Network schemes for TCP elastic traffic in the Internet

Lluís Fàbrega, Teodor Jové

Institute of Informatics and Applications (IIiA)

Campus Montilivi, University of Girona, 17071 Girona, Spain

Tel: +34 972418889 E-mail: { lluis.fabrega, teodor.jove }@udg.edu

Received: May 20, 2013 Accepted: October 24, 2013 Published: October 31, 2013

DOI: 10.5296/npa.v5i3.3728 URL: http://dx.doi.org/10.5296/npa.v5i3.3728

Abstract

TCP elastic traffic is generated by the traditional "data" applications in the Internet, such as web browsing, peer-to-peer file sharing, ftp, e-mail and other. These applications are built on top of TCP, which provides reliable transfers and adjusts the sending rate to the network conditions to achieve the maximum possible throughput, a feature that makes TCP flows to be called "elastic". From the point of view of the network, TCP elastic traffic requires the maximum possible throughput above a minimum value, a network service that we call the Minimum Throughput Service (MTS). In this paper we survey the main network schemes that have been proposed in the Internet to provide this service for TCP elastic traffic, classified in two broad groups, the ones that do not use Admission Control (AC) and the ones that do use it. For each network scheme we describe the main characteristics of the service (whether the minimum throughput can be different or is the same for all flows, whether isolation among flows is provided, etc.) and their architecture (the specific traffic conditioning, queue disciplines and AC mechanisms used, the required state, the use of signaling, etc.).

Keywords: TCP; elastic traffic; quality of service; admission control; minimum throughput service.

1. Introduction

The traditional “data” applications in the Internet (web browsing, peer-to-peer file sharing, ftp, e-mail and others) transfer discrete messages or “documents” (a web request, a basic web file, an embedded image, an ftp file, an ftp command, etc.), which are partitioned into blocks and sent through the network into a sequence of packets or “flows” within TCP connections [1, 2, 3]. The users of these applications expect that there is no error in the transfer of documents and also that the response time is the smallest possible below a certain maximum value [4]. Consequently, document transfers require reliability and the maximum possible rate above a minimum value. Therefore TCP flows generated by these applications are satisfactorily supported by a network service that provides a minimum throughput to the flow and if possible, an extra throughput. We will call this network service the Minimum Throughput Service (MTS).

A network service is provided by a network scheme, which is composed of a combination of resource provisioning (link’s capacity, queues, etc.) and mechanisms of management, routing, Admission Control (AC), traffic conditioning and queue disciplines. Different network schemes have been proposed in the Internet to provide the MTS for TCP elastic traffic. For example, the traditional network scheme in the Internet is simply based only on First-In-First-Out (FIFO) and Tail Drop queues, there is neither traffic conditioning nor AC mechanisms, and provisioning can be whatever. The strength of this scheme is the simplicity. However it does not provide isolation (or protection) between flows, i.e., flows sending at a higher rate than the fair throughput can damage other well-behaved flows. Isolation could be provided using other queue disciplines and/or adding traffic conditioning mechanisms, but at the cost of complicating the network scheme. Moreover, in the traditional network scheme, when resources in the followed network path are enough to satisfy the minimum throughput requirements of all flows, all of them are satisfied, but otherwise, i.e., during congestion situations, none of them is satisfied (it is said that this scheme provides the best-effort service, a service with no absolute guarantees). Congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques. If the network resources are over-provisioned so that congestion never or rarely occurs, then this scheme always provides the desired minimum throughput to all flows (it provides a service with absolute guarantees to all flows). Over-provisioning the network resources is a common practice in backbone networks, since it allows simple network schemes to be used. However, over-provisioning can be difficult to achieve since unexpected events may happen (inaccurate traffic forecasts, routing changes, link or router failures, etc.), and it can be very inefficient in using resources. If more efficient provisioning is desired, another possible option is using network schemes that include an AC mechanism. By using AC, when resources in the followed path are enough to satisfy the minimum throughput requirements of all flows, all of them are satisfied, and otherwise, i.e., during congestion situations, some of them receive the desired minimum throughput (they are “accepted”) and the rest do not (they are “rejected” or “blocked”). Again, congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques, but if congestion still occurs, AC achieves efficient use of resources by maximizing the number of satisfied flows. However, using AC complicates the network scheme, and therefore a major concern is making the AC as simple as possible.

In this paper we survey the main network schemes that have been proposed in the Internet to provide the MTS for TCP elastic traffic, classified in two broad groups, the ones that do not use AC and the ones that do use it. For each network scheme we describe the main characteristics of the service (whether the minimum throughput can be different or is the same for all flows, whether isolation among flows is provided, etc.) and their architecture (the specific traffic conditioning, queue disciplines and AC mechanisms used, the required state, the use of signaling, etc.). The paper is organized as follows. In Section 2 we describe in detail the characteristics of TCP elastic traffic. Then in Section 3 we review the network schemes for TCP elastic traffic without AC, and in Section 4, the ones with AC. Finally, in Section 5, we present the conclusions.

2. TCP elastic traffic

In this section we deal with TCP elastic traffic. Firstly, we discuss the Quality of Service (QoS) requirements of “data” applications, starting from the application QoS (i.e., the description of the application performance) and then the network QoS (i.e., the description of the network performance). Then we give a general definition of the MTS, the network service for TCP elastic flows. After that we review the two important functions of TCP: reliability through packet retransmission and resource sharing through rate-adaptive algorithms. Finally, we describe the characteristics of TCP elastic traffic at different levels, as seen as a set of sessions, documents, packets and flows.

2.1 QoS for elastic traffic

In “data” applications, the application’s processes transfer discrete (time-independent) messages or “documents” (a web request, a basic web file, an embedded image, an ftp file, an ftp command, a typed character in telnet, an e-mail message, etc.). The QoS at the application layer is described in terms of fidelity to the original documents and in terms of interactivity or response time (see Fig. 1). Fidelity refers to the errors in the transferred documents, while the definition of the response time varies depending on the application. For example, on the web, the response time may be defined as the waiting time between requesting a page (user “click”) and visualizing it, which includes the transfer of several documents (the initial request, the basic web file, the rest of the requests, some embedded images, etc.); in ftp, the response time may be defined as the waiting time between commands and status messages, and especially between a file request command and the end of the file transfer; in telnet, the response time may be the time between when a character is typed at the client and the visualization of the corresponding echo sent by the server. In general, the response time is composed of the transfer times of documents and the processing time by the application’s processes (e.g., a web server).

Specifically, users of these applications expect no errors in the transfer of documents, i.e., absolute fidelity. Moreover, the smaller the response time the more satisfied the user, but when the response time is too long, impatient users or high layer protocols may interrupt the transfer [4]. These aborted transfers imply a waste of resources, which can get even worse if the transfer is tried again. This means that there is a maximum response time. Its value depends on users' desires and the specific application. For example (see [5] and references therein), a typical user browsing small web pages expects a maximum response time of a few seconds (e.g., 5 s); in ftp, where files are typically larger, the maximum response times are also larger, and users would be willing to wait in proportion to the file size; or in telnet, the echo delays should be smaller than 150 ms. Moreover, some demanding users can want better performance than others, e.g., users using the web for business applications (e-commerce, online trading, etc.) may require smaller maximum response times than users browsing the web for a "normal" use. In conclusion, the users of these applications expect that there is no error in the transfer of documents and also that the response time is the smallest possible below a certain maximum value.

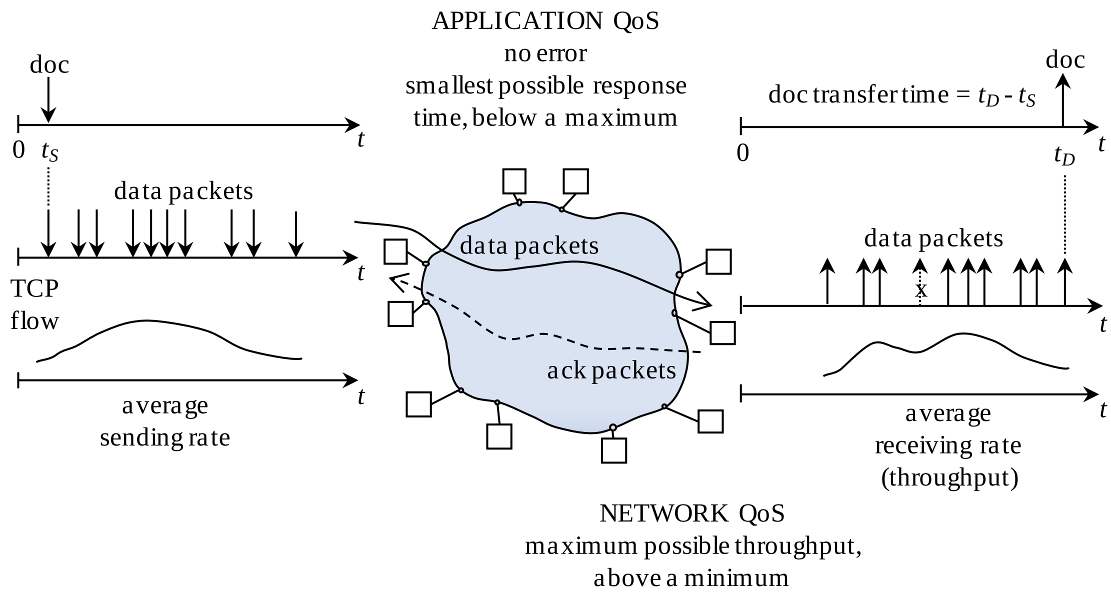
APPLICATION QoS
no error
smallest possible response time, below a maximum
doc
0
t_S
t
data packets
TCP flow
average sending rate
data packets
ack packets
doc
doc transfer time = t_D - t_S
0
t_D
data packets
x
average receiving rate (throughput)
NETWORK QoS
maximum possible throughput, above a minimum

Figure 1. Application QoS and network QoS in elastic applications.

For transferring an individual document, the above requirements imply that there should be no error and the smallest possible transfer time below a maximum value (see Fig. 1). Then:

- Absolute fidelity can be achieved through packet retransmission procedures, as in TCP. The TCP source divides the document into blocks (for small documents a single block may be enough) and sends a sequence of packets at a certain sending rate, which the network delivers to the destination occasionally with delays and some losses. From the acknowledgment packets sent back by the destination, the source detects and retransmits lost packets until the whole document is received correctly. Packet retransmission increases the packet delay and consequently the document transfer time, and moreover, it may cause duplication of packets, which are discarded by the destination. From the point of view of the network, the decisive QoS parameter is the average receiving rate (averaged in some time interval) or network throughput, which includes the duplicates. From the point of view of the application, the average receiving rate without including the duplicates (and without the TCP header overhead) is more important, known as goodput. The goodput, if averaged in a period equal to the transfer time, is simply the document size divided by the transfer time.

- From the point of view of the network, the requirement about the document transfer time turns into a requirement about the throughput, i.e., the document transfer should achieve the maximum possible throughput above a minimum value. Note that the traditional view is different since the minimum throughput requirement is not considered. Traditionally, the utility curve of these applications, which relates user's satisfaction to throughput, is considered to be strictly positive and concave [6]. This means that users always benefit by any increase in throughput (i.e., any reduction in the document transfer time) but also that users tolerate throughputs tending to zero (i.e., unlimited document transfer times). However, because users expect a maximum response time, a maximum document transfer time is required, and therefore, a minimum throughput is required. In conclusion, the requirement of the smallest possible document transfer time below a maximum value implies that the network should provide a minimum throughput and if possible, an extra throughput, and also that the source should be able to use it, as in TCP. TCP sources use rate-adaptive algorithms to achieve the maximum possible throughput while sharing network resources fairly between all TCP flows [7]. Since the maximum possible throughput changes over time, TCP increases and decreases the sending rate in order to match these variations and minimize packet loss. Due to this ability of adjusting the sending rate to different network conditions, "data" applications and TCP flows are called "elastic".

2.2 The Minimum Throughput Service (MTS)

Elastic flows require the maximum possible throughput above a minimum value from the network. Therefore, they are satisfactorily supported by a network service that provides a minimum throughput to the flow and if possible, an extra throughput, which we call the Minimum Throughput Service (MTS).

The input traffic profile of the service is defined by an average sending rate equal to the desired minimum throughput. Flows' packets can be considered to be in-profile or out-profile by comparing the actual average sending rate of the flow and this input traffic profile (see Fig. 2). Then:

- In-profile packets are delivered, i.e., they have no loss (there are no requirements on packet delay). This results in the minimum throughput.
- Out-profile packets can be delivered, i.e., they can have some loss. This depends on the remaining network resources, that is, the ones that are not used by the in-profile traffic of flows. These remaining network resources are shared between competing flows according to a defined sharing policy, e.g., using best-effort sharing, equal or

weighted sharing, giving priority to short flows over long flows, or other. This results in the extra throughput.

Finally, the delivery of the service from the provider to the user (and end-user or a neighboring domain) should be defined in a Service Level Agreement (SLA).

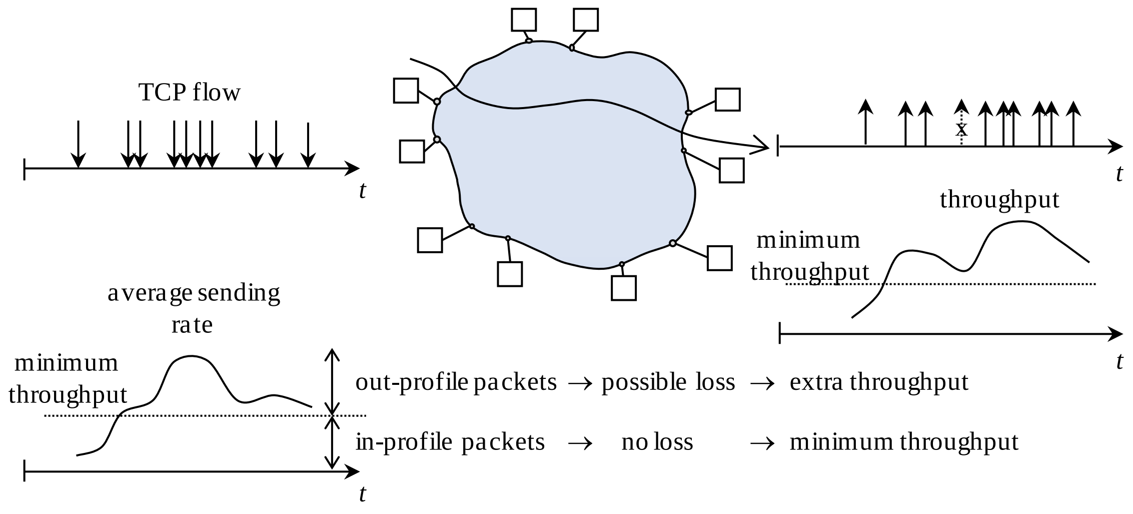
TCP flow
t
average sending rate
minimum throughput
t
throughput
minimum throughput
t
t
x
out-profile packets → possible loss → extra throughput
in-profile packets → no loss → minimum throughput

Figure 2. The definition of the Minimum Throughput Service.

2.3 Reliability and resource sharing in TCP

TCP is a transport protocol that provides a connection-oriented, reliable and ordered service to the application layer, besides performing multiplexing of traffic from different application's processes through the ports. The protocol is standardized in [1] but a large number of other RFCs deal with different aspects of TCP. In this subsection we review the two important functions of TCP, reliability and resource sharing.

2.3.1 Reliable delivery in TCP

The following is a summary of how TCP provides a reliable delivery [1, 2]:

- Application data is partitioned by the source in blocks of at most MSS (Maximum Segment Size) bytes and sent into TCP packets, which carry the (per-byte) sequence number of the first byte of the block.
- The destination only notifies correctly received packets, by sending back Acknowledgment (ACK) packets. ACKs are "cumulative", i.e., they indicate the correct reception of all bytes before the carried (per-byte) sequence number (which is the sequence number of the next expected byte). If an out-of-order packet arrives, a duplicated ACK is sent, and if an in-order packet arrives, the ACK sending may be delayed (but one ACK should be sent for at least every second packet arrival and no later than 500 ms after the first arrival [8]). Selective ACKs (SACK), which indicate non-contiguous blocks of consecutive bytes correctly received, have also been defined [9, 10].
- TCP uses pipelining together with the sliding window mechanism for flow control.

This combination allows the source to have multiple sent but yet-to-be-acknowledged packets, with a limit equal to the size of the window (new packets are allowed to be sent when ACKs corresponding to previously sent packets are received). The size of the window is the number of bytes the destination can accommodate in its receive buffer (it is called the receiver's advertised window), and it is notified at the source through ACK packets. Typically the resulting traffic is very bursty since the source sends a number of packets continuously (a burst) according to the window and then stops and waits for the ACKs before going on.

- There is a timeout-based mechanism at the source to detect the situations in which no ACK is received for a particular packet. When a packet is sent, a timer is initialized to a time called retransmit timeout (RTO).

- A packet is considered to be lost when the corresponding ACK is not received within the RTO (RTO expiration), or when three duplicated ACKs of a previous ACK are received (this second procedure is called "Fast Retransmit" [11]). A third possible loss indication comes from the SACK information, if it is used.

- When the loss of a packet is detected, a retransmission procedure is triggered. Depending on the actual state, only the lost packet is retransmitted ("selective repeat" style), or the lost packet and the next packets in the actual window ("go-back-N" style). Packet retransmissions can cause duplicated packets that are discarded by the destination.

It is worth commenting the following about the RTO expiration and the Fast Retransmit procedure, the two traditional indications of packet loss:

- RTO should be longer than the round-trip time (RTT) of packets to allow for ACK arrivals, but not too much so as not to delay retransmissions. RTT can be measured but, since RTT changes in time, it is difficult to estimate its actual "right" value (and consequently the "right" RTO). Since it is desired that the timer expires early only on rare occasions, RTO is obtained through a conservative calculation based on the average and deviation values of measured RTTs [12] (moreover, before the first RTT measurement has been made, RTO is set to a value of 3 s). On the other hand, the timer is initialized with the actual RTO when a packet is sent or retransmitted. Usually there is a single timer related to the oldest unacknowledged packet, and when this packet is acknowledged, the timer is reinitialized (with the actual RTO) for the next unacknowledged packet. If the timer expires, the actual RTO is doubled ("exponential back off") and the timer is reinitialized. Moreover, TCP implementations use coarse grain clocks to measure the RTT and trigger the RTO. This limits the precision of all these procedures and imposes a large minimum value on RTO. Moreover RFC 2988 [12] states, again in a conservative approach to avoid early retransmissions, that whenever RTO is computed, if it is less than 1 second then the RTO should be rounded up to 1 second. The conclusion is that RTO expiration may take a relatively long time.

- The Fast Retransmit procedure [11] is expected to detect a packet loss before RTO expiration, ideally after a time of about one RTT (sometimes a few RTT), so retransmissions are faster. However, since three duplicated ACKs are required, at least three later packets have to be sent and correctly received at the destination, and therefore, when the window is small, a late RTO expiration is more likely to occur.

2.3.2 Resource sharing in TCP

Besides the reliability function we have just described above, another important function of TCP is to achieve the maximum possible throughput while sharing network resources between TCP flows. This is the reason why TCP sources use rate-adaptive algorithms.

Resource sharing between TCP flows has the general goal of using the resources fully while maintaining a certain “fairness” in the allocations to flows. Fairness can be defined in different ways, such as max-min fairness, proportional fairness and other (see [13] for a discussion), leading to different allocations. For example, according to the classical fairness notion, the so-called max-min fairness, in a simple scenario of N flows sharing a single link of capacity C, the fair rate for each flow is equal to C/N. In the case of any network topology, this does not simply mean allocating the same share to each flow in a link-by-link basis, since this may not lead to full utilization. Then [14]:

- Max-min fairness is achieved when the rates allocated to flows are made as equal and large as possible, or more formally, when an increase in any allocated rate is at the cost of a decrease in some already smaller rate.
- Or alternatively, when each flow has a “bottleneck” link, i.e., a link that is fully utilized, and where the flow’s allocated rate is equal to or larger than the rates allocated to the rest of the flows using this link.

Another notion of fairness consists in minimizing the number of actual flows by giving priority to short flows over long flows. This has been shown to reduce the transfer time of short documents without hurting the performance for long flows, when considering heavy tailed document size distributions [15, 16].

Another point apart from the fairness type is that the fair rate of a flow changes during its lifetime. This is because the number of flows in the network changes in time, due to new arrivals and departures of finished transfers. Therefore, the average allocated rate of a flow (and the corresponding document transfer time) depends on two issues, the type of fairness and the variations in the number of flows [15].

The fair rate is not explicitly indicated to TCP sources. Instead sources use a probing method that reacts according to binary indications from the network, i.e., whether the sending rate is below the fair rate (“no congestion”) or the opposite (“congestion”). The classical congestion indication is packet loss. TCP sources use rate-adaptive algorithms (called congestion control algorithms) that increase the sending rate while there is no congestion, and decrease the sending rate when congestion occurs, oscillating around the fair rate (and adapting to changes in its value). The amplitude of the oscillations (which should be limited to avoid inefficiencies in link utilization) as well as the rate of convergence and adaptation to changes (packet loss should be minimized to reduce retransmissions) depend on the specific rate-adaptive algorithms. Moreover, the network does not enforce the fair rate on the TCP flow, and therefore the fairness in resource sharing is achieved relying on all TCP sources implementing the same algorithms. As a consequence, the type of fairness also depends on the specific algorithms used by all sources [13].

As mentioned above, the classical congestion indication from the network is packet loss, but others are possible. The following is a more complete summary of possible congestion indications:

- Packet loss detected from Fast Retransmit. This is considered a fast detection method. It does not work well when the window size is small.
- Packet loss detected from RTO expiration. It may take a relatively long time in comparison to Fast Retransmit. This is considered to be an indication of severe congestion, because it means that Fast Retransmit has not detected the packet loss before.
- Packet loss detected from SACK information.
- An increase in RTT. Before queues overflow (and packets are dropped), the RTT of packets increases, and this can be used by TCP sources to react in advance and reduce losses, with the consequent improvement in performance.
- Explicit Congestion Notification (ECN). With ECN [17], routers can provide an explicit binary indication of congestion to end-nodes before packet loss occurs. Two bits in the IP header are used: one for indicating the congestion and another for indicating the ECN capability. By using Active Queue Management mechanisms such as Random Early Detection (RED) [18], routers set the congestion indication bit in packets when the queue occupancy is high enough but before the queue overflows (and a packet has to be dropped). TCP uses ECN in the following way: when the destination TCP receives a packet with the congestion indication bit set, it echoes back this bit (through one dedicated flag of the TCP header) in its next ACK to the TCP source, which then reacts to congestion as if a single packet loss had occurred. With ECN, TCP performance improves because losses are reduced.

TCP sources vary the sending rate by controlling the window size, because the average sending rate (in RTT) is roughly equal to the window size divided by RTT (this comes from considering that TCP sends a burst of packets limited by the window size and then waits for ACKs before going on, which arrive after one RTT). This results in the so called congestion window (cwnd), which vary according to the TCP congestion control algorithms. The flow control's window is then the minimum value between the congestion window and the receiver's advertised window (i.e., it can vary from 1 (MSS) to the actual receiver's advertised window).

TCP congestion control algorithms have evolved over time, resulting in the so-called

"TCP versions" (see [2] for a general view). The first one, TCP Tahoe [7], defined the following mechanisms for increasing the congestion window (later standardized in [11]):

- Slow Start. The congestion window is set to a small value (less than or equal to 2 (MSS), typically 1), and then it is increased by 1 (MSS) after each new (i.e., non-duplicated) ACK is received ($cwnd = cwnd+1$). If the receiver acknowledges every packet, *cwnd* is doubled each RTT (a multiplicative increase by 2). When the congestion window reaches a value called "Slow Start threshold" (*ssthresh*), it continues increasing according to Congestion Avoidance.

- Congestion Avoidance. The congestion window is increased as $cwnd = cwnd+(\mathrm{MSS}\cdot\mathrm{MSS})/cwnd$, after each new ACK is received. If the receiver acknowledges every packet, *cwnd* is increased by approximately 1 (MSS) every time a full window is acknowledged, i.e., it is increased by 1 (MSS) each RTT (an additive increase by 1).

When a packet loss is detected, through Fast Retransmit or RTO expiration, *cwnd* is set to 1 (MSS), entering Slow Start, and *ssthresh* is set to *FlightSize*/2 (but no less than 2 MSS), where *FlightSize* is the amount of data that has been sent but not yet acknowledged. A "go-back-N" retransmission procedure is used. Therefore, TCP Tahoe starts from "one" and performs fast probing through Slow Start and slow probing through Congestion Avoidance. When a packet loss is detected, it starts again from "one", and *ssthresh* (which initially can be arbitrarily large, e.g., the receiver's advertised window) is adjusted dynamically so that the next slow probing is performed as the congestion window is near the value at which a loss previously occurred. Note also that if delayed ACKs are used, the congestion window is increased at a lower rate since less ACKs are sent.

The second version, TCP Reno [11], differs from the first one only in terms of its behavior after a Fast Retransmit, which is considered an indication of moderate congestion. The Fast Recovery algorithm was introduced:

- When a packet loss is detected through Fast Retransmit, *ssthresh* is set to *FlightSize*/2 (but no less than 2 MSS), and *cwnd* is set to *ssthresh*+3. The lost packet is retransmitted, and if allowed by the congestion window, new packets are sent (i.e., "selective repeat" style). For each additional duplicated ACK, *cwnd* is increased by 1 (MSS). When a new ACK is received, *cwnd* is set to the actual *ssthresh* (i.e., the previous *FlightSize*/2, a multiplicative decrease by 2), and it enters Congestion Avoidance.

However, it was shown that this procedure, by requiring every packet loss to be retransmitted strictly based on Fast Retransmit, may fail to recover from multiple losses in a single flight of packets, which leads to RTO expiration for the other lost packets. An improvement of Fast Recovery was introduced in a new version, TCP NewReno [19], which was extensively used. Basically, during Fast Recovery, "partial" ACKs (new ACKs not covering the highest sequence number sent) and "full" ACKs are distinguished: if a partial ACK is received, the next corresponding packet is considered to be lost and retransmitted, and if a "full" ACK is received, then Fast Recovery ends. Another way of dealing with the problem of multiple losses in a single flight of packets is using the SACK option, since its information can be used to selectively retransmit the lost packets. A modified TCP Reno with SACK was shown to outperform TCP NewReno in [20], especially when the number of losses is large. The SACK option is widely deployed and straightforward implementations have been proposed [21].

We have just seen above the classical increases and decreases of the TCP sending rate to achieve the fair rate (see Fig. 3): when losses do not occur and when in Congestion Avoidance, the congestion window is additively increased by one (*cwnd*+1) each RTT, and when losses occur and are detected through Fast Retransmit and recovered through Fast Recovery, the congestion window is multiplicatively decreased by two (*cwnd*/2). This is known as a particular case of the more general "Additive Increase and Multiplicative Decrease (AIMD)" control behavior.

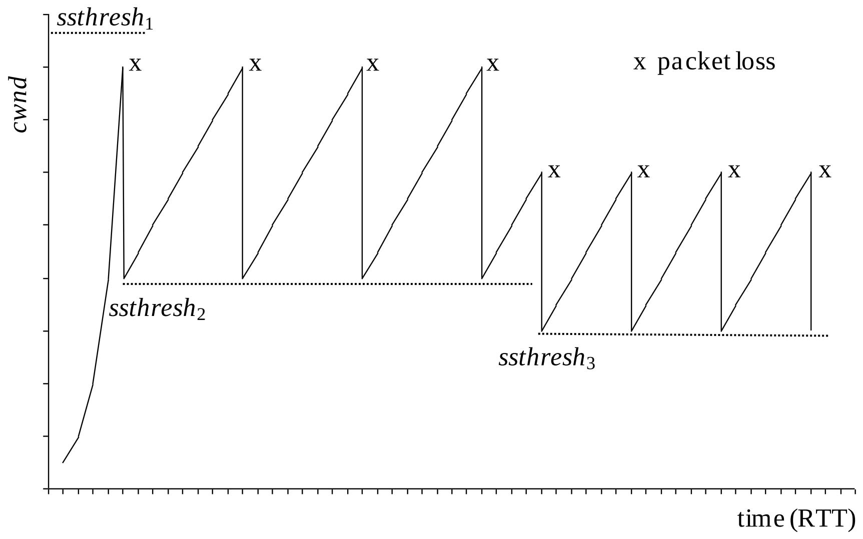

Figure 3. The ideal AIMD (Additive Increase Multiplicative Decrease) behavior of the TCP congestion window, after an initial Slow Start phase.

AIMD was studied in a single link in [22], which showed that it converges to fair resource sharing. Although it is often stated that AIMD rate variations provide max-min fairness in a general network, some authors (e.g., [13]) shown that they tend to provide rather another type of fairness called "proportional fairness" (which produces smaller allocations for flows passing through more hops to the advantage of greater overall throughput). Moreover, fairness in resource sharing between TCP flows depends strongly on the RTT and time duration of flows:

- Flows with large RTT achieve smaller throughput than flows with small RTT. This is because the value of the additive increase of the sending rate (the congestion window) is constant and independent of RTT, and it does not occur at fixed time intervals but in time periods of RTT due to the necessary feedback delay [23]. Therefore, the sending rate increases more quickly for flows with a smaller RTT, and achieves higher throughput. This happens with long flows, which are stable in Congestion Avoidance under AIMD control.

- Short flows tend to achieve smaller throughput than long flows. This is because short flows spend most of their lifetime in Slow Start, while long flows spend most of their lifetime in Congestion Avoidance, and flows in the Slow Start phase achieve smaller throughput than flows in the Congestion Avoidance phase. Firstly, flows during Slow Start double the sending rate each RTT (until a loss is detected), but meanwhile they achieve a lower throughput since it is necessary to start conservative from a small value. Secondly, during Slow Start the congestion window may be small, and if losses occur it is probable that they will be detected through RTO expiration and not through Fast Retransmit (as it has been observed in measurements, e.g., in [24]). The expiration may take a long time (RTO is usually large, since there are just a few samples of RTT at the beginning, a conservatively value is used), and moreover, the congestion window is severely decreased due to starting again from a small value in Slow Start. Thirdly, flows in Congestion Avoidance have larger congestion window and are less sensitive to losses (usually detected through Fast Retransmit), and are stable under AIMD control around the fair rate.

Finally it is worth commenting that the evolution of the TCP congestion control algorithms does not end with the classical algorithms we have seen above, and that there have been many other modifications and proposals for "new" TCPs in order to improve its performance (adding ECN to IP [17], using Active Queue Management such as RED [18], TCP Vegas [25], Fast TCP [26], TCP Westwood [27], XCP [28], TCP pacing schemes [29], TCP ACK pacing schemes [30], HighSpeed TCP [31], scalable TCP [32], BIC-TCP [33], Compound TCP [34], TCP in wireless networks [35], etc.; a performance comparison can be found in [36], and numerous references in [37]).

2.4 Characteristics of TCP elastic traffic

Traditional "data" applications in the Internet generate the majority of Internet traffic. Their traffic can be described at different levels by considering different entities as a set of sessions, documents, packets and flows. The notion of session generally refers to a time period of "continuous" and "related" user activity, so that user sessions can be considered statistically independent. A session has a starting time and duration, and is composed of a succession of documents. Documents are generated within a session, and are characterized by its sending time and its size. Each document results in a sequence of packets, each packet is characterized by its sending time and length. A flow is a sequence of "related" packets that are "close" in time, which can correspond to the transfer of a single document, several documents or an entire session. It is characterized by its starting time, duration, traffic parameters such as the average rate, peak rate, etc., document size, and others. The sequence of packets corresponding to a document transfer (which includes the retransmitted packets) is typically very bursty (the TCP source sends a number of packets continuously – a burst – according to the actual window and then stops and waits for the ACKs before going on) and has a variable average rate (due to the TCP rate-adaptive algorithms). Moreover, besides data packets, control packets for connection management and error correction are also sent.

The structure of sessions, in terms of documents, their interarrival times and sizes, and the TCP connections used, depends on the application. The following is a qualitative description of some applications:

- During a web session in a server, the user downloads a set of web pages, each composed of several parts called “objects”, usually a “basic” file and several embedded images (referenced in the basic file). A user “click” results in the basic file being requested, and once it is received, the client requests the rest of the objects of the web page. A TCP connection can be of two types, a “non-persistent” connection, when it is closed by the server after finishing the transfer of an object, or a “persistent” connection, when it remains open and is closed by the client or the server usually when there is inactivity during a given timeout interval. The set of documents may be transferred within several non-persistent TCP connections (one connection per document, opened sequentially or in parallel), or within a single or several persistent TCP connections (each one with sequential pairs of request-reply, or with “pipelined” requests, that is, several requests one after the other without waiting for each reply). The size of web requests usually fits in a single TCP packet, while the size of replies is extremely variable, since they can range from small basic files to very large files. The web also creates document interarrival times that are very variable. For example, very short interarrival times occur when clients open parallel TCP connections to transfer several embedded images of a web page; short interarrival times come from users browsing and reading different web pages; users taking a long break results in long interarrival times.

- During an ftp session, the commands sent by the client and the corresponding status messages from the server are transferred within a single TCP connection (for “control”). A separate TCP connection (for “data”) is established each time the user wants to transfer some data, for example, listing a directory or getting a file (two operations that usually occur close in time). Control commands are small, while the size of the files is extremely variable.

- During a telnet session, there is a single TCP connection, in which each character being typed by the user at the client is sent to the server, which echoes them back, as well as sending the responses to the commands. The size of the typed characters is obviously very small and their interarrival times are limited by the typing speed of the user.

Measuring traffic at the session or flow level may be difficult because, as we have just seen, the relation between these traffic entities and TCP connections is not obvious. In some cases (e.g., in ftp or telnet), a session can be simply equated to an entire single TCP connection, initiated by the connection request packets and ended by the corresponding release packets. On other occasions (e.g., in web), a session may include several and related TCP connections (persistent or non-persistent) and considered to be finished when there is no user activity during a given timeout period. Similarly, a flow can correspond to a sequence of packets within a single TCP connection, within several connections or an entire TCP connection. Usually a flow is identified by a 5-tuple in IPv4 (protocol, source and destination IP addresses and ports), initiated when the first packet arrives and finished when there are no more packets during a given timeout period. Another option is to equate a flow to an entire TCP connection.

Files sizes in the web have shown to exhibit a distribution with a heavy tail [38]. This means that there is a high variability in sizes, and that most web files are small but a few of them are very large (and consequently, the same is valid for the lifetime of flows, when each one corresponds to a single file: most of the flows are short and a few of them are very long). A reasonable fit to the form of the heavy tail is provided by the Pareto distribution.

3. Network schemes for TCP elastic traffic without admission control

In this section we review the main network schemes that have been proposed in the Internet to provide a network service for TCP elastic traffic, when the mechanisms used are basically traffic conditioning and/or queue disciplines, and AC is not considered. In consequence, when resources in the followed network path are enough to satisfy the minimum throughput requirements of all flows, all of them are satisfied; otherwise, i.e., during congestion situations, none of them is satisfied. We say that the network service has a relative guarantee, since the throughput received by a flow is defined as a function of the throughput received by other flows. For example, in a fair throughput service, the goal is to provide a throughput equal to the fair rate of the bottleneck link, i.e., the link’s capacity divided by the number of present flows (in fact, the max-min fairness, see Subsection 2.3); or in the weighted version, the proportional throughput service, flows’ throughputs and flows’ weights are proportional, and therefore different throughputs can be provided. Congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques. If the network resources are over-provisioned so that congestion never or rarely occurs, then these schemes always provide the desired minimum throughput to all flows (they provide a service with absolute guarantees to all flows).

A possible scheme would be using Fair Queuing (FQ) or Weighted Fair Queuing (WFQ) scheduling at flow level in all routers. With FQ each flow would receive the max-min fair rate, while with WFQ they would receive the weighted max-min fair rate (and therefore throughput differentiation according to flows’ weights). Isolation between flows would be provided by queues without needing specific traffic conditioning mechanisms. However, this scheme would be too complex because it would require per-flow state and per-flow management in all routers. For each arriving packet, the router would need to classify the packet into a flow, update some per-flow variables and perform per-flow operations. Per-flow state should be established and updated explicitly through per-flow signaling (this would result in a high overhead given that most elastic flows are short – see Subsection 2.4), or implicitly through flows’ data packets and timeout procedures.

In the next subsections we review the following set of schemes for TCP elastic traffic without AC (see Fig. 4): the "traditional" scheme in the Internet with FIFO and Tail Drop queues, a set of schemes based on packet classes, the scheme based on Core-stateless Fair Queuing and finally the User-Share Differentiation scheme. For each of them, we describe the main characteristics of the service, that is, whether they provide the same or different throughputs, and whether they provide isolation between flows (so that flows sending more traffic than their allocated throughput do not damage well-behaved flows that do send according to their allocated throughput), and also the architecture of the scheme, that is, the specific mechanisms used, the required state and the use of signaling.

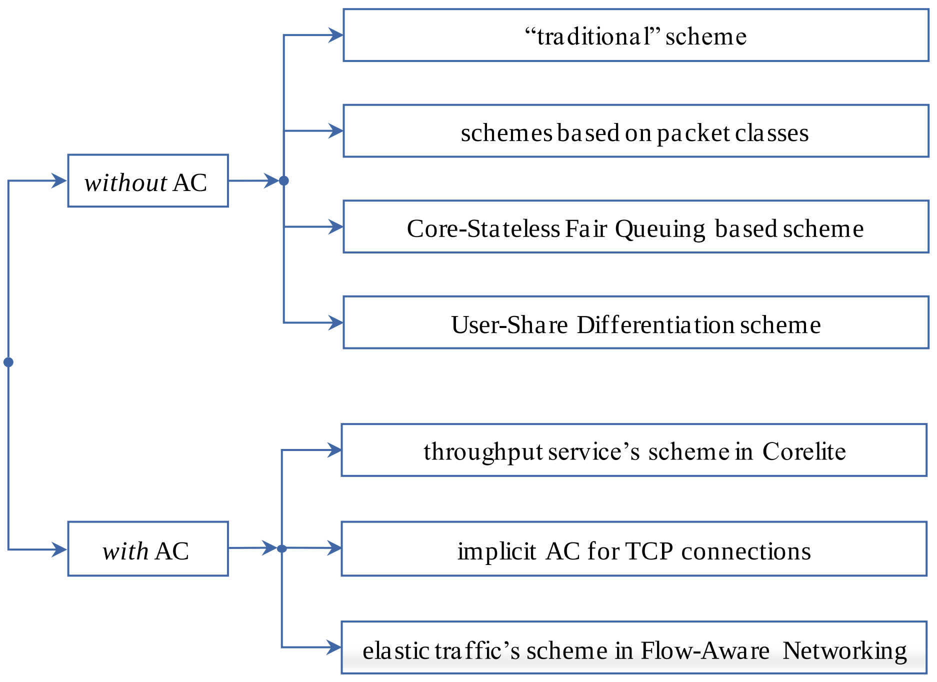
"traditional" scheme
schemes based on packet classes
without AC
Core-Stateless Fair Queuing based scheme
User-Share Differentiation scheme
throughput service's scheme in Corelite
with AC
implicit AC for TCP connections
elastic traffic's scheme in Flow-Aware Networking

Figure 4. Main network schemes proposed in the Internet to provide the MTS for TCP elastic traffic.

3.1 The "traditional" scheme

The "traditional" scheme in the Internet is only based on FIFO and Tail Drop queues. Traffic conditioning mechanisms are not used. All packets receive the same treatment and the service provided is best-effort. The main advantage of the scheme is the simplicity. However, it does not provide any isolation between flows, and in the case of traffic overload, flows injecting more traffic "steal" resources from the rest (the output average rate is proportional to the input average rate).

The combination of this scheme and TCP rate-adaptive algorithms (see Subsection 2.3) aims to provide a fair throughput service. The goal is to provide a throughput equal to the fair rate of the bottleneck link, i.e., the link's capacity divided by the number of present flows, although the effective resource sharing may exhibit unfairness in some situations (flows with large RTT versus flows with small RTT, or short flows versus long flows). An obvious consequence is that it is not possible to provide different throughputs to different TCP flows. Moreover, the fair throughput service is achieved by TCP sources through a probing method, increasing the sending rate while there is no congestion (e.g., no packet loss) and decreasing it when congestion occurs, oscillating around the fair rate. Therefore, this approach relies on cooperation between sources that implement the same algorithms. An advantage is that no support from the network is needed, since the fair rate is not indicated to the TCP sources nor enforced. However, some sources may not react against congestion (e.g., real-time sources that do not decrease the sending rate) or react in a different way, so that well-behaved TCP sources may receive smaller throughput (i.e., they are not protected).

An enhancement of the "traditional" scheme is achieved by replacing Tail Drop by an Active Queue Management such as RED [18, 39]. One of the main goals of RED is to avoid the so-called "TCP global synchronization problem", which arises from the interaction between TCP rate-adaptive algorithms and Tail Drop, in the following way: when a sequence of packets arrives and the queue occupancy is high, multiple packets may be discarded; flows experiencing this packet loss will decrease the sending rate at a similar time, and after a while, when losses do not occur, they will increase the sending rate at a similar time, and so on, becoming "synchronized". Moreover, it is likely that the number of losses in a single flight of packets of a flow is large, resulting in RTO expiration (a severe congestion indication), the flow entering Slow Start, and a strong reduction in the sending rate. The synchronized behavior together with the burstiness of TCP traffic leads to poor link utilization and low aggregated throughput. RED works in the following way:

- It measures the queue's average occupancy, *avg*, by using a low-pass filter or exponentially weighted moving average of the instantaneous queue occupancy.
- It discards packets before the queue is full according to a dropping probability P_{drop} that depends on the average occupancy *avg* and two thresholds, *min* and *max* (see Fig. 5): when $avg < min$, no packet is dropped; when $max < avg < min$, the packet dropping probability increases linearly with *avg*, from probability 0 to *Pmax*; when $avg > max$, all packets are dropped. Therefore, the dropping probability of the arriving packet is higher as *avg* increases.

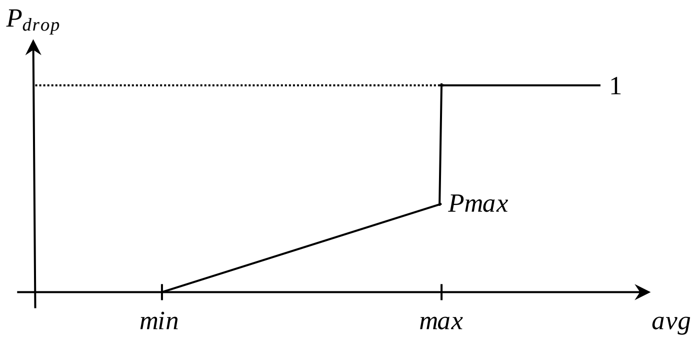

Figure 5. Dropping probability as a function of the queue's average occupancy in RED.

Since RED uses an average occupancy, short bursts of packets (sort-term congestion) are filtered, and thus ignored, without inducing packet loss. However, when bursts are longer (long-term congestion), the average occupancy increases and packets start to be discarded to indicate the congestion to sources. Note that RED detects incipient and light congestion and provides an early indication: before several packets are discarded (indication of severe congestion), it is likely that a single packet in a flight of packets of a flow is discarded, resulting in Fast Retransmit, the flow entering Congestion Avoidance, and a weak reduction of the sending rate. The probabilistic discarding avoids global synchronization, since only some of the sources will experience packet loss and decrease the sending rate. When congestion is stronger, the indication to sources is much more frequent (P_{drop} increases with *avg*). Moreover, the probabilistic packet discarding will tend to affect flows causing the congestion more (the ones that receive higher throughput), since most of the arriving packets will belong to them. Finally, note that when RED is used together with ECN [17], packets are marked instead of being discarded, on the assumption that sources will react in the same way as if a packet were lost. Numerous references about RED can be found in [37].

3.2 Schemes based on packet classes

These schemes are based on packet classes in a similar way to the Differentiated Services (Diffserv) architecture [40]. Diffserv networks are based on packet classes, i.e., flows' packets are assigned to a small number of classes at the ingress (a mark that identifies the class is written in the packet's header), and queue disciplines in the core apply a different treatment to packets belonging to different classes. The mechanisms used in these schemes are the following:

- Traffic conditioning mechanisms at the network ingress assign each flow's packet to a class and write a mark in the packet's header that identifies the class (the number of classes is small), according to an agreed traffic profile. Alternatively, the packets may arrive at the network ingress already marked (e.g., previously by sources), and then traffic conditioning at the network ingress checks and enforces the agreed traffic profile (out-profile packets can be remarked or even discarded).

- Queue disciplines in the network core are based on classes, i.e., they apply a different treatment to packets belonging to different classes.

Per-flow state is only kept at the edge while the core remains simple and highly scalable. The use of traffic conditioning and class-based queues can allow isolation between flows and different throughput to different flows to be provided, as well as the possibility to coexist with other different network services.

3.2.1 The *in* and *out* scheme of the Assured Service

The Assured Service, defined within the so-called "allocated-capacity" framework in [41], is able to provide different throughputs to flows from different users during congestion. Moreover, it protects TCP flows against non-responsive sources. The proposed scheme uses two packet classes, called in and out, with different discarding priorities (see Fig. 6):

- There is an input traffic profile (for each user) that defines the flow's desired minimum throughput r_{min}. The average sending rate of the flow r is measured and compared with r_{min} in order to classify each packet as an in-profile or out-profile. The goal of this classification is to obtain a sequence of in-profile packets with a rate equal to the minimum throughput, specifically, min(r, r_{min}), and a sequence of out-profile packets with a rate equal to the exceeding traffic above it, i.e., min(0, r- r_{min}). Packets are marked accordingly as *in* or *out*.

- There is a single FIFO queue (to maintain packet ordering) with priority discarding so that if a packet has to be discarded, the *out* class has a higher discarding priority than the *in* class (this behavior was generalized and standardized by the IETF Diffserv Working Group in the definition of the AF PHB [42]).

As a consequence, *in* packets have a higher assurance of delivery than *out* packets. The desired minimum throughput is provided when the aggregated in traffic does not cause an overload in any of the links of the network path. When an overload occurs, the throughput provided to each flow is a share of the bottleneck link's capacity that is proportional to (and smaller than) the desired one. Therefore, the difference between the provided throughputs during congestion comes from the different desired throughputs of the input traffic profile of flows (users).

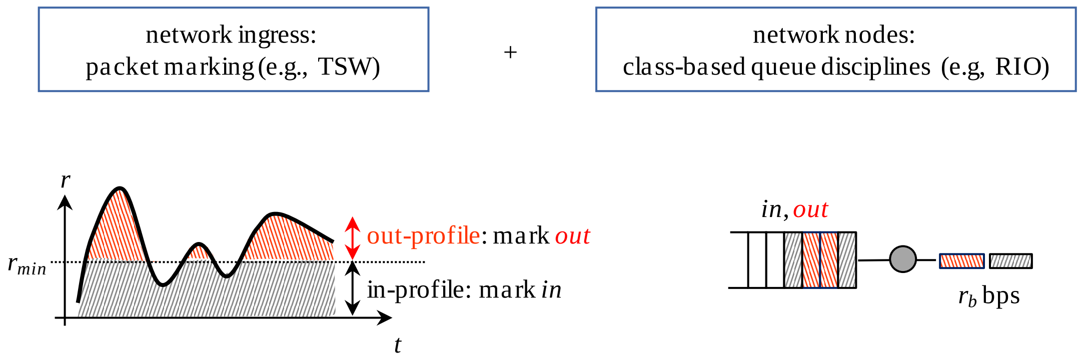

Figure 6. The *in* and *out* scheme of the Assured Service.

The following algorithms were proposed in [41] to implement this scheme (similar algorithms were also proposed in [43]):

- The flow's average rate is estimated using the TSW (Time-Sliding Window) algorithm, and the marker is based on a probabilistic function.
- The priority discarding in queues uses the RIO (RED with In and Out bits) algorithm.

TSW provides a smooth estimate of the TCP sending rate in a way suitable to the burstiness of TCP traffic. The average sending rate *avg_rate* is estimated upon each packet arrival and over the last period of time (or window), which considers a "past history" equal to the so-called *win_length* parameter. The algorithm is simple since the only state variables are the arrival time of the previous packet and the previous value of *avg_rate*. A difficulty is that the recommended value for *win_length* is the flow's RTT, which is usually not known at the network ingress. Therefore, a fixed value has to be used and the average rate is not optimally estimated. A proposed solution is to implement this algorithm and the marking in the TCP source itself, which has an estimate of the actual RTT (in this case, the network would then check and enforce the agreed traffic profile).

The marker is based on a probabilistic function. Once *avg_rate* for the arriving packet is calculated, the marker decides whether the packet is *in* or *out* in the following way: if *avg_rate* is smaller than the desired throughput R_T, the packet is *in*; otherwise, the packet is *out* with probability $P_o = (avg\_rate - R_T)/avg\_rate$ or *in* with probability $1-P_o$ (a variant is using $1.33 \cdot R_T$ instead of R_T as a threshold). The probabilistic function is used to space *out* packets and to reduce the probability of consecutive drops in a single flight of packets, which could lead TCP to enter Slow Start and severely reduce the sending rate. The design of TCP markers has been a subject of research and there have been more proposals (e.g., [43, 44]).

The RIO algorithm extends RED to work with two classes. Two sets of parameters are used and two separate average buffer occupancy calculations are tracked, one only for *in* packets and another one for all (*in* plus *out*) packets (see Fig. 7):

- The dropping probability of *in* packets depends only on the buffer occupancy of *in* packets avg_{in}, with parameters min_{in}, max_{in}, $Pmax_{in}$.
- The dropping probability of *out* packets depends on the buffer occupancy of *in* plus *out* packets avg_{tot}, with parameters min_{tot}, max_{tot}, $Pmax_{out}$.

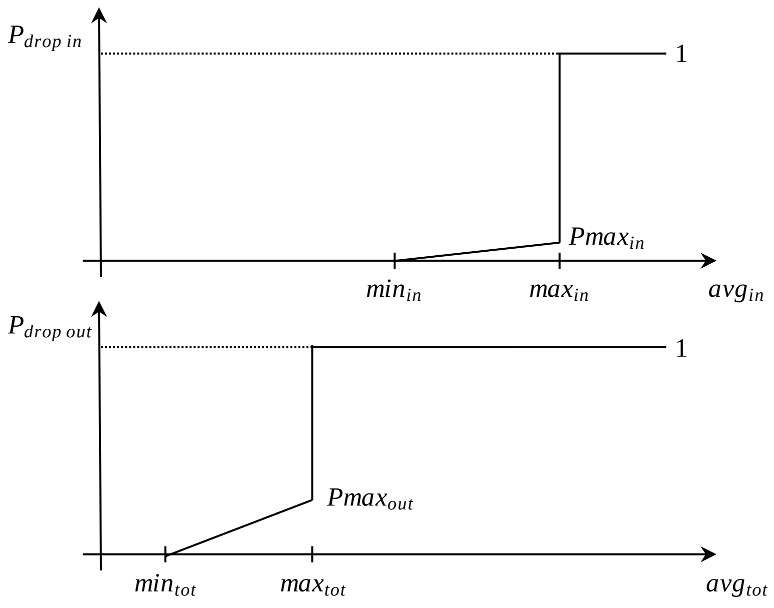

Figure 7. Dropping probability as a function of the queue’s average occupancy in RIO.

RIO’s objective is to discriminate *out* packets from *in* packets: when there is incipient congestion, RIO first drops some *out* packets; if the congestion persists, RIO drops all the *out* packets; finally, *in* packets are only dropped when the router is flooded with *in* packets. Therefore, RIO parameters have to be chosen carefully (e.g., 40, 70, 0.02 for *in* and 10, 30, 0.2 for *out*, are one of the choices in [41]).

3.2.2 TCP-state based differentiation

This scheme [5] uses three packet classes, called *high*, *med* and *low*, with different discarding priorities (following the AF PHB definition [42]):

- Packets are marked in the host by TCP sources as *high*, *med* or *low*. The marking algorithm depends on the TCP state, i.e., on the actual value of the window and on the identification of some "important" packets, as we describe below in more detail.

- An agreed traffic profile specifies the total aggregated rate of *high and med* packets per user, in the form of two token bucket profiles. At the network ingress, traffic conditioning mechanisms enforce this traffic profile, and out-profile packets can be remarked to a lower priority or even discarded. Note that traffic conditioning is made over the user's flow aggregate, and therefore it is in the best interest of sources to mark packets in conformance with the agreed traffic profiles.

- There is a single FIFO queue (to maintain packet ordering) with priority discarding so that if a packet has to be discarded, the *low* class has the highest discarding priority, the *med* class the medium discarding priority, and the *high* class the lowest discarding priority. The algorithm in queues is an extension of RED for three classes (like RIO is an extension of RED for two classes).

The marking algorithm at the TCP source considers two cases:

- In the first case, the marking is based on the actual value of the window (since the average sending rate – in RTT – is roughly equal to the window size divided by RTT). The algorithm considers that if a connection is performing well and the window is high, there is no need to protect its packets and it is better to use the *high* marks to improve the performance of other connections that need it; if then the connection suffers packet drops and its window is reduced, marking its packets as *high* can help it to recover. Following these ideas, the window-based marking compares the actual congestion window *cwnd* with two thresholds, $high_{thresh}$ and med_{thresh}, in the following way: if $cwnd \leq high_{thresh}$, packets are marked as *high*, if $high_{thresh} < cwnd \leq med_{thresh}$, packets are marked as *med*, and if $cwnd > med_{thresh}$, packets are marked as *low*.

- In the second case, the algorithm identifies some "special" packets, the ones that are more important for the stability of the TCP congestion control algorithms, and marks them as *high*. Specifically, these packets are the connection establishment packets (important for the initial RTT measurement and RTO calculation), the data packets sent when the window is small (since TCP is more vulnerable to losses), and the data packets retransmitted after an RTO expiration or Fast Retransmit (since their loss could lead to RTO expiration). Note that some of these packets could also be marked as *high* by the window-based marking.

This scheme compensates for the unfairness experienced by short TCP flows. As we have seen in Subsection 2.3, short flows tend to achieve smaller throughput than long flows because their initial window is usually small and because they are more vulnerable to losses. This scheme identifies these situations and prioritizes packets to reduce losses; therefore, it tends to provide a fair throughput service, i.e., to share network resources equally between flows. Moreover, note that traffic conditioning mechanisms at the network ingress provides isolation between flows from different users.

Throughput differentiation is achieved by using different marking thresholds for different flows. The differentiation can be at the application level (e.g., higher thresholds for web than for ftp), at the user level (e.g., demanding users have higher thresholds than “normal” users), or even for different transferred documents (e.g., higher thresholds for transferring basic web files than for the rest of the web page objects).

3.2.3 Preferential treatment to short TCP flows

These schemes [45, 46] give preferential treatment to short flows over long flows by using different packet classes. The aim is two-fold: to compensate for the unfairness experienced by short flows, which tends to get less than their fair share when they compete for the bottleneck link’s capacity; and giving priority to short flows, which has been shown to reduce the transfer time of short documents without hurting the performance for long flows, when considering heavy tailed document size distributions (see Subsection 2.3).

Neither scheme provides isolation between flows. Two packet classes are used, e.g., called *short* and *long*:

- At the network ingress, the first packets of each flow are marked as *short* and the rest as *long*, according to a defined threshold.
- *Short* packets are preferentially treated over *long* packets in queues.

Note that the proposed *short* and *long* marking does not result in a classification between short and long flows, since the first packets of long flows are also marked as *short*. However, this is a desired feature, since in fact the unfairness is not between short and long flows but rather between the first packets of flows and the rest of the packets (i.e., the first packets of long flows experience the same problems). Therefore, the preferential treatment to the *short* class helps all flows.

The preferential treatment in queues in [45] is based on RIO, i.e., *short* packets are discarded less than *long* packets, so that they experience fewer losses. The preferential treatment in [46] is based on priority queuing, i.e., *short* packets are served before *long* packets, so that they experience fewer losses and smaller delays (also note that packet ordering in a flow is still maintained).

3.3 Core-Stateless Fair Queuing (CSFQ) based scheme

This scheme [47] provides a fair throughput service as well as isolation between flows. Moreover, by assigning a weight to the flow, it can be extended to provide different throughputs to different flows, proportionally to the flows’ weights.

The scheme uses the CSFQ algorithm in queues, which closely emulates the behavior of the FQ algorithm, but without needing a per-flow state. Instead, per-flow state is carried by packets using the Dynamic Packet State (DPS) technique: the state variables are encoded in the packet’s header and then are used and modified by the queue disciplines. In this scheme the state is the flow’s rate:

- The incoming rate of each flow is estimated at the network ingress and a label is written on its packets carrying the value of this rate.

- The queue discipline uses FIFO together with the CSFQ algorithm, which probabilistically discards packets so that each flow receives the fair rate in the link. CSFQ only uses the packet's label and measurements over aggregates.

The ingress router, upon each packet arrival, classifies the packet into a flow, updates the estimation of the flow's rate r, and labels the packet with r. This estimation is based on an exponential weighted moving average of the instantaneous rate (with a weight that depends on the packet inter-arrival time).

CSFQ in all routers works as follows:

- Each router periodically estimates the fair rate f in the link.
- Upon receiving a packet labeled with incoming rate r, the router drops the packet with probability $P_{drop} = \max[0, (r\text{-}f)/r]$. Therefore, if $r \leq$ f, all packets of the flow are forwarded and the flow's output rate is kept to r; if $r > f$, some packets of the flow are probabilistically discarded (hopefully, $(r\text{-}f)/r$ is the fraction of discarded packets), so that the output rate is approximately decreased to f. In any case, when a packet is forwarded, the router updates the packet's label with the flow's output rate (the minimum between f and the incoming r), which is the new flow's arrival rate for the next router.

The fair rate f in the link is estimated at certain times. The router continuously measures the aggregated incoming rate A and the aggregated forwarded rate F (both with the same procedure used for the flow's incoming rate at the ingress) in the link of capacity C. If there is no congestion ($A < C$), f is chosen as the maximum flow's rate between the flows that traverse the link, i.e., the maximum packet label observed in that time (and therefore the discarding probability is 0 for all packets of all flows). If there is congestion ($A \geq C$), a heuristic and iterative algorithm varies f (and therefore P_{drop} and F) by a factor C/F until it converges, i.e., until F matches C.

This scheme provides a fair throughput service and isolation between flows without needing a per-flow state in the core. However, it requires the state to be processed and updated for each packet in each router, and the state in the packet's header to be encoded.

3.4 User-Share Differentiation (USD) scheme

This scheme [48, 49] is able to provide different throughputs to flows from different users proportionally to some agreed users' weights, but in an aggregated way. Moreover, it provides isolation between flows from different users. The basic points of the USD scheme are the following:

- Each user has a weight (defined in a user-provider agreement), which controls resource sharing between users for both its sending and its receiving traffic.
- The queue discipline uses the WFQ algorithm or similar, which shares the link's capacity fairly between the traffic from different users according to the weights.

The user is chosen as the basic unit that defines traffic control granularity, so that all traffic that has originated from a user or destined to a user is aggregated within the network (within the traffic of a single user it is up to the user to decide how the service is shared internally). The state needed inside the network is reduced since it is not flow-based but rather user based. In each router there is a table with the user identifier and its associated weight (the user identifier can be the IP address of an end-user, the network prefix for a network, or a set of network prefixes for a group of networks). The per-user state makes the scheme highly scalable in the hierarchical structure of recursive user-provider relationships of the Internet.

The user identifier and its corresponding weight could be distributed to routers inside the network through a network management protocol. For each arriving packet, the router looks up the weight of the sending user and the weight of the receiving user in the table, since both weights control the sharing. This conflict is solved by making the WFQ scheduler use the minimum of the two weights.

Isolation between traffic of active users is provided by WFQ without needing specific traffic conditioning mechanisms at the network ingress. If one user transmits more than its actual allocated throughput in a given link, it will cause its own packets to be dropped in the queues.

4. Network schemes for TCP elastic traffic with admission control

In this section we review the main network schemes that have been proposed in the Internet to provide a network service for TCP elastic traffic when the mechanisms used are basically traffic conditioning, queue disciplines and AC. Therefore, when resources in the followed network path are enough to satisfy the minimum throughput requirements of all flows, all of them are satisfied; otherwise, i.e., during congestion situations, some of them receive the minimum throughput (they are "accepted") and the rest do not receive it (they are "rejected" or "blocked"). Congestion situations can be reduced by increasing network resources or by optimizing their use through better routing techniques. The blocking rate depends on the behavior of users' demands, the chosen resource provisioning, the routing techniques used, and the capability of the AC mechanism to maximize the number of satisfied flows. If nevertheless, congestion occurs, using AC achieves an efficient use of network resources by maximizing the number of satisfied flows, although it complicates the network scheme.

A possible scheme would be using FQ or WFQ scheduling at flow level in all routers and a classical parameter-based hop-by-hop AC. With FQ each flow would receive the same minimum throughput and an extra throughput equal to the max-min fair share of the remaining resources. With WFQ different flows would receive different minimum throughputs according to the assigned weights, and an extra throughput equal to the weighted max-min fair share of remaining resources. Isolation between flows would be provided by queues without needing specific traffic conditioning mechanisms. Per-flow signaling would carry the flow's minimum throughput request from router to router through the path, and each router would perform a local AC decision to limit the number of flows in each link so that accepted flows would receive their desired minimum throughput. However, this scheme would be too complex. It would require per-flow state and per-flow management in all routers. Given that most elastic flows are short (Subsection 2.4), using per-flow signaling would imply a high overhead and a rather long duration of the AC phase.

In the next subsections we review the following set of schemes for TCP elastic traffic with AC (see Fig. 4): the scheme for a guaranteed throughput service in the Corelite architecture, the implicit AC for TCP connections and the scheme for elastic traffic in the Flow-Aware Networking architecture. For each of them, we describe the main characteristics of the service, that is, whether the minimum throughput can be different or is the same for all flows, the expected extra throughput that results from sharing the remaining resources, and whether isolation between flows is provided (so that flows sending more traffic than their allocated throughput do not damage well-behaved flows that do send according to their allocated throughput), and also the architecture of the scheme, that is, the specific mechanisms used, the required state and the use of signaling.

4.1 The scheme for a throughput service in Corelite

The Corelite architecture provides several throughput and delay services using the same set of basic mechanisms. The scheme for a throughput service [50, 51] is able to provide different minimum throughputs r_{min} to different flows, and an extra throughput according to a weight w. It has two modes, which differ in the kind of guarantees: it is deterministic in the "guaranteed" mode (there is no loss if the sending rate is not higher than r_{min}), and it is qualitative in the "predictive" mode (low loss if the sending rate is not higher than r_{min}). It provides isolation between flows through traffic conditioning at the network ingress. Per-flow signaling is used to indicate the start of the flow, the requested r_{min} and the AC response. Per-flow state in the core is not required. We explain the AC scheme below, but firstly we describe the mechanisms used when a flow (in either mode) has already been accepted by AC:

- The ingress router performs traffic conditioning over the flow depending on the comparison between the actual flow's average rate r and two thresholds, r_{min} and r_{max}, where r_{min} is the minimum throughput and r_{max} is equal to r_{min} plus an extra throughput that is adapted according to the feedback received from core routers. A token bucket algorithm is used for measuring r. Flow's packets are classified into three types (resulting in three "subflows"): in-profile packets, with a rate equal to min(r, r_{min}); out-profile packets, with a rate equal to min(0, r-r_{min}, r_{max}-r_{min}); and the exceeding packets, with a rate equal to min(0, r-r_{max}). The exceeding packets are discarded; the other packets are forwarded and some of them may also be turned into special packets called "markers", as we explain in the next point.

- The ingress router periodically turns some flow's packets into markers: one marker is inserted for every N number of "data" packets (or bytes) and each marker carries N. Therefore, the transmission rate of markers taking into account the carried N reflects the rate of the flow. Markers carry the source address of the ingress router, a unique identification of the flow within the ingress router, and the number of "data" packets (or bytes) that they represent. Markers are logically distinct packets, but are physically piggybacked to a "data" packet. The use of markers differs if the mode is "predictive" or "guaranteed", as we explain in the next point.

- For "predictive" flows, *p*-markers for in-profile packets and *w*-markers for out-profile are used. One *p*-marker is introduced for every $N_p = K_1 \cdot r_{min}$ "data" packets (or bytes) of in-profile traffic, where K_1 is a constant. Each *p*-marker carries N_p. One *w*-marker is introduced for every $N_w = K_2 \cdot w$ "data" packets (or bytes) of out-profile traffic, where K_2 is a constant and *w* is the weight. Each w-marker carries N_w. If the actual flow's rate is smaller than r_{min}, the rate of *p*-markers reflects this rate (and no *w*-markers are introduced); if the actual flow's rate is greater than r_{min}, the rate of *p*-markers reflects r_{min} and the rate of *w*-markers reflects the extra rate of the flow (above r_{min}), normalized according to the weight *w*.

- For "guaranteed" flows, *g*-markers for in-profile packets and *w*-markers for out-profile are used, in a similar way as for "predictive". The only difference is that if the actual flow's rate is smaller than r_{min}, the rate of *g*-markers does not reflect this rate but rather the minimum r_{min}.

- Routers use FIFO queues. They also extract the markers from packets and maintain a queue of *p*-markers, *g*-markers and *w*-markers. When congestion is detected, a random number of *w*-markers are selected and sent back to the ingress router that generated it (if there were no *w*-markers, firstly *p*-markers and then *g*-markers would be selected, but this is not likely to happen as AC is used). Also note that the *w*-markers of flows with a greater weight *w* are less likely to be selected.

- Periodically, the ingress router checks the markers received from core routers during the last time period corresponding to a flow. The flow's threshold r_{max} is reduced in proportion to the received markers, and if no marker has been received, it is increased additively by a constant.

The AC scheme is hop-by-hop since each router performs a local AC decision. Per-flow signaling carries the AC requests and responses, but it does not require a per-flow state in the core. The duration of the AC phase is about one RTT. The scheme is the following:

- A request signaling packet with the rate requirement r_{min} is sent along the path.

- Each router maintains the available bandwidth B_{av}, which is calculated using the received markers and updated at a certain time period.

- A router in the path receives the request packet. If the request can be accepted ($r_{min} < B_{av}$), the router reduces B_{av} by r_{min} and forwards the request packet to the next router; otherwise a reject response signaling packet is sent back to the ingress.

At the beginning of a given time period, each router knows the available bandwidth B_{av} for this period. A request is accepted if r_{min} is available. If so, B_{av} is reduced by r_{min} and the resulting value of B_{av} is used for the next AC decision until the end of this time period. During this time period the router estimates the value of B_{av} to be used for the next time period. It is calculated from the number of *p*-markers and *g*-markers received over that time (the router counts the number of packets – or bytes – that the marker represents, which is carried by the marker). Note that in B_{av}, due to the different ways that *p*-markers and *g*-markers are generated, the aggregated rate of "predictive" flows is based on "real" measurements that aim to take into account the multiplexing gain, while the aggregated rate of "guaranteed" flows is based on "virtual" measurements that aim to equal the sum of the reserved rates of flows (i.e., based on their declared traffic parameters). Therefore, the "guaranteed" mode provides a deterministic guarantee, probably at the cost of reducing resource utilization, while the "predictive" mode can be more efficient in using resources but it provides a qualitative guarantee.

This scheme is able to provide different minimum throughput to different flows and isolation. It does not require per-flow state in the core or per-flow queuing. However, it requires per-flow signaling, which could result in a high overhead, a rather long duration of the AC phase, and quite complex management for the markers. Finally, there are no details about how TCP flows are defined and identified in the scheme.

4.2 Implicit AC for TCP connections

These schemes [52, 53] provide the same minimum throughput to all flows, which here are defined as TCP connections. The guarantee is qualitative. It does not provide isolation to accepted flows. The start of the flow (connection) and the AC response are implicitly indicated without signaling. The AC is fast. Per-flow (connection) state is not required in any router. Therefore, their main advantage is the simplicity. The mechanisms used are the following:

- The queue discipline is FIFO and there is no traffic conditioning mechanisms at the ingress, as in the "traditional" scheme (see Subsection 3.1). Therefore, isolation between flows is not provided and the fairness in resource sharing between the accepted flows depends on the TCP rate-adaptive algorithms.
- The AC scheme is based on measurements and without signaling. The AC algorithm measures the actual use of resources through a particular parameter, which is compared to a threshold to make the AC decision.

The authors only consider the AC in a single link, although it could be extended to a hop-by-hop scheme (or obviously to a one-hop scheme on logical paths with reservation). The start of the flow (connection) is implicitly indicated to the router through its first packet, i.e., the TCP connection establishment packets (SYN or SYN/ACK). The AC response is also implicitly indicated to the flow: in the case of acceptance, the connection establishment is allowed to proceed by forwarding the detected establishment packet; in the case of rejection, the connection establishment is aborted, by sending an RST packet to the sender [52] or by discarding the detected establishment packet [53]. Therefore, the AC is simple and moreover, it is fast as it is made as soon as a new flow arrives (in a hop-by-hop scheme, no signaling packet carrying the AC response of acceptance or rejection in the whole path is sent back to the ingress, and an accepted flow does not have to wait to start to transmit). Moreover, the scheme assumes that aborting a connection implies that the TCP source will not transmit any packet, and that the accepted TCP connections will share resources fairly as usual. Therefore, the scheme does not have to control whether the packets entering the network belong to an accepted flow or to control the traffic sent by accepted flows to provide isolation, and per-flow (connection) state is not required in any router. Therefore, the scheme relies on TCP sources being well-behaved, as in the "traditional" scheme. Another disadvantage of this scheme is that it is not possible to detect sequences of packets that occur as bursts within persistent TCP connections.

The AC algorithm in [52] measures the actual occupancy of the link and compares it with a given threshold, and when it is exceeded, new arriving connections are rejected. Specifically, a hysteresis with two thresholds is built to avoid excessive oscillations: connections are rejected when the occupancy exceeds the higher threshold and until the occupancy decreases below a lower threshold. The authors suggest occupancy thresholds of around 90% of the link's capacity. The relationship between the occupancy threshold and the connections' throughput comes from an analytical model, which considers ideal fair resource sharing of a random number of flows. For example, the model predicts an average flow's throughput equal to 20% of the link's capacity when the occupancy is around 90%. In the case of [53], the AC algorithm measures the incoming traffic to the link's queue and derives the actual overflow (loss) probability using a statistical model. A new arriving connection is rejected whenever this packet loss probability exceeds a given threshold. In this way a minimum throughput is provided since TCP's throughput is related to packet loss. However, note that both AC algorithms use parameters loosely related to the flow's throughput, and therefore tuning the performance is not easy. Moreover, both AC algorithms do not immediately consider the effect of a recently accepted flow until future measurements take it into account. This takes some time and therefore a high rate of new arriving flows to a router may cause false acceptances (however, this has another consequence if a hop-by-hop scheme were used: the partial acceptance of a flow in a hop, which later is rejected in the following hops, would not prevent other flows from being accepted in this hop; therefore, it would not lead to false rejections).

4.3 The scheme for elastic traffic in Flow-Aware Networking

The Flow-Aware Networking (or Cross-Protect) architecture [54, 55, 56] provides two services, a low jitter and low loss service for real-time flows and a minimum throughput service for elastic flows. The minimum throughput's value is the same for all elastic flows while the peak rate of real-time flows should be smaller than a given value. The guarantees are qualitative. It provides isolation to accepted flows. The user-network interface remains as simple as in the traditional Internet, since implicit ways are used instead of per-flow signaling. The AC is fast. It requires per-flow state and per-flow queuing in all routers. The two basic mechanisms are the following:

- The queue discipline uses the Priority Fair Queuing (PFQ) algorithm [55], which shares the link's capacity fairly between all flows and also gives priority to flows whose peak rate is less than the current link's fair rate. It requires per-flow state in each router.

- The AC scheme is hop-by-hop, based on measurements and does not use signaling. It requires per-flow state in each router. It does not differentiate between elastic and real-time flows or different traffic rates. It ensures that the current priority traffic load is smaller than a given percentage of the link's capacity, and that the fair rate is higher than a given threshold. This threshold is chosen to be higher than the peak rate of expected real-time flows, so that they receive scheduling priority in PFQ queues.

PFQ is an enhancement of the FQ algorithms. Like FQ it shares the link's capacity fairly between all flows, so that each flow receives the max-min fair share and is isolated from other flows. In addition to this, PFQ gives scheduling priority to packets from flows whose peak rate is smaller than the current fair rate, so that these flows experience low jitter and low loss. In this way the requested flow's QoS (real-time or elastic) can be implicitly indicated (without signaling): a flow whose peak rate is smaller than the fair rate is considered a real-time flow; otherwise it is considered to be an elastic flow.

AC and PFQ help each other. AC maintains the fair rate above a threshold, which is chosen to be higher than the expected peak rates of real-time flows. PFQ maintains a list of active flows, which is smaller than the list of accepted flows, and scalability is assured by the fact the number of flows is bounded by AC. PFQ provides two measurements that are used by the AC algorithm: *fair_rate*, an estimation of the rate currently realized by backlogged flows, and *prio_load*, the current load of the traffic receiving scheduling priority.

The AC scheme is hop-by-hop since each router performs a local AC decision (it could also be used as a one-hop scheme on logical paths with reservation). It does not use any signaling and therefore it requires per-flow state in each router to detect the new flows. Each router maintains a list of accepted flows in an implicit way. A new flow is indicated to a router by the arrival of its first packet, the router indicates a local acceptance decision of the flow by forwarding this packet or a local rejection decision by discarding it, and the end of the flow is detected when no packet is received within a defined timeout interval. This way has the advantage of not requiring signaling, and in the case of elastic traffic, sequences of packets that occur as bursts within persistent TCP connections can be detected. Per-flow state consists in a flow identifier and the arrival time of the last packet of each flow. A flow is identified by the usual 5-tuple in IPv4 (protocol, source and destination IP addresses and ports) or by the more flexible 3-tuple in IPv6 (flow label, source and destination IP addresses). Specifically, the procedure is the following. For each arriving packet, the list is checked. If the packet belongs to a flow in the list, it is forwarded and the last packet arrival time of the flow in the list is updated. If the packet does not belong to any flow in the list, an AC decision for the new flow is made. If the flow is accepted, the packet is forwarded and a new entry is added to the list. If the flow is rejected, the packet is discarded. A flow is erased from the list when the time since the last packet arrival exceeds the defined timeout.

The AC scheme does not use any explicit indication of the requested service, neither the

QoS (real-time or elastic) nor the traffic parameters (the minimum throughput for elastic or the peak rate for real-time). The AC algorithm does not distinguish between elastic and real-time flows. The traffic parameter of the new arriving flow is implicitly supposed to be a given value, which is defined by the network (the maximum between the following values: the minimum throughput of elastic flows and the possible peak rates of real-time flows). This implicit approach has two important advantages: signaling carrying the flow's traffic parameters is not required, and the blocking probabilities of all flows are equal, independently from their requested traffic rate (see the "trunk reservation" mechanism in [57]).

The AC algorithm is based on measurements using the above mentioned *fair_rate* and *prio_load*. The general goal of the AC algorithm is to ensure that the current priority traffic load (*prio_load*) is smaller than a given percentage of the link's capacity, and that the fair rate (*fair_rate*) is higher than the mentioned threshold (i.e, a value higher than the expected peak rates of real-time flows). The detailed algorithm is not specified (e.g., the percentage of link's capacity for *prio_load*, or whether the measurements of *fair_rate* and *prio_load* are artificially updated once a flow is accepted in order to establish a reservation immediately), although a recommended threshold for the fair rate is about 1% of the link's capacity.

Note that the AC is fast, as it is made as soon as a new flow arrives, since no signaling packet carrying the AC response (of acceptance or rejection) in the whole path is sent back to the ingress, and an accepted flow does not have to wait to start to transmit. Also note that, as it happens in any hop-by-hop AC scheme, a partial reservation in a hop for a flow (established immediately when it is accepted), which is later rejected in other hops, may prevent other flows from being accepted in this hop (for some time), leading to false rejections. However, in this scheme, since no AC response signaling packet is sent back to the ingress, this situation can last for more time and be worse if a rejected (but partially accepted) flow persists in transmitting (although this is not likely to happen).

5. Conclusions

The users of "data" applications such as web browsing, peer-to-peer file sharing, ftp, e-mail and other, expect that there is no error in the transfer of documents and also that the response time is the smallest possible below a certain maximum value. Therefore TCP elastic flows generated by these applications are satisfactorily supported by a network service that provides a minimum throughput to the flow and if possible, an extra throughput, the Minimum Throughput Service (MTS). With this in mind, we have reviewed the main network schemes that have been proposed in the Internet for TCP elastic traffic, with and without AC, focusing on the main characteristics of the service and their architecture.

We have studied the following network schemes without AC:

- The "traditional" scheme provides the best-effort service, which in combination with TCP rate-adaptive algorithms provides a fair throughput service. Different throughputs and isolation between flows are not provided. It is based only on FIFO and Tail Drop (or RED) queues.

- The schemes based on packet classes can provide different throughputs and isolation between flows. Traffic conditioning mechanisms at the ingress assign each flow's packet to a class (e.g., by comparing the flow's average sending rate and the flow's desired minimum throughput, flow's packets are assigned to an *in or out* class) and queue disciplines are based on classes (e.g., the *out* class has a higher discarding priority than the *in* class). Per-flow state is only kept at the edge while the core remains simple and highly scalable.

- The scheme based on CSFQ provides a fair throughput service (a weighted version is also possible) and isolation between flows. The ingress router estimates the flow's incoming rate and writes it on a label in the packet's header. The CSFQ algorithm in queues discards packets probabilistically, using only the packet's label and aggregated measurements, so that each flow receives the fair rate. Per-flow state is only required at the edge. However, it requires processing and updating the label for each packet in each router, as well as encoding the label in the packet's header.

- The USD scheme provides different throughputs to flows from different users but in an aggregated way. It provides isolation between flows from different users. Each user is assigned a given weight and WFQ in queues share resources between users according to this weight for both the sending and the receiving traffic. It requires per-user state in all routers (it does not require per-flow state).

We have studied the following network schemes with AC:

- The scheme for a throughput service in Corelite provides different minimum throughputs to different flows as well as isolation. The ingress router turns some flows' packets into the so-called (*g* or p) markers, so that their rate indicates the minimum throughput, and other flows' packets into *w*-markers, so that their rate indicates the assigned extra throughput. In each router, ordinary and marker packets are scheduled together with FIFO. When congestion is detected *w*-markers are sent back to the ingress router, which then decreases the extra throughput assigned to the flow. The AC scheme is hop-by-hop, per-flow signaling carries the AC request and response, and each router determines the aggregated reservation by measuring the arriving *g* and *p* markers during a given time period. It neither requires per-flow state in the core nor per-flow queuing. However, it requires per-flow signaling, which could result in a high overhead and a rather long duration of the AC phase as well as quite complex management for the markers. Finally there are no details about how TCP flows are defined and identified in the scheme.

- The scheme with an implicit AC for TCP connections provides the same minimum throughput to all flows, which are defined here as TCP connections. It does not provide isolation. The data path is simply based on FIFO queues, and the AC scheme is hop-by-hop (or a one hop scheme on logical paths with reservation). It is based on measurements and it does not have signaling. The start of the flow (connection) is indicated to a router through the TCP connection establishment packets: in the case of acceptance, the connection establishment is allowed to proceed by forwarding these packets, and otherwise, it is aborted. Therefore, the AC is fast, as it is made as soon as a new flow arrives. For the local AC decision, the router measures the actual use of resources through a particular parameter, and when it exceeds a given threshold, new connections are rejected. Per-flow (connection) state is not required in the core or at the edge. However, the scheme relies on TCP sources being well behaved. It does not detect sequences of packets that occur as bursts within persistent TCP connections. Tuning the performance is not easy since the parameters measured are loosely related to the flow's throughput. The AC algorithms do not immediately consider the effect of a recently accepted flow until future measurements take it into account. This takes some time, and therefore, a high rate of new arriving flows to a router may cause false acceptances.

- The scheme for elastic traffic in Flow-Aware Networking provides the same minimum throughput to elastic flows, which are defined here as sequences of packets within TCP connections. A service for real-time flows is also provided. The scheme provides isolation. Queues use the PFQ algorithm, which shares the link's capacity fairly between flows, provides isolation and gives priority to flows whose peak rate is less than the current link's fair rate (i.e., for real-time flows). PFQ requires per-flow state. The AC scheme is hop-by-hop (or a one hop scheme on logical paths with reservation), based on measurements and without signaling. The AC requires a per-flow state. A new flow is indicated by the arrival of its first packet. The router indicates a local acceptance decision of the flow by forwarding this packet or a local rejection decision by discarding it. The end of the flow is detected when no packet is received within a defined timeout interval. Therefore, the AC is fast, as it is made as soon as a new flow arrives. The AC algorithm does not differentiate between elastic and real-time flows and the traffic rate of the new arriving flow is supposed to be the maximum possible value. It ensures that the current priority traffic load is smaller than a given percentage of the link's capacity, and that the fair rate is higher than a given threshold (which is chosen to be higher than the peak rate of the expected real-time flows).

From among the different network schemes without AC we have studied, the ones based on packet classes show a good trade-off between the simplicity (per-flow operations are kept at the edge only) and the service characteristics (they allow different throughputs and isolation between flows to be provided). Out of the network schemes with AC we have studied, we found that is of special interest the definition of flow used in the scheme for elastic traffic in Flow-Aware Networking, as it captures the sequences of packets that occur as bursts within persistent TCP connections, as well as the implicit way of detecting the start and end of these flows. Another interesting aspect of some of these schemes with AC is the utilization of implicit ways for indicating the requested service parameters (QoS and traffic), although they achieve this by providing the same minimum throughput to all flows. In all of them the AC is hop-by-hop and based on measurements. However, they require either per-flow signaling in the core, or are not able to provide different throughputs or isolation between flows, or require per-flow state and per-flow queuing in the core.

References

[1] Postel J., “Transmission Control Protocol”, RFC 793, 1981.
[2] Tobagi F., Noureddine W. , “The Transmission Control Protocol. An introduction to TCP and a research survey”, Technical Report, Stanford University, 2002.
[3] Comer D.E., “Internetworking with TCP/IP. Volume I”, published by Prentice Hall, ISBN 0-13-227836-7, 1995.
[4] Massoulié L., Roberts J.W., “Arguments in favor of admission control for TCP flows”, Proceedings of the 16th International Teletraffic Congress - ITC16, Edinburgh, UK, June 1999.
[5] Noureddine W., Tobagi F., “Improving the performance of interactive TCP applications using service differentiation”, Elsevier Computer Networks, vol. 40, no. 1, 2002.
[6] Shenker S., “Fundamental design issues for the future Internet”, IEEE Journal on Selected Areas in Communications, vol. 13, no. 7, 1995. http://dx.doi.org/10.1109/49.414637
[7] Jacobson V., “Congestion avoidance and control”, ACM SIGCOMM Computer Communication Review, vol. 18, no. 4, 1988. http://dx.doi.org/10.1145/52325.52356
[8] Braden R., “Requirements for Internet hosts - communication layers”, RFC 1122, 1989.
[9] Mathis M., Mahdavi J., Floyd S., Romanow A., “TCP selective acknowledgment options”, RFC 2018, 1996.
[10] Floyd S., Mahdavi J., Mathis M., Podolsky M., “An extension to the selective acknowledgement (SACK) option for TCP”, RFC 2883, 2000.
[11] Allman M., Paxson V., Stevens W., “TCP Congestion Control”, RFC 2581, 1999.
[12] Paxson V., Allman M., “Computing TCP's retransmission timer”, RFC 2988, 2000.
[13] Massoulié L., Roberts J.W., “Bandwidth sharing: objectives and algorithms”, IEEE/ACM Transactions on Networking, vol. 10, no. 3, 2002. http://dx.doi.org/10.1109/TNET.2002.1012364
[14] Bertsekas D., Gallager R., “Data networks”, published by Prentice Hall, ISBN 0-132-00916-1, 1987.
[15] Massoulié L., Roberts J.W., “Bandwidth sharing and admission control for elastic traffic”, Telecommunication Systems, vol. 15, no. 1-2, 2000.
[16] Bansal N., Harchol-Balter M., “Analysis of SRPT scheduling: investigating unfairness”, ACM SIGMETRICS Performance Evaluation Review, vol. 29, no. 1, 2001.
[17] Ramakrishnan K., Floyd S., Black D., “The addition of Explicit Congestion Notification (ECN) to IP”, RFC 3168, 2001.
[18] Floyd S., Jacobson V., “Random early detection gateways for congestion avoidance”, IEEE/ACM Transactions on Networking, vol. 1, no. 4, 1993. http://dx.doi.org/10.1109/90.251892
[19] Floyd S., Henderson T., Gurtov A., “The NewReno modification to TCP's Fast Recovery algorithm”, RFC 3782, 2004.
[20] Fall K., Floyd S., “Simulation-based comparisons of Tahoe, Reno, and SACK TCP”, ACM SIGCOMM Computer Communication Review, vol. 26, no. 3, 1996.

http://dx.doi.org/10.1145/235160.235162

[21]Blanton E., Allman M., Fall K., Wang L., "A conservative selective acknowledgment (SACK) – based loss recovery algorithm for TCP", RFC 3517, 2003.

[22]Chiu D.M., Jain R., "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks", Computer Networks and ISDN Systems, vol. 17, no. 1, 1989. http://dx.doi.org/10.1016/0169-7552(89)90019-6

[23]Floyd S., Jacobson V., "On traffic phase effects in packet-switched gateways", Internetworking: Research and Experience, vol. 3, no. 3, 1992.

[24]Balakrishnan H., Padmanabhan V.N., Seshan S., Stemm M., Katz R.H., "TCP behavior of a busy Internet server: analysis and improvements", Proceedings of the IEEE Infocom 1998. San Francisco, CA, USA.

[25]Brakmo L., Peterson L., "TCP Vegas: end to end congestion avoidance on a global Internet", IEEE Journal on Selected Areas in Communication, vol. 13, no. 8, 1995.

[26]Jin C., Wei D.X., Low S.H., Buhrmaster G., Bunn J., Choe D.H., Cottrell R.L.A., Doyle J.C., Feng W., Martin O., Newman H., Paganini F., Ravot S., Singh S., "Fast TCP: from theory to experiments", IEEE Network, vol. 19, no. 1, 2005.

[27]Casetti C., Gerla M., Mascolo S., Sanadidi M. Y., Wang R., "TCP Westwood: end-to-end congestion control for wired/wireless network", Wireless Networks, vol. 8, no. 5, 2002. http://dx.doi.org/10.1023/A:1016590112381

[28]Katabi D., Handley M., Rohrs C., "Congestion control for high bandwidth-delay product networks", ACM SIGCOMM Computer Communication Review, vol. 32 no. 4, 2002. http://dx.doi.org/10.1145/964725.633035

[29]Aggarwal A., Savage S., Anderson T., "Understanding the performance of TCP pacing", Proceedings of the IEEE Infocom 2000, Tel-Aviv, Israel.

[30]Aweya J., Ouellette M., Montuno D.Y., "A self-regulating TCP acknowledgment (ACK) pacing scheme", International Journal of Network Management, vol. 12, no. 3, 2002. http://dx.doi.org/10.1002/nem.426

[31]Floyd S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, 2003.

[32]Kelly T., "Scalable TCP: improving performance in high speed wide area networks", ACM SIGCOMM Computer Communication Review, vol. 32, no. 2, 2003.

[33]Xu L., Harfoush K., Rhee I., "Binary increase congestion control for fast long-distance networks", Proceedings of the IEEE Infocom 2004. Hong Kong, China, March 7-11, 2004.

[34]Tan K., Song J., Zhang Q., Sridharan M., "A Compound TCP approach for high-speed and long distance networks", Proceedings of the IEEE Infocom 2006. April 23 - 29, 2006, Barcelona, Spain.

[35]Tian Y., Xu K., Ansari N., "TCP in wireless environments: problems and solutions", IEEE Communications Magazine, vol. 43, no. 3, 2005.

[36]Lai Y., Yao C., "TCP congestion control algorithms and a performance comparison", Proceedings of the 10th IEEE International Conference on Computer Communications and Networks, 2001. 15-17 Oct. 2001

[37]Floyd S., homepage at ICIR (the Center for Internet Research at the International Computer Science Institute), http://www.icir.org/floyd/.

[38]Crovella M.E., Bestavros A., "Self-similarity in World Wide Web traffic: evidence and

possible causes", IEEE/ACM Transactions on Networking, vol. 5, no. 6, 1997. http://dx.doi.org/10.1109/90.650143

[39] Braden B., Clark D., Crowcroft J., Davie B., Deering S., Estrin D., Floyd S., Jacobson V., Minshall G., Partridge C., Peterson L., Ramakrishnan K., Shenker S., Wroclawski J., Zhang L., "Recommendations on queue management and congestion avoidance in the Internet", RFC 2309, 1998.

[40] Blake S., Black D., Carlson M., Davies E., Wang Z., Weiss W., "An architecture for Differentiated Services", RFC 2475, 1998.

[41] Clark D.D., Fang W., "Explicit allocation of best-effort packet delivery service", IEEE/ACM Transactions on Networking, vol.6, no. 4, 1998. http://dx.doi.org/10.1109/90.720870

[42] Heinanen J., Baker F., Weiss W., Wroclawsky J., "Assured Forwarding PHB group", RFC 2597, 1999.

[43] Feng W., Kandur D.D., "Adaptive packet marking for maintaining end-to-end throughput in a differentiated-services Internet", IEEE/ACM Transactions on Networking, vol. 7, no. 5, 1999.

[44] Kumar K.R.R., Ananda A.L., Jacob L., "TCP-friendly traffic conditioning in DiffServ networks: a memory-based approach", Elsevier Computer Networks, vol. 38, no. 6, 2002.

[45] Guo L., Matta I., "The war between mice and elephants", Proceedings of the IEEE International Conference on Network Protocols, 2001. 11-14 November 2001, Riverside, CA, USA.

[46] Avrachenkov K., Ayesta U., Brown P., Nyberg E., "Differentiation between short and long TCP flows: predictability of the response time", Proceedings of the IEEE Infocom, 2004. Hong Kong, China, March 7-11, 2004.

[47] Stoica I., Shenker S., Zhang H., "Core-stateless fair queuing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks", IEEE/ACM Transactions on Networking, vol. 11, no. 1, 2003. http://dx.doi.org/10.1109/ICDCS.2000.840929

[48] Wang Z., Basu A., "Resource allocation for elastic traffic: architecture and mechanisms", Proceedings of the IEEE/IFIP Network Operations and Management Symposium, 2000. Honolulu, HI, USA, April 10-14, 2000.

[49] Wang Z., "User-Share Differentiation (USD) Scalable bandwidth allocation for differentiated services", Internet draft draft-wang-diff-serv-usd-00.txt, 1998.

[50] Sivakumar R., Kim T., Venkitaraman N., Bharghavan V., "Achieving per-flow weighted rate fairness in a core stateless network", Proceedings of the IEEE Conference on Distributed Computing Systems, 2000. 10-13 April 2000, Taipei, Taiwan.

[51] Sivakumar R., Venkitaraman N., Kim T., Lu S., Nandagopal T., Bharghavan V., "The Corelite QoS architecture: providing a flexible service model with a stateless core", Research report, Illinois Mobile Environments Laboratory (TIMELY) research group at the University of Illinois at Urbana Champaign, 1999.

[52] Kumar A., Hegde M., Anand S.V.R., Bindu B.N., Thirumurthy D., Kherani A.A., "Nonintrusive TCP connection admission control for bandwidth management of an Internet access link", IEEE Communications Magazine, vol. 38, no. 5, 2000. http://dx.doi.org/10.1109/35.841841

[53]Mortier R., Pratt I., Clark C., Crosby S., "Implicit admission control", IEEE Journal on Selected Areas in Communications, vol. 18, no. 12, 2000.
http://dx.doi.org/10.1109/49.898743

[54]Roberts J.W., "Internet traffic, QoS, and pricing", Proceedings of the IEEE, vol. 92, no. 9, 2004. http://dx.doi.org/10.1109/JPROC.2004.832959

[55]Kortebi A., Oueslati S., Roberts J.W., "Cross-protect: implicit service differentiation and admission control", Proceedings of the IEEE Workshop on High Performance Switching and Routing, 2004. Arizona, USA.

[56]Oueslati S., Roberts J.W., "A new direction for quality of service: Flow-aware networking", Proceedings of the Euro-NGI Conference on Next Generation Internet Networks, 2005. Rome, Italy, 18-20 April, 2005.

[57]Roberts J.W., Mocci U., Virtamo J. (Eds.), "Broadband network teletraffic. Performance evaluation and design of broadband multiservice networks. Final report of Action COST 242", Lecture Notes in Computer Science (LNCS), vol. 1155, Springer-Verlag, ISBN 3-540-61815-5, 1996.