeder to the leecher. This allows us to determine that the transport efficiency for the experiment is 99.91%. This is the value that is predicted in Section 2.4. For convenience we present a summary of the theoretical prediction

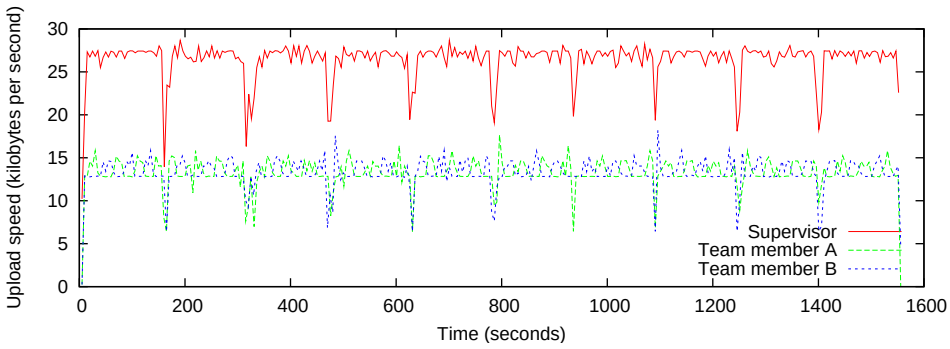


Figure 5.1: *The upload speed during transfer using Light Supervised Teaming. [F=40MB, P=4MB, B=32kB, SS=SL=30kB/s]*

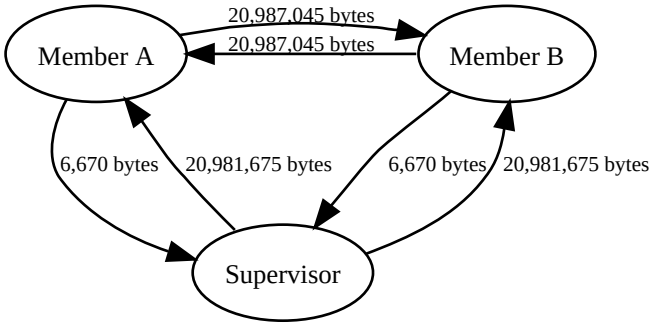


Figure 5.2: *The number of bytes transfered using Light Supervised Teaming. [F=40MB, P=4MB, B=32kB, SS=SL=30kB/s]*

from Table 2.2 and the results from the experiment in Table 5.1.

From the analysis of the experiment we also know that it takes 1370 seconds for the transfer to be completed. This allows us to determine that the time efficiency for the experiment is 99.67%. This is almost the same as the predicted 99.33%. For convenience these values are also presented in Table 5.1.

Differences between the theory and the experiment do exist, for example, the theory does not account for a handshake nor for any keep alive messages. However, the experiment shows that these differences have virtually no effect when transferring a 40MB file. Whether or not the comparison between the predicted and measured time efficiency can be trusted is cast into doubt because of the contradictory results from the same comparison for the Light Supervised Teaming protocol. The reasons for these doubts are described in the following section.

	Theory (1 piece)	Experiment (10 pieces)	Difference (10 pieces)
Bytes uploaded by supervisor	4,196,320	41,963,350	150
Bytes downloaded by supervisor	1,318	13,340	160
Byte efficiency for supervisor	199.84%	199.84%	0%
Bytes uploaded by team member	2,099,347	20,993,715	245
Bytes downloaded by team member	4,196,848	41,968,720	240
Byte efficiency for team member	66.62%	66.62%	0%

	Theory (1 piece)	Experiment (10 pieces)	Difference (10 pieces)
Time in seconds	244.11	1,556.00	-885.10
Time efficiency	56.10%	88.76%	32.66%

Table 5.2: *Light Supervised Teaming: theory versus practice. [P=4MB, B=32kB, L=200ms, SS=SL=30kB/s]*

5.1.2 Verify setup: Light Supervised Teaming protocol

Our second experiment intends to verify the predicted transport efficiency and time efficiency of the Light Supervised Teaming protocol that is described in Section 3.2. Again the experiment will mimic the settings from the theory presented in Tables 3.1 and 3.2 as much as possible. Because the theory is based on single pieces, we will allow the supervisor to supervise only one group at a time.

By plotting the upload speed statistics that were gathered during the experiment, Figure 5.1 is generated. It is clear that the supervisor is uploading at twice the speed of the team members. The figure clearly shows the ten distinct transfer phases separated by each setup phase where the transfer speed slows down. The number of bytes that are transferred between the three peers is also gathered by our progress logger, and is displayed in Figure 5.2. This figure confirms that the upload enforcement has the desired effect on the bandwidth usage of the team members.

As with the previous section, we summarize the difference between the theory and the experiment in Table 5.2. Because we know the amount of bandwidth that the supervisor used during the experiment, we can calculate the transport efficiency to be 199.84% which equals the value that is predicted in Section 3.2.1. Similarly we can calculate that the transport efficiency for the team members was 66.62% during the experiment, which also equals the predicted efficiency.

While not exactly visible in Figure 5.1, the transfer, of ten pieces, took 1556 seconds or 155.6 seconds for a single piece. This is much shorter than the 244.13 seconds that is predicted in Section 3.2.2. This is somewhat surprising because we expected that the theoretical transfer time would be shorter than the transfer time of our experiment. However, there are several factors that can account for discrepancies between the theory and our experiment:

- When running an application, the computer will add some delay to the

execution time, and thus the transfer time. Unfortunately we can not accurately measure how much time is added. From the statistics that are sent to the progress logger we know that the CPU load never exceeded the point where any process had to wait. However, it remains a multitasking system, thus delays must be introduced. Using a faster programming language or a more efficient implementation can reduce this influence, although it can not remove it. Note that this argument can only make the difference between the theory and the results of our experiment larger.

- Even though communication on local sockets is fast, there is still some delay that is introduced by the TCP stack. TCP uses complex algorithms that should enhance the throughput of data. However, we can only guess how this affects the transfer time. Again, this argument can only make the difference between the theory and our experiment larger.

- The theory uses a latency of 200 milliseconds which is not present in the experiment. The experiment is conducted on a single computer, removing most, if not all, latency cost. When we subtract the 39.20 seconds that was predicted to be the delay due to latency cost, the theoretical transfer time becomes 204.91 seconds with a 66.63% time efficiency. Unfortunately this still leaves a difference or 22.13% between the theory and our experiment.

- The peer-to-peer implementation is responsible for maintaining the bandwidth limit. It is possible that this mechanism is not acurate enouth, allowing the transfer at 'peak' values to be higher than the maximum allowed 30kB/s. The values that make up Figure 5.1 are refreshed every five seconds, however, when we increased the frequency to one second the maximum upload speed of 30kB/s was never reached either. The peer-to-peer client should limit the bandwidth based on a time window of maximal 1 second. It is likely that this window is to large.

Out of these factors, the last most likely explains the time discrepancy between the prediction and our experiment. However, increasing the acuracy of the bandwidth limiter would require either a complete rewrite of this mechanism or would result in higher CPU load. Unfortunately, we did not have the time to experiment with this further.

5.2 Experiment setup

The Supervised Teaming protocol is designed to be strong in the face of several specific problems. With these strengths we believe that Supervised Teaming is able to outperform the widely used BitTorrent protocol in the three problematic circumstances that are described in the introduction of this thesis. Namely: flash crowd, free riding, and initial risk. To provide a proof of concept for Supervised

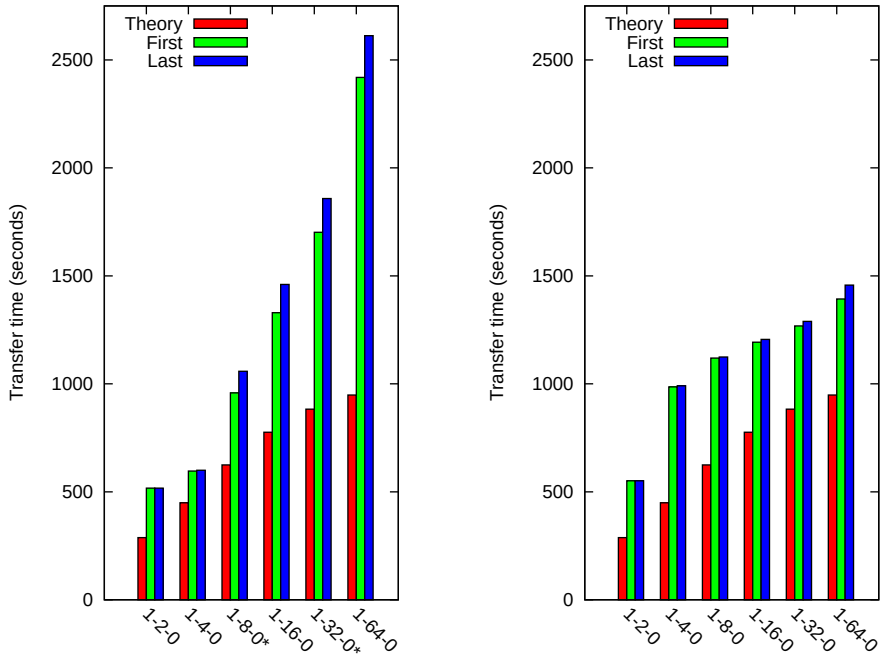
Teaming, several experiments are done with both the Light Supervised Teaming and the BitTorrent protocol for each of the three problems. Because these experiments are done in largely the same way, we discuss the general setup of these experiments here.

An experiment with one initial seeder, two leechers, and four free riders will be indicated as `1-2-4`. This notation is used in the remainder of this chapter. An initial seeder begins with all data as opposed to leechers and free riders who start with no data. With BitTorrent a free rider will never unchoke a connection and with Light Supervised Teaming a free rider will never supervise a team. Seeders and leechers will not be this selfish and will upload to other peers according to the design of the protocol.

Each experiment will transfer 100MB from one initial seeder to a varying number of leechers and free riders. To reduce file system overhead, this data is not read from or written to disk. A piece size of 256kB is used, giving the peer-to-peer clients many opportunities to trade with each other, this should benefit the tit-for-tat strategy used by the BitTorrent protocol. For the block size we use 8kB, this relatively small value should benefit risk reduction for the Light Supervised Teaming protocol. Furthermore, all leechers and free riders are allowed to upload at a rate of 100kB/s and download at a rate of 512kB/s. The initial seeders are allowed to upload and download at both 512kB/s.

At first we wanted to run experiments with hundreds of peers. However, with the current implementation of the Light Supervised Teaming protocol this proved to be a problem. The pushing strategy, that is used by Light Supervised Teaming is very effective when there are relatively few supervising peers. However, as time progresses, the number of supervisors grows. Near the end of the experiment, each peer that still requires data receives numerous offers for a team. However, a team can only be created when both potential team members accept the offer from the same supervisor. Therefore, the chance that a team is created decreases exponentially as the number of supervisors grows. Because of this, we decided to limit our experiments to around 50 peers. How this problem can be solved, is discussed in Section 4.1.6.

Furthermore we should note that the peer-to-peer client, the Light Supervised Teaming protocol, and the BitTorrent protocol were implemented in no more than six months by a single person. This is much less development time than commercial BitTorrent clients receive. We can only assume that these commercial applications will be faster and more efficient than the implementation that we are using. This has two consequences. The first is that the statistics that we present—the time required to complete a download and the amount of used bandwidth—is not always based on 100% of the peers completing 100% of the download, this is indicated by a '*' behind the experiment indicator like `1-8-0*`. And the second consequence is that the results of the experiments should be seen as a proof of concept.



(a) BitTorrent.

(b) Light Supervised Teaming.

Figure 5.3: *The transfer time for the first and last completed peer. [F=100MB, P=256kB, B=8kB, SS=512kB/s, SL=100kB/s]*

5.3 Flash crowd

With the first experiment we will try to show the robustness of the Light Supervised Teaming protocol in the face of a flash crowd. We have performed six experiments for both the BitTorrent and the Light Supervised Teaming protocols. In each experiment one initial seeder will transfer a file to a varying number of leechers. The number of leechers will increase exponentially with each experiment. By analyzing the statistics that are gathered during the experiments, the time to completion for each leecher can be retrieved. The times when the first and the last leecher completed the download is plotted in Figure 5.3.

As a reference the theoretical transfer time is also plotted in Figure 5.3. This theoretical transfer time represents the minimal time it takes to transfer the file to every leecher using the combined upload speed of the initial seeder and all the leechers and is obtained from the equations that are used to calculate the sharing ratio which are presented in Sections 2.4.3 and 3.2.3 for the BitTorrent and Light Supervised Teaming protocols, respectively.

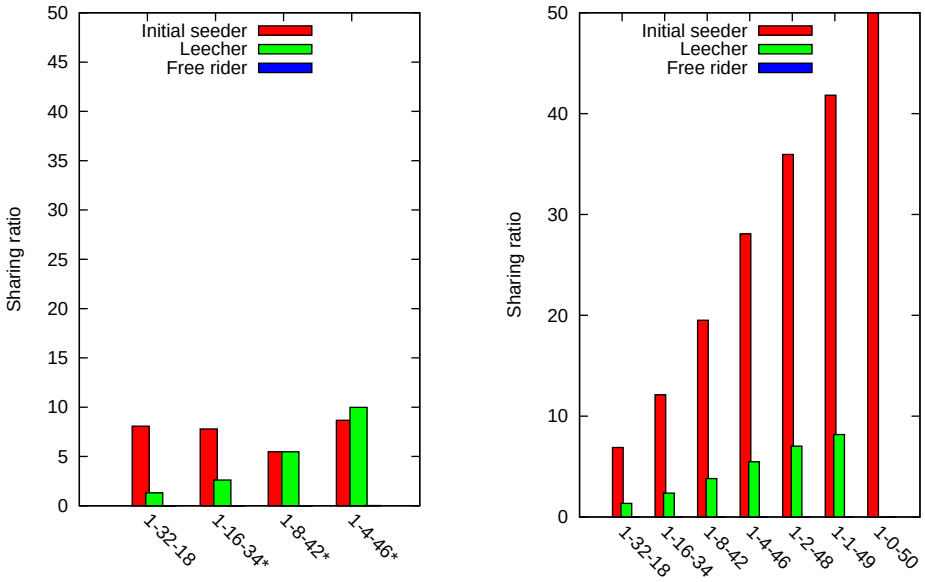
Unfortunately not all of the leechers in the BitTorrent experiments `1-8-0*` and `1-32-0*` were able to completely download the entire file. In the case of experiment `1-8-0*` all of the leechers downloaded at least 99.5% of the file

while 75% of the leechers were able to download 100% of the file. In the case of experiment `1-32-0*` all of the leechers downloaded at least 98.75% of the file while 93.75% of the leechers were able to download 100% of the file. Because only a small fraction of the download failed, we decided that it was not worth redoing the experiments.

When we look at Figure 5.3(a) we see that the difference between the theoretical transfer time and the transfer time that is measured in our BitTorrent experiments continues to increase. This is not what we were hoping to see. BitTorrent is known to have problems in flash crowd situations, this has already been addressed and several BitTorrent clients allow a special initial seeding mode to be used that alleviates this problem. However, the exponential increase in transfer time that we see in Figure 5.3(a) indicates that more is going on. The speed at which the transfers can occur is highly dependent on details in the implementation. Given the relatively short development time of our implementation, we can assume that there are much more efficient implementations available. Given our limited remaining time for this thesis, we decided against further experiments with an alternative BitTorrent client. We will therefore not compare the results of our Light Supervised Teaming protocol with those of our BitTorrent protocol.

Now that a comparison between the transfer times with the BitTorrent and the Light Supervised Teaming protocols has been ruled out, only the comparison between the theoretical transfer time and the experiments remain, which are shown in Figure 5.3(b). This figure shows that the difference between transfer times of the theory and experiments remains roughly the same. Were this to hold true for larger peer-to-peer communities, the Light Supervised Teaming protocol will definitely scale with the number of peers, providing a valuable solution to the flash crowd problem.

For completeness, we have provided graphs, showing the up and download speed for all the peers in all the experiments that were done in this chapter, in Appendix B. Looking at these figures, we see that there certainly is room to improve the implementation for both our BitTorrent and Light Supervised Teaming clients. When, for example, we look at the upload speed of the initial seeder during experiment `1-2-0` with the Light Supervised Teaming protocol, we see in Figure B.7(a) that she is uploading at her full capacity. However, during experiment `1-4-0` we see, in Figure B.8(a), that the initial seeder can no longer maintain her maximum allowed upload speed for the duration of the experiment. While this does reduce her bandwidth cost, it also increases the transfer time. We believe that this drop in upload speed is caused by the limited upload bandwidth of the leechers. The peer-to-peer client currently accepts all team requests and can therefore be in many teams, thus using a lot of upload bandwidth. Furthermore, each leecher is allowed to create several teams herself, using even more upload bandwidth. It would be an improvement if the leechers would choose whether to create or join a team, based on their remaining up and download bandwidth. However, there was not enough time remaining to implement these improvements.



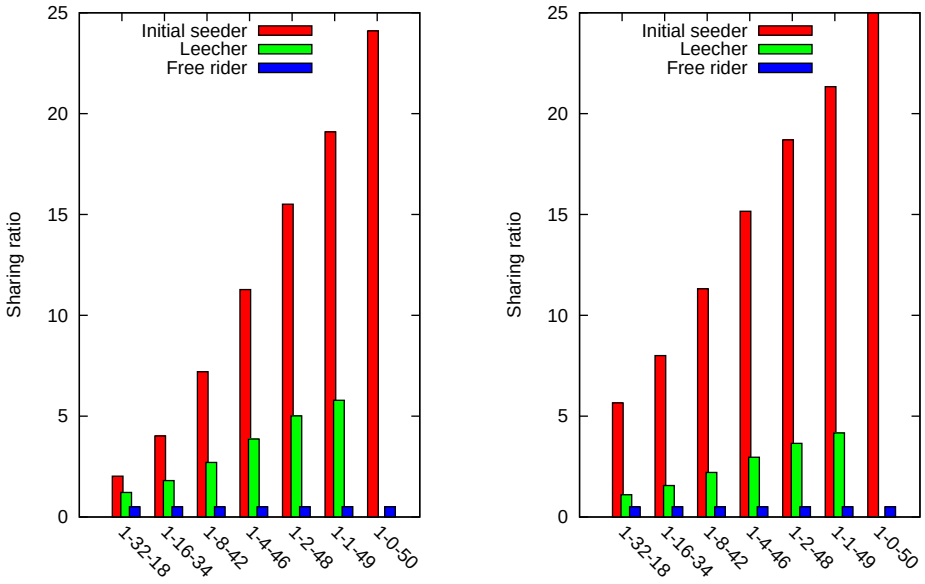
(a) Experiment. (b) Theory.

Figure 5.4: *Sharing ratio with the BitTorrent protocol. [F=100MB, P=256kB, B=8kB, SS=512kB/s, SL=SF=100kB/s]*

5.4 Free riding

In the experiments from the previous section, all peers were willing to upload to each other. However, this is not the case in an actual peer-to-peer community where not only the willingness, but also the available bandwidth can be different for each peer. There may be many peers that will attempt to download using as little upload bandwidth as possible. While the lack of upload bandwidth from these peers—who we call free riders—is often balanced by other altruistic peers, it can become a significant problem when the number of free riders increase, resulting in higher download times for every peer in the swarm. In this section we will show the true strength of Supervised Teaming, namely the ability to force free riders to contribute part of their bandwidth to the community, thereby reducing the bandwidth cost for initial seeders and altruistic peers.

All experiments in this section will involve one initial seeder and 50 downloading peers who are distributed between leechers and free riders. With each new experiment the number of leechers will decrease exponentially. Because there are 50 downloading peers, a total of 5000MB must be uploaded during each experiment. By analyzing the statistics from each experiment, we know how much of this data is uploaded by each peer. The resulting average sharing ratio for the initial seeder, the leechers, and the free riders are shown in Figures 5.4(a) and 5.5(a), for the BitTorrent and Light Supervised Teaming protocols, respectively.



(a) Experiment. (b) Theory.

Figure 5.5: *Sharing ratio with the Light Supervised Teaming protocol. [F=100MB, P=256kB, B=8kB, SS=512kB/s, SL=SF=100kB/s]*

The sharing ratio that is predicted by the theory from Sections 2.4.3 and 3.2.3 is presented in Figures 5.4(b) and 5.5(b), respectively.

Unfortunately, only four of the seven experiments that we intended to perform with the BitTorrent protocol were able to complete even though we let them run for more then 12 hours. From these four experiments there was only one where all peers were able to completely download the entire file. From the experiments that were unable to complete, the worst experiment was still able to let 94% of the peers download 99.5% of the file. The low sharing ratio in the four experiments, that were more or less able to complete, indicate that the initial seeder seems especially reluctant to upload any data. While we can verify this with the upload speed figures from Appendix B we have not been able to find the cause, and because we have limited time available for this thesis, we decided not to attempt to solve this issue. Instead we will compare the performance of the Light Supervised Teaming protocol against the theoretical sharing ratio.

When we compare the results from our experiments with the Light Supervised Teaming protocol, in Figure 5.5(a), with the optimal results from our theory, in Figure 5.5(b), we see the similarities. The only visible difference lies in the sharing ratio for the initial seeder which is lower in the experiment. This difference is caused by a reluctance to upload data. Unfortunately, we do not know the cause of this reluctance.

When we compare the results from the Light Supervised Teaming experiments

with the theoretical sharing ratio of the BitTorrent protocol, it is clear that the mechanism that forces peers to upload, when they are part of a team, helps reduce the pressure on the initial seeder and altruistic leechers. This is clearly seen in the `1-0-50` experiment where the initial seeder would have a sharing ratio of 1:50 with BitTorrent and only 1:25 with Light Supervised Teaming, effectively saving 2500MB worth of bandwidth. Therefore, the Supervised Teaming protocol is better equipped to handle free riding situations than the BitTorrent protocol.

So far all the peers have followed the prescribed protocols. However, free riders are known for their inventive ways to elude or even break a protocol. How this affects the performance is described in the following section.

5.5 Initial risk

Free riding can be achieved in many ways. One well known method is whitewashing, where a free rider continually exploits an initial seeder or a fellow leecher. Supervised Teaming was designed to alleviate this problem by reducing the amount of data that can be acquired for free, see Section 3.3. To view the effects of this mechanism, we will introduce free riders who are willing to perform protocol violation. Because BitTorrent is not designed to reduce free riding, we believe that a comparison between BitTorrent and Light Supervised Teaming would be unfair. However, it should be mentioned that, with the BitTorrent protocol, the free riders are not punished in any way, and will be able to download what they are interested in without uploading any data themselves. Even though the tit-for-tat mechanism will ensure that free riders will not have a high priority on the upload lists of the other peers in the swarm.

Our experiment will contain one initial seeder who will upload data to two leechers and two free riders. These free riders will not forward any data to their team member, thereby violating the protocol. Instead they will do nothing. Not receiving a confirmation message, the supervisor will—after a few seconds—disband the team. We specifically chose the `1-2-2` structure to keep the experiment small enough so we can clearly see the specific data flows between peers while we still have leechers who should be able to complete the file despite the presence of the free riders.

We stopped the experiment after the two leechers completed the download. The data flows that occurred during this time are shown in Figure 5.6. Because the free riders did not send more than 800 bytes to any of the other peers, we have chosen not to display these data flows in order to simplify the resulting figure. The figure shows that both the leechers and free riders were able to acquire data. However, the free riders received less team requests because each peer registers protocol violations, and protocol violations involving the free riders occurred more often. When the initial seeder creates a team with leecher A and free rider B, no conformation message will be received, however, there is no information available

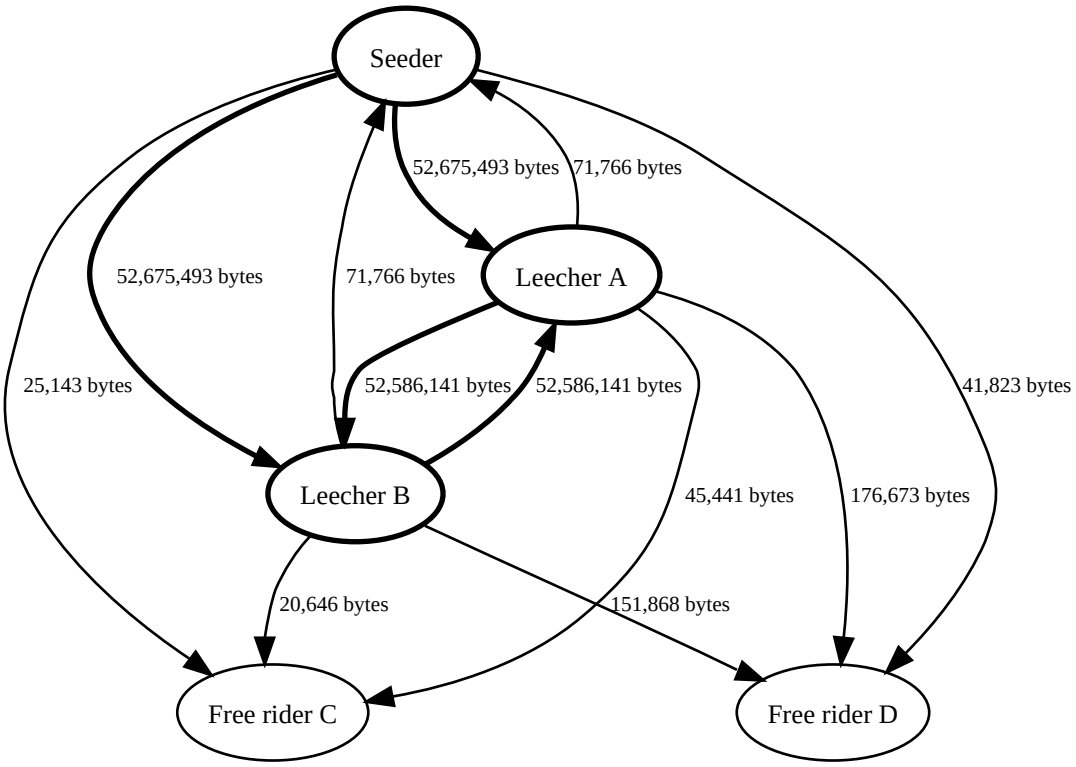


Figure 5.6: *The number of bytes transferred during a* `1-2-2` *experiment with protocol violation. [F=100MB, P=256kB, B=8kB, SS=512kB/s, SL=SF=100kB/s]*

to tell which one of the team members violated the protocol. It is for this reason, that the initial seeder and the leechers continue to invite the free riders to join a group. Each peer should choose how often they invite peers that may be involved in protocol violation.

Chapter 6

Conclusion

A conclusion is simply the place where someone got tired of thinking.
- Arthor Bloch

This thesis presents solutions to the problems of flash crowds, free riding, and initial risk by introducing the Supervised Teaming protocol. This protocol uses one peer, the supervisor, who has authority over several other peers, the team members. To download a piece, a peer has to be invited by a supervisor to join a team. Once a team is created, the supervisor will send blocks of data to the team members who, because of an incentive, forward these blocks to each other. While downloading, a peer will have to join a team for each piece in the file. A peer can increase her chance of being invited to join a team, by supervising teams for pieces that they have downloaded.

To evaluate the performance of the Supervised Teaming protocol, we use the transport efficiency, the time efficiency, and the sharing ratio. Using these metrics, we have determined that our protocol is just as efficient as the BitTorrent protocol when no free riding occurs. However, we have also determined that Supervised Teaming outperforms BitTorrent when free riders are present in the swarm.

To verify that the presented theories hold true in practice, we have implemented a fully functional peer-to-peer client that is able to use both the BitTorrent protocol and a simplified version of the Supervised Teaming protocol, where the simplified version is restricted to a team size of two. Our first experiment shows that our protocol takes a more or less constant amount of time more to distribute a file than is predicted by the theory. Our second experiment shows that our protocol ensures that every peer, even free riders, have a sharing ratio of at least 1:0.5 when the team size is two, thereby reducing the bandwidth cost of the initial seeder by half. Finally, our third experiment shows that a free rider will not obtain any benefits from using protocol violation.

The flash crowd problem, which occurs when large numbers of leechers join a swarm in a short amount of time, can be solved by increasing the team size. This reduces the bandwidth usage of the available seeders by using more bandwidth from the team members. Increasing the team size can ensure that the team members obtain a sharing ratio of almost 1:1, which is the best possible solution to a flash crowd.

The free riding problem, which occurs when peers do not contribute a fair amount—where fair indicates an amount equal to their benefit—of resources back to the community, can be solved by having the supervisors control the sharing ratio of their team members, which is possible by changing the team size. A large team results in a high sharing ratio, and a small team results in a low sharing ratio for the team members.

The initial risk problem, which occurs when a free rider receives data by promising, without actually delivering, data in return, is often caused by the inherent anonymity of the internet. This problem can be solved by ensuring that a team member is unable to use a received block of data until the associated reward is also received. An incentive is provided by giving the reward only after a successful forward.

Supervised Teaming has two disadvantages. The first disadvantage is that a supervisor relies on third party information, and is therefore unable to determine with certainty, which of the team members is not performing correctly. Therefore, the protocol becomes vulnerable to collusion attacks. However, collusion is only successful when all the team members are colluding together. A team with a single cooperating peer and several colluders results in no benefit to the colluders, and no additional cost to either the supervisor or the cooperating peer. With two or more cooperating peers in a team, collusion becomes more expensive than cooperation. Unfortunately it is not possible to identify colluders, therefore, even though it is unlikely, it is possible that a team consists entirely of colluders.

The second disadvantage of Supervised Teaming is that it requires the supervisor and every team member to be connected to each other. Therefore, there can only be a single peer, either the supervisor or one of the team members, that is behind a NAT or firewalled connection. However, we believe that this can be solved with proper NAT and firewall traversal techniques, or the introduction of the IPv6 protocol.

There are many challenges that are yet to be solved. Therefore, we suggest the following points of interest that can be researched in the future.

- Having connectable peers is very important for Supervised Teaming. Therefore, one of the NAT and or firewall traversal techniques should be implemented into the peer-to-peer client.

- Splitting up a connection into parallel channels is required when two or more peers are cooperating with each other for the same team.

- The team size is important for the transport efficiency and the sharing ratio. While the team size can be chosen based on the available `TEAM INTERESTED`, `TEAM NOT INTERESTED`, and `TEAM INTERESTED FIELD` messages, this thesis does not discuss any equations or guidelines to obtain an optimal team size from these messages.
- Using the confirmation messages, the supervisor is able to estimated the transfer speed of individual team members. This transfer speed can be used to guarantee specific download speeds which can be beneficial for the download of real time video streams.

Bibliography

[1] Massive Scaling 2006. *http://www.ams-ix.net/news/archive/2006/AMS-IX_Massive_scaling_GPF1_2006.pdf*.

[2] RatioMaster. *http://www.moofdev.org/ratiomaster*.

[3] R. Axelrod and WD Hamilton. The evolution of cooperation. *Science*, 211(4489):1390, 1981.

[4] A.R. Bharambe, C. Herley, and V.N. Padmanabhan. Analyzing and Improving a BitTorrent Network's Performance Mechanisms. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, 2006.

[5] B. Bobier. BitTorrent's Transfer Optimizations.

[6] Y. Chu, A. Ganjam, T.S.E. Ng, S.G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an internet broadcast system based on overlay multicast. *Proceedings of the USENIX Annual Technical Conference 2004 on USENIX Annual Technical Conference table of contents*, pages 12–12, 2004.

[7] B. Cohen. Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer Systems*, 6, 2003.

[8] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. *Proceedings of the 5th ACM conference on Electronic commerce*, pages 102–111, 2004.

[9] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica. Free-riding and whitewashing in peer-to-peer systems. *Selected Areas in Communications, IEEE Journal on*, 24(5):1010–1019, 2006.

[10] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. *Proceedings of the 2005 USENIX Annual Technical Conference*.

[11] G. Goth. Close to the edge: NAT vs. IPv6 just the tip of a larger problem. *Internet Computing, IEEE*, 9(2):6–9, 2005.

[12] S. Guha and P. Francis. Characterization and measurement of tcp traversal through nats and firewalls. *ACM IMC*, 2005.

[13] D. Hales and S. Patarin. How to cheat bittorrent and why nobody does. Technical report, TR UBLCS-2005-12, Department of Computer Science University of Bologna, May 2005.

[14] G.E.T.T. HARDIN. The Tragedy of the Commons. *Science Magazine's State of the Planet 2006-2007*, 2006.

[15] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. File-sharing in the Internet: A characterization of P2P traffic in the backbone. *University of California, Riverside, USA, Tech. Rep*, 2003.

[16] S.D. Koolen. Creating and Maintaining Relationships in Social Peer-to-Peer Networks.

[17] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting bittorrent for fun (but not profit). *Proc. 5th Itl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[18] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding in BitTorrent is Cheap. *Connections*, 300(400):500.

[19] A. Parker. CacheLogic.

[20] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis.

[21] A. Rogers, RK Dash, SD Ramchurn, P. Vytelingum, and NR Jennings. Coordinating team players within a noisy Iterated s Dilemma tournament. *Theoretical Computer Science*, 377:243–259, 2007.

[22] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. RFC3489: STUN-Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). *Internet RFCs*, 2003.

[23] P.B. Schoon. Stimulating fairness in peer to peer networks. Technical report.

[24] M. Siner. Verification of BitTorrent Seeder Utilization.

[25] W. Wang, H. Chang, A. Zeitoun, and S. Jamin. Characterizing guarded hosts in peer-to-peer file sharing systems. *Global Telecommunications Conference, 2004*.

Appendix A

Messages used by the protocols

Throughout this thesis we describe the protocols for BitTorrent, Light Supervised Teaming, and Extended Supervised Teaming. Because we calculate several efficiencies for these protocols, it is important to know exactly which messages there are and how large they can be. Therefore, the following sections will further described the messages as they have been used throughout this thesis. These messages are preceded with a message header containing a four byte message length followed by one byte indicating the message identifier.

A.1 BitTorrent messages

The messages that are used by the BitTorrent protocol, which are described in this section, are shown in Figure A.1.

`BITFIELD: bitfield`

The `BITFIELD` message consists of a `bitfield`. The length of this `bitfield` depends on the number of pieces in the torrent. The first bit indicates whether the sender has the first piece, the second bit indicates the same for the second piece, etc. Any spare bits at the end of the `bitfield` should be set to zero.

`HAVE: piece-id`

The `HAVE` message is sent when a peer has completely downloaded a certain piece. The `piece-id` consists of four bytes indicating the piece that the sender has completed.

`INTERESTED`

The `INTERESTED` message has no payload and indicates that the sender is interested in one or more pieces of the receiver.

`NOT INTERESTED`

The `NOT INTERESTED` message has no payload and indicates that the receiver does not have any pieces that the sender is interested in.

`UNCHOKE`

The `UNCHOKE` message has no payload and indicates that the sender is willing

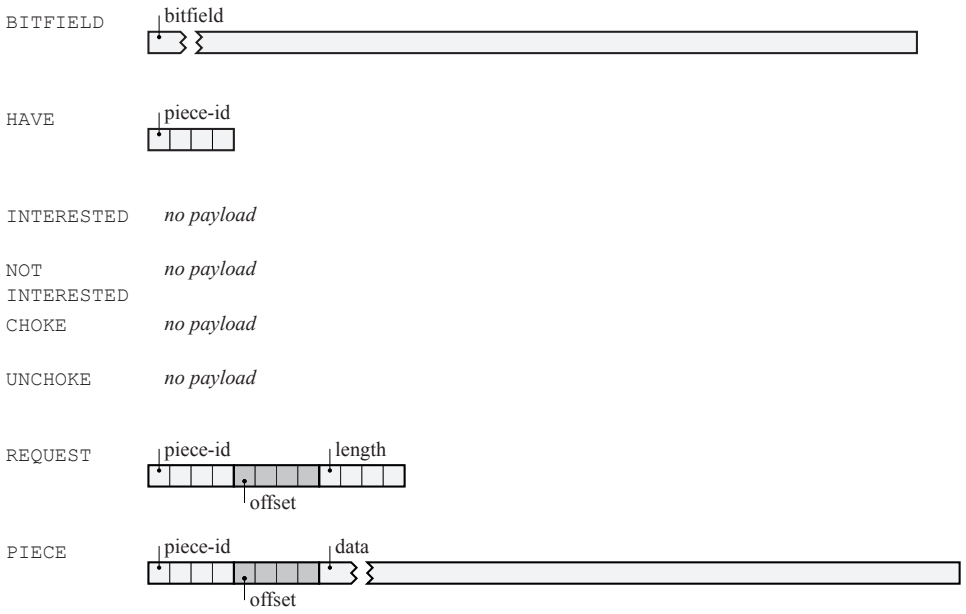


Figure A.1: *The messages and their payload used by the BitTorrent protocol.*

to upload data to the receiver. This message can, for instance, be sent after an `INTERESTED` message is received.

`CHOKE`

The `CHOKE` message has no payload and indicates that the sender is no longer willing to upload data to the receiver.

`REQUEST`: `piece-id`, `offset`, `length`

The `REQUEST` message is sent to a non choking peer that the sender is interested in. The `piece-id` consists of four bytes that indicate the piece that the data should come from. The `offset` consists of four bytes indicating the byte offset of the required data, this offset is relative to the beginning of the `piece-id`. And the `length` consists of four bytes indicating the length of the requested data.

`PIECE`: `piece-id`, `offset`, `data`

The `PIECE` message is sent in response to a `REQUEST` message and contains the requested data. The name of this message is somewhat misleading because this message only contains a small part of a piece, commonly referred to as a block. The `piece-id` consists of four bytes indicating the requested piece. The `offset` consists of four bytes indicating the offset of the requested data, relative to the beginning of the piece. And the `data` consists of a certain number of bytes containing the requested data.

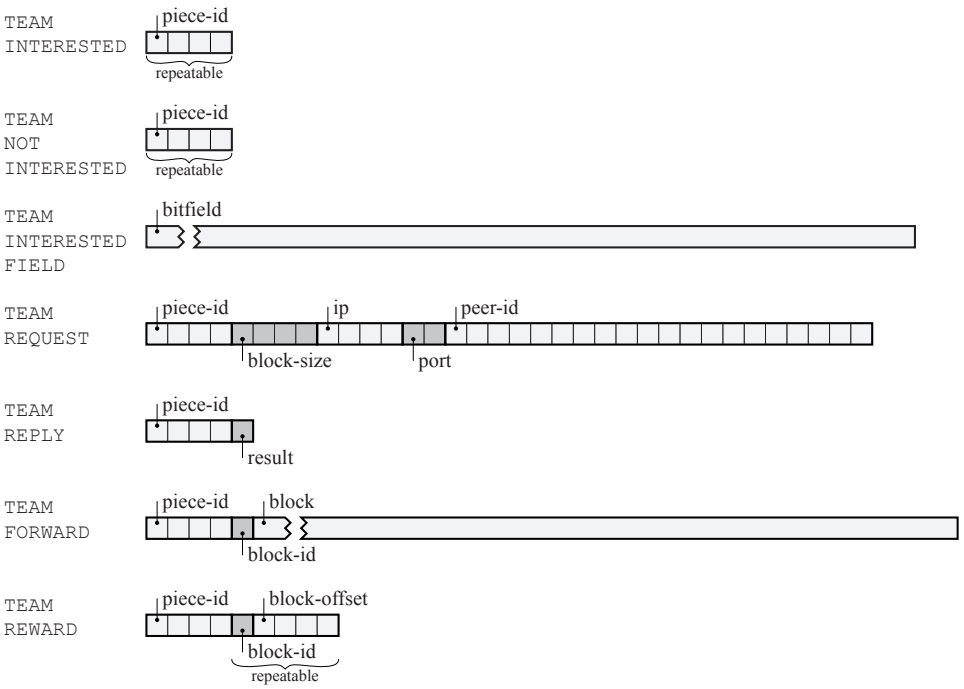


Figure A.2: *The messages and their payload used by the Light Supervised Teaming protocol.*

A.2 Light Supervised Teaming messages

The messages that are used by the Light Supervised Teaming protocol, which are described in this section, are shown in Figure A.2.

`TEAM INTERESTED`: `[piece-id]+`
The `TEAM INTERESTED` message consists of one or more `piece-id`'s. Each `piece-id` is a four byte number that indicates that the peer is interested in obtaining this piece.

`TEAM NOT INTERESTED`: `[piece-id]+`
The `TEAM NOT INTERESTED` message consists of one or more `piece-id`'s. Each `piece-id` is a four byte number that indicates that the peer is not interested in obtaining this piece.

`TEAM INTERESTED FIELD`: `bitfield`
The `TEAM INTERESTED FIELD` message consists of a `bitfield`. The length of this `bitfield` depends on the number of pieces in the file. The first byte represents whether the peer is interested in the first eight pieces, the second byte represents the next eight pieces, etc. Any spare bits should be set to zero.

`TEAM REQUEST`: `piece-id`, `block-size`, `ip`, `port`, `peer-id`
The `TEAM REQUEST` message is used by a potential supervisor to contact potential team members. The `piece-id` consists of four bytes indicating the

piece that is offered. The `block-size` consists of four bytes indicating the number of bytes in each block. And the `ip`, `port`, and `peer-id` consists of four, two and twenty bytes respectively, and indicate the other peer that will be part of the team.

`TEAM REPLY`: `piece-id`, `result`

The `TEAM REPLY` message is used in several situations. It is used by a potential team member to reply to a `TEAM REQUEST` message. It is used by a team member to tell the supervisor that the peer wishes to leave the team prematurely. And it is used by the supervisor to tell team members that the team will be disbanded prematurely. The `piece-id` consists of four bytes indicating the piece for which the message is intended. The `result` consists of one byte and can indicate different things depending on the situation that the message is used in, these specific values will not be discussed here.

`TEAM FORWARD`: `piece-id`, `block-id`, `data`

The `TEAM FORWARD` message is used to send a block of data from one peer to the other. The `piece-id` consists of four bytes indicating the piece that this message belongs to. The `block-id` consists of one byte indicating the randomly chosen identifier for this block. And the `data` contains a variable number of bytes containing the data for this block.

`TEAM REWARD`: `piece-id`, `[block-id, block-offset]+`

The `TEAM REWARD` message is used to send the (`block-id`, `block-offset`) pairs between the supervisor and the team members. Furthermore, when this message is sent to the supervisor it is allowed to omit the `block-offset` for the last pair in the message. The `piece-id` consists of four bytes indicating the piece that this message belongs to. The `block-id` consists of one byte indicating the randomly chosen identifier for this block. And the `block-offset` consists of four bytes indicating the byte offset for the associated block relative to the start of the piece.

A.3 Extended Supervised Teaming messages

The messages that are used by the Extended Supervised Teaming protocol, which are described in this section, are shown in Figure A.3.

`TEAM REQUEST`: `piece-id`, `eager-value`, `block-size`,
`min-team-size`, `max-team-size`, `min-upload-bandwidth`,
`min-download-bandwidth`

The `TEAM REQUEST` message is used by a potential supervisor to contact potential team members. While the Light Supervised Teaming version of the `TEAM REQUEST` message included the IP address of the team member, with the Extended Supervised Teaming version the addresses of the team members are provided in a separate message that is sent after the potential team members have accepted the conditions that are presented in this message. The `piece-id`

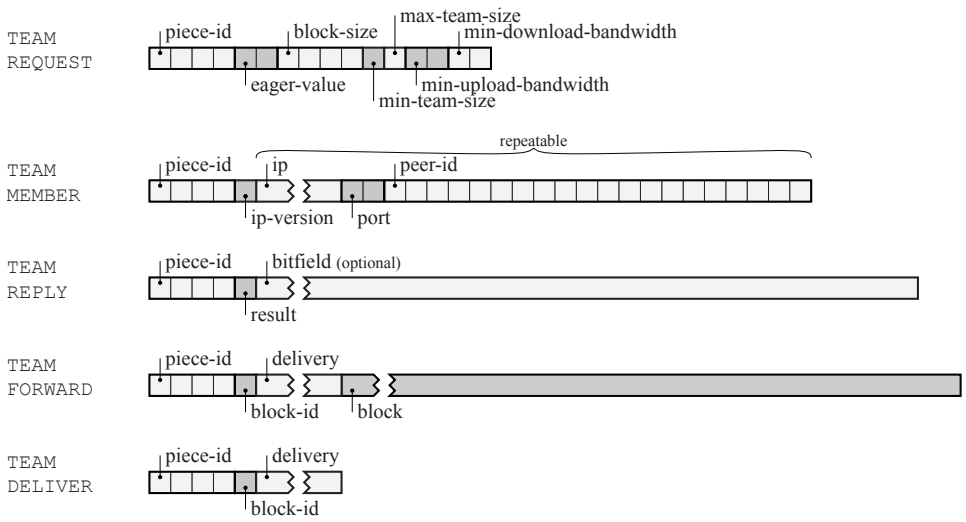


Figure A.3: *The messages and their payload used by the Extended Supervised Teaming protocol.*

consists of four bytes indicating the piece that will be transferred. The `eager-value` consists of two bytes indicating how eager a peer is to supervise this team. The first byte can be based on available bandwidth while the second byte should be chosen randomly. The `block-size` consists of four bytes indicating the number of bytes in each block. The `min-team-size` and `max-team-size` both consist of one byte indicating the minimum and maximum number of peers in the team respectively. The `min-upload-bandwidth` and `min-download-bandwidth` both consists of two bytes indicating the minimum upload bandwidth and minimum download bandwidth that the peer must have available for the team.

`TEAM MEMBER`: `piece-id`, `ip-version`, `[ip, port, peer-id]+`

The `TEAM MEMBER` message is used to notify potential team members, that replied positively to a `TEAM REQUEST`, who her team members will be. The `piece-id` consists of four bytes indicating the piece that will be transferred. The `ip-version` consists of one byte indicating either the IPv4 or IPv6 protocol. When using the IPv4 protocol the `ip`, `port`, and `peer-id` consists of four, two and twenty bytes respectively. When using the IPv6 protocol these items are sixteen, two and twenty bytes respectively. The `ip`, `port`, and `peer-id` represent the other peers that will be part of the team.

`TEAM REPLY`: `piece-id`, `result`, `bitfield`

The `TEAM REPLY` message is used in several situations. It is used by a potential team member in reply to the `TEAM REQUEST` and `TEAM MEMBER` messages. It is used by a team member to tell the supervisor that the node wishes to leave the team prematurely. And it is used by the supervisor to tell team members that the team will be disbanded prematurely. The `piece-id` consists of four bytes indicating

the piece for which the message is intended. The `result` consists of one byte and can indicate different things depending on the situation in which the message is used, these specific values will not be discussed here. The `bitfield`, which is only used when replying to a `TEAM MEMBER` message, consists of a variable number of bytes where each bit indicates whether the peer already has a block or not. Sending a shorter `bitfield` or none at all is allowed. Any blocks that are not indicated in the `bitfield` are believed to be required.

`TEAM FORWARD`: `piece-id`, `block-id`, `delivery`, `data`

The `TEAM FORWARD` message is used to send a block of data from supervisor to team member and from team member to team member. The `piece-id` consists of four bytes indicating the piece that this message belongs to. The `block-id` consists of one byte indicating the randomly chosen identifier for this block. The `delivery` consists of a variable number of bytes depending on the maximum team size, and indicates the team members to where the data must be forwarded. The bit for the team member that is forwarding is also set to indicate that she wants to receive the associated `TEAM REWARD` messages. And the `data` consists of a variable number of bytes containing the data for this block.

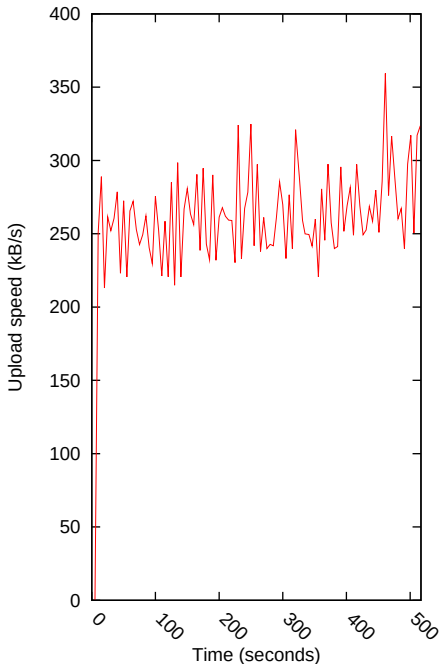
`TEAM DELIVER`: `piece-id`, `block-id`, `delivery`

The `TEAM DELIVER` message is sent by the supervisor to tell a team member to forward a block of already downloaded data. The `piece-id`, `block-id`, and `delivery` are the same as in the `TEAM FORWARD` message with the exception that in the `delivery` the bit for the team member that is forwarding is not set since she does not require the `TEAM REWARD` messages.

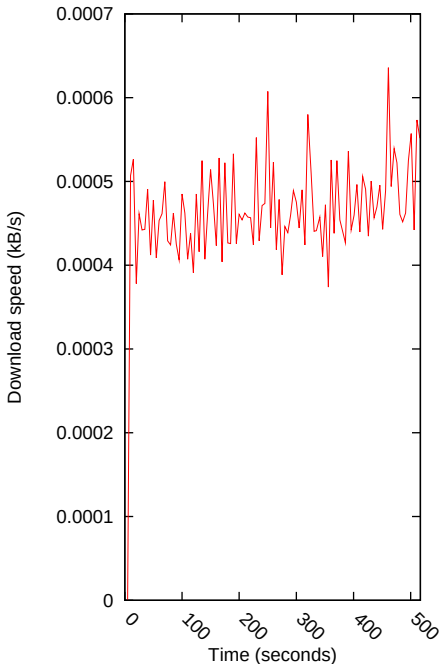
Appendix B

Detailed experiment results

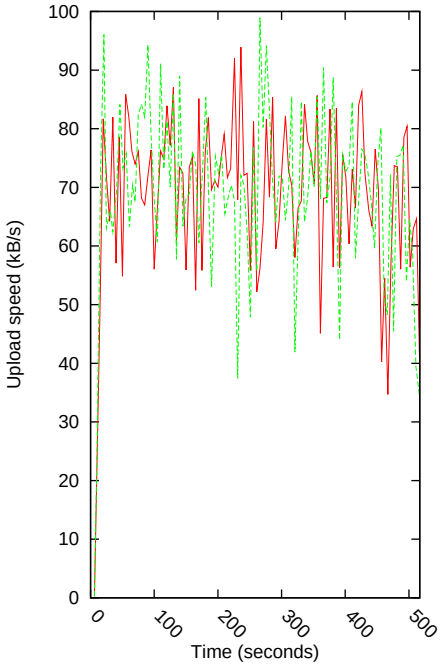
This appendix contains graphs displaying the up and download speed for all peers during the experiments that are described in Chapter 5. The statistics for the flash crowd, free riding, and initial risk experiments are combined in Figure 5.3, 5.4, and 5.6, respectively.



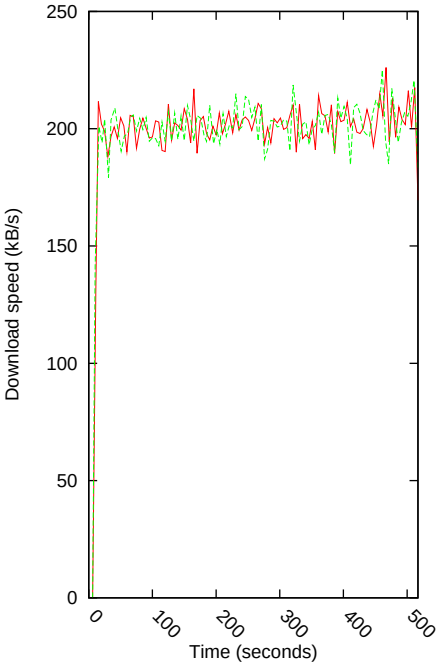
(a) Initial seeder upload speed



(b) Initial seeder download speed

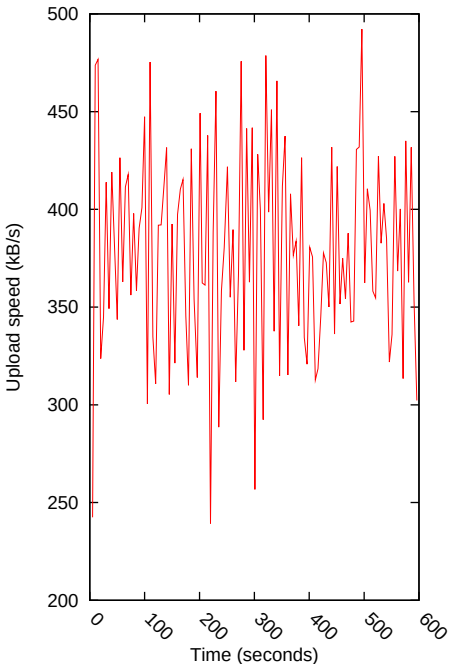


(c) Leecher upload speed

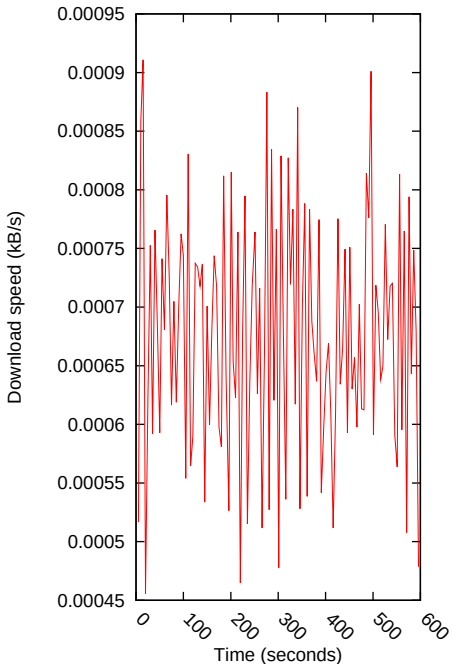


(d) Leecher download speed

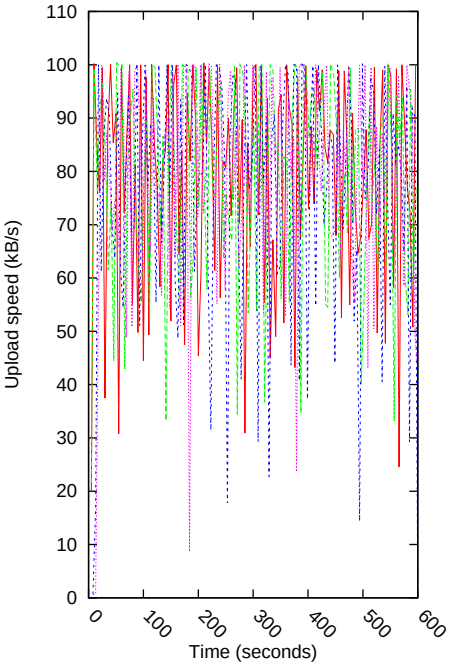
Figure B.1: *Flash crowd experiment* `1-2-0` *using BitTorrent*



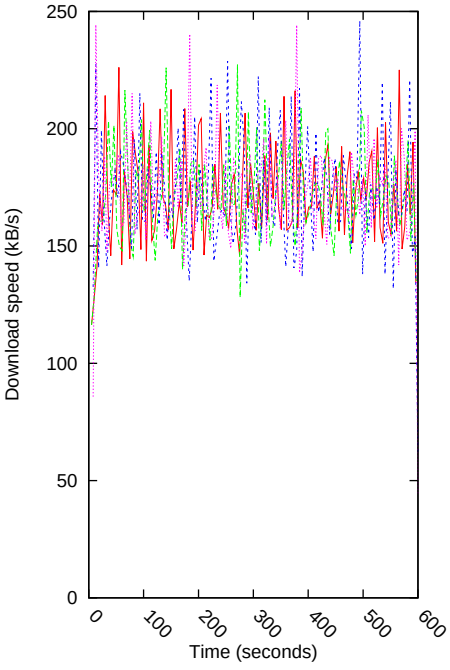
(a) Initial seeder upload speed



(b) Initial seeder download speed

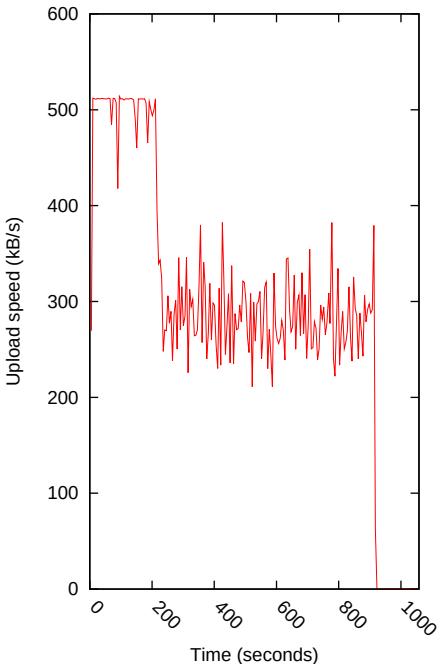


(c) Leecher upload speed

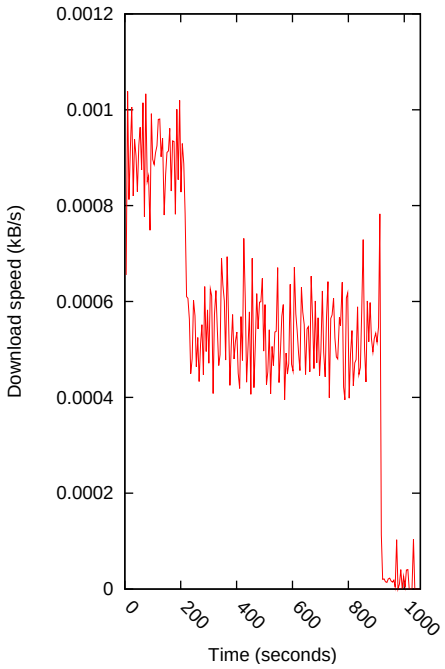


(d) Leecher download speed

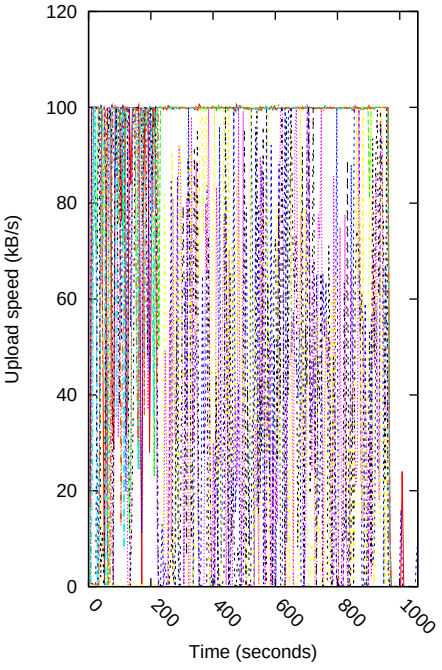
Figure B.2: *Flash crowd experiment* `1-4-0` *using BitTorrent*



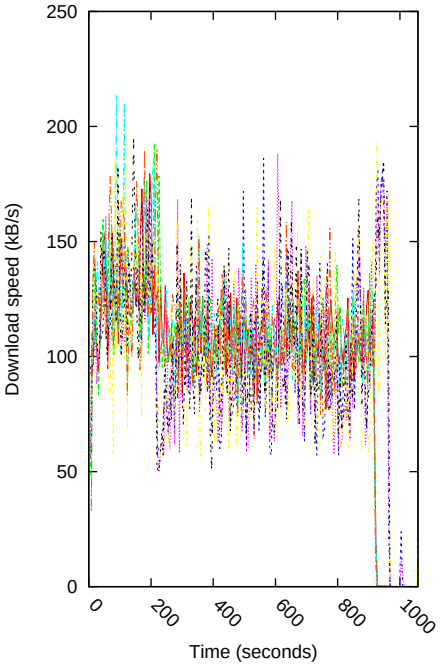
(a) Initial seeder upload speed



(b) Initial seeder download speed

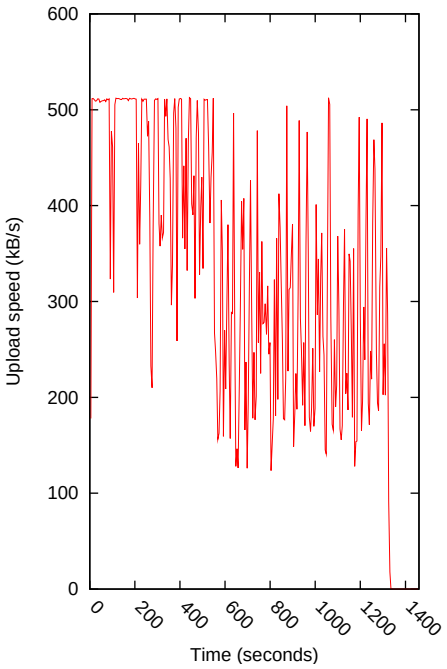


(c) Leecher upload speed

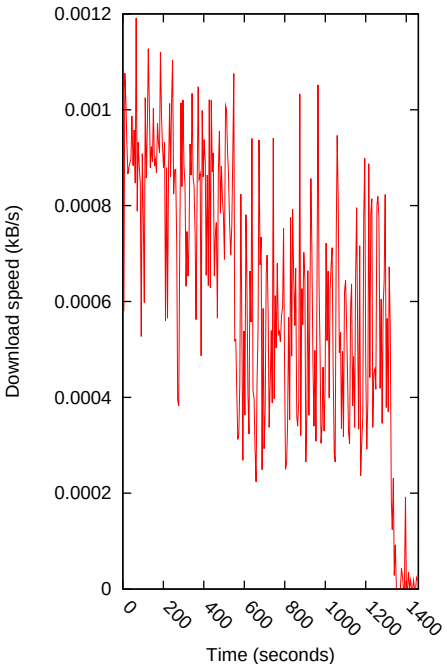


(d) Leecher download speed

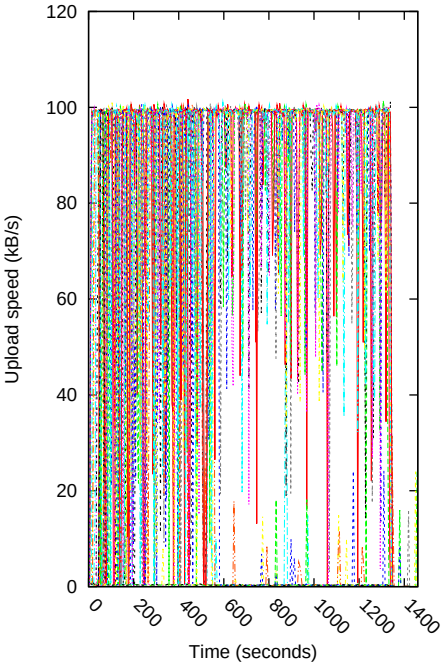
Figure B.3: *Flash crowd experiment* `1-8-0` *using BitTorrent*



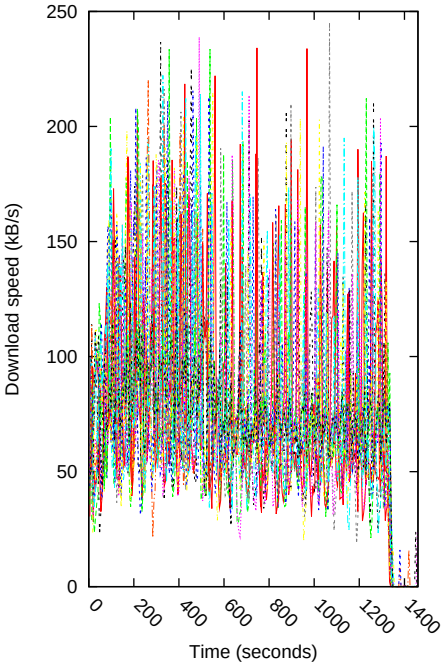
(a) Initial seeder upload speed



(b) Initial seeder download speed

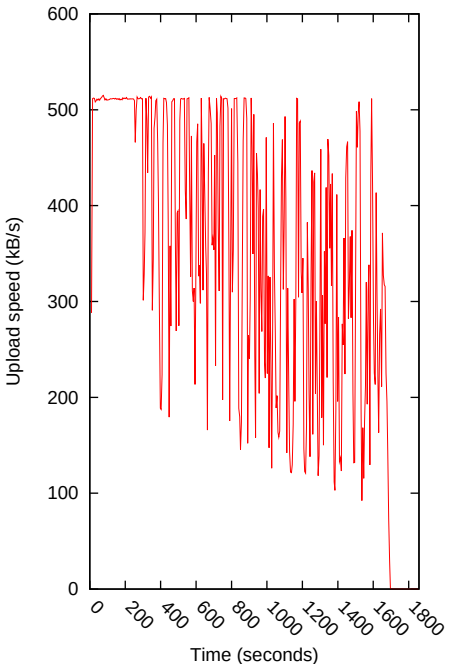


(c) Leecher upload speed

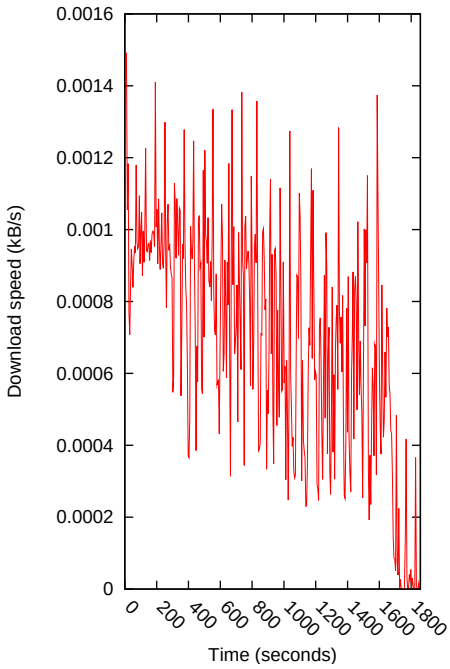


(d) Leecher download speed

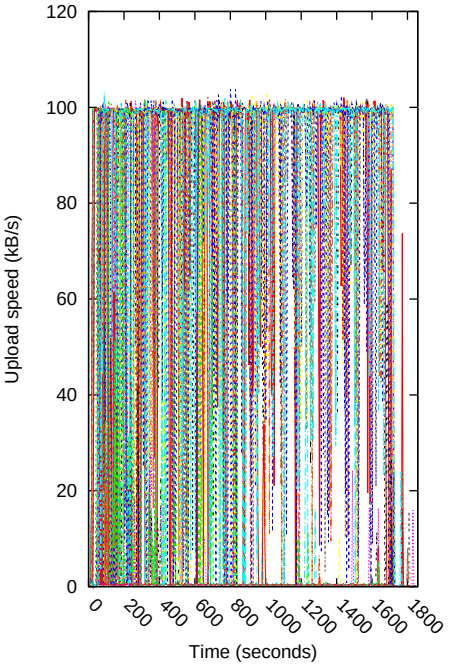
Figure B.4: *Flash crowd experiment* `1-16-0` *using BitTorrent*



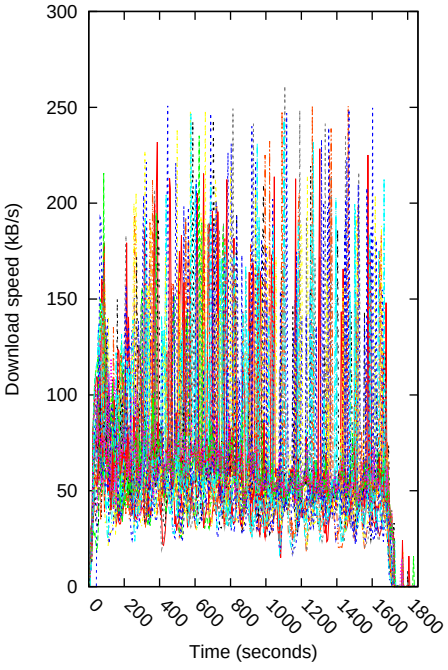
(a) Initial seeder upload speed



(b) Initial seeder download speed

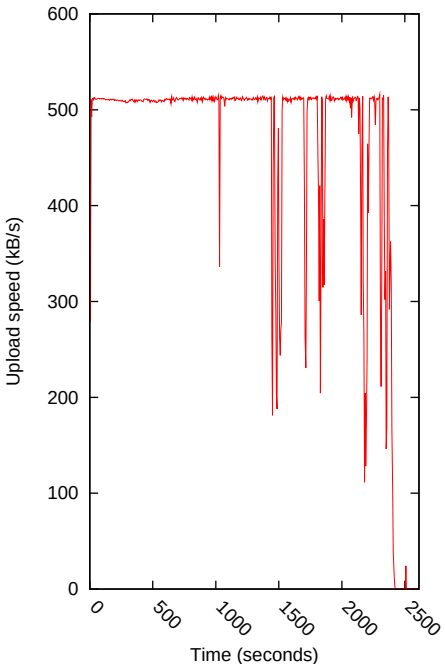


(c) Leecher upload speed

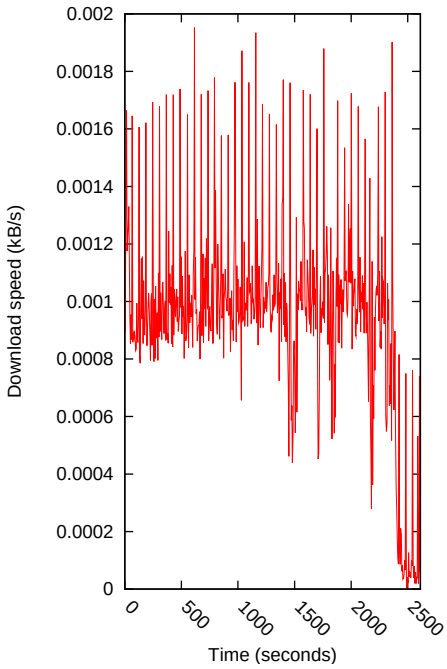


(d) Leecher download speed

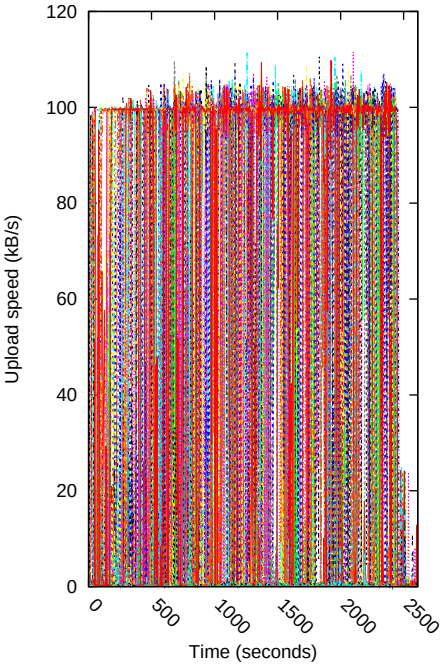
Figure B.5: *Flash crowd experiment* `1-32-0` *using BitTorrent*



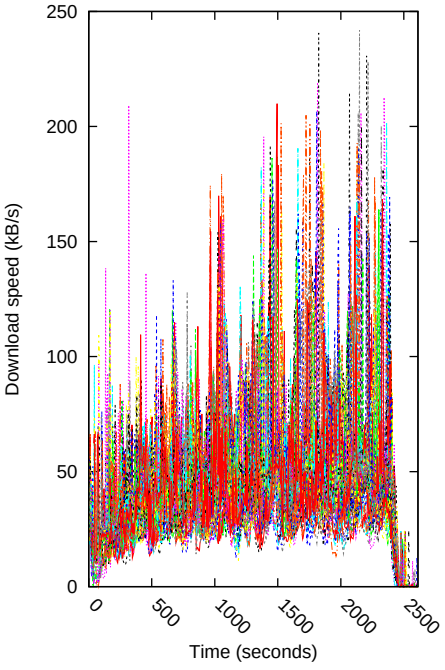
(a) Initial seeder upload speed



(b) Initial seeder download speed

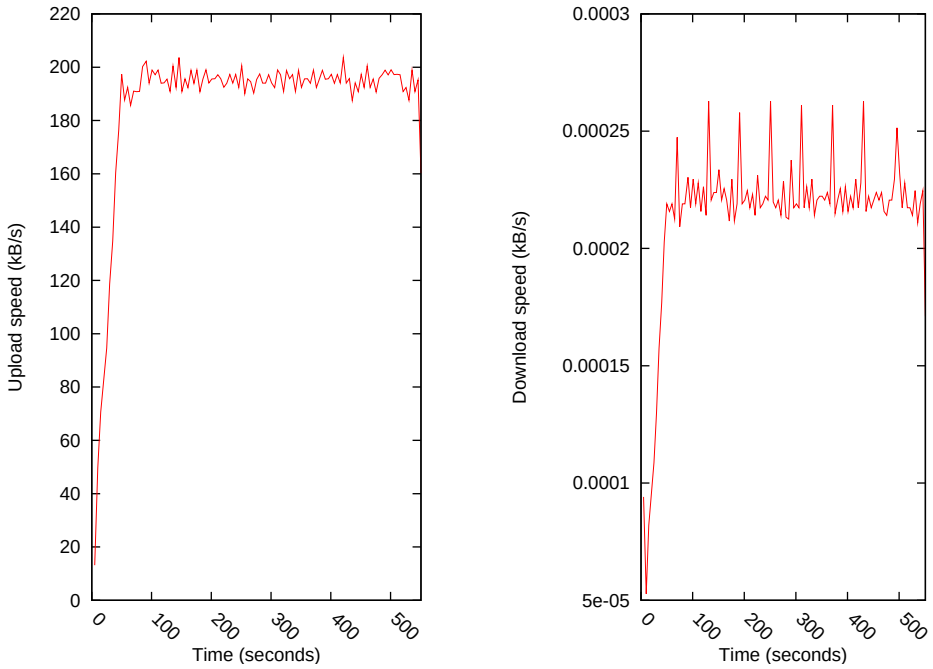


(c) Leecher upload speed



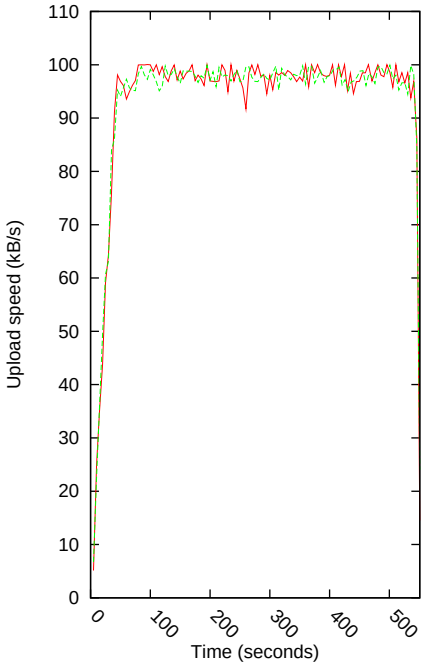
(d) Leecher download speed

Figure B.6: *Flash crowd experiment* `1-64-0` *using BitTorrent*

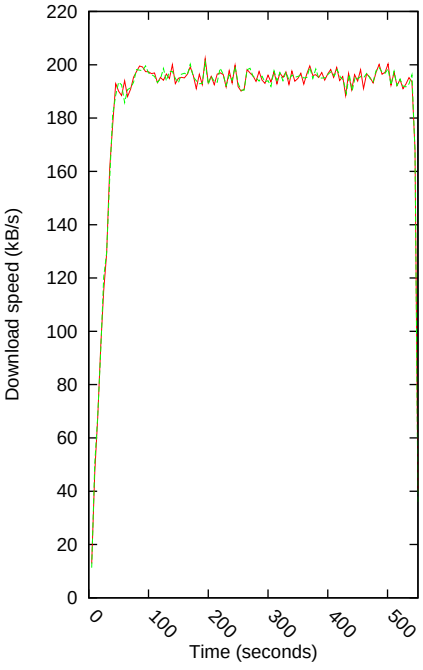


(a) Initial seeder upload speed

(b) Initial seeder download speed

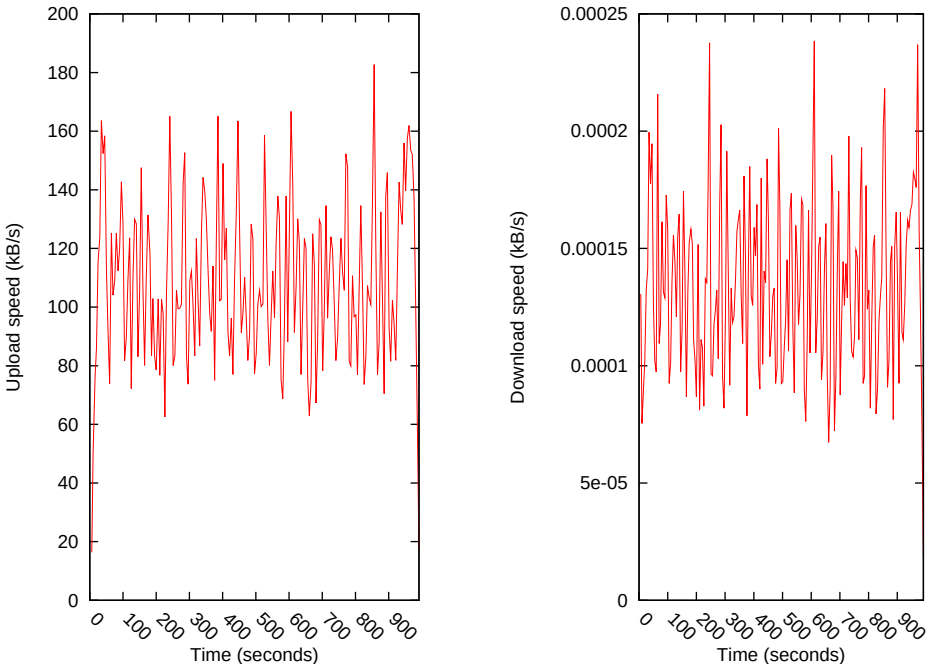


(c) Leecher upload speed



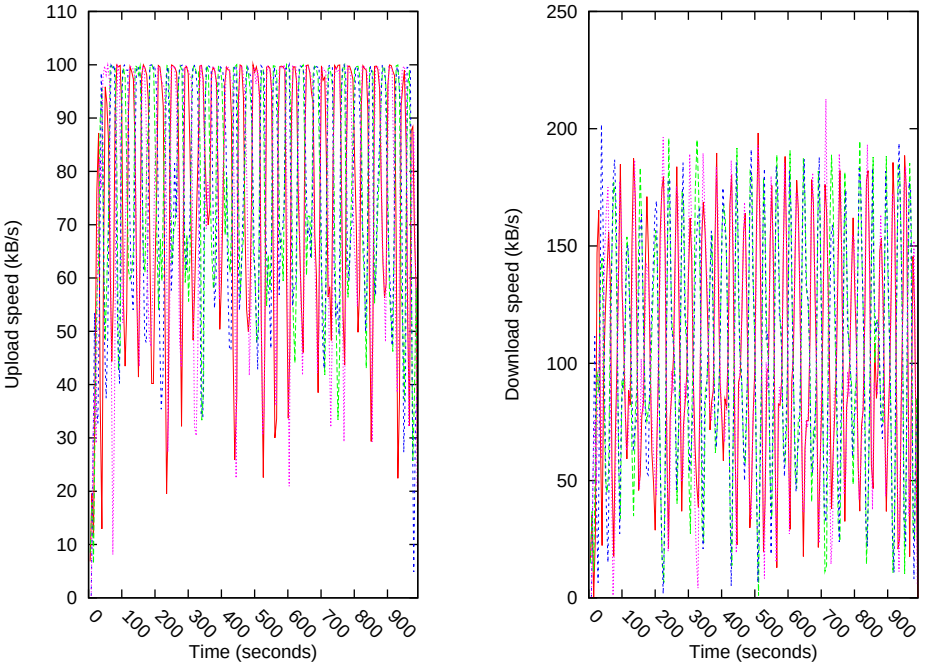
(d) Leecher download speed

Figure B.7: *Flash crowd experiment* `1-2-0` *using Light Supervised Teaming*



(a) Initial seeder upload speed

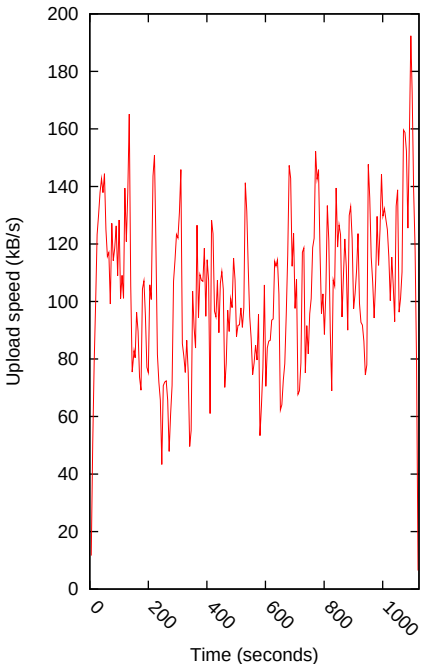
(b) Initial seeder download speed



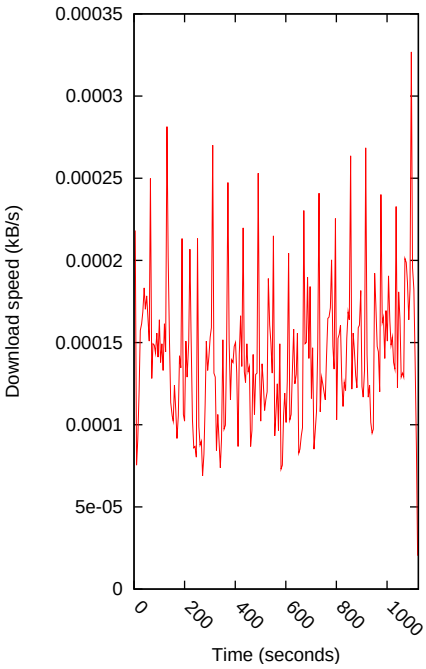
(c) Leecher upload speed

(d) Leecher download speed

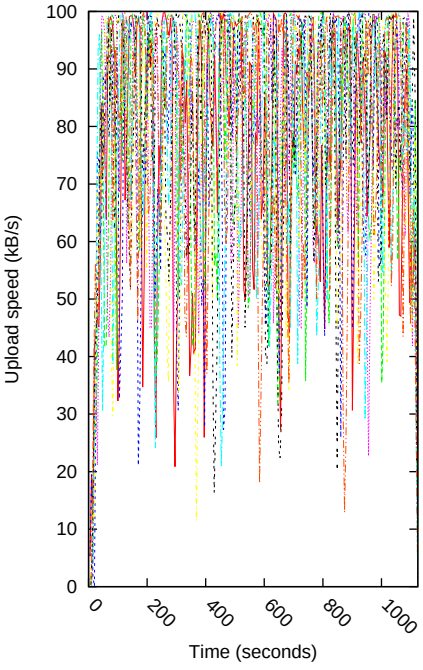
Figure B.8: *Flash crowd experiment* `1-4-0` *using Light Supervised Teaming*



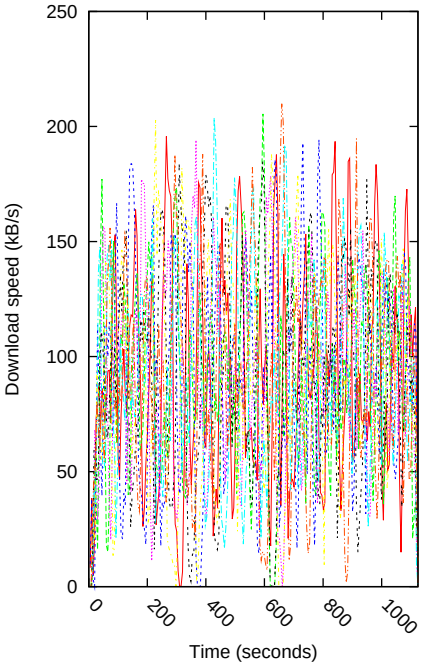
(a) Initial seeder upload speed



(b) Initial seeder download speed

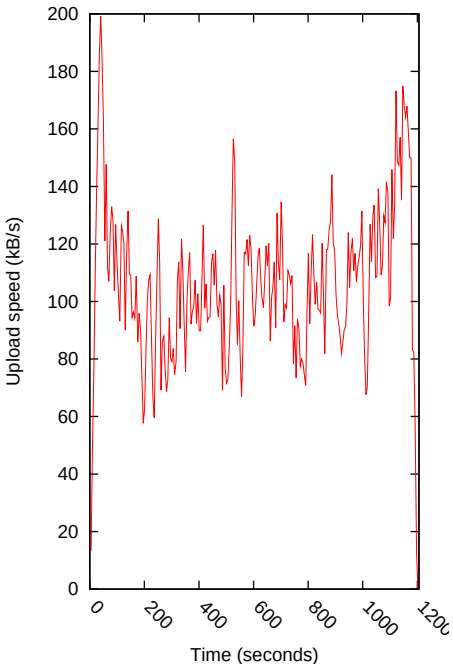


(c) Leecher upload speed

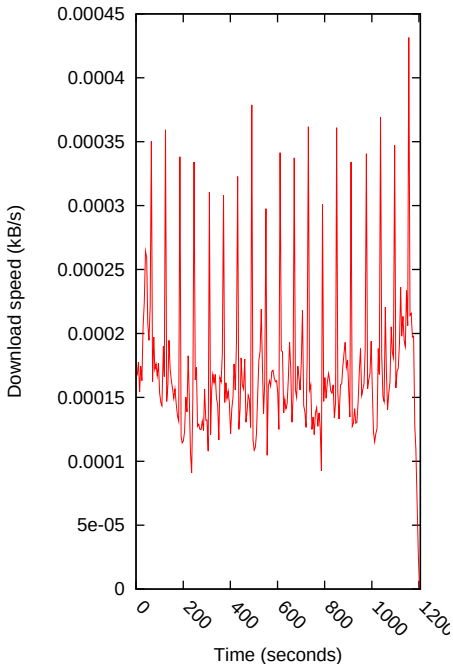


(d) Leecher download speed

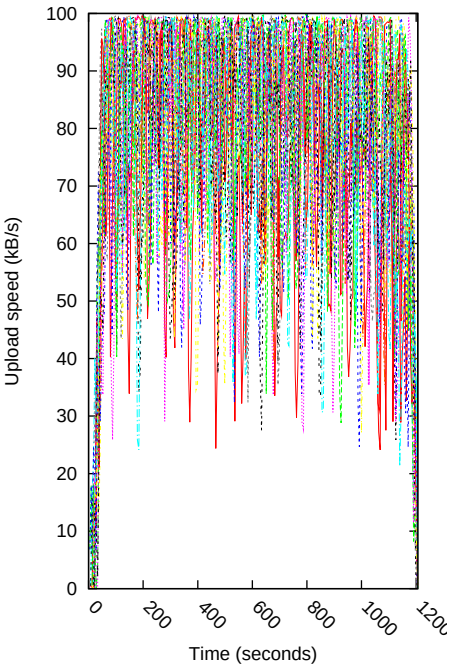
Figure B.9: *Flash crowd experiment* `1-8-0` *using Light Supervised Teaming*



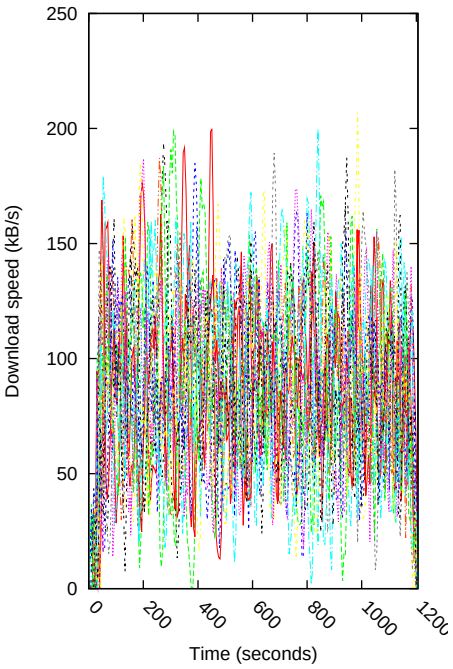
(a) Initial seeder upload speed



(b) Initial seeder download speed

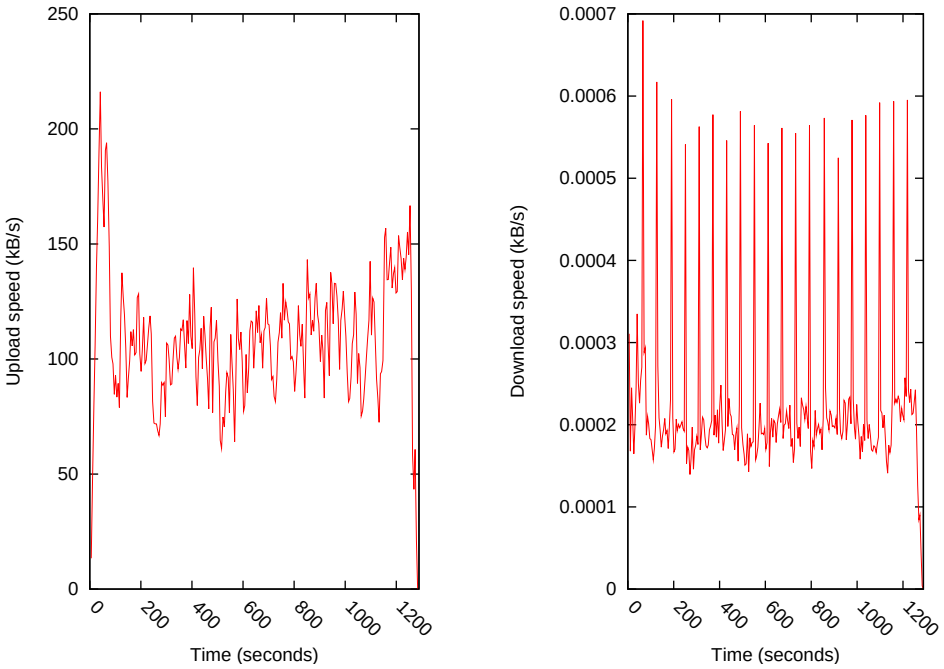


(c) Leecher upload speed



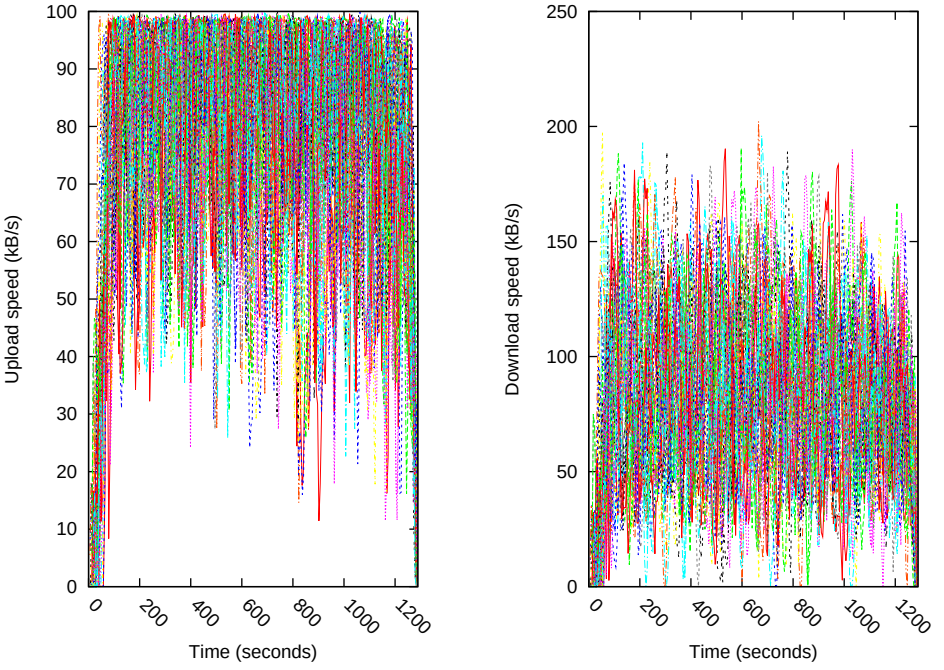
(d) Leecher download speed

Figure B.10: *Flash crowd experiment* `1-16-0` *using Light Supervised Teaming*



(a) Initial seeder upload speed

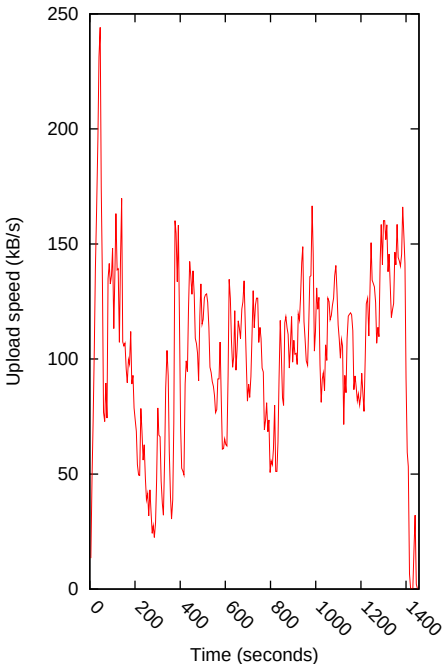
(b) Initial seeder download speed



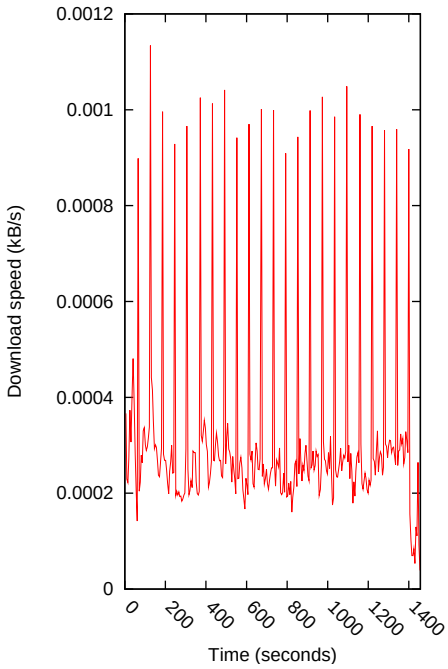
(c) Leecher upload speed

(d) Leecher download speed

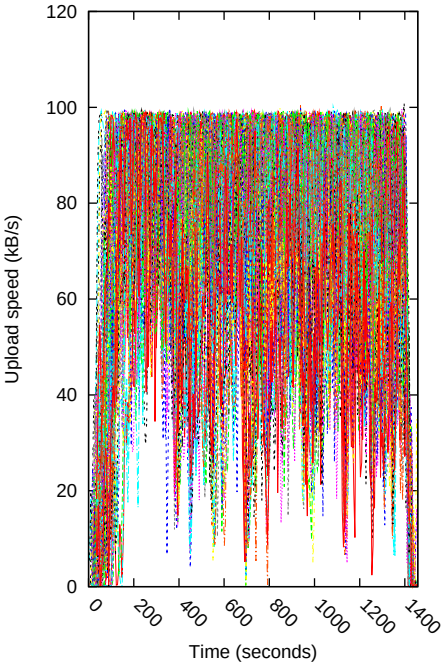
Figure B.11: *Flash crowd experiment* `1-32-0` *using Light Supervised Teaming*



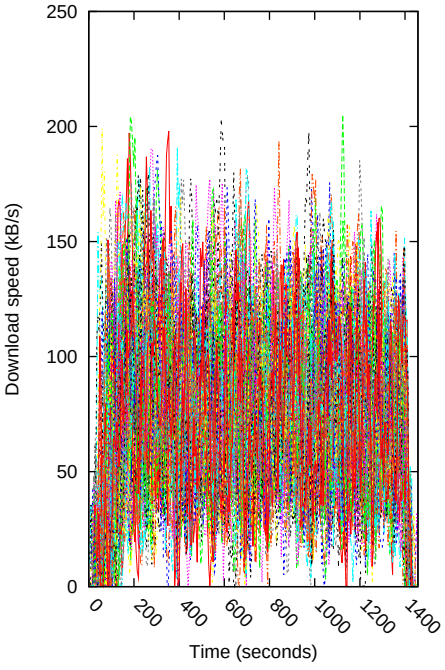
(a) Initial seeder upload speed



(b) Initial seeder download speed

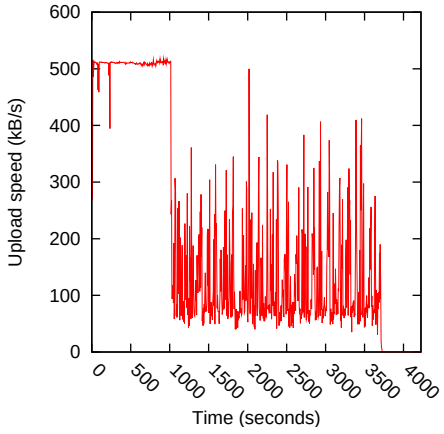


(c) Leecher upload speed

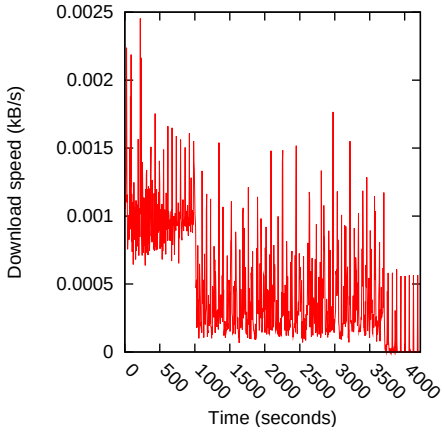


(d) Leecher download speed

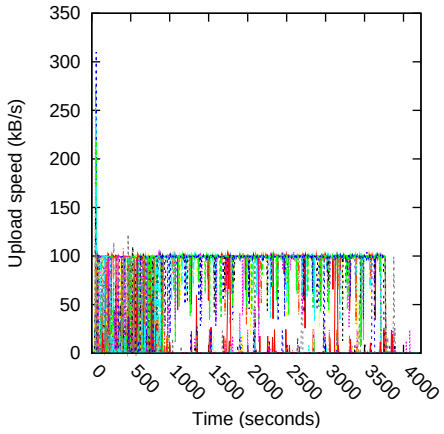
Figure B.12: *Flash crowd experiment* `1-64-0` *using Light Supervised Teaming*



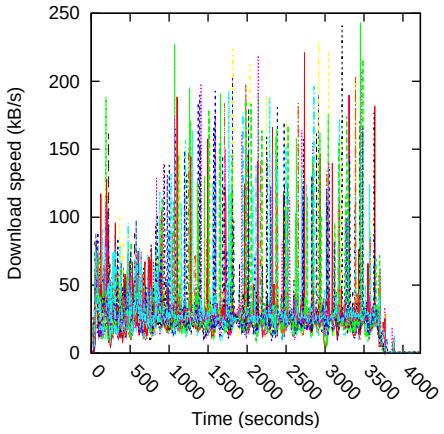
(a) Initial seeder upload speed



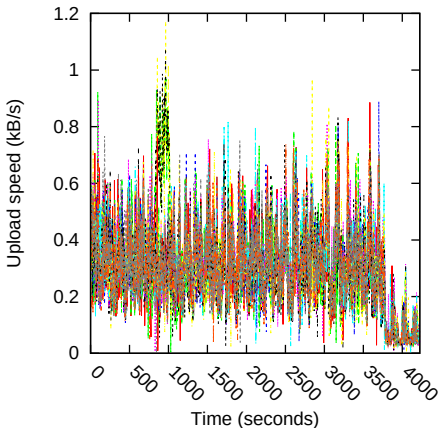
(b) Initial seeder download speed



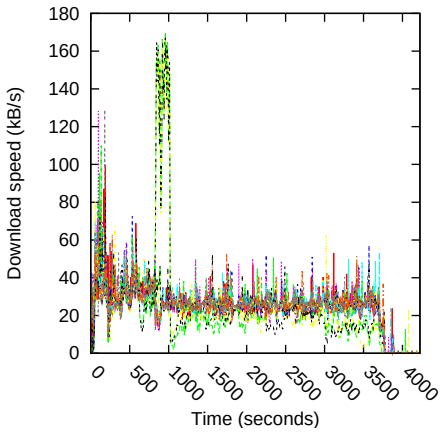
(c) Leecher upload speed



(d) Leecher download speed

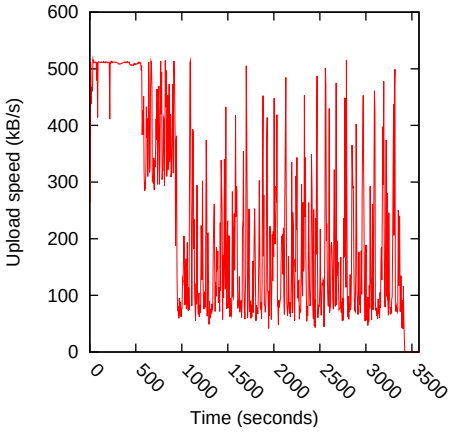


(e) Free rider upload speed

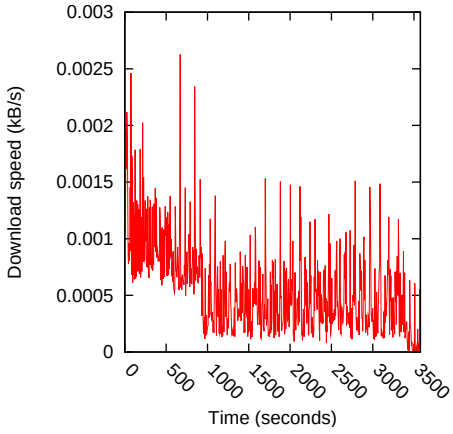


(f) Free rider download speed

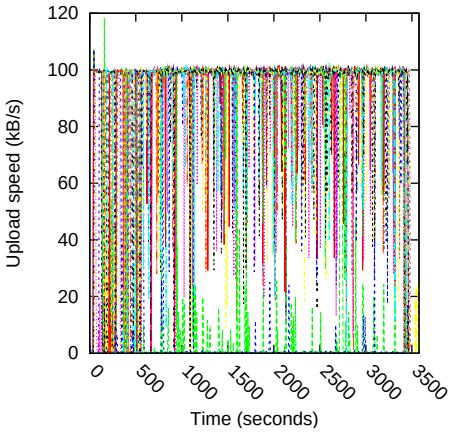
Figure B.13: *Free riding experiment* `1-32-18` *using BitTorrent*



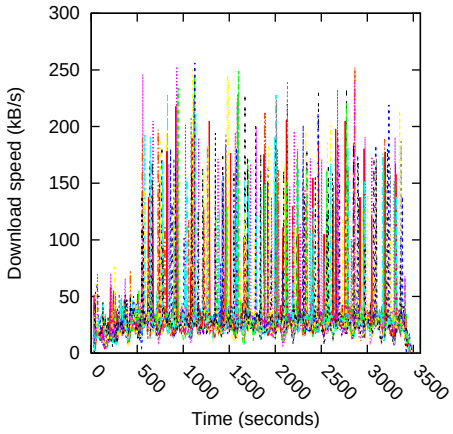
(a) Initial seeder upload speed



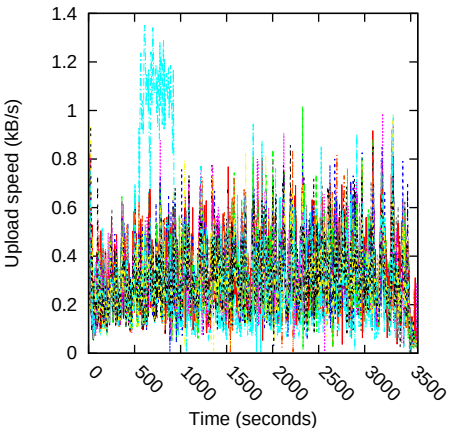
(b) Initial seeder download speed



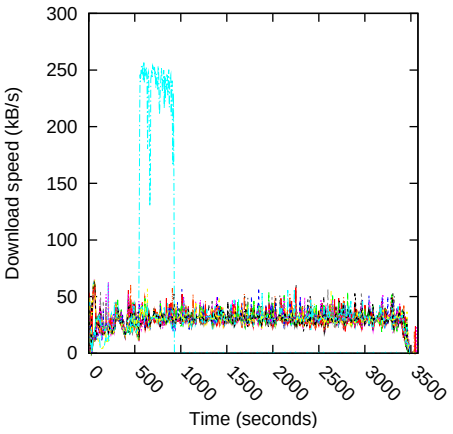
(c) Leecher upload speed



(d) Leecher download speed

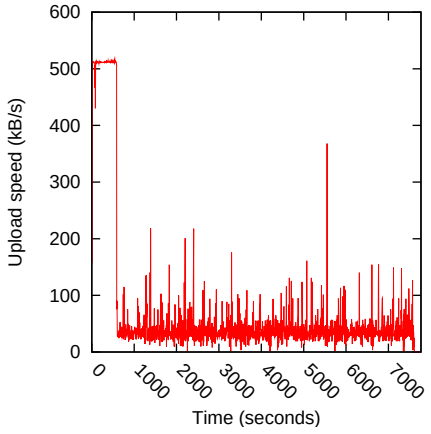


(e) Free rider upload speed

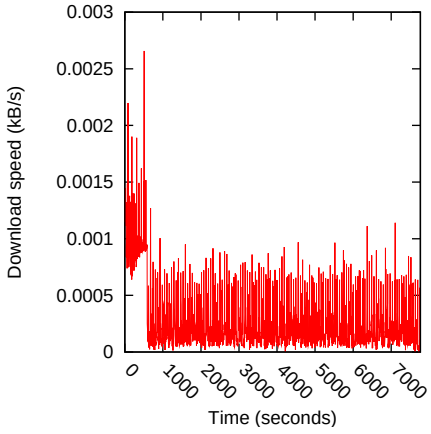


(f) Free rider download speed

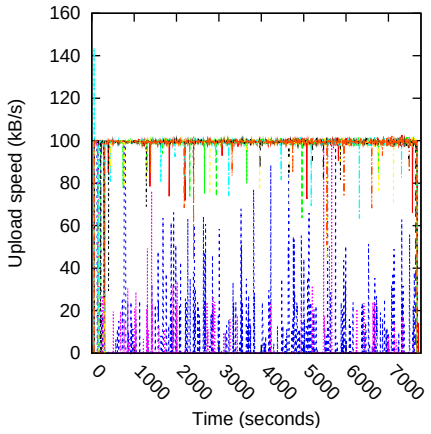
Figure B.14: *Free riding experiment* `1-16-34` *using BitTorrent*



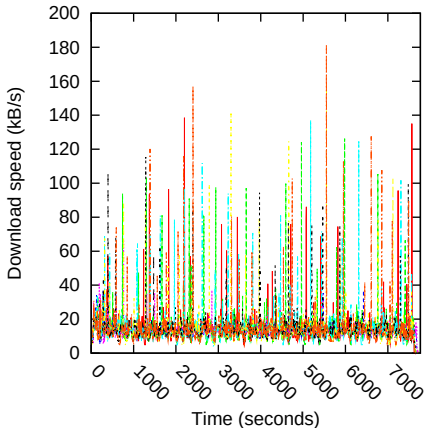
(a) Initial seeder upload speed



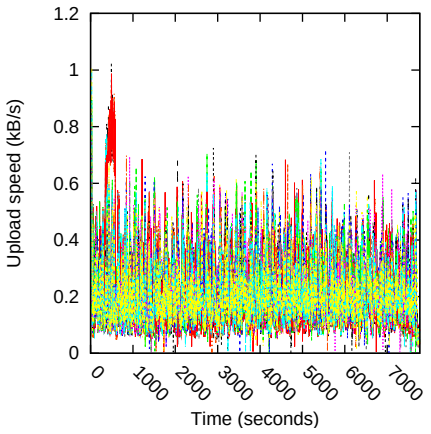
(b) Initial seeder download speed



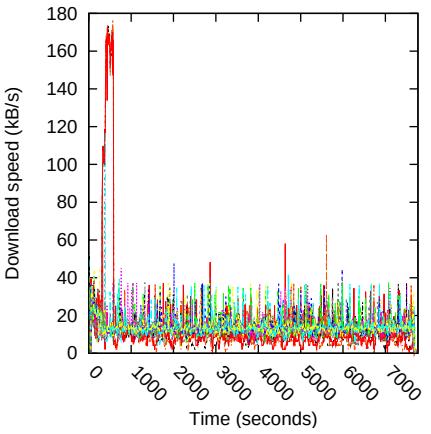
(c) Leecher upload speed



(d) Leecher download speed

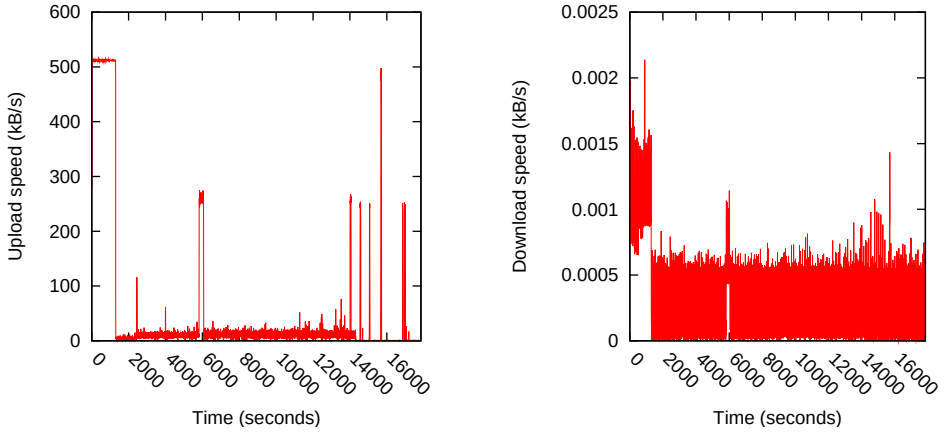


(e) Free rider upload speed



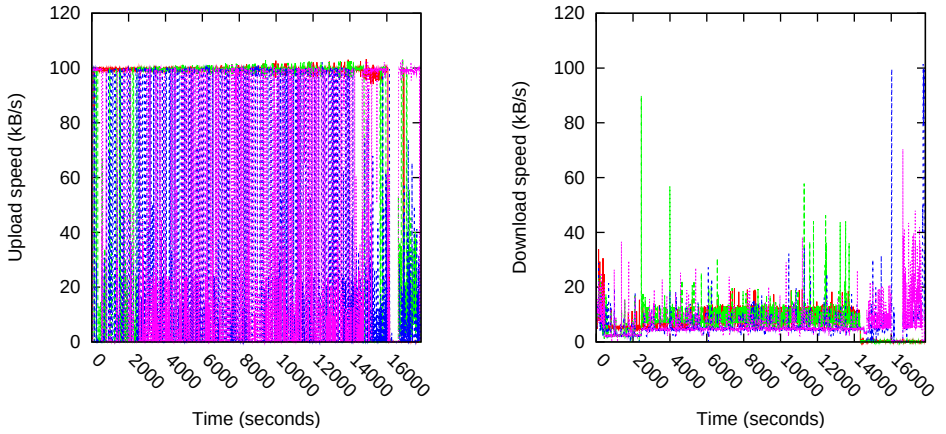
(f) Free rider download speed

Figure B.15: *Free riding experiment* `1-8-42` *using BitTorrent*



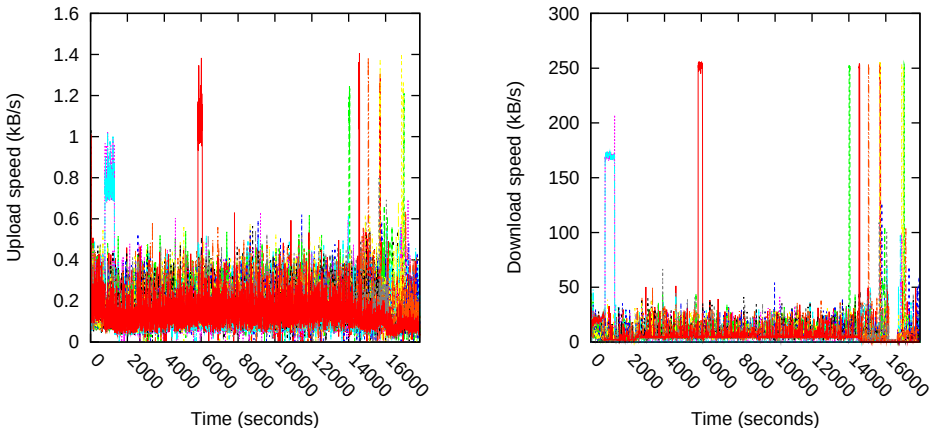
(a) Initial seeder upload speed

(b) Initial seeder download speed



(c) Leecher upload speed

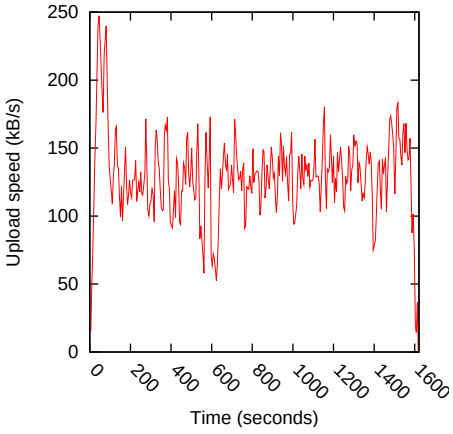
(d) Leecher download speed



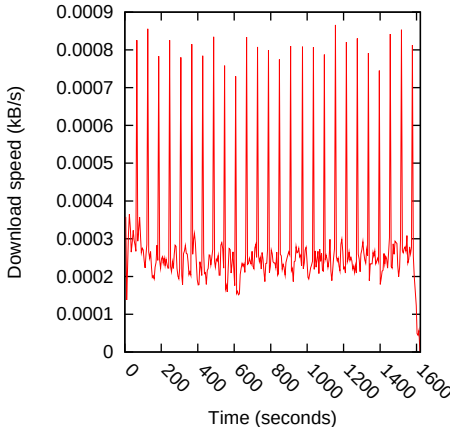
(e) Free rider upload speed

(f) Free rider download speed

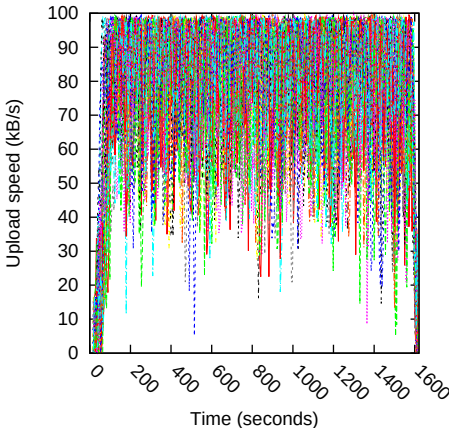
Figure B.16: *Free riding experiment* `1-4-46` *using BitTorrent*



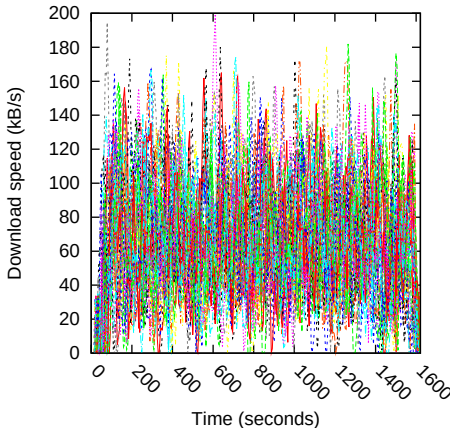
(a) Initial seeder upload speed



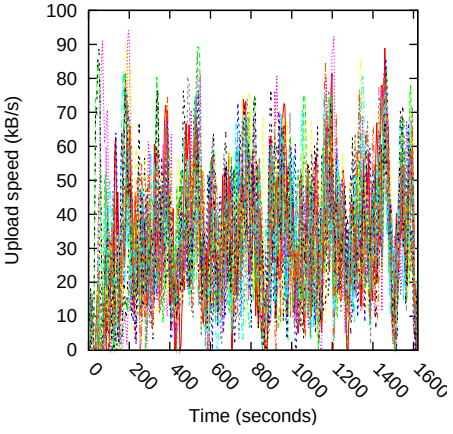
(b) Initial seeder download speed



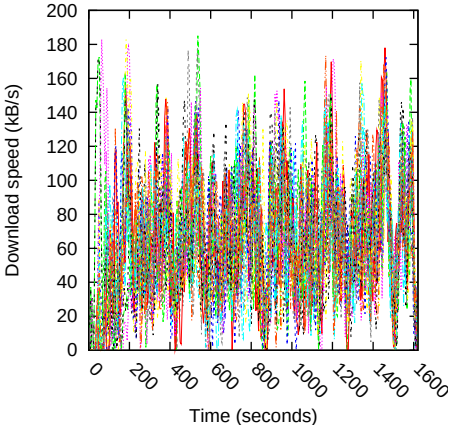
(c) Leecher upload speed



(d) Leecher download speed

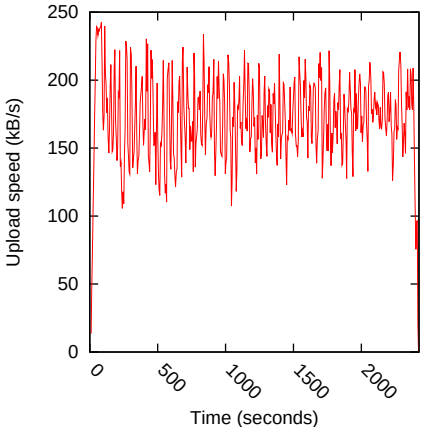


(e) Free rider upload speed

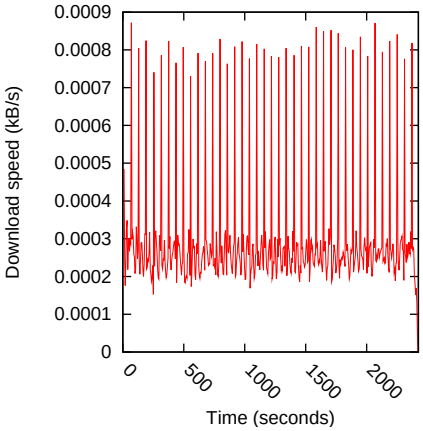


(f) Free rider download speed

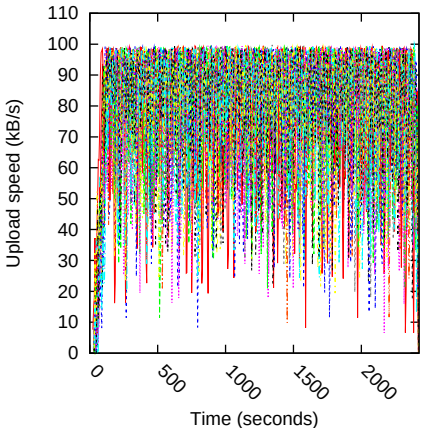
Figure B.17: *Free riding experiment* `1-32-18` *using Light Supervised Teaming*



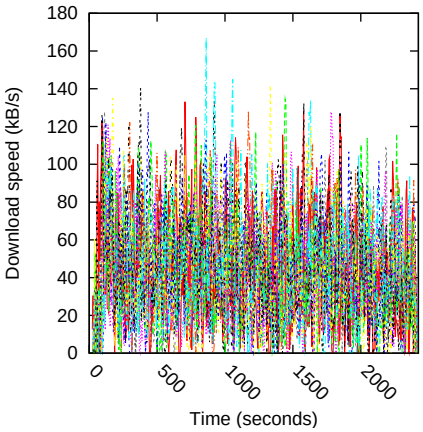
(a) Initial seeder upload speed



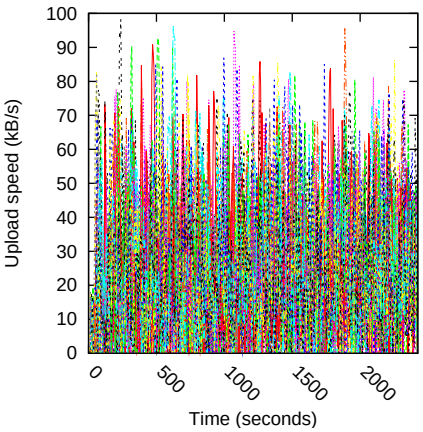
(b) Initial seeder download speed



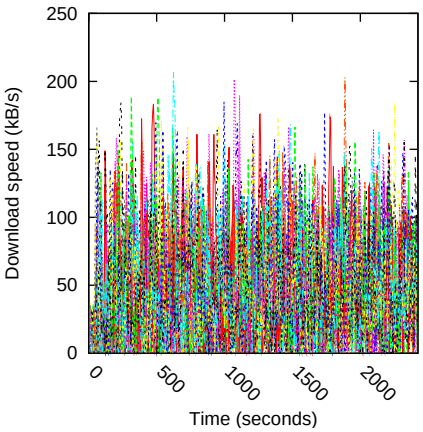
(c) Leecher upload speed



(d) Leecher download speed

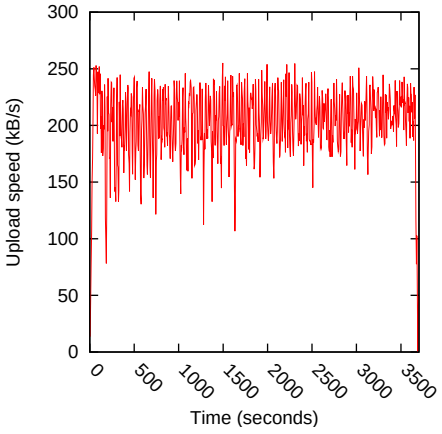


(e) Free rider upload speed

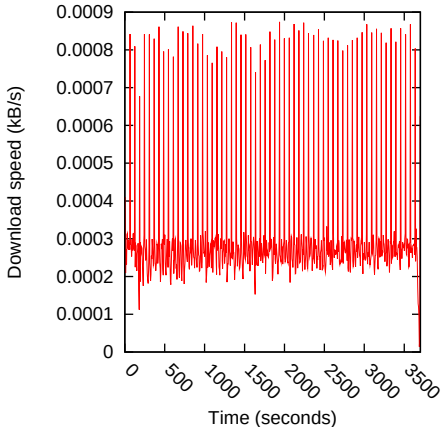


(f) Free rider download speed

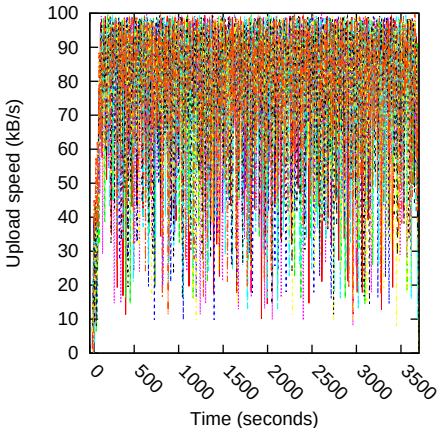
Figure B.18: *Free riding experiment* `1-16-34` *using Light Supervised Teaming*



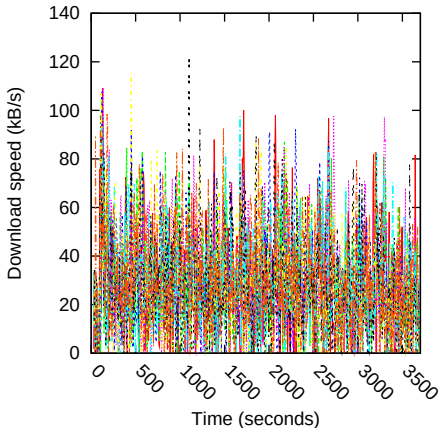
(a) Initial seeder upload speed



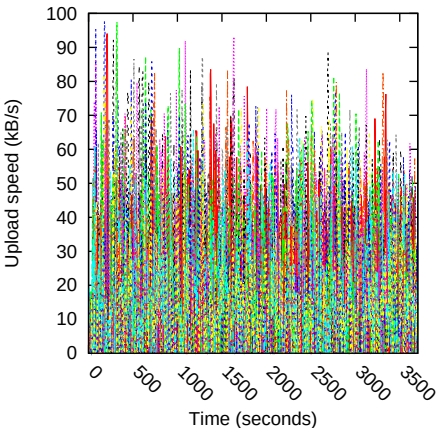
(b) Initial seeder download speed



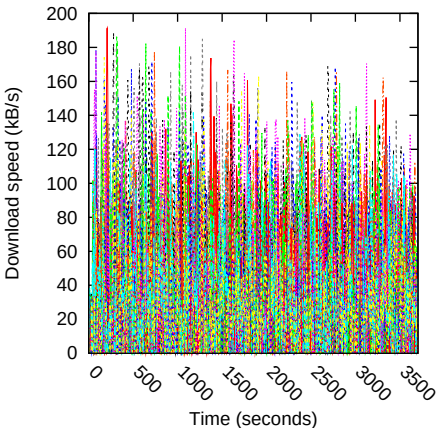
(c) Leecher upload speed



(d) Leecher download speed

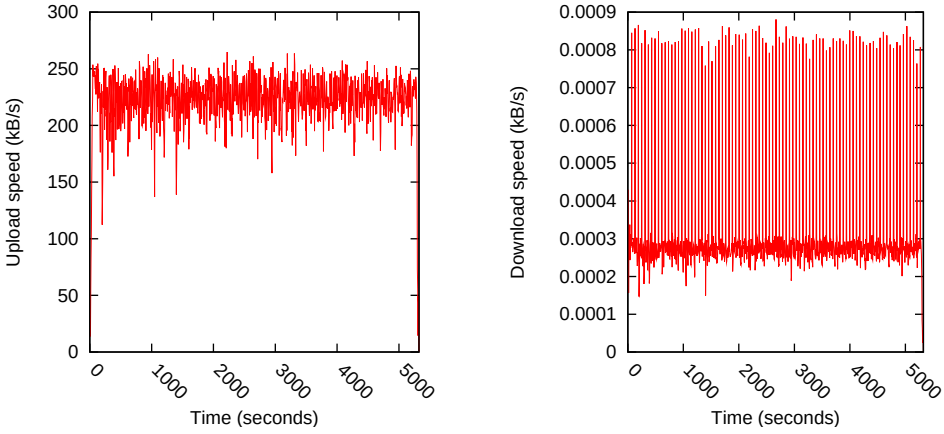


(e) Free rider upload speed



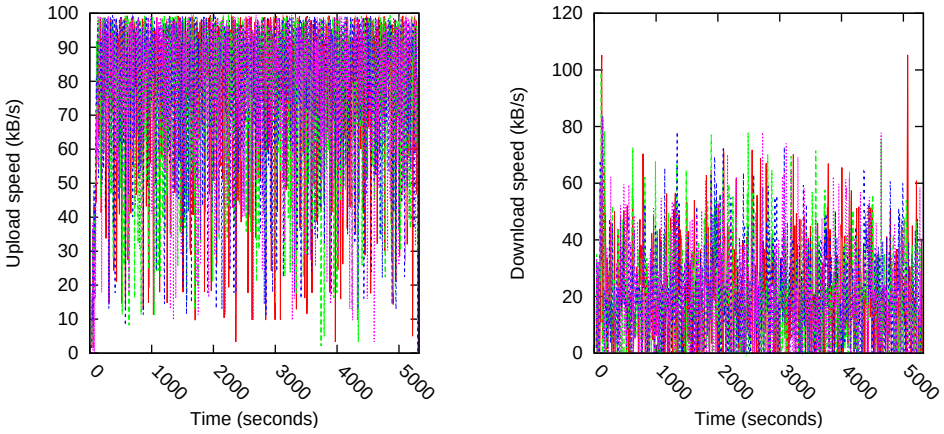
(f) Free rider download speed

Figure B.19: *Free riding experiment* `1-8-42` *using Light Supervised Teaming*



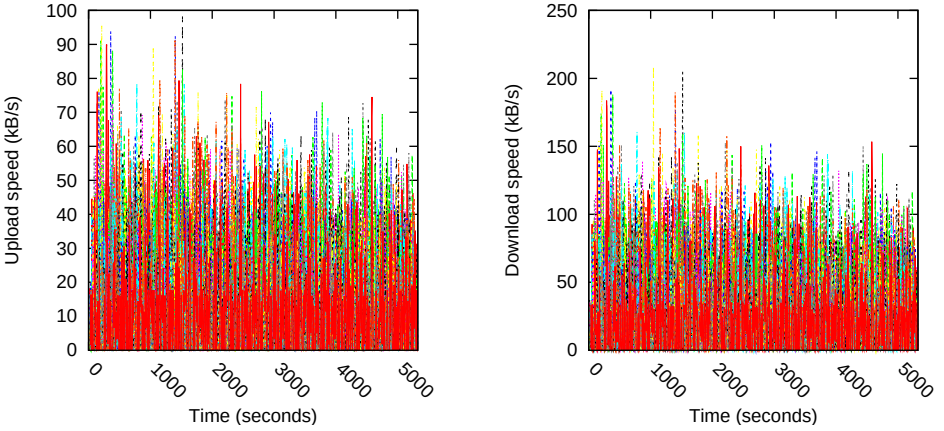
(a) Initial seeder upload speed

(b) Initial seeder download speed



(c) Leecher upload speed

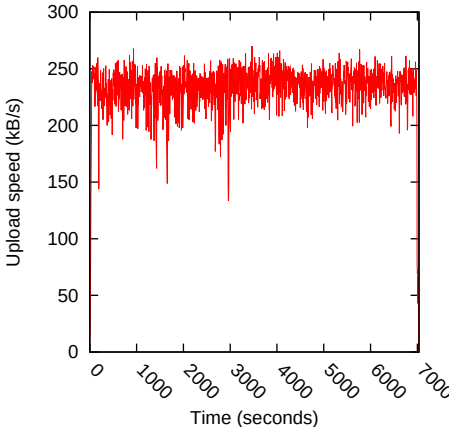
(d) Leecher download speed



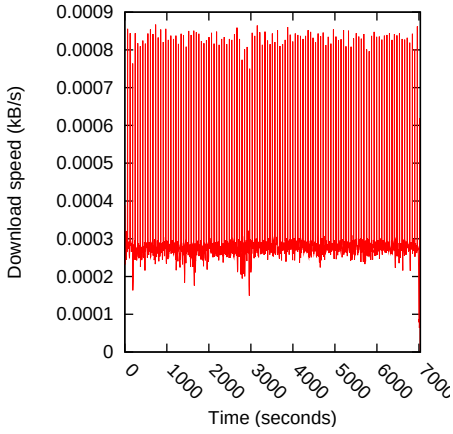
(e) Free rider upload speed

(f) Free rider download speed

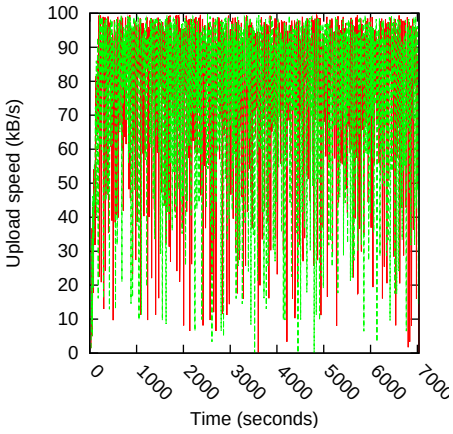
Figure B.20: *Free riding experiment* `1-4-46` *using Light Supervised Teaming*



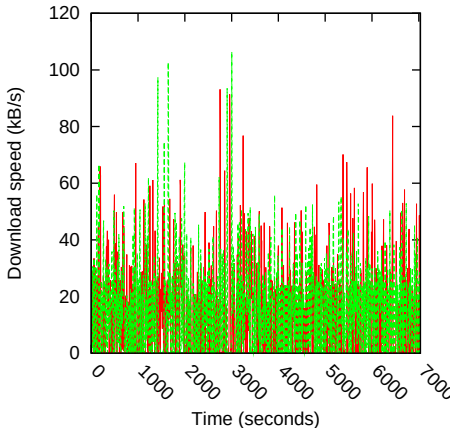
(a) Initial seeder upload speed



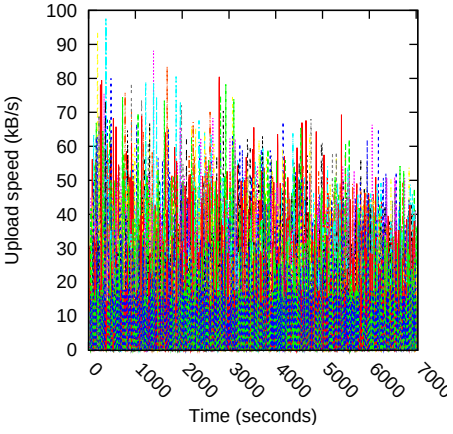
(b) Initial seeder download speed



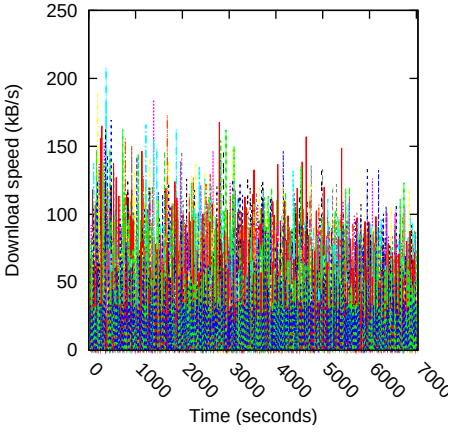
(c) Leecher upload speed



(d) Leecher download speed

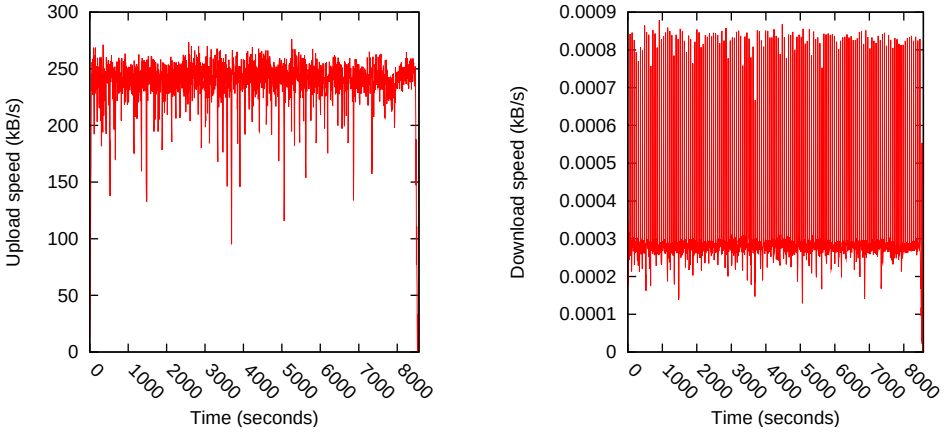


(e) Free rider upload speed



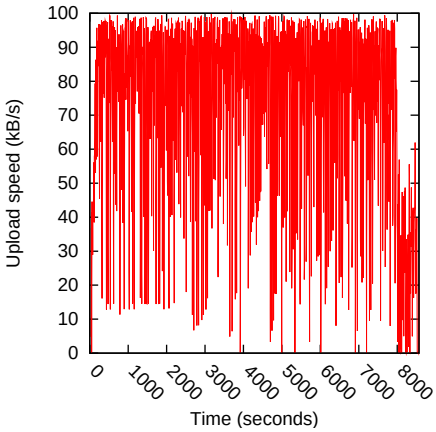
(f) Free rider download speed

Figure B.21: *Free riding experiment* `1-2-48` *using Light Supervised Teaming*

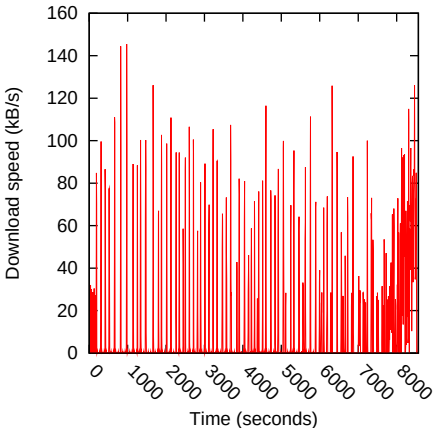


(a) Initial seeder upload speed

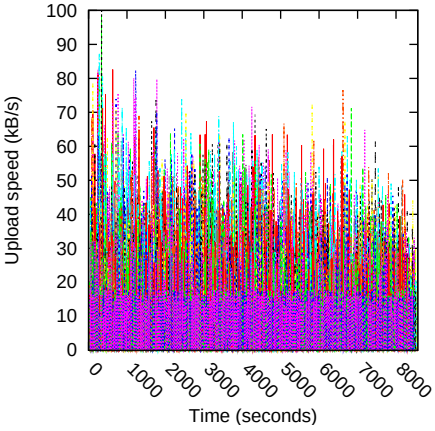
(b) Initial seeder download speed



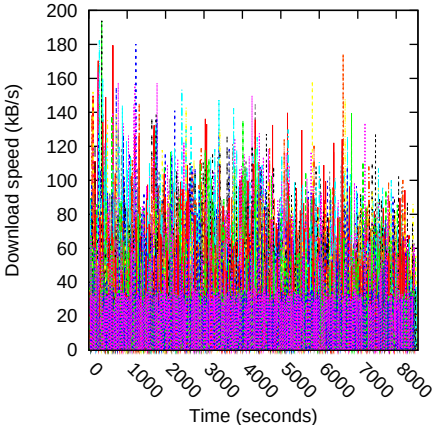
(c) Leecher upload speed



(d) Leecher download speed

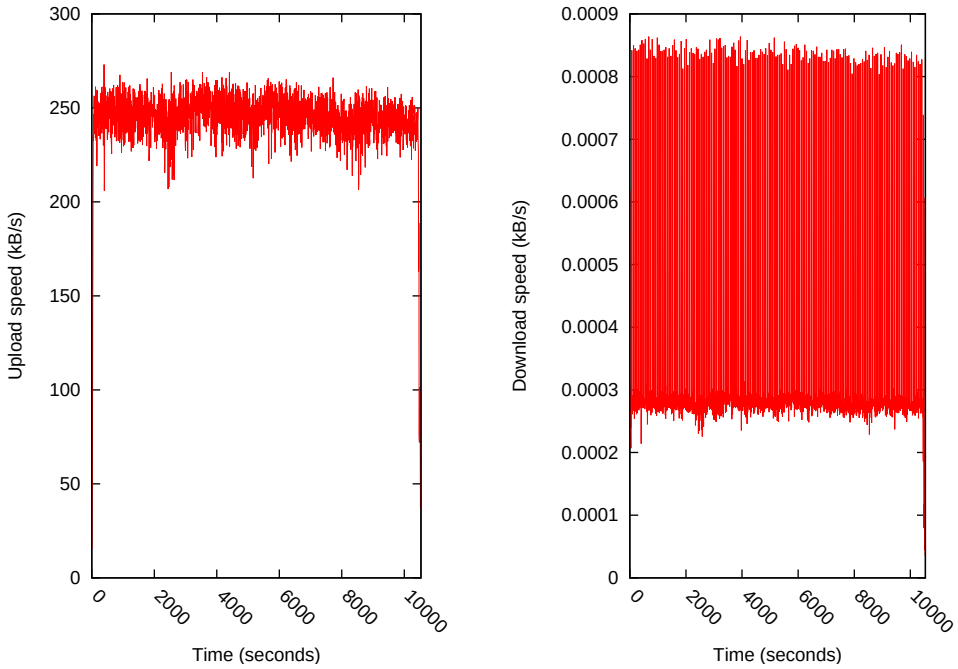


(e) Free rider upload speed



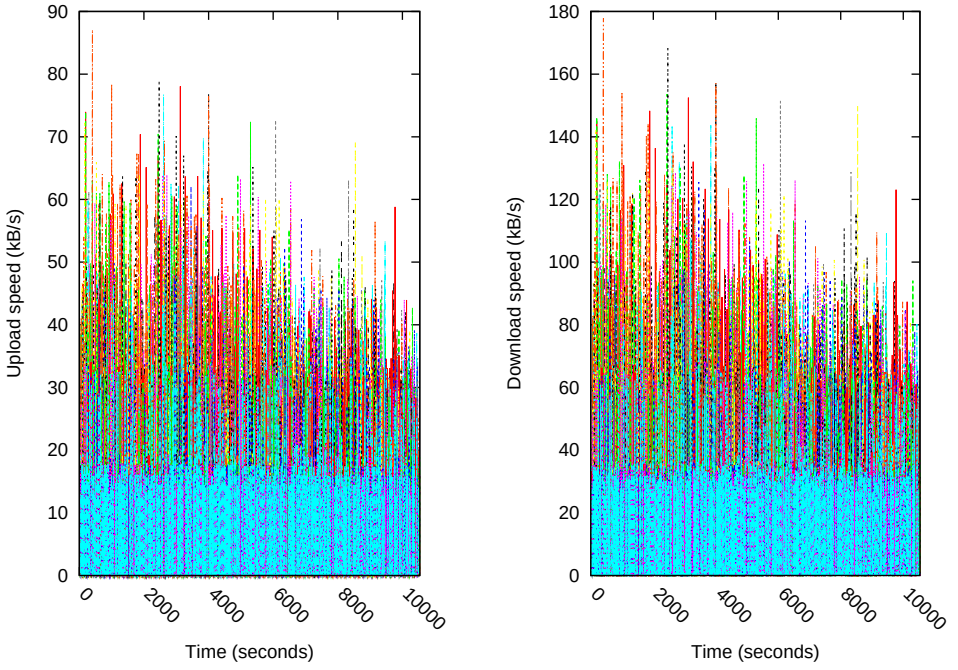
(f) Free rider download speed

Figure B.22: *Free riding experiment* `1-1-49` *using Light Supervised Teaming*



(a) Initial seeder upload speed

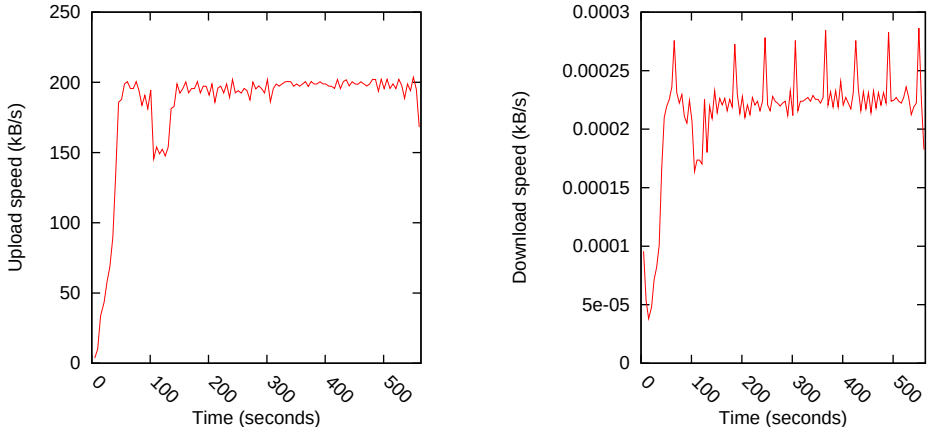
(b) Initial seeder download speed



(c) Free rider upload speed

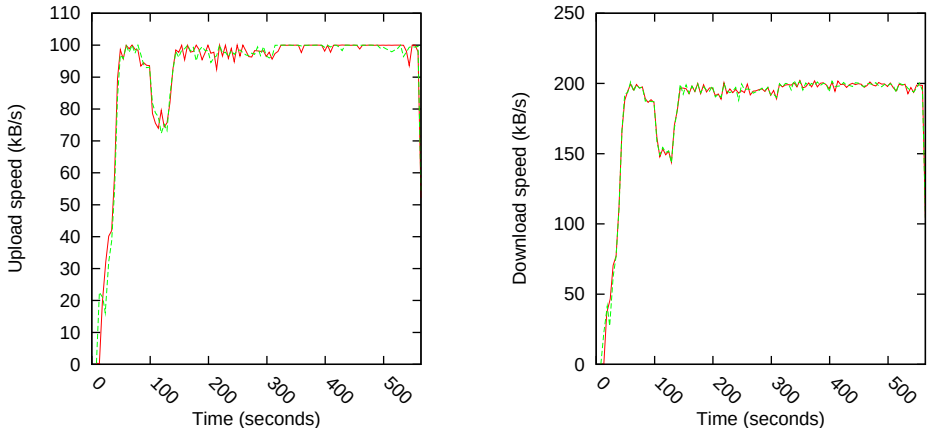
(d) Free rider download speed

Figure B.23: *Free riding experiment* `1-0-50` *using Light Supervised Teaming*



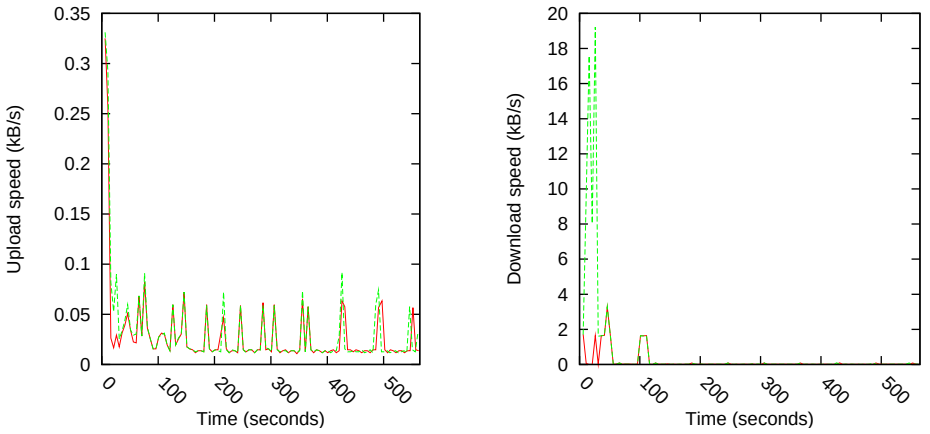
(a) Initial seeder upload speed

(b) Initial seeder download speed



(c) Leecher upload speed

(d) Leecher download speed



(e) Free rider upload speed

(f) Free rider download speed

Figure B.24: *Initial risk experiment* `1-2-2` *using Light Supervised Teaming*