Maximizing Resilient Throughput in Peer-to-Peer Network

Bo Liu¹, Fan Qiu², Yanchuan Cao², Bin Chang³, Yi Cui², Yuan Xue²

¹ School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
² Department of Electrical Engineering and Computer Science at Vanderbilt University, Nashville, USA
³ YouTube, LLC, San Francisco, USA

E-mail: bo.liu@hust.edu.cn, {fan.qiu, yanchuan.cao}@vanderbilt.edu, binchang@google.com, {yi.cui, yuan.xue}@Vanderbilt.edu

Received May 29, 2011; revised July 5, 2011; accepted July 12, 2011

Abstract

A unique challenge in P2P network is that the peer dynamics (departure or failure) cause unavoidable disruption to the downstream peers. While many works have been dedicated to consider fault resilience in peer selection, little understanding is achieved regarding the solvability and solution complexity of this problem from the optimization perspective. To this end, we propose an optimization framework based on the generalized flow theory. Key concepts introduced by this framework include resilience factor, resilience index, and generalized throughput, which collectively model the peer resilience in a probabilistic measure. Under this framework, we divide the domain of optimal peer selection along several dimensions including network topology, overlay organization, and the definition of resilience factor and generalized flow. Within each sub-problem, we focus on studying the problem complexity and finding optimal solutions. Simulation study is also performed to evaluate the effectiveness of our model and performance of the proposed algorithms.

Keywords: Peer-to-Peer, Resilient, Throughput, Maximizing

1. Introduction

Peer-to-peer (P2P) has been proved a highly cost-effective content distribution solution, where peers self-organize themselves into an overlay network and relay data to each other, thus reducing server load. A central problem in the overlay network construction is peer selection, the strategy a peer employs to select other peer(s) as its parent(s) to receive data from. Peer selections aggregate 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 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.

A significant amount of research has been conducted on this topic with different emphasis. While important heuristics have been proposed such as minimizing depth [1,2], multiple trees [3,4], bandwidth-first, age-first [5], or a hybrid of the two [6], some analytical works have tried to analyze and compare their performances under stochastic framework [7,8] 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, 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 heuristics could be quantitatively evaluated under the same framework.

In this paper, we report our initial research towards this direction. Our optimization framework is based on the generalized flow theory [9]. 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 [10-12], 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 only on peer’s access link.
- Regarding overlay organization, we consider cases where the number of trees interconnecting peers is unlimited or upper-bounded, e.g., single tree.
- Along the dimension of generalized flow definition, we consider concatenation model where the generalized flow delivered to a peer depends on the resilience of all its ancestors, and 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.

The rest of this paper is organized as follows. In Sec. 2, we introduce our optimization framework and formally define key concepts such as generalized flow and resilience index. In Sec. 3, we study the optimal peer selection problem under the general topology, and propose two algorithms employing linear programming techniques. In Sec. 4, we study the same problem under the star topology, and propose two algorithms based on combinatorial optimization techniques. Sec. 5 presents evaluation results. Finally, we discuss related works in Sec. 6 and conclude in Sec. 7.

2. Framework Overview

2.1. Network Model

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

2.2. Overlay Organization

To transfer data among peers, the simplest and most straightforward 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. We denote the tree set as \( T = \{t\} \), where each tree \( t \in T \) covers all peers and has a single rate \( f(t) \).

2.3. Resilience Factor and Generalized Flow

We assign a resilience factor \( r_v (0 < r_v < 1) \) 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. \( F(t) \), 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 \( \tau \). If we denote \( r_v = \Pr(T > \tau^*) \), where \( \tau^* \) is a fixed time point in the future, then it represents the chance of survival for \( v \) until \( \tau^* \).

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

2.3.1. Concatenation Model

For each peer \( v \) in tree \( t \), there is a path from the server \( s \)
to \( v \), denoted as:
\[
P_t(v) : s \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k \rightarrow v
\]

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 \( t \):
\[
R_t(v) = \prod_{i=1}^{k} r_{v_i}
\]

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:
\[
R(t) = \sum_{v \in V} R_t(v)
\]

Since \( R(v) \leq 1 \), 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.
\[
f_g(t) = f(t)R(t)
\]

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 depart.

2.3.2. 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.
\[
R_t(v) = r_{v_k}
\]

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.

Finally, in Table 1, we summarize findings when exploring along the three dimensions outlined in this section. Of the eight sub-problems, we find four of them polynomially solvable and present the optimal solutions. Of the four NP-hard problems, we are able to find a \( \Omega(\log) \)-approximation algorithm, and only find heuristics to the other three. We also summarize notations appeared in this paper in Table 2.

3. General Topology Model

In this section, we present our study on optimal generalized throughput under the general topology model, as shown in Figure 1.

3.1. Multiple Tress

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 Sec. 2, we formulate it into the following linear programming (LP) problem.

\[
\begin{align*}
\text{maximize} & \quad \sum_{t \in T} f(t)R(t) \\
\text{subject to} & \quad \sum_{t \in T} n_e(t)f(t) \leq c_e, \forall e \in \mathcal{E} \\
\end{align*}
\]

The objective of problem (5) is to maximize the generalized throughput (defined in Equation (3)) of all trees. Inequality (6) 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 (5) is that its number of variables is exponential to the size of the P2P network. Based on Cayley’s theorem [14], 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 \(|T|\) variables but exponential constraints.

<table>
<thead>
<tr>
<th>Table 1. Summary of findings.</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Topology</td>
</tr>
<tr>
<td>Star Topology</td>
</tr>
<tr>
<td>Multiple Trees (Concatenation)</td>
</tr>
<tr>
<td>Multiple Trees (Non-Concatenation)</td>
</tr>
<tr>
<td>Single Tree (Concatenation)</td>
</tr>
<tr>
<td>Single Tree (Non-Concatenation)</td>
</tr>
</tbody>
</table>
Table 2. Notations table.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G = (N, \varepsilon)$</td>
<td>Physical Network</td>
</tr>
<tr>
<td>$G = (V, L)$</td>
<td>Overlay Network</td>
</tr>
<tr>
<td>$s$</td>
<td>Server node</td>
</tr>
<tr>
<td>$\varepsilon = {e}$</td>
<td>Physical layer edges</td>
</tr>
<tr>
<td>$L = {l}$</td>
<td>Overlay layer links</td>
</tr>
<tr>
<td>$V = {v}$</td>
<td>Overlay nodes</td>
</tr>
<tr>
<td>$r, R$</td>
<td>Resilience index e.g. $r_e, R(t), R_v$</td>
</tr>
<tr>
<td>$T = {t}$</td>
<td>Overlay multicast trees</td>
</tr>
<tr>
<td>$f(t)$</td>
<td>Data flow over tree $t$</td>
</tr>
<tr>
<td>$c$</td>
<td>Bandwidth constraint e.g. $c_v, c_e$</td>
</tr>
<tr>
<td>$d_e$</td>
<td>Price of edge $e$</td>
</tr>
<tr>
<td>$P_{s, v}(t)$</td>
<td>Overlay routing path between $s$ and $v$ in overlay tree $t$</td>
</tr>
</tbody>
</table>

Figure 1. General topology model.

\[
\text{maximize } \sum_{e \in \varepsilon} c_e d_e 
\quad \text{(7)}
\]

subject to \[
\sum_{e \in \varepsilon} n_e(t) d_e \geq R(t), \forall t \in T
\quad \text{(8)}
\]

where $R(t)$ is the sum of resilience factors of all links in $t$. This allows us to reformulate Inequality (8) into the following.

\[
\sum_{e \in \varepsilon} n_e(t) d_e \leq \sum_{e \in \varepsilon} n_e(t) r_e, \forall t \in T
\]

It is now clear that the separation oracle is a minimum spanning tree algorithm that solves the cost on each link $e$ as $(d_e - r_e)$. Constraint (8) 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 an $\varepsilon$-approximate solution returning a result at least $(1 - \varepsilon)$ times the maximum value, for arbitrary error parameter $\varepsilon > 0$.

Table 3 shows the MultiTrees-General algorithm. It solves the primal and dual problems in an iterative fashion. It sets initial length to $\beta$ all links in $E$. In each iteration, it finds the minimum spanning tree $T_{mst}$ based on the cost $(d_e - r_e)$, and route traffic over $T_{mst}$. Based on the traffic increment, the length of each tree is updated as defined in line 10. Finally, the algorithm terminates when constraint (8) 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 $l_{\text{max}}$, the algorithm is guaranteed to return a feasible solution. We summarize the property of this algorithm in Theorem 1.

**Theorem 1:** Under the non-concatenation model, when

\[
\beta = \left(\frac{1 + \varepsilon}{\varepsilon}\right)^{\frac{1 - \alpha}{2}}
\]

the MultiTrees-General algorithm returns the solution at least $(1 - 2 \varepsilon)$ times the optimal result of problem (5), with running time

\[
O\left(\frac{1}{\varepsilon^2} \left[ \log U + 2 \log |P| \right] \cdot T_{\text{max}} \right)
\]

where $U$ is the length of the longest uni-cast route and $T_{\text{max}}$ is the running time of the minimum spanning tree algorithm.
### Table 3. Finding multiple trees under general topology model.

<table>
<thead>
<tr>
<th>MultiTrees-General ((E, T))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( \forall e \in E, d_e, \leftarrow \beta c_t, l_e \leftarrow 0 )</td>
</tr>
<tr>
<td>2 ( f(t) \leftarrow 0, t \in T )</td>
</tr>
<tr>
<td>3 loop</td>
</tr>
<tr>
<td>4 ( t^* \leftarrow ) minimum overlay spanning tree in ( T ) using ((d_e - r_e))</td>
</tr>
<tr>
<td>5 minlen ( \leftarrow \sum_{t \in T} n_t(t^*) ) ((d_e - r_e))</td>
</tr>
<tr>
<td>6 if minlen ( \geq 0 )</td>
</tr>
<tr>
<td>7 return</td>
</tr>
<tr>
<td>8 ( c \leftarrow \min \frac{c_t}{n_t(t)} )</td>
</tr>
<tr>
<td>9 ( f(t^<em>) \leftarrow f(t^</em>) + c )</td>
</tr>
<tr>
<td>10 ( \forall e \in E, d_e \leftarrow d_e + \left(1 + \rho \frac{n_t(t)}{c_t} \right) l_e \leftarrow l_e + \frac{n_t(t)}{c_t} )</td>
</tr>
<tr>
<td>11 end loop</td>
</tr>
<tr>
<td>12 ( l_{\text{max}} \leftarrow \max_{t \in T} l_t )</td>
</tr>
<tr>
<td>13 ( \forall t \in T, f(t) \leftarrow f(t)/l_{\text{max}} )</td>
</tr>
</tbody>
</table>

Now we turn to the concatenation model, where the resilience index is defined in Equation (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, 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.

**Theorem 2**: Under the concatenation model, the MultiTrees-General problem (5) is NP-hard.

### 3.2. 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 (5) into the following integer programming problem.

\[
\max \sum_{t \in T} f(t) R(t) x(t) \quad (10)
\]

\[
s_J \sum_{t \in T} n_t(t) f(t) x(t) \leq c_e, \forall e \in E \quad (11)
\]

\[
\sum_{t \in T} x(t) = k \quad (12)
\]

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

Problem (10) introduces a 0-1 variable \(x(t)\), and \(k\), the upper limit on the number of trees. This constraint is enforced by Equation (12). 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 [15] 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. The details of the algorithm can be found in Table 4.

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 (5).

### 4. STAR Topology Model

The algorithms presented in Sec. 3 rely on linear programming technique, and operate on both overlay network \(L\) and physical network whose measurement can be expensive. They require complete knowledge on the physical network, i.e., the capacity of any physical link \(e \in E\), if it appears in 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 outgoing bandwidth

### Table 4. Finding \(k\) trees under general topology model.

<table>
<thead>
<tr>
<th>k-Tree ((E, T))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( \forall e \in E, d_e, \leftarrow \beta c_t, l_e \leftarrow 0 )</td>
</tr>
<tr>
<td>2 for ( i = 1 ) to ( k ) do</td>
</tr>
<tr>
<td>3 ( t, ) minimum overlay spanning tree in ( T ) using ((d_e - r_e))</td>
</tr>
<tr>
<td>4 ( f(t) \leftarrow 1 )</td>
</tr>
<tr>
<td>5 ( \forall e \in E, d_e \leftarrow d_e + \left(1 + \rho \frac{n_t(t)}{c_t} \right) l_e \leftarrow l_e + \frac{n_t(t)}{c_t} )</td>
</tr>
<tr>
<td>6 ( l_{\text{max}} \leftarrow \max_{t \in T} l_t )</td>
</tr>
<tr>
<td>7 for ( i = 1 ) to ( k ) do</td>
</tr>
<tr>
<td>8 ( f(t) \leftarrow f(t)/l_{\text{max}} )</td>
</tr>
</tbody>
</table>
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 into a star network, whose central hub represents the Internet cloud reaching out to all peers, as shown in Figure 2. In this section, we study the same set of problems under such a special topology.

4.1. Multiple Trees

We again start with the case of multiple trees. To simplify the illustration, we remove notations associated with the general network \( \varepsilon \). Instead, we introduce notations \( c_v \) to denote outbound bandwidth of peer \( v \in V \), \( c_s \) to denote outbound 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:

\[
\text{Maximize } \sum_{t \in T} f(t) R(t) \\
\sum_{t \in T} \sum_{v \in V} n_v(t) f(t) \leq c_v, \forall v \in V \\
\sum_{t \in T} n_s(t) f(t) \leq c_s \\
f(t) \geq 0, t \in T
\]

Inequalities (14) and (15) refer to the capacity constraint. In fact, problem (13) is only a special case of the problem (5), 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.

Table 5 shows the MultiTrees-Star algorithm. It constructs at most \(|V| + 1\) trees in the order shown in Figure 3.

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.

**Theorem 3:** MultiTrees-Star algorithm returns the optimal result of problem (13).

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., Equation (1) for concatenation model and Equation (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.

4.2. 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 (13) in the same fashion we defined problem (10) under the general topology model.

\[
\text{Maximize} \sum_{t \in T} f(t) R(t) x(t) \quad (17)
\]
\[
\text{s.t.} \sum_{t \in T} n_i(t) f(t) x(t) \leq c_i, v \in V \quad (18)
\]
\[
\sum_{t \in T} n_i(t) f(t) x(t) \leq c_i \quad (19)
\]
\[
\sum_{t \in T} x(t) = k \quad (20)
\]
\[
f(t) \geq 0, x(t) = \{0,1\}, t \in T \quad (21)
\]

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. Table 6 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 Equation (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 \( t_s \) shown in Figure 3, which has the maximum resilience index \(|V|\). Since in each iteration of SingleTree-Star, at least one peer’s outbound degree will increase.

<table>
<thead>
<tr>
<th>Table 6. Finding single tree under star topology.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxRate-Star ((s, V))</td>
</tr>
<tr>
<td>1 sort ( V ) into ( V_s, \ldots, V_1 ) in descending order of resilience</td>
</tr>
<tr>
<td>2 ( f \leftarrow c_s )</td>
</tr>
<tr>
<td>3 find ( k ) such that ( c_s, f \geq c_i )</td>
</tr>
<tr>
<td>4 do</td>
</tr>
<tr>
<td>5 ( i \leftarrow 0, \text{sum} \leftarrow 0 )</td>
</tr>
<tr>
<td>6 while \text{sum} &lt; (</td>
</tr>
<tr>
<td>7 ( \text{sum} \leftarrow \text{sum} + \left\lfloor \frac{c_i}{f} \right\rfloor, i \leftarrow i + 1 )</td>
</tr>
<tr>
<td>8 if ( \text{sum} &lt;</td>
</tr>
<tr>
<td>9 ( f \leftarrow c_s, k \leftarrow k + 1 )</td>
</tr>
<tr>
<td>10 while \text{sum} &lt; (</td>
</tr>
<tr>
<td>11 return ( f )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MostResilientTree-Star ((s, V))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 sort ( V ) into ( v_{s}, \ldots, v_{</td>
</tr>
<tr>
<td>2 enqueue ( s, v_1, \ldots, v_{</td>
</tr>
<tr>
<td>3 enqueue ( v_{t} ), \ldots, ( v_{</td>
</tr>
<tr>
<td>4 ( \text{sum} \leftarrow 0 )</td>
</tr>
<tr>
<td>5 while ( C \neq \emptyset )</td>
</tr>
<tr>
<td>6 dequeue ( v_{\text{parent}} ) from ( P )</td>
</tr>
<tr>
<td>7 repeat ( \left\lfloor \frac{c_i}{f} \right\rfloor ) times</td>
</tr>
<tr>
<td>8 dequeue ( v_{\text{child}} ) from ( C )</td>
</tr>
<tr>
<td>9 make ( v_{\text{parent}} ) the parent of ( v_{\text{child}} )</td>
</tr>
<tr>
<td>10 ( \text{sum} \leftarrow \text{sum} + c_{\text{child}} )</td>
</tr>
<tr>
<td>11 return { ( f ) * \text{sum} ), resulting tree }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SingleTree-Star ((s, V))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( f \leftarrow ) MaxRate-Star ((s, V))</td>
</tr>
<tr>
<td>2 {max, tree} \leftarrow ) MostResilientTree-Star ((s, V))</td>
</tr>
<tr>
<td>3 while ( f*</td>
</tr>
<tr>
<td>4 ( f_{\text{new}} \leftarrow c_i / \left\lfloor \frac{c_i}{f} \right\rfloor + 1 )</td>
</tr>
<tr>
<td>5 for ( i = 1 ) to (</td>
</tr>
<tr>
<td>6 if ( f_{\text{new}} &lt; c_i / \left\lfloor \frac{c_i}{f} \right\rfloor + 1 )</td>
</tr>
<tr>
<td>7 ( f_{\text{new}} \leftarrow c_i / \left\lfloor \frac{c_i}{f} \right\rfloor + 1 )</td>
</tr>
<tr>
<td>8 ( f \leftarrow f_{\text{new}} )</td>
</tr>
<tr>
<td>9 {r, tree} \leftarrow ) MostResilientTree-Star ((s, V))</td>
</tr>
<tr>
<td>10 if ( r &gt; \text{max} )</td>
</tr>
<tr>
<td>11 \text{max} \leftarrow r, \text{tree} \leftarrow \text{tree}</td>
</tr>
<tr>
<td>12 return tree*</td>
</tr>
</tbody>
</table>
crease by 1, the number of iterations is bounded by \(O\left(\left|V^{\top}\right|\right)\). Combined with the linear running time of **MostResilientTree-Star**, its overall running time is \(O\left(\left|V^{\top}\right|\right)\). 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 (17) when \(k = 1\), with running time \(O\left(\left|V^{\top}\right|\right)\).

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 factors, which is given in Equation (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 show 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.

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 **Table 6** can be reused with no modification.

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 paper 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 \(c_i\) with \(\min\{c_s, c_{in}\}\) in **Tables 5 and 6**.

### 4.3. Discussions

We finally discuss the implement-ability of algorithms presented in this paper. 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 in **Table 4** 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. While it is reasonable to expect the server to keep the up-to-date information of the peers it serves, the distributed versions of these algorithms, if possible, are often desirable and constitutes the future direction of our research.

### 5. Evaluations

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.

#### 5.1. 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 by the Boston BRITE topology generator using the Waxman model. 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 100 Kbps to 1000 Kbps. The second topology follows the star configuration outlined in **Figure 2**.

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 begin-
ning 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.

5.2. 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 metric Volume, 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 Figure 4, 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 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 assigns priorities to the ones with higher outbound bandwidth. 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 Figure 5, performance ordering of these algorithms under different lifetime distributions are consistent in both figures.

5.3. 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 star topology and concatenation model, and normalize their results with the one achieved by the optimal MultiTree-Star algorithm. In Figure 6, we observe that all of them are able to maintain the performance ratio from 0.1 to 0.6 under different mean outbound bandwidths, mean lifetime, and lifetime distributions.

In Figure 7, 8, and 9, we display the sorted per-node generalized throughput and Volume for MultiTrees-Star, SingleTree-Star, and the two heuristics.
Figure 5. Performances of MultiTrees-Star and two heuristics under concatenation model (mean lifetime = 1500 s). (a) Volume; (b) generalized throughput.

Figure 6. Performances ratio of heuristics to MultiTrees-Star under concatenation model. (a) Outbound bandwidth; (b) lifetime.

Figure 7. Sorted nodes of MultiTrees-Star (mean outbound bandwidth = 100 Kbps). (a) Volume; (b) Generalized throughput.
6. Related Work

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 [1], [2], etc. Several well-known works, such as Bullet [16], SplitStream [3], and CoopNet [4], 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. [5] 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 paper, such as the optimal tree structure exhibited in the multi-tree setting under star network model.

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 [6]. Stochastic network analysis have been employed to study the resilience of DHT based P2P network [7] and decentralized P2P network [8] 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.

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 [17,18] presents the theoretical models for online unicast routing. In
the multicast domain, the work of [19] investigate the case of receivers within a multicast session arriving in batch, and the work in [20] presents a solution for receivers arriving separately. In the past, we have extended the multi-commodity flow theory to maximize the throughput of overlay multicasting [21]. This work is our initial effort to further incorporate fault resilience by introducing generalized flow theory.

Generalized flow has many applications [9], where the gain factors can model physical transformations of a commodity due to leakage, evaporation, breeding, theft, also transformations from one commodity into another as a result of manufacturing, scheduling, or currency exchange, etc. Exiting works have been focused on unicast routing [10,11]. In particular, Wayne et al. [12] present a Dijkstra-variant shortest path algorithm for minimum-cost unicast-based generalized flow problem if all gain factors are below one. It is our finding in this paper 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.

7. Conclusions

In this paper, 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, and propose an optimization framework targeting at maximizing generalized throughput as the product of raw throughput and resilience factors. We report our findings in this problem domain along several dimensions including network topology, overlay organization, and concatenation model.

Our future work will carry along two directions. On the theoretical front, we will study whether improvement space exists for optimal algorithms presented in this paper, such as SingleTree-Star. On the practical front, we will design the lightweight distributed solution of our algorithms, especially under the star topology model. We are also interested to search for simple heuristics and have quantitatively evaluated within our optimization framework.

8. Acknowledgements

This work was supported by NSF award 0643488, Vanderbilt Discovery grant, and a gift from Microsoft Research.

9. References


[13] Y. Cui, Y. Xue and K. Nahrstedt, “Max-min Overlay Multicast: Rate Allocation and Tree Construction,” IEEE International Workshop on Quality of Service, Montreal,

Appendix: 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 (5).

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

Proof: Let us consider any edge $e \in E$. 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 $R(t) = \sum_{e \in t} n_e(t) r_e$, 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 $\min_{i \in E} \frac{(1 + \epsilon)|V|}{\beta}$ yields a feasible primal solution.

Proof: In the $i$th iteration of the algorithm, the total flow on an edge $e \in E$ increases by a factor $0 \leq \gamma^i \leq 1$ of its capacity. Its length $d_e$ is multiplied by $1 + \gamma^i$. Since $(1 + \epsilon) \gamma^i \geq (1 + \epsilon) \gamma^i$ when $0 \leq \gamma^i \leq 1$, we have $\Pi_{i}(1 + \epsilon) = (1 + \epsilon)^{\sum_{i \in E} \gamma^i}$. 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 $\beta$, 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 $\beta = \frac{[1+(1+\epsilon)|V|]^{\frac{1}{1+\epsilon}}}{(|V|U)^\frac{1}{1+\epsilon}}$, 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: We make the following denotations. Regarding a set of edge length assignments $d_e(e \in E)$, the objective function in problem (7) is $L^e = \sum_{e \in E} c_e \cdot d_e \cdot t_e$ is the minimum overlay spanning tree in terms of $d_e = r_e$. We denote $d\left(t \Rightarrow e^i\right) = \sum_{e \in E} n_e(t) \cdot d_e$ as the length of $t \Rightarrow e^i$ in terms of solely $d_e$.

The objective of problem (7) is to minimize $L^e$, subject to the constraint that $d(t \Rightarrow e^i) \geq R(t \Rightarrow e^i)$. This constraint can be easily satisfied if we scale the length of all edges by $R\left(t \Rightarrow e^i\right)/d\left(t \Rightarrow e^i\right)$. So problem (7) is equivalent to finding a set of edge lengths, such that $\frac{L^e}{d(t \Rightarrow e^i)}$ is minimized. Thus the optimal value of problem (7) is $OPT = \min_{d_e} \frac{L^e}{d(t \Rightarrow e^i)}$.

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 $i$th iteration. $d_e^{(0)} = \beta$ is the initial weight of $d_e$. Regarding $d_e^{(i)}$, we simplify the following denotations $L^e$, $d_e^{(i)}$ and $d(t \Rightarrow e^i)$, into $L^e$, $t_e^{(i)}$ and $d(t \Rightarrow e^i)$. We also denote $f^0$ as the total flow that has been routed after the $i$th iteration. Then based on the edge length update...
function (Line 10 in Table 7), we have
\[
L^{(i)} = \sum_{e \in E} d_{e}^{(i-1)}c_{e} + \sum_{e \in E} n_{e}^{(i-1)}d_{e}^{(i-1)}(f_{e}^{(i)} - f_{e}^{(i-1)})
\]
\[
= L^{(i-1)} + \varepsilon \sum_{j=1}^{L^{(i-1)}} (f_{j}^{(i)} - f_{j}^{(i-1)})d_{j}^{(i-1)}
\]
which implies that
\[
L^{(i)} \leq L^{(0)} + \varepsilon \sum_{j=1}^{L^{(0)}} (f_{j}^{(i)} - f_{j}^{(0)})d_{j}^{(i-1)}
\]
(22)

Now let us consider the length function \(d_{e}^{(i-1)}\), 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 \(L^{(i-1)} = L^{(i)} - L^{(0)}\). Since \(d_{e}^{(i-1)}\) and \(d_{e}^{(i)}\) only differ by the constant \(\beta\) at each edge, \(t^{(i-1)}\) and \(t^{(i)}\) are the same tree. In addition, the length of the tree using \(d_{e}^{(i)}\) versus \(d_{e}^{(0)}\) differs by at most \(\beta |V| U\), \(U\) being the length of the longest unicast route. Hence
\[
OPT \leq \frac{L^{(i-1)} - L^{(0)}}{d_{e}^{(i-1)}} \leq \frac{L^{(i-1)} - L^{(0)}}{d_{e}^{(i-1)}} - \beta |V| U
\]
Substituting this bound on \(L^{(i)} - L^{(0)}\) in Equation (22) gives
\[
\frac{d_{e}^{(i)}}{R(t^{(i)})} \leq \frac{\beta |V| U}{R(t^{(i)})} + \frac{\varepsilon}{OPT} \sum_{j=1}^{L^{(0)}} (f_{j}^{(i)} - f_{j}^{(0)})d_{j}^{(i-1)}
\]
\[
\leq \beta |V| U + \frac{\varepsilon}{OPT} \sum_{j=1}^{L^{(0)}} (f_{j}^{(i)} - f_{j}^{(0)})d_{j}^{(i-1)}
\]
since \(R(t) \geq 1\).

Observe that, for fixed \(i\), this right hand side is maximized by setting \(d_{e}^{(i)}\) to its maximum possible value, for all \(0 \leq j \leq i\). Let us call this maximum value \(d_{e}^{*(i)}\). Hence
\[
\frac{d_{e}^{(i)}}{R(t^{(i)})} \leq \frac{\beta |V| U}{R(t^{(i)})} + \frac{\varepsilon}{OPT} \sum_{j=1}^{L^{(0)}} (f_{j}^{(i)} - f_{j}^{(0)})d_{j}^{*(i)}
\]

\[
= \frac{\beta |V| U}{R(t^{(i)})} + \frac{\varepsilon}{OPT} \sum_{j=1}^{L^{(0)}} (f_{j}^{(i)} - f_{j}^{(0)})d_{j}^{*(i)}
\]

\[
= \frac{\beta |V| U}{R(t^{(i)})} + \frac{\varepsilon}{OPT} \sum_{j=1}^{L^{(0)}} (f_{j}^{(i)} - f_{j}^{(0)})d_{j}^{*(i)}
\]

\[
\leq \frac{\beta |V| U}{R(t^{(i)})} + \frac{\varepsilon}{OPT} \sum_{j=1}^{L^{(0)}} (f_{j}^{(i)} - f_{j}^{(0)})d_{j}^{*(i)}
\]

Since \(d_{e}^{*(i)} / R(t^{(i)}) \leq \beta |V| U\), this implies that
\[
\frac{d_{e}^{(i)}}{R(t^{(i)})} \leq \beta |V| U
\]

\[
d_{e}^{(i)} \leq \beta |V| U t^{(i)}
\]

\[
1 \leq \beta |V| U t^{(i)}
\]

By Lemma 2, \(f^{*} \leq \log_{1+\varepsilon} (1+|V| U) / \beta\) is a feasible solution to problem (7). Then the ratio between the optimal value of problem (7) and the result returned by our algorithm is
\[
\frac{OPT}{f^{*}} \leq \frac{\varepsilon \log_{1+\varepsilon} (1+|V| U) / \beta}{\beta \ln (1+|V| U)}
\]
(23)

\[
= \frac{\varepsilon \ln (1+|V| U)}{\beta \ln (1+|V| U)}
\]

\[
= \frac{\varepsilon \ln (1+|V| U)}{\beta \ln (1+|V| U)}
\]

when \(\beta = \frac{\varepsilon \ln (1+|V| U)}{\beta \ln (1+|V| U)}\), the maximum number of iterations needed by the algorithm is

\[
(23) \leq \frac{\varepsilon}{(1-\varepsilon) \ln (1+\varepsilon)} \leq \frac{\varepsilon}{(1-\varepsilon)(\varepsilon^{2}/2 - \varepsilon)/2} \leq \frac{1}{(1-\varepsilon)^{2}}
\]

Now we are ready to proof Theorem 1 as follows.

Proof: By Lemma 1, the algorithm terminates after at most \(\varepsilon \log_{1+\varepsilon} \frac{1+\varepsilon}{\beta} \) rounds, each round containing a minimum spanning tree construction. When \(\beta = \frac{\varepsilon \log_{1+\varepsilon} (1+|V| U) / \beta}{\beta \ln (1+|V| U)}\), the maximum number of iterations needed by the algorithm is
\begin{align*}
|E| \log_{1+\varepsilon} \left[ \left( (1+\varepsilon)U \right)^{1/\varepsilon} |\mathcal{V}|^{2|\mathcal{E}|} \right] \\
\leq \frac{|E|}{\varepsilon} \left( 1 + \log_{1+\varepsilon} U + \log_{1+\varepsilon} |\mathcal{V}|^{|\mathcal{E}|} \right) \\
= \frac{|E|}{\varepsilon} \left( 1 + \log \left( U |\mathcal{V}|^{|\mathcal{E}|} \right) \right) \\
\leq |E| + \frac{|E|}{\varepsilon^2} \log \left( U |\mathcal{V}|^{|\mathcal{E}|} \right)
\end{align*}

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

Appendix: Proof of Theorem 2

Proof: We prove that problem (7), the dual problem of (5) 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 \( \{v_1, \ldots, v_i\} \) and \( \{\overline{v_1}, \ldots, \overline{v_i}\} \). In Figure 10, we build an overlay network, in which testing if constraint (8) is violated, i.e., the separation oracle of problem (5), 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 \( \overline{v_i} \). 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. Figure 10 only shows all edges with smaller lengths: they directly from \( s \) to all literal nodes, between each pair of literal nodes, and from each literal node to each clause node it appears in.

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: Proof of Theorem 3

Proof: 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 \(|\mathcal{V}| + 1\) trees shown in Figure 3 with higher generalized throughput. In the second part, we prove that the tree selection priority the Multi-Trees-Star algorithm follows at choosing among these \(|\mathcal{V}| + 1\) 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 \( R(t) \), which consists of all non-leaf nodes in \( t \). It is obvious that \( \mathcal{S} \subseteq R(t) \). For each node \( v \in R(t) \), we denote \( nc_v(t) \) as the number of children \( v \) has in \( t \). Then the bandwidth contribution by \( v \) is \( nc_v(t) f(t) r_v \). 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 \( nc_v(t) f(t) r_v R_v(t) \), the product of \( v \)'s bandwidth contribution, resilient factor, and resilient index. Under both models, the resilient throughput contributed by the server \( s \) is \( nc_s(t) f(t) r_s \), 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 \( R(t) \).

We now reorganize \( t \) as follows. For each peer node \( v \in R(t) \), we select tree \( t_v \) in Figure 3, where \( v \) is the relaying node. The flow rate of the selected tree is \( nc_v(t) f(t) r_v / |\mathcal{V}| - 1 \), 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 nonconcatenation 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 \( R(t) \), the total bandwidth contribution by \( s \) would have reached \( (|\mathcal{V}| - nc_s(t)) f(t) / |\mathcal{V}| - 1 \), which is also its resilient throughput contribution. To consume the bandwidth still
left at $s$, we construct tree $t_0$, the last tree in Figure 3, with rate \( \left( nc_i(t) - 1 \right) f(t) / |V| - 1 \). 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 \( nc_i(t) f(t) \). 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 $R(t)$ 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 $R(t)$.

As it is now clear that the $|V| + 1$ trees shown in Figure 3 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 \( c_v / |V| - 1 \) and the remaining bandwidth of $s$. $t_0$ consumes solely the bandwidth of $s$. As such, the bandwidth of $s$ becomes a 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 \( c_0 / |V| \). 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.

**Appendix: Proof of Theorem 4**

Proof: Our proof consists of three parts. First, we show that the rate returned by MaxRate-Star is feasible to construct a tree. Second, we show that given a fixed rate, MostResilientTree-Star constructs the tree with the maximum generalized throughput. Third, we show that the number of iterations by SingleTree-Star is bounded by \( O(|V|^2) \).

MaxRate-Star works in a trial-and-error fashion. In line 2, it sets the rate $f$ to $c_s$, the uplink bandwidth of the server, which the maximum possible value for $f$. From line 3 to 9, it finds $\text{sum}$, the total number of children that the server and all peers can support. If it is smaller than the target value $|V|$, a smaller value of $f$ will be tried until $\text{sum} \geq |V|$. Since the value of $m$ will advance by at least 1 in each iteration, the algorithm runs in no more than $|V|$ iterations.

Next, MaxRate-Star constructs a tree by the given streaming rate. It works in a greedy fashion by scheduling the peer with the highest resilience factors to take the maximum number of children it can, starting from the server whose resilience factor is the maximum. Given the definition of a tree’s generalized throughput in the non-concatenation model, the tree constructed will return the highest generalized throughput. The number of peers scheduled by the algorithm is bounded by $|V|$.

Finally, SingleTree-Star tries different streaming rates from the maximum rate returned by MaxRate-Star to the minimum value $c_s / |V|$, since any value smaller than that allows the server to stream to all peers, a tree with the maximum resilience index. Since in each iteration, at least one peer’s outbound degree (number of children it can support) will increase by 1, it takes at most $|V|$ iterations to increase the server’s outbound degree by 1. Therefore, the number of iterations by SingleTree-Star is bounded by $|V|^2$, which makes the running time of MostResilientTree-Star \( O(|V|^2) \).