MINIMIZING SERVICE DISRUPTION IN PEER-TO-PEER STREAMING

By

Yanchuan Cao

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

May, 2011

Nashville, Tennessee

Approved:

Professor Yi Cui

Professor Gautam Biswas

Professor Aniruddha Gokhale

Professor William Robinson

Professor Yuan Xue

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Peer-to-peer Streaming

When millions of people are cheering for the Super Bowl touchdowns, few may wonder what makes the undisrupted video streaming possible. From the surface of Mars to the bottom of Mexico gulf, live event broadcasting has changed the way people interact with the world. 3D Movies nowadays only one-touch-away from your smartphones, tablets or even live in the theater redefined the realm of video streaming. Behind the scenes of these multimedia streaming applications lies a common infrastructure, proven to be effective and scalable : the peer-to-peer (P2P) system.

In P2P systems, peers self-organize themselves into an overlay network and relay data to each other, thus reducing server load. A key feature that distinguishes the performance of one P2P solution to another is *peer selection*, the strategy a peer employs to select other peer(s) as its parent(s) from which to receive data. Peer selection algorithms aggregate peers into multicast tree(s) spanning from the server, the source of the data, to all peers. Given the data-intensive nature of P2P applications (e.g., video streaming or bulk data distribution), a common objective of peer selection optimization is to maximize the data throughput to all peers.

Already challenging in its static setup, the optimal peer selection problem is further aggravated by the high volatility of the P2P network. Due to various reasons such as user leaving or machine/network failure, unscheduled peer departure constantly happens, which results in service disruptions or outages on all the downstream peers. Therefore, we argue that when designing peer selection solutions, fault resilience deserves the same level of attention as first-class performance metrics, e.g. throughput,

delay, etc.

Contributions in This Dissertation

A significant amount of research has been conducted on this topic with different emphasis. While important heuristics have been proposed such as bandwidth first, age-first, or a hybrid of the two, some analytical works have tried to analyze and compare their performances under stochastic framework or real-system traces. However, this domain has been rarely examined from the optimization perspective. If we are able to model the fault-resilient peer selection problem under an optimization framework which combines fault resilience with key performance metrics such as throughput, then standard optimization techniques can be practiced to evaluate key questions such as the solvability of the problem and the complexity of its optimal solutions, if any. Also existing approaches could be quantitatively evaluated under the same framework.

A Optimization Framework to Maximize Resilient Throughput

In this dissertation, we report our research towards this direction. Our optimization framework is based on the *generalized flow* theory. It generalizes the classical network flow problem by specifying a gain factor to each link in the network. As such, the objective is to optimize the throughput of the generalized flow as the product of raw flow and the gain factor on each link, while the traditional capacity and flow conservation constraints still apply to the raw flow. Widely employed in operation research to model the loss, theft, or interest rate in commodity transportation[1], we find it a good match to the P2P domain. If we assign each peer a resilience factor as the probabilistic measure of its chance of survival within a given time horizon, this resilience factor could be considered as the gain factor in the generalized flow

setting. Under this framework, the problem of fault-resilient peer selection becomes to maximize the aggregation of generalized flow received by each peer, which is the product of the raw flow and resilience factors of peers it passes along.

We study this problem under a multitude of problem settings. Specifically,

- Regarding network model, we consider two types of topologies: the *general topology* which models the underlying physical network as a graph, and the *star topology* which assumes the bottleneck does not exist in the physical network, but on peer's access link.

- Regarding overlay organization, we consider cases where the number of trees interconnecting peers is *unlimited* versus *upper-bounded*, e.g., single tree.

- Along the dimension of generalized flow definition, we consider the *concatenation model* where the generalized flow delivered to a peer depends on the resilience of all its ancestors, and the *non-concatenation model* which only considers the resilience of its immediate parent.

Along these dimensions, we explore the entire spectrum of this domain, and focus on studying problem complexity and finding the optimal solutions within each subproblem.

A Peer Evaluation Metric to Minimize Service Disruption

After exploring the scalability and complexity of our optimization framework, we went along this direction to find a practical solution to minimize service disruption under this framework.

Any P2P streaming solution must have enough bandwidth and stability to support various content-rich applications and satisfy ever-questing users. Bandwidth capacity is needed to support high-quality streaming, while stability to minimize the service disruption caused by premature peer-leaving. Achieving one goal is easy. There are

bandwidth-first policy to maximize achievable throughput[9] and age-first policy to minimize occurrence of disruptions[5]. Reconciling both goals is hard. People may use different terminals, from hand-held tablets to Internet TV. The various P2P accesses render the uplink bandwidth heterogeneous and peer churning inherent, hence making the problem more challenging.

To solve this problem, we designed an *evaluation metric*. The metric measures the quality and readiness of a peer to serve other peers. Existing approaches include building core P2P infrastructure out of long-lived peers[31, 30] and Hybrid peer selection policy[5]. Like existing approaches, our solution considers bandwidth and lifetime. Distinguished from existing approaches, we designed our solution to feature easy measurement and limited overhead. To achieve this goal, we have three design criteria: (1) *Property Coverage*, the metric should cover the two fundamental properties of a peer: resource abundance and prospective longevity; (2) *Simple Representation*, preferably a single scalar, against which, a group of peers can be compared and sorted; (3) *Effective Communication*, it should be easily measured and communicated in a distributed fashion with limited overhead. Criteria (1) guaranteed our method tackles the same bandwidth-lifetime optimization issue as other approaches tried to do. Criteria (2) and (3) ensure the practical implementation of our solution.

Guided by the three design criteria, we propose the evaluation metric *PRW* (peer resilient weight). The intuition of our design is straightforward. In order to have less overhead, the metric should measure peer not tree. To cover the fundamental properties of a peer, the metric should involve available bandwidth and prospective lifetime. For a peer, longer prospective life means it is more resilient, and smaller bandwidth usage means more available uplink bandwidth. Therefore we have a prototype of the metric:

$$w = \frac{bandwidth\_usage}{prospective\_lifetime}$$

The question then becomes how to quantify the prospective lifetime, and formulate a framework that incorporates both factors. In this dissertation, we measure a peer's prospective lifetime with the resilience factor, and derive the evaluation metric from the generalized flow framework. Classic network flow maximizes flow under link capacity constraints, while generalized flow optimizes the product of flow and gain, subject to the constraints. We formulate our optimization framework to maximize the aggregated generalized flow received by all peers, subject to the uplink capacity constraints.

We tested the performance of *PRW* with two sets of real world traces in three P2P overlay structures. The traces are PPLive[15] and MSN Video[16]. The P2P overlay structures are *tree*, *multi-tree*, and *mesh*. We use two performance metrics, *service disruption* and *rejection*. Service disruption counts the number of disruptions a peer experiences due to ancestor(s) leaving. Rejection measures number of peers rejected due to insufficient uplink bandwidth. We compared the performance of our solution with existing algorithms: *age-first*, *bandwidth-first*, *hybrid* and *ROST*(Reliability-Oriented Switching Tree)[26]. The experiment results prove that PRW achieves both lower service disruption and peer rejection.

Dissertation Organization

The remaining of this dissertation is organized as follows. We discuss related works in Chapter II (Related Work). In Chapter III (Framework Overview), we introduce our optimization framework and formally define key concepts such as generalized flow and resilience index. In Chapter IV (General Topology Model), we study the optimal peer selection problem under the general topology, and propose two algorithms employing linear programming techniques. In Chapter V (Star Topology Model), we study the same problem under the star topology, and propose two algorithms based

on combinatorial optimization techniques. Chapter VI (Performance Evaluations - Optimization Framework) presents evaluation results. In Chapter VII (Peer Evaluation Metric Overview), we present the optimization framework of the generalized flow problem, from which *PRW* is derived. In Chapter VIII (Peer Evaluation Metric), we discuss the computing, maintenance, and distribution of *PRW*, and related practical issues. In Chapter IX (Performance Evaluation - Peer Evaluation Metric), we use simulation to compare our algorithm with other algorithms and heuristics. We conclude our findings in Chapter X (Conclusion) and give future research suggestions in Chapter XI (Future Work).

CHAPTER II

RELATED WORK

Our research falls into the category of Peer-to-peer streaming[19] of Overlay Multicast Networks. We focus on the *Peer Selection* algorithms of P2P fault resilience research, than the other two major aspects of this domain: *Depth Minimizing* and *Parent-Child Switching* (See Fig.II.1). Our approach tries to build an optimal parent selection with a peer evaluation metric based on the generalized flow optimization framework. In this section, we will introduce related research domains and backgrounds of this dissertation.

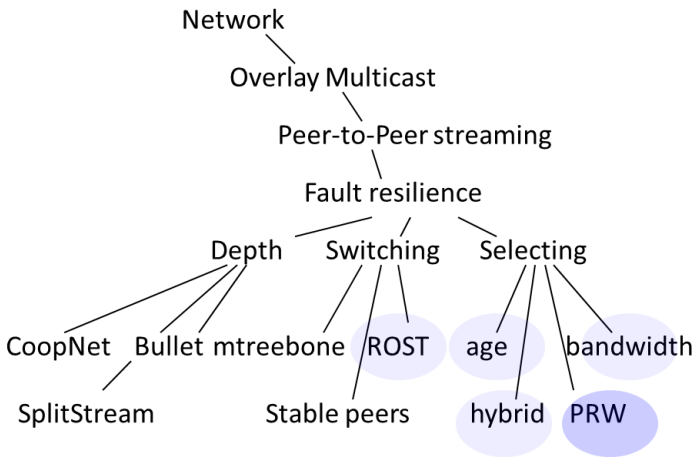


Figure II.1: Research Category of This Dissertation

P2P Overlay Structures

Several evaluative works have been conducted to study the impact of different overlay construction algorithms on the resilience of P2P network. Bishop et al. examines the effect of bandwidth- and age-priority heuristics on multicast tree reliability

using trace-based simulation[5]. Stochastic network analysis have been employed to study the resilience of DHT-based (Distributed Hash Table) P2P network[26] and decentralized P2P network[18] under given peer lifetime distribution. In contrast, our work does not make a priori assumptions on peer characteristics, but focuses on finding optimal peer selection algorithms that can take any input.

Numerous P2P streaming solutions have been proposed, where some representative works encompass the categories of single-tree[7], multi-tree[22, 6], mesh[17, 20], or hybrid solutions[29, 31]. Each solution is designed in its own way to maintain the P2P structure and handle peer churning. Although the algorithm presented here deals with peer dynamics by attaching new peers to the existing P2P structure incrementally, the central of focus of our work is the evaluation metric *PRW*, which can be of reference value to other P2P streaming solutions too.

P2P Fault Resilience

Depth Minimizing

Fault resilience has been considered in many existing solutions in overlay and P2P networks. An important approach is to reduce the tree depth to minimize failure propagation and service delay. Algorithms bearing the flavor of "minimizing depth" have been proposed in [14, 22]. Several well-known works, such as Bullet[17], SplitStream[6], and CoopNet[23], also employs the multi-tree approach to reduce the impact of peer failure, meanwhile increasing the aggregate throughput. In contrast to the depth-optimizing approach, Sripanidkulchai et al.[25] propose the longest-first algorithm which, by utilizing peers' heavy tailed lifetime distribution, grants the

longest-lived peer with higher priority. These algorithms coincide with many findings in our dissertation, such as the optimal tree structure exhibited in the multi-tree setting under star network model.

Our approach is also closely related with the rich body of works to enhance stability of P2P systems[12, 30, 31, 5, 27], largely following two different strategies.

Parent-Child Switching

The first strategy is to gradually improve the stability of P2P structure by switching existing parent-child pairs to move up the stable peers. Existing algorithms include *mtreebone*[31], *stable peers*[30], and *ROST*[27].

Optimal Parent Selecting

The second strategy[5] tries to achieve the same goal by letting each joining peer find the optimal existing peer as its parent, often with the aid of comparable metrics, which our work follow as well. This type of solutions require a publish-subscribe service to help bootstrap a new peer to find its ideal parent(s). Existing approaches include *age-first*, *bandwidth-first*, and *hybrid*.

We compared the performance of our approach with *age-first*, *bandwidth-first*, *hybrid*, and *ROST*.

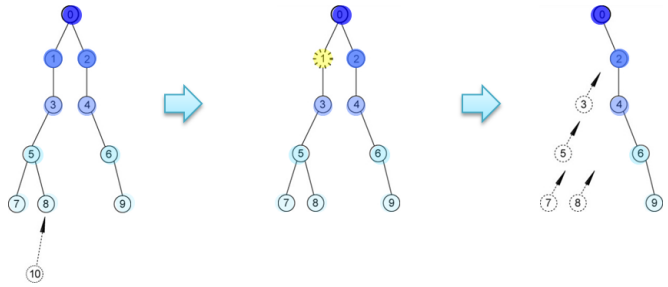


Figure II.2: *Age-first* approach

With *age-first* approach(See Fig.II.2), no nodes are older than higher level nodes. This may lead to total trees.

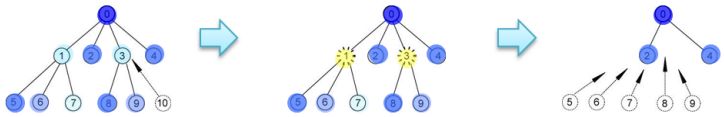


Figure II.3: *Bandwidth-first* approach

With *bandwidth-first* approach(See Fig.II.3), no nodes has larger bandwidth than higher level nodes. This may cause frequent peer rejoining to satisfy the bandwidth order.

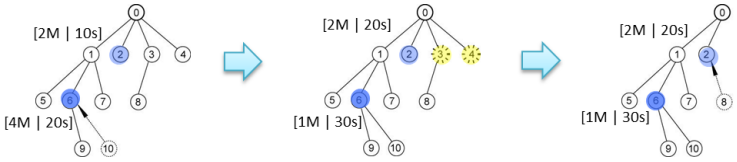


Figure II.4: *Hybrid* approach

Hybrid (See Fig.II.4) introduced $DegreeRatio$ and $AgeRatio$ for peer comparison. For two peers A and B, If their $DegreeRatio > 1$ and $AgeRatio > 1$, then peers A is more favorable than B when chosen as parent by newly joining peers. If their $(DegreeRatio^{-1})(AgeRatio^{-1}) < 0$, then peers A is more favorable than B, only if their $DegreeRatio > AgeRatio^{-p}$, in which p being an integer.

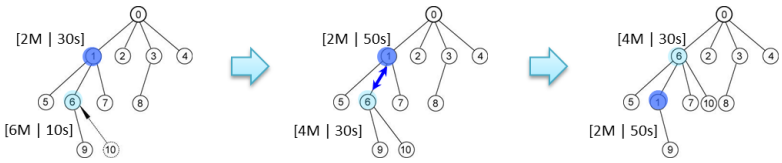


Figure II.5: *ROST* approach

ROST (See Fig.II.5) introduced a metric called BTP (Bandwidth-Time Product) for adjacent peer comparison. The parent-child pair will rotate when their BTP order

reverse.

Generalized Flow Optimization Framework

Optimization has long been practiced in network routing, primarily based on the multi-commodity flow theory. The basic idea is to assign weights to links to reflect their congestion conditions, and perform traffic routing based on the weights. In particular, the work of [2] and [4] present the theoretical models for online unicast routing. In the multicast domain, the work of [3] investigates the case of receivers within a multicast session arriving in batch, and the work in [13] presents a solution for receivers arriving separately. In the past, the multicommodity flow theory has extended to maximize the throughput of overlay multicasting[9]. This dissertation elaborates this effort to further incorporate fault resilience by introducing generalized flow theory.

Generalized flow has many applications[1], where the gain factors can model physical transformations of a commodity due to leakage, evaporation, breeding, theft, and transformations from one commodity into another as a result of manufacturing, scheduling, or currency exchange. Existing works have been focused on unicast routing[33, 28]. In particular, Wayne et al.[34] present a Dijkstra-variant shortest path algorithm for minimum-cost unicast-based generalized flow problem if all gain factors are below one. We find that when applying to multicasting, the difficulty of the problem increases rapidly due to the complexity brought up by the exponential cardinality of multicast tree set.

Finally, *PRW* is derived from the dual formulation of the generalized version of a multi-commodity problem. This methodology has been widely adopted in the area of traffic engineering[11], where a physical link is assigned a weight in reverse proportion of the link capacity, on which a shortest-path algorithm runs to find the lightest-load

route. In our work, the weights (dual variables) apply to peers directly, since their uplink capacities are the most precious resources in P2P streaming.

CHAPTER III

FRAMEWORK OVERVIEW

Network Model

We consider two kinds of network models: *general model* and *star model*.

General Network

We model the network as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, consisting of $\mathcal{N}$ nodes with capacity c_e on each physical edge $e \in \mathcal{E}$. On top of $\mathcal{G}$, an overlay network $G = (s, V, L)$ exists, where s is the server, and peers belong to the set $V = \{v\}$. Each overlay edge $l \in L$ connects two peers in V, and corresponds to the unicast route at the physical network $\mathcal{G}$.

Star Network

Many works have implicitly assumed that the bottleneck of a unicast path only happens at either access link of its two end hosts. In this way, we can simplify the general model into a star model. The central node of the star represents the Internet cloud, which reaches out to every peer. In this model, we denote the outbound bandwidth of peer $v \in V$ as c_v.

Overlay Organization

To transfer data among peers, the simplest and most straight forward strategy is a single multicast tree spanning from the server s to all peers in V. Although simple to

manage, this solution has clear drawback since a peer departure can cause complete disruption to all its descendants.

An alternative solution is the recently popular multi-tree or mesh solution, where each peer schedules to receive data from multiple parents. Since the mesh structure can be usually decomposed as the sum of multiple spanning trees, therefore will be categorized as multi-tree solution[1]. We denote the tree set as $T = \{t\}$, where each tree $t \in T$ covers all peers and has a single rate $f(t)$.

Resilience Factor and Generalized Flow

We assign a resilience factor r_v ($0 < r_v \leq 1$) to each peer $v \in V$. Our model makes no assumption on how r_v is defined. For the purpose of illustration, we introduce one way to define r_v. Suppose v follows certain lifetime distribution with c.d.f. $F(\tau)$, and T is a random variable denoting the time of departure, then the survival function of v is $1 - F(\tau) = Pr(T > \tau)$, the probability that its time of departure is later than time τ. If we denote $r_v = Pr(T > \tau^*)$, where τ^* is a fixed time point in the future, then it represents the chance of survival for v until τ^*.

Given the resilience factor of v, we consider two models to compute the rate of generalized flow.

Concatenation Model

For each peer v in tree t, there is a path from the server s to v, denoted as

$$\mathcal{P}_t(v) : s \mapsto v_1 \mapsto v_2 \mapsto \cdots \mapsto v_k \mapsto v$$

[1]We note that such categorization does not apply to the management of P2P network, but only suits the purpose of calculating throughput to each peer, which is the main focus of this research.

Given t's flow rate $f(t)$, the dependency model computes the generalized flow delivered to v as $f(t)$ timed by the concatenate product of r_{v_1} through r_{v_k}. We define such product as the resilience index of v in t:

$$R_t(v) = \prod_{i=1}^{k} r_{v_i} \tag{III.1}$$

Based on this definition, $f(t)R_t(v)$ is the generalized flow rate delivered to v in tree t. We can further define the resilience index of tree t as

$$R(t) = \sum_{v \in V} R_t(v) \tag{III.2}$$

Since $R_t(v) \leq 1$, it is obvious that $R(t) \leq |V|$. Now we are able to define generalized throughput of t, which is the sum of generalized flow rates to all peers.

$$f_g(t) = f(t)R(t) \tag{III.3}$$

This model computes a peer's generalized flow by factoring in the resilience factors of all its ancestors. It fits the live P2P streaming scenario where a peer failure can cause disruptions on all its descendants. Also an implicit assumption in the definition of $R_t(v)$ is that the resilience factor of server s is 1, i.e., s will not departure.

Non-Concatenation Model

In this model, we define the generalized flow to a peer to be only dependent on its immediate parent. Formally, in the same sample context of concatenation model, we define the resilience index of peer v in tree t as follows.

$$R_t(v) = r_{v_k} \tag{III.4}$$

This model fits better to P2P applications with no real time constraints. For example, in some on-demand streaming and downloading applications, the parent peer serves its children from its local cache. This gives its children buffering time to find new parent(s) upon its own departure or failure, thus absorbing the impact of cascading disruption.

Summary of Contributions

In Tab. III.1, we summarize findings when exploring along the three dimensions(topology model, P2P structure, and concatenation model) outlined in this section. Of the eight subproblems, we find four of them polynomially solvable and present the optimal solutions. Of the four NP-hard problems, we are able to find a $O(\log \mathcal{E})$-approximation algorithm, and only find heuristics to the other three.

	General Topology	**Star Topology**
Multiple Trees (Concatenation)	NP-hard (reduction to 3-SAT)	**MultiTrees-Star**, $O(n)$
Multiple Trees (Non-Concatenation)	**MultiTrees-General**, $O(\frac{\lvert\mathcal{E}\rvert}{\epsilon^2} \log U \cdot T_{mst})$	**MultiTrees-Star**, $O(n)$
Single Tree (Concatenation)	NP-hard (reduction to MPSP[8])	NP-hard (reduction to Hamilton Path[1])
Single Tree (Non-Concatenation)	NP-hard (linear-programming-relaxation is NP-hard)	**SingleTree-Star**, $O(n^3)$

Table III.1: Summary of Findings

We summarize notations that appeared in this chapter in Tab. III.2.

Notation	**Definition**
$\mathcal{G} = (\mathcal{N}, \mathcal{E})$	Physical Network
$G = (V, L)$	Overlay Network
s	server node
$\mathcal{E} = \{e\}$	physical layer edges
$L = \{l\}$	overlay layer links
$V = \{v\}$	overlay nodes
r, R	resilience index, e.g. r_v, $R(t)$, $R_t(v)$
$T = \{t\}$	overlay multicast trees
$f(t)$	data flow over tree t
$f_g(t)$	generalized flow over tree t
c	bandwidth constraint, e.g. , c_v, c_e, c_s
d_e	price of edge e
$\mathcal{P}_t(v)$	overlay routing path between s and v in overlay tree t

Table III.2: Notations Table

CHAPTER IV

GENERAL TOPOLOGY MODEL

In this section, we present our study on optimal generalized throughput under the general topology model(See Fig.IV.1).

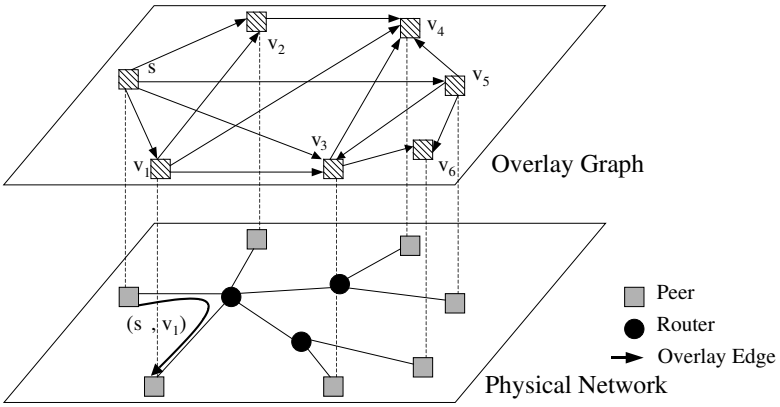


Figure IV.1: General Topology Model

Multiple Trees

We start with the most basic setting, where an unlimited number of trees can be constructed for the purpose of maximizing generalized throughput. With notions introduced in Chapter III (Framework Overview), we formulate it into the following linear programming (LP) problem.

$$\text{maximize} \quad \sum_{t \in T} f(t)R(t) \tag{IV.1}$$

$$\text{subject to} \quad \sum_{t \in T} n_e(t)f(t) \leq c_e, \forall e \in \mathcal{E} \tag{IV.2}$$

$$f(t) \geq 0, t \in T$$

The objective of problem (IV.1) is to maximize the generalized throughput (defined in Eq. (III.3)) of all trees. Inequality (IV.2) refers to the capacity constraint, i.e., the aggregate raw flow of all trees cannot exceed any physical link $e \in \mathcal{E}$. $n_e(t)$ is an integer variable indicating the number of times tree t has passed through e. Note since t is an overlay tree, $n_e(t)$ can be greater than 1.

The central difficulty of problem (IV.1) is that its number of variables is exponential to the size of the P2P network. Based on Cayley's theorem[10], the number of different spanning trees contained in T is $|T| = (|V|+1)^{(|V|-1)}$, $|V|$ being the number of peers in V.

On the other hand, the dimensionality of this problem, i.e., the number of constraints, is $|\mathcal{E}|$, the number of physical links. This gives us a chance to solve this problem via its dual presented as follows, which contains $|E|$ variables but exponential constraints.

$$\text{minimize} \quad \sum_{e \in \mathcal{E}} c_e d_e \tag{IV.3}$$

$$\text{subject to} \quad \sum_{e \in \mathcal{E}} n_e(t)d_e \geq R(t), \forall t \in \ T \tag{IV.4}$$

$$d_e \geq 0, e \in \mathcal{E}$$

Problem (IV.3) refers to assigning each link e a length d_e, and minimize the sum of d_e multiplied by the capacity c_e, subject to inequality (IV.4), which states that

the length of any spanning tree must be greater than its own resilience index $R(t)$, defined in Eq. (III.2).

Although there exists exponential number of trees in T, if we can find a separation oracle able to check whether constraint (IV.4) is met in polynomial time, then the dual problem (IV.3) is solvable in polynomial time, hence the primal problem.

To find if such an oracle exists, we first adapt the definition of $R(t)$ from peer-based to link-based, to be consistent with the left side of constraint (IV.4). This can be easily achieved as follows. We assign a resilience factor r_e to each link $e \in \mathcal{E}$, and define it as

$$r_e = \begin{cases} r_v & \text{if } e \text{ exits from } v \\ 0 & \text{otherwise} \end{cases} \tag{IV.5}$$

As articulated in Chapter III (Framework Overview), we have different definitions on $R(t)$ for concatenation and non-concatenation models. We start with the non-concatenation model first.

Based on the definition on resilience index $R_t(v)$ shown in Eq. (III.4), we can easily observe that $R(t)$ in this case is the sum of resilience factors of all non-leaf peers in tree t. Translated into the link-based definition, it is the sum of resilience factors of all links in t, i.e., $R(t) = \sum_{e \in \mathcal{E}} n_e(t) r_e$. This allows us to reformulate Inequality (IV.4) into the following.

$$\sum_{e \in \mathcal{E}} n_e(t) d_e \geq \sum_{e \in \mathcal{E}} n_e(t) r_e, \forall t \in \ T$$

It is now clear that the separation oracle is a minimum spanning tree algorithm that sees the cost on each link e as $(d_e - r_e)$. Constraint (IV.4) will be satisfied if the cost of the found minimum spanning tree is still greater than 0.

To this end, we present a fully polynomial time approximation scheme (FPTAS). FPTAS is a family of algorithms which finds a ϵ-approximate solution returning a result at least $(1-\epsilon)$ times the maximum value, for arbitrary error parameter $\epsilon > 0$.

MultiTrees-General$(\mathcal{E}, T)$
1 $\forall e \in \mathcal{E}$, $d_e \leftarrow \beta$, $l_e \leftarrow 0$
2 $f(t) \leftarrow 0$, $t \in T$
3 **loop**
4 $t^* \leftarrow$ minimum overlay spanning tree in T using $(d_e - r_e)$
5 $minlen \leftarrow \sum_{e \in \mathcal{E}} n_e(t^*)(d_e - r_e)$
6 **if** $minlen \geq 0$
7 **return**
8 $c \leftarrow \min_{e \in t^*} \frac{c_e}{n_e(t)}$
9 $f(t^*) \leftarrow f(t^*) + c$
10 $\forall e \in t$, $d_e \leftarrow d_e(1 + \epsilon \frac{n_e(t)c}{c_e})$,$l_e \leftarrow l_e + \frac{n_e(t)}{c_e}$
11 **end loop**
12 $l_{\max} \leftarrow \max_{e \in t} l_e$
13 $\forall t \in T$,$f(t) \leftarrow f(t)/l_{\max}$

Table IV.1: Finding Multiple Trees Under General Topology Model

Tab. IV.1 shows the **MultiTrees-General** algorithm. It solves the primal and dual problems in an iterative fashion. It sets initial length to β all links in $\mathcal{E}$. In each iteration, it finds the minimum spanning tree t^* based on the cost $(d_e - r_e)$, and route traffic over t^*. Based on the traffic increment, the length d_e is updated as defined in line 10. Finally, the algorithm terminates when constraint (IV.4) is satisfied, i.e., when the cost of the minimum spanning tree is greater than 0. Note that since the aggregated raw flow of all returned trees can exceed the capacity of certain physical links, we introduce the index l_e to record the congestion ratio on each link e. By scaling the rate of each tree with the maximum congestion noted by $l_{\max}$, the algorithm is guaranteed to return a feasible solution. We summarize the property of this algorithm in Theorem 1. See proof of Theorem 1 in Appendix A.

Theorem 1: Under the non-concatenation model, when $\beta = \frac{[(1+\epsilon)|V|]^{1-1/\epsilon}}{(|V|U)^{1/\epsilon}}$, the **MultiTrees-General** algorithm returns the solution at least $(1-2\epsilon)$ times the optimal result of problem (IV.1), with running time $O(\frac{|\mathcal{E}|}{\epsilon^2}[\log U + 2\log|V|] \cdot T_{mst})$. U is the length of the longest unicast route and T_{mst} is the running time of the minimum spanning tree algorithm.

Now we turn to the concatenation model, where the resilience index is defined in Eq. (III.1). In this case, each peer's resilience index is the product of resilience factors of all its ancestors. Although we can perform logarithm operation on resilience factors r_e and solve this problem using Dijkstra's algorithm(shortest path tree algorithm), it becomes extremely hard when combining with length assignment d_e, which needs to be solved by a minimum spanning tree algorithm. In the following theorem, we prove this problem to be NP-hard, by reducing its separation oracle to the problem of 3-SAT. See proof of Theorem 2 in Appendix B.

Theorem 2: Under the concatenation model, the **MultiTrees-General** problem (IV.1) is NP-hard.

Single Tree

A salient feature of the **MultiTrees-General** algorithm is that it reveals the maximum generalized throughput a P2P network can achieve. However, given the exponential selection space in tree set T, the algorithm often returns a high number of trees, which are hardly manageable in practice. For practical purposes, we enforce a limit on the number of trees we can construct. To achieve so, we modify problem (IV.1) into the following integer programming problem.

$$\text{maximize} \quad \sum_{t \in T} f(t)R(t)x(t) \tag{IV.6}$$

$$\text{subject to} \quad \sum_{t \in T} n_e(t)f(t)x(t) \leq c_e, \forall e \in \mathcal{E} \tag{IV.7}$$

$$\sum_{t \in T} x(t) = k \tag{IV.8}$$

$$f(t) \geq 0, x(t) = \{0, 1\}, t \in T$$

Problem (IV.6) introduces a 0-1 variable $x(t)$, and k, the upper limit on the number of trees. This constraint is enforced by Eq. (IV.8). This problem is NP-hard since its special case has been proved so. When $k = 1$ and resilience factor of all peers are 1, this problem reduces to maximizing the throughput of a single overlay multicast tree, which was shown NP-hard in [8] under the name MPSP (Multicast Path Set Problem).

Following the same idea of the **MultiTrees-General** algorithm, we assign length to each physical link to find minimum spanning tree, but only in an online fashion. The algorithm runs k iterations, in each of which a tree is returned (See Tab.IV.2).

$k-$**Tree**$(\mathcal{E}, T)$
1 $\quad \forall e \in \mathcal{E}$, $d_e \leftarrow \beta / c_e, l_e \leftarrow 0$
3 $\quad$ **for** $i = 1$ to k **do**
4 $\quad\quad t_i \leftarrow$ minimum overlay spanning tree in T using $(d_e - r_e)$
5 $\quad\quad f(t_i) \leftarrow 1$
6 $\quad\quad \forall e \in t$, $d_e \leftarrow d_e(1 + \rho \frac{n_e(t)}{c_e})$,$l_e \leftarrow l_e + \frac{n_e(t)}{c_e}$
7 $\quad l_{\max} \leftarrow \max_{e \in t} l_e$
8 $\quad$ **for** $i = 1$ to k **do**
9 $\quad\quad f(t_i) \leftarrow f(t_i)/l_{\max}$

Table IV.2: Finding k Trees Under General Topology Model

Finally, we note that the two algorithms presented in this section can be also applied to the concatenation model. However, theorem 1 will not apply due to the NP-hardness of separation oracle for problem (IV.1).

CHAPTER V

STAR TOPOLOGY MODEL

The algorithms presented in Chapter IV (General Topology Model) rely on linear programming technique, and operate on both overlay network L and physical network $\mathcal{E}$. They require complete knowledge on both networks, i.e., the capacity of each link $e \in \mathcal{E}$ and the underlying routing path that connects any overlay link $l \in L$.

Many P2P research works have chosen to rely on a simplified assumption, which imposes an outgoing bandwidth constraint on each peer and allow it to parent other peers until its outbound bandwidth depletes. In other words, the Internet cloud is assumed to have enough capacity supporting all peers. This effectively transforms the physical network $\mathcal{E}$ into a star network, whose central hub represents the Internet cloud reaching out to all peers, as shown in Fig. V.1. A typical scenario that can be best represented by this model is campus news video streaming system. Students subscribe to the news, and watch the video with smart phones. The bottleneck of the P2P system lies in the last hop connection (smart phone), rather than the internet cloud (video server). In this section, we study the same set of problems under such a special topology.

Multiple Trees

We again start with the case of multiple trees. To simplify the illustration, we remove notations associated with the general network $\mathcal{E}$. Instead, we introduce notations c_v to denote outbound bandwidth of peer $v \in V$, c_s to denote outbound

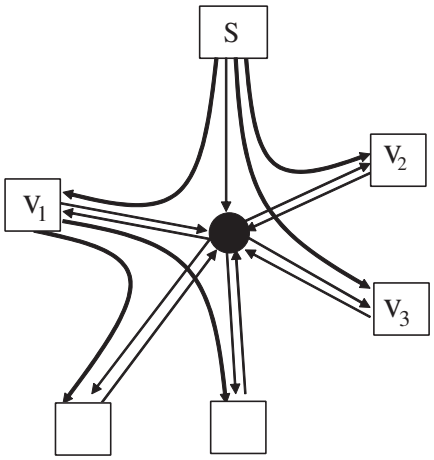


Figure V.1: Star Topology Model

bandwidth of the server s, and $n_v(t)$ or $n_s(t)$ to denote the number of children v or s have in tree t. The problem formulation is as follows[1].

$$\text{maximize} \quad \sum_{t \in T} f(t)R(t) \tag{V.1}$$

$$\text{subject to} \quad \sum_{t \in T} n_v(t)f(t) \leq c_v, v \in V \tag{V.2}$$

$$\sum_{t \in T} n_s(t)f(t) \leq c_s \tag{V.3}$$

$$f(t) \geq 0, t \in T \tag{V.4}$$

Inequalities (V.2) and (V.3) refer to the capacity constraint. In fact, problem (V.1) is only a special case of the problem (IV.1), thus can be solved by algorithm **MultiTrees-General** in the same linear programming fashion. However, given the simplified topology, we are interested to find out if this problem can be simply addressed through combinatorial optimization techniques.

[1]We note that unless otherwise notified, our discussion in this section assumes that the inbound bandwidth of each peer v is unbounded, thus removed from the problem formulation. By the end of this section, we will introduce how our algorithms could be adapted to incorporate the inbound bandwidth constraint.

MultiTrees-Star(s, V)
1 sort V into $v_1, \ldots, v_n$ in descending order of resilient index
2 for $i = 1$ to $|V|$
3 construct tree t_i, where v_i is the only child of s,
and v_j $(j \neq i)$ are children of v_i
4 $f(t_i) \leftarrow \min\{\frac{c_{v_i}}{|V|-1}, c_s\}$
5 $c_s \leftarrow c_s - \min\{\frac{c_{v_i}}{|V|-1}, c_s\}$
6 if $c_s > 0$
7 construct tree t_0, where s is the parent of v_i $(i = 1, \ldots, |V|)$
8 $f(t_0) \leftarrow c_s/|V|$

Table V.1: Finding Multiple Trees Under Star Topology Model

Tab. V.1 shows the **MultiTrees-Star** algorithm. It construct at most $|V| + 1$ trees in the order shown in Fig. V.2.

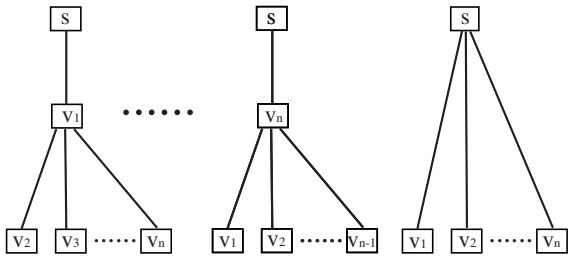


Figure V.2: Illustration of MultiTrees-Star Algorithm

Starting from peer v_1 with the maximum resilience factor, tree t_1 is constructed, which depletes the outbound bandwidth of v_1. The process continues until $v_{|V|}$ or the server bandwidth c_s runs out. If there is still residue of c_s after tree $t_{|V|}$ is finished, we construct a special tree t_0 to deplete c_s. We show the optimality of this simple algorithm as follows. See proof of Theorem 3 in Appendix C.

Theorem 3: **MultiTrees-Star** algorithm returns the optimal result of problem (V.1).

It is easy to observe that this theorem applies to both non-concatenation and concatenation models, since trees t_0 through t_n return the same resilience index under either definition, i.e., Eq. (III.1) for concatenation model and Eq. (III.4) for non-concatenation model.

The trees found by the **MultiTrees-Star** algorithm comply with many heuristics practiced by existing works. In terms of tree structure, t_0 through t_n are "fat trees" or "minimum-depth tree". In terms of construction order, the algorithm starts from the peer with maximum resilience factor, which suggests maximum lifetime. This complies with the "longest-first" approach that assigns higher priority to peers with longer expected lifetime.

Single Tree

The number of trees returned by the **MultiTrees-Star** algorithm scales up linearly with $|V|$, the size of the P2P network. Although more scalable than the **MultiTrees-General** algorithm, the number of trees can be still too big as the P2P network grows.

To limit the number of trees, we can impose an additional integer constraint over problem (V.1) in the same fashion we defined problem (IV.6) under the general topology model.

$$\text{maximize} \quad \sum_{t \in T} f(t)R(t)x(t) \tag{V.5}$$

$$\text{subject to} \quad \sum_{t \in T} n_v(t)f(t)x(t) \leq c_v, v \in V \tag{V.6}$$

$$\sum_{t \in T} n_s(t)f(t)x(t) \leq c_s \tag{V.7}$$

$$\sum_{t \in T} x(t) = k \tag{V.8}$$

$$f(t) \geq 0, x(t) = \{0, 1\}, t \in T \tag{V.9}$$

In particular, we are interested in the case when $k = 1$, i.e., when only one tree is allowed.

We start with the non-concatenation model. Tab. V.2 shows the **SingleTree-Star** algorithm, which further contains two sub-algorithms. **MaxRate-Star** finds out the maximum rate a single tree can possibly afford. It works in a trial-and-error fashion by proposing a rate f and learning the maximum outbound degree the server s and each peer in V can support based on f. Starting from the server outbound bandwidth c_s, f keeps shrinking until the sum of outbound degrees exceeds or equals to $|V|$, the number of peers. **MostResilientTree-Star** is a greedy algorithm constructing the tree with the highest resilience index. Given rate f, it gives priority to peers with the highest resilience factor and assign children to them up to their maximum outbound degrees. The algorithm returns the generalized throughput, which according to the definition in Eq. (III.3), is the product of f and the returned tree's resilient index.

With these two sub-algorithms, The **SingleTree-Star** algorithm finds the optimal tree by feeding different rates to **MostResilientTree-Star** and keeping track the tree returning the maximum generalized throughput. The trial starts from the maximum affordable rate found by **MaxRate-Star**, and ends when the outbound degree of the server s becomes $|V|$. In this case, we can construct tree t_0 shown in

MaxRate-Star(s, V)
1 sort V into $v_1, \ldots, v_{|V|}$ in descending order of bandwidth
2 $f \leftarrow c_s$
3 find k such that $c_{v_{k-1}} \geq f > c_{v_k}$
4 do
5 $i \leftarrow 0$, $sum \leftarrow 0$
6 while $sum < |V|$ and $i < |V|$
7 $sum \leftarrow sum + \lfloor c_{v_i}/f \rfloor$, $i \leftarrow i + 1$
8 if $sum < |V|$
9 $f \leftarrow c_{v_k}$,$k \leftarrow k + 1$
10 while $sum < |V|$
10 return f

MostResilientTree-Star(f, s, V)
1 sort V into $v_1, \ldots, v_{|V|}$ in descending order of resilience
2 enqueue s, v_1,. . .,$v_{|V|}$ into queue P
3 enqueue v_1,. . .,v_n into queue C
4 $sum \leftarrow 0$
5 while $C \neq \phi$
6 dequeue v_{parent} from P
6 repeat $\lfloor c_{v_{parent}}/f \rfloor$ times
7 dequeue v_{child} from C
8 make v_{parent} the parent of v_{child}
9 $sum \leftarrow sum + r_{parent}$
10 return $\{f * sum, \text{resulting tree}\}$

SingleTree-Star(s, V)
1 $f \leftarrow$ **MaxRate-Star**(s, V)
2 $\{max, tree^*\} \leftarrow$ **MostResilientTree-Star**(f, s, V)
3 while $f * |V| > c_s$
4 $f_{new} \leftarrow c_s/(\lfloor c_s/f \rfloor + 1)$
5 for $i = 1$ to $|V|$
6 if $f_{new} < c_{v_i}/(\lfloor c_{v_i}/f \rfloor + 1)$
7 $f_{new} \leftarrow c_{v_i}/(\lfloor c_{v_i}/f \rfloor + 1)$
8 $f \leftarrow f_{new}$
9 $\{r, tree\} \leftarrow$ **MostResilientTree-Star**(f, s, V)
10 if $r > max$
11 $max \leftarrow r$, $tree^* \leftarrow tree$
12 return $tree^*$

Table V.2: Finding Single Tree Under Star Topology

Fig. V.2, which has the maximum resilience index $|V|$. Since in each iteration of **SingleTree-Star**, at least one peer's outbound degree will increase by 1, the number of iterations is bounded by $O(|V|^2)$. Combined with the linear running time of **MostResilientTree-Star**, its overall running time is $O(|V|^3)$. The following theorem establishes the optimality of **SingleTree-Star**.

Theorem 4: Under the non-concatenation model, the **SingleTree-Star** algorithm returns the optimal solution for problem (V.5) when $k = 1$, with running time $O(|V|^3)$.

When applying the same algorithm to concatenation case, we find that the greedy approach of **MostResilientTree-Star** does not fit the multiplicative definition of resilience factor given in Eq. (III.1). Essentially, although finding the tree with the maximum resilience index is solvable by a multiplicative-variant of Dijkstra's algorithm, it becomes hard when imposing degree constraints on the peers. To prove NP-hardness of this problem, we consider its special case, the maximum multiplicative cost path problem (MMCP), then reduce it to the Hamiltonian path problem (finding a path in an undirected graph that visits each vertex exactly once).

Evidently, under the concatenation model, the intrinsic conflict between outbound bandwidth and resilience factor poses great barrier to our effort to assign priority in peer selection. On the other hand, the problem becomes polynomially solvable under the same framework of **SingleTree-Star** if all peers are identical over either of above metrics. For example, if all peers have the same resilience index, we only need to modify **MostResilientTree-Star** to greedily choose peers with the highest outbound bandwidth. If all peers have the same outbound bandwidth, the exact algorithm in Tab. V.2 can be reused with no modification.

Finally, we note that algorithms listed in this section have assumed the inbound bandwidth of all peers are unlimited. Nevertheless, they can be easily modified when applying this constraint. Since our research studies single-rate multicast, i.e., the rate of raw flow delivered to each peer is the same, we only consider c_{in}, the minimum inbound bandwidth of all peers. If $c_{in} \geq c_s$, then one should not be concerned since the raw flow delivered to a peer cannot exceed the outbound bandwidth of the server s. Otherwise, c_{in} replaces c_s as the bottleneck. As such, we only need to replace c_s with $\min\{c_s, c_{in}\}$ in Tab. V.1 and V.2.

Discussions

We conclude the optimization framework section, by discussing the implementability of algorithms presented so far. Given the unlimited number of trees the **MultiTrees-General** algorithm can produce, its main purpose remains to provide the theoretical optimal point against which other practical solutions can be measured. The k-**Tree** algorithm avoids this pitfall by limiting the number of trees. However, its functioning requires measurement overhead of the underlying physical network. The **MultiTrees-Star** and **SingleTree-Star** algorithms address both of the above issues. However, a centralized entity, e.g., the server s, needs to be in place. It collects uplink bandwidth information and resilience factors from all peers, then runs the algorithm. It is reasonable to expect the server to keep the up-to-date information of the peers it serves, however, distributed versions of these algorithm would be more desirable, we dedicate the following sections to a practical evaluation metric called PRW.

CHAPTER VI

PERFORMANCE EVALUATIONS – OPTIMIZATION FRAMEWORK

In this section, we present our evaluation study, which mainly carries two purposes. First, we will evaluate the validity of the generalized flow optimization framework at capturing the key characteristics of fault resilient peer selection problem. Second, we will study the performance of the algorithms proposed in this paper, as well as several well-known heuristics, at maximizing the generalized throughput and maintaining fairness.

Experimental Setup

We use simulation to evaluate the performance of our algorithm. Two experimental topologies are chosen. The first one is a 1000-node router-level network (2000 edges) created with the Boston BRITE topology generator [21] using the Waxman model [32]. Any pair of routers are connected by a pair of links with opposite directions. The bandwidth of physical links between routers, as well as peers' access links, are normally distributed from 100Kbps to 1000Kbps. The second topology follows the star configuration outlined in Fig. V.1.

Under both topologies, we create 100 peers with unlimited inbound bandwidth. Under the general topology, they are randomly attached to the routers in the network.

Each simulation run lasts a finite time period. Starting from time 0, each peer is assigned a lifetime based on exponential and Pareto lifetime distributions with mean lifetime varying from 1500 seconds to 3500 seconds. The simulation run expires when the lifetime of the longest-lived peer expires. In our simulation, we assign resilience factor to each peer based on its expected lifetime in each particular run.

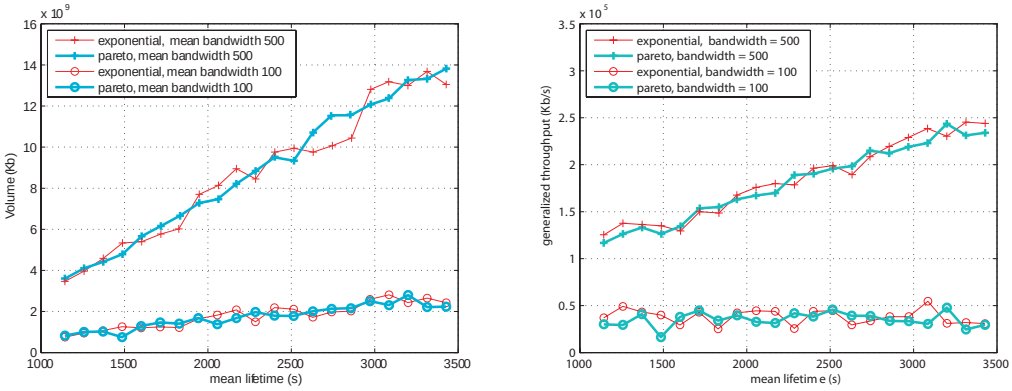
Our algorithms are executed at the beginning of each run, taking the resilience factors and outbound bandwidths of all peers as the input, and returning single or multiple trees whose combined generalized throughput is maximized.

As time proceeds, peers expire one by one, which gradually tears down the tree(s) constructed at the beginning of the simulation. To capture this effect, we accumulatively calculate the amount of data collected by each peer until its ancestor or itself fails. We term this result as *volume*, which represents the capability of the constructed tree(s) at collecting data for all peers before they demise.

Generalized Throughput vs. Volume

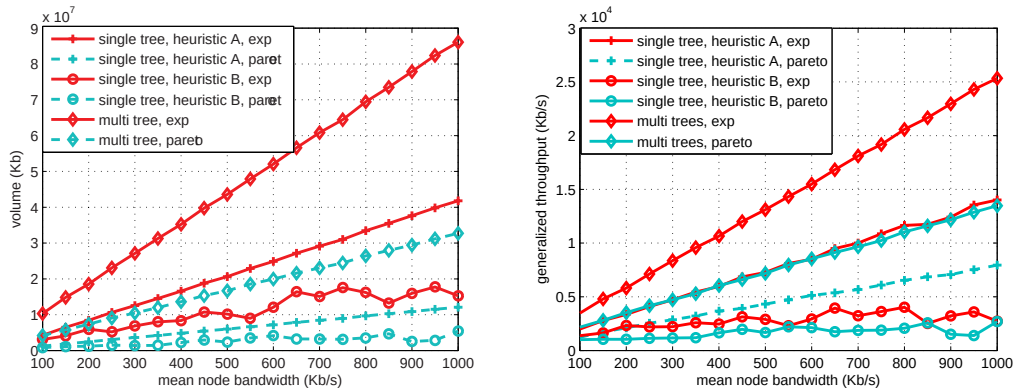
The objective of our algorithms is to maximize the generalized throughput, given resilience factors of all peers. However, it merely represents the expected amount of data the constructed tree(s) can possibly collect. Therefore, to test the fitness of our model under simulated P2P network with peer dynamics, we need the volume as a metric, which counts the total amount of data collected. If our experiment can establish a proportional relationship between generalized throughput and volume, then we can claim with high confidence level that our optimization framework can effectively model the dynamics of P2P network, and the developed optimization algorithms are able to increase the resilient throughput under such dynamics. Based on this consideration, our simulation does not include repairing mechanisms, i.e., a peer is not allowed to reconnect to the P2P network once disconnected due to the departure of either its ancestor or itself. This way, the recorded volume can more accurately reflect the resilience of the tree(s) constructed by our algorithm.

In Fig. VI.1, we run the **MultiTrees-General** algorithm under the general topology, and contrast the generalized throughput returned by the algorithm in (a), calculated volume in (b). We observe that the performance difference under two lifetime



(a) Volume (b) Generalized Throughput

Figure VI.1: **MultiTrees-General** under Non-Concatenation Model

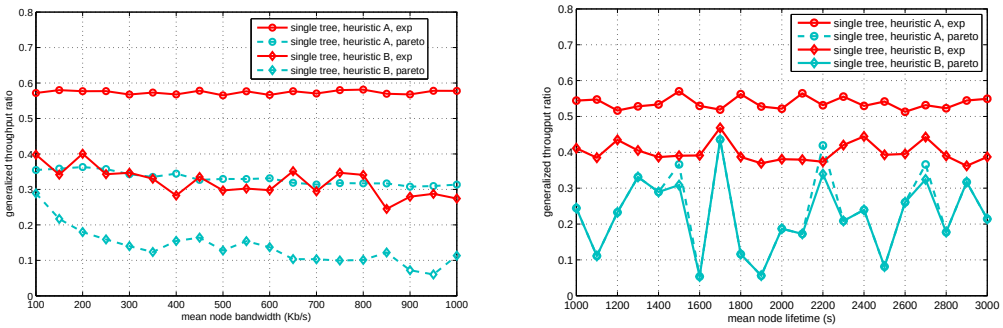


(a) Volume (b) Generalized Throughput

Figure VI.2: **MultiTrees-Star** and Two Heuristics under Concatenation Model (Mean Lifetime = 1500s)

distributions are consistently obeyed in both figures when varying the mean peer lifetime.

We then run the **MultiTrees-Star** under the star topology, and contrast the generalized throughput and volume by varying the mean outbound bandwidth. We further introduce two heuristic single-tree algorithms. In both heuristics, we compute the mean outbound bandwidth, and the mean resilience factors of all peers, then assign rate of the tree as the ratio of the two. Heuristic A constructs the tree by assigning priorities to peers with higher resilience factors, and heuristic B assign



(a) Outbound Bandwidth Mean Lifetime = 1500s

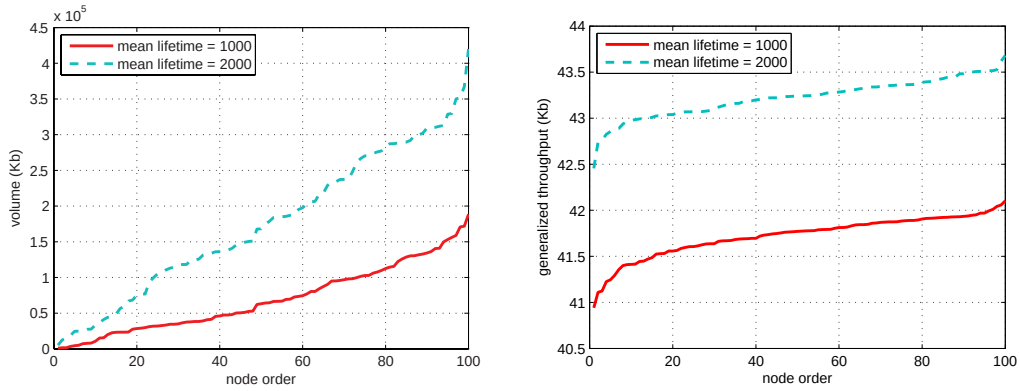
(b) Lifetime Mean Outbound Bandwidth = 100Kbps

Figure VI.3: Performance Ratio of Heuristics to **MultiTrees-Star** under Concatenation Model

priorities to the ones with higher outbound bandwidth. In Fig. VI.2, we showed the volume and generalized throughput of our MultiTrees-Star algorithm as well as the two heuristics, under the same experiment settings. The generalized throughput is the performance metrics we want to maximize under our optimization framework. Intuitively, it should reflect the traditional throughput metric: the volume of data delivered, which ultimately affect the viewer perception of video streaming quality. Our purpose is simple: if algorithms not developed under our optimization framework can still establish proportional relationship between generalized throughput and volume, then it becomes more convincing that the generalized flow model can effectively capture the dynamic characteristics of P2P network. As shown in Fig. VI.2, performance ordering of these algorithms under different lifetime distributions are consistent in both figures.

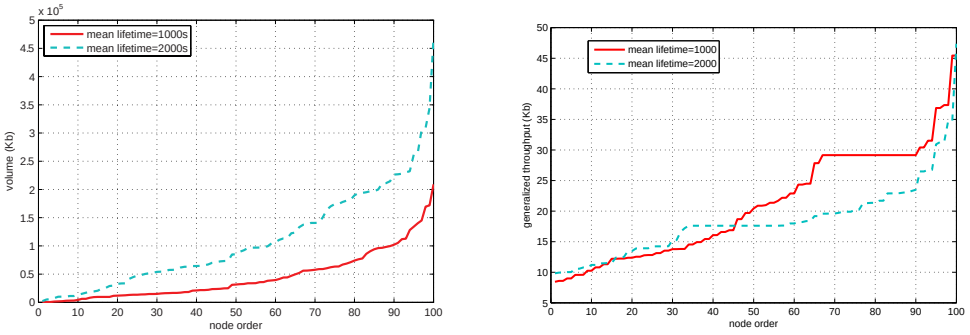
Performance of Single Tree Algorithms

To evaluate the ability of single tree algorithms at maximizing the generalized throughput, we run heuristics A and B, and **SingleTree-Star** algorithm under the



(a) Volume (b) Generalized Throughput

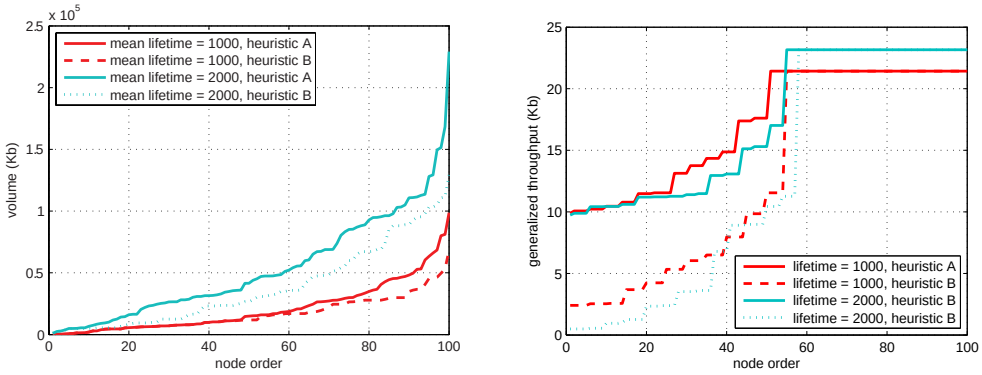
Figure VI.4: Sorted Nodes of **MultiTrees-Star** (Mean Outbound Bandwidth = 100Kbps)



(a) Volume (b) Generalized Throughput

Figure VI.5: Sorted Nodes of **SingleTree-Star** under Non-Concatenation Model (Mean Outbound Bandwidth = 100Kbps)

star topology and concatenation model, and normalize their results with the one achieved by the optimal **MultiTree-Star** algorithm. In Fig. VI.3, we observe that all of them are able to maintain the performance ratio (The ratio of generalized throughput of the heuristics to our MultiTrees-Star algorithm) from 0.1 to 0.6 under different mean outbound bandwidths, mean lifetime, and lifetime distributions. Which means, the performance of our algorithms are consistently better than the two heuristics. Specifically, the two heuristics perform only up to 60 percent as good in terms of generalized throughput.



(a) Volume (b) Generalized Throughput

Figure VI.6: Sorted Nodes of Heuristics under Concatenation Model (Mean Outbound Bandwidth = 100Kbps)

In Fig. VI.4, VI.5, and VI.6, we display the sorted per-node generalized throughput and volume for **MultiTrees-Star**, **SingleTree-Star**, and the two heuristics. We have showed the trend accordance of our optimization framework under variance mean lifetime and mean bandwidth in Fig. VI.1, VI.2, and VI.3, which reflect generalized throughput's lifetime-wise and bandwidth-wise capturing ability of the volume. Fig. VI.4, VI.5, and VI.6 showed the corresponding volume and generalized throughput of each node when using **MultiTrees-Star**, **SingleTree-Star**, and the heuristics. They demonstrate the node-wise capturing ability of the volume using generalized throughput.

CHAPTER VII

PEER EVALUATION METRIC OVERVIEW

Network Model

Similarly, we consider a server s and a set of peers $v \in V$. Each peer is associated with downlink(download link) bandwidth and uplink(upload link) bandwidth. This model also builds on the assumptions of many existing works that the bottleneck of a unicast path only happens at either access link of its two end hosts. Therefore, this assumption effectively simplifies the Internet to the star topology shown in Fig. V.1.

For simplicity purposes, we assume that the downlink bandwidth of a peer always exceeds the streaming rate. Therefore, we only denote the uplink bandwidth of peer v as $b(v)$.

Assumption 1 : *Network bottleneck in peer uplink*

We further assume that in a P2P streaming system, the streaming rate is the same across all peers. This is in accordance with the state of the art, where the content is encoded by a non-scalable codec once (often at the server s) and distributed to all peers. This choice is popular among existing P2P streaming systems due to its codec-neutral feature, which regards the media streams as meaningless bit flows. In order to support heterogeneous receiving rate, peer transcoding or scalable codec has to be in place, neither of which has produced reliable solutions for the purpose of large scale content distribution.

Assumption 2 : *Same streaming rate for all peers*

Resilience Factor of a Peer

We assign a resilience factor r_v ($0 < r_v \leq 1$) to each peer $v \in \mathcal{V}$. Our model makes no assumption on how r_v is defined. In this dissertation, we adopt a simple definition of r_v as follows.

$$r_v = \begin{cases} \frac{2\tau_v}{T_v} & \text{if } 2\tau_v < T_v \\ 1 & \text{otherwise} \end{cases} \tag{VII.1}$$

where τ_v is the age of peer v, i.e., the amount of time elapsed since v joins. T_v is the maximum lifetime of v. In video-on-demand applications, T_v can be denoted as the length of the movie v joins to watch. In the live streaming scenario, T_v can be the duration from v's joining time to the time when the event terminates.

Eq. (VII.1) follows the definition of stability index in [30]. It is consistent with the well-known observation that peers already living for an extended period of time might as well continue to live. Intuitively, Eq. (VII.1) states that if v stays over half of its maximum lifetime, it will stay until the application terminates.

Ways	**Resilience factor**
performance	$r_v = \sum_{i=1}^{N} a_i r_v^i, \sum_{i=1}^{N} a_i = 1, r_v^i = a\tau_v^i / T_v^i$
voluntariness	$r_v = a_1 r_v + a_2 b_v^{max}/c_v, a_1 + a_2 = 1$
patience	$r_v = a(\tau_v - \tau_v^{'})/T_v$

Table VII.1: Other Resilience Factor Definitions

Other resilience factor definitions could be, grading past performance, rewarding voluntariness or monitoring patience. Grading past performance incorporates past resilience factors (r_v^i) by weighted summation. Rewarding voluntariness [24] factors

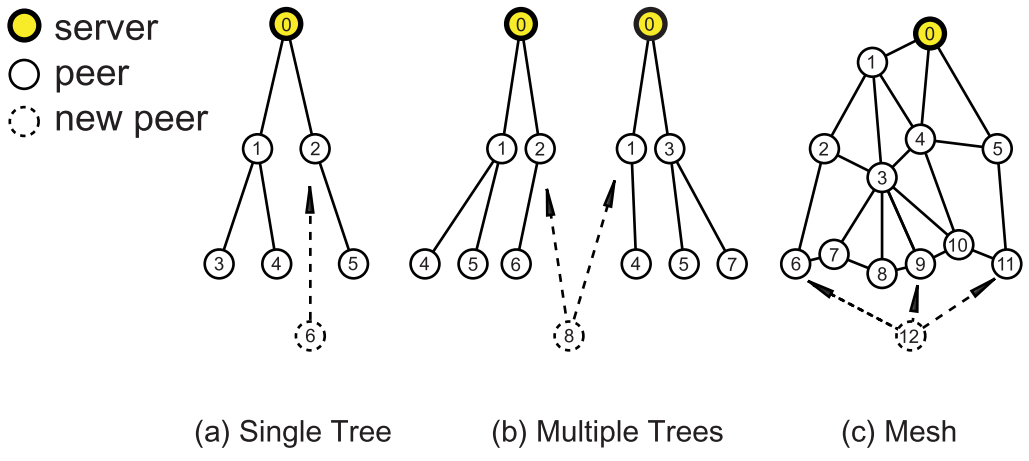


Figure VII.1: P2P Distribution Structure

in the proportion of its uplink bandwidth capacity (c_v) a peer sets as maximum uploading limit (b_v^{max}). Patience monitoring factors out the time duration a peer fast forwards by dragging the seek bar ($\tau_v^{'}$), from the actual watching time (τ_v).

Resilience Index of P2P Distribution Structures

We consider three distribution structures in P2P streaming, i.e., single overlay tree, multiple trees, and mesh (See Fig. VII.1). All structures are rooted at the server s.

Single Tree

Starting on the case of single overlay tree, consider a peer v in a tree t, there is a path from the server s to v, denoted as

$$s \mapsto v_1 \mapsto v_2 \mapsto \cdots \mapsto v_k \mapsto v$$

Given t's flow rate $f(t)$, we denote the generalized flow delivered to v as $f(t)$ timed by the concatenate product of r_{v_1} through r_{v_k}. We define such product as the resilience index of v in t:

$$R_t(v) = \prod_{i=1}^{k} r_{v_i} \tag{VII.2}$$

In other words, let $p_t(v)$ denote the peer which is the parent of peer v in tree t, then Eq. (VII.2) can be regarded as the product of resilience index and resilience factor of $p_t(v)$:

$$R_t(v) = R_t(p_t(v))r_{p_t(v)} \tag{VII.3}$$

Based on this definition, we can further define the resilience index of tree t as

$$R(t) = \sum_{v \in V} R_t(v) \tag{VII.4}$$

This model computes a peer's generalized flow by factoring in the resilience factors of all its ancestors. Also an implicit assumption in the definition of $R_t(v)$ is that the resilience factor of server s is 1, i.e., s will not depart.

Multiple Trees

In the case of multiple trees, peers can organize themselves into different forms of overlay trees, each carrying certain flow rate. Let $\mathcal{T}$ be the set containing all possible trees. Based on Cayley's theorem[10], the size of $\mathcal{T}$ is exponential, i.e., $|\mathcal{T}| = (|\mathcal{V}| + 1)^{(|\mathcal{V}|-1)}$, $|\mathcal{V}|$ being the number of peers in $\mathcal{V}$. However, any practical multi-tree solution only assigns traffic to a handful of trees in $\mathcal{T}$, and leaves the rates of dominating majority of trees as 0.

Furthermore, the total rate received by a peer is the sum of rates it collects from all trees it resides in. Formally, let $f(\mathcal{T})$ be the total receiving rate of the multi-tree structure, then we have:

$$f(\mathcal{T}) = \sum_{t \in \mathcal{T}} f(t)x(t)$$

where $x(t)$ is a 0-1 variable. $x(t) = 1$ if t is chosen to be constructed, and 0 otherwise.

Let $\phi(t) = f(t)/f(\mathcal{T})$ be the ratio of tree t's contribution to the total rate, then we have:

$$\sum_{t \in \mathcal{T}} \phi(t) = 1$$

Given the definition of $\phi(t)$, we define the resilience index of v in the tree set $\mathcal{T}$ as follows.

$$R_{\mathcal{T}}(v) = \sum_{t \in \mathcal{T}} R_t(v)\phi(t) \tag{VII.5}$$

Accordingly, the resilience index of the multi-tree structure is

$$R(\mathcal{T}) = \sum_{v \in \mathcal{V}} R_{\mathcal{T}}(v) = \sum_{t \in \mathcal{T}} \phi(t)R(t) \tag{VII.6}$$

Mesh

In mesh structure, each peer is allowed to have multiple parents, and each peer's receiving rate is the sum of the data rates it collects from all its parents. Since we assume all peers in mesh m to have the same receiving rate, then m can be decomposed into a set of trees, which degenerates itself to the case of multiple trees.

Nevertheless, the difference between these two structures is significant. In the mesh structure, a peer can dynamically adjust the receiving rates from its parents on

the fly, whereas in the multi-tree structure, doing so will change the rate assignment of the entire tree. Since each tree has a single receiving rate, such a change will force all peers to readjust their receiving rates from their parents.

Under the mesh structure, the resilience index of a peer v can be calculated in the similar fashion as the multi-tree structure. Let $\mathcal{P}_m(v)$ be the set of parents of v. For each parent $p \in \mathcal{P}_m(v)$, we denote $f_{p\to v}(m)$ as the flow rate from p to v, then the total receiving rate of v, hence the total receiving rate of the mesh structure m, is defined as:

$$f(m) = \sum_{p\in\mathcal{P}_m(v)} f_{p\to v}(m), \forall v \in \mathcal{V}$$

Let $\phi_{p\to v}(m) = f_{p\to v}(m)/f(m)$ be the ratio of p's contribution to the total receiving rate of v, then it is obvious that:

$$\sum_{p\in\mathcal{P}_m(v)} \phi_{p\to v}(m) = 1, \forall v \in \mathcal{V}$$

Given the definition of $\phi_{p\to v}(m)$, we define the resilience index of v in the mesh structure m as the weighted sum of the products of its parents' resilience index and its own residence factor.

$$R_m(v) = \sum_{p\in\mathcal{P}_m(v)} R_m(p) r_p \phi_{p\to v}(m) \tag{VII.7}$$

Then the resilience index of m is

$$R(m) = \sum_{v\in\mathcal{V}} R_m(v) \tag{VII.8}$$

Optimization Framework based on the Generalized Flow Theory

Based on our optimization framework, we define the generalized flow of a P2P distribution structure as the product of its total receiving rate and resilience index. Specifically, the generalized flow is $f(t)R(t)$ for the single-tree structure, $f(\mathcal{T})R(\mathcal{T})$ for the multi-tree structure, $f(m)R(m)$ for the mesh structure.

Problem Formulation

In the following integer linear programming (ILP) problem, we define our goal as to maximize the generalized flow. Note that we only present the formulation for the multi-tree structure, since the mesh structure can be effectively reformulated as a collection of trees, and the formulation of the single tree structure is only a special case of the multi-tree structure.

$$\text{maximize} \quad f(\mathcal{T})R(\mathcal{T}) = \sum_{t \in T} f(t)R(t)x(t) \tag{VII.9}$$

$$\text{subject to} \quad \sum_{t \in T} n_v(t)f(t)x(t) \leq b_v, \forall v \in \mathcal{V} \tag{VII.10}$$

$$\sum_{t \in T} n_s(t)f(t)x(t) \leq b_s \tag{VII.11}$$

$$\sum_{t \in T} x(t) \leq k \tag{VII.12}$$

$$f(t) \geq 0, x(t) = \{0, 1\}, t \in T$$

Here, Inequality (VII.10) and (VII.11) refer to the capacity constraint, i.e., the aggregate flow of all trees cannot exceed the uplink capacity of all peers in $\mathcal{V}$ and the server s. $n_v(t)$ and $n_s(t)$ are integer variables indicating the number of children that v and s have in the tree t. Inequality (VII.12) is the integral constraint limiting the number of trees to be no more than k. The single-tree structure can be enforced by setting $k = 1$.

The central difficulty of problem (VII.9) is that its number of variables is exponential to the size of the P2P network, i.e., there exist exponential number of trees in $\mathcal{T}$, as revealed in Chapter IV (General Topology Model). The existence of the integer variables $x(t)$ makes this problem more challenging.

Solution Methodology

In Chapter IV (General Topology Model), we examined the linear relaxation of problem (VII.9), which allows unlimited number of trees to be constructed[1]. We find the problem solvable in polynomial time for the following reason. Although containing exponential number of variables, the dimensionality of this problem, i.e., the number of constraints, is $|\mathcal{V}|$, the number of peers. This gives us a chance to solve this problem via the dual of the linear relaxation of this problem as follows, which contains $|\mathcal{V}|$ variables but exponential constraints.

$$\text{minimize} \quad c_s d_s + \sum_{v \in \mathcal{V}} c_v d_v \tag{VII.13}$$

$$\text{subject to} \quad n_s(t) d_s + \sum_{v \in \mathcal{V}} n_v(t) d_v \geq R(t), \forall t \in \mathcal{T} \tag{VII.14}$$

$$d_s \geq 0, d_v \geq 0, \forall v \in \mathcal{V}$$

Problem (VII.13) refers to assigning the server s and each peer v a weight (d_s and d_v), and minimizing the sum of d_v and d_s multiplied by their uplink capacities c_v and c_s, subject to Inequality (VII.14), which states that the aggregate weight of any tree t must be greater than its own resilience index $R(t)$, defined in Eq. (VII.4).

[1]This can be achieved by setting $k = |\mathcal{T}| = (|\mathcal{V}| + 1)^{(|\mathcal{V}|-1)}$, the size of the set $\mathcal{T}$

Although there exists exponential number of trees in $\mathcal{T}$, if we can find a separation oracle able to check whether constraint (VII.14) is met in polynomial time, then the dual problem (VII.13) is solvable in polynomial time, hence the primal problem.

Recall that, we proposed a primal-dual algorithm whose idea is as follows. Initially, we assign a weight to each peer $v \in \mathcal{V}$, as well as the server s. In each iteration of the algorithm, we find a tree violating constraint (VII.14), where its aggregate weight is smaller than its resilience index, and assign traffic to this tree whose rate saturates the peer with the bottleneck uplink capacity. Then we update the weight of the server and peers based on the uplink bandwidth they contribute. The algorithm terminates when all trees satisfy the constraint (VII.14).

The practicability of this algorithm is challenged in many fronts. It functions in a centralized fashion, and requires traffic scaling to deliver a feasible rate assignment to all peers. Furthermore, it is not designed to accommodate peer churning, as any peer joining/leaving will cause the algorithm to tear down the existing distribution structures and start from scratch again.

However, this algorithm constitutes the methodological foundation of our practical solution. The weight of each peer is an asymptotic function of the uploading traffic it carries. Therefore, choosing the tree with smaller aggregate weight equals to finding the route with lighter traffic. The value comparison of the aggregate weight and resilience index reflects the tradeoff between traffic load and resilience when choosing the best tree. In what follows, we will show how *PRW*, the evaluation metric of a peer, is derived from the weight update function, the key step of this algorithm.

We finally summarize notations appeared in this section of dissertation in Tab. VII.2.

Notation	Definition
s	server node
$\mathcal{V} = \{v\}$	peers
b_v	outbound bandwidth of peer v
τ_v	age of v
T_v	maximum lifetime of v
$\mathcal{T} = \{t\}$	set of overlay multicast trees
m	mesh structure
r, R	resilience index, e.g. r_v, $R(t)$, $R_t(v)$, $R_{\mathcal{T}}(v)$, $R_m(v)$
$p_t(v)$	the parent of v in the overlay tree t
$f(t)$	data flow over tree t
$x(t)$	0-1 variable indicating the selection of t in multi-tree structure
$\phi(t)$	ratio of t's contribution to the receiving rate in the multi-tree struture
$\mathcal{P}_m(v)$	set of parents of peer v in the mesh structure m
$\phi_{p \to v}(m)$	ratio of p's contribution to the receiving rate of v in the mesh structure
d_s, d_v	weight of server s and peer v

Table VII.2: Notations Table

CHAPTER VIII

PEER EVALUATION METRIC

In this section, we first present *PRW*, an evaluation metric to assess a peer's healthiness in terms of resource abundance and prospect longevity. We then introduce how *PRW* can be used to guide peer selection in various P2P distribution structures, and analyze the overhead involved. Finally, we discuss practical issues.

Metric

Given a peer v, we denote $R(v)$ as the resilience index of v. Depending on the type of P2P distribution structure v resides in, $R(v)$ could be $R_t(v)$ (Eq. (VII.3)) in the single-tree structure, $R_{\mathcal{T}}(v)$ (Eq. (VII.5)) in the multi-tree structure, and $R_m(v)$ (Eq. (VII.7)) in the mesh structure. Let $\mathcal{C}(v)$ be the set containing all children of v, and $f_{v\to c}$ the flow rate from v to each of its children c ($c \in \mathcal{C}(v)$). Then we define the *PRW* for peer v as follows:

$$w_v = \frac{\prod_{c\in\mathcal{C}(v)}(1+\rho\frac{f_{v\to c}}{b_v})}{R(v)} \tag{VIII.1}$$

The numerator of Eq. (VIII.1) refers to d_v, i.e., the weight of v defined in problem (VII.13). Designed to reflect the availability of a peer's uplink capacity, it increases in a super-linear fashion as the traffic load of v increases. ρ is a step size controlling the speed of growth. As such, when a new peer chooses among a group of old peers as its parent and available uplink capacity is the sole concern, it should give priority to the one with smaller weight.

Also in Eq. (VIII.1), this weight is normalized by $R(v)$, the resilience index of v. This is originated from Inequality (VII.14) (the necessary condition for problem (VII.13) to reach optimality), which is the aggregate weight of any tree to be greater than its own resilience index. Based on this idea, our primal-dual algorithm in Chapter IV (General Topology Model) gradually reaches optimality by repeatedly feeding traffic to the trees violating this constraint. Hence, the definition of w_v can be regarded as our original algorithm decomposed from the level of a complete tree into the level of a single peer. Intuitively, w_v balances the factors of resource availability and resilience. When the *PRW* of two candidate peers have the same numerator (indicating the same resource availability), the one with the greater resilience index produces smaller *PRW*, hence should be recommend. Likewise, when two peers have the same resilience index, the one with the smaller numerator wins.

In addition, we introduce an auxiliary metric called congestion indicator defined as follows.

$$\sigma_v = \frac{\sum_{c \in \mathcal{C}(v)} f_{v \to c}}{b_v} \tag{VIII.2}$$

Evidently, peer v still has uplink bandwidth available if $\sigma_v < 1$. Otherwise, admission control will have to be enforced to reject new requests since v can no longer serve another peer. Also it is easy to see that the definitions of w_v and σ_v fit the server s as well.

A notable characteristic of w_v is that it best measures a peer already existing in the P2P distribution structure. If v is an isolated peer with no parent(s) or children, then according to Eq. (VIII.1), its *PRW* is merely $1/r_v$, which carries little meaning. Therefore, we note that *PRW* fits best into the online peer selection algorithm, in which a P2P distribution structure grows in an incremental fashion. In what follows, we discuss how *PRW* coordinates and guides two key actions, *peer joining* and *leaving*.

Peer Selection Algorithm

We consider the set $\mathcal{V}$ which contains all existing peers. Together with the server s, these peers form a P2P distribution structure in the form of single-tree t, or multi-tree $\mathcal{T}$, or mesh m shown in Fig. VII.1.

When a new peer v_{new} joins the P2P structure, v_{new} needs to choose its parent(s) among all peers in $\mathcal{V}$ and the server s. In order to reach the best decision, v_{new} needs the following information for a peer $v \in \mathcal{V}$: its uplink capacity b_v, resilience index $R(v)$, *PRW* w_v, and congestion indicator σ_v. We also note that v_{new} treats server s the same way as all peers in $\mathcal{V}$. In other words, s is simply considered a peer whose resilience index is always 1.

Single Tree

In the case of single tree t, v_{new} first filters out the peers whose remaining uplink bandwidth is insufficient to support the streaming rate of t, i.e., $(1 - \sigma_v)b_v < f(t)$. If no candidate peers remain after the filtering stage, then v_{new} has to be rejected. Otherwise, the peer with the smallest *PRW* is chosen as the parent of v_{new}.

When an existing peer, say v_{old}, leaves the tree, v_{old} should notify its parent to reduce its *PRW* and congestion indicator. v_{old} should notify each of its children to find new parent by rejoining the tree as a new peer, whose procedure is introduced above. It is not unusual to witness sudden peer deaths due to machine failure or loss of network connection. We note that the children of the failed peer can quickly detect such events by noticing the disappeared data stream and start looking for new parent. On the other hand, the parent of the failed peer must depend on other means, such as the heartbeat messages from its children, to detect its failure.

Multiple Trees

In the case of multiple trees, v_{new} needs to join a group of existing trees, whose rates add up to $f(\mathcal{T})$, the streaming rate of the multi-tree structure $\mathcal{T}$. In multi-tree solution, the number of trees k is often predetermined (as outlined in the constraint (VII.12) of problem (VII.9)), and so is the streaming rate of each tree, e.g., assigning equal rate $f(\mathcal{T})/k$ to all trees. Therefore, v_{new} needs to join these trees in the same fashion as joining a single tree: filter out peer with insufficient uplink capacity, then choose the one with the smallest *PRW*.

Another issue worth noting is that preferring small *PRW* promotes the choice of resource-abundant peers, which often have enough uplink bandwidth to support v_{new} in multiple trees. On the other hand, doing so might invalidate an initial design objective of many multi-tree algorithms, which is to diversify parent selection for the sake of reliability. Here, we stress that the main purpose of our study is to propose an evaluation metric as a reference to serve the peer selection of existing algorithms, not modify them. Therefore, we leave the decision to the algorithms using our metric.

Nevertheless, we introduce a simple solution, in which v_{new} chooses k parents in an iterative fashion. In each round, it picks the peer whose *PRW*, upon selected as parent, would introduce the minimum increment. According to Eq. (VIII.1), the increment is $w_v \rho f_{v \to c}/b_v$. After updating the new parent's *PRW*, v_{new} moves on to the next round. This method guarantees parent selection which results in the minimum aggregate increment of *PRW* across all peers. Also if during the streaming, a number of v's parents leave, say k', then v will find another k' parents following the same procedure above.

Mesh

In mesh-based P2P streaming solutions, v_{new} usually connects to a limited number of parents, and employs a receiver-driven approach to request different packets of the

stream from each of its parents. This approach enjoys a greater flexibility than the multi-tree approach, since it only aims to make the total receiving rate of a peer higher than or equal to the streaming rate $f(m)$, whereas the flow rate of a specific parent-child pair is allowed to change constantly. Under this circumstance, it is hard and often unnecessary to find an optimal parent selection by any measure, such as minimum aggregate increment of *PRW*.

Same to the multi-tree case, there exists many potential parent selection solutions to which *PRW* can be of valuable assistance. Here, we propose a simple solution which is to find the k feasible peers with the smallest *PRW*. This can be achieved by sorting all peers by the ascending order of their *PRW*, then moving a window of k peers from the leftmost position to the right one by one, until the k peers in the current window have enough aggregate available uplink bandwidth to support the streaming rate $f(m)$. Also if during the streaming, a number of v's parents leave, say k', then we will find another k' parents with enough aggregate uplink bandwidth available to make up for loss of v. The procedure is the same as above.

Updating Overhead and Bootstrapping

We now analyze the messaging overhead of our algorithm. Across all P2P distribution structures, when v_{new} finds a parent p, changes occur to congestion indicator and *PRW* of p, and resilience index and *PRW* of v_{new}. The same changes occur to p when one of its children leaves. In the mesh structure, when a parent-chid pair $p \rightarrow v$ changes its flow rate, the same change happens as if v finds p as a new parent. Also the resilience factor r_v (Eq. (VII.1)), which basically reflects the age of peer v, keeps growing as time proceeds. This causes change of the resilience index of v as well. Furthermore, the above changes will propagate down to the descendants of p and v.

However, from the definitions of these values (Eq. (VII.7), (VIII.1) and (VIII.2)), we can see that a peer can compute them by learning the congestion indicator and resilience index from its own parent(s), and keeping updated of its own resilience factor. Therefore, the updating overhead of these values can be all absorbed by piggybacking on the data packets flown from parents to children.

What still remains a challenge is bootstrapping. As pointed out at the beginning of section "Peer Selection Algorithm" in this chapter, upon joining, v_{new} needs to learn uplink capacity, congestion index, resilience index, and PRW of existing peers before deciding which of them to be its parent(s). To acquire such knowledge, certain form of publish-subscribe service must exist to assist v_{new}.

We now analyze the overhead of running such a publish-subscribe service. To calculate the resilience factor of a peer (defined in Eq. (VII.1)), one only needs to know the age of the peer and the lifetime of the P2P application itself. While the application lifetime is often predetermined and easily accessible from the server, the peer age is the difference of the current time and the joining time of the peer. Since resilience factor is the foundation to calculate resilience index, we note that v_{new} can recreate by itself the resilience index and PRW of each existing peer, should it obtain information of their joining times, uplink capacities, and parent-child relations. The first two pieces of information only need to be updated once upon peer joining. In order to keep the up-to-date parent-child relations, a peer needs to update to the publish-subscribe service each time it finds new parent(s), as well as the streaming rates from their parents. In fact, under the single-tree and multi-tree structures, the operation of streaming rate update can be avoided since the streaming rate (or allocation of streaming rate among trees) is usually predetermined by the server.

We argue that it is reasonable to expect the publish-subscribe service of a P2P streaming system to maintain above information. The updating overhead can be further absorbed into the bootstrapping scheme which is the basic functionality of any

P2P system. One simple solution is to utilize server s as the centralized server for this purpose. Keeping the complete parent-child relations of the entire P2P distribution structure, s can send the above information to an inquiring peer in one condensed message. Each peer notifies s when it changes to some new parent(s). Such updating messages can be further merged with routine heartbeat messages if their scheduled sending times are close enough. Other sophisticated techniques, e.g., DHT, can also be employed, if it is adopted by the P2P application we serve.

Practical Issues

We now discuss various practical issues.

Coordination with Other Metrics

PRW is a single scalar value which is comparable and sortable. This property allows it to easily blend with traditional metrics such as delay and bandwidth availability. We illustrate two ways for *PRW* to coordinate with other metrics.

First, *PRW* can serve as the tie breaker. Consider a delay-sensitive P2P applications which emphasizes on minimizing delay between a parent-child pair or overall delay from server s to any peers. In this case, when multiple candidate peers qualify to have the minimum delay range, we can choose the one with minimum *PRW*, which not only satisfies the delay requirement of the application, but also offers potentially stable service. In a different P2P application, *PRW* can serve as the primary metric and delay as the tie breaker as well.

Second, *PRW* can appear with other metrics in the form of weighted summation, which is often employed in multi-objective optimization techniques. In the above example, we can assign weights to both delay and *PRW* of each peer, and have the

resulting weighted sum comparable and sortable. We can further normalize both metrics by their maximum values to remove the unit difference.

Coexistence with Other Applications

So far, our discussion has been constrained to a single P2P application. However, a peer v can be present at multiple P2P applications, or non-P2P application, all of which share its uplink capacity. We here describe how to modify the definition of w_v to accommodate these situations.

By examining Eq. (VIII.1), we can see that the numerator of w_v denotes the resource availability of v, which applies to all P2P applications which can utilize v's uplink capacity. However, the denominator of w_v is v's resilience index, which is defined individually by different P2P applications on v. As such, we can modify the definition of w_v as follows. Let $\mathcal{X}$ be the set of P2P applications run by v. We denote $\mathcal{C}(v)$ as all children that v has across all applications in $\mathcal{X}$, and $R_x(v)$ as the resilience index for v in the distribution structure belonging to the application $x \in \mathcal{X}$. Then we provide *PRW* for each P2P application x as below. With this definition, the peer selection algorithm, as well as the updating and bootstrapping procedures stay unchanged.

$$w_v^x = \frac{\prod_{c \in \mathcal{C}(v)} (1 + \rho \frac{f_{v \to c}}{b_v})}{R_x(v)}, \forall x \in \mathcal{X} \tag{VIII.3}$$

However, this change does not apply to non-P2P applications, or P2P applications which do not agree with the *PRW* metric. The key issue is not how the resilience index of v should be defined, but how applications compatible with *PRW* share the uplink capacity of v with those not compatible with *PRW*. In this case, a straightforward and realistic solution for v is to split its capacity between these two group of applications and make each portion non-sharable by applications of another group. This can be

easily done by modifying the value of uplink capacity b_v in Eq. (VIII.3). We note that b_v can be changed on the fly, when v changes the percentage of non-P2P applications. Similar to what is discussed in section "Updating Overhead and Bootstrapping" of this chapter, this change will cause the cascading change of PRW for all descendants of v, but can be absorbed into the data traffic flown from parents to children.

CHAPTER IX

PERFORMANCE EVALUATION – PEER EVALUATION METRIC

We also use simulation to evaluate the performance of *PRW* at assisting various P2P algorithms. We start by introducing two traces we use to drive our simulation, followed by the simulation setup, including performance metrics and existing algorithms for the purpose of comparison. Finally, we present the performance results.

Traces

We use two traces, PPLive and MSN video, to drive our simulation. Both traces measured highly popular systems providing live or on-demand video services. They both provide the joining/leaving timestamps of each individual peer. The MSN traces also provides the downlink bandwidth information of each peer.

MSN Video

This trace is provided by the seminal work[16], which studies the profitability of peer-assisted VoD service using a nine-month trace from the MSN Video service. It covers over 520 million streaming requests for more than 59,000 videos. We choose one hour length trace each from two popular video files, *msn-a* which attracted $20,245$ unique peers during this period, and *msn-b* which attracted $165,481$ unique peers during this period.

To infer the uplink capacity of each peer, we follow the strategy by [16], which introduces mapping between measured downlink bandwidth and the uplink capacity based on available DSL/Cable offerings listed in Tab. IX.1.

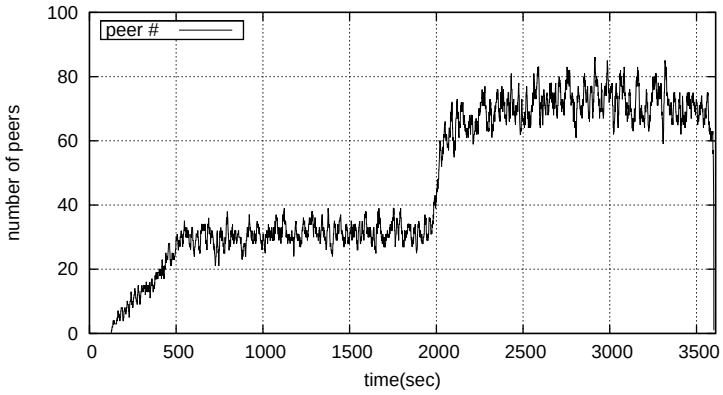


Figure IX.1: Peer Population of MSN Trace

	modem	ISDN	DSL1	DSL2	Cable	Ethernet
downlink	64	256	768	1500	3000	> 3000
uplink	64	256	128	384	768/384	> 768

Table IX.1: Bandwidth Breakdown (kbps)

PPLive

PPLive is the largest commercial peer-to-peer live streaming system to date, which attracts over 100,000 online users during peak times. Following the works by Wang et al.[30], we use PPLive traces gathered by an online crawler[15] that continuously collects information from each channel. We choose traces, *pplive-cctv* and *pplive-ball*, from two popular PPLive channels CCTV3 and DragonBall, from Wednesday Nov

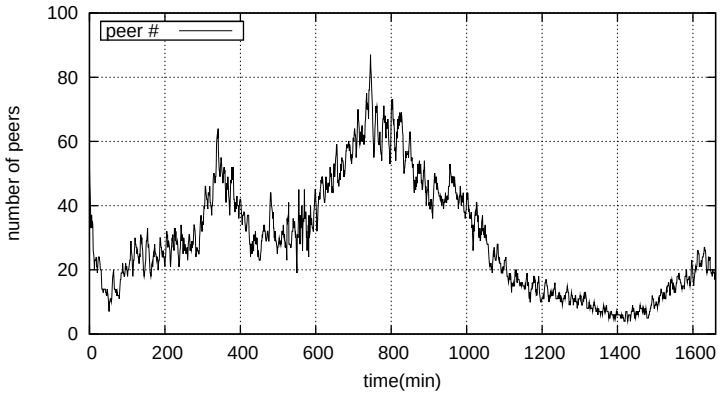


Figure IX.2: Peer Population of PPLive Trace

22 17:40 2006 to Thursday Nov 23 21:30 2006. *pplive-cctv* and *pplive-ball* attracted $52,126$ and $50,131$ unique peers during this period correspondingly. The PPLive traces do not contain any information regarding network bandwidth. So we assign uplink bandwidth to each peer according to the uplink bandwidth proportionment revealed in [16].

Fig. IX.1 and Fig. IX.2 illustrate the peer population of MSN and PPLive traces respectively. They demonstrate the number of peers requesting admission over time.

Setup

Other Algorithms

In Chapter VIII, we introduce *PRW* and how it guides the peer selection algorithms in single-tree, multi-tree, and mesh structures. We denote them as *PRW-single*, *PRW-multi*, and *PRW-mesh*, respectively.

We compare our algorithms with several well-known heuristics and existing algorithms. We consider two heuristics: *age-first* and *bandwidth-first*. They behave the same way as our algorithms in the sense that they also rely on a single scalar value (age or available uplink capacity) to guide the parent selection. As such, we apply the same procedures outlined in Chapter VIII to these two heuristics, except that the goal of minimum *PRW* is replaced by maximum age and maximum available uplink bandwidth.

We also implement two existing algorithms, namely the *hybrid* algorithm proposed in [5] and *ROST* proposed in [27]. In *hybrid*, peers A and B are compared by *DegreeRatio* and *AgeRatio* as the ratio of uplink capacity and age of A and B. If

both ratios are above 1, A has higher priority. If only one of the ratios is above 1, A has priority only if $DegreeRatio > AgeRatio^{-p}$, where p is a parameter. We set $p = 1$ in our experiment, such that bandwidth ratio and age ratio are treated with the same weight. In *ROST* (Reliability-Oriented Switching Tree), peers are measured by BTP (bandwidth-time product) defined as the product of its uplink capacity and age, and those with higher BTPs are switched higher in the tree. We choose *hybrid* and *ROST* among many existing P2P streaming solutions[29, 20, 31], primarily because both of them simultaneously study the heterogeneity of peer age and uplink capacity, and the tradeoff of the two, which are the focus of this dissertation as well.

Performance Metrics

We use two major performance metrics to evaluate the solutions mentioned above. The first is *service disruption*, i.e., under a given streaming rate, the number of disruptions a peer experiences during its lifetime due to the leaving of its ancestor. It is straightforward to count the number of disruptions under the single-tree model. When a peer has multiple parents, we modify it as the fractional loss of receiving rate due to leaving of a an ancestor. This is the same as the definitions of $\phi(t)$ for multi-tree structure and $\phi_{p\to v}(m)$ for mesh structure introduced in Chapter III. In our experiment, we deem all peer leaving events as disruptions to the descendants of the leaving peer.

Derived from *service disruption*, we propose a related metric *data loss ratio*, which is the percentage of data loss experienced by a peer due to service disruption. It is calculated as the sum of each service disruption value multiplied by its duration, then normalized by the peer's lifetime. We set the duration of each disruption as the unit time defined in the traces in which our algorithms run. In other words, we assume that the disrupted peer is able to find itself a new parent within one time

unit. Specifically, the duration is 1 second for MSN video traces, and 1 minute for PPLive traces.

The second major metric is *rejection*, i.e., the number of peers rejected due to insufficient uplink bandwidth availability. The number of rejections can be increased by unwise utilization of peer's uplink bandwidth, which creates considerable remaining bandwidth too fragmented to serve a peer.

Results

In this section, we first compare our algorithm with *ROST* and *hybrid* in single tree case(*ROST* and *hybrid* are single tree algorithms). Then we compare our algorithm with age-first and bandwidth-first heuristics in multi-tree and mesh cases. The streaming rate in the simulations is $384Kbps$, which is the same as the rate of most of the video provided by today's streaming service. The server's bandwidth is set to $1Mbps$.

Single Tree

We start by presenting the performance of various algorithms under the single-tree structure. Fig. IX.3 (a) and (b) show the percentage of peers suffering disruptions in the system over time. Since the numbers of rejected peers are different under different algorithms, we use percentage of peers suffering disruptions instead of absolute value of the number of disruptions. Fig. IX.3 (e) and (f) show the data loss ratio of each algorithm as the sorted view by peer index(the unique sequence number assigned to peers when they join the network). We notice that the curves in these figures do not align well. The reason is revealed in Fig. IX.3 (c) and (d), which display the number of peers rejected over time. Our experiment assumes that the rejected peers do not try to rejoin, hence do not deserve a place in Fig. IX.3 (e) and (f).

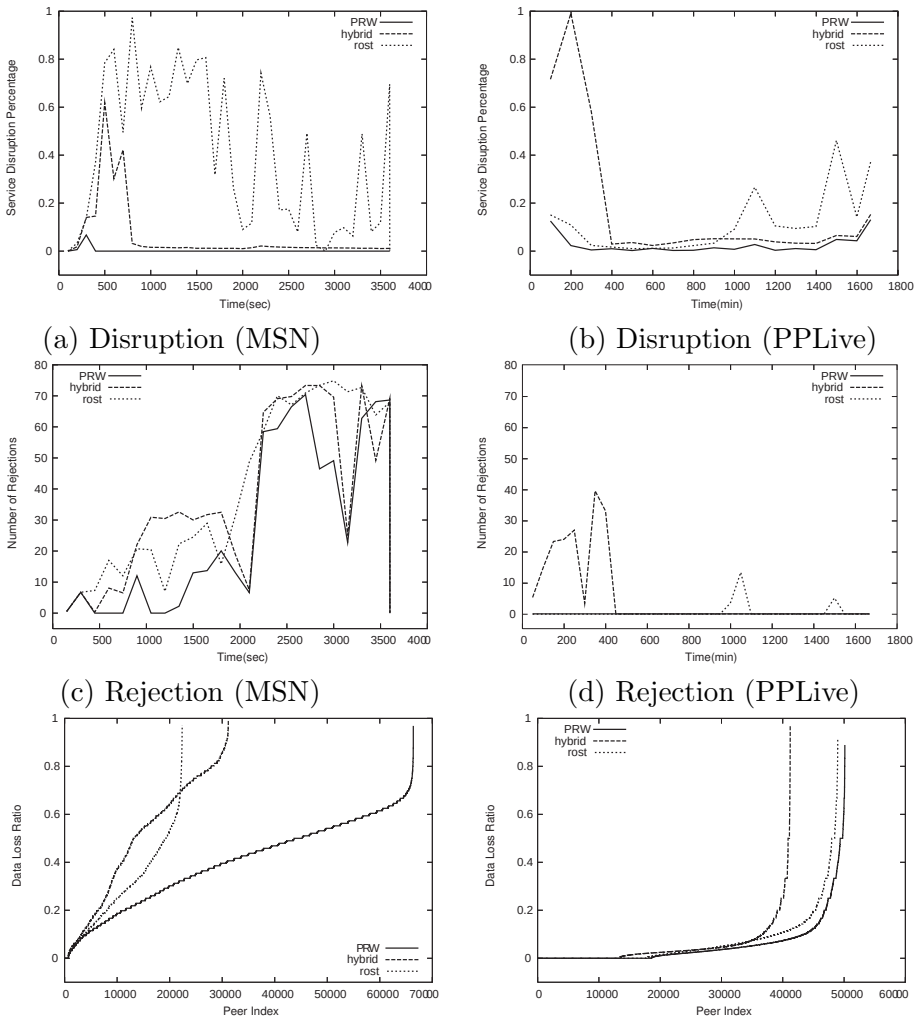
From these figures, we find out that our *PRW*-based algorithm has the lowest peer rejection rate among all algorithms due to its ability to efficiently allocate uplink bandwidth of resource-abundant peers. Especially for the PPLive traces, our *PRW*-based algorithm has no rejection. Also, our algorithm is able to achieve the lowest disruption rate, hence better data loss ratio than other algorithms. This result comes from the fact that *PRW*-based algorithm is able to select peers with high resilience factor(abundant bandwidth resource and prospect longevity) as the parents of other peers.

Multi-tree

Fig. IX.4 shows the performance of *PRW*-based, bandwidth-based and age-based algorithms in multi-tree case. Similar to single tree case, our algorithm achieves the lowest disruption rate. Age-based algorithm is able to achieve similar performance to our algorithm. In multi-tree case, all of these algorithms have no rejection, which is the reason why we do not plot the number of rejections in multi-tree case.

Mesh

Fig. IX.5 shows the performance of *PRW*-based, bandwidth-based and age-based algorithms in mesh case. Again, our *PRW*-based algorithm achieves both the lowest disruption rate and rejection rate. The only exception being bandwidth-based algorithm performs better in MSN traces, while age-based algorithm performs better in PPLive traces.



(a) Disruption (MSN)

(b) Disruption (PPLive)

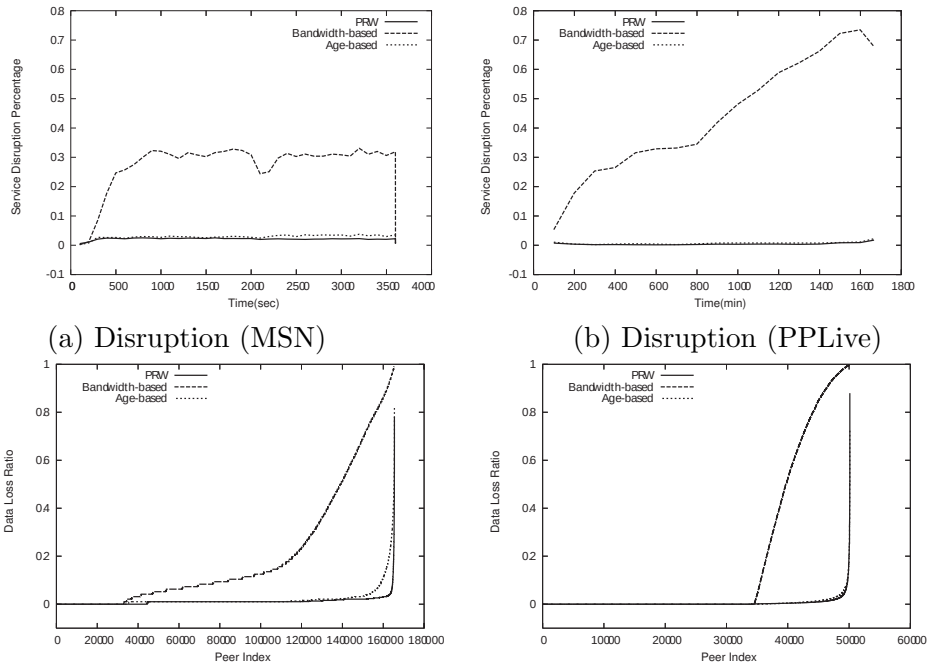
(c) Rejection (MSN)

(d) Rejection (PPLive)

(e) Data Loss Ratio (MSN)

(f) Data Loss Ratio (PPLive)

Figure IX.3: Single Tree



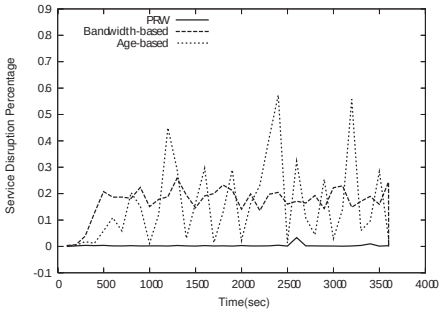
(a) Disruption (MSN)

(b) Disruption (PPLive)

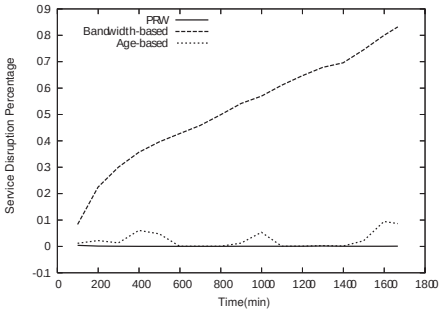
(c) Data Loss Ratio (MSN)

(d) Data Loss Ratio (PPLive)

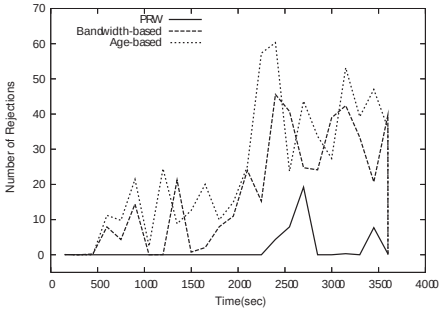
Figure IX.4: Multi-tree



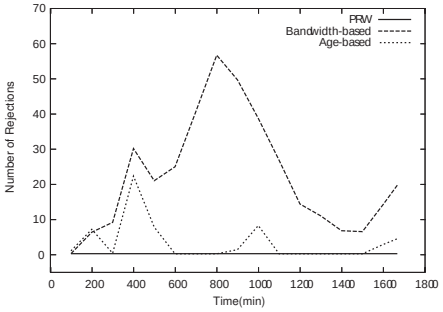
(a) Disruption (MSN)



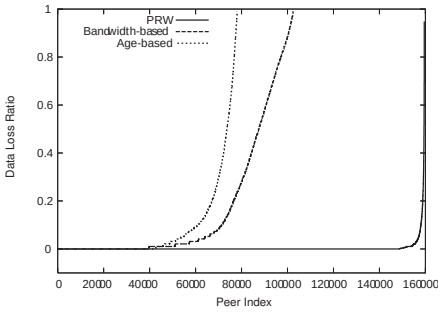
(b) Disruption (PPLive)



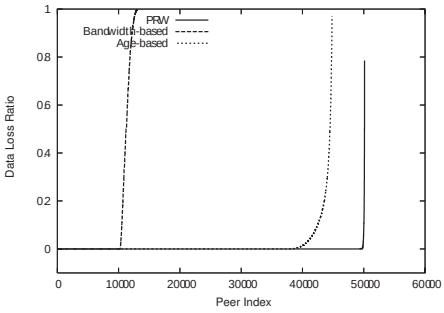
(c) Rejection (MSN)



(d) Rejection (PPLive)



(e) Data Loss Ratio (MSN)



(f) Data Loss Ratio (PPLive)

Figure IX.5: Mesh

CHAPTER X

CONCLUSION

In this dissertation, we propose an optimization framework based on the generalized flow theory. Utilizing the concept of gain factor in this theory, we introduce the resilience factor of peer to model its chance of survival in a probabilistic measure. Based on this idea, an optimization framework is constructed, whose objective is to maximize the P2P network's generalized throughput, the product of raw throughput and combined resilience factors of all peers. We report our findings in this problem domain along several dimensions including network topology, overlay organization, etc.

Based on the optimization framework, we propose an evaluation metric called *PRW* (peer resilient weight) to evaluate a peer's resource and prospect longevity and guide P2P streaming parent selection. Its theoretical background originates from the optimization framework based on the generalized flow theory, which generalizes the classical network flow problem by specifying a gain factor for each link in the network. While the referenced solutions deliver highly varied performances under different traces, solutions guided by *PRW* is able to maintain consistent performance under all traces and achieve both low service disruption and low rejection.

CHAPTER XI

FUTURE WORK

There are two strategies for enhancing stability of P2P streaming system. One is to gradually improve the stability of P2P structure by switching existing parent-child pairs to move up the stable peers. The second strategy is to achieve the same goal by letting each joining peer find the optimal existing peer as its parent, often with the aid of comparable metrics, which this work follow as well. Possible future work could be enhancing stability of P2P streaming system by merging the following two strategies.

***PRW*-Aided Switching**

Here we give an example, illustrating one possible research direction. By incorporating *PRW*-based peer selection with *PRW*-aided switching, we have a straight forward merge of the two strategies. In *ROST*[27], a BTP-based switching was proposed. BTP(Bandwidth-Time Product) is defined as the product of a nodes outbound bandwidth and its age. When a new peer initially enters the network, its BTP is 0. The server is preassigned an infinite BTP, and always remains at the top of the tree. In most cases, the high layers of the tree are occupied and the new peer becomes a low-layer node. As time goes on, a nodes BTP increases at a rate proportional to its bandwidth. If its bandwidth is larger than its parent, then there must be some time point in the future when its BTP exceeds its parent (if the parent does not leave before itself). At that time the algorithm will exchange the roles of these two nodes. Instead of switching based on a peer's BTP, we can use its *PRW* as the metric for

switching criterion, hence we have a *PRW*-aided Switching.

More Peer Evaluation Metrics

Beside *PRW*, we will keep exploring other peer selection metrics and compare their performance under *service disruption*, *data loss ratio*, and *rejection*. These metrics should bear similar distributed features as *PRW*: light-weight, easy to track and easy to update. Peer level is one of the candidates.

If consider *PRW*-aided switching as borrowing peer selection metric and used as peer switching criterion. Inversely, we may consider the status changed with peer switching, as one of the metric to guide peer selection. One goal of peer switching in *ROST* is to move nodes with large BTPs higher in the tree. The level a peer resides in tree can be measured by its distance to root(DTR) or distance to leaf(DTL). Length of a single path(e.g. distance to root) in tree is measured in hops. Distance of a peer to leaf can be obtained by averaging length of all its paths to leaf nodes. In case of multi-tree or mesh, distance to root or leaf can be calculated using a flow rate based weighted average of a peer's paths in different trees.

APPENDIX A

PROOF OF THEOREM 1

The proof of Theorem 1 follows the proofs of the following lemmas. We denote OPT as the optimal value of the problem (IV.1).

Lemma 1: **MultiTrees-General** terminates after at most $|\mathcal{E}| \log_{1+\epsilon} \frac{1+\epsilon}{\beta}$ iterations.

Proof 1 *Let us consider any edge $e \in \mathcal{E}$. Initially, $d_e = \beta$. The last time the length of e is updated, it is on a overlay spanning tree t whose length is less than $R(t) = \sum_{e \in \mathcal{E}} n_e(t) r_e$, and is increased by at most a factor of $1+\epsilon$. Since every augmentation increases the length of some edge by a factor of at least $1+\epsilon$, and $R(t) \leq |V|$, the number of possible augmentations is at most $|V| \log_{1+\epsilon} \frac{1+\epsilon}{\beta}$.*

Lemma 2: Scaling the final flow by $\log_{1+\epsilon} \frac{(1+\epsilon)|V|}{\beta}$ yields a feasible primal solution.

Proof 2 *In the ith iteration of the algorithm, the total flow on an edge $e \in \mathcal{E}$ increases by a fraction $0 \leq \gamma^{(i)} \leq 1$ of its capacity. Its length d_e is multiplied by $1+\epsilon\gamma^{(i)}$. Since $(1+\epsilon\gamma^{(i)}) \geq (1+\epsilon)^{\gamma^{(i)}}$ when $0 \leq \gamma^{(i)} \leq 1$, we have $\prod_i (1+\epsilon\gamma^{(i)}) \geq (1+\epsilon)^{\sum_i \gamma^{(i)}}$. Thus, every time the flow on e increases by its capacity, its length d_e increases by a factor of at least $(1+\epsilon)$. Since d_e is initialized as β, and ends up at most $(1+\epsilon)|V|$, its total flow cannot exceed $c_e \log_{1+\epsilon} \frac{(1+\epsilon)|V|}{\beta}$.*

Lemma 3: When $\beta = \frac{[(1+\epsilon)|V|]^{1-1/\epsilon}}{(|V|U)^{1/\epsilon}}$, the final flow scaled by $\log_{1+\epsilon} \frac{(1+\epsilon)|V|}{\beta}$ has a value at least $(1-2\epsilon)$ times OPT. U is the length of the longest unicast route.

Proof 3 *We make the following denotations. Regarding a set of edge length assignments d_e ($e \in \mathcal{E}$), the objective function in problem (IV.3) is $L^{d_e} \triangleq \sum_{e \in \mathcal{E}} c_e \cdot d_e$. t^{d_e} is the minimum overlay spanning tree in terms of $d_e - r_e$. We denote $d(t^{d_e}) \triangleq \sum_{e \in \mathcal{E}} n_e(t^{d_e}) \cdot d_e$ as the length of t^{d_e} in terms of solely d_e.*

The objective of problem (IV.3) is to minimize L^{d_e}, subject to the constraint that $d(t^{d_e}) \geq R(t^{d_e})$. This constraint can be easily satisfied if we scale the length of all edges by $R(t^{d_e})/d(t^{d_e})$. So problem (IV.3) is equivalent to finding a set of edge lengths, such that $\frac{L^{d_e}R(t^{d_e})}{d(t^{d_e})}$ is minimized. Thus the optimal value of problem (IV.3) is $OPT \triangleq \min_{d_e} \frac{L^{d_e}R(t^{d_e})}{d(t^{d_e})}$.

In each iteration of the algorithm, the length of an edge is updated. We use $d_e^{(i)}$ to denote the length of e after the ith iteration. $d_e^{(0)} = \beta$ is the initial weight of d_e. Regarding $d_e^{(i)}$, we simplify the following denotations $L^{d_e^{(i)}}$, $t^{d_e^{(i)}}$ and $d(t^{d_e^{(i)}})$, into $L^{(i)}$, $t^{(i)}$ and $d(t^{(i)})$. We also denote $f^{(i)}$ as the total flow that has been routed after the ith iteration. Then based on the edge length update function (Line 10 in Tab. IV.1), we have

$$\begin{aligned} L^{(i)} &= \sum_{e\in\mathcal{E}} d_e^{(i-1)} \cdot c_e + \epsilon \sum_{e\in t^{(i-1)}} n_e(t^{(i-1)}) d_e^{(i-1)} (f^{(i)} - f^{(i-1)}) \\ &= L^{(i-1)} + \epsilon (f^{(i)} - f^{(i-1)}) d(t^{(i-1)}) \end{aligned}$$

which implies that

$$L^{(i)} \leq L^{(0)} + \epsilon \sum_{j=1}^{i} (f^{(j)} - f^{(j-1)}) d(t^{(j-1)}) \tag{A.1}$$

Now let us consider the length function $d^{(i)-(0)}$, i.e., for each edge $e \in E$, its length is $d_e^{(i)} - d_e^{(0)} \geq 0$, since the length function is monotonically increasing. Thus, we have $L^{(i)-(0)} = L^{(i)} - L^{(0)}$. Since $d^{(i)-(0)}$ and $d^{(i)}$ only differs by the constant β at each edge, $t^{(i)-(0)}$ and $t^{(i)}$ are the same tree. In addition, the length of the tree using $d^{(i)}$ versus $d^{(i)-(0)}$ differs by at most $\beta|V|U$, U being the length of the longest unicast route. Hence

$$OPT \leq \frac{L^{(i)-(0)}R(t^{(i)-(0)})}{d(t^{(i)-(0)})} \leq \frac{(L^{(i)} - L^{(0)})R(t^{(i)})}{d(t^{(i)}) - \beta|V|U}$$

Substituting this bound on $L^{(i)} - L^{(0)}$ in Eq. (A.1) gives

$$\begin{aligned}\frac{d(t^{(i)})}{R(t^{(i)})} &\leq \frac{\beta|V|U}{R(t^{(i)})} + \frac{\epsilon}{OPT}\sum_{j=1}^{i}(f^{(j)} - f^{(j-1)})d(t^{(j-1)}) \\ &\leq \beta|V|U + \frac{\epsilon}{OPT}\sum_{j=1}^{i}(f^{(j)} - f^{(j-1)})d(t^{(j-1)})\end{aligned}$$

since $R(t) \geq 1$.

Observe that, for fixed i, this right hand side is maximized by setting $d(t^{(j)})$ to its maximum possible value, for all $0 \leq j < i$. Let us call this maximum value $d'(t^{(j)})$. Hence

$$\begin{aligned}\frac{d(t^{(i)})}{R(t^{(i)})} &\leq \frac{d'(t^{(i)})}{R(t^{(i)})} \\ &= \beta|V|U + \frac{\epsilon}{OPT}\sum_{j=1}^{i-1}(f^{(j)} - f^{(j-1)})d'(t^{(j-1)}) \\ &\quad + \frac{\epsilon|V|}{OPT}(f^{(i)} - f^{(i-1)})d'(t^{(i-1)}) \\ &= \frac{d'(t^{(i-1)})}{R(t^{(i-1)})}(1 + \frac{\epsilon R(t^{(i)})(f^{(i)} - f^{(i-1)})}{OPT}) \\ &\leq \frac{d'(t^{(i-1)})}{R(t^{(i-1)})}e^{\frac{\epsilon R(t^{(i)})(f^{(i)} - f^{(i-1)})}{OPT}}\end{aligned}$$

Since $d'(t^{(0)})/R(t^{(0)}) \leq \beta|V|U$, this implies that

$$\frac{d(t^{(i)})}{R(t^{(i)})} \leq \beta|V|Ue^{\epsilon f^*/OPT}$$

where $f^* = \sum_{j=0}^{i} R(t^{(i)})(f^{(i)} - f^{(i-1)})$, the objective of problem (IV.3).

The algorithm stops when the value of $d(t^{(i)}) \geq R(t^{(i)})$. Let f^* be the total flow routed, we have,

$$1 \leq \beta|V|Ue^{\epsilon f^*/OPT}$$

Hence,

$$\frac{OPT}{f^*} \leq \frac{\epsilon}{\ln(\frac{1}{\beta(|V|U}))}$$

By **Lemma 2**, $\frac{f^*}{\log_{1+\epsilon}\frac{(1+\epsilon)|V|}{\beta}}$ *is a feasible solution to problem (IV.3). Then the ratio between the optimal value of problem (IV.3) and the result returned by our algorithm is*

$$\begin{aligned} & \frac{OPT}{f^*}\log_{1+\epsilon}\frac{(1+\epsilon)|V|}{\beta} \\ \leq & \frac{\epsilon\log_{1+\epsilon}\frac{(1+\epsilon)|V|}{\beta}}{\ln(\frac{1}{\beta|V|U})} \\ = & \frac{\epsilon\ln\frac{(1+\epsilon)|V|}{\beta}}{\ln(1+\epsilon)\ln(\frac{1}{\beta|V|U})} \end{aligned} \tag{A.2}$$

When $\beta = \frac{[(1+\epsilon)|V|]^{1-1/\epsilon}}{(|V|U)^{1/\epsilon}}$, *the above inequality becomes*

$$(A.2) \leq \frac{\epsilon}{(1-\epsilon)\ln(1+\epsilon)} \leq \frac{\epsilon}{(1-\epsilon)(\epsilon^2-\epsilon/2)} \leq \frac{1}{(1-\epsilon)^2} \leq \frac{1}{1-2\epsilon}$$

Now we are ready to proof **Theorem 1** as follows.

Proof 4 *By* **Lemma 1**, *the algorithm terminates after at most* $|\mathcal{E}|\log_{1+\epsilon}\frac{1+\epsilon}{\beta}$ *rounds, each round containing a minimum spanning tree construction. When* $\beta = \frac{[(1+\epsilon)|V|]^{1-1/\epsilon}}{(|V|U)^{1/\epsilon}}$, *the maximum number of the iterations needed by the algorithm is*

$$\begin{aligned} & |\mathcal{E}|\log_{1+\epsilon}[((1+\epsilon)U)^{1/\epsilon}|V|^{2/\epsilon-1}] \\ \leq & \frac{|\mathcal{E}|}{\epsilon}(1+\log_{1+\epsilon}U+\log_{1+\epsilon}|V|^2) \\ = & \frac{|\mathcal{E}|}{\epsilon}(1+\frac{\log(U|V|^2)}{\log(1+\epsilon)}) \\ \leq & \frac{|\mathcal{E}|}{\epsilon}+\frac{|\mathcal{E}|}{\epsilon^2}\log(U|V|^2) \end{aligned}$$

Therefore, the running time is $O(\frac{|\mathcal{E}|}{\epsilon^2}[\log U + 2\log|V|]\cdot T_{mst})$.

APPENDIX B

PROOF OF THEOREM 2

Proof 5 *We prove that problem (IV.3), the dual problem of (IV.1) is NP-complete.*

The proof is by reduction from the 3-SAT problem. Let F be a 3-SAT formula in conjunctive normal form, where each clause consists of three literals from $\{v_1, \ldots, v_|V|\}$ and $\{\bar{v}_1, \ldots, \bar{v}_|V|\}$. In Fig. B.1, we build an overlay network, in which testing if constraint (IV.4) is violated, i.e., the separation oracle of problem (IV.1), corresponds to satisfying assignment of F.

Besides server s, There are two types of peers in this graph. The first type of peers correspond to literals v_i and $\bar{v}_i$ ($i = 1, \ldots, |V|$). The second type of peers correspond to clauses of F. All peers have the same resilience factor. The overlay network is a complete graph, whose edges are grouped into two subsets, where edges in each subset carry the same length. Fig. B.1 only shows all edges with smaller lengths: they direct from s to all literal nodes, between each pair of literal nodes, and from each literal node to each clause node it appears in.

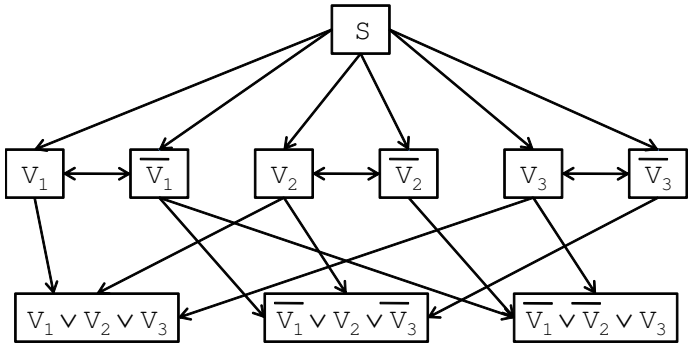


Figure B.1: Proof of Theorem 2

It is obvious that, in terms of the literal set size, there exist an exponential number of minimum spanning trees in this graph, in which all edges have the same length.

However, among these trees, we can decide if there exists a tree with the maximum resilience index only by solving the assignment F.

Since all peers have the same resilience factor, the greatest resilience index a peer can get is via the shortest path from s to itself. This is straightforward for literal nodes since s has a direct edge to each of them. A clause node will have to connect to s either through a literal node directly (two-edge path), or through a pair of literal nodes (three-edge path). Only if F is satisfied can we prove the existence of the minimum spanning tree with the maximum resilience index, in which a two-edge path exists for all clause nodes.

APPENDIX C

PROOF OF THEOREM 3

Proof 6 *Our proof consists of two parts. In the first part, we show that any tree can be reorganized into the collection of a subset of the* $|V|+1$ *trees shown in Fig. V.2 with higher generalized throughput. In the second part, we prove that the tree selection priority the* **MultiTrees-Star** *algorithm follows at choosing among these* $|V|+1$ *trees guarantees optimality.*

In the first part, we examine an arbitrary tree t*, with data flow rate* $f(t)$ *and resilient index* $R(t)$*. We introduce a set* $\mathcal{R}(t)$*, which consists of all non-leaf nodes in* t*. It is obvious that* $S \in \mathcal{R}(t)$*. For each node* $v \in \mathcal{R}(t)$*, we denote* $nc_v(t)$ *as the number of children* v *has in* t*. Then the bandwidth contribution by* v *is* $nc_v(t)f(t)$*. The resilient throughput contributed by* v *is the summation of generalized flow received by all its children. Under the non-concatenation model, this value is* $nc_v(t)f(t)r_v$*, the product of* v*'s bandwidth contribution and its resilient factor. Under the concatenation model, this value is* $nc_v(t)f(t)r_vR_t(v)$*, the product of* v*'s bandwidth contribution, resilient factor, and resilient index. Under both models, the resilient throughput contributed by the server* s *is* $nc_s(t)f(t)$*, since the resilience factor of* s *is 1. As such,* t*'s resilient throughput* $f(t)R(t)$ *can be considered as the summation of resilient throughput contribution by all nodes in* $\mathcal{R}(t)$*.*

We now reorganize t *as follows. For each peer node* $v \in \mathcal{R}(t)$*, we select tree* t_v *in Fig. V.2, where* v *is the relaying node. The flow rate of the selected tree is* $nc_v(t)f(t)/(|V|-1)$*, such that the bandwidth contribution of* v *equals to its contribution in* t*. In this tree, the resilient throughput contributed by* v *is* $nc_v(t)f(t)r_v$ *under both concatenation and non-concatenation models. Compared to the same value in* t*,* v*'s resilient throughput contribution in the new tree is higher in concatenation model,*

and stays the same in non-concatenation model. After conducting the above step for all peer nodes in $\mathcal{R}(t)$, the total bandwidth contribution by s would have reached $(|V|-nc_s(t))f(t)/(|V|-1)$, which is also its resilient throughput contribution. To consume the bandwidth still left at s, we construct tree t_0, the last tree in Fig. V.2, with rate $(nc_s(t)-1)f(t)/(|V|-1)$. Since s must have at least one child to ensure connectivity, this value is no smaller than 0. Then s's bandwidth contribution, as well as its resilient throughput contribution, adding over all trees constructed, is $nc_s(t)f(t)$. This value stays the same as in t under both concatenation and non-concatenation models. Now we can claim that, with the above reorganization, the generalized throughput contribution by each node in $\mathcal{R}(t)$ is greater or equal to its contribution in tree t. Hence collectively, the aggregate generalized throughput of these trees is greater or equal to the generalized throughput of t, with the same bandwidth contribution by each node in $\mathcal{R}(t)$.

As it is now clear that the $|V|+1$ trees shown in Fig. V.2 collectively achieve higher generalized throughput than any other tree, we proceed to the second part of our proof. Each tree t_v except t_0 only consumes the bandwidth of s and the relaying peer v, and has its rate upper bounded by the minimum of $c_{v_i}/(|V|-1)$ and the remaining bandwidth of s. t_0 consumes solely the bandwidth of s. As such, the bandwidth of s becomes bottleneck resource that all trees rely on. We introduce the "gain ratio" for each tree, which is the ratio of its generalized throughput and the bandwidth contribution by s. For all trees except t_0, such value is $1+r_v(|V|-1)$, where r_v is the resilience factor of the relaying peer v. For the last tree t_0, such value is $1/|V|$. The tree selection of **MultiTrees-Star** *is based on the descending order of their gain ratios. It is now clear that the algorithm follows a greedy strategy, in each round the tree with the highest gain ratio is chosen and fed with the maximum achievable rate, until the bandwidth of s is depleted.*

BIBLIOGRAPHY

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffss, NJ, 1993.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of ACM*, 44, 1997.

[3] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *IEEE Conference on Foundation of Computer Science (FOCS)*, 1993.

[4] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1995.

[5] M. Bishop, S. Rao, and K. Sripanidkulchai. Considering priority in overlay multicast protocols under heterogeneous environments. In *Proc. of INFOCOM*, April 2006.

[6] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[7] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, pages 1456–1471, October 2002.

[8] R. Cohen and G. Kaempfer. A unicast-based approach for streaming multicast. In *Proc. of IEEE INFOCOM*, 2001.

[9] Y. Cui, B. Li, and K. Nahrstedt. On achieving optimized capacity utilization in application overlahy networks with multiple competing sessions. In *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, 2004.

[10] N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. 1994.

[11] B. Fortz and M. Thorup. Internet traffic engineering by optimizing ospf weights. In *in Proc. IEEE INFOCOM*, pages 519–528, 2000.

[12] B. Godfrey, S. Shenker, and I. Stoica. Minimizing churn in distributed systems. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 147–158, New York, NY, USA, 2006. ACM.

[13] A. Goel, M. Henzinger, and S. Plotkin. Online throughput-competitive algorithm for multicast routing and admission control. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1998.

[14] M. Guo and M. Ammar. Scalable live video streaming to cooperative clients using time shifting and video patching. In *IEEE INFOCOM*, 2004.

[15] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. A measurement study of a large-scale p2p iptv system. *Multimedia, IEEE Transactions on*, 2007.

[16] C. Huang, J. Li, and K. Ross. Can internet video-on-demand be profitable? In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 133–144, New York, NY, USA, 2007. ACM.

[17] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[18] D. Leonard, Z. Yao, V. Rai, and D. Loguinov. On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks. *IEEE/ACM Transactions on Networking*, 15(3), 2007.

[19] Y. Liu, Y. Guo, and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, pages 18–28, 2008.

[20] N. Magharei and R. Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1415–1423, 2007.

[21] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: An approach to universal topology generation. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications System*, 2001.

[22] V. Padmanabhan, H. Wang, and P. Chou. Resilient peer-to-peer streaming. In *IEEE International Conference on Networking Protocols (ICNP)*, 2003.

[23] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 177–186, New York, NY, USA, 2002. ACM Press.

[24] R. Rahman, M. Meulpolder, D. Hales, J. Pouwelse, and H. Sips. Improving efficiency and fairness in p2p systems with effort-based incentives. In *In Proceedings of ICC*, page 10, 2010.

[25] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application endpoints. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 107–120, New York, NY, USA, 2004. ACM Press.

[26] G. Tan and S. Jarvis. On the reliability of dht-based multicast. In *Proc. of INFOCOM*, 2007.

[27] G. Tan, S. Jarvis, and D. Spooner. Improving the fault resilience of overlay multicast for media streaming. In *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pages 558–567, Washington, DC, USA, 2006. IEEE Computer Society.

[28] E. Tardos and K. Wayne. Simple generalized maximum flow algorithm. In *Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science*, volume 1412, 1998.

[29] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 2–11, Washington, DC, USA, 2006. IEEE Computer Society.

[30] F. Wang, J. Liu, and Y. Xiong. Stable peers: Existence, importance, and application in peer-to-peer live video streaming. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1364–1372, 2008.

[31] F. Wang, Y. Xiong, and J. Liu. mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 49, Washington, DC, USA, 2007. IEEE Computer Society.

[32] B. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[33] K. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. In *ACM Symposium on Theory of Computing*, 1999.

[34] K. Wayne and L. Fleischer. Faster approximation algorithms for generalized flow. In *ACM-SIAM symposium on Discrete algorithms*, 1999.