Accurate and Efficient Simulation of Bandwidth Dynamics for Peer-To-Peer Overlay Networks

Alexandros Gkogkas
Royal Institute of Tech. (KTH)
gkogkas@kth.se

Roberto Roverso
Royal Institute of Tech. (KTH)
Peerialism Inc.
roverso@kth.se

Seif Haridi
Royal Institute of Tech. (KTH)
haridi@kth.se

ABSTRACT

When evaluating Peer-to-Peer content distribution systems by means of simulation, it is of vital importance to correctly mimic the bandwidth dynamics behaviour of the underlying network. In this paper, we propose a scalable and accurate flow-level network simulation model based on an evolution of the classical progressive filling algorithm which follows the max-min fairness idea. We build on top of the current state of the art by applying an optimization to reduce the cost of each bandwidth allocation/deallocation operation on a node-based directed network model. Unlike other works, our evaluation of the chosen approach focuses both on efficiency and on accuracy. Our experiments show that, in terms of scalability, our bandwidth allocation algorithm outperforms existing directed models when simulating large-scale structured overlay networks. In terms of accuracy we show that allocation dynamics of our proposed solution follow those of the NS-2 packet-level simulator by a small and nearly constant offset for the same scenarios. To the best of our knowledge, this is the first time that an accuracy study has been conducted on an improvement of the classical progressive filling algorithm.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems, Modeling techniques; C.2.4 [**Computer Communication Networks**]: Distributed Systems, Distributed applications

General Terms

Network Simulation, Peer-to-Peer system evaluation and testing

Keywords

Bandwidth Dynamics simulation, Flow-level simulation, Peer-to-Peer systems, Progressive filling

1. INTRODUCTION

It is common for existing P2P networks to create complex interactions between thousands of participant peers. Each peer usually has a very high inbound and outbound connection degree [1] [23] [21]. Connections are used either for signalling or content propagation. In the latter case, each peer implements intricate multiplexing strategies to speed up transmission of large chunks of data [4], thus creating complex effects on the underlying physical network which translate into varying transmission delays and packet loss at the receiving side.

In order to study the complex interactions between overlays and physical networks, a proper performance evaluation technique is required. Currently, no consensus has been reached in the scientific community on a reference P2P testing platform [15]. It is of a common practice to test a P2P application in a simulator before real deployment in order to enable controlled testing and evaluation. However, accurate network simulators can usually scale up only to a limited number of simulated peers. This limitation makes it impossible to capture the behaviour and issues of a larger P2P real-word deployment, such as the effect of network congestion on segments of the overlay network. In order to achieve scalability, most of the existing P2P simulators abstract away network interactions by modeling only the structural dynamics of the overlay network [2] [10] [19] [14] [20] and thus totally ignoring the impact of the actual network on application performance.

Packet-level network simulation allows for in-depth study of the influence of the network characteristics and the lower level protocols on the performance of a P2P system. The trade-off for this detailed analysis comes in scalability: a single transfer can generate hundreds of low-level protocol packets. In a large and complex network characterised by links of high bandwidth capacity and intense in-line traffic, the sheer number of events needed to simulate the transfers require a prohibitive amount of computational and memory resources. As a result, only small sized networks can be simulated efficiently. A number of packet-level network simulation frameworks has been conceived in the last decade [17] [22], being NS-2 [18] the most prominent among them. In turn, a number of P2P network simulators have been developed on top of NS-2, e.g. P2PSim [7] and NDP2PSim [12].

Flow-level simulation focuses instead on a transfer as a whole rather than individual packets, introducing a viable trade-off between accuracy and scalability. A flow abstracts away the small time scale rate variation of a packet sequence with a constant rate allocated at the sender/receiver's bandwidth. The rate remains allocated for an amount of time which corresponds to the duration of the flow, i.e. the simulated packet sequence transmission time. This approach reduces drastically the number of events to be simulated. The driving force behind the event creation in flow-level simulation is the interaction between the flows, since an upload/download link might have many flows happening at the same time. A new or completed flow might cause a rate change on other flows competing for that same link's capacity. For instance, in order for a link to accommodate an extra flow, bandwidth needs to be allocated to it. This might lead to a decrease of the bandwidth rates of other competing flows. Similarly, when a flow is completed, more bandwidth becomes available for the other flows to share. A flow rate change may also propagate further in the simulated network graph. This phenomenon is known as *"the ripple effect"* and has been observed in a number of studies [11] [6] . The impact of the ripple effect on the scalability of the model is directly dependent on the efficiency of the bandwidth allocation algorithm.

In this paper we provide a description of the current state of the art in flow-level network bandwidth simulation in Section 2. Later we present a concise description of our contribution in Section 3, before delving into the details of the proposed model in Section 4. We then discuss the results of the scalability and accuracy evaluation of our implementation in Section 5 and, finally, in Section 6, we draw our conclusions and discuss future work.

2. RELATED WORK

Flow-level network simulators implement algorithms which mimic the bandwidth dynamics happening in the transport protocol layer used of the real network. Bertsekas and Gallager [3] introduce the concept of *max-min fairness* for modeling Additive-Increase Multiplicative-Decrease congestion control protocols like TCP. Max-min fairness tries to maximize the bandwidth allocated to the flows with minimum share thus guaranteeing that no flow can increase its rate at the cost of a flow with a lower rate. In every network exists a unique max-min fair bandwidth allocation and can be calculated using the progressive filling algorithm [3]. The basic idea behind this algorithm is to start from a situation where all flow rates are zero and then progressively increment each rate equally until reaching the link's capacity, i.e. the sum of all flow rates of a link equals its capacity. In this algorithm, the network, including its internal structure, e.g. routers and backbone links, is modelled as an undirected graph. A recent accuracy study [5] showed that this approach offers a good approximation of the actual network behaviour. Nevertheless, having to simulate the flow interactions that take place on the internal network links, magnifies the impact of the ripple effect on the algorithm's scalability by making the simulation significantly slower.

In order to gain more scalability, the *GPS* P2P simulator [24] uses a technique called *minimum-share allocation*, defined in [8], which avoids the propagation of rate changes through the network. Instead, only the flow rates of the directly affected nodes are updated, i.e. only the flow rates of the uploading and downloading nodes of the flow triggering the reallocation. Not considering the cross-traffic effects of the flows obviously has a positive impact on the simulation time but makes also the model highly inaccurate.

Narses [8] uses the same technique as GPS but it further promotes scalability by ignoring the internal network topology considering only the bandwidth capacity of the access links of the participating peers. The result is what we call an *end-to-end* network overlay where the backbone network is completely abstracted away from the modeling and rate changes happen between pairs of peers. This is a reasonable abstraction if we consider that the bottlenecks on a P2P network usually appear in the "last mile" rather than the Internet backbone. In doing so, the number of events simulated is further reduced, however in this case the inaccuracy remains since only the end-to-end effects are taken into account while the cascading effect on other nodes, as modelled by max-min fair allocation, is completely overlooked.

There exists two bandwidth allocation algorithms in the state of the art which apply the progressive filling idea on end-to-end network models, thus keeping the advantages of simulating only access links but still considering the effects and propagation of rate changes throughout the peer interconnections. The first algorithm proposed by F. Lo Piccolo et Al. [13] models the end-to-end network as an undirected graph. In each iteration, the algorithm finds the *bottleneck* nodes in the network, the nodes with the minimum fair bandwidth share available to their flows. Then it proceeds to allocate the calculated minimum fair share to their flows. The algorithm iterates until all nodes are found saturated or a rate is assigned to all their flows.

The main disadvantage of this node-based max-min fair bandwidth allocation algorithm lies in the modeling of the network as an undirected graph. In order to simulate a network with separate upload and download capacities, two node instances are required per actual network peer. The memory footprint is therefore larger than the one needed to model a direct network graph. Another weakness is that the computational complexity of this approach depends directly on the cardinality of the node set.

An alternative edge-based max-min bandwidth allocation algorithm is given by Anh Tuan Nguyen et al. [16]. It is an edge-based algorithm which uses a directed network model, differently from the approaches we introduced till now. In one iteration, the algorithm calculates the minimum fair share of the two ends of every unassigned flow. Then, on the same iteration and based on the previously calculated shares, the algorithm finds the bottleneck nodes, derives the flows' rates and applies them. The algorithm iterates until all flows have a rate assigned. It is important to underline that during the second phase of each iteration, the algorithm might find one or multiple bottleneck nodes, thus assigning rates to the flows of multiple nodes at the same iteration.

This edge-based max-min fair bandwidth allocation algorithm addresses the shortcomings of the undirected network modeling. However, the algorithm performance's dependence on the edge-set size constitutes a major drawback, since each iteration requires the calculation of the minimum fair share for each unassigned flow. In addition, a further iteration of the node set is required in order to find the saturated nodes. As we will see, this hidden complexity makes it unsuitable for simulating large networks with high adjacency.

It is common in large simulated networks for a new or finished flow to only affect the rates of a subset of the existing network flows, as the propagation of a rate change does not reach all nodes in the network but rather few of them.

Based on this observation, F. Lo Piccolo et Al. [13] partially outline an affected subgraph discovery algorithm that can be applied on an undirected network graph. Starting from the end nodes of the flow triggering the reallocation, the algorithm traverses the segment of the network graph that is affected by the change. In fact, this is an attempt to trace the aforementioned ripple effect. The traverse continues through affected flows/edges until all the affected network subgraph is visited. The nodes reached by the traversal are treated based on the parity of the hop distance (odd or even) from any of the two root nodes, i.e. the initiators of the allocation/deallocation. Depending on the parity, a different subset of their flows is affected. The traverse continues until no new nodes are affected. Since a node can be reached in an odd or even hop distance from a root, a node can be visited maximum twice during a traverse. This results in a linear complexity with respect to the node set cardinality, assuming that a node is not reconsidered when reached again in the same distance from a root.

Using this optimization algorithm before applying an undirected node-based max-min fair bandwidth allocation algorithm leads to a large performance gain. Unfortunately, F. Lo Piccolo et Al. apply this only on an undirected network model. Moreover, the authors provide only a sketch of the affected subgraph idea rather than a state-complete algorithm.

In general, the aforementioned works which propose an improvement to the classical progressive filling focus their evaluation of the algorithms on performance while completely overlooking the accuracy of their approaches.

3. CONTRIBUTION

We propose a scalable and efficient flow-level simulator which leverages the benefits of the affected subgraph optimization on a directed network model. We give a state-complete description of the subgraph optimization introduced by F. Lo Piccolo et Al. and we evaluate its performance gain when used in combination with our max-min fair bandwidth allocation algorithm. The result is an algorithm whose computational complexity is independent of the edge set size. Experimental results show that our solution constantly outperforms the edge-based algorithm proposed by Anh Tuan Nguyen et al. for large-scale and structured network overlays. Finally, we conduct a detailed accuracy study where we compare the simulated transfer times of our proposed solution with the ones obtained using NS-2 for the same realistic scenarios. We show that the bandwidth allocation follows the trends of the actual packet-level simulated bandwidth dynamics.

4. PROPOSED SOLUTION

We model the network as a directed graph $G = (U, F)$, where the vertex set U represents the set of end nodes and the edge set F represents the set of directed flows between pairs of end nodes. We define as c_i^u and c_i^d the uploading and downloading capacities of node i which belongs to the node set U. The rate of a flow f_{ij} is denoted as r_{ij} where $i, j \in U$ and $i \neq j$. The existence of a flow from i to j does not imply the existence of a flow from j to i. As a result, every node $i \in U$ has two separate sets of flows, F_i^u for the outgoing, and F_i^d for the incoming. Finally, $r^*_{i_u}$ and $r^*_{i_d}$ are the fair shares of the upload and download capacity respectively of

Algorithm 1: Node-based max-min fair bandwidth allocation

```
input : A set of nodes $U$ with their upload and
        download bandwidth capacities $c_i^u, c_i^d, \forall i \in U$,
        and their corresponding flow sets $F_i^u$, $F_i^d$.
output: The Max-Min fair rate allocation $\vec{r}$ of the
        bandwidth capacities to the flows.
1  begin
2      $SatUp \leftarrow \emptyset$; $SatDown \leftarrow \emptyset$;
3      while $F \neq \emptyset$ do
4          $SatUp \leftarrow \{s_u | r^*_{s_u} = min_{i\in U, c_i^u \neq 0} \frac{c_i^u}{|F_i^u|}\}$;
5          $SatDown \leftarrow \{s_d | r^*_{s_d} = min_{i\in U, c_i^d \neq 0} \frac{c_i^d}{|F_i^d|}\}$;
6          $r^* = min(r^*_{s_u}, r^*_{s_d})$;
7          if $r^*_{s_u} > r^*_{s_d}$ then
8              $SatUp \leftarrow \emptyset$;
9          else if $r^*_{s_u} < r^*_{s_d}$ then
10             $SatDown \leftarrow \emptyset$;
11         if $SatUp \neq \emptyset$ then
12             foreach $i \in SatUp$ do
13                 foreach $f_{ij} \in F_i^u$ do
14                     $r_{ij} = r^*$;
15                     $c_j^d = c_j^d - r^*$;
16                     $F_j^d \leftarrow F_j^d - \{f_{ij}\}$;
17                 $c_i^u = 0$; $F_i^u \leftarrow \emptyset$;
18         if $SatDown \neq \emptyset$ then
19             foreach $i \in SatDown$ do
20                 foreach $f_{ji} \in F_i^d$ do
21                     $r_{ji} = r^*$;
22                     $c_j^u = c_j^u - r^*$;
23                     $F_j^u \leftarrow F_j^u - \{f_{ji}\}$;
24                 $c_i^d = 0$; $F_i^d \leftarrow \emptyset$;
```

a node $i \in U$. A fair share is the equal division of a node's upload or download capacity to its flows.

4.1 Bandwidth allocation

We now present the first part of our solution: our node-based max-min fair bandwidth allocation in Algorithm 1.

The algorithm iterates until all flows have a rate assigned. Each iteration has two phases. In the first, we find the node(s) which provide the minimum fair share r^* (lines 4-10). The process happens as follows: we calculate the fair share of the upload and the download capacity of each node. Then, we find the nodes that provide the minimum upload fair share $r^*_{s_u}$ and the minimum download fair share $r^*_{s_d}$ and add them to node set *SatUp*, *SatDown* respectively (lines 4-5). The smallest rate between $r^*_{s_u}$ and $r^*_{s_d}$ is set as the minimum fair share r^* and the nodes of the corresponding set are considered saturated (line 6-10), i.e. they constitute the bottleneck of the network in this iteration. In the case where $r^*_{s_d} = r^*_{s_d}$, then all nodes in *SatUp* and *SatDown* are considered saturated.

In the second phase, we allocate the minimum fair share r^* to the flows of each saturated node, either to their downloading or uploading side (lines 11-24). The available bandwidth

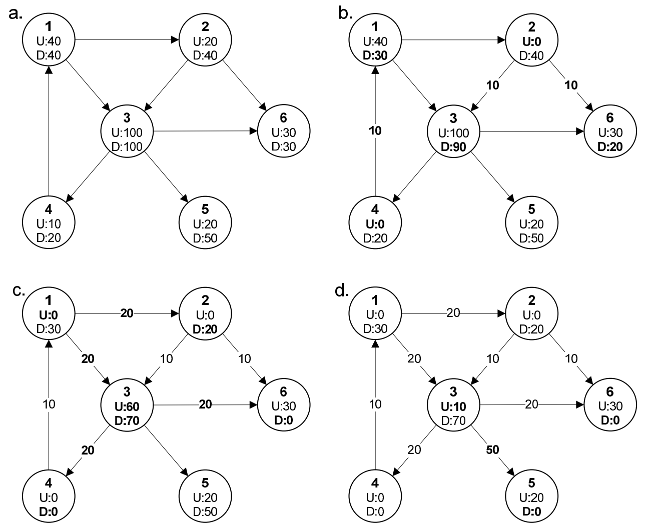
Figure 1: An example of the max-min bandwidth allocation algorithm.

on the other end of each flow (c_j^u for a downloading flow or c_j^d for an uploading flow) is reduced and the assigned flows are removed from the flow sets (lines 15-16, 22-23). The saturated nodes have no available capacity left on their affected side (downloading/uploading) and their corresponding flow sets are empty (17, 24).

An example of how our max-min fair allocation algorithm works is shown in Figure 1. We consider the network graph shown in Figure 1.a. In the first iteration, the minimum fair share in the graph is found to be 10, this is provided by the upload capacities of nodes 2 and 4. The state of the graph after the allocation of the minimum fair share to flows $f_{2,3}$, $f_{2,6}$ and $f_{4,1}$ is shown in Figure 1.b. In the second iteration, the minimum fair share is 20 and is provided by the upload side of node 1, as well as the downloading side of node 4 and node 6. The minimum fair share is assigned to flows $f_{1,2}$, $f_{1,3}$, $f_{3,4}$ and $f_{3,6}$, which results to the network state shown in Figure 1.c. Finally, in the last iteration, the download side of node 5 provides the minimum fair share of 50, which is allocated to its downloading flow $f_{3,5}$. The final rate allocation for this network graph is shown in Figure 1.d.

The complexity of the algorithm is $O(N^2)$ since at least one node side is found saturated in each iteration. In order to improve scalability, we adapt the affected subgraph discovery algorithm for use with directed end-to-end network models.

4.2 Affected subgraph discovery

This optimization is essentially an affected subgraph discovery algorithm. Given a flow that triggers a bandwidth reallocation, the algorithm initiates two graph traversals that each one has as root one of the flow's end nodes. In each hop of a traversal we find the affected flows of the last reached nodes and continue the traverse to their other ends. This procedure continues until no newly affected nodes are discovered by any of the two traversals. We characterize the nodes reached by one of these traversals based on three criteria: (1) *the parity of the distance in hops from the root node*, i.e. odd or even; (2) *the type of the root node from which the traversal originates*, i.e. the uploading or downloading end of the triggering flow; and (3) *the type of the triggering event*, i.e. new or finished flow.

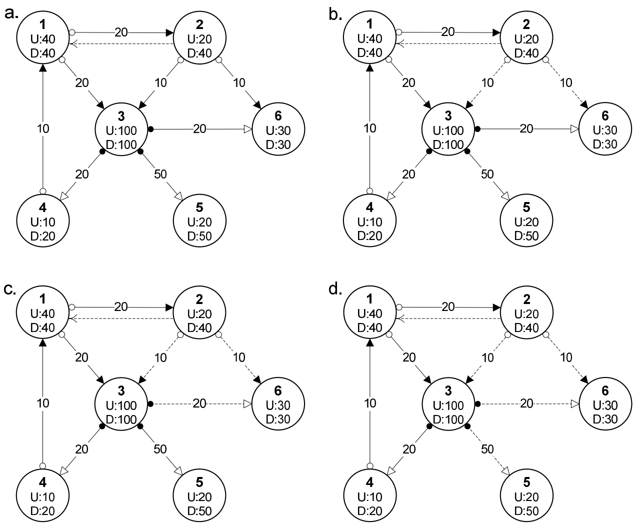
Figure 2: An example of the affected subgraph discovery.

Furthermore, the flows of a node i are distinguished between locally bottlenecked, L_i, and remotely bottlenecked, R_i. A node's flow is referred as *locally bottlenecked* if its own capacity is restricting the bandwidth rate of the flow. In contrast, a node's flow is referred as *remotely bottlenecked* if the node at the other end of the flow is the one restricting its bandwidth rate. For example, the flow $f_{4,1}$ in Figure 1.d is locally bottlenecked for node 4 since it is its upload capacity that restricts the flow's rate. In contrast, for node 1 the flow is considered remotely bottlenecked since its rate is restricted by the capacity available to the node on the other end of the flow.

A node can be affected in two different ways. First, it might need to accommodate a new flow or a rate increase of an already existing flow. Second, it might have some of its bandwidth freed due to a finished flow or a flow that has decreased its rate. Since we distinguish between the upload and download side of a node, we have a total of four different ways in which a node can be affected:

- **upload rate increase**
  A node in this category is asked for a rate increase. The criteria configurations which correspond to this group of nodes are {even, uploading, new} and {odd, downloading, finished}. This means that such a node will have all its locally bottlenecked uploading flows affected since they will need to reduce their rates in order to accommodate a new flow or a bandwidth increase. Moreover, it might happen that some or all of its remotely bottlenecked outgoing flows might turn into locally bottlenecked. We can find the remotely affected bottlenecked flows of a node x by evaluating the following expression, as proposed by [13]:

$$A : r_i < \frac{c_x^u - \sum_{j=1}^{i-1} r_j}{|F_x^u| - i + 1}, i \in |R_x^u|, \quad (1)$$

  c_x^u is the upload capacity of x, and r_i, where $i \in R_x^u$, the rate of a remotely bottlenecked uploading flow i

```
Algorithm 2: Affected subgraph discovery
  input : A set of nodes $U$ with their upload and
          download capacities $c_i^u, c_i^d \forall i \in U$, their flow
          sets $F_i^u$, $F_i^d$ and a triggering flow $f_{r_1,r_2}$.
  output: The subsets $AU$ and $AF$ of $U$ and $F$
          respectively, that correspond to the affected
          network subgraph.
1  begin
2      $hop \leftarrow 0$; $AU \leftarrow \emptyset$; $AF \leftarrow \emptyset$;
3      if $f_{r_1,r_2} \in F$ then
4          $AF \leftarrow AF \cup \{f_{r_1,r_2}\}$;
5      $SAU \leftarrow \{r_1\}$; $DAU \leftarrow \{r_2\}$;
6      while $SAU \neq DAU \neq \emptyset$ do
7          $AU \leftarrow AU \cup SAU \cup DAU$;
8          $SAU' \leftarrow \emptyset$; $DAU' \leftarrow \emptyset$;
9          foreach $i \in SAU$ do
10             if ($hop$ mod $2 \neq 0$ and $f_{r_1,r_2} \in F$) or
               ($step$ mod $2 == 0$ and $f_{r_1,r_2} \notin F$) then
11                 $SAU' \leftarrow SAU' \cup DownDecrease(i)$;
12             else if ($hop$ mod $2 \neq 0$ and $f_{r_1,r_2} \notin F$) or
               ($hop$ mod $2 == 0$ and $f_{r_1,r_2} \in F$) then
13                 $SAU' \leftarrow SAU' \cup DownIncrease(i)$;
14         foreach $i \in DAU$ do
15             if ($hop$ mod $2 \neq 0$ and $f_{r_1,r_2} \in F$) or
               ($hop$ mod $2 == 0$ and $f_{r_1,r_2} \notin F$) then
16                 $DAU' \leftarrow DAU' \cup UpDecrease(i)$;
17             else if ($hop$ mod $2 \neq 0$ and $f_{r_1,r_2} \notin F$) or
               ($hop$ mod $2 == 0$ and $f_{r_1,r_2} \in F$) then
18                 $DAU' \leftarrow DAU' \cup UpIncrease(i)$;
19         $SAU \leftarrow SAU'$; $DAU \leftarrow DAU'$;
20         $hop \leftarrow hop + 1$;
```

of x. We consider the remotely bottlenecked flows ordered by increasing rate: $r_1 \leq r_2 \leq \cdots \leq r_{|R_x^u|}$. The above condition is evaluated starting from $i = 1$ until it is either true for all $i \in R_x^u$ or becomes false for a certain i. In the first case all remotely bottlenecked flows remain unaffected. In the second case the flows with rates $r_i, r_{i+1}, ..., r_{|R_x^u|}$ might turn into locally bottlenecked by the rate increase and thus should be considered as affected.

- **upload rate decrease**
  The nodes included in this category got affected due to a rate decrease of one of their uploading flows. The criteria configurations for these nodes are {odd, downloading, new} and {even, uploading, finished}. Nodes falling into this category need to allocate a released amount of upload/download bandwidth to their currently active flows. This leads all their locally bottlenecked outgoing flows to be affected by the operation.

- **download rate increase**
  These nodes are affected by a rate increase in the same manner as the ones in the first category but by means of a downloading flow. The criteria configurations corresponding to this node category are {even, downloading, new} and {odd, uploading, finished}.

```
Procedure "UpDecrease(node)" returns the nodes
reached by means of locally bottlenecked uploading flows.
  input : A node $i$.
  output: The set of nodes reached by the affected
          locally bottlenecked uploading flows of $i$.
1  begin
2      $AF \leftarrow AF \cup \{f_{ij} | f_{ij} \in F_i^u$ and $f_{ij} \in L_i^u\}$;
3      return $\{j | f_{ij} \in F_i^u$ and j unaffected by download
       increase and $f_{ij} \in L_i^u\}$;
```

```
Procedure "UpIncrease(node)" returns the nodes
reached by means of locally and remotely bottlenecked
uploading flows.
  input : A node $i$.
  output: The set of nodes reached by affected locally
          and remotely bottlenecked uploading flows of $i$.
1  begin
2      $AF \leftarrow AF \cup \{f_{ij} | f_{ij} \in F_i^u$ and ($f_{ij} \in L_i^u$ or
       ($f_{ij} \in R_i^u$ and (1) false for $r_{ij}$))$\}$;
3      return $\{j | f_{ij} \in F_i^u$ and j unaffected by download
       decrease and ($f_{ij} \in L_i^u$ or ($f_{ij} \in R_i^u$ and (1) false
       for $r_{ij}$))$\}$;
```

- **download rate decrease**
  Like the nodes of the second category, these nodes experience a rate decrease but this time caused by a downloading flow. The criteria configurations which represent this category are {odd, uploading, new} and {even, downloading, finished}.

During the affected subgraph discovery, a node side can be reached both by a request to increase and by a request to decrease its flow rates. This makes for a clear conflict. In the original algorithm, the conflict is resolved by ignoring the change request that affects less children nodes from the current node's point of view. The approach clearly introduces inaccuracy since the propagation of a rate change is stopped abruptly. In our solution, we do not discriminate between the way by which a node is affected. However, this might introduce loops along the traversal path. In order to avoid this situation, we stop the traversal if a node gets affected twice in the same way, e.g. if it gets an increase request from its uploading flows twice in the same affected subgraph discovery.

The state-complete algorithm is shown in Algorithm 2 and Procedures *UpIncrease* and *UpDecrease*. We do not show procedures *DownIncrease* and *DownDecrease* since they are the complement of *UpIncrease* and *UpDecrease*. The algorithm begins by checking the type of the triggering event. In the case that it is a new flow f_{r_1,r_2}, it is added to the affected flow set AF (lines 3-4). A traversal is then initiated from each of the two root nodes, r_1, r_2. Two sets SAU and DAU containing the last reached nodes, for the traversals originating from r_1 and r_2 respectively, are updated at each hop. Initially, both sets contain only the root node (line 5). At each iteration, we check the last reached nodes and classify them based on the proposed cat-

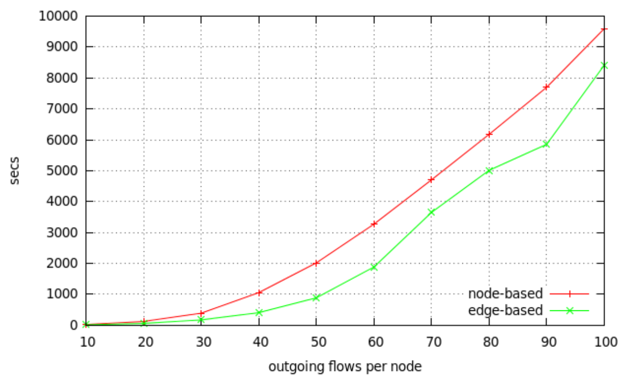

Figure 3: Simulation times of a random network overlay. 1000 nodes network size and varying number of outgoing flows per node.

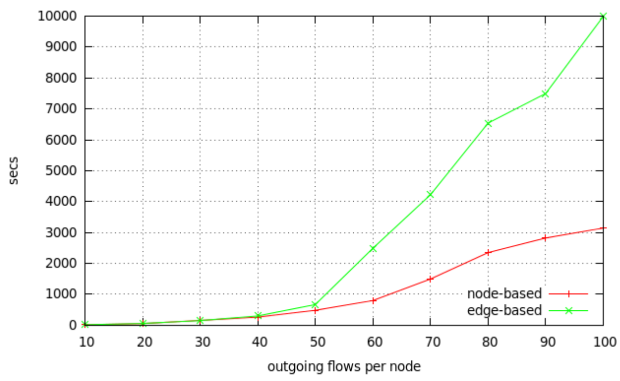

Figure 4: Simulation times for a DHT-like structured network overlay. 1000 nodes and varying number of outgoing flows per node.

egorization (lines 9-18). Depending on the parity of the hop distance, the root node (uploading/downloading) and the type of the triggering event (new or finished flow), we define which nodes will be affected in the next iteration using Procedures $UpIncrease$, $UpDecrease$, $DownIncrease$ and $DownDecrease$. As a result, two sets of newly affected nodes are produced after the iteration: one set (SAU') for the traversal originating from the uploading side of the triggering flow and one (DAU') for the traversal originating from the downloading side of the triggering flow. These node sets will be provided as input to the next iteration of the algorithm. The traversal continues until no more affected nodes are found (line 6). The complexity of the algorithm is $O(N)$ since each node can be reached in a maximum of 4 different ways.

In order to illustrate the affected subgraph discovery algorithm, we extend the annotation used in [13] in order to show both the direction and the bottleneck side of a flow. The source of a flow is depicted with a circle and the destination with an arrow. Their color shows the end of the flow that constitutes the bottleneck on a pairwise fashion. We denote with a white circle or arrow the restrictive end, i.e. the flow is locally bottlenecked. In contrast, we denote as a black circle or arrow the remotely bottlenecked end of a flow. If we use this annotation on the previous network graph example in Figure 1.d, we obtain the graph shown in Figure 2.a.

Lets consider the case of a new flow from node 2 to node 1. The algorithm firsts checks the outgoing flows of the uploading root node 2 and the incoming flows of the downloading root node 1 to find the affected ones, Figure 2.b. Since the triggering flow is a new one, all the locally and probably some remotely bottlenecked flows of its end nodes are affected. The uploading root node 2 has two locally bottlenecked uploading flows, $f_{2,3}$ and $f_{2,6}$, that are affected. The downloading root node 1 has only one remotely bottlenecked downloading flow $f_{4,1}$ which is not affected since (1) is true for its rate. In the next hop, the download flows of nodes 3 and 6 are checked, Figure 2.c. Since they are in an odd hop distance from the uploading side of a new transfer, only their locally bottlenecked downloading flows will be affected by means of a download decrease. Node 3 has no locally bottlenecked downloading flows while node 6 has only one, $f_{3,6}$. The next hop of the traversal reaches node 3 again, this time at an even hop distance from the uploading side of new flow which should affect all his locally and maybe some of his remotely bottlenecked uploading flows, Figure 2.d. Node 3 has only remotely bottlenecked uploading flows and one of them, $f_{3,5}$, has a rate for which (1) is evaluated as false. This flow is considered thus as affected. Finally, since node 5 has no unaffected flows, the algorithm terminates.

5. EVALUATION

5.1 Methodology

We focus the evaluation of our proposed solution on the two most important characteristics of a P2P network simulator: scalability and accuracy. In our scenarios, we consider two main parameters: the size of the node set and the number of outgoing flows per node. The destination of the flows is chosen either randomly or following a specific structure, i.e. a DHT-based one. The nodes enter the system in groups at defined time intervals. The starting times of a joined node's flows are distributed uniformly in that same time interval. The node's bandwidth capacities are either symmetric, asymmetric or mixed depending on the experiment. Finally, the amount of transferred bytes per flow is also a parameter.

For our accuracy study we implemented the same scenario both on our simulator and on the NS-2 packet-level simulator. In NS-2, we use a simple star topology, similar to the one used in [5] [9]. Each node has a single access link which we configure with corresponding upload and download capacities. All flows pass from the source access link to the destination access link through a central node with infinite bandwidth capacity. We use the TCP Reno implementation with a packet size of 1460 Bytes. The queue mechanism used is Drop Tail and all links have 2ms delay, resulting in a 8ms RTT. Finally, the TCP queue and the maximum window sizes are defined taking into consideration the bandwidth delay product.

We repeat each experiment a number of times with different random seeds and we take the average value of the resulting measurements. In the scalability study, we consider the average simulation time of the runs, while in the accuracy study we analyze the average deviation between

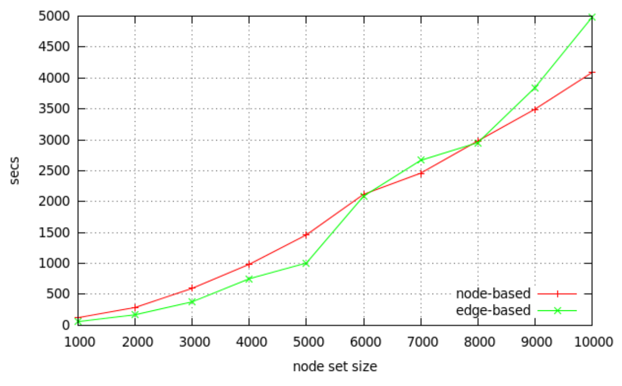

Figure 5: Simulation time for varying sizes of a random overlay networks with a fixed 20 outgoing flows per node.

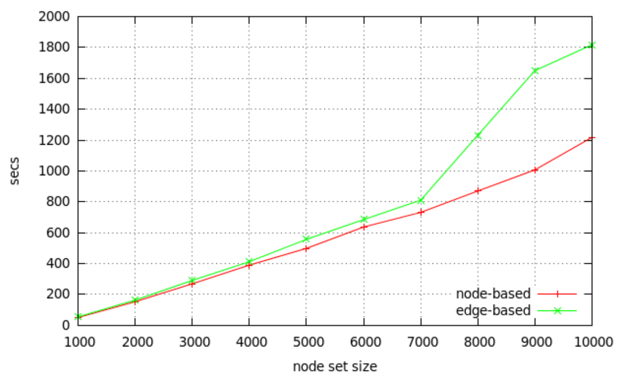

Figure 6: Simulation time for different sizes of a structured overlay networks with fixed 20 outgoing flows per node.

the NS-2 data transfers and the corresponding flows' time modeled in our simulator. The machine used for this evaluation has a dual core processor with 2.1GHz per core, 3MB cache and 3GB RAM.

5.2 Scalability

In our scalability evaluation, we vary the size of node sets, in the scale of thousands, and the number of outgoing flows per node, in increments of tens. Both random and structured overlays are characterized by a specific degree of inter-flow dependency. We can define the latter as how many flows on average are affected by a new or finished flow. i.e. the average ripple effect's scope. In the random scenario, it is more likely for the destination of the flow not being saturated, whereas in the structured, we guarantee a minimum level of saturation at each receiving end.

The rest of the scenario parameters are set as follows: the size of each flow is 2MB plus the size of TCP headers. Nodes join in groups of 50 every 20 seconds. The starting times of the flows of a newly joined node are uniformly distributed on the interval period. The bandwidth capacities of a node are chosen randomly from the set: {100Mbps/100Mbps,24Mbps /10Mbps,10Mbps/10Mbps,4Mbps/2Mbps,2Mbps/500Kbps} with corresponding probabilities of {20%,40%,10%,10%}.

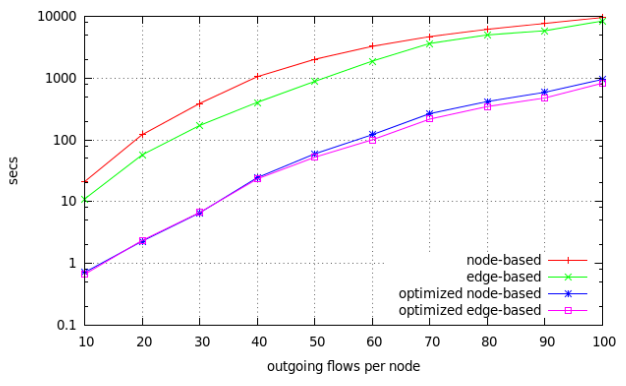

Figure 7: Performance improvement when using the affected subgraph discovery algorithm on random network overlays with 1000 nodes and different number of outgoing flows per node.

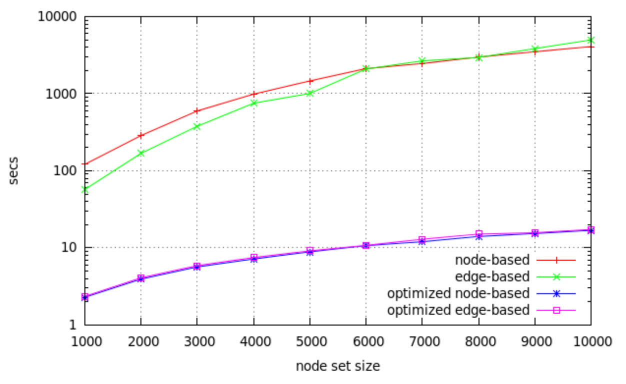

Figure 8: Performance improvement when using the affected subgraph discovery algorithm on random network overlays with varying size and 20 outgoing flows per node.

We first compare our proposed node-based max-min fair bandwidth allocation algorithm with the edge-based proposed by [16]. The simulation times required by both algorithms for a random scenario of networks with 1000 nodes and varying number outgoing flows per node are shown in Figure 3. The edge-based algorithm appears to perform slightly better than our solution. This is expected since the edge-based algorithm finds more saturated nodes per iteration, as mentioned in Section 2.

We run the same scenarios but this time selecting the destination nodes of the flows in a more structured manner. The selection is done in such a way that nodes form a circle where each node's outgoing flows are directed to the nearest neighbours in a clockwise fashion. The simulation times required by both algorithms for this scenario are shown in Figure 4. It is clear that the edge-based algorithm is significantly slower than our proposed algorithm in this case. This because the higher computational complexity of the edge-based algorithm emerges in those cases where stronger relations between the flows exist.

We next study the simulation times when dealing with larger size networks. We fix each node's outgoing flows to

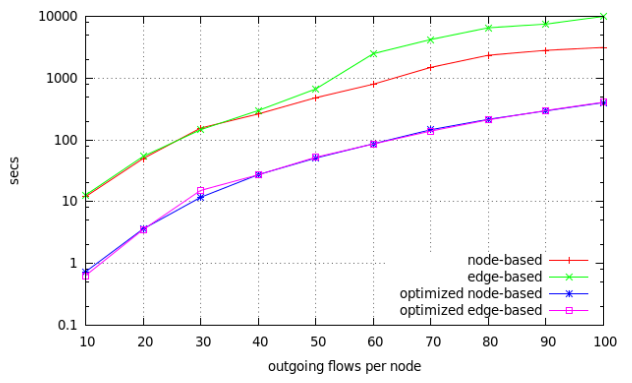

Figure 9: Performance improvement when using the affected subgraph discovery algorithm on structured network overlays with 1000 nodes and different number of outgoing flows per node.

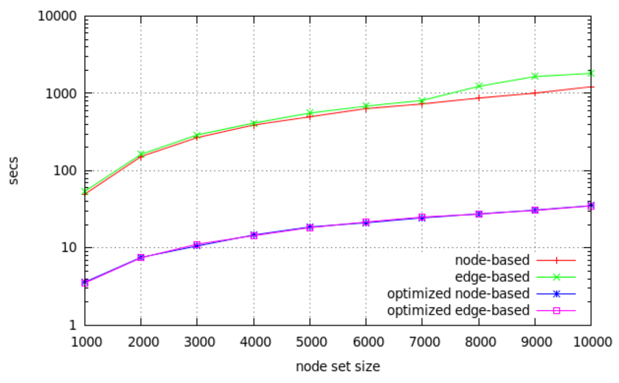

Figure 10: Performance improvement when using the affected subgraph discovery algorithm on structured network overlays with varying size and 20 outgoing flows per node.

20 and vary the node set size between 1000 and 10000 nodes. The results for the random and structured overlay scenarios are shown in Figure 5 and Figure 6 respectively. When random overlays are considered, we observe that, as the node set gets bigger, the edge-based algorithm performance deteriorates and is outperformed by our proposed algorithm. This can be explained by the fact that the edge-based algorithm includes an extra check of the complete node set per each iteration. This may not have a big impact when considering relatively small networks, but it might constitute a performance bottleneck for large scale simulations. The impact of this bottleneck becomes more apparent in the case of structured overlays where less nodes are found saturated at every iteration.

Based on these results, we can state that our proposed max-min fair bandwidth allocation algorithm performs better when simulating large and strongly connected networks with high inter-flow dependency. An other important conclusion drawn from these results is that the number of connections per node has a bigger impact in the performance of the simulation models rather than the node set size. For example, the simulation of a random overlay of 1000 nodes with 70 outgoing flows requires a similar simulation time to a 10000 nodes random overlay having 20 outgoing flows per node.

We also would like to point out that the performance gain when using flow-level simulation instead of packet-level is paramount. In order to simulate a random scenario of 1000 nodes with 10 flows each, a time of three orders of magnitude longer is required.

We run the same overlay scenarios using the directed affected subgraph discovery defined in Algorithm 2 before running the max-min fair bandwidth allocation algorithm. The results are shown in Figures 7-10. It is clear that the optimization reduces significantly the simulation time. In Figures 7 and 9 we can see that the required time is one order of magnitude lower when simulating network overlays with the same size but different number of outgoing flows per node. The performance improvement is much higher, three orders of magnitude, when we increase the network size and keep the number of flows per node fixed, as shown in Figures 8 and 10. This can be explained by the fact that the affected subgraph discovery size, on which a max-min fair bandwidth allocation algorithm is applied, mainly depends on the connectivity of the network. It should be also pointed out that the performance is similar for both our and the edge-based max-min bandwidth allocation algorithms when they are used together with the affected subgraph discovery algorithm. This is because the affected subgraphs in these cases are not big enough to point out the performance differences between the two algorithms.

5.3 Accuracy

The goal of this study is to find out how accurately the bandwidth capacity is allocated to competing flows with respect to a packet-level simulator, such as NS-2. Unfortunately, the size of our experiments is limited by the low scalability of NS-2. We run a scenario with 100 nodes joined at the start of the simulation. The destination of the nodes' outgoing flows is random, the size of each flow is 4MB and their number vary between 10 and 50 per node.

As in [5], we use the relative error in transfer times as our accuracy metric. Let t_{NS-2} be the time at which a flow terminates in NS-2 and $t_{max-min}$ the time at which our max-min fair flow terminates. The relative error is then given by:

$$RE = \frac{t_{flow} - t_{NS-2}}{t_{NS-2}} \qquad (2)$$

In our flow simulation, we ignore any segmentation or retransmission of packets. The packet header overhead introduced by the packet level simulation is added to the flow size.

We fist consider the case where the access link's capacities are symmetric, i.e. same upload and download capacity that we set to 10Mbps. Results for different number of outgoing flows per node are shown in Table 1. The standard and average deviation of the relative error of the transfer times are given for each of the scenarios.

We can see that the deviation is smaller when the network is more under stress, i.e. more outgoing flows per node. In these situations, TCP converges quicker causing the transfer times to deviate less. Moreover, since both sides of every flow have the same bandwidth capacity, the share that they provide should be similar. This leads to an oscillation of the

c^u/c^d	flows	std. deviation	avg. deviation
10/10	10	9.6±0.6%	7.9±0.6%
	20	7.3±0.3%	5.9±0.2%
	30	6.1±0.3%	5±0.2%
	40	5.1±0.3%	4±0.3%
	50	4.4±0.2%	3.6±0.2%
10/10, 20/20	10	12.7±1.0%	10.5±0.9%
	20	13.7±0.4%	11.3±0.4%
	30	12.6±0.3%	10.8±0.4%
	40	13.1±0.4%	11.4±0.5%
	50	13.2±0.3%	11.5±0.4%

Table 1: Deviation of simulated transfer times in the presence of symmetric node capacities.

c^u/c^d	flows	std. deviation	avg. deviation
20/10	10	3.8±0.4%	3±0.4%
	20	3.9±0.2%	3.1±0.1%
	30	4.1±0.3%	3.3±0.2%
	40	3.4±0.2%	2.8±0.2%
	50	3±0.1%	2.5±0.2%
20/10,10/10	10	6.4±0.4%	5±0.4%
	20	6±0.4%	4.9±0.3%
	30	4.8±0.4%	3.9±0.3%
	40	3.4±0.9%	3.2±0.3%
	50	3.5±0.2%	2.8±0.2%

Table 2: Deviation of simulated transfer times in the presence of asymmetric node capacities.

window size that translates into an extra transfer time deviation. However, while the absolute of the deviation varies with the number of links, we can still assert that the deviation is nearly constant, e.g. max 0.5%, between different iterations of the same experiment and different flows in the same experiment, for each one of the experiments conducted. In terms of absolute deviation, a flow-level simulator cannot compete with a packet-level because of the different level of abstraction. But we can safely state that, if the deviation is constant, the flow-level simulation follows the behavior of the packet-level simulation by the amount of that constant value, which is the desired effect of the simulator.

In the next experiment, we show the impact of the capacity size as well as the interaction between nodes with different symmetric capacities. For this purpose, we run the same scenario but this time setting half of the nodes in the network to symmetric bandwidth capacities of 20Mbps. The results are shown in Table 1. The introduction of nodes with higher capacity speeds up the transfer times, thus providing less time for TCP to converge. This effect leads to more deviation. We observe that the number of the outgoing flows per node does not affect the deviation. This might be because the impact of the higher bandwidth capacities to the time deviation is larger. Again, we can assert that the flow-level simulation follows the behavior of the packet-level one by a nearly constant degree.

Next we consider the case of asymmetric bandwidth capacities. We assign to every node 20Mbps of download capacity and 10Mbps of upload capacity. The results for different number of outgoing flows per node are shown in Table 2. It appears that the deviation is significantly smaller and not affected by the per node outgoing flows comparing to the previous symmetric scenario. This can be explained by the absence of oscillation and quicker TCP convergence due to the larger download capacity.

Finally, we investigate the deviation when both symmetric and asymmetric node capacities are used. We set half of the nodes with a symmetric 10Mbps capacity and the rest with asymmetric capacity of 20Mbps download and 10Mbps upload. The results are given in Table 2. We can see that the presence of the symmetric capacities affects negatively the transfer time deviation. The negative impact is more visible when fewer outgoing flows per node are used. When the links are less congested, the slower convergence of the flow rates of the nodes with smaller symmetric capacities is more apparent.

In the last two experiments, as in the first two, we measured a deviation from the NS-2 packet-level simulation but our max-min fair simulation followed the trends of the ones provided by NS-2 by an almost constant deviation factor.

6. CONCLUSION & FUTURE WORK

In this paper we presented a scalable and efficient flow-level network simulation model based on the max-min fairness idea. We evaluated our solution in terms of scalability by showing that it outperforms the existing state-of-the-art for large-scale and structured network overlays where a directed network is used for the modeling. In terms of accuracy we showed that our approach follows the trends of the NS-2 packet-level simulator network by a nearly constant factor throughout the experiments.

Our ongoing work includes the improvement of the model's accuracy by considering the time period that a flow requires to converge to a new rate, and also the use of time-stepping aggregation to achieve higher performance with minimum cost in accuracy.

7. REFERENCES

[1] A. Al Hamra, A. Legout, and C. Barakat. Understanding the properties of the bittorrent overlay. Technical report, INRIA, 2007.

[2] I. Baumgart, B. Heep, and S. Krause. Oversim: A flexible overlay network simulation framework. In *GI '07: Proceedings of 10th IEEE Global Internet Symposium*, pages 79–84, Anchorage, AL, USA, May 2007.

[3] D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, second edition, 1992.

[4] B. Cohen. Incentives Build Robustness in BitTorrent. In *Econ '04: Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkley, CA, USA, June 2003.

[5] A. Dandoush and A. Jean-Marie. Flow-level modeling of parallel download in distributed systems. In *CTRQ '10: Third International Conference on Communication Theory, Reliability, and Quality of Service*, pages 92 –97, 2010.

[6] D. R. Figueiredo, B. Liu, Y. Guo, J. F. Kurose, and D. F. Towsley. On the efficiency of fluid simulation of networks. *Computer Networks*, 50(12):1974–1994, 2006.

[7] T. M. Gil, F. Kaashoek, J. Li, R. Morris, and J. Stribling. p2psim: a simulator for peer-to-peer

(p2p) protocols. http://pdos.csail.mit.edu/p2psim/, October 2006.

[8] T. J. Giuli and M. Baker. Narses: A scalable flow-based network simulator. *Computing Research Repository*, cs.PF/0211024, 2002.

[9] T. Hossfeld, A. Binzenhofer, D. Schlosser, K. Eger, J. Oberender, I. Dedinski, and G. Kunzmann. Towards efficient simulation of large scale p2p networks. Technical Report 371, University of Wurzburg, Institute of Computer Science, Am Hubland, 97074 Wurzburg, Germany, October 2005.

[10] S. Joseph. An extendible open source p2p simulator. *P2P Journal*, 0:1–15, 2003.

[11] G. Kesidis, A. Singh, D. Cheung, and W. Kwok. Feasibility of fluid event-driven simulation for atm networks. In *GLOBECOM '96: Proceedings of the Global Communications Conference*, volume 3, pages 2013 –2017, November 1996.

[12] W. Kun, D. Han, Y. Zhang, S. Lu, D. Chen, , and L. Xie. Ndp2psim: A ns2-based platform for peer-to-peer network simulations. In *ISPA '05: Proceedings of Parallel and Distributed Processing and Applications, Workshops*, volume 3759, pages 520–529. Springer Berlin / Heidelberg, November 2005.

[13] F. Lo Piccolo, G. Bianchi, and S. Cassella. Efficient simulation of bandwidth allocation dynamics in p2p networks. In *GLOBECOM '06: Proceedings of the 49th Global Telecommunications Conference*, pages 1–6, San Franscisco, California, November 2006.

[14] A. M. Mark Jelasity and O. Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In *Engineering Self-Organising Systems*, pages 265–282, 2003.

[15] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *SIGCOMM Computer Communication Review*, 37(2):95–98, 2007.

[16] A. T. Nguyen and F. Eliassen. An efficient solution for max-min fair rate allocation in p2p simulation. In *ICUMT '09: Proceedings of the International Conference on Ultra Modern Telecommunications Workshops*, pages 1 –5, St. Petersburg, Russia, October 2009.

[17] D. M. Nicol, J. Liu, M. Liljenstam, and G. Yan. Simulation of large scale networks i: simulation of large-scale networks using ssf. In *WSC '03: Proceedings of the 35th Conference on Winter simulation*, pages 650–657. Winter Simulation Conference, December 2003.

[18] The network simulator ns-2. http://www.isi.edu/nsnam/ns/, October 2010.

[19] J. Pujol-Ahullo, P. Garcia-Lopez, M. Sanchez-Artigas, and M. Arrufat-Arias. An extensible simulation tool for overlay networks and services. In *SAC '09: Proceedings of the 24th ACM Symposium on Applied Computing*, pages 2072–2076, New York, NY, USA, March 2009. ACM.

[20] N. S. Ting and R. Deters. 3ls - a peer-to-peer network simulator. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 212. IEEE Computer Society, August 2003.

[21] G. Urvoy-Keller and P. Michiardi. Impact of inner parameters and overlay structure on the performance of bittorrent. In *INFOCOM '06: Proceedings of the 25th Conference on Computer Communications*, 2006.

[22] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Simutools '08: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, pages 1–10, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[23] C. Wu, B. Li, and S. Zhao. Magellan: Charting large-scale peer-to-peer live streaming topologies. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 62, Washington, DC, USA, 2007. IEEE Computer Society.

[24] W. Yang and N. Abu-Ghazaleh. Gps: a general peer-to-peer simulator and its use for modeling bittorrent. In *MASCOTS '05: Proceedings of 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 425–432, Atlanta, Georgia, USA, September 2005.