Fast Convergence to Network Fairness

Ageliki Tsioliaridou and Vassilis Tsaoussidis
Dept. of Electrical and Computer Engineering
Demokritos University of Thrace, Greece
Email: {atsiolia, vtsaousi}@ee.duth.gr

Abstract
Most AQM algorithms, such as RED, assure fairness through randomness in congestion notification. However, randomness results in fair allocation of network resources only when time limitations are not considered. This is not compatible with the current Internet, where traffic oscillations are frequent and the demand for fair treatment is rather urgent, due to short duration of most applications. Given the short duration of most modern Internet applications, fast convergence to fairness is necessitated. In this paper, we use fairness as the major criterion to adjust traffic and present a corresponding algorithm of active queue management, which is called Explicit Global Congestion Notifier (EGCN). EGCN notifies flows almost simultaneously about incipient congestion by marking packets arriving at the router’s queue, when the load in the network increases and buffer overflow is expected. This is a new approach compared with the random notification policy of RED or ECN. EGCN distributes the burden to adjust backward to more flows and consequently allows for smoother window adjustments. We elaborate on the properties of system-wide response in terms of fairness, smoothness and efficiency. Simulation results demonstrate a clear-cut advantage of the proposed scheme.

Key words:
fast convergence to fairness, system smoothness, congestion control, performance evaluation

1. Introduction

The variety of applications in the modern Internet has increased the management complexity of Internet resources. Traditionally, models that
described the Internet traffic patterns, user behavior and flow characteristics were simplified but with minor cost to the reliability of results. For example, the traditional models that determined the nature of congestion control, fairness, and efficiency of transport protocols, initially relied on assumptions of synchronized feedback, unified behavior of flows, and unrestricted time availability. The latter assumption in particular cannot really hold further; flows typically do not carry long FTP traffic and modern network resources allow for transfers of huge amount of data within milliseconds. Thus, the temporal dynamics of flows need to be incorporated in current studies. For example, it is not that important to investigate if a system converges to equilibrium but instead, to demonstrate that convergence time does not exceed the typical lifetime of a flow.

Departing from that observation, we argue that with existing mechanisms short-lived flows do not have the time to reach their fair-share during contention and thus they receive an unfair treatment from the network. This observation is not entirely new. For example, Guo and Matta [10] have presented results that reveal how packet drops damage short TCP connections performance and propose preferential treatment for short connections at the bottleneck link queues. Here, we propose a novel scheme for congestion management at the router, which targets faster system convergence to fairness; that is, flows can reach their fair-share faster, and this, as a side-effect and not as a targeted objective, becomes clear benefit for short-lived flows.

Most Active Queue Management (AQM) mechanisms incorporate randomness in packet dropping in order to guarantee fairness. However, since the responsive behavior of a system is only partially affected by random drops (i.e., a few flows will respond when contention increases), fairness within short time intervals cannot be guaranteed, given the heterogeneous nature of Internet flows and their corresponding diversity in duration. Fast convergence to equilibrium requires a system-wide, synchronous and responsive behavior, which, by and large, is feasible only when notification is global and responses are homogeneous, i.e. all flows respond to the same congestion event. In this context, a fairness policy that is originated by random drops may fail in its objective, due to time constraints. That is, notification itself becomes a time-consuming operation; unlike a possibly global notification, that is implemented as a simple event, which requires instants of time. That said, ”Global Notification” is inherently a more appropriate policy to achieve fast convergence to fairness. Furthermore, Global Notification exhibits other desirable properties as well. For example, it allows for modest reactions to
congestion events since the task of alleviating congestion could be shared by more flows.

Global notification is activated when congestion risk is detected. The proposed scheme relies on a simple concept: flows that synchronize their transmission tactics allow for a smoother per-flow behavior. This is true since the cumulative result of network utilization can be better managed if more flows contribute to congestion management with minor transmission gaps per flow, instead of less flows with more gaps. That is, more flows reduce their window size less instead of less flows reducing their window more. Hence, a system that synchronizes flows intentionally with the aim to alleviate congestion faster and with minor impact on each flow, makes sense.

Furthermore, system-wide response allows for smoother window adjustment at the sender and permits controlled system operations between the knee and the cliff \[5\]. Since more flows decrease their window at the same time, a smoother window adjustment is possible. We show experimentally that improvement of system efficiency and fairness is feasible; we also show that the proposed scheme enhances smoothness without compromising responsiveness.

Within this framework we propose an avoidance and control scheme that aims at notifying a large portion of flows about incipient congestion; we call this scheme Explicit Global Congestion Notifier (EGCN). When incipient congestion is detected, EGCN algorithm notifies competing flows by marking the ECN bit \[6\] in the IP header of the incoming packets. Note that explicit notification is rather necessary in order to accomplish a system-wide, synchronous response to congestion: firstly, the loss of the three handshake ACKs packets is avoided, which can cause severe performance degradation; secondly, it reduces timeout events.

Although the implementation of the proposed congestion control mechanism requires the contribution of both the transport and network layers, minor modifications are needed, which, moreover, do not require header extensions. The ECN bit, which is used for the network congestion feedback, already exists in the IP header; and the new algorithm does not involve much computational burden. The sole serious restriction for the deployment of the proposed algorithm is its associated buffer requirements. EGCN requires relatively large buffers at the routers; large enough to simultaneously accommodate packets from most flows. This is how packets of all competing flows will be marked for the same congestion avoidance event. In this paper, we do not investigate thoroughly the impact of the buffer size, how-
ever, we do not consider large buffer as a negative issue, since they eliminate
the problem of RTT diversity; indeed, the impact of propagation delay in
the RTT estimation is minor compared to that of the queueing delay during
congestion.

In our discussion, we depart from one-dimensional systems and incorpo-
rate a second dimension, namely, time. We monitor the system in short time
slots and study its equilibrium dynamics, in static and dynamically-changing
environments. At this stage of our work, we highlight the importance of
achieving fairness fast; and we quantify the potential gain in terms of fair-
ness, smoothness and efficiency. For this reason, a variety of metrics have
been used to capture system performance in short and long time periods. In
the same context, we propose two new metrics to capture system smoothness,
namely the MaxCwndDeviation and the MaxAvgCwndDeviation.

The rest of the paper is organized as follows: Section 2 details background
and related work, while in Section 3 we discuss further the fairness and
smoothness dynamics. Next, in Section 4, we detail the proposed congestion
control mechanism, discuss briefly its design guidelines and refer how EGCN
differs from RED(ECN) and DropTail. Finally, in Sections 5 and 6 we present
our experimental results and highlight our conclusive remarks.

2. Background and Related Work

Convergence to fairness has been a topic that attracted significant at-
tention from the networking research community. Several dimensions of this
topic have been highlighted in the recent past. Here we categorize algorithms
in three groups based on their perspective to system fairness.

The first group is consisted of algorithms that investigate long-term fair-
ness. Algorithms in this category consider traffic as bulk data transfer and
study bandwidth sharing among competing flows. Their goal is to reach a
stable operating point, where bandwidth is allocated equally among compet-
ing flows. This category includes the Additive Increase Multiplicative
Decrease (AIMD), algorithm which is based on the analysis of Chiu and Jain
[5]; from the fairness perspective, multiplicative decrease gradually ”releases”
those resources that have been allocated unfairly and the additive increase al-
llocates new resources fairly. A recent improvement of AIMD, Additive Increase
Multiplicative Decrease with Fast Convergence (AIMD-FC) was proposed in
[13]. AIMD-FC algorithm by Lahanas and Tsaoussidis introduces an optimiza-
tion to AIMD during the convergence procedure that enables the algorithm to
converge faster and achieve higher efficiency. Binomial algorithms [3], similarly to AIMD, possess the convergence property under simplified conditions of synchronized feedback to sources. SIMD [12] and GAIMD [19] are algorithms also included in this category. Both of them consume the same amount of bandwidth as AIMD, however they reach network equilibrium later than AIMD but improve system smoothness.

Algorithms in the second category attempt to differentiate application services based on flows’ lifetime and treats packets of short-lived flows preferentially to improve system fairness; they rely on the fact that short-lived flows do not have the time to reach their fair-share during contention and thus they receive an unfair treatment from the network. Guo and Matta [10] provide a detailed background and motivation for preferentially serving the packets of short-lived flows since the impact of loss is more significant on them. Many algorithms have been proposed at the network layer to improve system fairness by favoring flows with short- over long lifetime. We briefly present some of them. NCQ [14], [15] is a scheduling algorithm advocates scheduling packets according to their size and gives priority to short flows. The logic is that non-congestive flows, such as sensor applications, transmit periodically small packets and do not really contribute to congestion; hence, they should not suffer from delays. Authors in [11] propose scheduling packets according to the Least Attained Service (LAS). According to this policy, the router would track the work it has already performed on each flow and serve the next packet from that flow, which had the worst service, currently. The LAS policy requires routers to maintain per-flow state, and to potentially maintain per-flow queues, which make administration rather impractical. There are also algorithms in this category that use randomness aiming at penalizing long flows. RED [9] is such a simple algorithm, which aims penalizing greedy flows that consume more bandwidth than others. It is based on the fact that short flows spend the most part of their life-time in the initial slow start phase while long flows have relatively long congestion windows. Ramakrishnan and Floyd in [6] proposed an Explicit Congestion Notification (ECN) to be added to the IP protocol in order to trigger TCP congestion control. Since then ECN mechanism has been incorporated in RED algorithm. ECN enables routers to probabilistically mark a bit in the IP header, rather than drop the packet, to inform end hosts of pending congestion when the length of the queue exceeds a threshold. End hosts multiplicatively reduce their congestion windows upon receiving packets with ECN bit set, before the router buffer overflows and packet drops are
inevitable. A duality is served with ECN: TCP performance can be enhanced by means of avoiding losses of data windows due to limited buffer space at the bottleneck router, and congestive collapse can be avoided. The RED-PD algorithm (RED with Preferential Dropping) [2] proposed by Floyd uses the packet drop history at the router to detect high-bandwidth flows in time. CSFQ [17] estimates the incoming rate of each flow, and use it to label flow’s packets. The router uses this information in conjunction with the flows’ fair-share estimation, in order to decide whether a packet should be dropped. Flow-RED(FRED) is a modified version of RED, which uses per-active-flow accounting to impose on each flow a loss rate that depends on the flow’s buffer use. FRED is more likely to drop or mark packets from sources with a higher number of packets queued.

Finally, approaches in the third category incorporate time as a new dimension in fairness definition. Algorithms of this category target faster system convergence to fairness; this is a clear benefit for short-lived flows. In this paper, we propose a congestion avoidance and control scheme which allocates resources faster and allows each flow to consume bandwidth close to its fair-share. Our initial work is presented in [18], where we proposed a rather simple mechanism, called Global Notifier(GN), to notify a large portion of flows about incipient congestion. In this paper, we extend our evaluation and results using a new congestion control mechanism, namely Explicit Global Congestion Notifier (EGNC).

3. Fairness and Smoothness

Fairness is a major issue in all resource allocation schemes, where demand exceeds the supply. However, it is often measured as an index without incorporating its temporal characteristics [4]. For example, Chiu/Jain fairness index [5] captures throughput per flow without incorporating time as a dimension. Although time is incorporated in throughput, since it is a factor in both in numerator and denominator, it becomes void. Fairness Index is defined as:

\[
\text{FairnessIndex} = \frac{\left( \frac{\sum_{i=1}^{n} x_i}{n} \right)^2}{n \sum_{i=1}^{n} (x_i)^2}
\]  

(1)
where, 
\(x_i\) is the normalized throughput of the \(i_{th}\) TCP flow and 
\(n\) is the number of connections.

The *WorstCase Fairness* index is also used to measure fairness from another perspective. It captures cases where the system is very unfair to a small number of flows, something that will not be reflected by *Chiu/Jain fairness* index. It is defined as:

\[
WorseCaseFairnessIndex = \frac{\min_{1 \leq i \leq n} \{x_i\}}{\max_{1 \leq i \leq n} \{x_i\}}
\]  

(2)

where,
\(x_i\) is the throughput of the \(i_{th}\) TCP flow.

Both the above fairness metrics are designed to measure the fairness behavior of bulk data transfers in a long period of time. However, they do not suffice for measuring how fast a system converges to fairness: Consider a case where two flows compete for the same bandwidth; one of them consumes bandwidth close to its fair-share while the other experiences larger deviation, see Figure 1. Although at time \(t_o\) both flows have consume the same bandwidth and *Chiu/Jain fairness* index approaches to 1, bandwidth is not allocated fairly in a shorter time period since one of the two flows consumes more bandwidth than the other in short time slots. Moreover, the current Internet is characterized by application diversity: some applications last for a short time, while others have relatively longer lifetime. It is obvious that a fairness index that incorporates time as well is needed. That is, fairness convergence should also capture how fast resources can be allocated fairly. For this reason, in our evaluation experiments we use the *ShortTermFairness* index, which captures how resources are allocated among flows within short time slots:

\[
ShortTermFairness = E_t\{\left(\frac{\sum_{i=1}^{n} x_i}{n \sum_{i=1}^{n} (x_i)^2}\right)^2\}
\]  

(3)

*ShortTermFairness* index is defined similarly to *Fairness* index, however, it is sampled in short time scales (usually 2-3 RTTs) and provides a measure
of fairness with better granularity.

![Figure 1: Throughput allocation in time](image)

Smoothness is another important criterion for the evaluation of a notification scheme, since window adjustment is explicitly related to the frequency of congestion notification. Along the same lines, in order to investigate and compare the window value and adjustment behavior of competing flows, in [18], we proposed two new metrics to measure system smoothness, the \( \text{MaxAvgCwndDeviation} \) and the \( \text{MaxCwndDeviation} \).

\( \text{MaxAvgCwndDeviation} \) captures the maximum gap of the average congestion window for a time duration \( t \); while \( \text{MaxCwndDeviation} \) reflects the maximum gap of congestion windows that are regularly sampled every \( t_{\text{current}} \). Therefore, the \( \text{MaxCwndDeviation} \) metric attempts to capture oscillations that may be missed by the \( \text{MaxAvgCwndDeviation} \) metric. In this context, we use both a short- and a long-term metric to capture the window oscillations and system smoothness. That said, we define \( t_{\text{current}} \) as the time instant, where samples are taken; and we calculate the average window and deviation at every such instant. The granularity of measurements has been set equal to a time-slot scale that corresponds to 2-3 RTTs. Unlike \( t_{\text{current}} \), time \( t \) is defined as the period from \( t_0 \) to \( t_{\text{current}} \).

Consequently, we define \( \text{MaxAvgCwndDeviation} \) and \( \text{MaxCwndDeviation} \), respectively, as:

\[
\text{MaxAvgCwndDeviation} = \max_{1 \leq i \leq n} |\text{SystemAvgCwnd}(t) - \text{awnd}_i(t)| \quad (4)
\]

\[
\text{MaxCwndDeviation} = \max_{1 \leq i \leq n} |\text{avg_cwnd}(t_{\text{current}}) - \text{cwnd}_i(t_{\text{current}})| \quad (5)
\]
where, \( awnd_i \) is the average congestion window of the \( i^{th} \) flow, \( cwnd_i \) is the congestion window of the \( i^{th} \) flow, and \( SystemAvgCwnd \) is the system-wide average congestion window given by the following equation:

\[
SystemAvgCwnd(t) = \frac{\sum_{i=1}^{n} awnd_i}{n}
\]  

(6)

\( avg_cwnd(t_{current}) \) is the average of all flows’ congestion window size at time \( t_{current} \) and is defined as:

\[
avg_cwnd(t_{current}) = \frac{\sum_{i=1}^{n} cwnd_i(t_{current})}{n}
\]  

(7)

Note, that the proposed metrics allow for short-term as well as long-term measurement of system fairness, since (4) captures the coarse-grained smoothness and (5) the fine-grained smoothness.

4. Proposed Congestion Control Mechanism

The proposed congestion control mechanism is consisted of two collaborating sub-mechanisms. The first one relies on an AQM algorithm at the network layer, while the other incorporates a transmission strategy at the transport layer. Note that collaboration here is a necessary condition for success; that is, each mechanism alone cannot achieve any gains. This condition, however, is not required by our scheme only, but for all congestion and avoidance schemes, presently. In this context, our inherent assumption of collaborative layers holds. This means that the router will (implicitly or explicitly) notify senders about incipient congestion and will expect a corresponding response: a typical interaction between TCP and DropTail or RED. We investigate here both the notification and the response strategies.

4.1. Explicit Global Congestion Notifier

The EGCN algorithm is an AQM algorithm which marks incoming packets, based on (i) the absolute value of the current average queue size and (ii) the variation of the average queue size. In Figure 2, the EGCN algorithm is briefly presented through a diagram. Upon the arrival of each packet, the average queue size is being estimated. The equation for the average queue size is as in RED, that is:
If (queue is not empty)
\[ avg = (1 - w_q)avg + w_q q \]
else
\[ avg = (1 - w_q)^m avg \]
where,
- \( avg \) is the average queue size
- \( q \) is the current queue length
- \( w_q \) is the queue weight
- \( m \) is the number of packets that could have been transmitted by the gateway during the idle period

If the buffer is fully utilized, the packet is being discarded, otherwise it checks the variation of the current \( avg \). If the \( avg \) is decreased, the incoming packet is simply buffered, otherwise, a decision has to be made whether the arriving packet is going to be dropped, marked or just buffered based on the value of \( avg \). More particularly, EGCN uses two thresholds: a minimum
threshold \( (\text{min}_{th}) \) and a maximum threshold \( (\text{max}_{th}) \). If a packet arrives when the queue average is below \( \text{min}_{th} \), the packet is enqueued. If the \( \text{avg} \) lies between \( \text{min}_{th} \) and \( \text{max}_{th} \), the ECN-bit of the incoming packet is set to 1 with a probability \( p_a \). The packet marking probability \( p_a \) varies linearly from 0 to 1, as \( \text{avg} \) oscillates between \( \text{min}_{th} \) and \( \text{max}_{th} \) and is given by the following equation:

\[
p_a = \frac{(\text{avg} - \text{min}_{th})}{(\text{max}_{th} - \text{min}_{th})} \tag{8}
\]

The marking probability follows the \( \text{avg} \) position between \( \text{min}_{th} \) and \( \text{max}_{th} \), which clearly depicts the load in the buffer. As the load increases the probability increases as well. Finally, if the \( \text{avg} \) is above \( \text{max}_{th} \) the incoming packet is marked.

It is worth noticing that the proposed algorithm preferably applies marking instead of dropping incoming packets. This is because flow synchronization cannot be accomplished by packet drops mainly, due to asymmetric timeout values. Moreover, packet marking minimizes the probability of starvation due to loss potential of control packets (e.g. SYN, SYN-ACK).

The EGCN thresholds are used as the regulating tool for controlling the amount of notified flows. This was deemed necessary due to the possibility of faster system response to congestion: a higher value for \( \text{min}_{th} \) could assist in avoiding link under-utilization.; while a lower value of \( \text{max}_{th} \) thresholds could guarantee global and synchronized notification, and packet drop avoidance. We found experimentally that the optimal values of \( \text{min}_{th} \) and \( \text{max}_{th} \) are:

\[
\text{min}_{th} = \text{buffersize}/4 \tag{9}
\]

\[
\text{max}_{th} = (5/4)\text{min}_{th} \tag{10}
\]

The value of \( \text{min}_{th} \) and \( \text{max}_{th} \) thresholds should be chosen to guarantee synchronous notification of all competing flows. When the level of contention is low, randomness can guarantee global and synchronous notification. However, as the link contention increases, system-wide synchronous notification requires marking\(^2\) of all arriving packets when the average queue length exceeds \( \text{max}_{th} \).

---

\(^2\)The EGCN algorithm does not deal with cases where many packets in a row are marked. In order to prevent multiple window reduction for the same incipient congestion event, in the presence of burst marked packets, sources adjust backwards only once.
4.2. Window Adjustment

Since our algorithm triggers a synchronized reduction of multiple flow windows, the aggregate system response to incipient congestion is effective, and based on the typical adjustment strategies, could be rapid as well. This calls for a smoother window adjustment to prevent link under-utilization. Therefore, adjustments need to be smooth enough to guarantee sufficient utilization; and sharp enough to preserve system equilibrium.

Noteworthy, the window adjustment is correlated to the level of contention at the bottleneck link. That is, high level of contention requires smoother window adjustment than lower level, since there are more flows that contribute simultaneously to congestion avoidance. We therefore need to incorporate a more dynamic window adjustment scheme. More particularly, based on previous work [16], we attempt to define three different thresholds and classify contention based on each flow measurement according to its maximum congestion window ($cwnd_{\text{max}}$) value. This corresponds roughly to the current level of flow contention. We define three different thresholds ($\text{Threshold}_1$ to 3); $\text{Threshold}_1$ corresponds to high contention, $\text{Threshold}_2$ to medium contention while $\text{Threshold}_3$ refers to the situation where few flows share a link. We also assign three sets of window adjustment ($a_1$, $b_1$), ($a_2$, $b_2$) and ($a_3$, $b_3$) corresponding to the three predetermined intervals ($0, \text{Threshold}_1$), ($\text{Threshold}_1, \text{Threshold}_2$) and ($\text{Threshold}_2, \text{Threshold}_3$), where $a_1 < a_2 < a_3$ and $b_1 > b_2 > b_3$. The choice of additive increase rate $\alpha$ and multiplicative decrease rate $\beta$ is according to the level of contention. By tuning these parameters we improve both smoothness and responsiveness. Note that the adjustment itself is the fairness tool of the participating flows, since, this way, the transmission gap among different windows is reduced.

The aforementioned contention-estimation scheme allows for dynamic adjustments of congestion window, based on the current level of contention at the bottleneck link. That is, this scheme can provide roughly the basis for calculating how many flows will share the burden of backward adjustments: the more the flows, the smoother the transmission rate per flow.

The proposed mechanism EGCN results in a synchronous responsive behavior to network changes; flows are notified almost simultaneously about incipient congestion and adjust their sending rate close to their fair-share. Each flow obtains a max-window value per epoch, which corresponds to the current level of contention. That said, we followed a simple but reliable methodology for the estimation of contention. The sender measures contention by means of the congestion window. More precisely, we employ the
following idea:

During communication, the sender detects contention by monitoring the maximum value of its congestion window per epoch. When contention increases, the sender may experience congestion with much smaller window, and vice versa.

Therefore, with a small reliability risk we adjust the window to correspond to the level of contention that is, in turn, estimated by the maximum value of the congestion window by each sender. In order to allow some flexibility at the transmission rates, we establish our adjustment scales to correspond to high, medium and low contention. Thresholds 1 to 3 are set 10, 30 and 50, respectively and the corresponding parameters are set as follows:

**High contention** \((a_1=1, b_1=0.75)\)

**Medium contention** \((a_1=0.9, b_1=0.7)\)

**Low contention** \((a_1=0.8, b_1=0.65)\)

In order to implement progressively smoother behavior, we integrate this congestion window adjustment strategy into Adaptive TCP (ATCP).

4.3. Discussion

The main design goal for EGCN gateways is to achieve fast fair allocation of network resources among competing flows. Chiu and Jain model[5] was based on the inherent assumption that all competing flows reduce their sending rate simultaneously. It is obvious that through synchronous flows backward responses, the system converges to fair bandwidth allocation faster.

A widely-deployed dropping mechanism, which causes global synchronization, is DropTail. The DropTail mechanism drops incoming packets when the queue memory resources are completely utilized. Due to synchronous packet loses at the router, flows experience congestion event almost simultaneously and flow synchronization occurs. However, burst packet drops result in numerous undesirable events, such as excessive packet looses, queue underutilization, large queue fluctuation and system performance degradation. To avoid synchronous packet drops, EGCN, unlike DropTail, marks incoming packets instead of dropping them. The idea is to synchronize flows, albeit to prevent buffer overflow. In this concept, when incipient congestion is detected, EGCN marks the ECN bits of the packets that are buffered at the router.
Since flows are notified about congestion almost simultaneously, in the presence of EGCN, the aggregate window follows a "sawtooth" pattern in Figure 3 and the adjustments of average queue length is captured to that in Figure 4. More particularly, due to global and synchronous notification, senders increase their window almost simultaneously; at this stage, the load exceeds the link capacity and bottleneck queue steadily starts building up. The flows continue to additively expand their window sizes, until the average queue reaches the minimum threshold. At this point, the ECN bit of the packets that are buffered at the router is marked with probability $p$, which increases with the average queue size. Senders receive the ECN packet and reduce their sending rate. At this stage, the system has responded to incipient congestion; the average queue length is decreasing but no packets need to be marked. Figure 5 depicts the value of mark probability in proportion to the variation of the average queue length.
EGCN vs. RED(ECN)

Although both AQM [6] mechanisms, RED(ECN) and EGCN, rely on explicit notification, the sending rate of corresponding flows is adjusted differently in the presence of the two mechanisms. In particular, RED(ECN) scheme marks incoming packets randomly when the average queue size is between the min and max thresholds of RED. Note that, its target is to notify competing flows randomly and break flow synchronization. Contrary, EGCN aims to notify flows almost simultaneously about incipient congestion by incorporating the variation of the average queue size in its notification scheme. EGCN considers that once the system responds to congestion event, the load in the network is decreased and no further retreat is needed.

Another difference between RED(ECN) and EGCN mechanisms is that RED(ECN) incorporates early dropping mechanism while EGCN drops packets only when there is no available space at the router. In particular, when the rate of incoming packets is high the explicit notification scheme of RED(ECN) fails to keep the load at low levels; as a result the average queue size exceeds maximum threshold and incoming packets are discarded, trying to avoid buffer overflow. As a consequence, when the contention of the link is high, RED(ECN) requires more time to notify all flows. Through simulation we show that, when the number of competing flows is high, the explicit notification mechanism of RED causes significant degradation of system efficiency. Practically, the situation worsens when contention increases.

RED(ECN) and EGCN mechanisms also allow for different dynamic of smoothness. EGCN notifies flows almost simultaneously and flows adjust their sending rate close to their fair-share; consequently flows experience small window oscillation. RED(ECN) notifies flows randomly and flows adjust their window irregularly, causing more severe adjustments. It is obvious that system smoothness is damaged in the presence of RED(ECN). Through experiments we show that less contention of the link results in worse system smoothness in the presence of RED(ECN). This is due to the fact that it takes more time for a flow to decrease its window, resulting in larger window adjustments.

Finally, we argue that short-term fairness is accomplished when flows operate close to their fair-share, which is not feasible when flows are notified randomly in large period of time. That is why RED(ECN) fails to allocate the bandwidth fairly in short periods of time. In our experiments, we use different time scales to monitor system. More particularly, we measure ShortTermFairness and AllotedThroughput every 2-3 RTTs and every 30 RTTs. However, simulation results reveal that EGCN outperforms RED(ECN)
even in large scale.

**EGCN vs. DropTail**

DropTail is the most well known dropping mechanism that introduces flow synchronization; when the buffer runs out of space, a large amount of incoming packets are dropped and flows decrease their sending rate almost at the same time. However, not all flows experience congestion event, since the packets of some flows are enqueued, resulting in partial system synchronization. Although EGCN results in synchronized window adjustment too, network resources are allocated differently in the presence of EGCN compared with DropTail. That is EGCN notifies almost all flows about congestion and bandwidth is allocated fairly among flows. This is clearly revealed in our experiments, where EGCN achieves much higher values in all fairness indices.

EGCN relies on explicit early notification about congestion and avoids excessive packet drops. Moreover, since flows adjust their sending rate to the level of contention, buffer under-utilization is avoided and system efficiency in terms of Throughput is remarkably improved; EGCN operates between the knee and the cliff.

Finally, EGCN administers queue length better; that is, it keeps it low enough to cancel delay/jitter but sufficiency high to avoid underutilization.

5. Performance Evaluation

In this section, the proposed congestion scheme is evaluated thoroughly in terms of (i) fairness, (ii) smoothness and (iii) efficiency. For this reason, we evaluate and compare the three queueing/dropping policies, namely classic RED(ECN), DropTail and EGCN for a wide range of network conditions.

We follow the following evaluation plan:

We first conduct simulations (Section 5.3) to observe how the system is notified about congestion in the presence of EGCN and attempt to quantify the comparative efficiency of our notification scheme itself. That is, we evaluate the notification mechanisms alone. For this reason, we investigate how long it takes with RED(ECN) and EGCN, respectively, to notify all flows about incipient congestion.

We then study system behavior and convergence to equilibrium (Section 5.4) with EGCN, RED(ECN) and DropTail. More particularly, we study bandwidth tradeoff among flows, flow synchronization and fairness.

In Section 5.5, we monitor and quantify system fairness in the presence of the three aforementioned mechanisms. We compare long- versus short-term
fairness and comment on metrics suitability.

Next, in Section 5.6, we study system behavior with varying link capacity. We study also cases where flows experience different RTTs to capture the performance of the proposed scheme in a more realistic scenario.

Finally, in Section 5.7, we implement experiments in dynamic environments to illustrate how system reacts to sudden traffic changes.

For our simulation experiments, we use the ns-2 network simulator [1]. Unless stated differently, simulation time is fixed to 100 seconds and all senders start transmitting packets within the first 2 seconds. All simulations last long enough to ensure the system has reached a consistent behavior.

5.1. Simulation Setup

In our evaluation scenarios, we use the following set of RED(ECN) and EGCN parameters:

**RED with Explicit Congestion Notification (RED(ECN))**: The RED(ECN) parameters are set according to [7], [8]. That is, use the ”gentle” mode, the maximum threshold is set to three times the minimum threshold, and the minimum threshold is set to 1/8 of the buffer size.

**Explicit Global Congestion Notifier (EGCN)**: EGCN parameters are set according to our proposal in Section 4.1. That is, we \( \text{min}_{th} \) and \( \text{max}_{th} \) are set to 1/4 and 5/16 of the buffer size, respectively.

Moreover, in our experiments we set the TCP parameter \( \text{old}_\text{ecn} \) as true. This setting guarantees that the TCP sender will reduce its window only once in the presence of a burst of marked packets.

Finally, the queue buffer size is set based on the Bandwidth Delay product.

5.2. Performance Metrics

In order to zoom at the speed of convergence to equilibrium rather than simply calculate a fairness index at the end of a long-lived application, we use the _ShortTermFairness_ index. As a reference, we also use a well-known fairness index [5] and the _WorstCaseFairness_ index to describe the long-term behavior of large number of flows.

The _MaxAvgCwndDeviation_ and the _MaxCwndDeviation_ are used to measure system smoothness.
System efficiency is measured by:

- **System Goodput**, which is defined as:

\[
SystemGoodput = \sum_{1 \leq i \leq n} \frac{OriginalData}{TransmissionTime}
\]  

(11)

where \(OriginalData\) is the number of Bytes delivered from a sender to the corresponding receiver during their connection (\(TransmissionTime\)), excluding the size of retransmitted packets of this flow and the overhead induced by packet headers.

- **System Throughput**, which is defined as:

\[
SystemThroughput = \sum_{1 \leq i \leq n} \frac{TotalDataSent}{TransmissionTime}
\]  

(12)

where \(TotalDataSent\) is equal to the sum of original data, retransmitted data and packets’ header size (in Bytes).

Finally, we use **Allotted Throughput** to investigate flow responsiveness in dynamic environments.

\[
AllottedThroughput = E\{\frac{TotalDataSent}{TransmissionTime}\}
\]  

(13)

This allows us to capture the throughput within a short sampling period, usually at a time scale of a few RTTs.

5.3. Notification Efficiency

At the first stage of our experiments we attempt to quantify the level of notification using our notification scheme. We use a single bottleneck link (see Figure 6) shared by multiple flows, where the bandwidth of the links are set to 100Mbps and the Round Trip Time to 40ms. We implement experiments varying the number of flows from 20 to 240 and measure the time it takes with **RED(ECN)** and **EGCN** algorithm, respectively, to notify all competing flows about incipient congestion.
In Figure 7, we present sample but representative \(^1\) results with 120 and 220 flows. We depict the time it takes each mechanism to notify all flows a fixed number of times (here is four times) since simulation start up. Note that we consider as a notification signal either a packet drop or mark.

We see in Figure 7 that the EGCN notifies all flows much earlier than RED(ECN). It is worth mentioning that at the period of time that EGCN has notified all flows four times, RED(ECN) has not notified them even once. Obviously RED(ECN), due to its randomness, needs more time than EGCN to notify all senders. This observation justifies our high expectations regarding the fast convergence to fairness and stands as a notable result on its own right.

\(^1\)More experiments have been conducted with varying number of flows and notification thresholds (i.e., percentages) to confirm the findings.
Figure 8 illustrates the average time for system-wide notification. We notice that the average time for system-wide notification in the presence of RED(ECN) ranges irregularly between 0.5 and 1.2 seconds, due to randomness; EGCN however, exhibits a more stable behavior. More particularly, when the link contention is low (between 40 and 100 flows) and the fair-share of each flow is at least 1Mbps the average notification time approaches 0.5sec. As contention of the link is getting higher (more than 120), EGCN notification frequency increases since the frequency of congestion events increases along with the contention of the link; and senders have to adjust their sending rate more frequently.

Simulation results in Figure 8 show also that RED(ECN) requires more time than EGCN to notify all flows about congestion; this is again due to its inherent randomness. Figures 9 and 10 depict how 40 and 180 flows, that share the same bottleneck link, are notified in the presence of EGCN and RED(ECN), respectively, during the 30th and 32th sec of simulation time. EGCN notifies senders almost simultaneously, while RED(ECN) requires more time. Moreover, in high contention (180 flows) RED(ECN) fails to keep low the load in the link and drops packets; next in our experiments, we show that this results in degradation of system efficiency.

Figure 8: Average time for system-wide notification

5.4. System Convergence to Equilibrium

Next, we study i) how fairly resources are allocated among competing flows and ii) how well the bandwidth of the link is utilized by senders in the presence of DropTail, RED(ECN) and EGCN. We use the topology in Figure (6) and implement a scenario where 23 FTP flows, with the same RTT (30ms), compete for a single link 10 Mbps bandwidth. Three flows start their session
one-by-one every 50 secs (at the 0th, 50th and 100th seconds, respectively), and 20 new flows divided into two equal groups enter the link at the 150th and 200th sec. The scenario settings are chosen in a way to monitor how flows respond to i) the increase of competing flows and ii) different level of contention in the presence of the three aforementioned mechanisms. The buffer size is set to 160 packets and the simulation last for 250 sec, which is enough time to monitor system behavior.

In order to capture system convergence to equilibrium, we monitor the congestion window of the first four incoming flows and the current queue length. We also measure ShortTermFairness every 0.06 secs3 and Allotted Throughput of each flow every 1 ms4. The reason that we use a rather large time-scale is that since RED(ECN) requires more time to notify all senders a

---

3which is approximately 2 RTTs
4which is equally to 33 RTTs
time-scale equal to 2 RTTs may be too short, for RED(ECN).

Figure 11: DropTail with TCP(1,0.5)

Figure 11(a) shows the failure of DropTail to notify all competing flows about congestion events. For instance, the first flow has not been notified to decrease its sending rate during the 50th and 100th of simulation time; as a result, its congestion window is maximized (it is set to 120pckts) consuming more bandwidth than the second flow. Figures 11(b) and 11(c) show that system resources are not equally allocated among flows, consequently the system fails to converge to equilibrium. Finally, Figure 11(d) depicts the drawbacks of partial system synchronization. Due to the large fluctuations of the load in the network, packets may experience large delay/jitter or even be dropped due to buffer overflow, inducing instability into the network and wasting of resources.

Figure 12(a) shows that with RED(ECN), flows increase and decrease their window asynchronously. As a result, a flow may constantly increase its congestion window more than other competing flows, resulting negatively on ShortTerm Fairness (see Figure 12(c)). Furthermore, Figure 12(b) shows that due to random notification, network resources are allocated among competing flows unpredictably and flows cannot approach their fair-share even
in a large time-period. Figure 12(d) show that due to random signals it takes time for each flow to get its turn to respond to congestion and this is reflected by the rapid increase of the current queue length when new flows enter the link.

On the contrary, EGCN manages successfully to reallocate bandwidth among flows (see Fig. 13(b)); even when new flows enter the link their sending rate is adjusted around their fair-share. Figure 13(a) clearly reveals that thank to global notification the maximum congestion window of each flow is almost equal and correspond to the current level of contention. Figure 13(c) shows that system-wide response improves fairness convergence speed and Figure 13(d) proves that system operates between the knee and the cliff points, consequently the system operates near equilibrium. Figure 13(d) illustrates also that EGCN succeeds to cancel the typical problems of flow synchronization.

5.5. System Fairness

In this Section, we study EGCN short-term and long-term system fairness. We also attempt to evaluate the accuracy of different fairness indices along with their suitability for different scenarios.
We implement the same set of experiments as in Section 5.3, where a number of flows compete for a 100Mbps link and they all experience the same RTT, equal to 40ms. We vary the number of FTP sessions from 20 to 240 to represent cases of low, medium and high contention.

Figure 14 shows the results of D. Chiu’s and R. Jain’s index, when experimental time is sufficiently long (e.g. 100sec), ATCP with EGCN and TCP-Reno
with RED(ECN) allocate bandwidth among FTP flows fairly ($Fairness$ index is close to 1). On the contrary, DropTail fails to allocate resources fairly, especially when link contention is low; DropTail simplicity is associated with uniformly-distributed packet losses and consequently multiple packet losses may occur at the same flow.

![Figure 15: System WorstCase Fairness](image)

Figure 15 illustrates the results based on $WorstCase$ $Fairness$ index for the same set of experiments. As mentioned earlier, this metric is more sensitive and may capture unfairness with better granularity. The results illustrate that the proposed congestion control scheme treats flows more fairly than the other two schemes: EGCN improves $WorstCase$ $Fairness$ Index dramatically compared to DropTail (see case with 20 flows) and significantly compared to RED(ECN) (see case with 200 flows).

Next, we use the short-term fairness index and monitor how the bandwidth is temporarily allocated among the flows. Figure 16 shows the maximum, minimum and average values of $ShortTerm$ $Fairness$ after the $5^{th}$ sec, when the system is considered stable. The variance of $ShortTerm$ $Fairness$ values in Figures 16(a) and 16(b) reveal the unfair allocation of resources through out the experiment. Instead, short-term fairness is better approached, by EGCN (see Fig. 16(c)). Even when the $ShortTerm$ $Fairness$ gets similar values in the presence of RED(ECN) and EGCN (see the cases of 80, 100 and 120 flows) the variance of the window values differ. More particularly, the $MaxCongestionDeviation$ index in Figure 17 shows that i) randomness results in sharper window fluctuations than EGCN and ii) EGCN improves system smoothness as the number of competing flows increases: the value of $MaxAvgCongestionDeviation$ index stabilizes after the 15th sec with RED(ECN) and much sooner with EGCN.
We also highlight that our proposed scheme exploits well the fairness/efficiency tradeoff: it does not improve system fairness at the cost of efficiency. Figures 19 and 20 depict system performance in the presence of the three mechanisms. DropTail causes multiple TCP connections to suffer losses simultaneously. RED(ECN) cancels synchronization and improves performance as long as link contention is low enough to notify all flows uniformly but fails to maintain...
this property when contention grows. Under such conditions, the rate of incoming packets at the router is high, the average queue length exceeds the maximum threshold and packet dropping increases. Therefore, Throughput increase is justified by packet drops and retransmissions and does not correspond to extra Goodput (compare Fig. 19 and 20). On the contrary, with EGCN flows operate close to their fair-share and system operates around the knee and the cliff, regardless of the level of contention. Fig. 20 shows that EGCN achieves better Goodput without extra effort, since system Throughput approaches link bandwidth (see corresponding results of Throughput Fig. 19).

5.6. System Performance

So far, we have evaluated our congestion control scheme with a simple scenario, where a number of FTP flows with the same Round Trip Time share a single bottleneck. We studied the performance of the proposed scheme in low- and high contention; the results show that our scheme outperform
RED(ECN) in terms of fairness and smoothness; however the conditions and parameters of the experiments were fixed. In this section we evaluate the proposed scheme under varying network conditions.

First, we study the impact of the link capacity on the performance of EGCN. For this set of experiments we use a dumbbell topology, where 20 FTP flows share a single link and experience the same RTT equal to 40ms. Experiments in the last section reveal that RED(ECN) causes multiple droppings when contention is high. Therefore, we choose a rather small number of flows (20 flows) that allows us to compare better system behavior when flows are notified explicitly in the presence of RED(ECN) and EGCN.

Then we study the performance of the proposed algorithm in a more realistic scenario, where competing flows experience different RTTs. We use the topology in Figure 6) and we set the delay at the senders’ side randomly from 0.01sec to delay_{max}. In order to uncover any potential undesirable property of EGCN due to RTT diversity we vary the value of delay_{max} from 1ms to 5 ms.

**Impact of Bottleneck Capacity**

In this set of experiments we range the bandwidth at the bottleneck between 2Mbs and 300Mbps. Figures 21(a) and 21(b) show the maximum, minimum and average values of short-term fairness. It is obvious that EGCN leads to better system smoothness. Note that the EGCN average value of ShortTerm Fairness index reaches values beyond 0.9; unlike RED for which that value is the upperbound. Figures 24(b), 24(a), 33(c) and 33(b) confirm our claims that better convergence to fairness in terms of short time slots improves system smoothness. The variance of the window values is significant (more than half the measuring scale). The results of MaxCongestionDeviation and MaxAvgCongestionDeviation indices confirm that EGCN manages to reform the operational spectrum of flows around their fair-share regardless of the bandwidth availability (see Figure 22). Finally, results in Figure 23 strengthen our claim that EGCN allocates better the available bandwidth among the flows further.
Impact of Different RTT
In this set of experiments, we consider a number of flows competing for the same 100Mbps bottleneck. As mentioned earlier, propagation delays at the
senders’ side are set randomly from 0.01sec to delay\textsubscript{max}. We perform these sets of experiments with (i) low contention (40 flows) (ii) medium contention (100 flows) and (iii) high contention (200 flows).

We conclude, based on Figures 26 and 27, that EGCN competes RED(ECN). The results of ShortTerm Fairness show that with EGCN the allotted bandwidth-per-flow reaches the actual fair-share faster than the other two notification schemes. Finally, Figures 31 32 and 33 reveal that flows experience smoother window adjustment with EGCN.
5.7. Dynamic Network Conditions

In this scenario, we use a more complex topology (see Figure 34), where 10 FTP flows (Sender 1 to 10) send packets for 5 sec every 10 sec. This creates a situation where new incoming flows with short lifetime compete for link resources. Apart from these 10 FTP connections, we included 10 more FTP cross traffic (Peripheral Sender 1 to 10), which start their session
at 0sec and finish at the end of the experiment. This creates a situation where FTP flows with long lifetime have to adjust their sending rate properly, according to the dynamics of link contention. This scenario, therefore, is appropriate to measure: ShortTerm Fairness, packets send, packets received and the MaxCongestionDeviation.

Figures 35 and 36 demonstrate EGCN’s superiority over RED(ECN) and DropTail schemes, where ShortTermFairness and Max CongestionDeviation indices re-
reflect the corresponding improvement in system fairness and smoothness with EGCN. Remarkably, this fairness and smoothness superiority of EGCN does not come at the expense of bandwidth utilization (see Figure 37).
6. Conclusions

We claim that fairness can be achieved faster when congestion notification is global; and smoothness can be enhanced by synchronization. We proposed a new notification mechanism, namely EGCN, that focuses on synchronizing flow notifications about incipient congestion. We have presented results to support our arguments that system-wide notification allows for better and faster convergence to fairness. More particularly, we monitored system convergence to equilibrium in the presence of EGCN and have shown experimentally that EGCN achieves continuously better allocation of bandwidth, while it also improves system efficiency. For the evaluation of system smoothness, we proposed two new metrics to capture system smoothness, namely the MaxCwndDeviation and the MaxAvgCwndDeviation. We implemented extensive experiments and studied system performance under different network conditions, such as different level of contention, network bandwidth and propagation delays. Simulation results confirmed the gains of the proposed scheme in terms of fairness, smoothness and efficiency.
References


