HAL open science

Download Process in Distributed Systems, Flow-level vs. Packet-level Simulation Analysis

Abdulhalim Dandoush, Alain Jean-Marie

To cite this version:

Abdulhalim Dandoush, Alain Jean-Marie. Download Process in Distributed Systems, Flow-level vs. Packet-level Simulation Analysis. [Research Report] RR-7159, INRIA. 2009, pp.22. inria-00442131

HAL Id: inria-00442131

https://inria.hal.science/inria-00442131v1

Submitted on 18 Dec 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Download Process in Distributed Systems, Flow-level Algorithm vs. Packet-level Simulation Model

Abdulhalim Dandoush — Alain Jean-Marie

N° 7159

December 2009

Thème COM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7159--FR+ENG

Download Process in Distributed Systems, Flow-level Algorithm vs. Packet-level Simulation Model

Abdulhalim Dandoush *, Alain Jean-Marie†

Thème COM — Systèmes communicants
Projet MAESTRO

Rapport de recherche n° 7159 — December 2009 — 22 pages

Abstract: Parallelism in the download process of large files is an efficient mechanism for distributed systems. In such systems, some peers (clients) exploit the power of parallelism to download blocks of data stored in a distributed way over some other peers (servers). Determining response times in parallel downloading with capacity constraints on both the client downloads and server uploads necessitates understanding the instantaneous shares of the bandwidths of each client/server is devoted to each data transfer flow.

In this report, we explore the practical relevance of the hypothesis that flows share the network bandwidth according to the max-min fairness paradigm. We have implemented into a flow-level simulator a version of the algorithm which calculates such a bandwidth allocation, which we have called the "progressive-filling flow-level algorithm". We have programmed a similar model over NS2 and compared the empirical distributions resulting from both simulations.

Our results indicate that flow-level predictions are very accurate in symmetric networks and good in asymmetric networks. Therefore, PFFLA would be extremely useful to build flow-level simulators and, possibly, to perform probabilistic performance calculations in general P2P networks.

Key-words: distributed systems, performance evaluation, download time, response service time, data distribution, simulation model, max-min fairness

* INRIA, 2004 Route des Lucioles, BP 92, F-06902 Sophia-Antipolis, Abdulhalim.Dandoush@sophia.inria.fr.
† INRIA and LIRMM, CNRS/Université Montpellier 2, 161 Rue Ada, F-34392 Montpellier, ajm@lirmm.fr.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Le Processus de Téléchargement dans des Systèmes Distribués, Analyse au Niveau Flux vs. au Niveau Paquet par Simulation

Résumé : L'application du parallélisme dans le processus de téléchargement de gros fichiers est apparue comme un mécanisme efficace dans les systèmes de stockage de données distribués. Dans un tel système, certains pairs (clients) exploitent la puissance de parallélisme afin de télécharger des données stockées de manière distribuée sur certains autres pairs (serveurs). Calculer les temps de réponse dans ces systèmes nécessite de bien comprendre comment les bandes passantes des clients et des serveurs sont partagées entre les différents flots d'information.

Dans ce rapport, nous explorons la validité pratique de l'idée que les flots se partagent les ressources du réseau selon le principe de l'équité max-min. Nous avons implémenté une version de l'algorithme qui calcule une telle répartition (nommée « PFFLA » pour « Progressive Filling Flow Level Algorithm ») dans un simulateur au niveau flot de téléchargements en parallèle. Nous avons programmé un modèle similaire au-dessus du simulateur de réseau au niveau paquet NS-2. Les temps de réponse dans le simulateur au niveau flux ont été comparés à aux ceux du simulateur au niveau paquet (leurs distributions et moyennes).

Nos résultats montrent que les prédictions de PFFLA au niveau flot sont très bonnes pour des réseaux symétriques, et bonnes pour des réseaux asymétriques. Nous concluons que PFFLA pourrait être extrêmement utile pour construire des simulateurs de réseaux au niveau flot, et possiblement pour faire des calculs probabilistes d'évaluation de performances.

Mots-clés : systèmes distribués, évaluation de performance, temps de téléchargement, distribution de données, modèle de simulation, équité max-min

1 Introduction and related work

The growth of storage volume, bandwidth, and computational resources for PCs has fundamentally changed the way applications are constructed. Almost 10 years ago, a new network paradigm has been proposed where computers or peers can build a virtual network (called overlay) on top of another network or an existing architecture (e.g. Internet). This new network paradigm has been labeled peer-to-peer (P2P) distributed network. A peer in this paradigm is a computer that play the role of both supplier and consumer of resources, in contrast to the traditional client-server model where only servers supply, and computers consume. Applications that use this distributed network provides enhanced scalability and service robustness as all the connected computers or peers provide some services.

This distributed network model has proved to be an alternative to the Client/Server model and a cheap, scalable, self-repairing and promising paradigm for grid computing, grid delivery network (GDN), file sharing, voice over IP (VoIP), backup and storage applications.

Such distributed systems rely on data fragmentation and distributed storage. Files are partitioned into fixed-size blocks that are themselves partitioned into fragments. Fragments are usually stored on different peers. Given this configuration, a user wishing to retrieve a given block of data would need to perform multiple downloads, generally in parallel for an enhanced service. The transfer of sequences of packets on one long-term TCP connection (e.g. download a fragment of data between two peers in a P2P system or between a client and a server through FTP protocol) defines a "flow". A flow can as well refer to the sequences of packets that constitute a block of data and that follow several TCP connections simultaneously. In this work, we will consider the former definition.

One measure of the quality of the service given by the distributed storage/parallel download infrastructure is the time it takes to retreive the complete document. This in turns depends on the throughput of the different flows created to obtain the fragments of this document. Their values are, *a priori*, a function of the demand and capacities of the complete network entities: clients, servers and links.

The basic problem of predicting the instantaneous shares of the bandwidth received by each flow of a TCP-based network has received quite some attention in the last 15 years, in connection with the notion of *fairness*; yet, there is no clear consensus in the literature on a simple formula or algorithm to give a reasonable solution of this problem. Such an algorithm would be extremely useful to build flow-level simulators and, possibly, to perform probabilistic performance calculations.

On the one hand, some authors have shown that the dynamics of TCP have been shown to be quite chaotic is some situations. Other authors on the other hand, have argued that TCP tends to share the bandwidth between flows reasonably. For instance, Heyman *et al.* [8], followed by Fredj *et al.* [1], have studied a single bottleneck link shared by a given number of identical sources that alternately send documents though the shared link and stop sending for randomly thinking time. They showed through simulations that TCP shares *fairly* the bottleneck (that is, in equal shares) and they introduced analytical tools that can predict the expectation of the transmitting rate. Varki proposed in [11] a simple approximation for the expected response time based on the fork-join model. Massoulié and Roberts proposed in [10] a model similar to that of [8] where the inter-flows arrival times are i.i.d. exponentially distributed random variables. They studied the network

as M/G/1 PS queue. In [5], Chiu and Eun, the authors have focused on the average download time of each user in a P2P network while considering the heterogeneity of service capacities of peers. They point out that the common approach of analyzing the average download time based on average service capacity is fundamentally flawed.

Other studies have put forward the concepts of max-min fairness, proportional fairness, balanced fairness and utility-based resource-sharing models (see *e.g.* [4] and the reference therein). One conclusion of these studies is that throughput allocations resulting from the use of the TCP protocol for infinitely long flows are usually *not* max-min fair. However, the results of Bonald and Proutière [3] suggested that when the flows are dynamic (flows are continuously created and have a finite duration), the average throughput obtained by flows under various sharing mechanism tend to be similar. It is quite possible that, from a practical perspective, the predictions obtained with a max-min fair sharing mechanism may be "good enough".

The purpose of this paper is to assess whether max-min fairness for the allocation throughput is a proper model when evaluating response times of parallel downloads.

Given the variety of situations to be studied, we begin with the simplest scenario: a symmetric network in which we assume that capacity constraints are located at the client/server nodes, and not inside the network. We also assume that all RTTs are equal. [1]

We use an algorithm which calculates an instantaneous throughput for each individual flows in a certain set of flows, given the upload and download capacities of the client and server nodes. This algorithm can be seen as a variant of the "progressive filling" [4] or "water filling" algorithm of [2]: we name it as the Progressive Filling Flow Level Algorithm (PFFLA). The validation of this algorithm consists in characterizing the response time of parallel downloads in a distributed storage system, through simulations. We have implemented the PFFLA in a flow-level simulator of parallel downloads, and we have programmed a similar model over NS2. The response times in the flow level simulator have been compared to that of the packet-level simulations in NS (both distributions and averages). This experimental setting is, to the best of our knowledge, original in at least three features. First, we consider flows related to downloads in parallel, which are synchronized when they are created. Second, we consider that the possible bottlenecks for flows occur only at the edge of the network, never inside. Finally, we consider large numbers of nodes (up to 500) and flows.

Our results show that the relative error between PFFLA and NS-2 for the expected value is less than 2% for relatively large loads in the system (e.g. $\rho = 60\%$) and less than 1% for low loads in the symmetric case and less than 7% respectively for relatively large loads in the system (e.g. $\rho = 50\%$) and less than 1% for low loads in the asymmetric case. We conclude that PFFLA is a reliable mechanism to analyze the service response time in many systems based on P2P and Grid computing concepts such as Storage Systems and Grid Delivery Networks.

The rest of this paper is organized as follows. Section 2 overviews the system assumptions and notation. Section 3 describes the flow-level simulation algorithm "PFFLA". In Section 4, comparisons between packet-level and PFFLA are introduced and discussed. Last, Section 5 concludes the paper and highlights some future directions.

[1] The question of how to handle different round trip times is left for future work.

2 System description and notation

In the following, we will distinguish the *servers*, which are computers that provide a storage service, from the *clients* whose objective is to retrieve data from the servers to account for the fact that flows (transfer of sequences of data units from a server to a client) have a direction. In the terminology of P2P-based systems, each "peer" has the role of both a client and a server. It is usual that the communication link from the network to the peer (upload link) and the one from the peer to the network (download link) are not shared. Their capacity may actually not even be the same, as with ADSL network accesses. In that case, the entities client and server can be considered as two distinct nodes. On the other hand, if the network access is indeed shared between input and output, the peer is represented by one node. In the following, we shall only consider the first situation.

In this study, we are interested in systems where blocks of data are partitioned into several equally sized fragments stored randomly over different servers. We will consider both homogeneous (upload/download capacities are identical) and asymmetric situations.

We consider a distributed system in which the following assumptions and notations will be enforced throughout the paper.

Network assumptions

- The considered network consists in a set of nodes $\mathcal{N}$. In a P2P context, there will be $\mathcal{N}/2$ peers according to this notation. In other words, we will have $\mathcal{N}/2$ servers and $\mathcal{N}/2$ clients.
- The logical structure of the network is that of a star, with an infinite-capacity central node. In other word, the interconnection network underlying the parallel download application is assumed not to introduce capacity constraints. Only the upload or download links (the branches of the star) have a limited capacity.

  The capacity of upload links (from servers to the network) is C_u, the capacity of download links (from network to clients) is C_d.
- The temporal distance (measured as the round-trip time, RTT) is assumed to be the same between pairs of nodes (clients or server).

Data and traffic assumptions

- Each block of data D of size S_B is partitioned into s fragments of size S_F.
- We assume that the s servers that hold fragments of a given block of data D are uniformly selected over all servers in the system, and are all distinct.
- Each download request of a block of data issued by a client will generate s parallel requests toward s servers to retrieve s distinct fragments of the requested block. A request generates s flows.

The assumption on the uniform distribution of the blocks of some document corresponds to the situation where a very large number of documents exist, and/or each fragment of each document has been replicated a large number of times. In that situation, it is unlikely that the set of blocks needed by two distinct requests will be correlated. The network being symmetric, it is reasonable to assume that fragments have been uniformy distributed. The assumption that different fragments of some document are stored on different peers is common in P2P-based systems: it results mainly from privacy and data ownership issues.

3 Description of the algorithm

Before describing the algorithm we have used, we recall the principle of max-min fairness, and the algorithm (referred to as the "Progressive Filling" algorithm in [4]).

The notion of max-min (or maximin) fairness originates from the field of political philosophy and economics, and was introduced in the context of networking by Bertsekas and Gallager [2, ch. 6] as a design objective for communication networks, in particular, the design of flow control schemes. The main idea of max-min fairness is to maximize the allocation of each flow f subject to the constraint that an incremental increase in f's allocation does not cause a decrease in some other flow's allocation that is already as small as f's or smaller [2, p. 526].

For the purpose of formalizing the description of algorithms, introduce the following notation. The network is assumed to be made of a set $\mathcal{A}$ of links. Each link a has a capacity C_a. The traffic is formed by a set $\mathcal{F}$ of flows. We assume that flows cannot be split between several routes of the network. This implies that we can assume that each flow f has a throughput $\theta_f \geq 0$, and crosses certain links of $\mathcal{A}$. We write $f \nabla a$ to denote the fact that f crosses a.

Using this notation, the total flow on link a of the network is then given by:

$$F_a = \sum_{f \nabla a} \theta_f \ .$$

The capacity constraint for the network is then:

$$F_a \leq C_a, \quad \forall a \in \mathcal{A} \ . \tag{1}$$

A vector of throughputs $\{\theta_f\}$ satisfying these constraints is said to be *feasible*.

The existence of a max-min fair allocation of throughputs is not entirely obvious. Indeed, satisfying the capacity constraints implies that the increase of some flow's throughput may result in the decrease of another flow's throughput, and conversely.

Bertsekas and Gallager have proved that there exists one unique feasible allocation of throughputs which is max-min fair. It can be constructed using the following algorithm.

The idea is to start with an all-zero rate vector and to increase the rates on all paths together until $F_a = C_a$ for one or more links a. At this point, each flow using a saturated link has the same throughput at every other flow using that link. Thus, these saturated links serve as bottleneck links for all flows using them.

At the next step, all flows not using the saturated links are incremented equally in rate until one or more new links become saturated. The newly saturated links serve as bottleneck links for those flows that pass through them but do not use the previously saturated links. The algorithm continues until all flows pass through at least one saturated link. This process is often visualized as progressively augmenting the throughput until capacities are "filled", hence the name of "progressive filling". Information flows are also sometimes visualized as some "fluid" which is poured into the network. For this reason, the algorithm is also referred to as a "water filling" algorithm.

We have used in our analysis the following version of this algorithm.

```
Data: A set of links 𝒜 with their capacities C_a, and a set of flows ℱ
Result: A throughput value for each flow, satisfying the throughput constraint (1)
begin
    Remove from 𝒜 nodes without flows ;
    while 𝒜 not empty do
        foreach a ∈ 𝒜 do
            N_a ← #{f ∈ ℱ | f∇a} ;
        end
        calculate θ* = min_{a∈𝒜} C_a/N_a ;
        calculate a* = arg min_{a∈𝒜} C_a/N_a ;
        foreach f, f∇a* do
            set θ_f = θ* ;
            foreach a, f∇a do
                C_a ← C_a − θ*
            end
            remove f from ℱ ;
        end
        remove from 𝒜 links without flows ;
    end
    return {θ_f}
end
```

Algorithm 1: Algorithm PFFLA

The fact that this algorithm produces a max-min allocation can be checked the same way as for the Progressive Filling Algorithm: according to [2, p. 527], max-min solutions are characterized by the "bottleneck" property:

$$\forall f \in \mathcal{F}, \quad \exists a \in \mathcal{A}, \quad f \nabla a \quad \text{and} \quad \sum_{g \in \mathcal{F}, g \nabla a} \theta_g \; = \; C_a \quad \text{and} \quad \forall h \in \mathcal{F}, \quad h \nabla a, \theta_h \geq \theta_f \; . \tag{2}$$

This property can be checked almost by construction on our algorithm.

More remarks concerning this algorithm:

- The algorithm eventually stops because at least one link is removed from $\mathcal{A}$ at each loop. The number of loops is therefore bounded by the cardinal of the initial set of links. Since several links can be removed in each loop, the algorithm may actually stop faster.
- When $arg\min_{a\in\mathcal{A}} C_a/N_a$ contains more than one element, it does not matter which one is chosen, in the sense that the outcome of the algorithm does not depend on that particular choice.

  This results, by contradiction, from the fact that the max-min allocation is unique. It can also be proved that other links of the set are still in the "argmin" at the next step, so that each of them will be chosen eventually. Equivalently, one may remove simultaneously from the network all flows that are connected to links in the "argmin" set.
- It is possible to add constraints on the throughput of flows. For instance, the throughput of a TCP flow on a lossless connexion with RTT τ and maximum window size w is always less than w/τ.

In our situation, we have made the assumption that the network can be represented by a star, and the flows cross exactly two links: one upload link and one download link. The algorithm is capable to handle general situations however.

4 Experimental results

4.1 Parameter values

We ran a total of fifteen experiments; ten in symmetric peers download/upload capacities scenarios (homogeneous networks) and five in asymmetric scenarios.

The set-up of the common parameters between the two simulation levels is summarized in Table 1. The capacities that we have selected in the simulations vary between the values of the ISDN and ADSL technologies (384, 576 and 1500 $kbps$). In experiments 1–10, nodes are homogeneous: they have all the same network access capacity. In Experiments 11–17, capacities of clients and servers are asymmetric.

Download requests at each client node arrive according to some Poisson process of given rate λ. The different request processes are independent. This assumption is reasonable in practice: Guha *et al.* have shown in [7] that in real networks, and when the number of clients N_c is large, the request arrival process can be reasonably modeled by a Poisson process. We vary the value of the request generation rate across the experiments such that the total load in the system ρ (see below) varies from low (e.g. 6$\%$) up to high value (e.g. 70$\%$) as reported in Table 1.

The last setting concerns the blocks and fragments sizes that are stored in the system. Fragment sizes S_F (resp. block sizes S_B) in P2P systems, for instance, are typically between 256KB and 4MB each (resp. between 4MB and 9MB each). We will consider in most of our experiments $S_F = 2$MB and $S_B = 8$MB, except in Experiment 1 where $S_F = 1$MB and $S_B = 4$MB. Therefore $s = 4$ in all experiments. In the asymmetric scenarios, we have chosen the two values 1500/384 so that the capacity of a server is slightly larger than $1/s$ times the capacity of a client.

For the packet-level simulation details, we consider a fixed constant value of 2ms for the link propagation delays. The main TCP configurations are as follow: we use TCP segment size (S_{pkt}) of 1460 Bytes, the upper bound on the advertised window for the TCP connection is set to 40, the initial size of the congestion window on slow-start is 2, and the TCP/IP header size (h_{ip}) is 40 Bytes. The P2P application layer header (h_a), which is implemented over the NS transport layer, is 13 Bytes for each fragment. The queue management type used in the links is "DropTail" with size of 500 packets. The maximum window size is left to NS2's default of 64kB. Given our assumptions on propagation delay, this gives a maximum TCP throughput of $64kB/8ms = 4MB/s$, largely superior to the capacity of the links. Therefore, maximum window effects are not expected to restrict the throughput of file transfers.

In the flow-level simulation, and when calculating the total amount of data sent in the TCP flows, we neglect the fact that one data packet may be incomplete after segmentation. We also neglect the packets sent during the opening and the closing of the TCP connection, and we assume that no retransmission occurs. The total amount of data transported during the download of one document is then calculated by multiplying the application data size by the overhead factor due to packet headers, that is:

$$L(bits) \;=\; s \;\times\; (S_F(bits) + h_a(bits)) \;\times\; (1 + h_{ip}/S_{pkt})\;. \tag{3}$$

Consider a client node with link capacity C. The time to download a complete document would be, when no interferences from other downloads occur:

$$\sigma \;=\; s \;\times\; \frac{(S_F(bits) + h_a(bits)) \;\times\; (1 + h_{ip}/S_{pkt})}{C(bps)}\;. \tag{4}$$

On the other hand, if the global arrival rate of document requests is λ, the rate of requests arriving at a particular client is $\lambda/(\mathcal{N}/2)$. Accordingly, the load factor of a client link of capacity C in the network is:

$$\rho \;=\; \lambda s \;\times\; \frac{(S_F(bits) + h_a(bits)) \;\times\; (1 + h_{ip}/S_{pkt})}{C(bps)\mathcal{N}/2}\;. \tag{5}$$

Consider now a server node with link capacity C. Given our assumption on the uniform repartition of blocks on servers, the rate of arrivals of fragment requests at the servers is $\lambda s/(\mathcal{N}/2)$. The duration of one request should be σ/s since only one fragment is concerned. Finally, the load factor of the server's link is given by Equation (5) also.

In the homogeneous cases, this value of ρ can also be interpreted as the load of the whole network (ratio of global data requests to global transfer capacity). In the asymmetric cases, we take ρ as the load of the links with the smallest capacity.

4.2 Simulators and Metrics

We have developed a packet-level simulator and a flow-level simulator for our model. The packet-level simulator is build using NS2. Its implementation details can be found in [6].

The flow-level simulator consists in the embedding of Algorithm 1 into a discrete-event simulator handling the arrival and the departure of flows. The principle is that every time the set of flows present in the network changes, the bandwidth shares are re-computed with the algorithm, and it

is assumed that these throughputs are obtained instantaneously. The program keeps track of the remaining quantities to be downloaded in each flow, and can compute the date of the next event: arrival or end of download.

Both simulators are instrumented so as to produce response times for fragments and complete documents.

The metric we are interested in is the *download time* of a document. For a given request, this is the maximum between the download time of the s fragments of the document. Of course, this is a random variable, and we measure its empirical distribution and empirical average.

In addition to the simulations, we have compared the average document download time, denoted by $E[T_{NS}]$ which is calculated from the packet-level simulation, with the average response time in a simple queueing system. The rationale for this is that, if the throughput of the connections were limited only by the client's capacity, then the link would behave as a Processor Sharing queue. This is because the size of the fragments is the same, so that the response time of all s fragments is the same, and all s fragments can be actually considered as a single "client". The client's bandwidth is then shared between different requests. Since requests arrive according to a Poisson process, the model is that of a $M/D/1$ processor sharing (PS) queue. This model is expected to work well when the load is small: indeed, in that case it is unlikely that flows will be limited on the server side. Since the average response time in this queue, denoted by $E[T_{PS}]$, can be computed with a simple formula, we can easily test this conjecture.

The average response time in the $M/D/1/PS$ queue is known to be (see *e.g.* [9]):

$$E[T_{PS}] \;=\; \frac{\sigma}{1-\rho} \;=\; \frac{s(\cdot S_F(bits) + h_a(bits)) \times (1 + h_{ip}/S_{pkt})}{C(bps) \times (1-\rho)} \quad (s)\,. \tag{6}$$

We will compute the relative error (RR) between $E[T_{NS}]$ and $E[T_{PS}]$, on one hand, and between $E[T_{NS}]$ and $E[T_{FLA}]$ on the other hand.

Results on the PS model also include the distribution of the response time. The relevant formulas are provided in the Appendix.

4.3 Results

We have run both the flow-level simulator and the packet-level simulator on the seventeen sets of values described in Table 1.

For the flow-level simulations, we have collected 100000 samples of the document download time in every case. The number of samples collected with the packet-level simulation is reported in Table 1 varies from 30000 to 70000 samples for most experiments. In Experiment 10, we collected roughly 280000 samples.

We also report in Table 1 the execution time of the packet-level simulations. The experiments were run on a machine with the following principal characteristics: multithreaded processor Intel(R) Core(TM)2 Duo of 2.66GHz, 4GB RAM + 4GB swap running Fedora Core 5. The analysis of number of samples issued by unit time of computation (figure not reported) reveals that this number decreases with the number of nodes. Interestingly, it tends to slowly decrease when the load ρ of the

Table 1: Experiments setup

Experiment number	$\mathcal{N}/2$ peers	C_d/C_u kbps	S_B/S_F MB	$1/\lambda$ sec.	ρ %	samples	Required time hours
1	25	384/384	4/1	60	6	44853	20
2	25	576/576	8/2	39.88	12	77651	18
3	250	1500/1500	8/2	1.536	12	9157	7
4	250	1500/1500	8/2	1.024	18	9801	5
5	250	576/576	8/2	1.913	25	31695	33
6	250	1500/1500	8/2	0.734	25	37688	31
7	250	1500/1500	8/2	0.510	36	41104	30
8	250	1500/1500	8/2	0.367	50	41877	36
9	250	1500/1500	8/2	0.306	60	67500	58
10	250	1500/1500	8/2	0.262	70	279333	264
11	25	1500/384	8/2	59.81	12	31966	23
12	250	1500/384	8/2	5.98	12	33509	54
13	500	1500/384	8/2	2.99	12	31817	25
14	250	1500/384	8/2	1.99	36	57645	54
15	500	1500/384	8/2	0.996	36	11601	6
16	500	1500/384	8/2	0.718	50	12580	6
17	500	2000/384	8/2	0.718	50	9000	5

network increases, but it is actually increasing for small values of ρ. We do not have an explanation for this observation.

The execution times for the flow-level simulation are not reported in details. It varies between one second and several minutes on a machine with characteristics: processor Intel(R) Core(TM)2 Duo of 2.00GHz, 2GB RAM running Fedora Core 5 (slightly less power than the machine used for packet-level simulations). The simulation time per sample increases with the load of the system, due to the fact that the load sharing must be re-computed every time a flow starts or stops. For a given load, it also increases with the number of nodes.

We conclude that the flow-level algorithm is very efficient in terms of time. However, one question remains: how good is it in term of accuracy?

To answer this question, we first depict in Figures 1–10 the empirical complementary cumulative distribution function (CCDF) of the block download time obtained form both simulators, and for all the experiments. We report then in Table 2 the expected block service time obtained from both simulation levels and from the PS formula (6). Table 2 reports as well the relative error between results, on one hand, of NS-2 and, on the other hand, of flow-level or PS models. The relative error, for instance between results from NS-2 and FLA is calculated as follow:

$$RR(NS, FLA) \;=\; \frac{E[T_{NS}] - E[T_{FLA}]}{E[T_{NS}]}$$

Table 2: Measurements for the PFFLA and the packet-level simulation; comparison with the PS model

Experiment number	$\mathcal{N}/2$ peers	ρ %	$E[T_{NS}]$ sec.	$E[T_{FLA}]$ sec.	$RR\%$ (NS, FLA)	$E[T_{PS}]$ sec.	$RR\%$ (NS, PS)
1 symm.	25	6	96.062	95.45	0.6%	95.44	0.6%
2 symm.	25	12	135.04	136.071	-0.7%	141.59	-4.8%
3 symm.	250	12	51.42	52.089	-1.3%	52.195	-1.5%
4 symm.	250	18	55.465	55.96	-0.8%	56.015	-0.9%
5 symm.	250	25	161.252	160.196	0.6%	166.132	-3%
6 symm.	250	25	61.068	61.517	-0.7%	61.243	-0.3%
7 symm.	250	36	73.547	73.346	0.2%	71.7692	2.4%
8 symm.	250	50	99.501	97.75	1.7%	91.864	7.6%
9 symm.	250	60	129.066	127.691	1%	114.83	11%
10 symm.	250	70	176.45	180.05	-2%	153.107	13.2%
11 asymm.	25	12	61.137	62.901	-2.8%	52.19	17%
12 asymm.	250	12	64.738	64.935	-0.3%	52.19	19.3%
13 asymm.	500	12	65.298	65.182	-0.18%	52.19	20%
14 asymm.	250	36	103.70	110.231	-6.2%	71.76	30.8%
15 asymm.	500	36	105.82	110.396	-4.3%	71.76	32.1%
16 asymm.	500	50	143.75	149.213	-3.8%	91.865	36%
17 asymm.	500	50	140.94	149.213	-5.9%	68.45	51.4%

where $E[T_{FLA}]$ is the measured empirical average download time for the FLA, and $E[T_{NS}]$ is the measured download time for NS, corrected by a constant value so that the minimal values for both simulators are the same. Indeed, there is a very small difference between the minimum values in all experiments shown in the figures, which is explained by the fact that the flow-level simulator ignores the delay for establishing and closing the TCP connections and propagation delays. We have not corrected this systematic error in the figures, but we have eliminated it for relative errors.

The results show that for small system load, the download time predicted by the PFFLA fits exactly that of the NS-2. The relative error between the average values is very small as shown in Table 2. The average value calculated from the PS formula is also very close but the relative error between average values of PS and NS-2 is slightly larger than that between PFFLA and NS-2. This confirms that the prediction of the duration of TCP flow is accurate. Indeed, since these are long flows, the slow start phase can be easily neglected. Other phenomena which typically perturb the throughput of TCP (packet losses, buffer fluctuations) probably happen very rarely in this case. The flow sharing algorithm apparently provides a very good approximation, for average response times as well as for distributions.

When ρ is relatively large, some buffers can fill up more frequently, and then some flows tend to be relatively long in the NS-2 simulation. However, the relative errors between average values of

PFFLA and NS-2 are slightly more important in this case but still very small, in particular, RR is less than 2% in the symmetric case and less than 8% in the asymmetric case. It is clear from Figures 5– 6 that the distributions measured by both simulators are different, but average values turn out to be almost identical. The same observation holds for asymmetric cases, see Figures 8(b) to 10.

The accuracy of the PS approximation for the average download time is acceptable for symmetric cases up to $\rho = 36\%$, and degrades above $\rho = 50\%$. The accuracy for the complete distribution can be assessed on Figure 4(b) for a load of $\rho = 36\%$. In the asymmetric cases, the approximation is bad at low loads, and very bad at large loads. The explanation for this is the following. The download of a block at a client can be slowed down by two phenomena. The first one is that a second request arrives at the node. This is taken into account by the PS model. The second one is that one TCP flow is slowed down at the server side. This requires that at least sC_u/C_d blocks are downloaded simultaneously from the server. In the symmetric cases, this value is 4 and the event rarely happens, even for moderate loads. In the asymmetric case, this value is 1 and this is much more frequent. See the Appendix for more comments on the PS approximation. In order to understand better the differences between flow-level and packet-level distributions, a careful analysis of the congestion avoidance mechanism in congested networks and an extension of the algorithm to account for the overloaded system (ρ around one) are the objective of ongoing research.

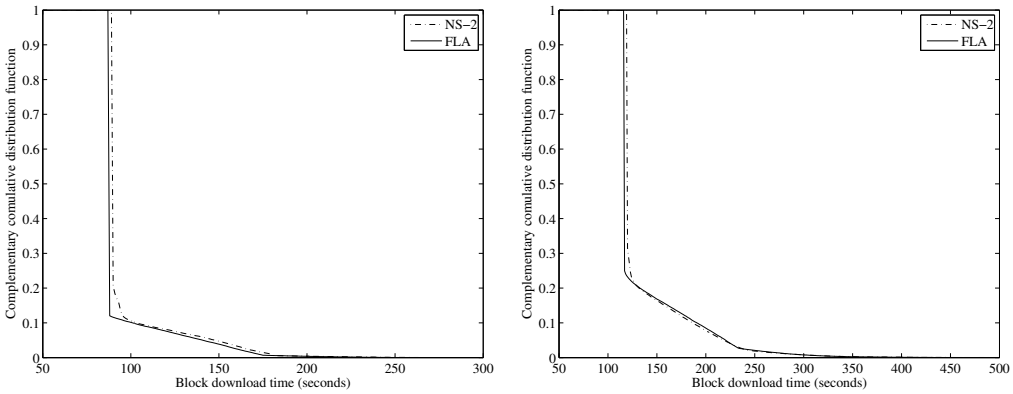
Another observation is that larger the network size, better the performance of PFFLA and worse the performance of PS model as illustrated by Experiments 11, 12 and 13 (resp. 14 and 15). The number of peers in these experiments are 25, 250 and 500 respectively (resp. 250 and 500) for same load and capacities. However, the relative error occurred in Experiment 14 is less than that of 13 which is, in turn, less than that of Experiment 12. Respectively, the relative error occurred in Experiment 16 is less than that of 15. Indeed, the performance does not depend only on the system load but also on the number of peers and their capacities.

Clearly, larger the buffer sizes, better the performance in real networks. To address this point, we depict in Figure 11 the CCDF of block download time for two values of the queue limit (100 and 500 packets) in NS-2 simulation for 25 peers (50 nodes), $C = 1500kbps$ and $\rho = 70\%$. The relative error between the two expected download time is 6.56% (159.575 seconds for 500 packets and 170.038 for 100 packets) and then the buffer size can affect the performance of the system with high load. The main problem here is that this parameter can not take large value because it is very expensive.

We conclude that PFFLA is a reliable mechanism to analyze the service response time in many non-overloaded systems based on P2P and Grid computing concepts such as Storage Systems and Grid Delivery Networks. In particular, when the size of networks is relatively large, PFFLA predictions are very accurate as long as the system is not overloaded or close to be overloaded.

5 Conclusion and future work

In this report, we have proposed and analyzed the PFFLA algorithm. The algorithm is quite simple and uses the concept of “Progressive-Filling” (or max-min fairness). We have implemented the PFFLA in a flow-level simulator of parallel downloads, and we have programmed a similar model



(a) $\rho = 6\%$, C=384kbps, $\mathcal{N}$=50, S_B=4MB, S_F=1MB (b) $\rho = 12\%$, C=576kbps, $\mathcal{N}$=50, S_B=8MB, S_F=2MB

Figure 1: Experiments 1 (left) and 2 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2

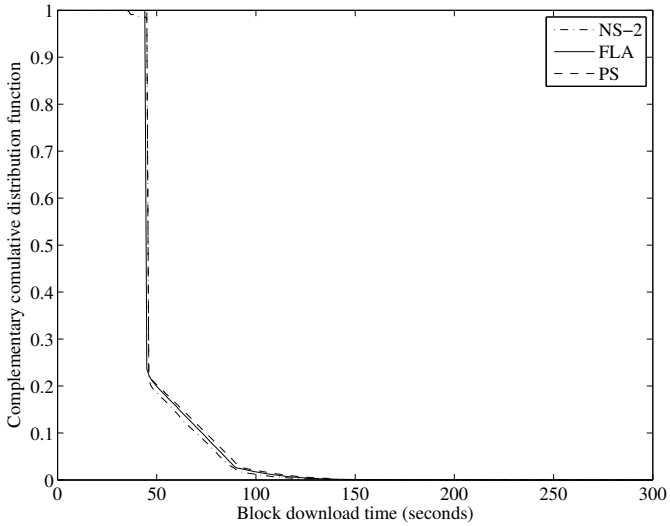
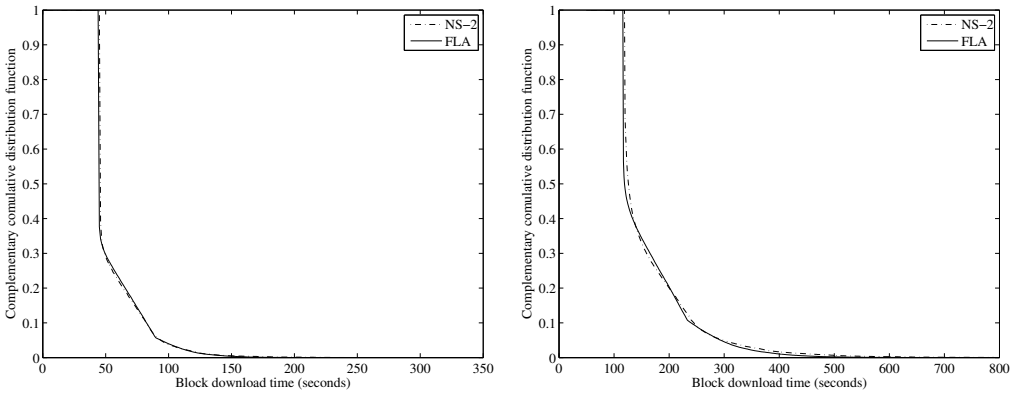


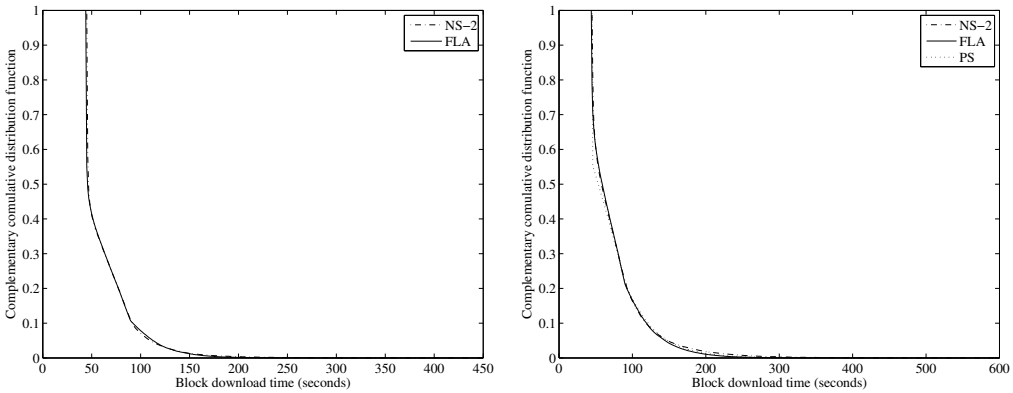
Figure 2: Experiment 3: Packet-level simulation NS-2 vs progressive-filling flow-level algorithm FLA & PS for $\rho = 12\%$, C = 1500kbps, $\mathcal{N}$ = 500, S_B = 8MB.

over NS2. The response times in the flow level simulator have been compared to that of the packet-level simulations in NS (both distributions and averages).



(a) $\rho = 18\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB. (b) $\rho = 25\%$, C=576kbps, $\mathcal{N}$=500, S_B=8MB.

Figure 3: Experiments 4 (left) and 5 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.



(a) $\rho = 25\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB. (b) $\rho = 36\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB.

Figure 4: Experiments 6 (left) and 7 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.

Our results conclude that PFFLA is a reliable mechanism to analyze the service response time in many systems based on P2P and Grid computing concepts such as Storage Systems and Grid Delivery Networks.

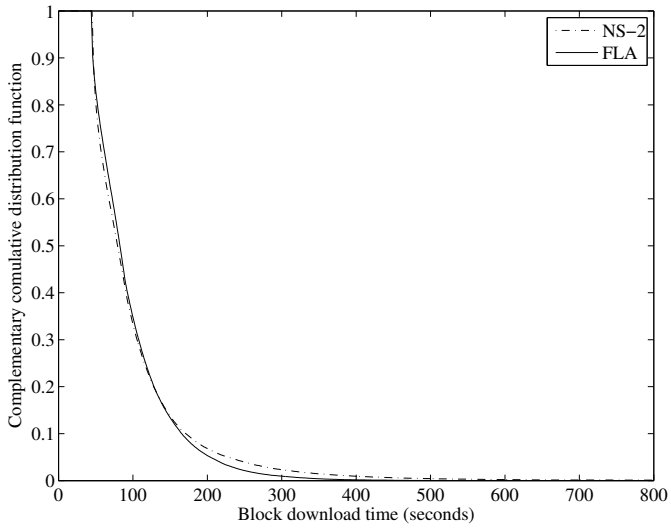
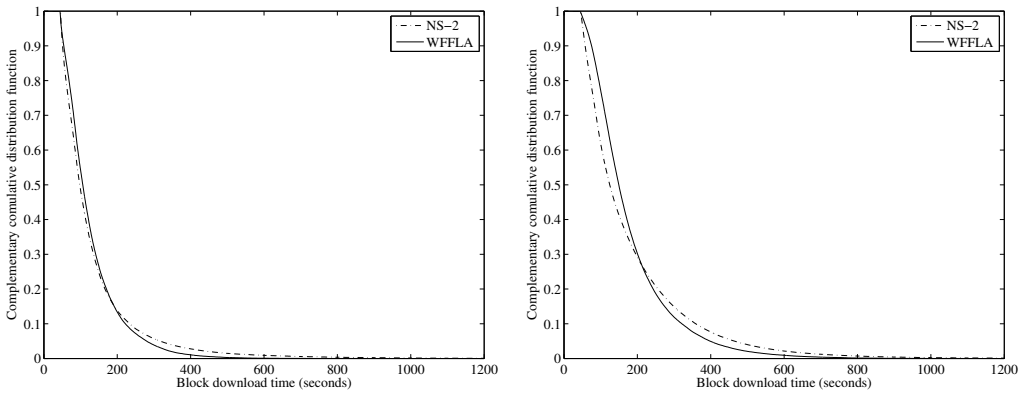


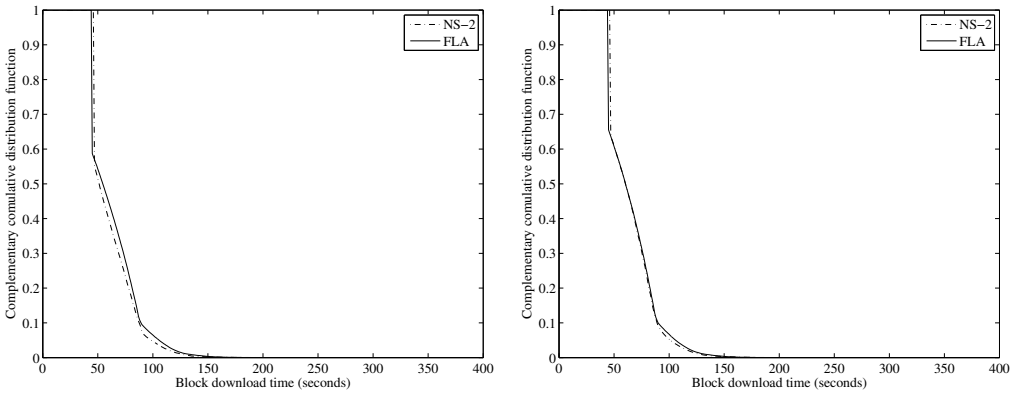
Figure 5: Experiment 8: progressive-filling flow-level algorithm FLA vs Packet-level simulation NS-2 for $\rho = 50\%$, C = 1500kbps, $\mathcal{N}$ = 500, S_B = 8MB.



(a) $\rho = 60\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB. (b) $\rho = 70\%$, C=1500kbps, $\mathcal{N}$=500, S_B=8MB.

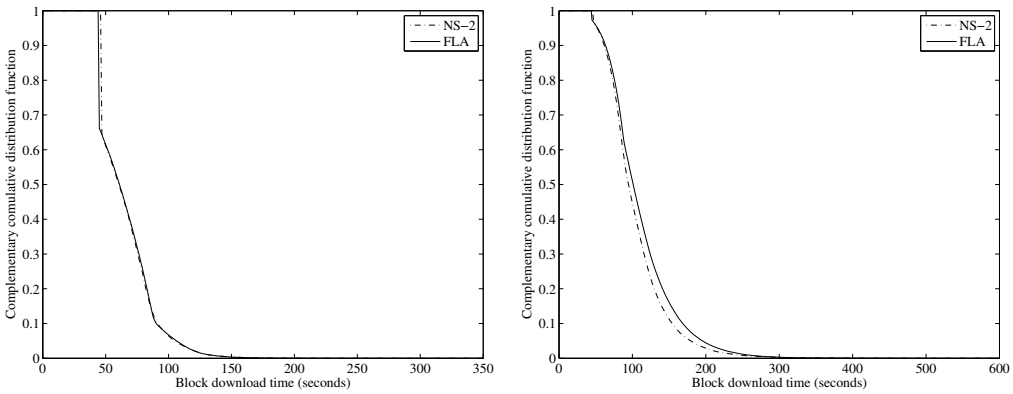
Figure 6: Experiments 9 (left) and 10 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.

A conclusion from the literature is that different RTTs do introduce some "unfairness" in bandwidth allocations. Our next step will therefore be to find a simple yet efficient modification of Algorithm 1 to handle this situation.



(a) $\rho = 12\%$, C_d=1500kbps, C_u=384kbps, $\mathcal{N}$=50, S_B=8MB. (b) $\rho = 12\%$, C_d = 1500kbps, C_u = 384kbps, $\mathcal{N}$ = 500, S_B = 8MB.

Figure 7: Experiments 11 (left) and 12 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.



(a) $\rho = 12\%$, C_d=1500kbps, C_u=384kbps, $\mathcal{N}$=1000, S_B=8MB. (b) $\rho = 36\%$, C_d=1500kbps, C_u=384kbps, $\mathcal{N}$=500, S_B=8MB.

Figure 8: Experiments 13 (left) and 14 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.

References

[1] S. Ben Fredj, T. Bonald, A. Proutiere, G. Régnié, and J. W. Roberts. Statistical bandwidth sharing: a study of congestion at flow level. *SIGCOMM Comput. Commun. Rev.*, 31(4):111–

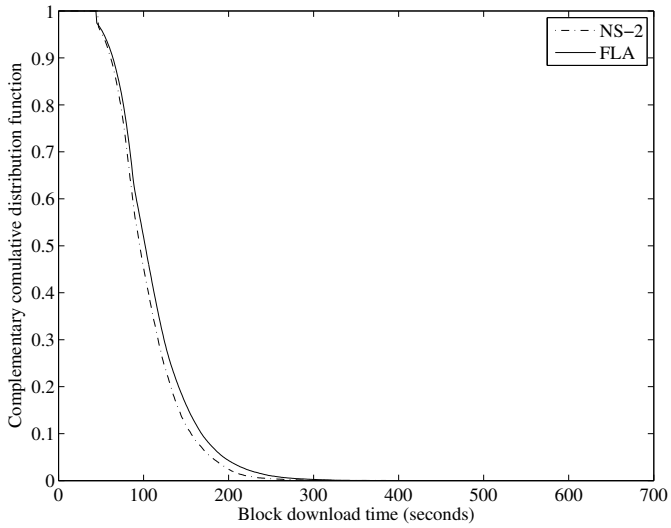
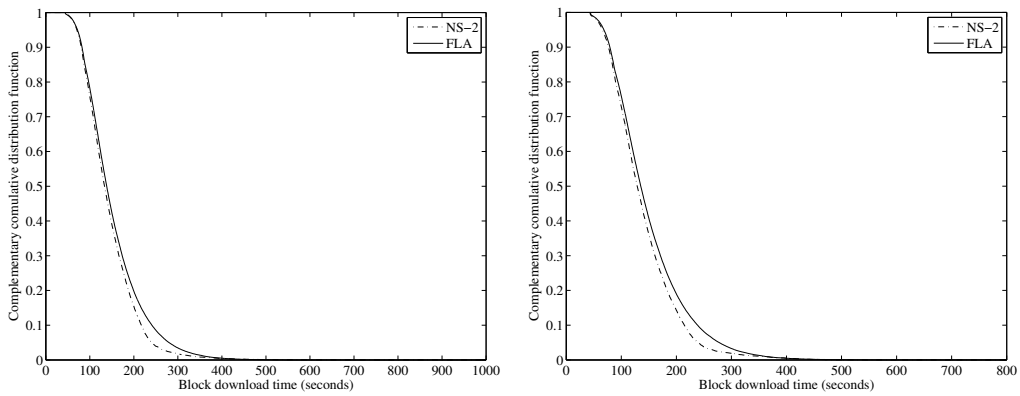


Figure 9: Experiment 15: progressive-filling flow-level algorithm FLA vs Packet-level simulation NS-2 for $\rho = 36\%$, C_d = 1500kbps, C_u = 384kbps, $\mathcal{N}$ = 1000, S_B = 8MB.



(a) $\rho = 50\%$, C_d=1500kbps, C_u=384kbps, $\mathcal{N}$=1000, S_B=8MB.

(b) $\rho = 50\%$, C_d=2000kbps, C_u=384kbps, $\mathcal{N}$=1000, S_B=8MB.

Figure 10: Experiments 16 (left) and 17 (right): progressive-filling flow-level algorithm PFFLA vs Packet-level simulation NS-2.

122, 2001.

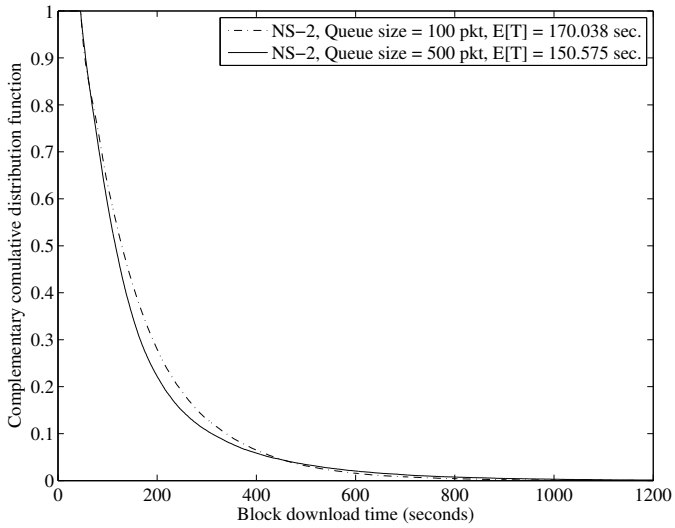


Figure 11: Queue size effect in Packet-level simulation NS-2 for $\rho = 70\%$, C = 1500kbps, $\mathcal{N}$ = 50, S_B = 8MB.

[2] D. Bertsekas and R. Gallager. *Data Networks, 2nd ed.* Prentice Hall, New Jersey, 1992.

[3] T. Bonald and A. Proutière. Insensitive bandwidth sharing in data networks. *Queueing Systems*, 44:69–100, 2003.

[4] Jean-Yves Le Boudec. *Rate adaptation, Congestion Control and Fairness: A Tutorial*. Ecole Polytechnique Fédérale de Lausanne (EPFL), Dec 2008.

[5] Yuh-Ming Chiu and Do Young Eun. Minimizing file download time in stochastic peer-to-peer networks. *IEEE/ACM Trans. Netw.*, 16(2):253–266, 2008.

[6] A. Dandoush, S. Alouf, and P. Nain. A realistic simulation model for peer-to-peer storage systems. In *Proc. of 2nd International ICST Workshop on Network Simulation Tools (NSTOOLS09), in conjunction with the 4th International Conference (VALUETOOLS'09)*, Pisa, Italy, October 19 2009.

[7] A. Guha, N. Daswani, and R. Jain. An experimental study of the skype peer-to-peer VoIP system. In *Proc. of 5th IPTPS*, Santa Barbara, California, February 2006.

[8] D. P. Heyman, T. V. Lakshman, and A. L. Neidhardt. A new method for analysing feedback-based protocols with applications to engineering web traffic over the internet. In *SIGMETRICS '97: Proceedings of the 1997 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 24–38, New York, NY, USA, 1997. ACM.

[9] L. Kleinrock. *Queueing Systems, Vol. 2*. J. Wiley, New York, 1975.

[10] L. Massoulié and J. W. Roberts. Bandwidth sharing and admission control for elastic traffic. *Telecommunication Systems*, 15(1-2):185–201, 2000.

[11] Elizabeth Varki. Response time analysis of parallel computer and storage systems. *IEEE Trans. Parallel Distrib. Syst.*, 12(11):1146–1161, 2001.

[12] S. F. Yashkov and A. S. Yashkova. Processor sharing: A survey of the mathematical theory. *Automation and Remote Control*, 2007.

A Approximations with Processor Sharing

A.1 Distribution of the response time in the $M/D/1/PS$ queue

According to Yashkov and Yashkova [12, Corollary 2.11] the distribution of the response time in a $M/D/1/PS$ queue with arrival rate λ and service time d, say $V(d)$, is given by its Laplace transform as:

$$E(e^{-sV(d)}) \;=\; (1-\rho)\,\frac{(s+\lambda)^2 e^{-d(s+\lambda)}}{s^2+\lambda(s+(s+\lambda)(1-\rho))e^{-d(s+\lambda)}}\;,$$

where the load factor is $\rho = \lambda d$. The inversion of this Laplace transform yields the series:

$$\begin{aligned} & P(V(d) \le d+t) \qquad (7)\\ &= (1-\rho)e^{-\rho}\sum_{n=0}^{\infty}(-1)^n e^{-n\rho}1_{\{t\ge nd\}}\\ &\quad \sum_{m=0}^{n}\binom{n}{m}(2-\rho)^m(1-\rho)^{n-m}\frac{[\lambda(t-nd)]^{2n-m}}{(2n-m)!}\\ &\qquad \left[1+2\lambda\frac{t-nd}{2n-m+1}+\frac{\lambda^2(t-nd)^2}{(2n-m+1)(2n-m+2)}\right]\;. \end{aligned}$$

For every $t \in [0, 2d]$, this formula reduces to:

$$\begin{aligned} & P(V(d) \le d+t) \qquad (8)\\ &= (1-\rho)e^{-\rho}\left[1+2\lambda t+\frac{\lambda^2}{2}t^2\right]\\ &\quad +(1-\rho)e^{-2\rho}1_{\{t\ge d\}}\Bigg\{(1-\rho)\frac{\lambda^2}{2}(t-d)^2\left[1+\frac{2}{3}\lambda(t-d)+\frac{\lambda^2}{12}(t-d)^2\right]\\ &\qquad (1-2\rho)\lambda(t-d)\left[1+\lambda(t-d)+\frac{\lambda^2}{6}(t-d)^2\right]\Bigg\}\;. \end{aligned}$$

A.2 Small load approximations

We briefly discuss now approximations that can be performed when the load or the arrival rate is small.

Consider a client downloading s flows in parallel. The nominal duration of each flow is σ/s, so that the total duration is σ if the flows are not disturbed.

Assume that the arrival of flows is a Poisson process of rate λ at all nodes: at the client node and the server nodes. Given some "tagged" download request, the probability that another request interferes with it *at the client* is $e^{-\lambda\times(2d)} = e^{-2\rho}$ because the request interferes if it arrives less than d units of time before or after the arrival of our tagged request. The same probability holds at each server.

The result of a request interfering *at the client* is that the tagged download is longer. If the arrival date of the interfering request relative to the tagged request is u, the additional response time of the tagged request is $d - |u|$.

The result of a request interfering *at the server* depends on the capacity of the server. If the capacity is enough, the interference will *not* slow down the flow, and the response time will not change. This is the case for the symmetric cases in our experiments. If the capacity is not enough, the flow will be slowed down. Take the case where the capacity of servers is precisely that of one of the s parallel flows. This is the case for asymmetric cases in our experiments. If an interference occurs at the server, the flow will be slowed down to half its throughput. The result is then exactly the same as when two requests interfere at the client.

Suppose now that the arrival rate λ and the load ρ are small. Ignoring the events that happen with probability $o(\rho)$, only three events are to be considered: a) no interferences; b) one single interference at the client, none at the server; c) one single interference at the server, none at the client. According to the discussion above, we can calculate the statistics of the response time T in the two situations:

Large server capacity: the probability that $T = d$ is the probability of events a) and c), since c) does not have an influence on T. This probability is: $e^{-2\rho} = 1 - 2\rho + o(\rho)$. For $x \in [0, d]$, $P(T > d + x) = P($ b) and $|u| < d - x) = 2\rho(1 - x/d)$, and $E[T] = d(1 + \rho)$. These formulas are in accordance with Equations 8 and (6). The prediction that $P(T > d) \sim 2\rho$ can be observed on Figures 1 to 4. The linear behavior of the distribution of $P(T > x)$ for $x \in [d, 2d]$ is also clear on these figures.

Minimal server capacity: the probability that $T = d$ is the probability of event a), that is, $(e^{-2\rho})^2 = 1-4\rho+o(\rho)$. For $x \in [0, d]$, $P(T > d+x) = P($ b) or c) and $|u| < d-x) = 4\rho(1-x/d)$, and $E[T] = d(1 + 2\rho)$. These formulas are not in accordance anymore with the PS queueing model. This explains the bad results of the approximation in Table 2 for asymmetric cases. At the same time, this suggests a possible correction for the PS approximation formula. The prediction that $P(T > d) \sim 4\rho$ can however be observed on Figures 7 and 8(a) (with $\rho = 12\%$). The almost-linear behavior of the distribution of $P(T > x)$ for $x \in [d, 2d]$ is also clear on these figures.

Contents

1 Introduction and related work **3**

2 System description and notation **5**

3 Description of the algorithm **6**

4 Experimental results **8**
4.1 Parameter values 8
4.2 Simulators and Metrics 9
4.3 Results 10

5 Conclusion and future work **13**

A Approximations with Processor Sharing **20**
A.1 Distribution of the response time in the $M/D/1/PS$ queue 20
A.2 Small load approximations 21

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399