THE HONG KONG POLYTECHNIC UNIVERSITY
DEPARTMENT OF COMPUTING

TOWARDS UNDERSTANDING, IMPROVING AND SECURING BITTORRENT PROTOCOL AND SYSTEM

By
Jiaqing LUO

A thesis submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
June, 2011

CERTIFICATE OF ORIGINALITY

Date: **June, 2011**

Author: **Jiaqing LUO**

Title: **Towards Understanding, Improving and Securing BitTorrent Protocol and System**

Department: **Department of Computing**

Degree: **Ph.D.** Convocation: **July 20** Year: **2011**

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

Signature of Author Jiaqing LUO

Abstract

BitTorrent (BT) is one of the most common Peer-to-Peer (P2P) file sharing protocols. It was reported that the BT system has on average a day 20 millions users from 120 countries. The study of BT protocol and system has become an important aspect of P2P computing.

Towards understanding, improving and securing BT protocol and system, we address several issues including modeling, algorithms and worms. 1) Modeling work provides insights into the BT system performance and the BT protocol improvement. Existing BT models are limited to a certain level – peer level, cluster level, or swarm level. We build a bottom-up probabilistic model to analyze the BT system from the individual peer behavior to the overall system performance. 2) Piece-related algorithms play an inherently crucial role in the BT protocol, because the BT system relies upon peers to cooperatively trade their *pieces* with one another. Although existing algorithms were long believed to be good enough, our understanding of them is still far from complete. We propose a distributed credit method to prevent *under-reporting*, a utility-driven strategy to balance piece supply and demand, and a possible fix to address the conflict between *piece selection* and *piece queuing*. 3) P2P worms have become a serious threat to the Internet security, due to their fast and large-scale propagation in P2P systems. We present a novel *Adaptive BitTorrent worm* (A-BT

worm) to discuss potential ways to design a powerful P2P worm. We then propose a hybrid model to estimate the worm damage, a statistical method to detect the worm behavior, and a safe strategy to slow down the worm propagation.

Publications

1. **Jiaqing Luo**, Bin Xiao, Zirong Yang and Shijie Zhou, "*A Clone of Social Networks to Decentralized Bootstrapping P2P Networks*", in Proc. of the 18th IEEE International Workshop on Quality of Service (IWQoS 2010) (poster paper), pp. 1-2, Beijing China, June 2010.

2. **Jiaqing Luo**, Bin Xiao, Qingjun Xiao, Jiannong Cao and Minyi Guo, "*Modeling and Defending Against Adaptive BitTorrent Worms in Peer-to-Peer Networks*", accepted in ACM Transactions on Autonomous and Adaptive Systems (ACM TAAS).

3. **Jiaqing Luo**, Bin Xiao, Guobin Liu, Qingjun Xiao, Shijie Zhou, "*Modeling and analysis of self-stopping BTWorms using dynamic hit list in P2P networks*", in Proceedings of IPDPS 2009 IEEE International Symposium on Parallel and Distributed Processing (SSN2009), pp.1-8, Rome - Italy, May 2009.

4. Guobing Liu, **Jiaqing Luo**, Qingjun Xiao and Bin Xiao, "*EDJam: Effective Dynamic Jamming Against IEEE 802.15.4-Compliant Wireless Personal Area Networks*", in Proc. of the IEEE International Conference on Communications (ICC 2011), Kyoto Japan, June 2011.

5. Yong Tang, **Jiaqing Luo**, Bin Xiao and Guiyi Wei, "*Concept, Characteristics and Defending Mechanism of Worms*", IEICE Transactions on Information and Systems, Vol. E92-D, No.5, pp. 799-809, May 2009.

6. Qingjun Xiao, Bin Xiao, **Jiaqing Luo** and Guobin Liu, "*Reliable Navigation of Mobile Sensors in Wireless Sensor Networks without Localization Service*", in Proceedings of the 17th IEEE International Workshop on Quality of Service (IWQoS 2009), pp. 1-9, Charleston, South Carolina, July 13-15, 2009.

Acknowledgements

I would like to thank my supervisor Dr. Bin Xiao, who is a really nice person. He treats me like a friend, and provides the freedom to pursue my own ideas. He shares the same interests as me on traveling and photography. I will remember our nice trip in the North of China. I am also grateful to Prof. Keith C.C. Chan and Prof. Jiannong Cao for their kind support during my 3-year Ph.D study. Many thanks to my colleagues Qingjun Xiao, Guobin Liu, Kai Bu and Xuan Liu.

Hong Kong S.A.R., China Jiaqing LUO

August 9, 2011

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

BitTorrent (BT) is one of the most common Peer-to-Peer (P2P) file sharing protocols, and it has been estimated that it accounts for more than roughly 27%-55% of all Internet traffic as of February 2009 [Ernesto, 2009]. Programmer Bram Cohen designed the BT protocol in April 2001 and released a first implementation on July 2, 2001. The protocol distributes a large file without the heavy load on the source computer and network. Rather than downloading a file from a single source, the protocol allows users to join a *swarm* of peers to download and upload from each other simultaneously. The protocol can also work as an alternative method to distribute data and can work over systems with low bandwidth so even small computers, like mobile phones, are able to distribute files to many recipients. There have been numerous BT clients available for a variety of computing platforms.

A BT system commonly consists of trackers, seeders and leechers. A *tracker* keeps track of peers who are participating in the process of downloading and/or uploading a

particular file, and makes this information available to others who want to download the file. Peers in the BT system are either *seeders* who have a complete file and are willing to serve it to others, or *leechers* who are still downloading the file and are willing to serve the pieces that they already have to others. Before joining a *swarm* which is a set of peers sharing a particular file, a peer firstly downloads a .torrent file from a web server, which contains a URL list of trackers and other information related to the sharing file. After that, the peer makes an HTTP GET request to a tracker known from the URL list. Upon receiving the request, the tracker randomly returns a subset of peers sharing the file. Finally, the peer attempts to initiate TCP connections with the returned peers, which then become its *neighbors*. Files in the BT system consists of pieces (32-256KB in size) which in turn consist of blocks (16KB in size). A block is a portion of data that a peer requests from neighbors.

1.2 Research gaps

This thesis addresses several issues in BT protocol and system, including modeling, algorithms and worms. The existing research on these issues is limited in some way.

Modeling: the challenge of accurate BT modeling lies in that the BT system is highly heterogenous and dynamic, where peers with different bandwidths join or leave the system frequently. Previous BT models can be roughly grouped into three levels: peer level, cluster level, and swarm level. Peer-level models [Levin et al., 2008, Rai et al., 2007] aim to model the download process of an individual peer. Although they perceive protocol functions in detail, these models are unable to describe what kind of neighbors a peer has. They may fail to evaluate the performance of the whole system. Cluster-level models [Chow et al., 2009, Liao et al., 2007] assume clusters of peers, and

measure the download rate of each cluster. Despite the fact that these models firstly discuss system heterogeneity, their assumption of peer clustering may not be real. There is a lot of random variation in peer interactions (e.g., neighbor bootstrapping and optimistic unchoking). Moreover, their description of system heterogeneity is not complete. They assume homogeneous bandwidth in each peer cluster. Swarm-level models [Guo et al., 2005, Qiu and Srikant, 2004, Yang and de Veciana, 2004] attempt to study system performance and stability. While they evaluate the evolution of peers, these models discard protocol details and assume homogeneous peer bandwidths. They may not really help in understanding individual peer decision, system dynamics, and system heterogeneity.

Algorithms: when exchanging pieces, peers need three algorithms for different tasks. First, they tell neighbors which pieces they have, called *piece revelation*. After that, they decide what pieces to download, called *piece selection*. Finally, they scatter requests on multiple neighbors, called *piece queuing*. Our understanding of these algorithms is still far from complete, because some fairly basic questions have, to date, gone unanswered: Why is the piece revelation strategy vulnerable to selfish gaming? Are existing piece selection strategies really good enough? Why is the design of piece queueing algorithm still under dispute?

Worms: P2P worms have the potential to compromise hosts in a large scale, which propagate quickly by exploiting the P2P topology. The study of P2P worms includes three major aspects: 1) worm behavior investigation, 2) worm propagation modeling, and 3) worm detection and containment. Worm behavior investigation tries to understand the strategies adopted by worms for target finding and detection evasion. Nowadays, the high penetration of P2P protocols and implementations has

provided the worm writers chances to design some powerful worms that are previously unknown. Worm propagation modeling is to study the effect of system/worm parameters on the worm propagation. P2P worms behave differently in different P2P systems. There is no completely universal model for all P2P worms. Worm detection and containment aim to find worm activities and minimize worm damage. Although existing methods can efficiently detect random scanning worms, they may fail to discover P2P worms because P2P worms can blend their traffic into the normal P2P traffic.

1.3 Contributions

This thesis makes the following contributions towards understanding, improving, and securing BT protocol and system.

A bottom-up probabilistic BT model: we propose a bottom-up probabilistic BT model to simplify the performance analysis and increase the modeling accuracy, in which local transitions of the system lead to global macroscopic descriptions by integration. To be specific, we evaluate the download rate of each individual peer by estimating the probability of a connection being made, and the probability of a peer being unchoked, and then later, integrate the analysis of each individual peer into the description of the whole system. Such a model characterizes both core parts of the BT protocol and main features of the BT system, and requires far less restrictive assumptions than that used before. By using the model, we provide some interesting insights into the setting of parameters, the power of free-riding, the effect of bandwidth distribution, and the effectiveness of unchoking.

The understanding and improving of piece-related algorithms: previous

study has shown that the piece revelation strategy is vulnerable to *under-reporting*. We provide a game theoretic analysis for this selfish gaming, and propose a distributed credit method to prevent it. Existing piece selection strategies, though long believed to be good enough, may fail to balance piece supply and demand. We propose a unified strategy to fasten the download time of peers by applying the utility theory. The design of piece queuing algorithm has a conflict with that of piece selection strategy, because it is not possible to assume that the queued requests for a selected piece can always be available on multiple neighbors. We give a possible fix to address the conflict by allowing peers to dynamically manage their unfulfilled requests.

The modeling and containment of the BT worm: we present a novel *Adaptive BitTorrent worm* (A-BT worm), which locates next victims by sending forged requests to a *tracker* in the BT system. Such a worm can adaptively adjust its speed to evade detection according to the sensed fraction of infected peers. To evaluate the worm damage, we build a hybrid model which combines the fluid model with the epidemic model. To detect the worm behavior, we propose a statistical method which measures the variance in the time intervals of requests. To slow down the worm propagation, we describe a safe strategy which returns a biased set of peers in response to a request.

1.4 Thesis roadmap

Chapter 2 builds a bottom-up probabilistic BT model. Chapter 3 analyzes piece-related algorithms and proposes some improvements to them. Chapter 4 models, detects and contains a new BT worm. Chapter 5 concludes this work and describes our future work.

Chapter 2

A Bottom-up Probabilistic Model for the Heterogenous BitTorrent System

2.1 Overview

In this Chapter, we propose a bottom-up probabilistic model to analyze the performance of the BT system. We begin our model from BT's *bottom* – an individual peer. The download rate of an individual peer is mainly affected by two factors: connectivity and unchoking. Connectivity is productivity in the system, which helps peers exchange pieces with one another. Rather than assuming fully connected peers, we consider both outgoing and incoming connections of a peer to calculate the probability of a connection being made. Unchoking is a key strategy in the protocol, in the sense that a peer has a chance to get pieces from a neighbor, which is related to its upload. Assuming neither homogeneous bandwidth nor peer clustering, we sort peers

according to their upload bandwidth and calculate the probability of a peer being unchoked based on the bandwidth ranking of the peer. To *fully* characterize system heterogeneity, we specify different upload bandwidths for different peers. We then enhance our model to include some additional factors, such as free-riding, churn and neighbor interest. To *precisely* describe system dynamics, we configure peer arrival and seeding times based on real-world measurements. At last, we extend our model to BT's *top* – the whole system. We integrate the analysis of each individual peer into the evaluation of the overall performance.

Our model increases the accuracy of the performance analysis for the following reasons: 1) it firstly relates individual to the whole system; 2) it requires far less restrictive assumptions than that used before; 3) it captures the core parts of the BT protocol, including neighbor bootstrapping, peer unchoking and piece exchanging; 4) it characterizes main features of the BT system, such as heterogeneity and dynamics.

Using our model, we provide some new insights that may be helpful for the improvements of the BT protocol design. First, the theoretical number of peers returned by the tracker can be set as a half of the maximum size of the neighbor set. Second, it is possible for a free-rider with *the large view exploit* to download faster than a large fraction of peers. To the best of our knowledge, this is the first analytical model that can quantify the gain of an *individual* free-rider. Third, long-time seeding is significantly beneficial for low-bandwidth peers, but it has a limited impact on high-bandwidth ones. Fourth, high-bandwidth peers take advantages of a large neighbor set, while low-bandwidth ones may not benefit from it. Assume that peers stay in the swarm for a short time after finishing their download. There might be an overall performance loss, if all peers enlarge their neighbor set. Fifth, unchoking can ensure

BT's fairness to some extent, which means that peers who upload more download faster. However, that does not necessarily mean that it is effective to motivate users to contribute their upload bandwidth.

We validate our model through both simulations and experiments. We implement a java-based BT client, and run several experiments in a live swarm consisting of 120 peers. The preliminary results show that our model is significantly more accurate than both heterogenous model [Chow et al., 2009] and fluid model [Qiu and Srikant, 2004].

The rest of the Chapter is organized as follows. Section 2.2 summarizes assumptions used in our model. Section 2.3 builds a probabilistic model to measure the download rate of an individual peer. Section 2.4 enhances the model to include more factors, and extends it to evaluate the overall performance. Section 2.5 validates the model through simulations and experiments, and gives some insights into the BT protocol design.

2.2 Related work and assumptions

In this section, we review previous models, and describe some assumptions and probabilistic events used in our model.

2.2.1 Related work

Modeling the BT system plays an important role in understanding the system performance and improving the protocol design. Existing models can be roughly classified into three levels: peer level, cluster level and swarm level.

Peer-level models aim to provide insights into the behavior of an individual peer. A Markov model [Rai et al., 2007] is proposed to capture three phases in the download process of a peer: *bootstrap*, *efficient download* and *last download*. In such a model, peers are assumed to use regular unchoking only, and have homogeneous upload bandwidth. An auction-based model [Levin et al., 2008] is given to study unchoking in the BT protocol. In this model, each peer places bids in the form of bandwidth to its neighbors, who in return give bandwidth to the highest bidders from the previous time slot. Though these models can evaluate individual peer decision, they may fail to characterize system performance due to their limited scope of analysis. Our model measures the connectivity among peer pairs, which makes it to be extendable.

Cluster-level models assume that peers form into clusters according to upload bandwidth and share data within the peers in the cluster. Some heterogenous models [Chow et al., 2009, Liao et al., 2007] are proposed to describe the average download rate of each cluster in unchoking. While they characterize system heterogeneity to some extent, these models have the following limitations. First, the assumptions of clustering behavior of peers and homogeneous bandwidth of each cluster are not realistic [Chow et al., 2009, Liao et al., 2007]. It is not possible to assume that peers only share data with those who have similar upload bandwidth with them, because randomization occurs in both neighbor bootstrapping and optimistic unchoking. Even in the explicitly-configured experiments [Legout et al., 2007] (e.g., a single initial seeder, two-cluster or three-cluster leechers), clusters are no longer formed if the initial seeder has a limited bandwidth, because high-bandwidth peers will assist the seeder in disseminating data to low-bandwidth ones. Second, the description of system heterogeneity is incomplete. To relax restrictive assumptions, Liao et al. [Liao

et al., 2007] include imperfect clustering in their model. However, the discussion of imperfect clustering dramatically increases model complexity, which causes the analysis of more than three clusters to become very difficult. Our model does not assume peer clustering, and can accept any upload bandwidth distribution.

Swarm-level models attempt to understand stability and performance of the BT system. Fluid models [Guo et al., 2005, Qiu and Srikant, 2004, Yang and de Veciana, 2004] are used to describe the evolution of peers (changes in the numbers of seeders and leechers) by using aggregate parameters, such as utilization of bandwidth, arrival/departure rate. These models may not really help in understanding system dynamics and heterogeneity, because they discard protocol details and assume homogeneous peer bandwidth. To study unchoking, Qiu et.al [Qiu and Srikant, 2004] assume that all peers are fully connected and have demands from each other. However, in the BT system, each peer makes its own decision regarding neighbors and pieces based on a limited view of the complete swarm. Our model assumes neither fully connected swarm nor homogeneous bandwidth, and evaluates system performance based on aggregate peer behaviors rather than aggregate parameters.

2.2.2 Assumptions and events

To model the BT system, we need to ignore some trivial details and make some reasonable assumptions, because peers behave in a very complicated way in the heterogenous and dynamic system. The assumptions used in our model in Table 2.1:

For the ease of presentation, we list some probabilistic events before using them. Given two peers i and j, we define revelent events in Table 2.2:

Table 2.1: Assumptions used in the BT model

$\mathbb{a}$	There is no limitation to the neighbor set size. In other words, there is always enough room for both outgoing and incoming connections. This assumption enables us to study the effect of some parameters on the neighbor set size.
$\mathbb{b}$	Upload bandwidth is the bottleneck. This is a fairly typical assumption in the literature (e.g., it is also made in [Chow et al., 2009, Fan et al., 2006]).
$\mathbb{c}$	A peer updates the neighbor set at the beginning of each time slot. In the real-world BT system, this happens when a peer does not have enough neighbors.
$\mathbb{d}$	Peers join a swarm in a flash crowd scenario, and do not leave the swarm until completing the download. We will relax this assumption in Section 2.4.3.

Table 2.2: Probabilistic events defined in the BT model

$\mathbb{A}_{ij}$	i is a neighbor of j.
$\mathbb{B}_{ij}$	i ranks in the top 4 for upload bandwidth among leechers in the neighbor set of j (both i and j are leechers).
$\mathbb{C}_{ij}$	i is selected by j in optimistic unchoking (both i and j are leechers).
$\mathbb{D}_{ij}$	i is selected by j in seeder unchoking (i is a leecher and j is a seeder).
$\mathbb{E}_{ij}$	i has all pieces that j has (i is not interested in j) at the beginning of a time slot.

2.3 BT's bottom model

In this section, we construct a probabilistic model to describe BT's bottom – an individual peer. In particular, we measure the download rate of an individual peer by considering both neighbor set and unchoking. To avoid the repetition or omission of peers in the probability calculation process, we sort peers by their upload bandwidth such that the first peer has the highest upload bandwidth in the whole system. If two or more peers have the same upload bandwidth, they are randomly ordered. We distinguish peers using their bandwidth ranking, namely, we choose bandwidth ranking as the peer ID. For example, peer i denotes the peer who has the ith highest bandwidth in the swarm.

2.3.1 Neighbor set in the BT protocol

Background of neighbor set: the set of peers to which a peer is connected is called its *neighbor set* (or peer set). While joining the swarm, a peer initiates outgoing connections with random peers obtained from the tracker, and then, waits for incoming connections from newly arriving peers. As specified in the BT protocol, the connection between each peer pair is bidirectional. That means, given two peers i and j, if i is in the neighbor set of j, then j is in the neighbor set of i. The maximum neighbor set size is 80, the maximum number of outgoing connections is 40, and the minimum number of neighbors is 20. A peer will contact to the tracker again to collect a list of additional peers for more connections, if the size of its neighbor set is below 20. The default number of random peers returned by the tracker is 50. Note that the number will be smaller if there are fewer peers in the swarm, and can be changed by a client-side parameter *numwant*, which is the number of other peers that

a peer would like to receive from the tracker.

Modeling of neighbor set: let S be the total number of peers in a swarm, we describe the number of peers returned by a tracker, R, as:

$$R = min\{50, S\}$$

Given two peers i and j, there are two ways for i to become a neighbor of j. One is that j establishes an outgoing connection to i. If j knows i from the tracker, i will be an outgoing neighbor of j. The probability is $\frac{R}{S}$. The other is that j waits for an incoming connection from i. If j doesn't know i from the tracker, but i knows j from the tracker, i will be an incoming neighbor of j. The probability is $(1 - \frac{R}{S})\frac{R}{S}$. Given above two probabilities, we describe the probability of $\mathbb{A}_{ij}$, $P(\mathbb{A}_{ij})$, as:

$$P(\mathbb{A}_{ij}) = \frac{R}{S} + (1 - \frac{R}{S})\frac{R}{S} = \frac{2RS - R^2}{S^2} \tag{2.1}$$

Then, we write the neighbor set size of a peer, N, as:

$$N = P(\mathbb{A}_{ij})S = \frac{2RS - R^2}{S} \tag{2.2}$$

By Equation 2.2, we can study the effect of parameters R and S on N.

Setting of parameter: as mentioned earlier, the maximum value of N is 80, and the default value of R is 50. Both of them are empirical values. A question is

raised here: if we expect N to be smaller than 80, can we explain why R should be 50? Suppose S is infinite, we have:

$$\lim_{S\to\infty} N = \lim_{S\to\infty} \frac{2RS - R^2}{S} = 2R \tag{2.3}$$

From Equation 2.3, we know that, if we address the raised question, R should be 40 rather than 50. Theoretically, the default number of returned peers can be set as a half of the maximum size of neighbor set.

2.3.2 Unchoking in the BT protocol

Background of unchoking: uploading in the BT protocol is called *unchoking* which dictates to whom and how many to unchoke. Every 10 seconds, a leecher unchokes only 4 neighbors which have the highest upload rates and are interested in downloading from it, called *regular unchoking*. This maximizes the leecher's download rate. To try out unused connections, at least 1 random neighbor is optimistically unchoked by a leecher regardless of that neighbor's contribution, called *optimistic unchoking*. A seeder sorts neighbors according to their download rates, and uploads to the top 5 neighbors, called *seeder unchoking*.

Modeling of unchoking: the amount of download of a peer in a time slot is equal to the total amount of upload from all its neighbors. Roughly speaking, the core idea of unchoking can be represented as "$4 + 1$". To be specific, a peer usually unchokes 4 best leechers and 1 random leecher in its neighbor set according to the upload/download rate. As neighbors of a peer can be either leechers or seeders, we study regular, optimistic and seeder unchokings.

Regular unchoking: given two leecher i and j, i can download from j due to regular unchoking, if the two events $\mathbb{A}_{ij}$ and $\mathbb{B}_{ij}$ occur together. Let U_j be the upload bandwidth of j (in terms of pieces). Because j uniformly assigns its upload bandwidth to 5 unchoked leechers (4 best neighbors and 1 random neighbor), i can download $\frac{U_j}{5}$ pieces from j if it is regularly unchoked by j. We describe the amount of download of i from j in time slot n due to regular unchoking, $\Delta D^r_{j\to i}[n]$, as:

$$\Delta D^r_{j\to i}[n] = \frac{U_j}{5} P(\mathbb{A}_{ij}\mathbb{B}_{ij}) = \frac{U_j}{5} P(\mathbb{A}_{ij}) P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij}) \tag{2.4}$$

The calculation of $P(\mathbb{A}_{ij})$ has been shown in Equation 2.1. In the following discussion, we will describe the calculation of $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij})$. Let $D_x[n]$ be the total amount of download of a peer x till time slot n (in terms of pieces), and Q be the total number of pieces in the shared file. We can determine whether x is a seeder or not at time slot n by an indicator variable $\alpha_x[n]$:

$$\alpha_x[n] = \begin{cases} 1 & \text{if } \lfloor \frac{D_x[n]}{Q} \rfloor = 0 \\ 0 & \text{otherwise} \end{cases}$$

If $\alpha_x[n] = 1$, x is a leecher. Otherwise, x is a seeder. For a given leecher i (it ranks ith in all peers for upload bandwidth), we describe the bandwidth ranking of i in all leechers (not including seeders) at the beginning of time slot n, $B_i[n]$, as:

$$B_i[n] = \sum_{x=1}^{i} \alpha_x[n]$$

There are $B_i[n]-1$ leechers having higher upload bandwidth than i, called *preferred leechers*. If i ranks kth for upload bandwidth among leechers in the neighbor set of j, there will be exactly $k-1$ preferred leechers to become j's neighbors. Note that, i will be regularly unchoked by j, only if $1 \leq k \leq 4$. These $k-1$ neighbors can be either outgoing or incoming. Assume that h $(0 \leq h \leq k-1)$ of them are outgoing. The probability that j knows h preferred leechers from the tracker is $\frac{\binom{B_i[n]-1}{h}\binom{S-B_i[n]}{R-1-h}}{\binom{S-1}{R-1}}$, which is a classical probabilistic formula. The rest $k-1-h$ of them must be incoming. There are $B_i[n]-1-h$ remaining preferred leechers, and each of them can know j from the tracker with a probability $\frac{R}{S}$. The probability that $k-1-h$ remaining preferred leechers know j from the tracker is $\binom{B_i[n]-1-h}{k-1-h}(\frac{R}{S})^{k-1-h}(1-\frac{R}{S})^{B_i[n]-k}$, which is a binomial probabilistic formula. Through the above analysis, we describe the conditional probability of $\mathbb{B}_{ij}$, given $\mathbb{A}_{ij}$, $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij})$, as:

$$P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij}) = \sum_{k=1}^{4} \sum_{h=0}^{k-1} \frac{\binom{B_i[n]-1}{h}\binom{S-B_i[n]}{R-1-h}}{\binom{S-1}{R-1}} \binom{B_i[n]-1-h}{k-1-h} (\frac{R}{S})^{k-1-h} (1-\frac{R}{S})^{B_i[n]-k} \tag{2.5}$$

Equation 2.5 is valid for the case that $B_i[n] > 4$. If $B_i[n] \leq 4$, $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij}) = 1$.

Figure 2.1 illustrates an example of the calculation of $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij})$, when $R = 50$, $S = 100$, $i = 5$, and $B_i[n] = i$ (all peers are leechers). Let's consider the case for $k = 4$ and $h = 2$. Figure 2.2 shows details in the case. When $i = 5$, there are totally 4

($B_i[n] - 1 = 4$) leechers who have higher bandwidth than i. When $h = 2$, j initializes outgoing connections with 2 of them. Each of the remaining 2 ($B_i[n] - 1 - h = 2$) leechers still has a probability 0.5 ($\frac{R}{S} = 0.5$) of being a neighbor of j. When $k = 4$, j accepts only 1 ($k - h - 1 = 1$) incoming connection from the remaining leechers. Corresponding to the case ($k = 4$ and $h = 2$), we have a probability 0.1913. By considering all the cases ($1 \leq k \leq 4$ and $0 \leq h \leq k - 1$), we have $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij}) = 0.69$.

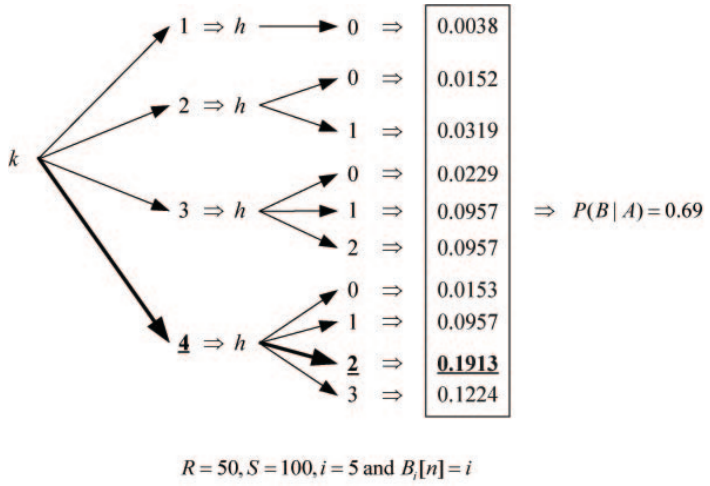

Fig. 2.1: An example of calculation of $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij})$.

Optimistic unchoking: if i ranks below 4th for upload bandwidth among leechers in the neighbor set of j, i has a chance to be optimistically unchoked. The number of leechers in the swarm at the beginning of time slot n, $L[n]$, is:

$$L[n] = \sum_{x=1}^{S} \alpha_x[n]$$

There are $L[n]P(\mathbb{A}_{ij})$ leechers in the neighbor set of j, and 4 of them with the highest

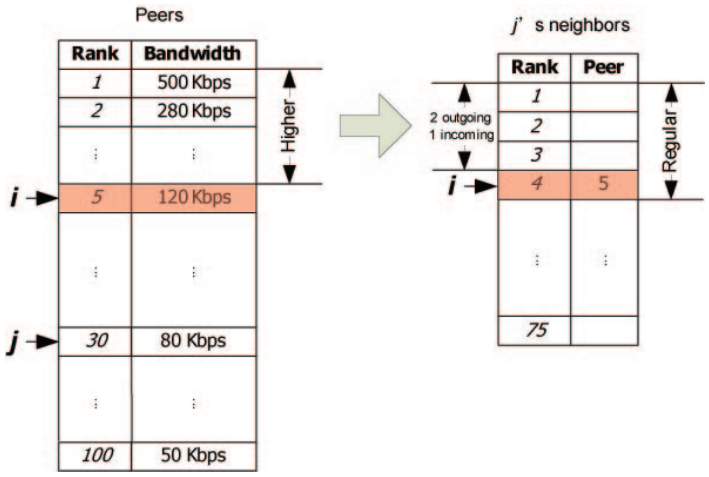

Fig. 2.2: Details in the case for $k = 4$ and $h = 2$.

bandwidths have already been regularly unchoked. The probability of $\mathbb{C}_{ij}$, $P(\mathbb{C}_{ij})$, is:

$$P(\mathbb{C}_{ij}) = \frac{1}{L[n]P(\mathbb{A}_{ij}) - 4} \tag{2.6}$$

We describe the amount of download of i from j in time slot n due to optimistic unchoking, $\Delta D^o_{j\to i}[n]$, as:

$$\Delta D^o_{j\to i}[n] = \frac{U_j}{5} P(\mathbb{A}_{ij}) \Big(1 - P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij})\Big) P(\mathbb{C}_{ij}) \tag{2.7}$$

Seeder unchoking: under assumption $\mathbb{b}$, seeder unchoking is unbiased, because unchoked leechers download from a seeder at an equal rate. In other words, if j is a seeder, j uploads to 5 random leechers in its neighbor set. Given a leecher i, the

probability of $\mathbb{D}_{ij}$, $P(\mathbb{D}_{ij})$, can be calculated as:

$$P(\mathbb{D}_{ij}) = \frac{5}{L[n]P(\mathbb{A}_{ij})} \tag{2.8}$$

The amount of download of i from j in time slot n due to seeder unchoking, $\Delta D^s_{j\to i}[n]$, can be given by:

$$\Delta D^s_{j\to i}[n] = \frac{U_j}{5} P(\mathbb{A}_{ij})P(\mathbb{D}_{ij}) = \frac{U_j}{L[n]} \tag{2.9}$$

From Equation 2.9, we can see that a seeder uploads uniformly to all leechers. This is an approximation to the protocol [Chow et al., 2009]. By Equations 2.4, 2.7 and 2.9, we can describe the amount of download of i in time slot n due to regular, optimistic and seeder unchokings, $\Delta D^r_i[n]$, $\Delta D^s_i[n]$, and $\Delta D^o_i[n]$, as follows:

$$\begin{aligned}
\Delta D^r_i[n] &= \alpha_i[n] \sum_{j=1, j\neq i}^{S} \alpha_j[n] \Delta D^r_{j\to i}[n] \\
\Delta D^o_i[n] &= \alpha_i[n] \sum_{j=1, j\neq i}^{S} \alpha_j[n] \Delta D^o_{j\to i}[n] \\
\Delta D^s_i[n] &= \alpha_i[n] \sum_{j=1, j\neq i}^{S} (1-\alpha_j[n]) \Delta D^s_{j\to i}[n]
\end{aligned}$$

Parameters $\alpha_i[n]$ and $\alpha_j[n]$ determine whether i and j are leechers or seeders. We need to count i's download rate only when i is a leecher. If i is a seeder, it won't download from others. We also need to know j's type of unchokings. If j is a seeder,

it will use seeder unchoking. Otherwise, it will use regular and optimistic unchokings.

We then can state the amount of download of i in time slot n, $\Delta D_i[n] = \Delta D_i^r[n] + \Delta D_i^o[n] + \Delta D_i^s[n]$. Finally, we begin an iterative procedure to calculate the total amount of download of peer i till time slot $n+1$. In particular, we add the incremental variable and get $D_i[n+1] = D_i[n] + \Delta D_i[n]$. The boundary condition for the equation above is $D_i[0] = 0$.

2.4 BT's top model

In this section, we enhance our model to include some additional factors (e.g., free-riding, neighbor interest and churn), and extend it to describe BT's top – the whole system.

2.4.1 Free-riding in the BT system

Background of free-riding: the behavior of optimistic unchoking can be abused by a free-rider. Normally, there is a time slot of 30 seconds granted to a neighbor who is being optimistically unchoked. This time slot is enough for a free-rider to rely on for completing a download without ever having to upload a piece [Locher et al., 2006, Sirivianos et al., 2007]. To increase its chances to be unchoked, a free-rider can obtain a larger than normal view of the swarm by requesting the tracker frequently, or setting the parameter *numwant* to be a large value (much larger than the default value 50), called *the large view exploit* [Sirivianos et al., 2007].

Modeling of free-riding: our model can model free-riding with the large view

exploit by making some modifications. Free-riders always rank at bottom for upload bandwidth, because they download without contributing uploads. Let F be the number of free-riders. The number of normal leechers, $L^n[n]$, and that of free-riders, $L^f[n]$, can be computed as:

$$L^n[n] = \sum_{x=1}^{S-F} \alpha_x[n]$$
$$L^f[n] = \sum_{x=S-F+1}^{S} \alpha_x[n]$$

Assume that each free-rider can obtain a full peer list from the tracker (the best case for a free-rider). Given two peers i and j, $P(\mathbb{C}_{ij})$ and $P(\mathbb{D}_{ij})$ need to be rewritten as:

$$P(\mathbb{C}_{ij}) = \frac{1}{L^n[n]P(\mathbb{A}_{ij}) + L^f[n] - 4} \tag{2.10}$$
$$P(\mathbb{D}_{ij}) = \frac{5}{L^n[n]P(\mathbb{A}_{ij}) + L^f[n]} \tag{2.11}$$

Equations 2.10 and 2.11 indicate that the statement [Chow et al., 2009] that the download rate of a free-rider increases linearly with its neighbor set size may not be true, because, when the free-rider makes more outgoing connections, the neighbor set of other peers also increases (peer connection is bidirectional), which results in a smaller probability of being unchoked (both optimistic and seeder unchokings).

Suppose i is a free-rider ($i > S - F$, $U_i = 0$ and $P(\mathbb{A}_{ij}) = 1$), the bandwidth ranking of i in all free-riders, $B_i^f[n]$, is:

$$B_i^f[n] = \sum_{x=S-F+1}^{i} \alpha_x[n]$$

In such a case, $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij})$ can be computed as:

$$P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij}) = \sum_{k=B_i^f[n]}^{4} \sum_{h=0}^{k-B_i^f[n]} \frac{\binom{B_i[n]-B_i^f[n]}{h}\binom{S-B_i[n]-B_i^f[n]+1}{R-1-h}}{\binom{S-1}{R-1}} \binom{B_i[n]-B_i^f[n]-h}{k-1-h} (\frac{R}{S})^{k-1-h}(1-\frac{R}{S})^{B_i[n]-k+1} \quad (2.12)$$

Equation 2.12 is valid for the case that $B_i[n] - B_i^f[n] \geq 4$ and $i > S - F$. If $B_i[n] - B_i^f[n] < 4$, $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij}) = 1$. If $i \leq S - F$, $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij})$ is the same as that in Equation 2.5.

Power of free-riding: we are interested in how many peers download slower than a free-rider. Assume that all peers are leechers ($L[n] = S$ and $B_i[n] = i$); there is only 1 free-rider in the swarm ($F = 1$); normal peers have similar upload bandwidths. Under above assumptions, the download of the free-rider ($i = S$) only relies on optimistic unchoking, while that of normal peers ($i \neq S$) depends on either regular or optimistic unchoking. Then, we have:

$$
\begin{aligned}
&\Delta D_i[n] < \Delta D_S[n] \quad for \quad i \neq S, j \neq S \\
\Rightarrow\ &(S-2)\Big(\Delta D^r_{j\rightarrow i}[n] + \Delta D^o_{j\rightarrow i}[n]\Big) < (S-1)\Delta D^o_{j\rightarrow S}[n] \\
\Rightarrow\ &(S-2)P(\mathbb{A}_{ij})\Big[P(\mathbb{B}_{ij}\,|\,\mathbb{A}_{ij}) + \Big(1-P(\mathbb{B}_{ij}\,|\,\mathbb{A}_{ij})\Big)P(\mathbb{C}_{ij})\Big] < (S-1)P(\mathbb{C}_{ij}) \\
\Rightarrow\ &P(\mathbb{B}_{ij}\,|\,\mathbb{A}_{ij}) < \frac{(S-1)P(\mathbb{C}_{ij}) - (S-2)P(\mathbb{A}_{ij})P(\mathbb{C}_{ij})}{(S-2)P(\mathbb{A}_{ij})(1-P(\mathbb{C}_{ij}))}
\end{aligned} \tag{2.13}
$$

In Inequality 2.13, $P(\mathbb{B}_{ij} \mid \mathbb{A}_{ij})$ can be viewed as a function of i ($i \neq S$). Given R and S, we can find out a set of i that makes $\Delta D_i[n] < \Delta D_S[n]$. For example, if we set $R = 10$ and $S = 100$, we obtain $P(\mathbb{A}_{ij}) = 0.19$ and $P(\mathbb{C}_{ij}) = 0.06$ by solving Equations 2.1 and 2.10 respectively. Surprisingly, Inequality 2.13 is satisfied for $i \geq 27$. That means, the free-rider, who has an unusually large neighbor set, can download faster than over 70% peers under given conditions. Free-riding is really very cheap in the BT system.

2.4.2 Neighbor interest in the BT protocol

Background of neighbor interest: a peer sends pieces only to those interested in it, and equivalently, receives pieces only from those in whom it is interested. According to the BT protocol, a peer keeps an initial *bitfield*, which is a bit array of its pieces, from each neighbor, and updates it with every *have* message. To keep neighbor interested, two kinds of strategies are used to determine which pieces to download based on these bitfields. One is the *rarest-first* strategy in which peers download pieces in a rarest first order. The download of the rarest piece would make a peer more attractive to its neighbors. The other is the *random* strategy in which

peers choose to download pieces in a random order. The rarest piece might only be available from few peers making it slower to download. In some cases (e.g., in the pieceless mode [Lilja Fjeldsted, 2005]), it is essential for a peer to quickly get as many complete pieces as possible in order to make itself interesting to neighbors.

Modeling of neighbor interest: in our model, we only discuss random for the following two reasons. First, the adoption of random simplifies our analysis. Second, the updated official BT protocol [Cohen, 2008] suggests using random rather than rarest-first. We will analyze both random and rarest-first in Chapter 3.

Let $\lfloor D_i[n] \rfloor$ and $\lfloor D_j[n] \rfloor$ be the number of complete pieces that peers i and j have at the beginning of time slot n, respectively. The probability of $\mathbb{E}_{ij}$, $P(\mathbb{E}_{ij})$, is:

$$P(\mathbb{E}_{ij}) = \frac{\binom{\lfloor D_i[n] \rfloor}{\lfloor D_j[n] \rfloor}}{\binom{Q}{\lfloor D_j[n] \rfloor}} \tag{2.14}$$

From Equation 2.14, we can see that $P(\mathbb{E}_{ij})$ is close to 0 when Q is large. This indicates that a peer is very easy to attract neighbors and keep them attracted so that it can trade pieces using unchoking. In other words, the BT system is very efficient in sharing files. We then rewrite $\Delta D^r_{j \to i}[n]$ and $\Delta D^o_{j \to i}[n]$ as follows:

$$\begin{aligned}
\Delta D^r_{j \to i}[n] &= \frac{U_j}{5} \Big(1 - P(\mathbb{E}_{ij})\Big) P(\mathbb{A}_{ij}) P(\mathbb{B}_{ij} \,|\, \mathbb{A}_{ij}) \\
\Delta D^o_{j \to i}[n] &= \frac{U_j}{5} \Big(1 - P(\mathbb{E}_{ij})\Big) P(\mathbb{A}_{ij}) \Big(1 - P(\mathbb{B}_{ij} \,|\, \mathbb{A}_{ij})\Big) P(\mathbb{C}_{ij})
\end{aligned}$$

If j is a seeder, we have $P(\mathbb{E}_{ij}) = 0$, because i is always interested in j. As a result, $\Delta D^s_{j \to i}[n]$ is the same as that in Equation 2.9.

2.4.3 Churn of the BT system

Background of churn: the dynamics of peer participation, or *churn*, are an inherent property of the BT system. Accurately characterizing churn requires some information about the arrival and departure of peers, which is challenging to acquire. Previous measurement studies on churn [Guo et al., 2005, Stutzbach and Rejaie, 2006] illustrate that the distribution of peer arrival times follows the Exponential or Weibull distribution. The study [Stutzbach and Rejaie, 2006] also concludes that *session lengths* are Weibull or Log-normal distributions rather than Exponential, which are lengths of join-participate-leave cycles of peers. An interesting finding is that many peers linger for a few hours after their download is complete, and a few peers linger for days or weeks. Hence, *download time* (completion time in [Stutzbach and Rejaie, 2006]) and *seeding time* (lingering time in [Stutzbach and Rejaie, 2006]) are further investigated. The first one is the time to completely download the file. The second one is the additional time that a peer lingers in the swarm. The traces show that seeding time distribution can be also modeled by a Weibull distribution.

Churn in our model: since churn in the BT system is related to users' behaviors, churn modeling is beyond the scope of this work. Rather than making some assumptions of churn, we allow our model to configure peer arrival and seeding times based on real traces, so as to study in detail the effect of churn on the download rate of each individual peer. For a peer x, we set arrival time, n_x^a, and seeding time, T_x^s, according to the measurement results. After that, we run the iterative procedure described in Section 2.3.2, and determine the time when x becomes a seeder, n_x^s, by using the following conditions: $\alpha_x[n_x^s - 1] = 1$ and $\alpha_x[n_x^s] = 0$. Finally, we are able to describe download time, T_x^d, departure time, n_x^d, and session length, T_x^l, as follows:

$$T_x^d = n_x^s - n_x^a$$
$$n_x^d = n_x^s + T_x^s$$
$$T_x^l = n_x^d - n_x^a = T_x^d + T_x^s$$

In short, we input n_x^a and T_x^s into the model using distributions suggested by measurements, and output T_x^d, n_x^d and T_x^l from the model after determining n_x^s through the iterative procedure.

To include churn in our model, we need to modify the model slightly. We define a indicator variable $\beta_x[n]$ to illustrate whether x is active or not at the beginning of time slot n:

$$\beta_x[n] = \begin{cases} 1 & \text{if } n_x^a \leq n < n_x^d \\ 0 & \text{otherwise} \end{cases}$$

If $\beta_x[n] = 1$, x is active. Otherwise, x does not participate in the file sharing. At the beginning of time slot n, we compute number of active peers, $S[n]$, that of returned peers, $R[n]$, that of active leechers, $L[n]$, and bandwidth ranking of i in active leechers, $B_i[n]$, as follows:

$$S[n] = \sum_{x=1}^{S} \beta_x[n]$$

$$R[n] = min\{50, S[n]\}$$

$$L[n] = \sum_{x=1}^{S} \beta_x[n]\alpha_x[n]$$

$$B_i[n] = \sum_{x=1}^{i} \beta_x[n]\alpha_x[n]$$

We simply set these four variables in Equations 2.1, 2.5, 2.6 and 2.8 ($S[n]$ and $R[n]$ replace S and R, respectively), and rewrite $\Delta D_i^r[n]$, $\Delta D_i^o[n]$, and $\Delta D_i^s[n]$, as follows:

$$\Delta D_i^r[n] = \beta_i[n]\alpha_i[n] \sum_{j=1,j\neq i}^{S} \beta_j[n]\alpha_j[n]\Delta D_{j\rightarrow i}^r[n]$$

$$\Delta D_i^o[n] = \beta_i[n]\alpha_i[n] \sum_{j=1,j\neq i}^{S} \beta_j[n]\alpha_j[n]\Delta D_{j\rightarrow i}^o[n]$$

$$\Delta D_i^s[n] = \beta_i[n]\alpha_i[n] \sum_{j=1,j\neq i}^{S} \beta_j[n](1-\alpha_j[n])\Delta D_{j\rightarrow i}^s[n]$$

2.4.4 Overall system performance

We integrate the analysis of each peer into the evaluation of the overall performance. From the system point of view, we are care about the average download rate and time of all peers.

The average download rate of peers in time slot n, $\overline{\Delta D[n]}$, can be calculated as:

$$\overline{\Delta D[n]} = \frac{\sum_{x=1}^{S} \Delta D_x[n]}{\sum_{x=1}^{S} \beta_x[n]}$$

The average download rate of all peers, $\overline{D}$, can be represented as:

$$\overline{D} = \frac{\sum_{x=1}^{S} \sum_{n=n_x^a}^{n_x^s} \Delta D_x[n]}{\sum_{x=1}^{S} T_x^d}$$

The average download time of peers, $\overline{T^d}$, can be given by:

$$\overline{T^d} = \frac{\sum_{x=1}^{S} T_x^d}{S}$$

After defining $\overline{\Delta D[n]}$, $\overline{D}$, and $\overline{T^d}$, we successfully extend our model from bottom to top.

2.5 Model validation and insights

In this section, we validate our model using simulations and experiments. We also illustrate how our model can be used to obtain insights into the BT system. Before starting this section, we would like to point out that the real-world BT system is very complex, the attempt of this work is only to increase the accuracy of analysis of the system.

2.5.1 Simulation verification

We develop a time-based simulator to simulate piece exchanging in the BT system. In each time slot, each peer randomly connects to a subset of peers in the swarm, and uniformly assigns its upload bandwidth to 5 neighbors according to unchoking in the BT protocol. Unless specified otherwise, we use the following settings in our simulations. The shared file consists of 5000 pieces. The system starts with 1 initial seeder that uploads at 20 pieces/time slot, which is the average upload bandwidth of all peers. The initial seeder stays throughout the duration of the simulation. The swarm size is 150. The maximum neighbor set size of a peer is 60. We assign an Exponential distribution to their arrival time and seeding time with a rate of λ ($\lambda = 0.05$). The upload bandwidth of peers follows a Pareto distribution. We generate a random variable from a given distribution using the following methods.

Exponential distribution: to obtain an exponential random variable x with rate λ, we generated a uniform random number $r \in [0, 1]$ and set $x = -\frac{\ln r}{\lambda}$. The coefficient of variation of x, c_v, is 1.

Pareto distribution: a random variable x following the Pareto distribution has the following cumulative distribution function:

$$F(x) = 1 - \left(\frac{x_m}{x}\right)^{\alpha} \quad for \quad x_m, \alpha > 0$$

After drawing a uniform random number $r \in [0, 1]$, we can generate a Pareto random variable x by setting $x = \frac{x_m}{(1-r)^{\frac{1}{\alpha}}}$. In our simulations, we set $x_m = \frac{1}{\lambda}\frac{\sqrt{2}}{1+\sqrt{2}}$ and $\alpha = 1 + \sqrt{2}$ in order to keep the rate at λ and $c_v = 1$.

Uniform distribution: after drawing a uniform random number $r \in [0, 1]$, a uniform random variable x with rate λ can be obtained by setting $x = \frac{2r}{\lambda}$. We have $c_v = \frac{1}{\sqrt{3}}$.

Neighbor set size: we begin to study the effect of parameters on the neighbor set size. We set $R = 50$. Figure 2.3 depicts the the neighbor set size as a function of the swarm size. We can observe that, when S increases, N increases rapidly and then levels off. For example, when S increases from 100 to 150, N increases by 11%. However, when S goes up from 500 to 550, N only increases by 0.5%. According to our analysis in Section 2.3.1, if $R = 50$, N will not excess 100, no matter how large S is. Theoretically, we can use parameter R alone to limit the N to a reasonable size. But, in the real-world system, it is necessary to set a maximum value for N, because some peers may request the tracker multiple times, when they do not have enough neighbors (due to connection failures). We then set $S = 500$. Figure 2.4 illustrates the neighbor set size as a function of the number of returned peers. We can see that N increases along with R increases. For example, when R increases 20, N increases by 14%. Note that the curve in Figure 2.4 is close to linear but not exactly it.

We then show the effect of the neighbor set size on the download time of a peer. We set $N = \{10, 15, 30\}$. Figure 2.5 depicts that high-bandwidth peers can benefit from a large neighbor set. For example, T^d_{20} decreases by almost 24%, when N increases from 10 to 15. The reason is that high-bandwidth peers are more likely to be unchoked by others. The more neighbors they have, the faster they can download. On the other hand, if all peers choose a large neighbor set, performance degradation happens on low-bandwidth peers. As we can see that T^d_{120} increases by 16% when N increases from 10 to 15. The early departure of high-bandwidth peers is the main reason for

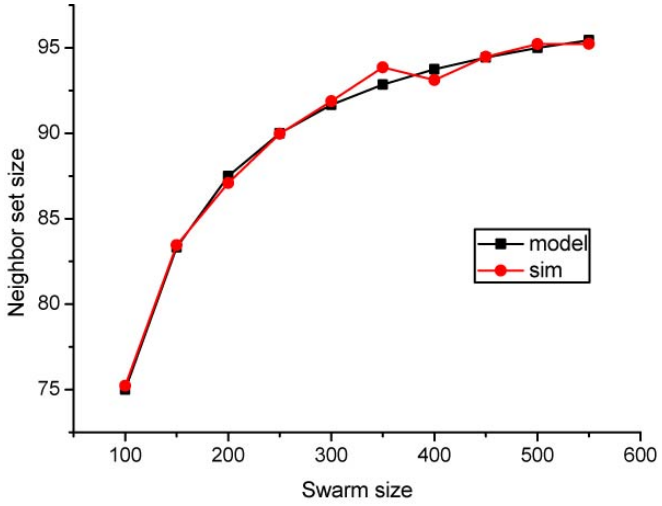

Fig. 2.3: Effect of swarm size on neighbor set size.

the performance degradation. Besides that, a large neighbor set may not be helpful to low-bandwidth peers, because it reduces the probability of being unchoked (both optimistic and seeder unchokings). We also observe that the average download time increases by 8% when N increases from 10 to 15. A larger neighbor set may not lead to a better overall performance. As real-world measurements [Isdal et al., 2007, Pouwelse et al., 2005] suggest that majority of peers (70%) are low-bandwidth ones, it is worthwhile to consider some schemes on tracker to balance the download rates of peers. For example, the tracker can determine the number of returned peers according to some system parameters, such as swarm size, departure rate and download rate, etc. In order to rationally utilize the bandwidths of peers, it can also return high-bandwidth peers or seeders to some pieceless or low-bandwidth peers. (the tracker can simply assume that peers who download at high rate are high-bandwidth ones).

Upload bandwidth: we measure the effect of different bandwidth distributions

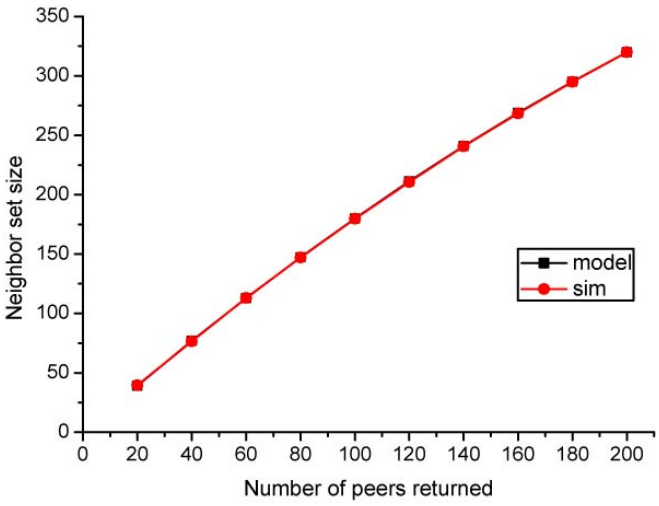

Fig. 2.4: Effect of number of returned peers on neighbor set size.

on the download time of a peer. From Figure 2.6, we can see that the Pareto distribution (Power-law distribution) is more efficient than other two distributions for file sharing. The average download time for the Pareto distribution is only 37% of that for the Exponential distribution, and 59% of that for the Uniform distribution, respectively. The possible reason is that majority peers have similar upload bandwidths in the Pareto distribution, while some peers have very low upload bandwidths in the other two distributions (their mean values are the same). Low-bandwidth peers usually need a long time to complete their download, because, after high-bandwidth peers leave the swarm, they have to share pieces with those who have similar bandwidths with them.

Seeding time: we study the effect of the average seeding time on the download time of a peer. We set the average seeding time to be {20, 40, 80}. From Figure 2.7, we can see that long-time seeding is significantly beneficial for low-bandwidth peers. For example, T^d_{120} decreases by almost 20%, when the average seeding time

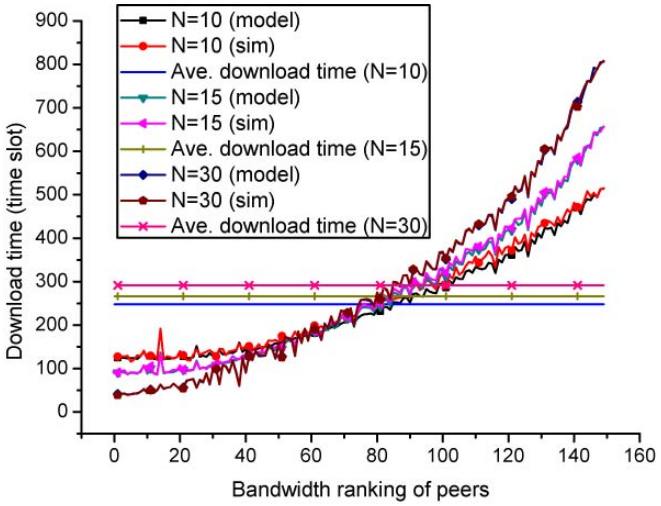

Fig. 2.5: Effect of neighbor set size on download time of a peer.

increases from 40 to 80. On the other hand, the download time of high-bandwidth peers seems to change little when seeders stay more time in the swarm. After a peer becomes a seeder, low-bandwidth peers are more likely to be unchoked by that peer, because the probability of a seeder unchoking is higher than that of an optimistic unchoking. On the contrary, high-bandwidth peers usually have less chance of being unchoked, because the probability of a seeder unchoking may be lower than that of a regular unchoking. Another observation is that overall performance increases when the average seeding time increases. Not surprisingly, $\overline{T^d}$ decreases by 11% when the seeding time increases from 20 to 40. To improve overall performance, it is essential to encourage seeders to stay in the swarm as long as possible.

Free-riding: we firstly study the effect of free-rider ratio on the download time of free-riders. We set the ratio of free-riders to be $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. Figure 2.8 shows that the average download time increases along with the ratio of free-riders. For example, when the ratio of free-riders increases from 0.1 to 0.2, the download

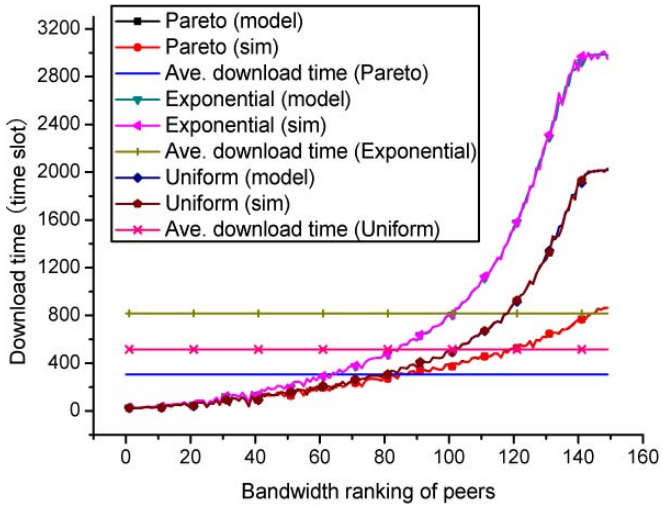

Fig. 2.6: Effect of bandwidth distributions on download time of peer.

time of normal peers and free-riders increase by 10% and 15%, respectively. Note that the performance degradation of free-riders is more serous than that of normal peers. That means, free-riders cannot benefit from the large view exploit when they hold majority.

We then evaluate the effect of the neighbor set size on the download time of free-riders. We set the ratio of free-riders to be 0.2, all peers have similar upload bandwidth about 20 pieces/time slot, and $N = \{20, 40, 60, 80, 100\}$. Figure 2.9 shows that free-riders download slower when N increases. For example, free-rider's download time increases by 7%, when N increases from 20 to 40. The reason is that, when N is smaller, free-riders have a higher probability of being optimistically unchoked. On the other hand, normal peers download faster when N becomes larger. As we can see that, when N increases from 20 to 40, normal peer's download time decreases by 10%.

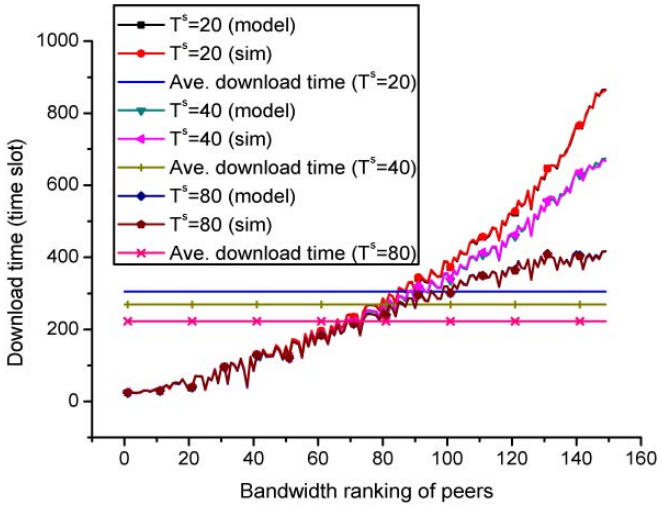

Fig. 2.7: Effect of average seeding time on download time of a peer.

2.5.2 Experiment comparison

We implement a java-based BT client referring to [Lilja Fjeldsted, 2005, Theory-Org, 2004], which is now available at FTP [Luo, 2010] for the research and testing purpose. We decide to implement a client rather than use an existing one for the following reasons: 1) we want to know more about the BT protocol and technical details, which helps a lot in our modeling. 2) our client only contains some basic functions, which makes setup and configuration easy. We run all our experiments in a controlled environment consisting of 4 hosts. Each host has 1.86GHz Intel Core 2 CPU, 2 GB RAM, and 1 GigE connection. One host installs a local tracker. Each host runs 4 BitComet clients [BitComet, 2009] and 30 java-based clients. The BitComet clients act as initial seeders. Peers join the swarm in a flush crowd scenario, and leave the swarm as soon as they have the complete file. Upload bandwidths of peers follow a uniform distribution between 32KB/s and 304KB/s. The shared file is

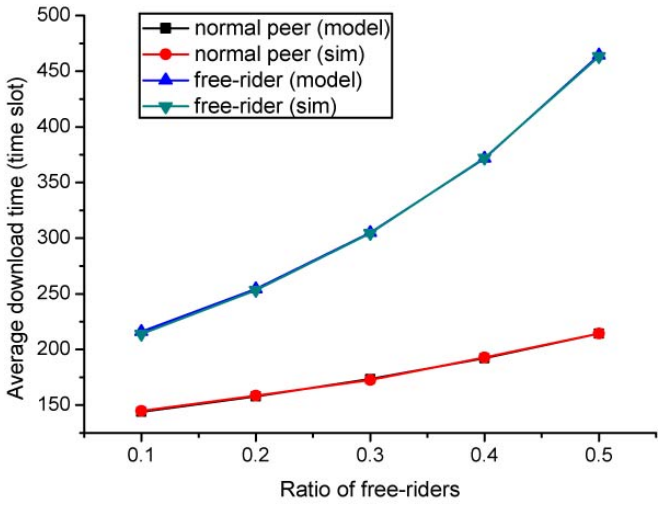

Fig. 2.8: The effect of the free-rider ratio on the download time of free-riders.

20MB, which is fragmented into 128KB pieces. The maximum number of neighbors for each peer is 80. The maximum number of queued pieces is 10, namely, 80 blocks to be requested. Each point in the figures that follow represents the average over at least 10 runs, and error bars denote 95% confidence intervals.

Comparison with the heterogenous model: we compare our model with the heterogenous model [Chow et al., 2009]. We measure the download time of each peer through experiments, and compare it with that generated by models. As it is unclear how to classify peers in a fully heterogeneous system, we uniformly divide peers into three clusters: high-bandwidth, middle-bandwidth and low-bandwidth. From Figure 2.10, we can see that the average download time of the real trace is closer to that of the bottom-up model than it is to that of the heterogeneous model. Although there is an obvious error, the bottom-up model is significantly accurate than the heterogeneous one, as it can better reflect the trend of download time variation. These results also indicate that unchoking can to some extent ensure that the peers who upload more

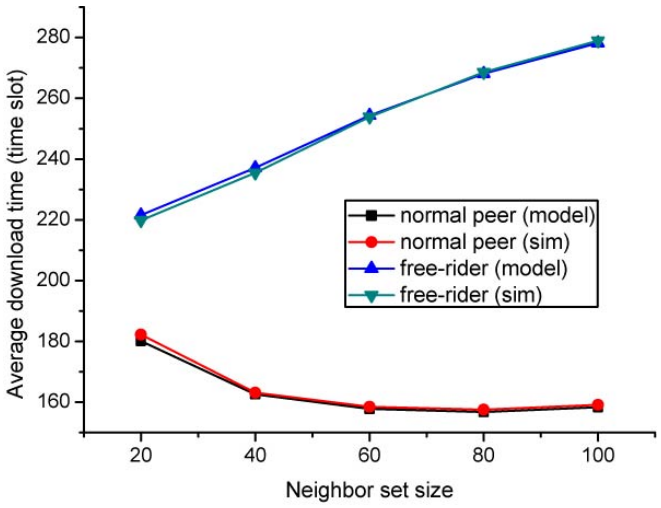

Fig. 2.9: Effect of neighbor set size on download time of free-riders.

download faster.

Interestingly, many real measurements [Isdal et al., 2007], [Pouwelse et al., 2005] show that unchoking is ineffective in motivating most users to contribute their bandwidth. The possible reason for this lies in our incomplete understanding and inappropriate assumptions of users [Feldman and Chuang, 2005]. There is clearly a need for learning about the factors that may affect users' decisions. Download rate is an important factor, but it is not all. Unchoking does not completely understand what users care (e.g., the credit gained from uploading). It is also important to qualify the potential outcomes of users. According to the expectancy valence theory [Vroom and MacCrimmon, 1968], the strength of motivation equals the perceived value of the outcome times the perceived probability of the behavior resulting in the outcome. Unchoking is short in providing some useful information for users (e.g., how much time users could save by contributing more upload bandwidth). The further discussion of incentives in the BT protocol could be our future work.

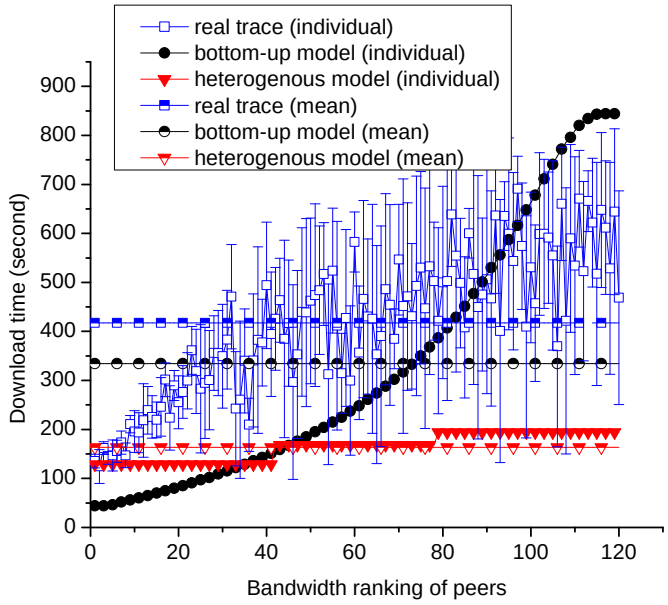

Fig. 2.10: Comparison between bottom-up model and heterogenous model.

Comparison with the fluid model: we also compare our model with the fluid model [Qiu and Srikant, 2004]. Since the fluid model cannot evaluate the download time of each peer, we count the number of peers in the swarm every second. According to our experimental settings, we describe the fluid model as the following:

$$\frac{ds(t)}{dt} = -us(t) \Rightarrow s(t) = e^{-ut} + c$$

, where $s(t)$ is the normalized number of peers in the swarm at time t, and u is the average upload bandwidth of all peers, c is a constant. Figure 2.11 shows that the real trace curve is closer to that of our model than it is to that of the fluid model. For example, at the 500th second, the number of peers in the real trace is 27% and

77.8% higher than that in our model and that in the fluid model, respectively.

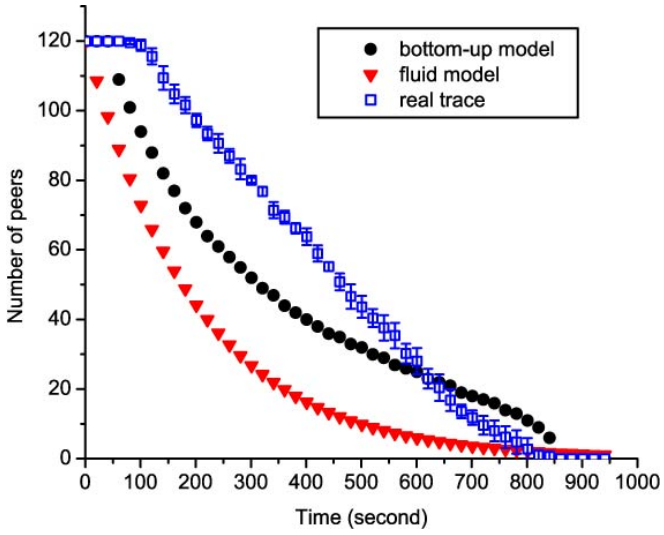

Fig. 2.11: Comparison between bottom-up model and fluid model.

Chapter 3

Understanding and Improving Piece-related Algorithms in the BitTorrent Protocol

3.1 Overview

In this Chapter, we borrow some ideas from economics to understand and improve piece-related algorithms, including piece revelation, piece selection and piece queuing, in order for both *individual good* – fast download time of an individual peer, and *social good* – fast average download time of all peers.

We provide a game theoretic analysis for piece revelation, and propose a *distributed* credit method to prevent strategic manipulations. Experimental results [Levin et al., 2008] have shown that *under-reporting*, where peers do not disclose the pieces they truly have, may degrade overall performance. However, there is no clear explanation for the performance degradation. Our study finds that under-reporting can enhance

peer's competitiveness in non-cooperative games. In particular, a peer who under-reports its pieces can gain some extra interest from its neighbors, because it is able to upload some pieces that should be uploaded by its opponents. However, when all peers under-report their pieces to others, there may be an overall performance degradation, because each peer will spend more time on downloading from low-speed neighbors. To encourage neighbors to reveal as many pieces as they truly have, we suggest that peers work cooperatively to credit their neighbors by examining *have* messages.

We apply the utility theory to piece selection, and design a *unified* strategy to balance piece supply and demand. Nowadays, two strategies are well known in piece selection. One is *random* in which peers choose to download pieces in a random order. While it can enrich the diversity of piece supply, a random piece might not meet the needs of neighbors. The other is *rarest-first* in which peers download pieces in a rarest first order. Though it can make a peer more attractive to others, a rarest piece may only be available from very few neighbors, which makes it slow to download. Moreover, it might not fulfill emergent needs of some neighbors despite the fact that it is in a high demand. To improve the efficiency of piece sharing, we propose a *utility-driven* strategy, which covers both rarest-first and random, by constructing two utility functions – *neighbor's utility* and *peer's utility*. Rather than selecting a random or rarest piece, we suggest that a peer selects the one that can maximize its utility.

We address the conflict in piece queueing, and give a possible fix to *dynamically* mange unfulfilled requests. As pointed out by some BT protocol specifications

[Lilja Fjeldsted, 2005, TheoryOrg, 2004], there exists a conflict between piece selection and piece queuing. On the one hand, a peer tends to select a rarest piece to make itself more attractive to others [Lilja Fjeldsted, 2005]. On the other hand, it expects to queue the requests for a selected piece at each neighbor to fully utilize others' bandwidth [TheoryOrg, 2004]. Unfortunately, the requests for a selected piece are not always spread over multiple neighbors. For the sake of low memory consumption and efficient bandwidth usage, we suggest that a peer separates queued pieces into two groups – *rational* and *sub-rational*, and keeps a ratio between them to dynamically control the size of request queue.

Our primary experimental results show that the proposed algorithms can achieve both individual good and social good. The download time of an under-reporter will increases 21.8%, if it is credited and punished by 20% neighbors. Compared with rarest-first and random strategies, the utility-driven strategy decreases the average download time by 41.8% and 35%, respectively. The revised queuing algorithm outperforms the scatter algorithm [Lilja Fjeldsted, 2005] by saving 62.3% average download time.

The rest of the Chapter is organized as follows. Section 3.2 describes the goals of this work. Section 3.3 provides a new insight into under-reporting. Section 3.4 designs a distributed credit method to prevent under-reporting. Section 3.5 considers various factors in piece selection, and shows rarest-first may fail to achieve its design goal. Section 3.6 proposes a utility-driven strategy to shorten the download time of peers. Section 3.7 revises the scatter algorithm to resolve the dispute in piece queuing. Section 3.8 presents experimental results of the proposed algorithms.

3.2 Goals and assumptions

In this section, we describe the goals – individual good and social good, and make some reasonable assumptions – rational peers and a fair BT protocol.

The BT system can be viewed as a market where peers exchange their pieces with one another. A peer is *interested* in a neighbor, if the neighbor has at least one piece that it does not have. A peer's interest in one of its neighbors is based solely on what pieces that neighbor claims to have. Since a peer sends only to those interested in it, and equivalently, receives only from those in whom it is interested, the term *interest* reflects piece supply vs. piece demand. The probability of piece sharing will increase, if *all* peers can maximize the number of neighbors who are interested in them. We decompose our goal of this work into two subgoals. Our preliminary goal is to maximize neighbor interest. Our ultimate goal is to achieve both individual good and social good.

We assume that all peers are *rational*, that is, they are smart enough to predict others' decisions, and behave to garner maximum neighbor interest in a unit cost. We further assume that the BT protocol is fair, where the incentive mechanism is well designed, such that peers can get more if they upload more. Though the current incentive mechanism – unchoking, is not perfectly fair [Levin et al., 2008], some of our discussions are still based on it, because it is widely deployed. We do not assume cooperative peers, that means, peers can choose either cooperative or noncooperative strategies. Yet, we would like to point out that individual maximization may not lead to socially optimal outcome [Fudenberg and Tirole, 1992]. To achieve social good, peers should cooperate instead of compete.

3.3 Study of piece revelation

In this section, we describe possible strategies that can be used for piece revelation, and provide a game-theoretical analysis to explain why the BT system is vulnerable to under-reporting.

3.3.1 Piece revelation strategies

Generally, a peer has three strategies to reveal pieces to neighbors: *over-reporting*, *full-reporting* and *under-reporting* (including non-reporting). According to the BT protocol, a peer should honestly disclose all its pieces so as to make others clearly know the rarity of pieces, called *fully-reporting*. However, a selfish peer has incentive to strategically manipulate others into helping it download faster. To prolong neighbor interest or keep neighbors interested in it, a selfish peer can exaggerate the pieces it has, called *over-reporting*, or hide some pieces it has, called *under-reporting*. In economics and contract theory, these strategic manipulations can be characterized by *information asymmetry* where some parties have an information advantage over others [Fudenberg and Tirole, 1992].

Over-reporting can be easily detected, because a peer cannot respond to the requests for the pieces that it falsely claimed. Compared with over-reporting, under-reporting works more stealthily. A selfish peer only reveals a piece to a neighbor when the neighbor is going to lose interest. As providing a neighbor with a rare piece would make the neighbor potentially removing some interest from it, it reveals the most common piece it has that the neighbor does not. Experimental results [Levin et al., 2008] have shown that under-reporting will cause 12% overall performance loss when it becomes widespread.

Although previous work [Levin et al., 2008] has revealed that under-reporting is a potential threat to BT's fairness and performance, some questions are left unanswered: Why is under-reporting degrading overall performance? What kind of swarms are more vulnerable to under-reporting? In the following sections, we provide some answers to above questions through a game-theoretical analysis.

3.3.2 Under-reporting in heterogenous swarms

We design a two-player game to show that under-reporting causes more overall performance degradation in the heterogenous swarm than in a homogeneous swarm. In such a game, players (peers) a and b upload their pieces to another peer ℓ. Each player has two strategies *Fully report* and *Under-report*. If a player plays *Fully report*, it reveals all its pieces to ℓ. One the other hand, if the player plays *Under-report*, it won't reveal the rarest piece until ℓ has completed the most common piece. Both players want to maximize their payoffs in terms of ℓ's interest measured in time slots. As shown in Figure 3.1, a has pieces 1 and 3, b has pieces 2 and 3. If a and b play *Under-report*, they will reveal the most common piece 3 first.

Without loss of generality, we consider two cases. In Case A, b has the same upload speed as a. Assume that the upload speed for each player is one piece per time slot, we show players' payoffs of Case A in Figure 3.2. We then have three observations. 1) For both players, *Under-report* strictly dominates *Fully report*, called *dominant strategy*. Therefore, (*Under-report*, *Under-report*) is a strict *Dominant Strategy Equilibrium* which is a set of strategies in which every player is playing a dominant strategy. 2) *Under-report* could make a player gain some extra interest from ℓ, because the player is possible to upload some pieces that should be uploaded by its opponent. For

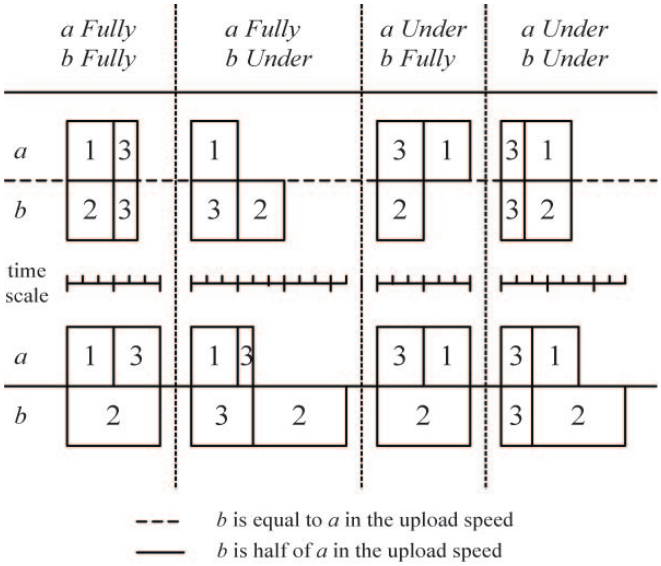

Fig. 3.1: An upload processes, showing that a and b upload to ℓ.

example, if both a and b play *Fully report*, a can upload a half of piece 3. However, if b changes its strategy to *Under-report*, a has no chance to upload that piece. 3) The outcome of (*Under-report*, *Under-report*) is the same as that of (*Fully report*, *Fully report*). This implies that the *Under-report* strategy may have a limited effect on overall performance in homogeneous swarm where players have similar upload speeds.

In Case B, b has a lower upload speed than a. Assume that b is half of a, we illustrate players' payoffs of Case B in Figure 3.3. We then have two observations. 1) *Under-report* is a weakly dominant strategy for a, while it is a strictly dominant strategy for b. As we can see that if b plays *Fully report*, a is indifferent between playing *Under-report* and playing *Fully report*. But, a never plays *Fully report* because it is rational and knows that b will never play *Fully report*. 2) The outcome of (*Under-report*, *Under-report*) is not good for the high-speed player, which could result in a

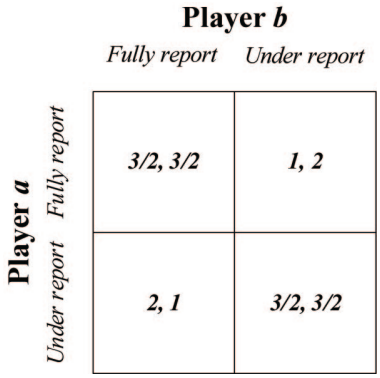

Fig. 3.2: The payoffs of Case A where b is equal to a in the upload speed.

degradation of overall performance, because each peer would spend more time on downloading from the low-speed players. For example, ℓ's download time increases from 2 to $\frac{8}{3}$.

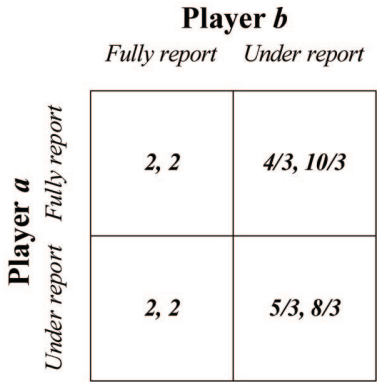

Fig. 3.3: The payoffs of Case B where b is half the upload speed of a.

Within the game theoretic analysis, we can draw some useful conclusions. 1) a rational player always plays *Under-report*. 2) *Under-report* is a non-cooperative strategy that could enhance peer's competitiveness. However, the non-cooperative

equilibrium does not necessarily mean the best payoff for each player. 3) *Under-report* may degrade overall performance in a heterogeneous swarm where peers have different upload speeds.

3.3.3 Under-reporting in large swarms

We use an n-player game to illustrate that peers are more likely to adopt under-reporting (or non-reporting) when swarm size becomes larger. In this game, n players know each other in a swarm (a fully connected swarm of n peers). Each player has some pieces that others do not have, and would like to share pieces with others but prefers that someone else reports its pieces. Suppose each player attaches the value v to a piece being reported (thereby the piece being shared), and bears a cost c of reporting the piece. The value v could represent the prolonged interest for the reporter (it uploads to others), or a newly complete piece for others (they download from the reporter). The cost c could state the bandwidth consumption of broadcasting *have* messages and the potential risk of losing others' interest. We assume $v > c > 0$. Player i has two strategies, *Report* and *Not Report*, where $\forall i \in \{1, 2, ..., n\}$. The payoff of player i, u_i, is defined as follows:

- $u_i(\text{Report}) = v - c$
- $u_i(\text{Not Report, at least one other player reports}) = v$
- $u_i(\text{Not Report, none of the other players report}) = 0$

Figure 3.4 illustrates players' payoffs of a two-player game. We have two pure-strategy Nash equilibria, where one player plays *Report* and the other plays *Not report*. Both of them are asymmetric, where different players play different strategies. For the

n-player game, we have n pure-strategy Nash equilibria, where a player i plays *Report*, and the remaining $n-1$ players play *Not report*. But again these are asymmetric, and hence, do not resolve the issue of which of the players play *Report*. For the symmetric equilibrium, we look for the one involving mixed strategies. Let p_i be the probability that player i plays *Report*. Since we are trying to find the symmetric equilibrium, it must be true that at equilibrium all players will be reporting with the same probability. Let that be p^*. Without lose of generality, we use the indifference condition for player i.

$$
\begin{aligned}
& u_i(\text{Reports}) = u_i(\text{Not Report}) \\
\Rightarrow\ & v - c = 0 \cdot Prob(\text{no one reports}) + v \cdot Prob(\text{someone reports}) \\
\Rightarrow\ & v - c = v\big(1 - Prob(\text{no one else reports})\big) \\
\Rightarrow\ & v - c = v\big(1 - (1-p^*)^{n-1}\big) \\
\Rightarrow\ & p^* = 1 - \big(\frac{c}{v}\big)^{\frac{1}{n-1}}
\end{aligned}
$$

Suppose that p^* is a continuous function on n, the first derivative of p^* with respect to n is:

$$
\begin{aligned}
(p^*)' &= -\big((\frac{c}{v})^{\frac{1}{n-1}} \ln(\frac{c}{v})\big)(\frac{1}{n-1})' \\
&= (\frac{c}{v})^{\frac{1}{n-1}} \ln(\frac{c}{v}) \frac{1}{(n-1)^2}
\end{aligned}
$$

Since $v > c > 0$, we have $(p^*)' < 0$. This indicates that p^* decreases when n increases. If n is large, players tend to play *Not report*, because they are more likely to believe that someone else will play *Report*.

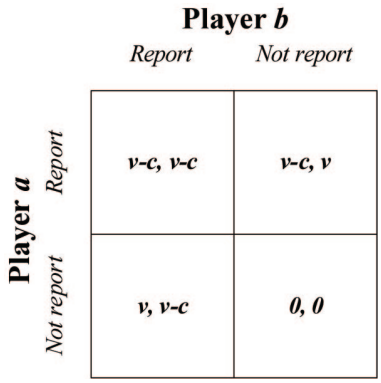

Fig. 3.4: The two-player illustration of Peer's Dilemma.

3.4 A distributed credit method

In this section, we design a distributed credit method to detect and punish under-reporters, and propose some improvements to make the method more practical.

3.4.1 Method design and limitation

The basic idea of the distributed credit method is that three fully connected peers, called a *basic component*, work together to credit each other. According to the BT protocol, a peer will broadcast a *have* message to all neighbors once it has a new piece, which contains the index of that piece. Thus, it is possible for us to detect under-reporters by examining *have* messages. Figure 3.5 shows a credit process.

Peers a, b and c are fully connected. By exchanging neighborhood information, a and b knows that c is an overlapping neighbor. To test whether c is honest or not, a informs b of a recent piece that c requested from it. Upon receiving the *have* from c, b relays it to a immediately. If the *have* arrives within a short time, a will increase c's credit. Assuming peers who are honest can earn credits more easily than peers who are dishonest, by earning credits peers can signal their honesty to others. If a peer has " bad credit", it will be punished by others. A similar idea can be found in the signaling game [Fudenberg and Tirole, 1992], which is a non-cooperative game with incomplete information. In the job market, new employee signal their capacity for learning to prospective employers by finishing college.

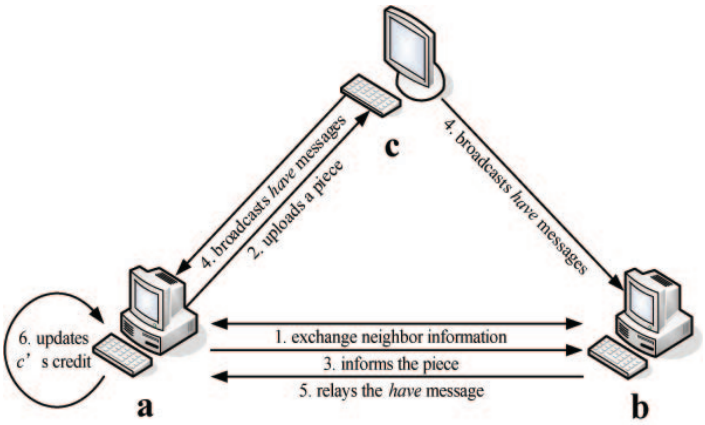

Fig. 3.5: A credit process, showing that a and b work together to credit c.

The limitation of the credit method is that a basic component may not be formed in a large swarm. Assume that there are n peers in the swarm, and each of them has r random neighbors. According to the BT protocol, the connection between each peer pair is bidirectional. In other words, given any two peers i and j, if i is in the neighbor set of j, then j is in the neighbor set of i. While joining the swarm, a

peer initiates outgoing connections with random peers obtained from the tracker, and then, waits for incoming connections from newly arriving peers. Suppose that there are n peers in the swarm, and each peer has r outgoing neighbors. If j knows i from the tracker, i will become an outgoing neighbor of j. The probability is $\frac{r}{n}$. On the other hand, if i knows j from the tracker, i will be an incoming neighbor of j. The probability is $(1-\frac{r}{n})\frac{r}{n}$. Considering these two cases, the probability that there is a connection between any two peers, $p_{=}$, is $\frac{r}{n}+(1-\frac{r}{n})\frac{r}{n}$. Since peer connections are independent from each other, the probability that any three peers are fully connected, $p_{\triangle}$, is $\left(\frac{r}{n}+(1-\frac{r}{n})\frac{r}{n}\right)^3$. In most cases, $p_{\triangle}$ is higher than 0.084, because r is 50 by default [Cohen, 2008, TheoryOrg, 2004], and n is less than 300 for more than 95% swarms [Wang et al., 2009].

3.4.2 Method improvement and configuration

A possible improvement for the credit method is to allow peers construct a basic component by the shallow flooding. If a peer, called *creditor*, wants to credit one of its neighbors, called *creditee*, it will broadcast a *query* message with a small TTL (time to live) and the creditee's IP. The *query* will then propagate through the swarm hop by hop till TTL has expired. When receiving a *query*, a peer, called *intermediator*, will check whether the creditee exists in its neighbor list. If not, it will decrease TTL by 1 and forward the *query* to all its neighbors. Otherwise, it will respond a *queryhit* message which contains its own IP. The *queryhit* will route back to the creditor along the inverse of the *query* path. With such a message, the creditor can communicate with the intermediator, and then create a basic component. As shown in Figure 3.6, a tries to credit c, and sends a *query* to d. Since d doesn't know c, it forwards the

query to b. As b is a neighbor of c, it returns a *queryhit*. After receiving the *queryhit*, a contacts b and starts the credit process. For fast and low-overhead transmissions, a connects to b using UDP instead of TCP.

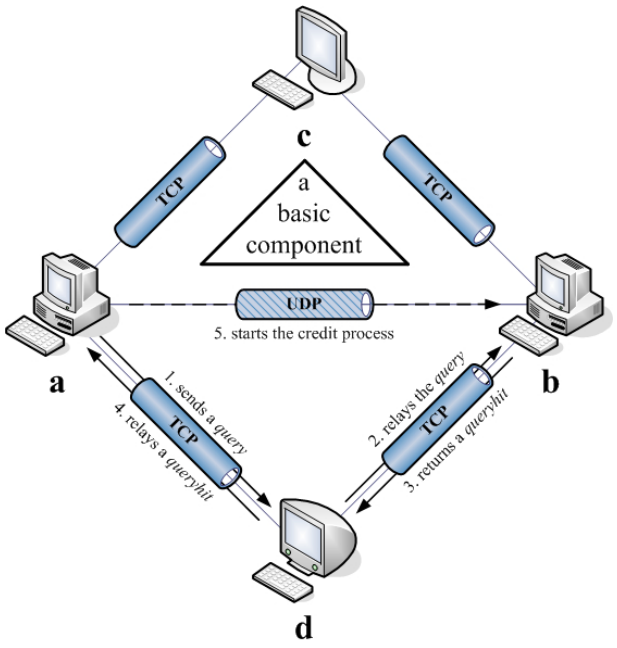

Fig. 3.6: The improvement for the credit method, showing that a, b and c form a basic component.

We discuss the setting of TTL to reduce the overhead caused by the *query* flood. We compute the number of peers hit by *query* at each hop by using the epidemic model [Luo et al., 2009], and then determine the smallest value of TTL. The number of peers hit by *query* at hop 1, h_1, is $p_{\rightleftharpoons}N$. The probability that a peer is not hit at hop 2 is $(1-p_{\rightleftharpoons})^{h_1}$. Thus, the number of peers hit at hop 2, h_2, is $\left(1-(1-p_{\rightleftharpoons})^{h_1}\right)(n-h_1)$. By analogy, the number of peers hit at hop TTL, h_{TTL}, can be written as:

$$h_{TTL} = \big(1 - (1 - p_{\rightleftharpoons})^{h_{TTL-1}}\big)(n - \sum_{i=1}^{TTL-1} h_i) \tag{3.1}$$

Suppose that the *query* aims to hit at least n_{hit} peers, the smallest value of TTL, TTL_{min}, can be determined by the following inequalities:

$$n_{hit} > \sum_{i=1}^{TTL_{min}-1} h_i, \quad n_{hit} \leq \sum_{i=1}^{TTL_{min}} h_i$$

3.5 Study of piece selection

In this section, we describe existing strategies that are used for piece selection. We then consider various factors affecting piece supply and demand to explain why rarest-first may fail to achieve its design goal of maximizing neighbor interest.

3.5.1 Piece selection strategies

There are two strategies that are commonly used in the BT protocol – *random* and *rarest-first*. In a pieceless mode, a peer selects pieces in a random order, called the *random* strategy, to quickly get as many complete pieces as possible so as to make others have interest in it. In a normal mode, a peer selects pieces in a rarest-first order, called the *rarest-first* strategy, to keep neighbors interested in it so that it can trade pieces using unchoking. Our discussions will mainly focus on rarest-first, because most of time for a peer to download is spent in the normal mode [Lilja Fjeldsted, 2005].

In marketing, a firm needs to increase its stock to meet the market demand. Similarly, the primary goal of piece selection is to maximize neighbor interest in order to make the piece supply meet the demand. Intuitively, the rarer the piece that a peer selects, the more interest it can gain. A question is raised here: Can rarest-first maximize neighbor interest? In the following sections, we show rarest-first is not good enough for piece selection by considering various factors influenced piece supply and demand.

3.5.2 Effect of download cost

Piece demand means not just how many peers download a given piece, but how many peers download that piece at its cost, and how many peers would download that piece if its cost changed. We give a simple example in Figure 3.7 to illustrate the effect of download cost on neighbor interest. For simplicity, we assume that neighbors are stable during the time period to select and download a piece, and, in each time slot, a peer only downloads one piece from one of its neighbors. In Example A, peer ℓ downloads slowly from its neighbor a (a half piece per time slot), and the rarest pieces are 1 and 3. ℓ prefers to select piece 3. Suppose all its neighbors select piece 1, ℓ will lose all neighbors' interest at the second time slot, because it cannot complete piece 3 before its neighbors complete piece 1. A rational choice for ℓ is piece 4 rather than piece 1, because a complete piece can make it more attractive to others. Choosing piece 4, ℓ can prolong a's interest.

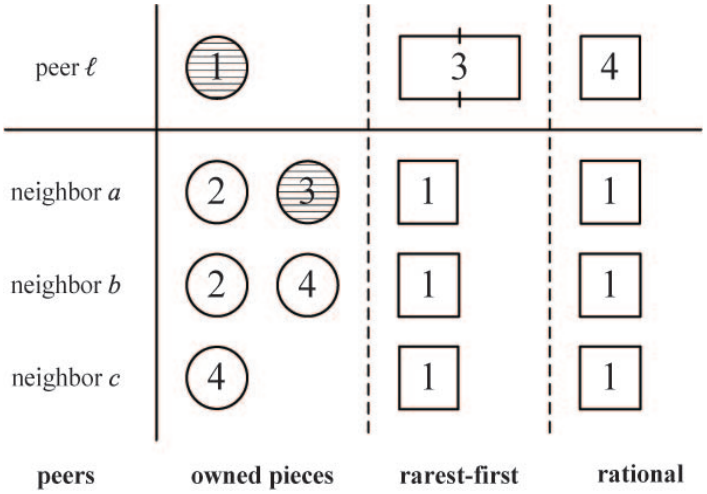

Fig. 3.7: Example A: ℓ will lose all neighbors' interest if it selects the rarest piece 3.

3.5.3 Effect of neighbor surplus

Neighbor surplus, occurring when quantity supplied is more than quantity demanded, is an important factor for peers to consider when entertaining thoughts about changing services that they offer. We give another example in Figure 3.8 to show the effect of surplus pieces on neighbor interest. In Example B, we see that pieces 3 and 4 are rarest. ℓ would like to choose piece 4. Suppose all its neighbors select piece 3, ℓ will lose a's interest at the second time slot. An interesting observation is that, if ℓ chooses piece 5 rather than 4, it will maintain mutual interest with all its neighbors. The reason is that either b or c has 2 surplus pieces from ℓ (ℓ can provide 2 pieces to either b or c), while a only has 1. For a further explanation, we quantity a neighbor's interest to ℓ by a utility function on the number of surplus pieces from ℓ. If ℓ selects piece 5, it will prolong a's interest. The number of surplus pieces for a will increase from 1 to 2. If ℓ chooses 4, it will garner interest from both b and c. For

each of them, the number of surplus pieces will go up from 2 to 3. The total utility increases along with the number of surplus pieces. Usually, we do not look at the total utility but at the *marginal utility* which denotes the increase in utility for one more surplus piece. Following *the law of diminishing marginal utility* [Merton, 1990], the marginal utility usually decreases as the number of surplus pieces increases. In other words, the utility of selecting piece 5 may be higher than that of selecting piece 4. We will give a more detailed discussion in Section 3.6.1, after we define the utility functions.

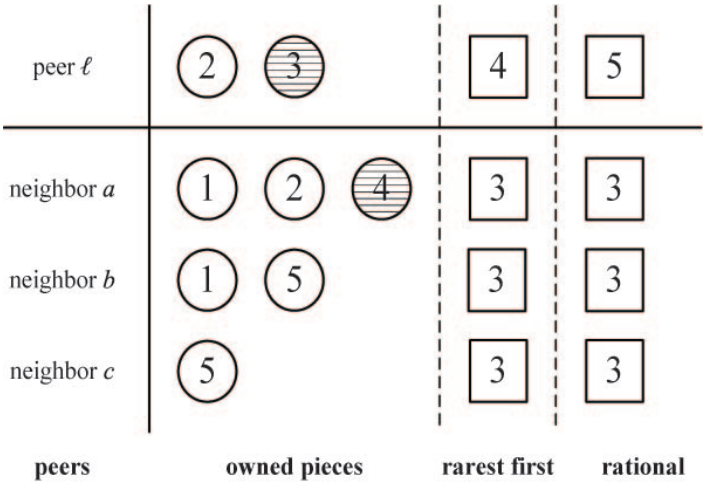

Fig. 3.8: Example B: ℓ will lose a's interest if it selects the rarest piece 4.

3.5.4 Effect of piece rarity

One of the biggest advantages of rarest-first might be the "high" data availability that avoids *local* rare pieces from appearing. Here's an interesting question: why peers should not emphasize too much on piece rarity when selecting a piece? We provide some answers to this question as follows. From the protocol perspective, maximizing

interest, which depends on several factors (e.g., download cost, neighbor surplus and piece rarity), could make unchoking work more effectively as well. Piece rarity is just *one* factor – not *all* of them. From the bandwidth usage perspective, rarest-first may waste limited upload bandwidths, because a rarest piece might not be available on a majority of neighbors who have ability to serve. From the measurement perspective, data availability might not be a critical problem in the BT system, though the system could be highly dynamic. Many real-word measurements [Guo et al., 2005, Legout et al., 2006, Yang and de Veciana, 2004] suggest that, in most swarms, there exists a significant number of seeders. Many peers stay for a few hours after their download is complete, and some peers even stay for days or weeks [Stutzbach and Rejaie, 2006]. From the overhead perspective, it is very costly to let each peer to have global knowledge of others. To be practical, each peer makes its own decision based on a limited view of the whole system. There is no guarantee that the *local* rarest piece will be the *global* rarest one. Overall, piece rarity has an important role in piece selection, but it might not be as important as what we used to believe. Rather than *always* selecting the rarest pieces, we suggest that a peer selects them only if needed.

3.6 A utility-driven strategy

In this section, we propose a utility-driven strategy to maximize neighbor interest based on the factors analyzed in the above section (e.g., download cost, neighbor surplus and piece rarity). We also discuss rules for choosing utility functions, and give some analysis of existing strategies.

3.6.1 Design of the utility-driven strategy

We assume that peer ℓ has N neighbors, and the file shared contains P pieces. We use n_i and p_j to represent the i-th neighbor and j-th piece respectively, where $\forall i \in \{1, 2, .., N\}, \forall j \in \{1, 2, ..., P\}$. If ℓ is not choked by n_i, ℓ will select and download a piece p_j if necessary. We use three terms, utility, cost and preference, to describe how a utility-driven strategy is made.

Neighbor's utility: as mentioned before, if n_i does not have p_j, ℓ can potentially prolong n_i's interest by downloading p_j. To determine whether n_i has p_j or not, we define an indicator variable $\alpha_i(p_j)$:

$$\alpha_i(p_j) = \begin{cases} 0 & \text{if } n_i \text{ does not have } p_j \\ 1 & \text{if } n_i \text{ has } p_j \end{cases} \tag{3.2}$$

From the viewpoint of n_i, there are two parameters, d_i and e_i, that may affect n_i's interest to ℓ. 1) d_i is n_i's full download speed. The larger d_i is, the sooner n_i loses its interest in ℓ. By observing the rate of *have* messages sent by n_i, ℓ can roughly estimate d_i. 2) e_i stands for the number of surplus pieces for n_i. The less e_i is, the more likely n_i becomes uninterested in ℓ. By considering these two parameters, we simply define n_i's utility function on e_i as a power function.

$$U_i(e_i) = -\frac{d_i}{e_i} \tag{3.3}$$

By Equation 3.3, we can measure n_i's interest to ℓ. We stress that this utility function

can be re-defined according to different application requirements. We will discuss the rules for choosing them in Section 3.6.2. Suppose that n_i does not have p_j, namely, $\alpha_i(p_j) = 0$, e_i will increase by 1, if ℓ downloads p_j. The marginal utility $MU_i(e_i)$ can be calculated as:

$$MU_i(e_i) = \frac{d_i}{e_i(e_i + 1)} \tag{3.4}$$

From Equation 3.4, we see that an additional e_i provides smaller and smaller increases in utility (*diminishing marginal utility*).

Download cost: once ℓ selects p_j, it will split the piece into $\frac{S_p}{S_b}$ blocks, where S_p and S_b represents the size of piece and that of block, respectively. Then, ℓ requests blocks from the neighbors who have p_j and unchoke it. Here, we define another indicator variable β_i.

$$\beta_i = \begin{cases} 0 & \text{if } n_i \text{ chokes } \ell \\ 1 & \text{if } n_i \text{ does not choke } \ell \end{cases} \tag{3.5}$$

As blocks can be queued and downloaded in parallel, it is difficult to estimate the time cost to download p_j, denoted as $C(p_j)$. Roughly speaking, there are three parameters, $b_i(\cdot)$, $b_i(p_j)$, and u_i, that may have some effects on $C(p_j)$. 1) $b_i(\cdot)$ is the number of blocks queued at n_i. Before getting a new block from a neighbor, ℓ needs to wait responses for all blocks queued at that neighbor. 2) $b_i(p_j)$ denotes the number of blocks from p_j assigned to n_i. 3) u_i represents n_i's upload speed to ℓ. We capture the minimum $C(p_j)$ as follows:

$$C(p_j) = min\Big(max\big\{\alpha_i(p_j)\beta_i S_b \frac{b_i(\,\cdot\,) + b_i(p_j)}{u_i}\big\}\Big)$$
$$s.t. \sum_{i=1}^{N} b_i(p_j) = \frac{S_p}{S_b} \quad and \quad \forall i : b_i(\,\cdot\,) \leq \frac{S_p}{S_b} P \tag{3.6}$$

This is a parallel scheduling problem which is known as a NP problem [Fogel, 1993]. In this work, we simplify the calculation of $C(p_j)$ in order for easy implementation and real-time piece selection. In particular, we ignore the queued pieces, and assume uniform block assignment, namely, $b_i(\,\cdot\,) = 0$, and $b_i(p_j) = \frac{S_p}{S_b \sum_{i=1}^{N} \alpha_i(p_j)\beta_i}$. Then, we let $C(p_j)$ to be the average download cost:

$$C(p_j) = \frac{S_p}{\sum_{i=1}^{N} u_i \alpha_i(p_j)\beta_i} \tag{3.7}$$

Peer's preference: a peer's utility is associated with utility functions of all its neighbors, as interest from each individual neighbor is the basic unit for aggregating to its utility. This is very close to the concept of *social welfare* in welfare economics. We define ℓ's utility $U_\ell(p_j)$ as:

$$U_\ell(p_j) = \frac{\sum_{i=1}^{N} \big(1 - \alpha_i(p_j)\big) MU_i(e_i)}{C(p_j)} \tag{3.8}$$

In Equation 3.8, we consider both download cost and neighbor surplus. By considering the download cost, we can speed up piece completion time. Note that the most common piece may not be the one with the lowest cost, if peers upload at different

speeds. By considering the neighbor surplus, we can achieve a *win-win* cooperation between a peer and its neighbors. For the peer, it is less likely to be choked, because it can continuously contribute its upload. For the neighbors, most of them can find what they need from the peer. Thus, all peers can benefit from selecting the piece that maximizes $U_\ell(p_j)$.

Given any two pieces p_j and p_k for $j \neq k$, if $U_\ell(p_j) > U_\ell(p_k)$, ℓ prefers p_j to p_k, denoted as $p_j \succ p_k$. If $U_\ell(p_j) = U_\ell(p_k)$, ℓ is indifferent between p_j and p_k, denoted as $p_j \sim p_k$. In such a case, ℓ can select either of them.

Examples: we use examples discussed in Section 3.5 to show that a peer could make a rational choice if it employs the utility-driven strategy.

In Example A, we know that $u_a = \frac{1}{2}$, and $u_b = u_c = 1$. Assume ℓ only downloads from one of its neighbors, we have $C(p_3) = 2$ and $C(p_4) = 1$. If ℓ selects p_3, it will gain interest from b and c. We get $U_\ell(p_3) = \frac{\frac{2}{1 \cdot 2}}{2}$. If ℓ chooses p_4, it will attract a's attention. We have $U_\ell(p_4) = \frac{1}{1 \cdot 2}$. Thus, $U_\ell(p_3) = U_\ell(p_4)$, and $p_3 \sim p_4$. That means ℓ can choose either of them. But, ℓ will definitely choose p_3, if it employs rarest-first.

In Example B, we can see that ℓ has two pieces, p_2 and p_3. Since it already has p_2, a needs p_3 only. Other two neighbors need both p_2 and p_3. Thus, we have $e_a = 1$, $e_b = e_c = 2$. Suppose $C(p_4) = C(p_5) = 1$, if ℓ selects p_4, it will become more attractive to b and c. Then, we have $U_\ell(p_4) = 2\frac{1}{2 \cdot 3}$. If ℓ selects p_5, it will keep a interested. We get $U_\ell(p_5) = \frac{1}{1 \cdot 2}$. Therefore, $U_\ell(p_5) > U_\ell(p_4)$, and $p_5 \succ p_4$. ℓ will select p_5, which is the result expected.

3.6.2 Rules for choosing utility functions

We provide a guide to choosing utility functions based on function properties. When a peer cares more about data availability, it can set $U_i(e_i)$ to be a function with slowly diminishing $MU_i(e_i)$ (e.g., $U_i(e_i) = e_i$). In this way, $\alpha_i(p_j)$ will play an important role in $U_\ell(p_j)$, which reflects the rarity of pieces. The selected pieces will tend to be those that appear least frequently in bitfields. When the peer cares more about neighbor interest, it can set $U_i(e_i)$ to be a function with rapidly diminishing $MU_i(e_i)$ (e.g., $U_i(e_i) = -e_i^{-1}$ or $U_i(e_i) = -e_i^{-2}$). In doing so, e_i will become a main parameter in $U_\ell(p_j)$, which is the quantity of surplus pieces. The selected pieces will be biased to those that are likely to be requested in the future. To balance the two, a peer can choose a function with moderately diminishing $MU_i(e_i)$ (e.g., $U_i(e_i) = e_i^{0.5}$ or $U_i(e_i) = \log(e_i)$). As mentioned earlier, we should emphasize more on neighbor interest, because maximizing interest is the prime goal of piece selection.

3.6.3 Analysis of existing strategies

We firstly look at the utility functions chosen by rarest-first and random, respectively. Assume that the download cost for each piece is constant. The rarest-first strategy always selects the least common piece. Therefore, we have $U_\ell(p_j) = \sum_{i=1}^{N}(1 - \alpha_i(p_j))$, namely, $MU_i(e_i) = 1$. We can simply set $U_i(e_i) = e_i$. The random strategy treats all pieces equally. Hence, we get $U_\ell(p_j) = 0$, that means, $MU_i(e_i) = 0$. We can let $U_i(e_i) = 0$. Observing these utility functions, we find that both rarest-first and random pay little attention to neighbor surplus, because their $MU_i(e_i)$ is constant rather than diminishing. We also note that random rarely considers piece rarity, because its $U_\ell(p_j)$ is also constant.

We then discuss the download cost to a selected piece in the two strategies. Assume that ℓ has no piece; all neighbors unchoke ℓ, and upload with an equal speed. The download cost of the rarest piece is $max\{\frac{S_p}{\sum_{i=1}^{N}\alpha_i(p_j)}\}$, while that of a random one is $\sum_{j=1}^{P}\frac{S_p}{P\sum_{i=1}^{N}\alpha_i(p_j)}$. Compared with the rarest piece, the random one may have a lower download cost.

Through the above analysis, we show that the utility-driven strategy covers both rarest-first and random, which can make the protocol more clear and flexible.

3.7 Study of piece queuing

In this section, we describe the conflict in piece queuing, and give a possible fix to address the conflict.

3.7.1 A conflict in piece queuing

The design of piece queuing algorithm is still under dispute [TheoryOrg, 2004]. A peer tends to select a least common piece for more interest from neighbors, while it expects to queue the requests for a selected piece at each neighbor for the better use of bandwidth. If the number of queued requests is configurable, it is not possible to assume that requests for a selected piece are always spread over multiple neighbors. The scatter algorithm suggests that a peer maintains a queue of blocks to be requested. Such an algorithm is simply a matter of going through the following steps:

1. When a neighbor connection needs to queue a new request, a peer will first search through the request queue for the requests that can be satisfied by the neighbor.

2. If none can be found, it will select a piece using random or rarest-first, and then, divide the selected piece into requests and add them to the request queue and finally select one of the newly added requests from the request queue.

A flaw in the algorithm is that a peer may still fail to find a request that can be satisfied by the certain neighbor after step 2. Even though it can finally find one (assume the size of request queue is very large), the peer cannot predict how many piece will be added into the request queue. Figure 3.9 shows an example of how the scatter works. We see that b and c are uploading piece 3 to ℓ, while a is waiting for the requests from ℓ. If ℓ uses rarest-first, it will queue rarest pieces 2 and 4 before piece 1, which are not available on a. We believe that it is not reasonable to insert too many pieces at a time, because, as time goes by, the queued pieces may become suboptimal to download. Besides that, the more pieces are queued for downloading, the more memory is allocated to incomplete pieces.

3.7.2 A revised queuing algorithm

We revise the scatter algorithm to address the conflict in piece queuing. We suggest that, in step 2, a peer directly selects a piece that the neighbor has. Then, no matter what piece selection strategy is used, we can classify queued pieces into two groups: *rational* and *sub-rational*. Rational pieces are selected from those are available on at least one neighbors, while sub-rational pieces are selected from those are available on the certain neighbor. By adjusting the ratio between the two, a peer can dynamically control the number of queued pieces. In particular, it can scan the request queue before step 1, and add several rational pieces if needed. Note that the maximum number of queued pieces is bounded by that of unchoked neighbors.

For example, if the ratio is set to 1, the maximum number of queued pieces will not excess two times that of unchoked neighbors. Figure 3.9 illustrates how the revised algorithm works. We can see that, if piece 3 is rational, ℓ will only queue piece 1 which is available on a.

The revised algorithm ensures fast piece completion time because of the efficient use of neighbor bandwidths. In addition, it reduces the memory allocated to incomplete pieces, and satisfies the real time requirement in piece selection, because the number of queued pieces is not only configurable but also controllable.

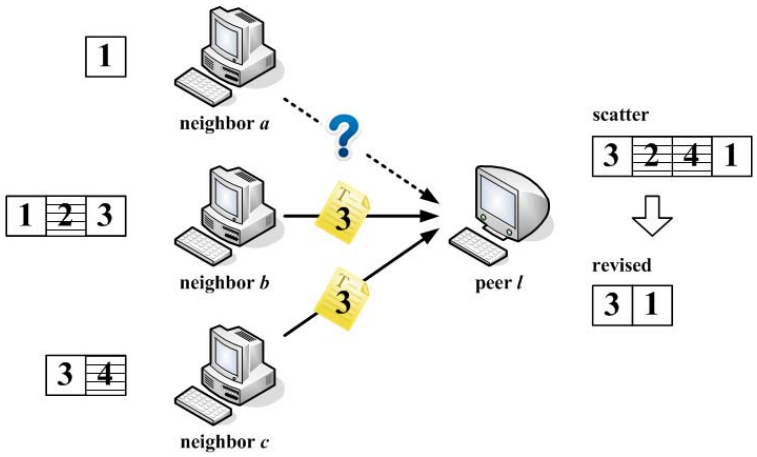

Fig. 3.9: The queuing process, showing that ℓ can reduce memory consumption by queuing piece 1.

3.8 Performance evaluation

In this section, we evaluate the effectiveness and performance of the proposed algorithms through experiments. Our experiment results show that the proposed algorithms can achieve both *individual good* and *social good* when compared with

previous well-known ones (e.g., rarest-first, random and scatter).

3.8.1 Experimental setup

We implemented a java-based BT client referring to [Lilja Fjeldsted, 2005, TheoryOrg, 2004], and modified it to realize the credit method, utility-driven strategy and revised queuing algorithm. We decide to implement a client rather than use an existing one for the following reasons: 1) We want to know more about both the BT protocol and technical details. 2) Some algorithms in mainstream clients (e.g., CTorrent and Vuze) are different from what specified in the BT protocol, which may affect the comparison results. We upload our source code to FTP at [Luo, 2010] for research and testing purposes.

We run all our experiments in a controlled environment consisting of 4 hosts. Each host has 1.86GHz Intel Core 2 CPU, 2 GB RAM, and 1 GigE connection. One host installs a local tracker. Each host runs 4 BitComet clients [BitComet, 2009] and 30 java-based clients. The BitComet clients acted as initial seeders. Unless otherwise specified, we use the following default parameters. Each peer has up to 10 neighbors, and uploads at 128KB/s. Peers share a 20MB file which is fragmented into 128KB pieces. The maximum size of the request queue is 10 pieces, namely, 80 blocks to be requested. Each peer leaves the swarm as soon as it has completed its download. Each point in the figures that follow represents the average over at least 10 runs, and error bars denote 95% confidence intervals. When comparing with rarest-first or random, we let $U_i(e_i) = -e_i^{-1}$ for utility-driven. Experiments on a large-scale platform or the PlanetLab testbed will be our future work.

3.8.2 Effectiveness of the distributed credit method

We start to show that the credit method increases the download time of under-reporters. We randomly choose a peer to under-report its pieces. Each peer randomly selects a subset of neighbors to credit their honesty. Thus, each peer has a probability of being credited by another peer. If a peer suspects that a neighbor under-reports its pieces, the peer will choke that neighbor. Figure 3.10 shows that the download time of the under-reporter increases with the probability of being credited. For example, the download time increases by 21.8% when the probability increases from 0 to 0.2. The under-reporter downloads slower, because it is more likely to be detected and punished by others if the probability is higher.

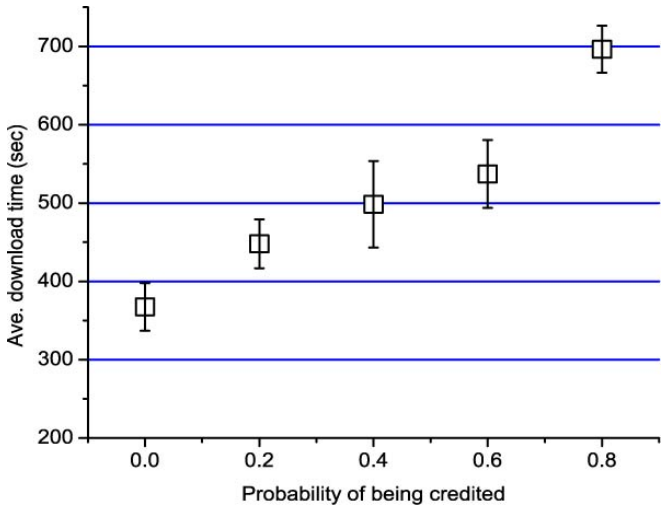

Fig. 3.10: Download time of under-reporters.

We then measure the overhead of the credit method caused by the *query* flood. We set the TTL to 2. By Equation 3.1, we know that the numbers of peers hit at hop 1 and 2 are 10 and 73, respectively. In other words, the broadcast of *query*

covers 69% peers with TTL of 2. Figure 3.11 illustrates that the number of overhead messages increases along with the probability of being credited. When the probability increases from 0.2 to 0.4, the number of overhead messages per peer is doubled. To punish under-reporters to the greatest extent while maintaining low overhead, a peer should credit each neighbor with a high probability, and broadcast *query* with a small TTL.

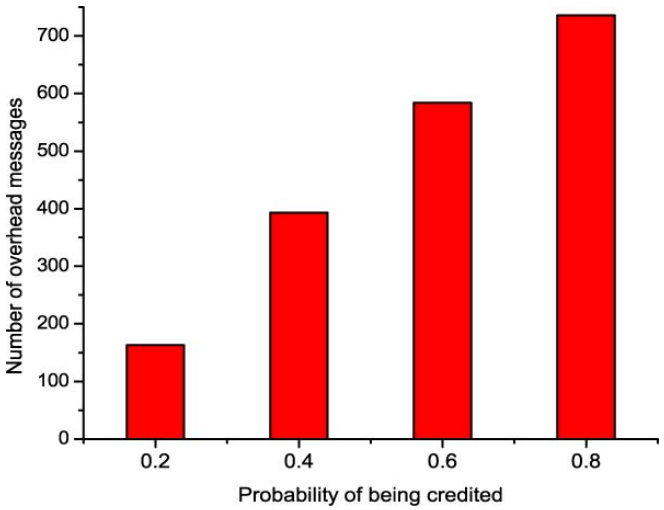

Fig. 3.11: Number of overhead messages.

3.8.3 Performance of the utility-driven strategy

We begin to show that the utility-driven strategy reduces the download time of an individual peer. The first experiment runs in the system consisting predominately of rarest-first. This experiment requires 6 runs. In each run, all but one peer start at the same time and run rarest-first. The remaining peer starts 30 seconds after all the other peers, and adopts either random or utility-driven. From Figure 3.12,

we can see that utility-driven performs better than the other two strategies. For example, compared with rarest-first and random, utility-driven reduces 37.3% and 11.3% individual download time, respectively. The second experiment runs in the swarm where all peers use the same strategy. This experiment also consists 6 runs. In each run, all peers run the same strategy and start at the same time. After each run, all peers change their strategy or utility function, and then, continue the experiment. Figure 3.13 shows that there is an obvious decrease of individual download time, when all peers adopt utility-driven. As we can see that the individual download time for utility-driven is 41.8% and 35% lower than that for rarest-first and that for random, respectively. The main reason is that neither rarest-first nor random considers the download cost that may result in slow piece completion time and inefficient bandwidth usage.

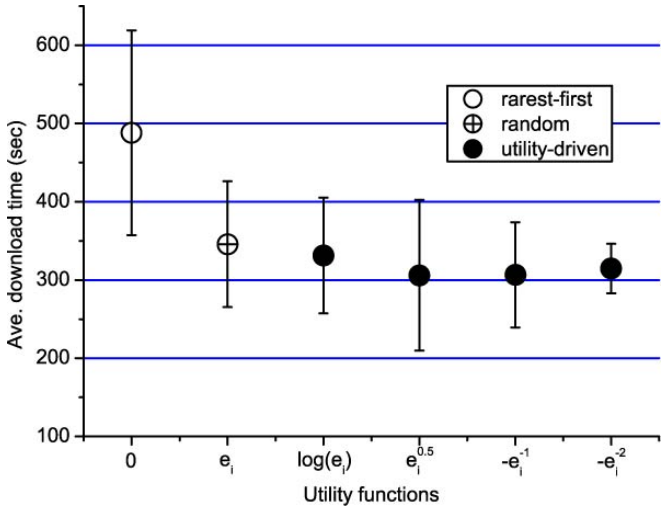

Fig. 3.12: Comparison of individual performance in the swarm where majority of peers use the rarest-first strategy.

We then illustrate that the utility-driven strategy saves average download time

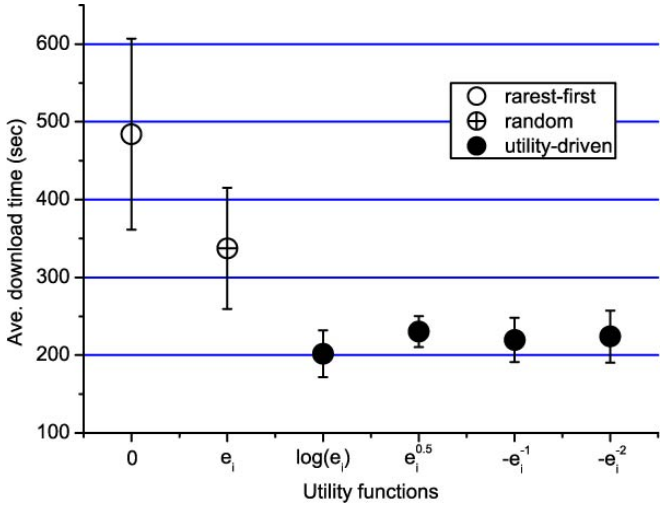

Fig. 3.13: Comparison of individual performance in the swarm where all peers use the same strategy.

of all peers. In the first experiment, we assume all peers adopt the same strategy. Figure 3.14 shows the power of maximizing neighbor interest. As we can see that utility-driven saves 57% and 41.3% system-wide download time when compared with the other two strategies. We also note that random outperforms rarest-first because of the better randomization of available pieces and the faster completion time of selected pieces. This is one possible reason why the updated official protocol [Cohen, 2008] suggests that peers generally download pieces in a random order. In the other experiment, we vary the ratio between utility-driven and rarest-first. Figure 3.15 depicts that the trend of average download time is a clear downtrend when utility-driven becomes more popular. When the ratio increases from 0.2 to 0.4, the average download time for utility-driven and that for rarest-first decrease by 14.2% and 17.7%, respectively. Another notable observation is that utility-driven is always better than rarest-first. When the ratio is 0.2, the average download time for utility-driven is

almost a half of that for rarest-first.

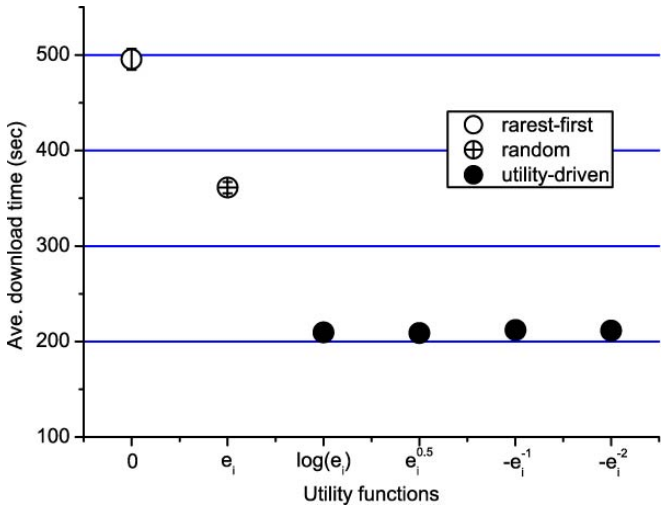

Fig. 3.14: Comparison of overall performance.

We conclude that an individual peer can benefit from utility-driven. It is a nature that a single peer who uses utility-driven can perform well in a rarest-first dominated swarm. Moreover, the strategy is incrementally deployable. The more widely it deployed, the better overall performance can be achieved.

3.8.4 Performance of the revised queuing algorithm

We firstly show that the revised queuing algorithm saves the average download time of all peers. As mentioned before, the revised algorithm will add several rational pieces to the request queue, if the ratio between rational and sub-rational pieces is lower than a threshold. We vary the threshold ratio from 0 to 3. Figure 3.16 illustrates that, no matter what piece selection strategy is used, the revised algorithm performs better than scatter. When all peers employ rarest-first, the revised algorithm reduces

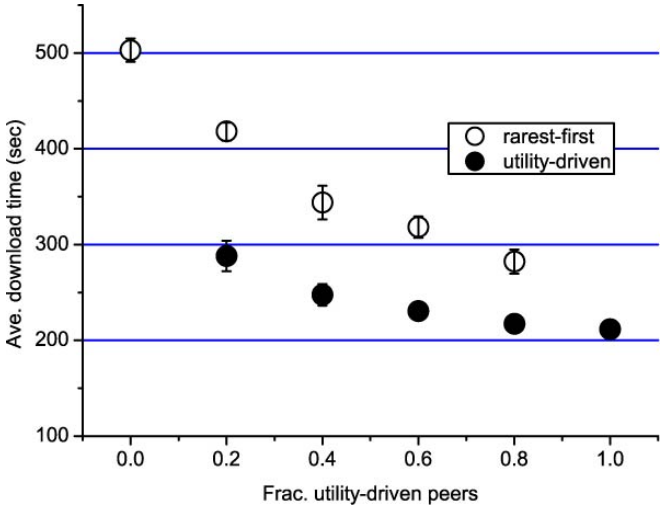

Fig. 3.15: Effect of strategy ratio on overall performance.

62.3% system-wide download time. Another interesting observation is that the overall performance decreases when more rational pieces are selected. The reason is that each peer scatters available neighbor bandwidths more widely on multiple pieces. To benefit the whole system, it is necessary for each peer to queue several rational pieces.

We then depict that the revised queuing algorithm reduces memory allocated to incomplete pieces. We check the request queue of each peer every second. Figure 3.17 shows that the revised algorithm significantly reduces the average number of queued pieces. When the threshold ratio is 1 and all peers use rarest-first, the average number of queued pieces decrease by 44.4%. We suggest that the threshold ratio can be 1 or 2, because a bigger threshold ratio may lead to higher memory consumption.

We conclude that the revised queueing algorithm improves overall performance while ensuring low memory consumption.

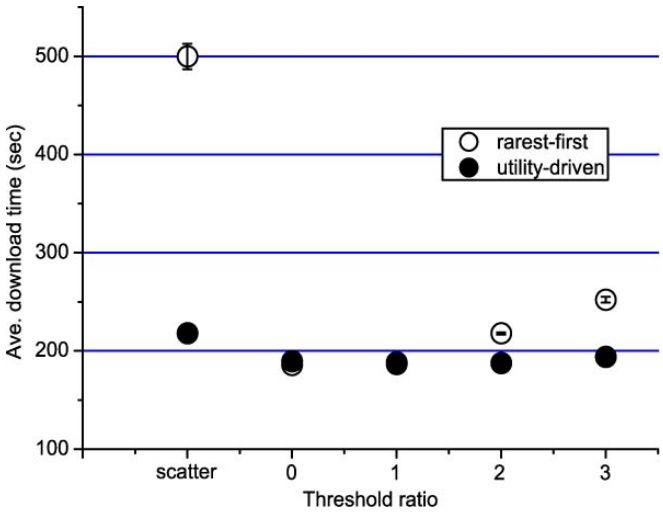

Fig. 3.16: Effect of threshold on overall performance.

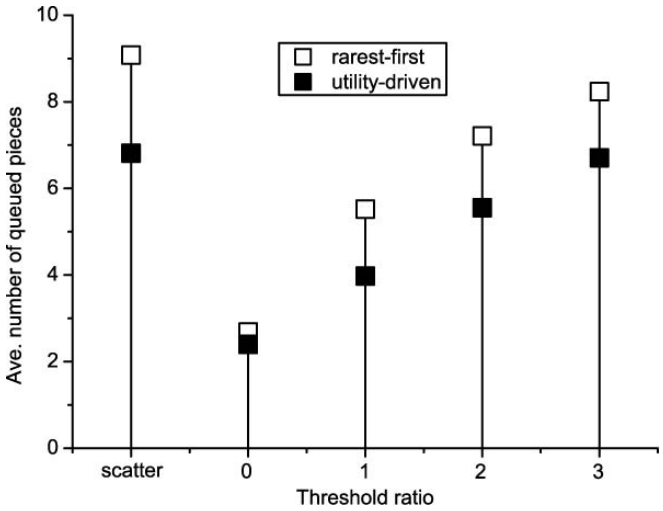

Fig. 3.17: Comparison of memory consumption.

Chapter 4

Modeling and Containing the Adaptive BitTorrent Worm in BitTorrent-like Systems

4.1 Overview

In this Chapter, we present a powerful BT worm, called *Adaptive BitTorrent worm* (A-BT worm), to identify potential security problems in P2P systems. Such a worm finds new victims by sending forged requests to a tracker. To reduce its abnormal behavior, it sends out only one forged request within a defined time window. By adjusting the time window size, it can adaptively control its propagation speed. The A-BT worm is a potential threat to BT-like systems for it has the following properties:

- **Timeliness**: it has the ability to locate the most recent active peers even in a highly dynamic BT system.

- **Feasibility**: it is easy to be implemented by taking advantages of the high penetration of the BT protocol.
- **Adaptivity**: it can significantly reduce its abnormal behavior by adaptively adjusting its propagation speed.

We build a hybrid model by combining the fluid model with the epidemic model to estimate the damage caused by the worm. The fluid model evaluates the number of peers that could be infected in a swarm. The epidemic model measures the number of peers that have been infected. Our model shows the effect of system parameters on the worm propagation.

We propose a statistical method to automatically detect the worm from the tracker by estimating the variance of the time intervals of requests. A normal peer usually requests at a stable interval, while an infected peer may request at a variable interval. To slow down the spread of the worm, we design a strategy in which the tracker returns a biased list of peers (e.g., more secured peers) when recieves a request.

Our simulation results show that our hybrid model precisely depicts the worm damage (number of infected peers in a swarm). The A-BT worm can propagate as fast as the topological P2P worm, more importantly, it is unaffected by the P2P topology. Our statistical method is effective to detect the worm, though the worm can minimize its abnormal behavior.

The rest of this Chapter is organized as follows. Section 4.2 reviews the related work, and shows some protocol details. Section 4.3 describes how the A-BT worm propagates itself and adaptively adjusts its propagation speed. Section 4.4 builds a hybrid model to measure the worm damage. Section 4.5 proposes a statistical method and a safe strategy to detect and contain the worm. Section 4.6 validates the hybrid

model through simulations, and compares the A-BT worm with the topological P2P worm.

4.2 Related work and protocol details

In this section, we review the pervious work related to P2P worms, and show some details of the peer-to-tracker communication in the BT protocol.

4.2.1 Related work

P2P worms find targets and propagate themselves in P2P systems. To throttle against P2P worms, we need to investigate them in terms of: 1) target finding strategies, 2) worm modeling, and 3) worm containment.

Target finding strategies: P2P worms have various ways to locate potential victims in P2P systems. Three strategies are commonly used for target finding. ***Passive***: passive P2P worms, such as Gnuman worm [Vamosi, 2001] and Banjamin worm [Singer, 2002], attach themselves to shared files and wait for potential victims to discover and download them. ***Topological***: topological P2P worms propagate through P2P neighbors. These worms can be further classified into two categories: reactive and proactive. 1) *Reactive* worms, also called contagion worms, propagate parasitically along with normal communications. 2) *Proactive* worms achieve much faster propagation speed by directly connecting to other peers, whose addresses were cached after previous file requests and sharing. ***Hit list***: some open source P2P clients, such as Gnutella and BT, can be explored to generate a hit list. The modified clients can crawl P2P systems actively to discover as many hosts as possible.

P2P worm modeling: modeling P2P worms can help us to understand factors affecting the spread of P2P worms. Most existing models originate from the epidemic model, but they have different focus. ***Propagation process***: some models focus on how a worm propagates itself. Two states, *susceptible* and *infected*, are considered in a simple passive P2P worm model [Ma et al., 2006]. Two additional states, *exposed* and *recovery*, are added in SEI and SEIR models [Yao et al., 2006]. TF-SEI (Two-Factor-SEI) model considers a *quarantine* state for anti-worms. Both on-line and off-line behavior are discussed in [Thommes and Coates, 2006]. ***Propagation environment***: other models focus on where a worm propagates itself. K. Ramachandran, et al. [Ramachandran and Sikdar, 2006] argue that the assumption in [Thommes and Coates, 2006] may not be true in Gnutella-like systems that a vulnerable peer can be infected by any of infected ones. J. Luo, et al. [Luo et al., 2009] models a dynamic hit list worm which only propagates in BT-like systems.

P2P worm containment: P2P worm containment methods aims to slow down the worm propagation. There are three strategies for P2P worm containment. ***Self-defense infrastructure***: self-defense infrastructure is a secure P2P architecture that can prevent or isolate P2P worms. Two methods are proposed to design such architecture in P2P networks. 1) *Guardian nodes* can be set to automatically detect P2P worms [Zhou et al., 2005]. Though simple it is, this method is hard to be implemented because the wide scale deployment of guardian nodes is resource consuming. 2) *Software diversity* can also be used to slow down P2P worms. In Verme [Freitas et al., 2007], all peers are divided into different groups, called *islands*, according to the type of vulnerabilities. To help peers select neighbors, D. McIlwraith, et al. [Douglas et al., 2008] employ a server, called di-jest server, to compute the distance of each

peer pair which corresponds to the infection probability of a worm. Y. Zhou et al. [Zhou et al., 2006] give a model to study the effect of software diversity on P2P worm propagation. However, these methods are not practical, because software developers may not willing to provide several clients for the same protocol. ***Worm-anti-worm***: the anti-worm spreads itself using the same mechanism as the malicious worm [Yao et al., 2008]. This method has some obvious limitations. One is that the spread of anti-worm consumes a lot of bandwidth. The other is that most computer crime laws do not distinguish "worms" from "anti-worms". ***Feedback control***: the feed back scheme [Wu and Feng, 2006] slows down P2P worms by delaying a peer's request when the peer tries to make connections at a high rate. Unfortunately, in P2P systems, a normal peer may also attempt to connect a large number of peers for a high download rate. It is difficult to distinguish the abnormal behavior from the normal one.

4.2.2 Peer-to-tracker communication

To exploit the BT protocol for P2P worm design, we are interested in the details of the communication between a peer and a tracker. The *HTTP GET* request sent by a peer contains a tracker's URL and some CGI parameters. URL and parameters are separated by "?". Parameter and value are separated by "=". Different parameters are separated by "&". Figure 4.1 illustrates an example of a request:

Note that only the first three parameters are indispensable. *info_hash*, is the hashed value of the information field in a .torrent file. A tracker can classify peers into different swarms according to this parameter. *peer_id* is the ID of a peer and parameter *port* defines its port used for file sharing. Another useful parameter, *numwant*,

Fig. 4.1: An example of the HTTP GET request.

indicates the number of peers to be returned from a tracker to a requesting peer. Its default value is 50, but it could be much larger. For example, it is 200 in BitComet, which is a popular BT client.

After receiving a request, the tracker sends back a response that includes: a list of active peers (*peer_id*, *IP* and *port*), number of seeders, number of leechers, and time interval between two requests. The sum of seeders and leechers is the size of a swarm, showing the total number of peers sharing a particular file. The time interval between two requests is 20 minutes. If the tracker doesn't receive a request from a peer within 30 minutes, it will assume that the peer has left the swarm, and delete the corresponding record.

An interesting observation is that the tracker distinguishes different peers depending on their *IP* and *port* rather than *peer_id*. In other words, a tracker will add a new record only if it receives a request containing a new pair of *IP* and *port*.

4.3 A-BT worm design

In this section, we describe the strategies used by the A-BT worm to find next targets and reduce abnormal behavior.

4.3.1 Target finding strategy

An A-BT worm finds potential victims in a BT system by mimicking a normal client requesting a tracker. Figure 4.2 shows that peers P_a, and P_b sharing file 1 are in swarm A. Peers P_c and P_d sharing file 2 and 3 are in swarm B and C, respectively. Assume that P_a and P_b are vulnerable. Figure 4.2 illustrates the propagation of the A-BT worm in swarm A, which contains the following steps:

1. The peer who initiates the worm propagation, called *initiator*, downloads a .torrent file from a web server (e.g., a .torrent file related to file 1), which contains a URL list of trackers.

2. The initiator contacts with a tracker known from the list by sending an *HTTP GET* request.

3. The tracker responds to the request by returning a list of active peers who are sharing file 1. For example, it returns a list containing information of P_a, such as *IP*, *port*, and *peer_id*.

4. After obtaining P_a's information, the initiator forwards a worm copy to P_a. The payload of the worm copy contains an *HTTP GET* string including *URL*, *port*, and *peer_id*. Note that *peer_id* can be randomly generated, but *port* should be the same with the one used by P_a. The reason is that a different *port* may create a new record on the tracker.

After infected, P_a generates a forged request by using the *HTTP GET* string in the worm payload, and then attempts to infect P_b by repeating steps 2, 3 and 4.

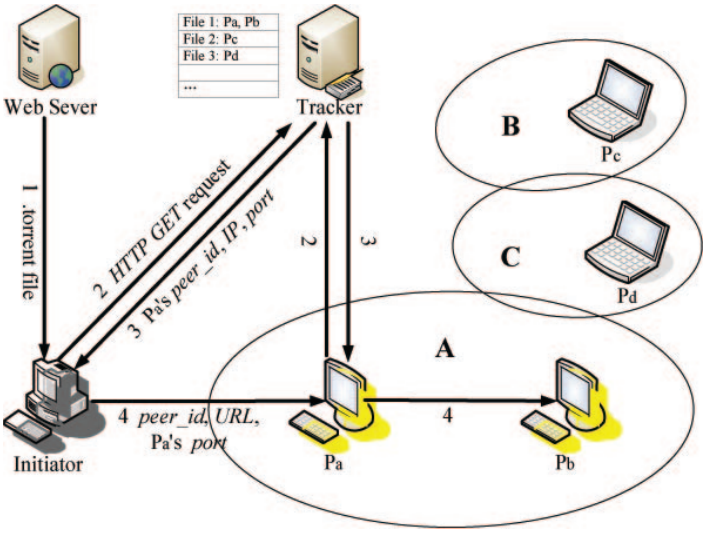

Fig. 4.2: The A-BT worm propagation in swarm A.

4.3.2 Adaptive speed control

An A-BT worm has the ability to adaptively adjust its propagation speed, which makes itself more stealthy. If an infected peer generates more forged requests to the tracker, it will find more victims and propagate faster, but will take more risk to be detected. The total number of requests is a time-varying quantity, which can be regarded as a contiguous-time signal. To justify the burst of received requests, the tracker can sample the signal at a fixed time interval, called *sampling period*. The sequence of samples can be represented by a function of a *time index* n. The tracker can easily detect the worm by measuring the request rate of peers, and slow down the worm by simply shutting down itself for a while or blacklisting suspected peers.

To reduce abnormal behaviors while maintaining a reasonable speed to propagate worms, the infected peer delays a certain amount of time before sending a forged request. As shown in Figure 4.3, it setups a fixed time window and requests the

tracker at random time within the time window. We name the worm using a fixed time window as the *regular BitTorrent worm* (R-BT worm). Assume that a time window covers W sampling periods, called *window size*, the probability β that an infected peer generates a forged request within a sampling period is $\frac{1}{W}$.

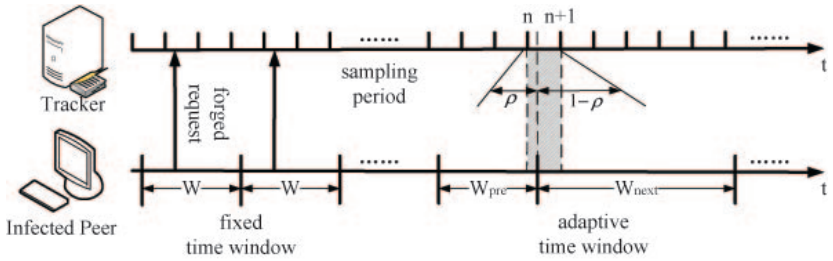

Fig. 4.3: The adaptive speed control for the A-BT worm.

By borrowing some ideas from the network congestion control, the infected peer adaptively and contiguously adjusts the time window size to control the worm speed. Instead of keeping a fixed time window, it determines the next time window size W_{next} depending on the rate of being accessed by other infected peers, denoted as A_{pre}, within the previous time window W_{pre}. A high A_{pre} indicates that a large fraction of peers has been infected in the swarm. If A_{pre} is smaller than a threshold A_{th}, the infected peer decreases the time window size linearly. Otherwise, it increases the time window size exponentially. Thus, we have:

$$W_{next} = W_{pre} - \phi T_{life}, \qquad if\ A_{pre} < A_{th} \tag{4.1}$$

$$W_{next} = \varphi W_{pre}, \qquad if\ A_{pre} \geq A_{th} \tag{4.2}$$

where T_{life} is the swarm lifespan [Guo et al., 2005], which can be viewed as a constant for a particular swarm. ϕT_{life} represents the decreasing amount of the time window

size for $0 \leq \phi \leq \frac{W_{pre}}{T_{life}}$. φ is an increasing ratio of the time window size for $\varphi \geq 1$. Note that there is no requirement for the synchronization between infected peers and the tracker.

From Figure 4.3, we can see that the n-th sampling period is separated into two parts ρ and $(1-\rho)$ for $0 \leq \rho \leq 1$. The probability that the infected peer sends a forged request within the n-th sampling period β is $\frac{\rho}{W_{pre}} + \frac{1-\rho}{W_{next}}$. During the interval $[n+1, n+W_{next}-1]$, we have $\beta = \frac{1}{W_{next}}$.

The A-BT worm is different from the R-BT worm in that the probability β for A-BT worm is determined by Equations 4.1 and 4.2, rather than a constant. In fact, the R-BT worm can be regarded as a special case of the A-BT worm with $\phi = 0$ and $\varphi = 1$.

4.4 A-BT worm modeling

In this section, we firstly describe some terminology and notations used in our model. We then formulate a hybrid model to display the propagation of the A-BT worm in a swarm.

4.4.1 Terminology and notations

Before describing our model, we define some basic terminology and notations. Figure 4.4 shows a complete classification tree of vulnerable peers during an infection. The vulnerable peers are classified into two categories: *infected peers*, which are vulnerable and infected, and *uninfected peers*, which are vulnerable but not infected.

The *infected peers* can be further classified into two categories: *effective peers*,

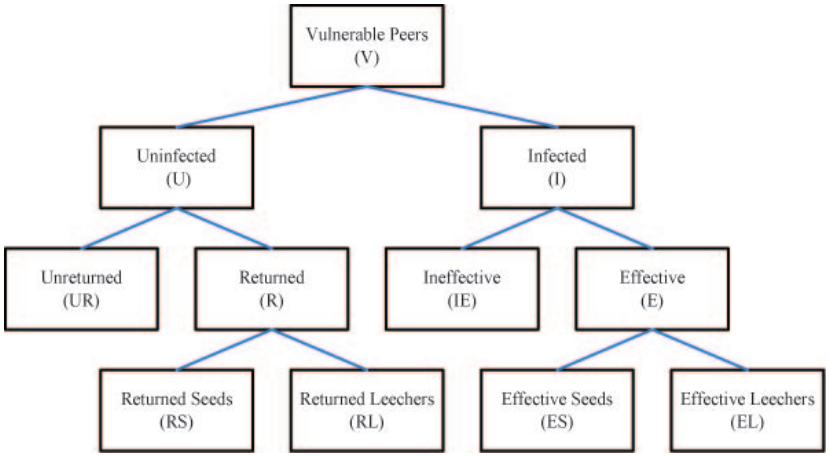

Fig. 4.4: The classification tree of vulnerable peers.

which are active propagating worms, and *ineffective peers*, which stop propagating worms. This classification is based on an assumption that infected peers who leave the swarm are not effective any more. Such an assumption can make the swarm size predictable, and thus, simplify our model. All effective peers will eventually leave the swarm and become ineffective. Based on whether they have a completed file or not, we further group effective peers into two subcategories: *effective seeders* and *effective leechers*.

The *uninfected peers* can be classified into two categories according to whether or not they have been returned in a request: *returned peers* and *unreturned peers*. A returned peer can be either a seeder or a leecher, called *returned seeder* or *returned leecher*, respectively.

Figure 4.5 shows the class transition diagram of peers. Assume that there is no delay for state transition. An uninfected peer will instantly become infected when an infected peer locates it through the tracker. Similarly, a returned seeder (leecher) will directly become an effective seeder (leecher). We suppose that the increasing

number of forged requests within the n-th sampling period is only related to the number of effective peers at time index n. This assumption could be approximately satisfied, if each newly infected peer has a large time window size. Since effective seeders (leechers) are a part of seeders (leechers), they are possible to leave the swarm and become ineffective immediately (complete their download and become ineffective seeders). The notations used in our model are listed in Table 4.1.

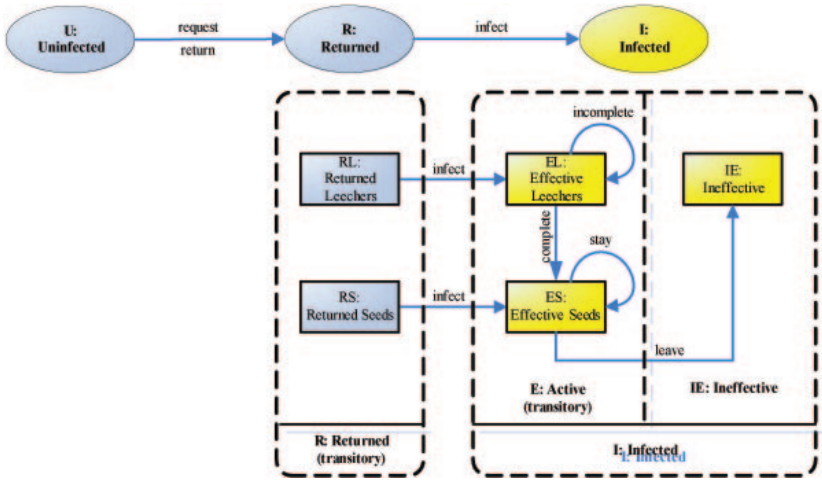

Fig. 4.5: The transition diagram of peers.

4.4.2 A hybrid A-BT worm model

The challenge of A-BT worm modeling lies in measuring the lifespan of infected peers in the highly dynamic BT system where peers join and leave the system frequently. Our model combines two models: 1) a fluid model, which is amenable to evaluate the number of peers that could be affected, and 2) an epidemic model, which is used to measure the number of peers that have been infected.

Table 4.1: Parameters used in the A-BT worm model

$L(t)$	number of leechers at time index t
$S(t)$	number of seeders at time index t
T	length of a sampling period
$J[n]$	number of leechers that join a swarm within the n-th sampling period
$B[n]$	number of leechers that become seeders within the n-th sampling period
$D[n]$	number of seeders that depart from a swarm within the n-th sampling period
$L[n]$	number of leechers in a swarm at time index n
$S[n]$	number of seeders in a swarm at time index n
$N[n]$	total number of peers in a swarm at time index n
$R_L[n]$	number of returned leechers within the n-th sampling period
$R_S[n]$	number of returned seeders within the n-th sampling period
$E_L[n]$	number of effective leechers at time index n
$E_S[n]$	number of effective seeders at time index n
$I[n]$	total number of infected peers at time index n
$\Delta L[n]$	changes of leechers in a swarm within the n-th sampling period
$\Delta S[n]$	changes of seeders in a swarm within the n-th sampling period
$\Delta E_L[n]$	changes of effective leechers within the n-th sampling period
$\Delta E_S[n]$	changes of effective seeders within the n-th sampling period
$\Delta I[n]$	increasing number of infected peers within the n-th sampling period
n_0	initial time index of worm attack
λ_0	initial value of peer arrival rate
τ	attenuation parameter of peer arrival rate
μ	uploading bandwidth
η	mean upload utilization of leechers
γ	rate at which seeders leave a swarm
α	probability that a peer to be vulnerable
ω	number of returned peers in a request
β_i	probability that the i-th infected peer generates a forged request within the n-th sampling period

Following the idea of the fluid model proposed in [Guo et al., 2005, Qiu and Srikant, 2004], we have:

$$J[n] = \int_{nT}^{(n+1)T} \lambda_0 e^{-\frac{t}{\tau}} dt \tag{4.3}$$

$$B[n] = \int_{nT}^{(n+1)T} \mu\big(\eta L(t) + S(t)\big) dt \tag{4.4}$$

$$D[n] = \int_{nT}^{(n+1)T} \gamma S(t) dt \tag{4.5}$$

$$\Delta L[n] = J[n] - B[n] \tag{4.6}$$

$$\Delta S[n] = B[n] - D[n] \tag{4.7}$$

By Equation 4.6 and 4.7, we can obtain the number of leechers $L[n]$ and the number seeders $S[n]$. We focus here on the physical meaning of each parameter. The detailed resolution of the fluid model can be find in [Guo et al., 2005]. The total number of peers at time index n is given by:

$$N[n] = L[n] + S[n] \tag{4.8}$$

We now derive how $E_L[n]$, $E_S[n]$ and $I[n]$ change over time index n. Below we compute the amounts, $\Delta E_L[n]$, $\Delta E_S[n]$ and $\Delta I[n]$, by which they change respectively over a small sampling period after time index n. This will give us a set of difference equations that together characterize the A-BT worm propagation.

- $\boldsymbol{\Delta} I[n]$: The number of uninfected peers in a swarm at time index n is $\alpha N[n] - E_L[n] - E_S[n]$. Since each peer in the swarm, no matter infected or not, has

an equal chance to be returned in a request, the probability that an uninfected peer is not returned to any infected peer within the n-th time period is $\left(1 - \frac{min\{N[n],\omega\}}{N[n]}\right)^{\sum_{i=1}^{E_L[n]+E_S[n]}\beta_i}$. Whenever an uninfected peer is returned, it will be infected immediately. By the epidemic model, $\Delta I[n] = \left(\alpha N[n] - E_L[n] - E_S[n]\right)\left(1 - \left(1 - \frac{min\{N[n],\omega\}}{N[n]}\right)^{\sum_{i=1}^{E_L[n]+E_S[n]}\beta_i}\right)$.

- $\mathbf{R_L}[\mathbf{n}]$: The fraction of returned uninfected leechers in all returned uninfected peers is proportional to that of uninfected leechers in all uninfected peers. Hence, $R_L[n] = \frac{\alpha L[n] - E_L[n]}{\alpha N[n] - E_L[n] - E_S[n]}\Delta I[n]$.

- $\mathbf{R_S}[\mathbf{n}]$: The ratio of returned uninfected seeders in all returned uninfected peers is $\frac{\alpha S[n] - E_S[n]}{\alpha N[n] - E_L[n] - E_S[n]}$. We have $R_S[n] = \frac{\alpha S[n] - E_S[n]}{\alpha N[n] - E_L[n] - E_S[n]}\Delta I[n]$.

- $\mathbf{\Delta E_L}[\mathbf{n}]$: Recall that all returned leechers directly become effective leechers. This increases the number of effective leechers by $R_L[n]$. On the other hand, some effective leechers may complete their download and become effective seeders instantly. There are totally $B[n]$ leechers who become seeders, and the fraction of effective leechers in all leechers is $\frac{E_L[n]+R_L[n]}{L[n]}$. Hence, the reducing number of effective leechers is $\frac{E_L[n]+R_L[n]}{L[n]}B[n]$. Combining these two numbers and representing the gross change, we have $\Delta E_L[n] = R_L(n) - \frac{E_L(n)+R_L(n)}{L(n)}B[n]$.

- $\mathbf{\Delta E_S}[\mathbf{n}]$: $R_S[n]$ returned seeders instantly become effective seeders. As discussed above, there are $\frac{E_L[n]+R_L[n]}{L[n]}B[n]$ effective leechers who become effective seeders. Whenever an effective seeder leaves a swarm, it becomes ineffective. Within the n-th sampling period, there are $D[n]$ seeders that leave the swarm, and the fraction of effective seeders is $\frac{E_S[n]+R_S[n]}{S[n]}$. Thus, there are $\frac{E_S[n]+R_S[n]}{S[n]}D[n]$ newly ineffective peers. Combining these three numbers to represent the gross change,

we have $\Delta E_S[n] = R_S[n] + \frac{E_L[n]+R_L[n]}{L[n]} B[n] - \frac{E_S[n]+R_S[n]}{S[n]} D[n]$.

Thus, we have the following equations:

$$\Delta I[n] = \big(\alpha N[n] - E_L[n] - E_S[n]\big)\left(1 - \big(1 - \frac{min\{N[n], \omega\}}{N[n]}\big)^{\sum_{i=1}^{E_L[n]+E_S[n]} \beta_i}\right) \tag{4.9}$$

$$R_L[n] = \frac{\alpha L[n] - E_L[n]}{\alpha N[n] - E_L[n] - E_S[n]} \Delta I[n] \tag{4.10}$$

$$R_S[n] = \frac{\alpha S[n] - E_S[n]}{\alpha N[n] - E_L[n] - E_S[n]} \Delta I[n] \tag{4.11}$$

$$\Delta E_L[n] = R_L[n] - \frac{E_L[n] + R_L[n]}{L[n]} B[n] \tag{4.12}$$

$$\Delta E_S[n] = R_S[n] + \frac{E_L[n] + R_L[n]}{L[n]} B[n] - \frac{E_S[n] + R_S[n]}{S[n]} D[n] \tag{4.13}$$

We should note that, in Equation 4.9, the parameter β_i is essential to the worm speed, which has been discussed in section 4.3.2. Finally, we add the incremental variables and get $I[n+1] = I[n] + \Delta I[n]$, $E_L[n+1] = E_L[n] + \Delta E_L[n]$. The boundary conditions for the set of equations above are: $L[0] = E_S[n_0] = 0$, $S[0] = N[0] = E_L[n_0] = I[n_0] = 1$.

4.5 A-BT worm detection and containment

Most existing defense methods may fail to detect and contain the A-BT worm for the following reasons. The self-defense infrastructure methods [Douglas et al., 2008, Freitas et al., 2007, Zhou et al., 2005, 2006] cannot slow down the worm because the worm is not affected by the P2P topology. The feedback control approach [Wu and Feng, 2006] is unable to detect the worm since the worm can adaptively adjust its

speed. Thus, there is clearly a need for finding a new way to defend against the A-BT worm. In this section, we employ a statistical method to automatically detect the worm from the tracker by investigating the variance of time intervals of requests.

The variance analysis, including the sample mean analysis, can be used to measure the spread of samples, which has been widely used in many areas, such as seismic detection and investment management in business. To estimate the request generation rate of a peer, which may be greatly affected by the worm, we sample time interval between two consecutive requests of the peer. As shown in Figure 4.6, the samples represented as a chronological sequence $\{X[n]\}$ can be viewed as discrete-time signals. In normal cases, the amplitude of signals is stable, because regular requests are generated at a regular time interval (20 minutes), and the "background noise" (e.g., network delay) has little effect on the signal to noise ratio (SNR). However, when a peer is compromised by the worm, the amplitude of the signal varies a lot, because the forged requests may divide regular time intervals into some irregular short time intervals. By measuring the similarity of the signal on multiple peers, we can estimate the damage caused by the worm.

Assume that we have obtained M signals, and the signal length is N. The sample mean of M signals is defined by:

$$\overline{X}[n] = \frac{1}{M} \sum_{m=0}^{M-1} X_m[n] \tag{4.14}$$

We use the sample mean $\overline{X}[n]$ to denote a "reference" signal. The sample variance is given by:

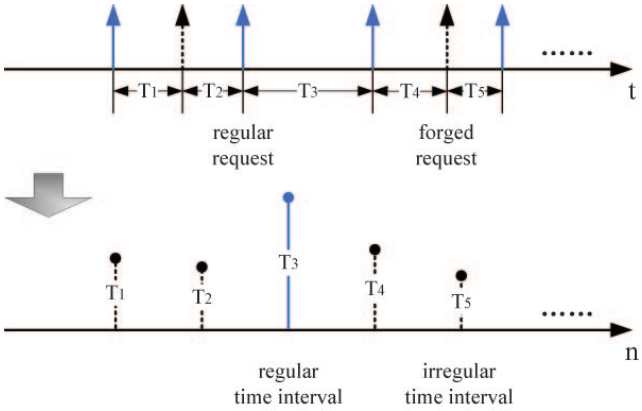

Fig. 4.6: An example of the sample sequence.

$$
\begin{aligned}
Var(X[n]) &= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(X_m[n]-\overline{X}[n]\right)^2 \\
&= \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}X_m[n] - M\sum_{n=0}^{N-1}\overline{X}^2[n]
\end{aligned}
\tag{4.15}
$$

We transform the sample variance $Var(X[n])$ into the range of [0, 1]. Then, the normalized sample variance $Var'(X[n])$ can be written as:

$$
\begin{aligned}
Var'(X[n]) &= \frac{Var(X[n])}{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}X_m^2[n]} \\
&= 1 - \frac{M\sum_{n=0}^{N-1}\overline{X}^2[n]}{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}X_m^2[n]}
\end{aligned}
\tag{4.16}
$$

When no attack occurs, $Var'(X[n])$ will be close to 0. When the worm is widespread, $Var'(X[n])$ will become bigger. Thus, we can set a threshold to detect the worm. We

should point out that, in real cases, peers will request the tracker frequently, if they do not have sufficient neighbors. The threshold setting should be based on some statistical results. Some advanced techniques can also be applied for the worm detection, such as data mining, Fourier analysis, coherence analysis and wavelet analysis. With these techniques, we can identify not only worm activities but also infected peers.

For the worm containment, the tracker can collect some useful information on peers, such as the number of OS vulnerabilities and the version of the antivirus software installed. This information could be easily obtained, because most antivirus softwares provide the function of the vulnerability scanning. The tracker employs a safe strategy to returns a biased peer set, rather than a random peer set, in response to a request. The biased peer set, where peers have less OS vulnerabilities or have an up-to-date antivirus software, can reduce the probability that an uninfected peer is returned in a request. We believe that the safe strategy will not seriously affect the fairness of uploading/downloading strategy in the BT system, because it is used only when a worm attack is detected.

4.6 Simulation results

In this section, we firstly describe the environment and settings of our simulations. We then setup several simulations to verify our hybrid model and detection method.

4.6.1 Simulation environment and settings

The study of the worm propagation in a real-world BT system may cause some legal issues. We deploy our simulations on a powerful simulation platform developed

by University of Electronic Science and Technology of China. This simulation platform has three components: 1) *tracker*: on one host, we setup a real-world tracker by using MyBT server 1.0, which is a software to setup a BT server. 2) *initial seeder*: on the same host, we install BitComet 0.93, which is a well-known BT download software, and use the software to create a .torrent file. 3) *simulated peers*: on another host, we install PeerSim [Jesi and Patarin, 2005], which is a java-based discrete-event engine that can simulate hundreds of peers and observe their activities (e.g., arrivals, departures, pieces exchange), and implement the basic BT protocol (e.g., unchoking, rarest-first). Each simulated peer can communicate with the tracker and share real pieces with both initial seed and other simulated peers.

For simplicity, we do not consider free-ridering [Barbera et al., 2005] in our simulations. In other words, all peers are willing to contribute their upload bandwidth. We also do not strictly follow unchoking specified in the BT protocol. A peer simply chokes those neighbors who do not upload any data within a time interval. We believe that this change may have a very little effect on the results.

Since the piece transmission is resource consuming, we limit the maximum scale of our simulation to 200 peers. We define the sampling period (1 cycle in PeerSim) as the unit of time. Unless otherwise specified, the default settings in our simulations are shown in Table 4.1.

In our simulations, we evaluate the changes in number of leechers $L[n]$, number of seeds $S[n]$, number of effective leechers $E_L[n]$, number of effective seeders $E_S[n]$, and number of infected peers $I[n]$.

Table 4.2: Simulation settings of the A-BT worm

File size	15.6MB (500 pieces)
Number of initial seeds $S[0]$	1
Normalized maximum upload bandwidth μ	0.01 (5 pieces)
Normalized maximum download bandwidth	0.03 (15 pieces)
Number of neighbors of each peer	10
Initial value of peer arrival rate λ_0	0.6
Attenuation parameter of peer arrival rate τ	300
Initial time index of worm attack n_0	150
Rate at which seeds leave a swarm γ	0.08
Probability that a peer to be vulnerable α	0.8
Number of returned peers in a request ω	10
Mean upload utilization of leechers η	1
Time interval between two normal requests	100
Time window size W	50

4.6.2 Verification of the hybrid model

We compare the results of $E_L[n]$, $E_S[n]$ and $I[n]$ from Equations with that from simulations. Figure 4.7 shows that there exists a small gap between numerical results and simulation results. The peak of effective seeds in the simulation result is around 60, while that in the numerical result is less than 40. We guess that the gap is mainly caused by the uncertainty in the measurement of the complicated peer behavior in the BT system. We further measure the mean upload bandwidth utilization of leechers and seeders. From Figure 4.8, we can see that the upload bandwidth of seeds is fully utilized at the beginning, because the initial seed is called upon to do much of the serving when the swarm size is small. However, when more peers join the system, the upload bandwidth utilization of seeders drops dramatically from 1.0 to 0.27. On the other hand, the upload bandwidth utilization of leechers increases and arrives at its first peak 0.56 at time index 129, which indicates that P2P serving becomes very effective. We input the simulation results of $L[n]$ and $S[n]$ into the hybrid model, and

recompute $E_L[n]$, $E_S[n]$ and $I[n]$. Figure 4.9 shows that the numerical results match with the simulation results very well.

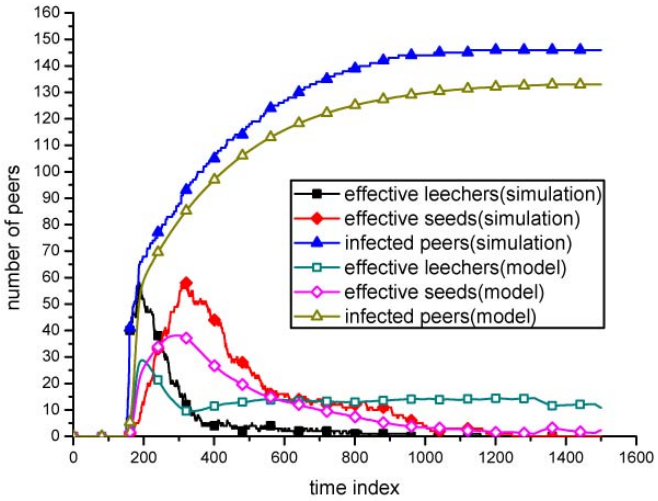

Fig. 4.7: Model validation using parameters obtained from the fluid model.

4.6.3 Evaluation of the worm damage

We compare the A-BT worm with the topological P2P worm. From Figure 4.10, we can see that both topological P2P worm and A-BT worm can infect around 140 peers in a swarm. This result indicates that the A-BT worm is as efficient as the topological P2P worm. Another interesting observation is that the increasing rate of effective seeders in topological P2P worm is slower than that in A-BT worm. One reason is that, before infecting a certain peer, the topological P2P worm needs to infect at least one neighbors of that peer. The A-BT worm, however, can infect any peer in the swarm with an equal probability. Another reason is that a few high-degree peers play a significant role in speeding up the topological P2P worm. If these peer

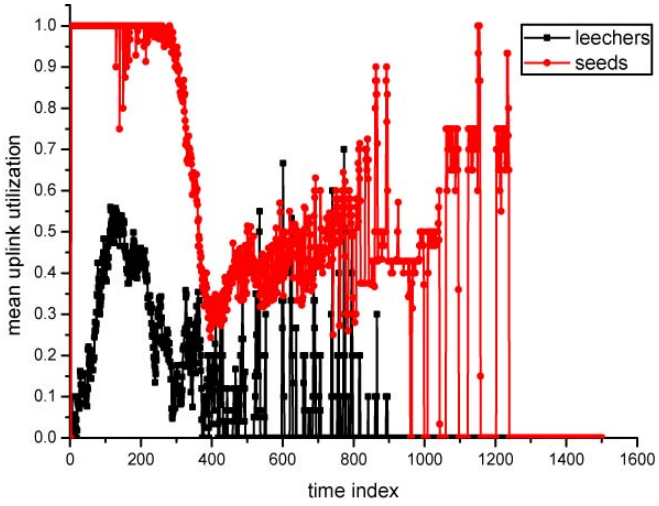

Fig. 4.8: Mean upload bandwidth utilization of peers.

are patched or well protected, the topological P2P worm cannot spread very fast. The A-BT worm, on the other hand, appears to be less affected by the P2P topology.

We also compare the A-BT worm with the R-BT worm. Figure 4.11 shows the damage caused by the two worms. For the R-BT worm, we see that the number of infected peers drops dramatically from 144 to 72, when the time window size W increases from 5 to 500. One possible reason is that most infected peers who set a large time window leave the swarm before they duplicate the worm to others. Another reason might be that the swarm size shrinks when time goes by. The later the worm attack is lunched, the less the vulnerable peers are infected. For the A-BT worm, we set the initial time window size to 5, the threshold rate of being accessed A_{th} to 0.4, and the increasing ratio of the time window size φ to 10. Recall that the time window size will increase exponentially if A_{pre} is larger than A_{th}. We then observe that the A-BT worm can be as efficient as the R-BT worm. Figure 4.12 shows the abnormal behavior of the two worms. We see that the forged requests generated by the A-BT

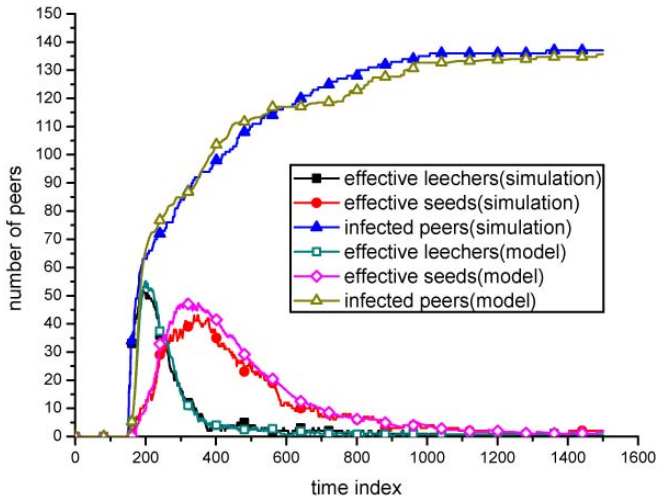

Fig. 4.9: Model validation using parameters obtained from simulations.

worm are far less than that generated by the R-BT worm.

4.6.4 Effectiveness of the detection method

We firstly study the impact of worm parameters on the detection method. We set the time window size W to be one of $\{50, 100, 150, 200, 250\}$, the leaving rate γ to 0.002. For the A-BT worm, we set the increasing ratio of the window size φ to 5. We randomly select 20 peers from the swarm, and analyze the variance in the time intervals of requests. Figure 4.13 shows that $Var'(X[n])$ for R-BT worm drops from 0.273 to 0.201 when W increases from 50 to 100, and then goes up from 0.201 to 0.253. $Var'(X[n])$ for A-BT worm decreases from 0.503 to 0.274 when W increases from 50 to 100, and then increases to 0.359. When W increases, the forged request generation rate decreases. If W becomes too large, it may result in large variations in the time interval. That might be the reason why $Var'(X[n])$ of A-BT worm drops

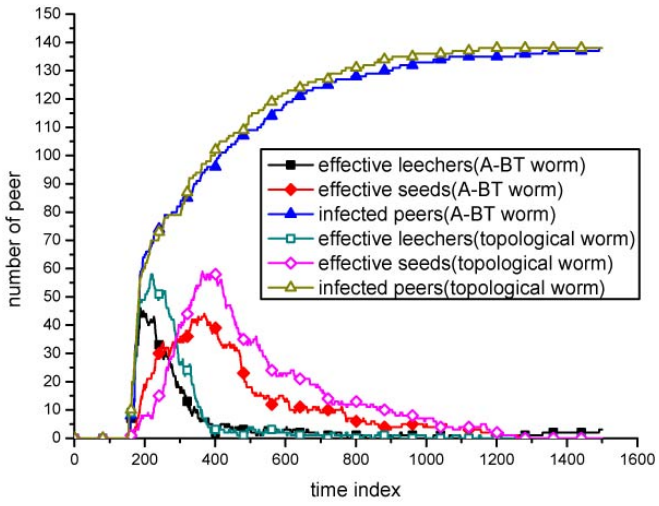

Fig. 4.10: Comparison between topological P2P worm and A-BT worm.

first and then goes up slightly. Even though the A-BT worm is "smart" enough to request the tracker at a very low rate, we can detect it from the tracker by setting an appropriate threshold for $Var'(X[n])$.

We then show the impact of system parameters on the detection method. We select the vulnerability density α from the set $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. Other parameters are the same as that in the previous simulation. From Figure 4.14, we see that $Var'(X[n])$ increases when the vulnerability density increases. We can also see that $Var'(X[n])$ for A-BT worm is much higher that for R-BT worm, when α is large. These results show that we can estimate the vulnerability density of the swarm by observing $Var'(X[n])$. We can take the safe strategy only when a large fraction peers are vulnerable for the worm attack.

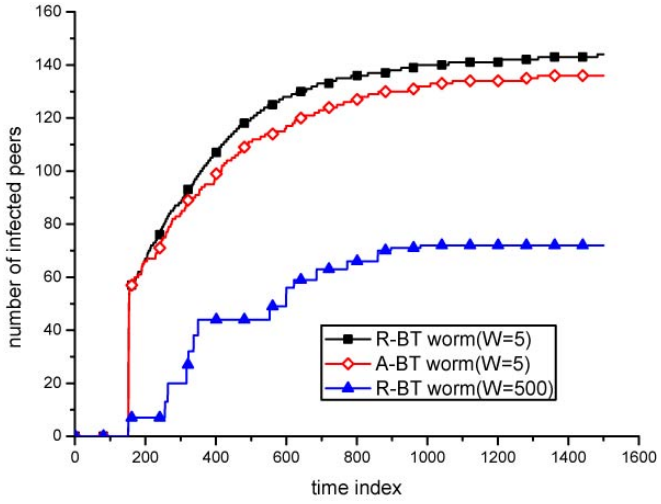

Fig. 4.11: Comparison between R-BT worm and A-BT worm.

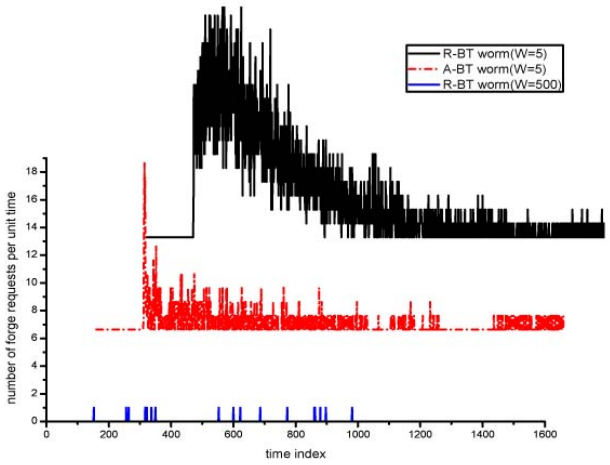

Fig. 4.12: Number of forged requests generated by worms.

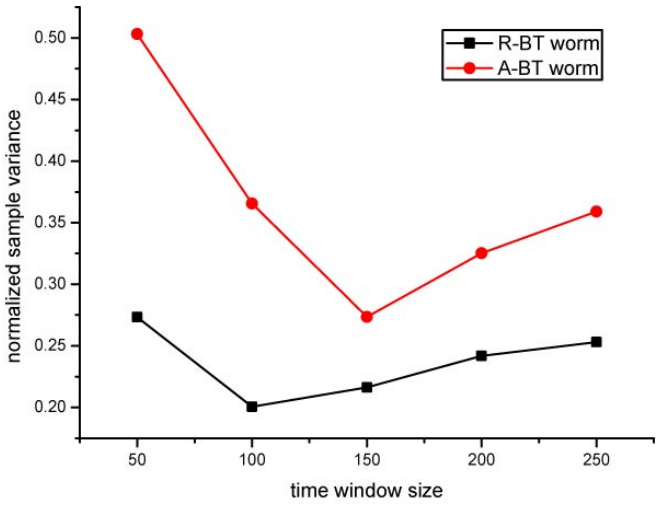

Fig. 4.13: Effect of time window size on worm detection.

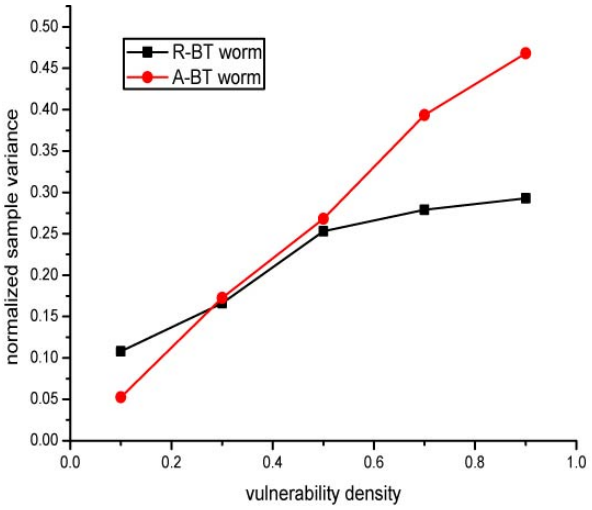

Fig. 4.14: Effect of vulnerability density on worm detection.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we explore several issues in BT protocol and system including modeling, algorithms and worms, in order to understand, improve and secure protocol and system design.

We look at the BT system from a new angle, and build a bottom-up probabilistic model, which analyzes the system from the individual peer behavior to the overall system performance. Our model increases the accuracy of performance analysis, because it captures many features of the system, including heterogeneity and dynamics, and requires far less restrictive assumptions than ever before. Through the model, we have some interesting observations. Based on these observations, we suggest that the tracker decides who and how many peers to return according to some network parameters, such as, swarm size, departure rate and download rate. The benefit of doing so is two folds. First, it can prevent the system from the large view exploit. If the number of returned peers is controllable, free-riders are unable to gain extra

benefits from the unusually large neighbor set. Second, it is helpful to rationally utilize the bandwidth of high-bandwidth peers. The tracker would better not return too many high quality peers to high-bandwidth ones, because they may leave earlier if they download faster, which results in an unsatisfactory overall performance. We also give some expiations for why unchoking fails to motivate most users to upload more, and suggest reconsidering BT's incentives from the management's or psychological perspective.

We dispel some myths about three piece-related algorithms, and propose some remedies for them as well. We propose a credit method to evaluate peer honesty by examining *have* messages which are already been included in the BT protocol. The method is fully distributed, which requires no centralized sever to operate. We propose a utility-driven strategy to maximize interest relying on two utility functions. Such a strategy is unified, win-win, feasible, deployable. In particular, it covers both rarest-first and random; all peers can benefit from it by serving others more and being served more; it requires neither a modification to the wire protocol nor a full deployment to benefit. We revise the scatter algorithm to manage the size of request queue by dividing queued pieces into two groups: *rational* and *sub-rational* and keeping a ratio between them. The revised algorithm is dynamic, real-time and greedy. More specifically, the number of queued pieces is dynamically controlled by a ratio; it decides when to add a new piece based on a controllable ratio that meets the real-time requirement of piece selection; it fully utilizes every idle neighbor connection. Our primary experiment results show that they can outperform existing ones, and achieve both individual good and social good.

We present a novel P2P worm, called *Adaptive BitTorrent worm* (A-BT worm), in

BT-like systems, which propagates itself by requesting a tracker for vulnerable peers. We believe that such a worm could pose a vital threat to the P2P security for the following reasons: 1) it has the ability to locate most recent active peers; 2) it is easy to be implemented; 3) it can adaptively adjust its speed to reduce its abnormal behavior. By combining the fluid model with the epidemic model, we build a hybrid model to measure the worm damage. We also discuss the possible strategies to detect and contain the worm.

5.2 Future work

A number of areas for future work have been pointed out along the way. These include the following:

A decentralized P2P bootstrapping: bootstrapping is a process in which a new peer who intends to join a P2P system tries to discover contact information of other peers that have already been in the system. Existing P2P systems [BitTorrent, 2011, Lilja Fjeldsted, 2005, T. Klingberg, 2002]use either centralized servers or static peers for bootstrapping. Both of them have single point failure problem. In Dec. 2009, the Chinese government cracked down on the BT system by shutting down more than 530 BT web severs [SARFT, 2009]. Without the help of these severs, new peers cannot initialize their neighbors, and thus, fail to join the system. Some resent work suggests using random address probing [Dinger and Waldhorst, 2009, GauthierDickey and Grothoff, 2008] and existing P2P systems [David I. Wolinsky and Figueiredo, 2010] for bootstrapping. However, these methods are impractical and small-scale. We try to exploit social networks to design a fully decentralized bootstrapping in which new peers grabs instant messaging (QQ or MSN) packets to

obtain pre-existing peer's IPs. Such a bootstrapping can become an essential part of the pure P2P system, where peers share their resources freely without the help of any centralized server.

A user-oriented P2P incentive: the role of incentives is to motivate users to contribute their resources to others so as to achieve fast download times for all peers. Most existing P2P incentives are peer-oriented, which manage the neighbors of a peer. These incentives may fail to motivate most users to contribute their upload bandwidth, because they are short in providing some useful information for users. We attempt to design a user-oriented P2P incentive, which manages the behaviors of a user. The incentive design contains two steps. *Qualitative analysis* aims to understand the factors that may affect users' decisions, which requires some knowledge of psychology. We will design a questionnaire, which consists of a series of questions and responses, and distribute it a certain number of users. *Quantitative analysis* is used to quantify the factors that users care. The user-oriented incentive should have the ability to estimate the potential outcome of users, and reward or punish users based on their decisions.

Bibliography

Barbera, M., Lombardo, A., Schembra, G., and Tribastone, M. (2005). A markov model of a freerider in a bittorrent p2p network. In *GLOBECOM '05: Proceedings of the IEEE Global Telecommunications Conference*, volume 2, pages 985–989, St. Louis, MO, USA.

BitComet (2009). http://www.bitcomet.com/doc/download-zh.htm.

BitTorrent (2011). http://www.bittorrent.com/.

Chow, A. L. H., Golubchik, L., and Misra, V. (2009). Bittorrent: An extensible heterogeneous model. In *INFOCOM'09: Proceedings of the 28th Conference of the IEEE Computer and Communications Societies*, pages 585–593, San Diego, CA, USA.

Cohen, B. (2008). The bittorrent protocol specification. Technical report, BitTorrent.org.

David I. Wolinsky, Pierre St. Juste, P. O. B. and Figueiredo, R. J. O. (2010). Addressing the p2p bootstrap problem for small overlay networks. In *P2P'2010: Proceedings of the Peer-to-peer Computing*, pages 1–10.

Dinger, J. and Waldhorst, O. (2009). Decentralized bootstrapping of p2p systems: A practical view. In *NETWORKING '09: Proceedings of the 8th International IFIP-TC 6 Networking Conference*.

Douglas, M., Micael, P., and Evangelos, K. (2008). di-jest: Autonomic neighbour management for worm resilience in p2p systems. In *WoWMoM'08: Proceedings of the International Symposium World of Wireless, Mobile and Multimedia Networks*, pages 1–6.

Ernesto (2009). Bittorrent still king of p2p traffic.

Fan, B., Chiu, D.-m., and Lui, J. (2006). The delicate tradeoffs in bittorrent-like file sharing protocol design. In *ICNP '06: Proceedings of the 14th IEEE International Conference on Network Protocols*, pages 239–248, Washington, DC, USA. IEEE Computer Society.

Feldman, M. and Chuang, J. (2005). Overcoming free-riding behavior in peer-to-peer systems. *ACM Sigecom Exchanges*, 6:2005.

Fogel, D. B. (1993). Applying evolutionary programming to selected traveling salesman problems. *Cybernetics and Systems*, 24(1):27–36.

Freitas, F., Rodrigues, R., Ribeiro, C., Ferreira, P., and Rodrigues, L. (2007). Verme: Worm containment in peer-to-peer overlays. In *IPTPS'07: Proceeding of the 6th International Workshop on Peer-to-Peer Systems*.

Fudenberg and Tirole (1992). *Game Theory*. MIT Press.

GauthierDickey, C. and Grothoff, C. (2008). Bootstrapping of peer-to-peer networks. In *SAINT*, pages 205–208. IEEE Computer Society.

Guo, L., Chen, S., Xiao, Z., Tan, E., Ding, X., and Zhang, X. (2005). Measurements, analysis, and modeling of bittorrent-like systems. In *IMC '05: Proceeding of the Internet Measurement Conference*, pages 35–48.

Isdal, T., Piatek, M., Krishnamurthy, A., and Anderson, T. (2007). Leveraging bittorrent for end host measurements. In *PAM'07: Proceedings of the 8th international conference on Passive and active network measurement*, pages 32–41, Berlin, Heidelberg. Springer-Verlag.

Jesi, G. P. and Patarin, S. (2005). *PeerSim HOWTO: Build a new protocol for the PeerSim 1.0 simulator*.

Legout, A., Liogkas, N., Kohler, E., and Zhang, L. (2007). Clustering and sharing incentives in bittorrent systems. In *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 301–312, New York, NY, USA. ACM.

Legout, A., Urvoy-Keller, G., and Michiardi, P. (2006). Rarest first and choke algorithms are enough. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 203–216, New York, NY, USA.

Levin, D., Lacurts, K., Spring, N., and Bhattacharjee, B. (2008). Bittorrent is an auction: analyzing and improving bittorrent's incentives. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 243–254, New York, NY, USA.

Liao, W.-C., Papadopoulos, F., and Psounis, K. (2007). Performance analysis of

bittorrent-like systems with heterogeneous users. volume 64, pages 876–891, Amsterdam, The Netherlands, The Netherlands. Elsevier Science Publishers B. V.

Lilja Fjeldsted, Jonas Fonseca, B. R. (2005). Specification and implementation of the bittorrent protocol. Technical report.

Locher, T., Moor, P., Schmid, S., and Wattenhofer, R. (2006). Free riding in bittorrent is cheap. In *HotNets-V: Proceedings of the 5th Workshop on Hot Topics in Networks*, Irvine, California, USA.

Luo, J. (2010). The source code of the java-based bittorrent client.

Luo, J., Xiao, B., Liu, G., Xiao, Q., and Zhou, S. (2009). Modeling and analysis of self-stopping bt worms using dynamic hit list in p2p networks. In *SSN'09: Proceedings of The 5th International Workshop on Security in Systems and Networks*.

Ma, J., Chen, X., and Xiang, G. (2006). Modeling passive worm propagation in peer-to-peer system. In *CIS'06: Proceedings of the International Conference on Computational Intelligence and Security*, pages 1129–1132.

Merton, R. C. (1990). *Continuous-time Finance*. Blackwell.

Pouwelse, J. A., Garbacki, P., Epema, D. H. J., and Sips, H. J. (2005). The bittorrent p2p file-sharing system: Measurements and analysis. In *IPTPS'05: Proceedings of the 4th International Workshop on Peer-to-Peer Systems*.

Qiu, D. and Srikant, R. (2004). Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *SIGCOMM'04: Proceedings of the ACM SIGCOMM 2004 conference on Data communication*, pages 367–378.

Rai, V., Sivasubramanian, S., Bhulai, S., Garbacki, P., and van Steen, M. (2007). A multiphased approach for modeling and analysis of the bittorrent protocol. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 10, Washington, DC, USA.

Ramachandran, K. and Sikdar, B. (2006). Modeling malware propagation in gnutella type peer-to-peer networks. In *IPDPS'06: Proceedings of Parallel and Distributed Processing Symposium*.

SARFT (2009). http://www.sarft.gov.cn/.

Singer, M. (2002). "benjamin" worm plagues kazaa. Technical report, siliconvalley.internet.com.

Sirivianos, M., Han, J., Rex, P., and Yang, C. X. (2007). Free-riding in bittorrent networks with the large view exploit. In *IPTPS '07: Proceedings of the 6th International workshop on Peer-To-Peer Systems*.

Stutzbach, D. and Rejaie, R. (2006). Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202.

T. Klingberg, R. M. (2002). Gnutella protocol development. Technical report.

TheoryOrg (2004). Bittorrent protocol specification v 1.0. Technical report, TheoryOrg.

Thommes, R. and Coates, M. (2006). Epidemiological modeling of peer-to-peer viruses and pollution. In *INFOCOM'06*.

Vamosi, R. (2001). Gnutella worm: How to deal with it.

Vroom, V. H. and MacCrimmon, K. R. (1968). Toward a stochastic model of managerial careers. *Administrative Science Quarterly*, 13(1):26–46.

Wang, H., Liu, J., and Xu, K. (2009). On the locality of bittorrent-based video file swarming. In *IPTPS'09: Proceedings of the 8th international conference on Peer-to-peer systems*, pages 12–12. USENIX Association.

Wu, K. and Feng, Y. (2006). Proactive worm prevention based on p2p networks. In *IJCSNS'06: Proceedings of the International Journal of Computer Science and Network Security*.

Yang, X. and de Veciana, G. (2004). Service capacity in peer-to-peer networks. In *INFOCOM'04: Proceedings of the 23th Conference of the IEEE Computer and Communications Societies*, pages 2242– 2252.

Yao, Y., Luo, X., Gao, F., and Ai, S. (2006). Research of a potential worm propagation model based on pure p2p principle. In *ICCT '06: Proceedings of the International Conference on Communication Technology*, pages 1–4.

Yao, Y., Wu, L., Gao, F., Yang, W., and Yu, G. (2008). A waw model of p2p-based anti-worm. In *ICNSC'08: Proceedings of the IEEE International Conference on Networking, Sensing and Control*, pages 1131 – 1136.

Zhou, L., Zhang, L., Mcsherry, F., Immorlica, N., Costa, M., and Chien, S. (2005). A first look at peer-to-peer worms: Threats and defenses. In *IPTPS'05: Proceeding of the 4th International Workshop on Peer-to-Peer Systems*.

Zhou, Y., Wu, Z., Wang, H., Zhong, J., Feng, Y., and Zhu, Z. (2006). Breaking monocultures in p2p networks for worm prevention. In *ICMLC'06: Proceedings of the 5th International Conference on Machine Learning and Cybernetics*.