Practical Performance Evaluation of Ethernet Networks with Flow-Level Network Modeling

Fabien Geyer[1,2] Stefan Schneele[1] Georg Carle[2]

[1]EADS Innovation Works
Dept. IW-SI-CP
D-81663 München, Germany
{fabien.geyer,stefan.schneele}@eads.net

[2]Technische Universität München
Institut für Informatik, I-8
D-85748 Garching b. München, Germany
carle@in.tum.de

ABSTRACT

Network models for evaluating the behavior of networks are important tools in traffic engineering for dimensioning networks and provisioning bandwidth for different applications. We present in this paper a flow-level network model for the performance evaluation of IP networks with support of long-lived TCP and UDP flows. While flow-level network models for TCP and UDP flows have already been investigated, a vast majority of previous studies often do not take into account the importance of cross-traffic. This paper presents topologies where cross-traffic has a major impact on the performance of TCP flows and shows how previous models are not accurate enough. We consider in our study Ethernet LANs with low latencies and show how to apply our framework to networks with Ethernet switches using priority based scheduling, fair-queuing scheduling, or hierarchical scheduling based on the former algorithms. We assess the accuracy of our approach by comparing the results of our model with results of the discrete event simulator OMNeT++.

Categories and Subject Descriptors

I.6 [**Computing Methodologies**]: Simulation and Modeling; C.2 [**Computer Systems Organization**]: Computer-Communication Networks

General Terms

Theory, Measurement, Performance

Keywords

Flow-level network modeling, Network traffic modeling, Performance evaluation

1. INTRODUCTION

In the last few decades, analog functions and isolated processing devices are increasingly being replaced with numeric and interconnected devices with the support of Ethernet and TCP/IP based networks. In order to function correctly, those devices and applications are in demand for efficient and predictable network performance. Traffic engineering aims at bringing an answer to this need by avoiding congestion and optimizing network layout to support an increasing number of applications.

Network models are an important part of this process in order to evaluate how a network will behave. Those models should be able to analyze different types of network protocols, among which TCP and UDP based applications. Different techniques have been developed for this purpose, each with their advantages and drawbacks.

Accurate discrete-event simulators and models are often used for this task, where we can cite OMNeT++[3] or ns-3 [2] as two well-known open-source packet simulation tools used by the network research community. While simulations produce accurate results as it aims at replicating the different processes taking place in a network, it often fails at scalability and efficiency.

Mathematical frameworks have been proposed for the deterministic study of networks, such as the ones presented in [12] and [21], also known as Network Calculus. Compared to simulations, such models have the major advantage of bringing deterministic behavior in a network and thus enabling real-time communications in highly critical environments. But it comes at the cost of using restricted types of flows, which do not include elastic flows, meaning flows adapting their behavior to the network conditions, such as protocols with congestion control like TCP.

In order to evaluate the performance of elastic traffic, flow-level network models have been proposed as an efficient alternative to discrete-event simulation and are used on large networks topologies. As illustrated later, such modeling achieve an accuracy comparable to simulation, but with a smaller calculation overhead.

We propose in this paper a framework for evaluating the steady-state performance of long-lived TCP and UDP flows in the context of Ethernet LANs. While previous studies in the domain of flow-level network modeling are often neglecting the impact of cross-traffic, we demonstrate that it can lead to major errors on the evaluation of performances of TCP on specific topologies. This phenomenon of cross-traffic is well known in traffic engineering and was firstly attributed to ACK compression in [32]. More recently [19] proposed the principle of data pendulum to explain it.

Our solution takes into account this phenomenon by including TCP acknowledgments into our flow-level network

ValueTools'13, December 10 – 12 2013, Turin, Italy

model. With our framework, we aim at evaluating Ethernet LANs where nodes communicate using TCP or UDP, and give the following results: average throughput, end-to-end delay and loss probability. We also extend our approach to network supporting Strict-Priority Queuing (SPQ), packetized versions of Generalized Processor Sharing (GPS) such as Weighted Fair Queuing (WFQ) [27], as well as hierarchical scheduling based on the former algorithms.

This framework was developed for the performance evaluation of large network topologies using standard TCP protocols for doing traffic engineering. But it may also be used for other purposes such as the investigation of the impact of mixed congestion control protocols on the same network.

This work is structured as follows. In Section 2, we present similar research studies. Section 3 highlights the basic principles of our framework, while details about flow modeling are introduced in Section 4, and details about FIFO queues and schedulers are given in Section 5. We present in Section 6 our algorithm for finding a solution to the model. With Section 7, we evaluate our framework across different topologies where we highlight the flaws of a model not taking into account cross-traffic. Finally, Section 8 summarizes and concludes our work, and gives an overview of future improvements for our framework.

2. RELATED WORK

Flow-level modeling is based on previous effort on TCP packet-level models, where the throughput of a TCP connection is defined as a function of loss probability and round-trip time (RTT). The two prominent packet-level models are the so-called square-root formula [23], and the PFTK formula [26].

Using those packet-level models, flow-level models have been developed using a fixed point evaluation in order to evaluate the steady-state throughput of multiple TCP flows in various topologies. Gibbens *et al.* proposed one of the early model on this subject in [16] using the square-root formula for the evaluation of TCP flows. Separately, Firoiu *et al.* as well as Bu *et al.* proposed a similar model based on the PFTK formula in [13] and [10] for arbitrary networks with TCP and non-TCP flows and RED queue management. Altman *et al.* proposed various mathematical formalisms and proofs to flow-level models in [5], by building on the results of [10]. The models previously cited were extended by Hassan *et al.* in [18] to include scheduling algorithms, namely priority queuing and weighted fair queuing.

Velho *et al.* noted in [30, 31] that previous work on flow-level modeling did not include the effect of cross-traffic on TCP flows. They proposed a solution to overcome this problem by including TCP acknowledgments flows into a fixed point formulation using a RTT-aware max-min model. While the solution proposed in [31] seems appropriate for the proposed use cases, it is not clear if the evaluation of the TCP model takes account of other behavior of TCP than RTT-unfairness, such as TCP timeouts. Indeed, the work that lead to the PFTK formula showed that TCP timeouts have a significant impact on TCP sending rate. We present in this paper a solution to the cross-traffic problem based on the early work presented in [13].

Separately to the evaluation of the steady-state behavior of TCP flows, researchers also focused on models for the dynamic behavior of TCP traffic. Misra *et al.* described in [24] the behavior of TCP flows using a set of coupled ordinary differential equations. This formulation was then extended by various researchers such as the work presented in [25] and [22]. Similarly to the work on flow-level modeling, the influence of cross-traffic was only taken later into account, such as the work presented in [8].

3. FRAMEWORK FOR FLOW-LEVEL NETWORK MODELING

3.1 Elements of the studied network

We define the following assumptions for the topologies studied in this paper. We target the performance evaluation of Ethernet Local Area Networks (LANs) where entities communicate using standard Ethernet. Computers are interconnected through Ethernet switches and communicate with each other either by using protocols on top of TCP, or by using fixed rate flows (streaming) which is considered here to be UDP based. For the scope of this paper, we consider that all communications are unicast and that the routing is static, meaning that we have a single path between a source and a destination.

The network is composed of Ethernet switches functioning on the principle of store-and-forward, meaning that switches need to first receive and store the complete frame before being able to forward it, as opposed to the principle of cut-through. Links between nodes of the network are assumed to be Ethernet cables, and can have different link speed. A switch can have an internal processing delay for each frame. As we study Ethernet LANs with low latencies, meaning networks where queuing delay has a large influence on end-to-end delays, we do not neglect queuing delay in switches.

When discussing packet size and flow throughput in the rest of the paper, we consider them from the Ethernet point of view. In order to also take into account the preamble, start of frame delimiter and interframe gap of Ethernet, the packet size shall account for it.

3.2 Flow-level network model

Our flow-level network model consists of *servers*, which model the different queues of the network, as well as *flows*, which represent the communications between the nodes of the network.

We define a *server* as an entity receiving packets and forwarding them on a link. A server, noted here s_k with $k \in \mathbb{N}$, is defined by the following parameters: C_k is the maximum output bandwidth, D_k is an additional delay (which can be used to model propagation and processing delay), $F_k = \{f_n\}_k$ is the set of flows going through this server, Q_k the buffer size of the server as the result of the function $H_k^Q(F)$ depending on a set of flows F, p_k the drop probability of the server as the result of the function $H_k^p(F)$ depending on a set of flows F. Details about the functions H_k^Q and H_k^p depend on which model to use and will be described in Section 5.

We define a *flow* as a sequence of packets sent from a particular source to a particular unicast destination of a specific transport connection or media stream. A flow, noted here f_i with $i \in \mathbb{N}$, is defined by the following parameters: $S_i = \{s_n\}_i$ the path of servers traversed by the flow from source to destination, and r_i the bandwidth of a flow at its source as the result of the function $\rho_i(S)$ depending on the path of servers S. We also define the throughput of a flow as the rate of successful message delivered to the destination. According to this definition, if a protocol is specified by requests and replies, two flows have to be used. We also define $\overline{S_i}$ as as the path which will be used for the reply packets of flow f_i. Details about the function ρ_i depend on which model to use and will be described in Section 4.

Based on those parameters, we describe the behavior of a network using the axioms presented hereafter.

Axiom 1. *The end-to-end drop rate $e2e_p$ of the path of servers S is defined by:*

$$e2e_p(S) = 1 - \prod_{k \in S} (1 - p_k) \tag{1}$$

Axiom 2. *The aggregated ingress bandwidth of server s_k is defined by the sum of bandwidth of the set of flows F_k traversing the server:*

$$B_k^{inp} = \sum_{i \in F_k} \left[r_i \cdot (1 - e2e_p(\mathcal{U}(S_i, s_k))) \right] \tag{2}$$

where $\mathcal{U}(S_i, s_k)$ corresponds to the set of servers the flow i traverses before reaching s_k.

We account in Equation (2) for the fact that part of the bandwidth of the traversing flows is already dropped on the different paths leading to the studied server (noted here $\mathcal{U}(S_i, s_k)$)).

Axiom 3. *The egress bandwidth of server s_k is equal to:*

$$B_k^{out} = (1 - p_k) \cdot B_k^{inp} \tag{3}$$

and must satisfy the constraint:

$$B_k^{out} \le C_k \tag{4}$$

Axiom 4. *The end-to-end delay $e2e_D$ of a frame of size M along the set of servers S is defined by:*

$$e2e_D(S, M) = \sum_{k \in S} ((M + Q_k) \cdot C_k + D_k) \tag{5}$$

We account in Equation (5) for the forwarding time of the frame ($M \cdot C_k$), the time needed to process the queue ($Q_k \cdot C_k$) as well as an additional delay D_k for modeling propagation and processing delay.

Axiom 5. *The round-trip delay time for a flow with a request frame of size M_{req} and a reply size of M_{rsp} is:*

$$RTT(S, M_{req}, M_{rsp}) = e2e_D(S, M_{req}) + e2e_D(\overline{S}, M_{rsp}) \tag{6}$$

4. FLOW MODELS

The goal of the flow model is to define the function $\rho_i(S)$ representing the bandwidth of the flow as a function of the set of servers S traversed by the flow. We present in this section the flow modeling for two types of flows: constant bitrate flows representing multimedia streaming based on UDP, and long-lived TCP flows.

4.1 Long-lived TCP flow model

The congestion control algorithm of TCP works in two different phases. The first phase, called *slow start* as in RFC 5681 [4], occurs at the beginning of the TCP connection and is used to estimate the link capacity. During this phase, only

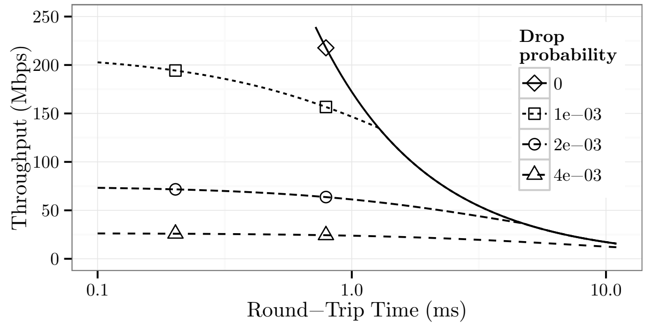


Figure 1: Maximum bandwidth of TCP based on the approximate PFTK model, with $MSS = 1518$B, $W = 14$, $T_0 = 1$s, $b = 2$

a small amount of data is transmitted. Once this phase is finished, a *congestion avoidance* phase takes place and transmits the rest of the data. For this study, we consider that TCP is used to transfer large data, meaning that we only account for the *congestion avoidance* phase, and we call this type of flows *long-lived TCP flows*.

Although various TCP congestion-avoidance algorithms have been developed, we limit this study to TCP Reno [20]. Other congestion-avoidance algorithms may be included following the same methodology presented here. We define W as the maximum window size of a TCP connection, in number of packets.

Axiom 6. *In case of a network without loss ($e2e_p(S) = 0$), the average bandwidth of TCP is limited by:*

$$\rho_{e2e_p(S)=0}(S) = \frac{MSS \cdot W}{RTT(S, MSS, M_{ACK})} \tag{7}$$

with MSS the maximum segment size, W the maximum windows size and M_{ACK} size of a TCP ACK packet.

We note that we already use the size of an ACK packet for the RTT in Equation (7) in order to have a better accuracy of the model.

In case of packet loss, we use the bandwidth model developed in [26], also known as the PFTK formula which models the bandwidth of the TCP Reno protocol. We use here the approximated version of the PFTK formula, where the bandwidth of TCP connection is defined as the minimum of Equations (7) and (8).

$$\frac{MSS}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1 + 32p^2)} \tag{8}$$

with p the drop probability, T_0 the sender timeout delay, and b the number of packets that are acknowledged by a received ACK.

We illustrate the bandwidth of TCP as a function of RTT and drop probability as presented in the Equations (7) and (8) in Figure 1.

4.2 Improved TCP model with ACKs

As illustrated later with the evaluation of topologies with cross-traffic in Section 7, the model presented before does not take into account the impact of cross-traffic on the bandwidth of a flow which can lead to significant errors. This problem comes from the fact that we modeled the TCP data flow as unidirectional, where a real TCP flow has acknowledgments (ACK) which can be affected by cross-traffic.

In this improved model, we consider that a TCP connection is made of two flows: the TCP data flow, and the TCP ACK flow. We consider that the raw bandwidth of the ACK flow corresponds to a certain fraction ϵ of the bandwidth of the Data flow: $\epsilon \cdot \rho_{data}$. We derive ϵ from the ratio of frames sizes between an ACK packet and a data packet, as well as b the number of packets that are acknowledged by a received ACK. For the numerical results presented later in Section 7, we choose $\epsilon = \frac{84\text{B}}{1538\text{B} \cdot b} \approx 5\%$, with $b = 1$.

Axiom 7. *In order to account for cross-traffic, the bandwidth of the ACK flow ρ^{ACK} and the TCP Data flow ρ^{Data} are constrained by the following set of equations:*

$$\rho_{data}(S_{data}) \leq \mathcal{M}_{TCP}(S_{data}) \tag{9}$$
$$\rho_{ACK}(S_{ACK}) \leq \mathcal{M}_{TCP}(S_{ACK}) \tag{10}$$
$$\rho_{ACK}(S_{ACK}) = \rho_{data}(S_{data}) \cdot \epsilon \tag{11}$$

with S_{data} the path of the data packets, S_{ACK} the path of the ACK packets, and $\mathcal{M}_{TCP}(S)$ the value of the basic TCP model (PFTK formula Equation (8)).

With this set of equations, we specify the dependencies between the bandwidth of the TCP data flow and the TCP ACK flow. With Equation (9) we constrain the bandwidth of the TCP data flow by the TCP bandwidth model on the path of the data packets $\mathcal{M}_{TCP}(S_{data})$. Similarly, with Equation (10) we constrain the bandwidth of the TCP ACK flow by the TCP bandwidth model on the path of the ACK packets $\mathcal{M}_{TCP}(S_{ACK})$. We take into account with this equation the effects of other flows on the path of the ACK packets (S_{ACK}) which corresponds to the cross-traffic, as well as asymmetric bandwidth. Finally we establish the relation between the TCP data and TCP ACK bandwidth with Equation (11).

When the ACK flow is affected by cross-traffic and has a reduced bandwidth due to Equation (10), it has a direct impact on the bandwidth of the data flow using Equation (11).

4.3 Constant bitrate streaming flow model

Axiom 8. *For a flow f with constant bitrate (CBR) b without feedback or bandwidth adaptation, the bandwidth model can be expressed as:*

$$\rho(S) = b \tag{12}$$

This model is used for representing multimedia streaming flows based on UDP. As we model such flows with no feedback loop, the bandwidth of the flow is simply a constant value independent of the path.

5. SERVER MODEL

Regarding our framework, a server corresponds to a queue in the network. Queues can be directly connected to an Ethernet physical interface or be regulated by a scheduler. Our model is able to support different types of scheduling algorithms. In this paper, we describe the following elements constituting a server:

- Drop tail First-In-First-Out (FIFO) queue,

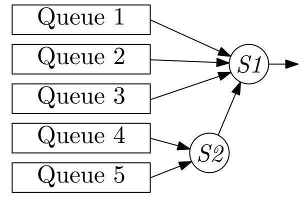


Figure 2: Example of hierarchical scheduling with two schedulers *S1* and *S2*

- Drop tail FIFO queue with traffic shaping,
- Strict Priority Queuing (SPQ) scheduling,
- Approximations and packetized versions of Generalized Processor Sharing (GPS) scheduling,
- Hierarchical scheduler based of SPQ and GPS, as illustrated by Figure 2.

Although we restrict this study to the aforementioned elements, other algorithms may be used, such as for instance Random Early Detection (RED) [14] which is often used in previous literature about flow-level network modeling, such as for instance in [13].

As defined earlier, a server is parameterized by C its maximum output bandwidth, D its additional delay, $F = \{f_n\}$ the set of traversing flows, Q its queue size specified by the function $H^Q(F)$, and p its drop probability specified by the function $H_k^p(F)$ depending on a set of flows F. The purpose of this section is to define the queue size function H^Q and drop probability function H^p of the queues. We consider that D the additional delay used for modeling propagation and processing delay is a constant value. More advanced models may define D as a function of the packet size or the usage of the server.

Schedulers regulate the queues by allocating a specific bandwidth limit to the queues according to their available bandwidth limit $C_{scheduler}$. To increase the accuracy of our model, the scheduler model should also adjust the additional delay due to the non preemptive property of Ethernet, but we ignore it in the context of this paper.

5.1 Drop-tail First-In-First-Out queue

With a drop-tail FIFO queue, packets are served in their order of arrival. When the queue has no more space available for storing arriving packets, packets are simply dropped.

Previous research based the modeling of a queue on queuing theory, such as the work presented in [16] or [6] which used a M/M/1/K queue and the assumption that TCP packets arrive following a Poisson process. We propose to use here a simpler model which does not make any assumption on the input traffic.

The bandwidth available to the queue is noted C_Q.

5.1.1 *Packet drop function* $H^p(F)$

We consider here that the queue drop packets as soon as the incoming bandwidth is superior to the allowed output bandwidth.

Axiom 9. *The packet drop function of a drop-tail FIFO queue is expressed as followed:*

$$H^p(F) = \frac{\left[B^{inp} - C_Q\right]^+}{B^{inp}} \tag{13}$$

with $[x]^+ = x$ *if* $x \geq 0$, *and* 0 *otherwise.*

Equation (13) guarantees that $B^{out} \leq C_Q$ as defined in Equation (4).

5.1.2 Queue size function $H^q(F)$

We model the queue size as follows:

Axiom 10. *The queue size function of a drop-tail FIFO queue is expressed as followed:*

$$H^q(F) = \begin{cases} M_Q & \text{if } B^{inp} > C_Q \\ \max\left\{q \middle| B^{out}(q) = \max B^{out}\right\} & \text{otherwise} \end{cases} \tag{14}$$

The queue is considered to be full (and equal to the maximum buffer size M_Q) when the incoming bandwidth is superior to the allowed output bandwidth. This is model by the first case of Equation (14).

When the queue is not full, the queue size will depend on how many packets may be transfered by the flows. As presented in Section 4.1 and Figure 1, TCP is able to fully utilize a link up to a certain limit of the round-trip time, or in other words by the buffer size of the different queues traversed by the flow. This means that the bandwidth of TCP, and hence B^{out}, is a function of the queue size. The queue size corresponds then to the maximum number of bits that has no impact on the bandwidth of the flows going through the queue. This is model by the second case of Equation (14).

5.2 Queue with bandwidth limiter

This model is the same as the FIFO drop tail queue, but here we modify the parameter C_Q of the server to allow a lower bandwidth than the available link bandwidth.

5.3 Strict Priority Queuing

Strict Priority Queuing (SPQ) is a scheduling algorithm where each queue is assigned a priority. The algorithm works as follows: all queues are polled in their priority order, until a non-empty queue is found and served. This process is restarted each time a packet needs to be dequeued.

This means that a queue served by SPQ can use the bandwidth that was unused by the queues of higher priorities.

Axiom 11. *When the SPQ scheduler has an available bandwidth of* $C_{scheduler}$*, each server* s_k *served by SPQ (queue or other scheduler) , with* k *from* 0 *(highest priority) to* N_q *(lowest priority), has the following available bandwidth:*

$$C_k = \begin{cases} C_{scheduler} & \text{if } q = 0 \\ \left[C_{scheduler} - \sum_{i<k} B_i^{out}\right]^+ & \text{otherwise} \end{cases} \tag{15}$$

with $[x]^+ = x$ *if* $x \geq 0$, *and* 0 *otherwise.*

We describe in Equation (15) that the queue with the highest priority ($q = 0$) has access to the all the available bandwidth ($C_{scheduler}$), while the rest of the queues have access to the bandwidth that is unused by the queues of higher priority.

5.4 Generalized Processor Sharing

With Generalized Processor Sharing (GPS), each queue i has a weight w_i, and is allocated the following bandwidth:

$$\mathcal{B}_i = \frac{w_i}{\sum_{j \in \mathcal{Q}} w_j} \tag{16}$$

with $\mathcal{Q}$ the set of queues that currently hold packets. When a queue is using less that its allocated bandwidth, the remaining bandwidth is redistributed to the other queues, according to their respective weights.

This model corresponds to a simplification of Weighted Fair Queuing (WFQ) [27], Worst-Case Fair Weighted Fair Queuing (WF^{2}Q) [9], Deficit Round Robin (DRR) [28] or similar packet scheduling algorithm with proportional fairness with regards to the bandwidth.

Axiom 12. *In order to compute the allocated bandwidth of each queue, we use the following iterative process. We use the index* n *to mark the iteration step. We define* $\mathcal{R}^n$ *as the remaining unused bandwidth,* C_q^n *the bandwidth allocated to queue* q*, and* $\mathcal{Q}^n$ *as the set of queues using more than their currently allocated bandwidth* C_q^n*. We defined the following initial values:*

$$C_q^{n=0} = 0, \forall q \in \mathcal{Q}^{n=0} \tag{17}$$

$$\mathcal{Q}^{n=0} = \{q | 0 \leq q \leq N_q \text{ and } F_q \neq \varnothing\} \tag{18}$$

$$\mathcal{R}^{n=0} = C_{scheduler} \tag{19}$$

$Q^{n=0}$ *corresponds to all queues with traversing flows as the initially allocated bandwidth is 0. We run the following iterative process until* $\mathcal{Q}^n = \varnothing$ *or* $\mathcal{R}^n = 0$*:*

$$C_q^{n+1} = C_q^n + \frac{w_q}{\sum_{i \in \mathcal{Q}^n} w_i} \cdot \mathcal{R}^n, \forall q \in \mathcal{Q}^n \tag{20}$$

$$\mathcal{Q}^{n+1} = \left\{q : B_q^{out} \geq \frac{w_q}{\sum_{i \in \mathcal{Q}^n} w_i} \cdot \mathcal{R}^n\right\} \tag{21}$$

$$\mathcal{R}^{n+1} = \sum_{q \in \mathcal{Q}^n} \left[\frac{w_q}{\sum_{i \in \mathcal{Q}^n} w_i} \cdot \mathcal{R}^n - B_q^{out}\right]^+ \tag{22}$$

We define in Axiom 12 an iterative process. In the first iteration of the process ($n = 1$), we allocate the total bandwidth of the scheduler $C_{scheduler}$ to the non-empty queues following their respective weights. At each step of the iteration, we then determine how much of the bandwidth is unused with $\mathcal{R}^n$. We allocate this bandwidth to the set of queues $\mathcal{Q}^n$ which are using more than their allocated bandwidth according to their respective weights. We iterate the process until all the bandwidth is used ($\mathcal{R}^n = 0$) or there are no more queues able to use the unused bandwidth ($\mathcal{Q}^n = \varnothing$).

5.5 Hierarchical scheduling

As noted earlier, our model for a scheduling algorithm redistributes its available bandwidth $C_{scheduler}$ to the queues according to the output bandwidth B^{out} of the queues. Hence when using hierarchical scheduling, as illustrated by Figure 2, a scheduler acts on the bandwidth of a sub-scheduler in the similar way it acts on a queue.

In the hierarchical scheduler presented in Figure 2, $S1$ will allocate some bandwidth to $S2$ in the same way as it allocates it to Queue 1 to 3. Then $S2$ will redistribute this bandwidth to Queue 4 and Queue 5.

6. SOLVING THE MODEL

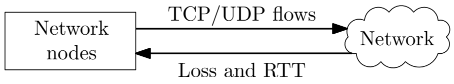


Figure 3: Congestion control

As presented in Figure 3 and based on the different models previously described, we have the following relation: flows react on network changes by adjusting their packet sending rate, while the network reacts on flows by queuing and dropping packets.

The performance evaluation of the system is equivalent to finding the values Q_k, p_k and r_i of the different servers and flows which lead to an equilibrium or fixed point of the system described by the different axioms previously enumerated.

Algorithm 1 describes the procedure to find the equilibrium of the system. We distinguish two parts in the algorithm. The first part (lines 1 to 5) initializes the variables Q_k, p_k and r_i to 0. The second part (lines 6 to 13) evaluates the functions until the fixed point is reached.

While a proof of existence of an equilibrium point was already given in [5] for TCP flows, we define a safeguard function in order to avoid an infinite loop (line 12) in case an equilibrium cannot be reached, as we don't necessarily limit our framework to TCP flows using TCP Reno. The simplest function to achieve this is to limit the number of iteration of the loop (line 6 to 13). An alternative way is to look at the evolution of Q_k, p_k and r_i, and determine if an equilibrium is reachable.

Algorithm 1 Equilibrium algorithm

Require: Set of servers S
Require: Set of flows F
1: **for all** $k = 0 : |S|$ **do**
2: $\quad Q_k \leftarrow 0$
3: $\quad p_k \leftarrow 0$
4: **for all** $i = 0 : |F|$ **do**
5: $\quad r_i \leftarrow 0$

6: **while** equilibrium not reached **do**
7: $\quad$ **for all** $k = 0 : |S|$ **do**
8: $\quad\quad Q_k \leftarrow H_k^Q(F_k)$
9: $\quad\quad p_k \leftarrow H_k^p(F_k)$
10: $\quad$ **for all** $i = 0 : |F|$ **do**
11: $\quad\quad r_i \leftarrow \rho_i(S_i)$
12: $\quad$ SAFEGUARD() $\triangleright$ Function to avoid infinite loop
13: **end while**

7. EVALUATION

We evaluate in this section different topologies. When not otherwise specified, we consider that the links between nodes are full-duplex, using a 10m Ethernet cable, with a propagation delay of $5 \cdot 10^{-8}$s, and a link speed of 100Mbps. All elements of the network are considered to have no internal processing delay. Ethernet switches have an internal drop-tail queue with a default maximum number of 10 packets for each port. Computers are considered to have no queue and no scheduling element for the egress part of the Ethernet interface.

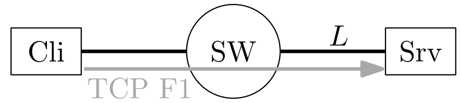


Figure 4: First topology with a variable latency L between *SW* and *Srv*

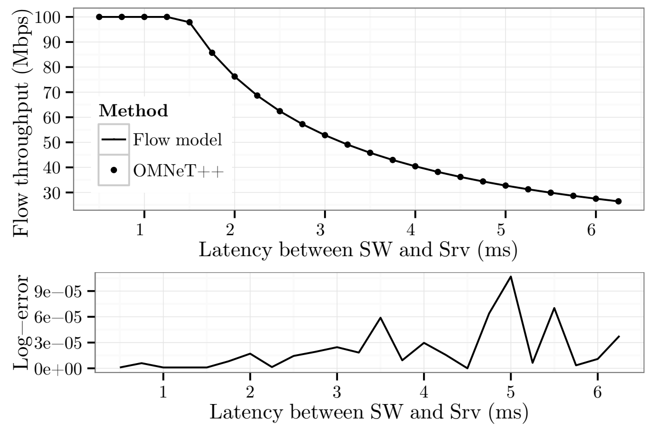


Figure 5: Comparison between the flow model and the OMNeT++ results for the topology presented in Figure 4, with the bandwidth (above) and the log-error (below)

For the configuration of TCP, we define a maximum window size W of 14 packets, a maximum segment size of 1538 Bytes (Ethernet frame size with preamble and interframe gap) and a timeout time T_0 of 1s. b, the number of packets acknowledged by an ACK, is set to 1. As noted earlier, we use here TCP Reno.

We use the results of the discrete event simulator OMNeT++ [3] and its framework INET [1] as a comparison for our model. The standard modules `StandardHost` and `EtherSwitch` from the INET framework were used for modeling computers and switches, and we configure the TCP stack to use the values noted earlier.

To evaluate the difference between the flow-level network model and the results of the OMNeT++ simulation, we use the log-error of the throughput of flow f_i, as defined in [30]:

$$LogErr(f_i) = |log(r_i^{\text{Flow model}}) - log(r_i^{\text{Simulation}})| \quad (23)$$

7.1 Validation of the TCP model

In order to validate the behavior observed in Figure 1, we evaluate a simple topology where two PCs, *Cli* and *Srv*, are connected to the switch SW, as presented in Figure 4. We define the latency of packets going from SW to *Srv* as a parameter for this study.

The bandwidth of the TCP flow between *Cli* and *Srv* is presented in Figure 5. The log-error between the results of our model and the results of OMNeT++ suggests that the flow-model is indeed relevant regarding the influence of round-trip time.

7.2 Dumbbell topology without cross-traffic

We study here the influence of asymmetrical latency on a dumbbell topology, as illustrated by Figure 6. All links have the same delay, except for packets going from *SW1* to *Srv2*,

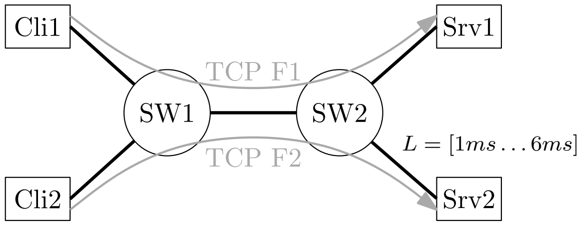


Figure 6: Dumbbell topology without cross-traffic

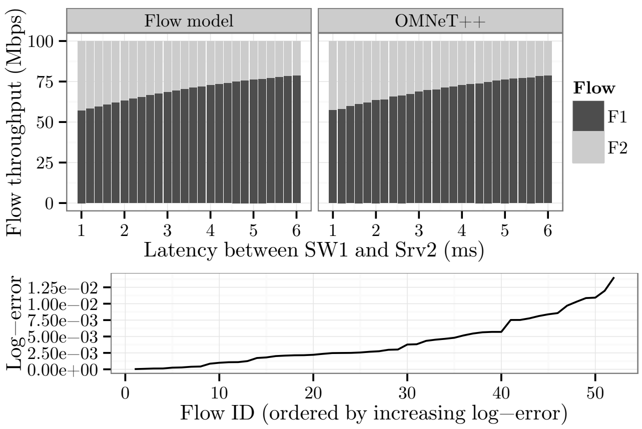


Figure 7: Comparison between the flow model and the OMNeT++ results for the dumbbell topology (see Figure 6), with the bandwidth (above) and the log-error (below)

experiencing a delay between 1 and 6ms. The maximum number of packets for the queues inside *SW1* and *SW2* is set to 30.

The individual bandwidth of each flow for this topology are presented in Figure 7. As expected, we do not see a fair sharing of the bandwidth between Flow *F1* and Flow *F2*, as it is known that TCP Reno favors flows with a lower round-trip delay time.

7.3 Dumbbell topology with cross-traffic

We study in this case the effect of cross-traffic on TCP flows. We use the same dumbbell topology as in Section 7.2, but we add TCP flow *F3* from node *Srv2* to node *Cli1*, as presented in Figure 8.

We first present the results of this topology using the TCP model without the ACK flows in Figure 9. Results for flows *F1* and *F2* are comparable to the one presented in the previous topology. But for flow *F3*, we see that the results of the flow-level network model do not match the results from OMNeT++. Indeed, the effect of cross-traffic is visible here: *F3* is not able to fully use the bandwidth available between *Srv1* and *Cli1* although all links are full-duplex. The throughput is equal to only about half the available bandwidth, because the acknowledgments of *F3* are competing with the packets of *F1* and *F2*. This phenomenon is well known in the literature, first explained by ACK compression in [32], and more recently by the principle of data pendulum in [19]. Techniques exist to overcome this problem such as in RFC 3449 [7], where a simple solution is to schedule the TCP ACK packets with a higher priority than the TCP data packets.

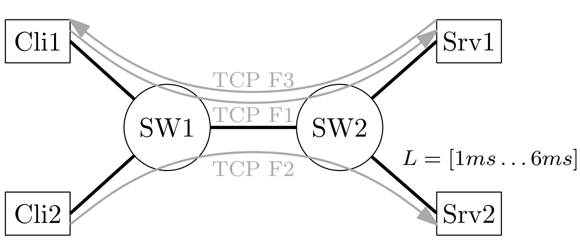


Figure 8: Dumbbell topology with crosstraffic

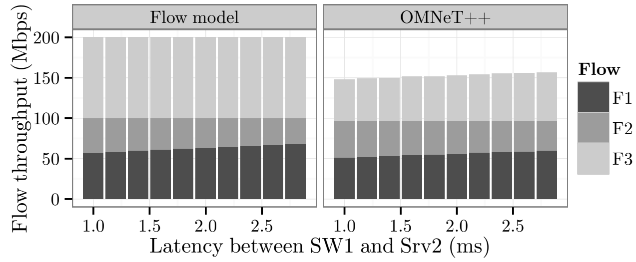


Figure 9: Dumbbell topology with cross-traffic: throughput of the TCP flows without taking into account the TCP ACKs packets

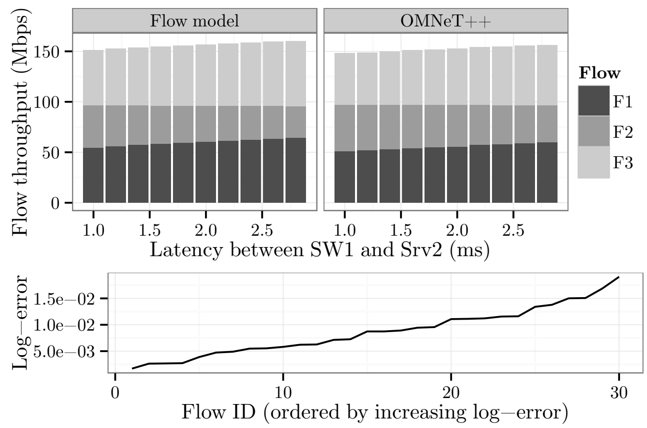


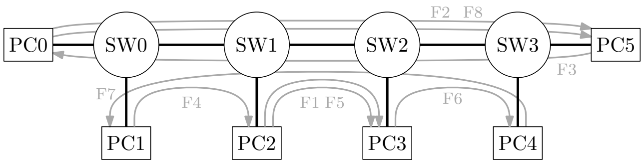
Figure 10: Comparison between the flow model and the OMNeT++ results for the dumbbell topology with crosstraffic (see Figure 8) with the improved model taking into account TCP ACKs, with the bandwidth (above) and the log-error (below)

As explained earlier, previous work on flow-level network model often neglect this problem by studying only topologies where there is no cross-traffic, and the models proposed will give a similar error as in Figure 9.

By using the improved TCP model presented in Section 4.2, we obtain the same behavior as in OMNeT++, as shown in Figure 10.

7.4 Topology with cross-traffic, WFQ scheduling and streaming traffic

We demonstrate here the ability of our framework to support the scheduling algorithms previously described as well as streaming traffic. We use the topology presented in Figure 11. The cross-traffic here is generated by flows *F3* and *F7*. The egress part of the switches uses Weighted Fair



Flow	Source	Dest	Flow type	Priority
F1	PC2	PC3	UDP rate 5Mbps	1
F2	PC0	PC5	TCP	1
F3	PC5	PC0	TCP	1
F4	PC1	PC2	TCP	2
F5	PC2	PC3	TCP	2
F6	PC3	PC4	TCP	2
F7	PC4	PC1	TCP	0
F8	PC0	PC5	TCP	0

Figure 11: Daisy chain topology with 4 switches and WFQ scheduling

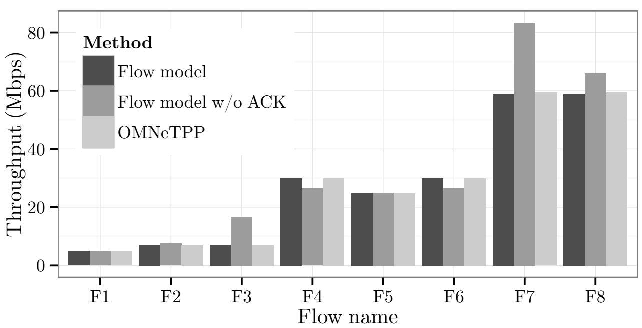


Figure 12: Throughput of the flows for the daisy-chain topology

Queuing, with 3 priorities, from 0 to 2, with respective weights of 5, 1 and 2. We define a maximum queue size of 50 packets.

Results for this topology are presented in Figure 12, where we compare the results of the simple TCP model with the results of the improved TCP model. When the TCP ACKs are not taken into account large errors appear in the results. Our improved TCP model is indeed relevant compared to the OMNeT++ results.

7.5 Random tree topology

In order to evaluate our framework on other topologies, we used randomly generated trees following Algorithm 2. We used a tree topology as it guarantees a unique path between two nodes of the topology, meaning that the path of flow is guaranteed to be the same in the model and in the simulation. The leaves of the tree correspond to computers, while the internal vertices correspond to switches. The algorithm generates only TCP flows, via the function TCP_FLOW(<SOURCE>, <DESTINATION>). Switches are considered to have an infinite buffer.

We generated four random tree topologies using Algorithm 2 with parameters $maxDepth = 4$, $minLeaves = 4$, $maxLeaves = 8$, $minFlows = 1$, $maxFlows = 1$ and evaluated the log error for the flow throughputs. The topologies correspond to the topologies presented in Figure 13.

Algorithm 2 Random tree generation algorithm

Require: $maxDepth \geq 0$, $minLeaves \geq 0$, $maxLeaves \geq 0$, $minFlows \geq 0$, $maxFlows \geq 0$

function GENERATETOPOLOGY
    $root \leftarrow$ CREATENODE
    $leaves \leftarrow$ GENERATELEAVES($root$, $maxDepth$)
    for all $leaf$ in $leaves$ **do**
        for RANDOMINT($minFlows$, $maxFlows$) **do**
            TCP_FLOW($leaf$, RANDOM($leaves$))
        end for
    end for
end function

function GENERATELEAVES($root$, $depth$)
    if $depth = 0$ **then**
        return $root$
    end if
    $leaves \leftarrow []$
    for RANDOMINT($minLeaves$, $maxLeaves$) **do**
        $node \leftarrow$ CREATENODE
        CREATELINK($node$, $root$, $100Mbps$)
        $d \leftarrow$ RANDOMINT(0, $depth - 1$)
        $leaves \leftarrow$ [$leaves$, GENERATELEAVES($node$, d)]
    end for
    return $leaves$
end function

We first study the evaluation of the TCP model without acknowledgments as presented in Figure 14. We see that the log-error reaches a maximum value of 1.37, which corresponds to an error of $exp(1.37) - 1 = 294\%$. The four topologies are then evaluated with the improved TCP model including acknowledgments, and results are presented in Figure 15. As expected, the accuracy of the model is improved, with a maximum log error of 0.1 which corresponds to an error of $exp(0.1) - 1 = 10\%$.

8. CONCLUSION AND FUTURE WORK

We presented in this paper our flow-level network framework for the performance evaluation of Ethernet topologies. This framework is based on two building blocks: servers for representing Ethernet interfaces and queues, and flows for representing Ethernet communications between nodes of the topology. Our model for servers support FIFO queues as well as scheduling functions such as priority based scheduling and fair-bandwidth sharing schedulers. Our model for Ethernet flows supports long-lived TCP connections as well as UDP multimedia streams.

The results of our framework were compared to the results of the discrete event simulator OMNeT++. Different topologies where used in order to evaluate the accuracy of our model. Our framework delivers results in accordance to the results of simulations, even on large networks. Compared to previous work on the subject, we showed the importance of modeling the TCP acknowledgments in topologies with cross-traffic.

As presented in Section 4.1, we limit TCP flows to long-lived connections, which is not necessarily a realistic view of nowadays Internet traffic. We would like to introduce short lived TCP connections, with the use of the model described in [11] in our framework. Along with short TCP connections, we would like to include realistic traffic patterns, based on our previous work on realistic traffic simulation [15]. Finally other TCP congestion avoidance algorithms such as TCP CUBIC [17] or TCP Compound [29] would be beneficial to have models in accordance to the TCP stacks used in current operating systems.

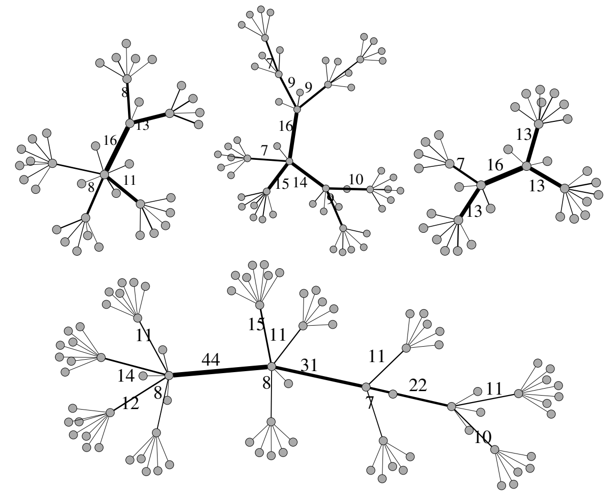


Figure 13: Example of four generated topologies with Algorithm 2 using the following parameters: $maxDepth = 4$, $minLeaves = 4$ and $maxLeaves = 8$. The width and the label of the edges represent the number of flows on the edge (for edges with more than 5 flows).

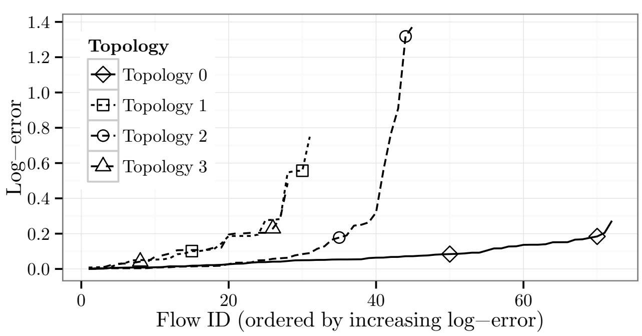


Figure 14: Log error of the flow throughput on four random tree topologies generated by Algorithm 2, with the basic TCP model ignoring TCP acknowledgments described in Section 4.1

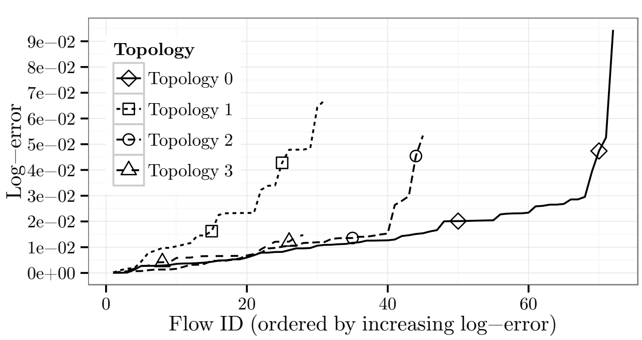


Figure 15: Log error of the flow throughput on four random tree topologies generated by Algorithm 2, with the improved TCP model including TCP acknowledgments described in Section 4.2

9. REFERENCES

[1] INET Framework for OMNeT++/OMNEST. `http://inet.omnetpp.org/`, Accessed 03.06.2013.

[2] ns-3 Discrete-Event Network Simulator. `http://www.nsnam.org/`, Accessed 03.06.2013.

[3] OMNeT++ 4.3 Network Simulation Framework. `http://www.omnetpp.org`, Accessed 03.06.2013.

[4] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), Sept. 2009.

[5] E. Altman, K. Avrachenkov, and C. Barakat. TCP Network Calculus: The case of large delay-bandwidth product. In *Proceedings of INFOCOM 2002*, pages 417–426. IEEE, 2002.

[6] U. Ayesta, K. Avratchenkov, E. Altman, C. Barakat, and P. Dube. Multilevel Approach for Modeling TCP/IP. In *Proceedings of ITC-18*, Sept. 2003.

[7] H. Balakrishnan, V. Padmanabhan, G. Fairhurst, and M. Sooriyabandara. TCP Performance Implications of Network Path Asymmetry. RFC 3449 (Best Current Practice), Dec. 2002.

[8] M. Barbera, A. Lombardo, G. Schembra, and A. Trecarichi. Fluid flow analysis of TCP flows in a DiffServ environment. *European Transactions on Telecommunications*, 17:505–524, 2006.

[9] J. C. R. Bennett and H. Zhang. WF^{2}Q: Worst-case Fair Weighted Fair Queueing. In *Proceedings of INFOCOM 1996*, volume 1, pages 120–128, 1996.

[10] T. Bu and D. Towsley. Fixed Point Approximations for TCP behavior in an AQM Network. In *ACM SIGMETRICS Performance Evaluation Review*, volume 29, pages 216–225. ACM, 2001.

[11] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *Proceedings of INFOCOM 2000*, volume 3, pages 1742–1751. IEEE, 2000.

[12] C.-S. Chang. *Performance Guarantees in Communication Networks*. Springer-Verlag New York Incorporated, 2000.

[13] V. Firoiu, I. Yeom, and X. Zhang. A Framework for Practical Performance Evaluation and Traffic Engineering in IP Networks. In *IEEE International Conference on Telecommunications*, 2001.

[14] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, 1993.

[15] F. Geyer, S. Schneele, and G. Carle. RENETO, a Realistic Network Traffic Generator for OMNeT++/INET. In *Proceedings of the Sixth International Conference on Simulation Tools and Techniques - SIMUTOOLS 2013*. ICST, Mar. 2013.

[16] R. Gibbens, S. Sargood, C. Van Eijl, F. Kelly, H. Azmoodeh, R. Macfadyen, and N. Macfadyen. Fixed-Point Models for the End-to-End Performance Analysis of IP Networks. In *13th ITC specialist*

seminar: IP traffic measurement, modeling and management, 2000.

[17] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[18] H. Hassan, O. Brun, J. M. Garcia, and D. Gauchard. Integration of Streaming and Elastic Traffic: A Fixed Point Approach. In *Proceedings of the 1st international conference on Simulation Tools and Techniques, SIMUTOOLS 2008*, pages 25:1–25:10. ICST, Mar. 2008.

[19] M. Heusse, S. A. Merritt, T. X. Brown, and A. Duda. Two-way TCP Connections: Old Problem, New Insight. *ACM SIGCOMM Computer Communication Review*, 41(2):5–15, Apr. 2011.

[20] V. Jacobson. Modified TCP congestion avoidance algorithm. end2end-interest mailing list, Apr. 1990.

[21] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Qystems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[22] Y. Liu, F. L. Presti, V. Misra, D. F. Towsley, and Y. Gu. Fluid Models and Solutions for Large-Scale IP Networks. *Sigmetrics Performance Evaluation Review*, 31:91–101, 2003.

[23] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, June 1997.

[24] V. Misra, W.-B. Gong, and D. Towsley. Stochastic Differential Equation Modeling and Analysis of TCP Windowsize Behavior. In *Proceedings of IFIP WG 7.3 Performance*, November 1999.

[25] V. Misra, W.-B. Gong, and D. Towsley. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. In *ACM SIGCOMM Computer Communication Review*, volume 30, pages 151–160. ACM, 2000.

[26] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. *IEEE/ACM Trans. Netw.*, 8(2):133–145, Apr. 2000.

[27] A. Parekh and R. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. *IEEE/ACM Trans. Netw.*, 1(3):344–357, June 1993.

[28] M. Shreedhar and G. Varghese. Efficient Fair Queuing Using Deficit Round-Robin. *IEEE/ACM Trans. Netw.*, 4(3):375–385, June 1996.

[29] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. Technical Report MSR-TR-2005-86, Microsoft Research, July 2005.

[30] P. Velho and A. Legrand. Accuracy Study and Improvement of Network Simulation in the SimGrid Framework. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques, SIMUTOOLS 2009*, page 13. ICST, Mar. 2009.

[31] P. Velho, L. Schnorr, H. Casanova, and A. Legrand. Flow-level network models: have we reached the limits? Technical Report 7821, INRIA, Nov. 2011.

[32] L. Zhang, S. Shenker, and D. D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. In *ACM SIGCOMM Computer Communication Review*, volume 21, pages 133–147. ACM, 1991.