Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2010, Article ID 379652, 10 pages
doi:10.1155/2010/379652

Research Article

Pricing and Unresponsive Flows Purging for Global Rate Enhancement

G. Abbas,[1] A. K. Nagar,[1] H. Tawfik,[1] and J. Y. Goulermas[2]

[1] *Intelligent and Distributed Systems Laboratory, Liverpool Hope University, Liverpool L16 9JD, UK*
[2] *Centre for Intelligent Monitoring Systems, University of Liverpool, Liverpool L69 3BX, UK*

Correspondence should be addressed to A. K. Nagar, nagara@hope.ac.uk

Received 17 November 2009; Revised 13 March 2010; Accepted 27 March 2010

Academic Editor: Petri Mähönen

Pricing-based Active Queue Management (AQM), such as Random Exponential Marking (REM), outperforms other probabilistic counterpart techniques, like Random Early Detection (RED), in terms of both high utilization and negligible loss and delay. However, the pricing-based protocols do not take account of unresponsive flows that can significantly alter the subsequent rate allocation. This letter presents Purge (Pricing and Un-Responsive flows purging for Global rate Enhancement) that extends the REM framework to regulate unresponsive flows. We show that Purge is effective at providing fairness and requires small memory and low-complexity operations.

1. Introduction

Recent theoretical advances in Network Utility Maximization (NUM) [1, 2] have facilitated development of AQM protocols, such as REM [3], wherein congestion signals, or so-called prices, are computed and communicated by network to sources for a closed-loop dynamic rate allocation. The fundamental design philosophy of NUM provides advantage over traditional window-based heuristic flow control, in that maximal bandwidth utility is achieved as sources adaptively adjust their transmission rates [1, 3, 4]. Even so, the pricing-based NUM approaches, due to their convex optimization framework, are typically limited to applications with elastic bandwidth utilities, as exemplified by TCP-friendly web (HTTP) and file-transfer (FTP) services. Thus, QoS tradeoffs are inevitable where inelastic applications with versatile bandwidth utilities are coexistent [5], such as in the Internet. More recently, a few attempts have been made to extend the NUM framework to applications with inelastic utilities [5], such as voice (VoIP) and video (IPTV) services. Nevertheless, an important argument is that all NUM approaches essentially operate on the premise that each source *does* respond to the price signals by accordingly adjusting its rate. In actuality, this is far from being valid.

Lately, the intense proliferation of multimedia and real-time audio/video streaming applications has practically paralleled the growth of Internet. These applications are typically bandwidth-hungry, generate large streams, prefer a steady data rate, or transmit at a fixed rate, called Constant-Bit-Rate (CBR)—as in on-demand interactive applications, such as video conferencing and gaming. Although loss-tolerant to some extent, multimedia applications are usually sensitive to delay and jitter. These characteristics make UDP the favorite transport-protocol when designing such applications, so a negligible degradation in quality occurs rather than substantial delays if lost packets are retransmitted. However, UDP does not implement closed-loop flow control and, as such, the traffic transported by it is not TCP-friendly or, even worse, is unresponsive to the price signals.

The coexistence of *multiclass* traffic in the highly heterogeneous Internet leads to a fundamental tension between responsive and unresponsive flows and can cause three maladies, namely unfairness, congestion-collapse, and security. Unfairness originates during periods of congestion when the well-behaved TCP-friendly responsive flows back-off, unlike unresponsive flows which are unable to do so. Consequently, unresponsive flows benefit from their greedy nature by aggressively consuming increasingly larger portions of bandwidth unfairly [6]. In an extreme case this phenomenon leads to the malady of congestion-collapse [7], wherein a network remains in a persistent congestion as bandwidth is continually consumed by packets that are

repeatedly dropped by routers, leaving the system with no worthwhile communication.

In addition to the unresponsive UDP flows, there is another class of misbehaving traffic, namely, unresponsive TCP sessions that cause the third malady of security. Greedy users can exploit the vulnerabilities in TCP to receive superior service, such as by modifying the source code, in an open-source Linux context, to deactivate flow control mechanisms in the TCP/IP stack [8]. This also allows malicious users to initiate Distributed Denial of Service (DDoS) (brute-force flood based) attacks that can have a serious impact on network security.

Although the control of unresponsive flows is generally ignored in the designs of queue management, it is envisaged that this will become an inevitable and integral part of all AQM schemes due to the recent growing traffic trends. For instance, a report [9] by Arbor Networks reveals that DDoS attacks consistently account for 1%–3% of all interdomain traffic. Yet, as evident from more recent attacks on Twitter and Facebook [10], these activities are likely to spread further posing a serious security concern. Conversely, UDP flows constitute 12%–20% of overall Internet traffic [11, 12]. This ratio is also likely to increase as the unresponsive transmission phenomena of UDP may encourage application designers to employ it in an effort to receive superior performance unfairly. For instance, BitTorrent, a common P2P application, has announced switching to UDP [13]. The criticality then is that P2P constitutes more than 50% of internet traffic and BitTorrent is the most widely used P2P protocol worldwide [14].

This article presents *Purge* that complements REM to incorporate therein the control of high_bandwidth unresponsive flows, in an effort to encourage application designers to use TCP-friendly protocols, so as to minimize the impact of the maladies due to misbehaving flows.

The rest of this letter is organized as follows. Section 2 presents existing work. Section 3 presents the Purge algorithm. Section 4 presents experimental results, followed by the conclusion in Section 5.

2. Related Work

Internet flow control comprises two components: an end-to-end algorithm, such as TCP, for sources and a link algorithm (AQM scheme), for routers. The former defines precisely how the source rates are adjusted, while the latter defines how the congestion measure is updated. In traditional TCP/AQM models, TCP follows some Additive Increase, Multiplicative Decrease (AIMD) mechanism to adjust its transmission window size, based on the congestion notification from AQM. An alternative to this approach is NUM that relates the economic concept of utility to the TCP/AQM operation. In NUM frameworks, AQM determines link price, based on which the source algorithm buys bandwidth by maximizing a TCP utility function to adaptively adjust its transmission rate. We restrict ourselves to AQM here, as routers are the central place to effectively regulate unresponsive flows (for overview of TCP/AQM models, see [1, 15, 16]). The most well-known AQM schemes are RED [17], employed in the traditional AIMD/AQM models, and REM [3], which is more suitable in NUM frameworks. Unlike RED, that measures congestion with average queue length, REM decouples its congestion measure (price) from the performance measure (loss and delay), to stabilize the latter irrespective of the number of flows and, as such, can significantly outperform RED.

Several important and recent improvements to the basic RED and REM are worth mentioning here. For instance, time-delay control theory has been applied to TCP/RED dynamic models in [16, 18], in order to establish explicit stability conditions for RED to stabilize the average queue length and thereby the entire TCP/RED system. The authors have rigorously demonstrated that, by carefully choosing key RED parameters, superior performance can be achieved in terms of arbitrary delay, capacity, and load. Another recent work [15] establishes a theoretical price-based flow control scheme, where the link algorithm extends REM to generate a virtual price. These techniques [15, 16, 18] have been shown to outperform numerous other well-known AQM schemes including REM.

However, neither the basic RED and REM nor their improved variants take account of unresponsive flows, thus the desired performance is subject to the responsiveness of all flows. In the following, we discuss improvements made to the basic AQM schemes to particularly regulate unresponsive flows.

The existing solutions for controlling unresponsive flows can be classified into two major categories. The first category algorithms, such as Fair-RED and Balanced-RED, essentially operate on full *per-flow* state information by identifying flows and thereupon treating them independently, typically by means of separate queues. Thus, unfairness is effectively alleviated, but their complexities are proportional to the number of flows and as such, these algorithms are contrary to the Internet scalability argument. The other category, not requiring full per-flow information, can be further divided into two subcategories. Algorithms in the first subcategory estimate the number of active flows to bring about fairness, unlike the other subcategory that does not need such information. The accuracy of the latter can be uncertain [19], thence we focus on the former, which can offer a balance between complexity and accuracy, and review prominent existing solutions from this subcategory.

To that end, a notable technique is Stabilized-RED (SRED) [20] that aims at stabilizing an FCFS buffer by preemptively discarding packets with a probability that depends on both buffer occupancy and the estimates of the number of active flows ($\mathbb{N}act$). It detects misbehaving flows by maintaining a so-called Zombie-list that serves as a proxy for information about recently seen flows. The drawback, however, is the inaccuracy in $\mathbb{N}act$ estimation which is based on the assumption that all flows have the same traffic intensity. We will elaborate more on this problem in later sections.

CARE [21] is another established technique based on a Capture-Recapture estimation model to estimate $\mathbb{N}act$ and the arrival rate of flows. On the other hand, in order to increase its accuracy, CARE has to make large amounts

of captures, which is the major limiting factor from the perspective of scalability.

Recently, HaDQ (Hashing & caching-based Dynamic Quarantine) [8] has been proposed as an extension to SRED. This work distinguishes misbehaving TCP from UDP, whereas we do not make any such distinctions and work on the basis of high-bandwidth flows.

A more recent approach is BREATH [6] that outperforms several predecessors. Yet again, the underlying mechanism is based on a heavy-hitter set technique of [22] that requires two passes over the dataset. This approach is also relatively inefficient for high-speed networks. We will come back to this point in Section 3.2.

Another eminent technique is BLACK [7] that uses a sampling technique to approximate the buffer occupancy fraction of only high-bandwidth flows to reduce the number of per-flow state information. It has been proven numerically [7, 19] that BLACK outperforms most of the above and numerous techniques of other categories. However, the problem associated with BLACK is its simplified technique for estimating $\mathbb{N}act$, which leads to inaccuracies, as with SRED.

An important remark is that these and all of the router-based unresponsive flow control approaches of other categories are based upon variants of RED and work in the traditional window-based AIMD/RED frameworks (see for overview [6, 8, 19]). As such, these solutions are not amenable to the pricing-based NUM frameworks, where a link algorithm also needs to imbed price marks in packet headers, which are then used by source algorithms to optimize their rates. To the best of the authors' knowledge, there is no such AQM scheme that currently deals with unresponsive flows in NUM frameworks. Purge complements REM to effectively regulate high-bandwidth unresponsive flows. The purpose to build upon a pricing scheme is that the NUM frameworks have enhanced QoS provisioning compared to the window based AIMD/RED [1, 3, 4], and by means of Purge, we intend to retain the superior performance of REM even in the presence of unresponsive flows.

3. Purge

The fundamental idea of Purge is that, in a price-based rate control, if packets have to be dropped due to a buffer overflow then packets from high-bandwidth misbehaving flows must be considered the primary candidates of dropping and thus be constrained. To that end, Purge utilizes the proficient idea of BLACK, addresses its shortcomings, and incorporates it to work in conjugation with REM.

3.1. Queue Management. The basic AQM scheme in Purge is REM that periodically updates its link price to determine the marking probability and thereby match source rates to network capacity, while stabilizing queue around a small target. Precisely, the price $p_\ell(t)$ for link ℓ in period t is updated according to

$$p_\ell(t) = \left[p_\ell(t-1) + \gamma\left(\alpha_\ell\left(b_\ell(t) - b_\ell^*\right) + X_\ell(t) - c_\ell(t)\right)\right]^+, \tag{1}$$

where $\alpha_\ell > 0$ and $\gamma > 0$ are small constants, $b_\ell(t)$ and $X_\ell(t)$ are the aggregate buffer occupancy and the aggregate input rate, respectively, at link ℓ in period t, $c_\ell(t)$ is the available bandwidth to link ℓ in period t, $b_\ell^* \geq 0$ is a predefined target queue-length and $[z]^+ = \max\{z, 0\}$. The constant α_ℓ is the weight of buffer that trades off utilization and queuing delay during transient. The constant γ is the step-size that controls responsiveness of REM to changes in network conditions. To convey the price to source, REM marks packets with the probability $1 - \varphi^{-p_\ell(t)}$, where $\varphi > 1$ is a constant.

The price is increased if the weighted sum of queue-mismatch $b_\ell(t) - b_\ell^*$ and rate-mismatch $X_\ell(t) - c_\ell(t)$ is positive, and is decreased otherwise. The weighted sum is positive when either the source rates exceed the link capacity or there is excessive backlog to be cleared, and is negative otherwise. When the source rates are too small, the negative weighted sum pushes down price and thus marking probability, thereby allowing sources to increase their transmission rates, until eventually the mismatches are driven to zero. In equilibrium, when source rates equal capacity $X_\ell = c_\ell$ and backlog equals target $b_\ell = b_\ell^*$, the price stabilizes as the weighted sum is zero, yielding high utilization and negligible loss and delay. In overloaded situations, the mismatches in rate and queue enlarge, pushing up price and marking probability, thereby causing the sources to reduce their rates, in order to bring the system back to equilibrium.

However, we argue that the equilibrium is not achievable in the presence of unresponsive flows, which do not cut down their transmission rates and, therefore, lead to the aforementioned maladies. Purge allows REM to retain its inherent capabilities even in the presence of unresponsive flows.

The unresponsive flow control mechanism in Purge is based on the BLACK's [7] concepts of Buffer Occupancy Fraction (BOF), used as an indicator of a flow's share of the bandwidth, and a High-Bandwidth Flows (HBF) cache-memory that keeps track of misbehaving flows. The idea is that bandwidth given to active flows is roughly proportional to their share of buffer space, thus fair bandwidth allocation can be achieved at a high degree if the buffer is allocated evenly among all active flows under an FCFS queue.

Precisely, for each packet arrival at link ℓ, Purge randomly samples a packet from the buffer whenever the buffer occupancy b_ℓ exceeds its target b_ℓ^*. With this event, the FlowID_i of the sampled packet from flow i is recorded in the HBF cache-memory, and a Hit is declared for flow i at link ℓ. Next time, if a sampled packet is from flow i again, its $\text{Hit}_{\ell,i}$ is incremented by one. Using the memory management approach of [7], only high-bandwidth flows are more likely to stay in the cache-memory. When the weighted sum of queue mismatch and rate mismatch is positive, each FlowID_i is checked against its number of hits. A *Hit-Fraction* $\Omega_{\ell,i}$ is approximated to be a flow i's *average buffer occupancy fraction* at link ℓ in period t as

$$\Omega_{\ell,i}(t) = \frac{\text{Hit}_{\ell,i}(t) + \Omega_{\ell,i}(t-1)\Psi_\ell(t-1)}{\Psi_\ell(t-1)\Psi_\ell(t)}, \tag{2}$$

where $\Psi_\ell(t)$ is the number of samplings at link ℓ in period t. A flow with larger Hit-Fraction $\Omega_{\ell,i}(t)$ than a *fair*-BOF is

potentially a candidate of being a high-bandwidth flow. The *fair*-BOF is determined by $1/\mathbb{N}act(t)$, where $\mathbb{N}act(t)$ is the number of active flows in period t. An estimation procedure for $\mathbb{N}act(t)$ will be discussed in the next subsection. The detected high-bandwidth flow is subject to be dropped with a probability that depends on how many times extra buffer space it consumes than the fair share $1/\mathbb{N}act(t)$. This can be measured using the flow's Hit-Fraction and the fair BOF as $\Omega_{\ell,i}(t)\mathbb{N}act_\ell(t) - 1$ [7]. While this measure could effectively yield a flow-specific dropping probability for high-bandwidth flows, nevertheless, keeping this as a fixed dropping function will enforce packet-drops even in mild queue and rate mismatch conditions. Thus, the dropping function to be used must also be adjusted according to the network saturation levels to allow large flows to take some portion of the bandwidth during less overloaded situations. Accordingly, as the buffer occupancy increases, the per-flow dropping probability should also increase gradually in a linear proportion. For this purpose, we utilize the REM's price update procedure to derive the dropping function as $(\Omega_{\ell,i}(t)\mathbb{N}act_\ell(t) - 1) \cdot \gamma(\alpha_\ell(b_\ell(t) - b_\ell^*) + X_\ell(t) - c_\ell(t))$. A unified formulation of the Purge's per-flow dropping function $d_{\ell,i}(t)$ for flow i at link ℓ in period t can precisely be given as

$$d_{\ell,i}(t) = \left[\left(\frac{\mathrm{Hit}_{\ell,i}(t)\Omega_{\ell,i}(t-1)\Psi_\ell(t-1)}{\Psi_\ell(t-1)\Psi_\ell(t)}\mathbb{N}act_\ell(t) - 1\right) \cdot\gamma\left(\alpha_\ell(b_\ell(t) - b_\ell^*) + X_\ell(t) - c_\ell(t)\right)\right]^+, \tag{3}$$

where $[z]^+ = \max\{z, 0\}$. Under mild queue mismatch situations in Purge, REM operates as normal by updating prices and marking packets in order to match source rates to available capacity, while clearing buffers and stabilizing queues around b_ℓ^*. At the same time, Purge manages HBF cache memory to keep track of potentially high-bandwidth flows, as shown in Figure 1. Note that Purge requires b_ℓ^* to be nonzero, whereas the actual REM proposal [3] allows it to be zero as well. Typically, a b_ℓ^* value between 15%–20% of the total buffer-size yields a better performance for Purge, as shall be seen in the next section. As the weighted sum of queue mismatch and rate mismatch becomes increasingly positive, $d_{\ell,i}(t)$ is also increased proportionally. During more overload conditions, indicated by increasing queue mismatch and rate mismatch, in order to keep the queue size low and thus the queuing delay and to send a stronger price signal to HBF sources, the HBF packets are marked (rather than dropping) as a last resort, with their per-flow probability $1 - \varphi^{-d_{\ell,i}(t)}$, instead of the REM's generic probability $1 - \varphi^{-p_\ell(t)}$. Dropping does not occur under only queue mismatch conditions—that is, if the buffer is persistently occupied due to excessive backlog. However, despite the stronger price signal, if rate mismatch persists, this strongly indicates existence of misbehaving unresponsive flows, in which case the HBF packets are dropped.

3.2. $\mathbb{N}act$ Estimation. On one hand, the advantage of BLACK consists in the use of Hit Fraction $\Omega_{\ell,i}$ that approximates the buffer occupancy fraction of only high-bandwidth flows, instead of maintaining per-flow states for every active flow. This makes BLACK highly scalable. On the other hand, since no per-flow state is maintained, the number of active flows ($\mathbb{N}act$) needs to be estimated to determine the fair-BOF. Hence, the effective detection and throttling of misbehaving flows become largely dependent on the *accuracy* of the $\mathbb{N}act$ estimates. The limitation of BLACK, however, is its simplified estimation of $\mathbb{N}act$. For each packet arrival at link ℓ, BLACK compares a sampled packet with the arriving packet. If both packets belong to the same flow, a *match event* is declared. Let τ flows, numbered, $1, 2, \ldots, \tau$, arrive at the router, BLACK assumes the probability that an arriving packet belongs to flow i is $1/\tau$, for all $1 \le i \le \tau$. Thereupon, $\mathbb{N}act_\ell$ is simply determined based on the total match events over the total packet arrivals during a sampling period—that is $\Psi_\ell(t)/(\textit{number of match events})$. Hence, a very strong assumption imposed is that all τ arrived flows have the same traffic intensity (ratio of the arrival rate to the service rate during a specific time period) $1/\tau$. This leads to inaccuracies in more realistic scenarios; for instance, in case of Internet, traffic intensities are vastly different [19].

To address this shortcoming, we pose the $\mathbb{N}act$ estimation problem in terms of finding the cardinality estimates of a *multiset.* A multiset is defined as a set where each element can appear several times. The size N of the multiset is the total number of elements, including repetitions, while its cardinality n is the number of distinct elements in the multiset. This approach constitutes a framework of a multiset of N packets from n connections or flows. Such a multiset is naturally constructed in a router's queue that serves realistic Internet traffic as large amounts of packets arrive in different patterns with varying intensities. The problem then is to determine the cardinality of the multiset of N packets at a given period t, in order to obtain the number n that would indicate the distinct flows, which will be $\mathbb{N}act(t)$ or total active flows in period t.

Though the idea of multiset here is very straightforward, nonetheless, to determine its cardinality, we need a very robust technique. In the following, we discuss the significance of the desired robustness and present a solution.

The problem of cardinality estimation arises frequently in databases due to natural operations on large datasets [23]. However, the considered problem is different in a sense that the multiset of packets changes extremely dynamically. The easiest and most accurate way to determine the cardinality could be to equip the router with an algorithm to count, for instance, by sequential selection and comparison operations, the distinct packets n out of the multiset of N packets at a given period t. However, this could require up to N passes over the multiset and a memory of up to n words, which is highly unsalable solution as the complexity would increase with the number of flows and there are generally hundreds of thousands of flows in a router. Thus, the execution time and low memory are essentially the most crucial criteria for choosing the cardinality estimation technique to operate on a router.

Hashing, on the other hand, is an effective technique with a potential to simplify the considered estimation problem.

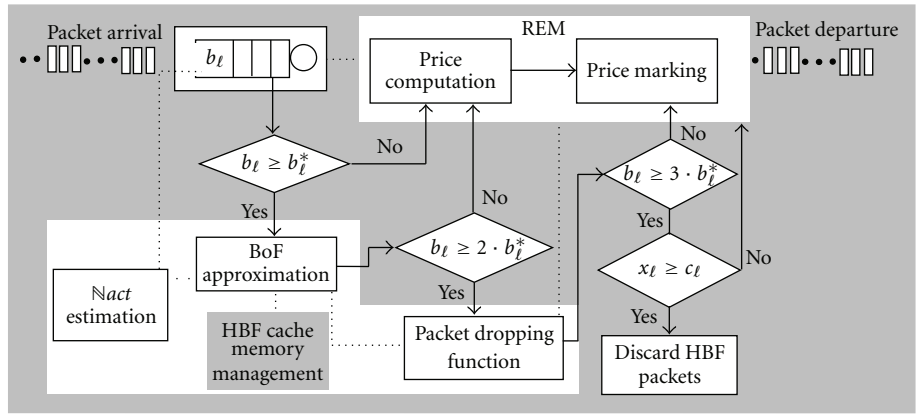

Figure 1: Block diagram of Purge.

A Hash function is mostly used to index large unordered data into small datum to make the record lookups efficient for data comparisons, for instance, in detecting duplicated items or finding identical stretches of DNA sequences. The values returned by a hash function are called hash values (see [23] for an overview). There are various types of hashing techniques, such as the heavy-hitter data structure [22] that requires two passes over the multiset to determine the cardinality n. Nevertheless, it is expected that any algorithm involving more than one passes over the huge multiset of N packets to obtain n distinct flows will lead to an outdated result, since packets pass through the router with enormous speeds. Thus, the desired solution is to treat the multiset in one pass using a simple loop and with a small auxiliary memory. We adopt a recently proposed order-statistics-based MINCOUNT technique [24] that is placed in the class of best known algorithms so far due to its excellent trade-off between memory, execution time, and accuracy, which makes it the most suitable $\mathbb{N}act$ estimation approach for Purge. The technique works here as follows.

Let $\Xi_\ell(t) = \{\rho_1, \rho_2, \ldots, \rho_N\}$ be a *multiset* of N packets from n, $n \leq N$, distinct flows in Ξ at link ℓ in period t. A modular arithmetic-based hash function $\hbar$ maps the FlowID (This could simply be the flow ID field of an IPv6 packet or a combination of pairs "source IP, destination IP", "source port, destination port".) of each packet to a real value that is uniformly distributed in the unit interval $[0, 1]$. Irrespective of the nature of traffic, $\{\hbar(\rho_1), \hbar(\rho_2), \ldots, \hbar(\rho_N)\}$ yields a set of hashed packets built from n real values taken independently uniformly at random in $[0, 1]$, and then replicated and permuted in an arbitrary way. Such a set is called an *ideal-multiset* [24], the key idea wherein consists in that kth minimum of the values of the ideal-multiset neither depends on the replication structure of the data nor on their order of appearance. Thus estimating the cardinality of the ideal-multiset yields the number n of distinct values as the required $\mathbb{N}act$ estimate in Purge. Note that minimum of a sequence of numbers is found with a single pass over the elements. The algorithm averages over several similar experiments to improve its precision, based on the fact that the arithmetic mean of m i.i.d. random variables with expectation μ and standard deviation σ has the same μ but a σ scaled down by $1/\sqrt{m}$. However, performing m experiments involves using m different hashing functions which is unreasonable due to complexity.

To that end, the principle is to construct an observable, based on the kth minimum, and to combine it with a *stochastic averaging* process that simulates the effect of m experiments. This is done by distributing hashed values in m different buckets by dividing $[0, 1]$ into m intervals of size $1/m$, while using a single hash function, and then averaging an observable over m from the kth minimum of each bucket. A hashed value falls in the ith bucket if $(i-1)/m \leq {}_i \leq i/m$. A precise estimate is then built as

$$\mathbb{N}act_\ell(t) = m\left(\frac{\Gamma(k-1/m)}{\Gamma(k)}\right)^{-m} \cdot \exp\left(\frac{1}{m}\sum_{i=1}^{m} \ln\left(M_{\ell,i}^k(t)\right)\right), \tag{4}$$

where Γ is the Euler's Gamma function and $M_{\ell,i}^k(t)$ is the kth minimum of the ith bucket of the ideal-multiset built from $\Xi_\ell(t)$.

4. Simulation Results

The results of our analysis are derived from OPNET (Modeler ver. 14.5) simulations and are based on topology shown in Figure 2 and parameters listed in Table 1, unless specified otherwise. The efficiency of Purge in regulating unresponsive flows is evaluated by means of several performance metrics including accuracy of $\mathbb{N}act$ estimates, execution time, throughput, fairness, goodput, and scalability. All results presented are based on 25 replicated simulation runs for each scenario, by maintaining fixed values of input parameters and only varying the random-seed values in each run, in order to compute average results using 95% confidence interval. The graphs only plot mean values for better readability; confidence intervals are omitted as they are very tight. In all scenarios, TCP flows cover large proportion of traffic, while the percentage of UDP flows is kept 12% of the overall traffic to reflect the practical ratio currently prevalent on the Internet.

The MINCOUNT parameters m and k have been chosen so as to yield the best practical estimates with minimal

Table 1: Simulation parameters.

Buffer size	1250 KB
Maximum segment size	8 KB
b_ℓ^*	250 KB
α	0.1
γ	0.001
φ	1.001
HBF cache size	50
m	1024
k	3
TCP implementation	Vegas ([1, Equation (15)])
Operation mode	Simultaneous

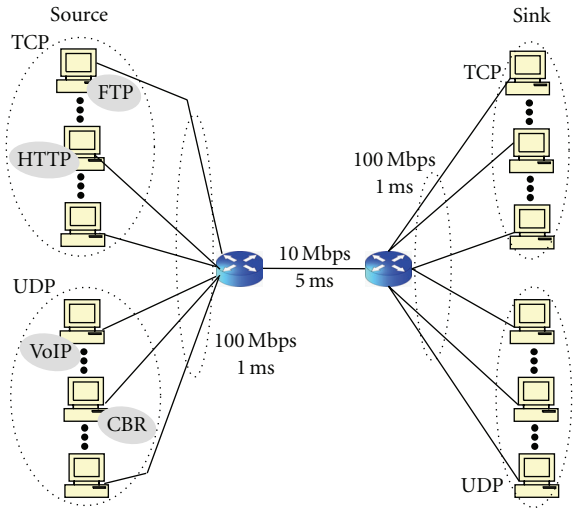

Figure 2: Simulation topology.

memory requirement, as shall be discussed in Section 4.2. The REM parameters α, γ, and φ have been set according to the author's recommendations and the chosen values are more suitable in the considered scenarios of variant intensities. The HFB cache size is recommended to be neither too small nor too large [7]. BLACK generally works well when size is around 40–50, as in [7, 19]. Purge is configured with a cache size of 50. The target queue-length b_ℓ^* has been set to 20% of the buffer-size as discussed in Section 3.1.

4.1. $\mathbb{N}act$ Estimation Accuracy. We first compare the $\mathbb{N}act$ estimation techniques of BLACK and Purge, which is crucial in determining the *fair-BoF* and thus in the overall regulation of misbehaving unresponsive flows. To test the accuracy, traffic is generated by TCP-based FTP and UDP-based VoIP flows and simulations run for 60 seconds, where $\mathbb{N}act$ is estimated every 1 second such that $b_\ell \geq b_\ell^*$ for all t.

For a 600-flow scenario presented in Figure 3, during the first 20 seconds, the arrival rate of all flows is uniformly distributed, transmission rates are all identical with 25 Kbps and all flows transmit simultaneously from 1 to 20 seconds. Under these settings all flows have similar intensity (Ratio of arrival rate to the service rate during a specified time

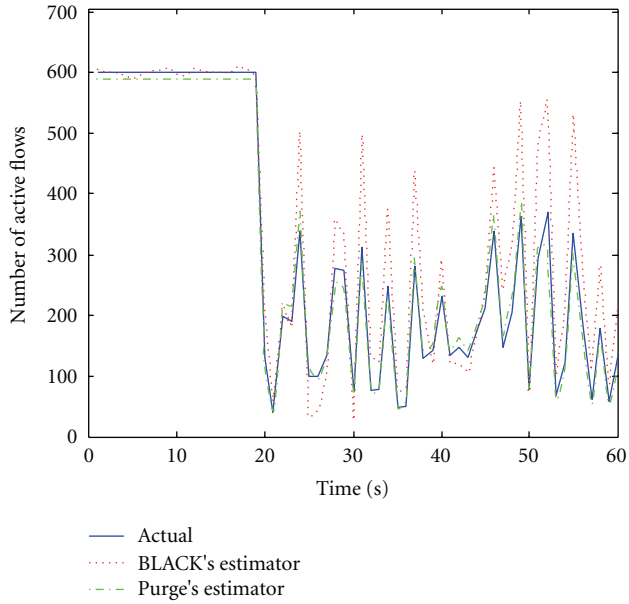

Figure 3: Comparison of $\mathbb{N}act$ estimators.

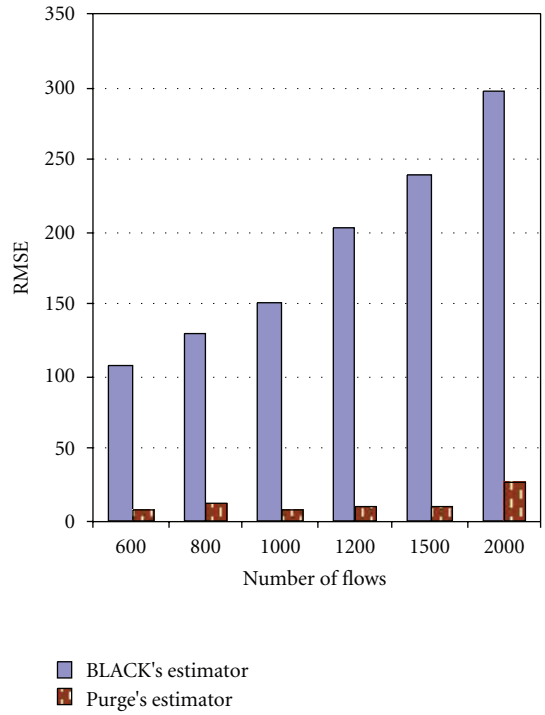

Figure 4: RMSE comparison.

period. For IP flows of unicast traffic between two specific IP addresses, OPNET configures traffic intensity in packets/sec and bits/sec (packet size is computed as a ratio of these two values), type of service used and how long the traffic lasts.). From a multiset perspective, the buffer has $n \approx N$ flows at any period t, in which case both techniques produce estimates reasonably close to the actual number of 600 active flows.

During 21 and 60 seconds, traffic is generated by flows drawn from exponential distribution so their arrival times are dissimilar, such that flows have different ON-OFF periods. Half of the TCP flows are now HTTP and are configured by TCP-Reno ([1, Equation (8)]); whereas for UDP flows the call volumes are assigned randomly in 100–1000 Erlangs. Under these settings, the overall traffic intensity varies significantly with frequent bursty intervals. Note that this is the closest representation of the real Internet traffic patterns [11]. Consequently, the number of active flows n in the buffer at any period t fluctuates between 1 and 600 such that $n \ll N$, in which case, as opposed to Purge, BLACK's estimates deviate substantially from the actual numbers of active flows. The Root Mean Squared Error (RMSE) of accuracy for both techniques is shown in Figure 4, which is negligible in case of Purge and remains almost unaffected by increasing flow scale.

4.2. Execution Time. In high-speed networks, such as OC-192 based Internet backbone connections with 10 Gbps link speed, there are up to 1.25 million packets per second (This traffic load is for illustration purpose. Network links are seldom utilized at 100%.) to be processed [25], considering the realistic packet size distribution of 1 KB [26]. This does not allow much time for a backbone router to perform complex operations on each packet. The execution time of an algorithm operating on a router is crucial in this context.

Purge's $\mathbb{N}act$ estimation benefits from the very simple internal loop of MINCOUNT [24] that gives it an advantage in terms of execution time, memory, and accuracy. For instance, it has been demonstrated in [24] that, when $k = 3$ and $m = 1024$, a memory of only 12 KB is enough for MINCOUNT to process 3 million elements per second, on a 2.5 GHz processor, and to build a cardinality estimate with an accuracy of order 2 percent. This processing speed exceeds the link speed of OC-192. However, the results in [24] are based on estimating the cardinality of a large static file, whereas, in the case of Purge, the multiset ($\Xi_\ell(t)$ of packets in the router at link ℓ in period t) to be estimated depends on traffic load and changes dynamically as packets pass through the router at enormous speeds.

To evaluate the computation time of MINCOUNT-based $\mathbb{N}act$ estimator of Purge and compare it with that of BLACK, we make the following modifications to the topology of Figure 2. The access link capacities are varied in the range [10 Mbps, 200 Mbps] and the bottleneck link capacities are equalized with those of the access links in each scenario. The number of flows is also varied in the range $[100, 2000]$ and the traffic intensities are kept identical (as discussed in Section 4.1). The constantly persistent intensity of the traffic along with the absence of the bottleneck link results in the increased traffic load and large multisets to be processed (Identical traffic intensities result in identical packet sizes, composing identical multiset sizes to be processed by both estimators. This makes comparison of computation time more like for like.). The simulation is run for 30 seconds, for each scenario, the time to process a multiset $\Xi_\ell(t)$ is recorded at $t = 30$, and results are then averaged over 25 runs. The results are presented in Figure 5, showing typical execution times to build a single estimate of the number of active flows, on a 2.9 GHz processor. The execution times of both the estimators remain almost similar for small-sized problems, but scale nonlinearly under BLACK. Thus, BLACK may be suitable for low-speed links (ignoring any inaccuracies of the estimates) but would take too long for high-speed links. The consequence can be outcomes representing outdated number of active flows. On the other hand, Purge's execution time scales linearly and can effectively keep up with the link speed of OC-192 (approximately 0.6 seconds to process the full load of traffic on a 10 Gbps link). Moreover, the memory requirements of Purge's estimator remain constant at 12 KB (with $k = 3$ and $m = 1024$), whereas that of the BLACK's estimator increase linearly with the number of match events for i identical flows, as $\sum_{i=1}^{\tau} 1/\tau$. Thus, BLACK may not be suitable for routers with either small memory or large-scale flows.

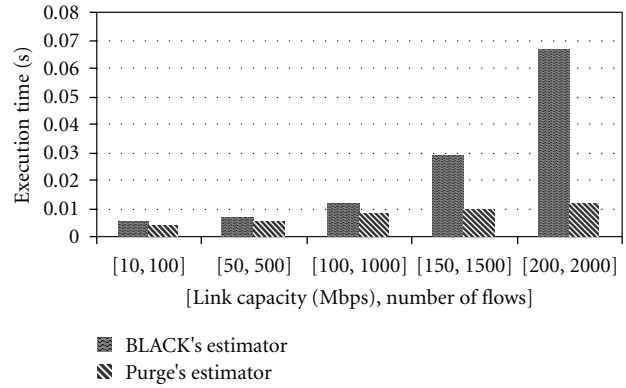

Figure 5: Execution time of the $\mathbb{N}act$ estimators.

4.3. Throughput and Fairness. The inaccuracies of BLACK's $\mathbb{N}act$ estimates diminish its ability to efficiently throttle unresponsive flows in presence of variant traffic intensities. The performance of REM, BLACK, and Purge can be seen in Figure 6, representing a scenario where only one UDP-based VoIP flow with 9 Mbps rate competes with 10 TCP-based FTP flows and simulations run for 100 seconds. The average throughput for this scenario is presented in Figure 7. Clearly, under REM the TCP traffic is almost shutout; it is therefore excluded from further comparisons. Under BLACK, UDP is still privileged as compared to the Purge case.

Fairness is of a significant importance, lack of which leads to the maladies described in Section 1. We evaluate fairness using the Jain's index [27], given as

$$f := \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}, \tag{5}$$

where a value of f, $0 \le f \le 1$, closer to 1 represents fairer rate allocation, and x_i is the throughput achieved by flow i. To compare the impact of scalability and fairness of the allocated throughput, we introduce UDP-based CBR video flows with 3 Mbps rate and TCP-based FTP flows transmitting large files and lasting till the end of simulations. Additionally,

Table 2: Average throughput in multiple unresponsive flows scenario.

	No. of flows	Average TCP throughput (Kbps)	Average UDP throughput (Kbps)	Average unresponsive TCP throughput (Kbps)	Jain's fairness index
BLACK	100	82.944	105.472	112.64	0.9902
Purge		94.208	103.424	104.448	0.9988
BLACK	500	17.408	22.528	25.6	0.9869
Purge		19.456	21.504	22.528	0.9983
BLACK	1000	8.4992	11.6736	12.8	0.9818
Purge		9.3184	10.6496	11.1616	0.9971
BLACK	2000	3.7888	5.4272	6.5536	0.9722
Purge		4.608	5.5296	5.4272	0.9937
BLACK	4000	1.1264	2.048	3.2768	0.8897
Purge		2.1504	2.6624	2.8672	0.992

Table 3: Goodput in multiple unresponsive flows scenario.

	No. of flows	Average TCP goodput (Kbps)	Average UDP goodput (Kbps)	Average unresponsive TCP goodput (Kbps)	System goodput (Mbps)	Jain's fairness index
BLACK	100	81.291	105.472	104.606	8.2903	0.9899
Purge		92.884	103.424	96.145	9.2038	0.9987
BLACK	500	14.89	22.528	17.447	7.7555	0.9761
Purge		17.967	21.504	18.891	8.9937	0.9961
BLACK	1000	6.5066	11.6736	9.899	7.059	0.9452
Purge		8.5407	10.6496	8.773	8.5945	0.994
BLACK	2000	2.7201	5.4272	5.975	6.1379	0.9065
Purge		4.189	5.5296	4.592	8.5195	0.9901
BLACK	4000	0.8641	2.048	2.1915	4.0859	0.8534
Purge		1.9967	2.6624	2.148	8.1294	0.9893

there are 3% short-lived unresponsive malicious TCP flows with 5 Mbps rate. The results are presented in Table 2, which show that the performance of Purge is reasonably scalable to large number of flows. The $\mathbb{N}act$ estimation inaccuracy accumulates and affects throughput allocation and hence the fairness, under BLACK.

4.4. Goodput. Throughput, in Table 2, represents the average number of bits successfully received by the receiver, per second. However, it is also important to evaluate the overall system efficiency in terms of useful bandwidth utilization. Goodput measures the total amount of effective data delivered through the network [1]. The effective data is the useful (nonduplicate) received bits per second. For each flow, goodput can be measured as

$$\begin{aligned} \text{Goodput(TCP)} &:= \frac{\text{rec} - \text{retx}}{T}, \\ \text{Goodput(UDP)} &:= \frac{\text{rec}}{T}, \end{aligned} \tag{6}$$

where rec is the number of bits received, retx is the number of bits retransmitted, and T is the duration of the flow. Using the scenario presented in Section 4.3, the average per flow goodput is the average per flow throughput excluding retransmissions, across the set of flows. For a set of similar flows, the average goodput is the number of useful bits received by all receivers, per second, divided by the number of flows. Consequently, the system goodput is the sum of the goodput of all flows and represents the overall system efficiency in useful bandwidth utilization. Table 3 presents the average goodput results for TCP, UDP, and unresponsive TCP flows. The system goodput along with JFI among the flows, based on their received goodput, is also listed in Table 3. The average goodput for TCP flows is obviously lower than their average throughput (Table 2), due to retransmissions, under both BLACK and Purge. The JFI is also affected slightly for smaller number of flows under both techniques, but deteriorates further under BLACK as the number of flows increases. The system goodput, under BLACK, also deteriorates with the increasing number of flows and reduces to half of that of Purge at 4000 flows.

A common weakness of both BLACK and Purge consists in the use of $1/\mathbb{N}act$ as the standard criteria to determine the fair Buffer Occupancy Fraction. This can lead to occasional

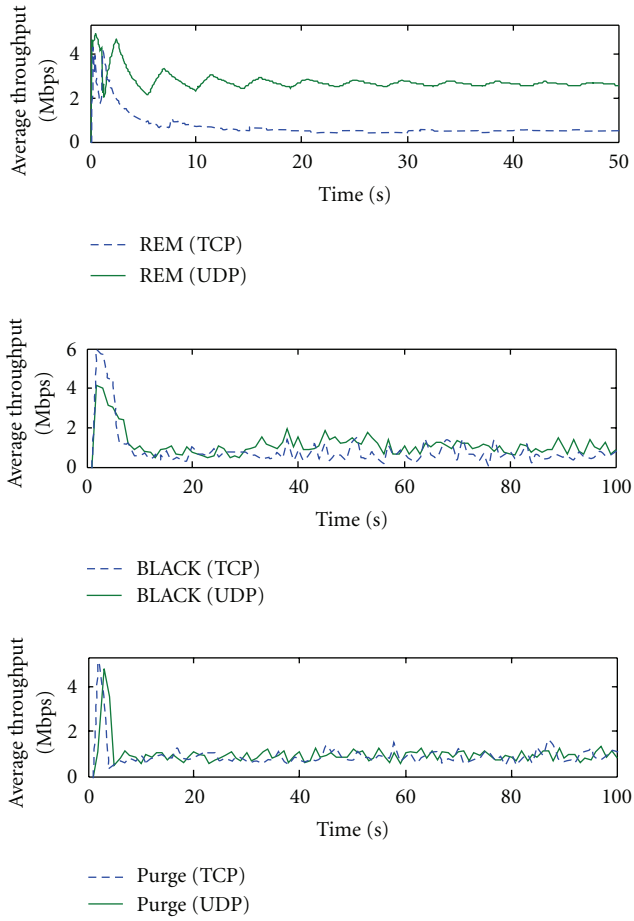

Figure 6: Rate allocation of REM, BLACK, and Purge.

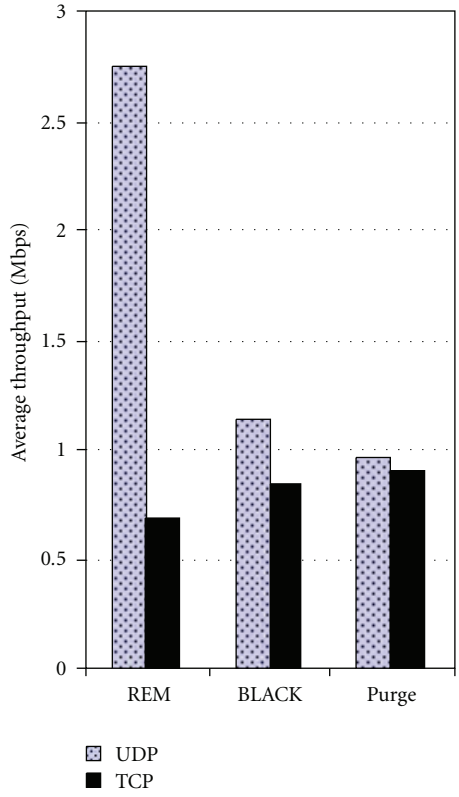

Figure 7: Average throughput in a single unresponsive flow scenario.

unfair dropping for high-bandwidth TCP-friendly responsive sources that are willing to pay a higher price. Our future work will mainly concern this issue.

5. Concluding Remarks

This article presents Purge that employs the Buffer Occupancy Fraction concept of BLACK to provide unresponsive flow control and makes two contributions. Firstly, it addresses the limitation of inaccurate number of active flows estimation in BLACK. We have demonstrated that in realistic Internet scenarios, where traffic intensities vary significantly, the inaccuracies result in suboptimal rate control and unfairness. To that end, we incorporate the MINCOUNT algorithm, the low complexity and memory requirements of which enable Purge-based routers to effectively regulate traffic with variant intensities. Simulation results demonstrate sufficient estimation accuracy of Purge, which scales well to large number of flows. Secondly and more importantly, Purge complements REM to enable it to retain its inherent capabilities in the presence of misbehaving flows. This is marked by the overall performance of Purge that outperforms BLACK in providing fairness and global rate enhancement.

Acknowledgments

The authors would like to thank Frédéric Giroire for providing the MINCOUNT source-code. Also thanks are paid to to Y. Tian, R. Rawnsley, and S. Margetts for useful discussions and valuable suggestions.

References

[1] S. H. Low and R. Srikant, "A mathematical framework for designing a low-loss, low-delay internet," *Networks and Spatial Economics*, vol. 4, no. 1, pp. 75–101, 2004.

[2] S. Shakkottai and R. Srikant, "Network optimization and control," *Foundations and Trends in Networking*, vol. 2, no. 3, pp. 271–379, 2007.

[3] S. Athuraliya, S. H. Low, V. H. Li, and Q. Yin, "REM: active queue management," *IEEE Network*, vol. 15, no. 3, pp. 48–53, 2001.

[4] H. Zhang, Z. Jiang, Y. Fan, and S. Panwar, "Optimization based flow control with improved performance," *Communications in Information & Systems*, vol. 4, no. 3, pp. 235–252, 2004.

[5] G. Abbas, A. K. Nagar, H. Tawfik, and J. Y. Goulermas, "Quality of service issues and nonconvex network utility maximization for inelastic services in the internet," in *Proceedings of the 17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '09)*, pp. 537–547, London, UK, September 2009.

[6] C.-W. Chang and B. Lin, "A simple mechanism for throttling high-bandwidth flows," *Research Letters in Communications*, vol. 2008, Article ID 704878, 5 pages, 2008.

[7] G. Chatranon, M. A. Labrador, and S. Banerjee, "Black: detection and preferential dropping of high bandwidth unresponsive flows," in *Proceedings of International Conference on Communications (ICC '03)*, vol. 1, pp. 664–668, Anchorage, Alaska, USA, May 2003.

[8] S. Yi, X. Deng, G. Kesidis, and C. R. Das, "A dynamic quarantine scheme for controlling unresponsive TCP sessions," *Telecommunication Systems*, vol. 37, no. 4, pp. 169–189, 2008.

[9] D. McPherson, "2% of internet traffic raw sewage," Tech. Rep., ARBOR Networks, March 2008.

[10] M. Hachman and B. Heater, *Twitter Hit by DDoS Attack; Other Sites Wobble*, PCMag, 2009.

[11] M.-S. Kim, Y. J. Won, and J. W. Hong, "Characteristic analysis of internet traffic from the perspective of flows," *Computer Communications*, vol. 29, no. 10, pp. 1639–1652, 2006.

[12] Y. Pessach, "UDP delivers: take total control of your networking with. NET and UDP," *Microsoft MSDN Magazine*, pp. 56–65, 2006.

[13] R. Bennett, "Bittorrent declares war on VoIP, gamers: the next internet meltdown," *Networks*, The Register, December 2008.

[14] H. Schulze and K. Mochalski, "Internet study 2008/2009," Research Report, ipoque, February 2009.

[15] L. Tan, C. Yuan, and M. Zukerman, "A price-based internet congestion control scheme," *IEEE Communications Letters*, vol. 12, no. 4, pp. 331–333, 2008.

[16] W. Zhang, L. Tan, and G. Peng, "Dynamic queue level control of TCP/RED systems in AQM routers," *Computers and Electrical Engineering*, vol. 35, no. 1, pp. 59–70, 2009.

[17] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[18] L. Tan, W. Zhang, G. Peng, and G. Chen, "Stability of TCP/RED systems in AQM routers," *IEEE Transactions on Automatic Control*, vol. 51, no. 8, pp. 1393–1398, 2006.

[19] G. Chatranon, M. A. Labrador, and S. Banerjee, "A survey of TCP-friendly router-based AQM schemes," *Computer Communications*, vol. 27, no. 15, pp. 1424–1440, 2004.

[20] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: stabilized RED," in *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societie (INFOCOM '99)*, vol. 3, pp. 1346–1355, New York, NY, USA, March 1999.

[21] M.-K. Chan and M. Hamdi, "An active queue management scheme based on a capture-recapture model," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 572–583, 2003.

[22] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Transactions on Database Systems*, vol. 28, no. 1, pp. 51–55, 2003.

[23] P. Flajolet and G. Nigel Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, no. 2, pp. 182–209, 1985.

[24] F. Giroire, "Order statistics and estimating cardinalities of massive data sets," *Discrete Applied Mathematics*, vol. 157, no. 2, pp. 406–427, 2009.

[25] M. Peyravian and J. Calvignac, "Fundamental architectural considerations for network processors," *Computer Networks*, vol. 41, no. 5, pp. 587–600, 2003.

[26] G. Xie, G. Zhang, J. Yang, Y. Min, V. Issarny, and A. Conte, "Survey on traffic of metro area network with measurement on-line," in *Proceedings of the 20th International Teletraffic Congress (ITC '07)*, vol. 4516 of *Lecture Notes in Computer Science*, pp. 666–677, Ottawa, Canada, June 2007.

[27] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, New York, NY, USA, 1991.