Flow-Level Modeling of Parallel Download in Distributed Systems

Abdulhalim Dandoush, Alain Jean-Marie

Flow-Level Modeling of Parallel Download in Distributed Systems

Abdulhalim Dandoush
INRIA, 2004 Route des Lucioles, BP 92,
F-06902 Sophia-Antipolis,
Abdulhalim.Dandoush@sophia.inria.fr

Alain Jean-Marie
INRIA and LIRMM, CNRS/Université Montpellier 2,
161 Rue Ada, F-34095 Montpellier
ajm@lirmm.fr

***Abstract*—Response time is the primary Quality of Service metric for parallel download systems, where pieces of large files can be simultaneously downloaded from several servers. Determining response times in such systems is still a difficult issue, because the way the network bandwidth is shared between flows is as yet not well understood. We address the issue by exploring 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" (PFFLA). 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 QoS calculations in general P2P networks.**

***Keywords*-distributed systems, performance evaluation, response time, simulation model, max-min fairness**

I. Introduction and related work

The growth of storage volume, bandwidth, and computational resources for PCs has opened the way to parallel download systems, which 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 locations. 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 the 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 retrieve 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 can be quite chaotic is some situations. Other authors on the other hand, have argued that TCP tends to share the bandwidth between flows quite *fairly*: for instance, Heyman *et al.* [1], followed by Fredj *et al.* [2].

Other studies have put forward the concepts of max-min fairness, proportional fairness, balanced fairness and utility-based resource-sharing models (see e.g., [3] 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 [4] 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: the question of how to handle different round trip times is left for future work.

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" [3] algorithm of [5]: 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. In addition, the performance metric is the global response time, not that of individual flows. 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 (up to $4{\cdot}10^5$).

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., 70%) and less than 1% for low loads in the symmetric up/down case and less than 5% respectively for relatively large loads in the system (e.g., 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 II overviews the system assumptions and notation. Section III describes the flow-level simulation algorithm "PFFLA". In Section IV, comparisons between packet-level and PFFLA are introduced and discussed. Last, Section V concludes the paper and highlights some future directions.

II. 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 peers are connected to the core network through a single link, and 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 symmetric (upload/download capacities are identical) and asymmetric situations. The following assumptions and notations will be enforced throughout the paper.

Network assumptions. The considered network consists in a set of $\mathcal{N}$ nodes: $\mathcal{N}/2$ are servers and $\mathcal{N}/2$ are clients.

Its logical structure 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 all pairs of nodes (clients or servers).

Data and traffic assumptions.

Each block of data D of size S_B is partitioned into s fragments of size S_F. 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 flows (parallel requests toward s distinct servers).

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 uniformly 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.

III. Description of the algorithm

The notion of max-min (or maximin) fairness was first introduced in the context of networking by Bertsekas and Gallager [5] as a design objective for 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 [5, p. 526]. The algorithm provided in this reference for computing the unique max-min fair allocation, has been later described as a "Progressive Filling". We have used in our analysis the following variant using the same concept. Accordingly, we named it the Progressive Filling Flow Level Algorithm (PFFLA). For more details on the max-min fairness, refer to [5, ch. 6], [3] or to our Technical Report [6].

For the purpose of formalizing the description of the PFFLA, introduce the following notation. A node (server

or client) will be represented by the link that connects it to the network core. 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:

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

Data: A set of links $\mathcal{A}$ with their capacities C_a, and a set of flows $\mathcal{F}$
Result: A throughput value for each flow
begin
  Remove from $\mathcal{A}$ nodes without flows ;
  while $\mathcal{A}$ *not empty* **do**
    foreach $a \in \mathcal{A}$ **do**
      // N_a: the number of flows crossing link a
      $N_a \leftarrow \#\{f \in \mathcal{F} | f \nabla a\}$;
    end
    // find the bottleneck link a^* and its throughput
    calculate $\theta^* = \min_{a \in \mathcal{A}} C_a / N_a$;
    calculate $a^* = arg \min_{a \in \mathcal{A}} C_a / N_a$;
    foreach f, $f \nabla a^*$ **do**
      set $\theta_f = \theta^*$;
      foreach $a \in \mathcal{A}$, $f \nabla a$ **do**
        $C_a \leftarrow C_a - \theta^*$
      end
      remove f from $\mathcal{F}$;
    end
    remove from $\mathcal{A}$ links without flows ;
  end
  return $\{\theta_f\}$
end

Algorithm 1: Algorithm PFFLA

The fact that this algorithm produces a max-min allocation satisfying the throughput constraint (1), can be checked the same way as for the Progressive Filling Algorithm: see again [6] for details.

Notice that 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. Note also that 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.

IV. Experimental results

A. Parameter values

We ran a total of eleven experiments; six in symmetric peers download/upload capacities scenarios and five in asymmetric scenarios (more experiments are described in the Technical Report [6]).

The set-up of the simulation parameters is summarized in Table I. 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 1–6, nodes are homogeneous: they have all the same network access capacity. In Experiments 7–11, 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 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 6% up to 70%.

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 link capacities values 1500/384 kbps, except in Experiment 11 where the capacities values are 2000/384 kbps. So, in all the asymmetric experiments, except in the last one, the capacity of a server is slightly larger than $1/s$ times the capacity of a client.

For the packet-level simulation, we consider a fixed constant value of 2ms for the link propagation delays. The main TCP configurations are as follows: 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 $64kB/8ms = 8MB/s$, 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:

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

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

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

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 (2) also.

In the symmetric cases, 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.

B. 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; implementation details are reported in [8].

The flow-level simulator consists in the embedding the PFFLA (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. The empirical average obtained with the packet-level and flow level simulators are denoted with $E[T_{NS}]$ and $E[T_{FLA}]$, respectively. In the view of the fact that the flow-level simulator ignores the delay for establishing and closing the TCP connections and propagation delays, there will be, for any experiment, a very small difference between the minimum values obtained from both simulators. Therefore, we denote by $\hat{E}[T_{NS}]$ the measured download time for NS, corrected by a constant value so that the minimal values for both simulators are the same. This last metric will be used later to compare the average response times in both simulators. However, we have not corrected this systematic error in the figures, presented later in this paper, for illustrative purpose.

In addition, we have compared the average document download times 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 "customer". 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. We can test easily this conjecture since the average response time in this queue is well-known to be as follows (measured in seconds, with the value of σ computed previously):

$$E[T_{PS}] \;\;=\;\; \frac{\sigma}{1-\rho} \;. \tag{3}$$

Known results on the PS queueing model also include the distribution of the response time. The relevant formulas and more comparisons are provided in [6].

We will compute the relative error (RR) between $\hat{E}[T_{NS}]$ and $E[T_{PS}]$, on one hand, and between $\hat{E}[T_{NS}]$ and $E[T_{FLA}]$ on the other hand. The relative error, for instance in the first case, is calculated as follows:

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

C. Results

For flow-level simulations, we have collected 100000 samples of the document download time in every case, whereas this number was at least of 30000 for packet-level simulations. The execution times (not fully reported here for lack of space) are: from milliseconds to minutes for PFFLA, and from 20 hours and up for NS simulation. We conclude that the flow-level algorithm is very efficient in terms of time. How good is it in term of accuracy?

To answer this question, we first depict in Figures 1 and 2 the empirical complementary cumulative distribution function (CCDF) of the block download time obtained form both simulators, and for some selected experiments.

Table I
EXPERIMENTS SETUP

Experiment number	$\mathcal{N}/2$ peers	C_d/C_u kbps	S_B/S_F MB	$1/\lambda$ sec.	ρ %
1	25	384/384	4/1	60	6
2	250	576/576	8/2	1.913	25
3	250	1500/1500	8/2	0.510	36
4	250	1500/1500	8/2	0.367	50
5	250	1500/1500	8/2	0.306	60
6	250	1500/1500	8/2	0.262	70
7	25	1500/384	8/2	59.81	12
8	250	1500/384	8/2	5.98	12
9	500	1500/384	8/2	2.99	12
10	500	1500/384	8/2	0.718	50
11	500	2000/384	8/2	0.718	50

Table II
MEASUREMENTS FOR THE PFFLA AND THE PACKET-LEVEL SIMULATION; COMPARISON WITH THE PS MODEL

Ex. nb	$\hat{E}[T_{NS}]$ sec.	$E[T_{FLA}]$ sec.	$RR\%$ NS/FLA	$E[T_{PS}]$ sec.	$RR\%$ NS/PS
1	96.062	95.45	0.6%	95.44	0.6%
2	161.252	160.196	0.6%	166.132	-3%
3	73.547	73.346	0.2%	71.7692	2.4%
4	99.501	97.75	1.7%	91.864	7.6%
5	129.066	127.691	1%	114.83	11%
6	176.45	180.05	-2%	153.107	13.2%
7	61.137	62.901	-2.8%	52.19	17%
8	64.738	64.935	-0.3%	52.19	19.3%
9	65.298	65.182	-0.18%	52.19	20%
10	144.615	149.213	-3.1%	91.865	36%
11	142.1	149.213	-5%	68.45	51.8%

We report then in Table II the expected block download time obtained from both simulation levels and from the PS formula (3) for all experiments, together with relative errors.

The results show that for small system load, the download time predicted by the PFFLA fits exactly that of NS-2, for average values as well as for distributions. The relative error between the average values is very small as shown in Table II. 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. The various phenomena, which typically perturb the throughput of TCP (slow-start phase, packet losses, buffer fluctuations) happen very rarely of have little influence in this case.

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, the RR is less than 2% in the symmetric case and less than 5% in the asymmetric case. It is clear from Figure 1(b) that the distributions measured by both simulators are different in the symmetric case for very high load, but average values turn out to be almost identical. The same observation holds for asymmetric cases, see Figure 2(b).

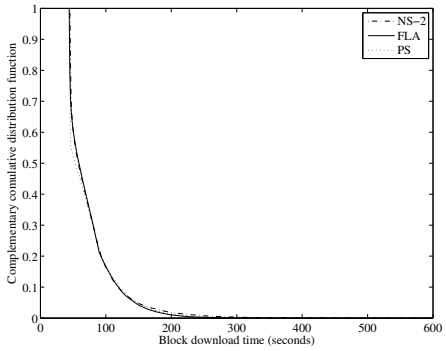
The accuracy of the PS approximation for the average download time is acceptable for symmetric cases up to $\rho = 36\%$, and degrades above $\rho = 50\%$. The accuracy for the complete distribution can be assessed on Figure 1(a) for a load of 36%. In the asymmetric cases, the approximation is bad at 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 client 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 $s = 4$ and the event rarely happens, even for moderate loads. In the asymmetric case, this value is close to 1 and the slowdown is much more frequent. The PS queueing model breaks down when such events occur, which explains the bad accuracy of $E[T_{PS}]$.

Another observation is that larger the network size, better the performance of PFFLA and worse the performance of PS model as illustrated by Experiments 7, 8 and 9. The number of peers in these experiments are 25, 250 and 500 respectively for same load and capacities. However, the relative error occurred in Experiment 9 is less than that of 8, which is, in turn, less than that of Experiment 7. Indeed, the accuracy does not depend only on the system load but also on the number of peers and their capacities.

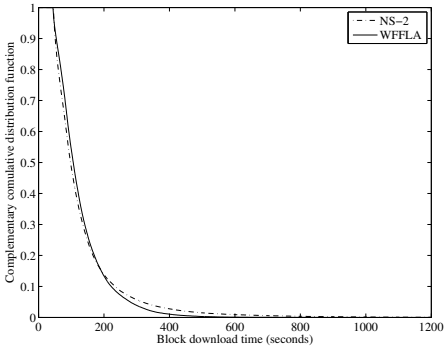
Clearly, the larger the buffer sizes, the better the performance in real networks. To address this point, we depict in Figure 3 the CCDF of download times for two values of the queue limit (100 and 500 packets) in the NS-2 simulation. Other parameters are: 50 nodes, $C = 1500kbps$ and $\rho = 70\%$. The relative difference between the two average download times is 6.56%. So, indeed, the buffer size can affect the performance of the system at high loads.

V. CONCLUSION AND FUTURE WORK

In this paper, we have proposed and analyzed the PFFLA. The algorithm is quite simple and uses the concept of "Progressive-Filling" (or max-min fairness). We have implemented it 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). Our results conclude that PFFLA is a fast and 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. 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.

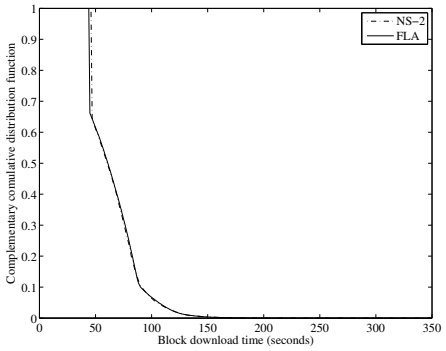


(a) $\rho = 36\%$, $\mathcal{N}$=500.

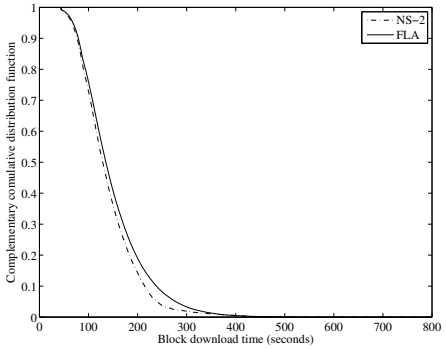


(b) $\rho = 60\%$, $\mathcal{N}$=500.

Figure 1. Experiments 3 and 5: PFFLA vs NS-2.



(a) $\rho = 12\%$, C_d=1500kbps, $\mathcal{N}$=1000.



(b) $\rho = 50\%$, C_d=2000kbps, $\mathcal{N}$=1000.

Figure 2. Experiments 9 and 11: PFFLA vs 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 the PFFLA to handle this situation.

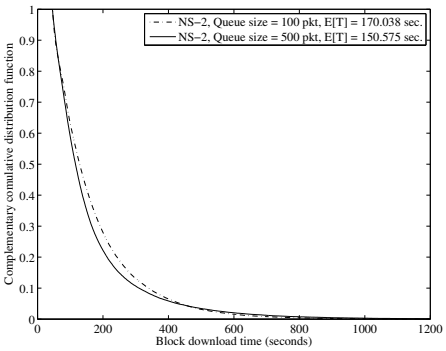


Figure 3. Queue size effect for $\rho = 70\%$, $\mathcal{N}$ = 50

REFERENCES

[1] 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 *Proc. of 1997 ACM SIGMETRICS Intl. Conf. on Measurement and modeling of comp. systs.*, New York, USA, 1997, pp. 24–38.

[2] S. Ben Fredj, T. Bonald, A. Proutière, G. Régnié, and J. W. Roberts, "Statistical bandwidth sharing: a study of congestion at flow level," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 111–122, 2001.

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

[4] T. Bonald and A. Proutière, "Insensitive bandwidth sharing in data networks," *Queueing Systems*, vol. 44, pp. 69–100, 2003.

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

[6] A. Dandoush and A. Jean-Marie, "Download Process in Distributed Systems, Flow-level Algorithm vs. Packet-level Simulation Model," INRIA, Research Report RR-7159, 2009, last accessed: 24 Mar 2010. [Online]. Available: http://hal.inria.fr/inria-00442030/en/.

[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] 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)*, Pisa, Italy, October 19 2009.