A Generic Traffic Conditioning Model for Differentiated Services

Longsong Lin[†] Tianji Jiang[*] Jeffrey Lo[‡]

[†] MMC Networks, Inc
1134 E. Arques Ave.
Sunnyvale, CA 94086
llin@mmcnet.com

[*] College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
tjiang@cc.gatech.edu

[‡]C&C Research Lab., NEC USA
System Architecture Department
110 Rio Robles. San Jose, CA 95134
jlo@ccrl.sj.nec.com

Abstract—
Differentiated service (Diffserv) is a set of technologies used to enable the provisioning of class-based, differentiated quality of service. One of the essential building blocks to realize Diffserv is the use of traffic conditioning mechanisms to support the required forwarding behavior. In this paper, a generic model for traffic conditioning is proposed. The generic traffic conditioner (GTC) can be used to provide the traffic control functionality required in support of various defined per-hop forwarding behaviors. To demonstrate this capability, a configuration of GTC for an expedited forwarding that guarantees low packet loss, delay and jitter bounds is presented. Analysis of per-hop delay and jitter due to conditioning traffic is presented and buffer requirements for GTC and interface are also derived. The GTC is currently implemented in Linux platform. Experimental results have shown that the GTC can achieve minimal packet loss with bounded jitter and delay.

Keywords— **Quality-of-Service Network, Differentiated services, Traffic Conditioning**

I. Introduction

Motivated by rapid change of requirements in network applications, the Internet has been evolving towards providing wide variety of services to meet the qualities of information delivery demanded by the applications. For the past few years, there have been two major efforts focused on augmenting the single-class, best effort Internet to include different levels of guarantee in service quality - Integrated service (Intserv) [1] and Differentiated service (Diffserv)[2][3].

The most salient difference between these two approaches lies in the treatment of the packet streams. Inteserv tends to emulate circuit-switch networks, focusing on guaranteeing QoS on individual packet flows between communication end-points. To ensure the level of guarantee on a per-flow basis, it requires explicit signaling to reserve corresponding resources along the path between these end-points. One protocol called Resource Reservation Protocol (RSVP) [4] serves for this purpose to setup the states at each node along a path. One of the major dilemmas faced by this approach is that in the core of the Internet where several millions of flows are usual per se, it may not be feasible to maintain and control the forwarding states efficiently. These scalability and management problems are addressed lately by Diffserv approach. Instead of on per-flow basis, Diffserv aims at treatments on an aggregate of flows, a set of micro-flows with similar service requirements. By limiting the number of classes of services, it is anticipated that this approach scales well in backbone nodes and requires no setup signaling and state maintenance. Nevertheless, the loss of per-flow states and granularity due to aggregation may results in more diverse and coarse level of assurance on quality.

Despite of the discrepancy in the facet of service deployment, both approaches employ a common set of mechanisms to realize the services. First, a classifier is used to categorize packets into a micro-flow (Intserv) or an aggregate class (Diffserv). This classifier implements an ordered set of filters that are used to distribute an incoming packet into one of the classes. Packets pertaining to a class are buffered at a specific queue to be scheduled for transmission. In general, an egress interface could be equipped with several such queues that share the bandwidth of the link. A queue discipline is used to undertake the role of apportioning the link bandwidth according to the requirement of each class.

The crux of Diffserv model lies in differentiation of micro-flows at boundary of a DS-domain and aggregation of micro-flows of the same service class at core of the DS-domain. At each ingress interface of a boundary node in a DS domain, the traffic is passed to a Multi-Field Classifier (MFC) which may classifies the packet based on 5-tuples of the IP header, MAC addresses, VLAN identifiers, link-layer traffic classes or other higher layer protocol addresses. After the class of a flow is identified, each packet belonging to the same class will be conditioned by a traffic conditioning (TC) block according to the TCS and TP. Leaving the boundary node, each packet is marked with a Diffserv codepoint (DSCP). Thereafter, a Behavior Aggregate Classifier (BAC) in each interior node of a DS domain will classify the packets based only on the DSCP. At each egress interface of a DS node, a scheduling discipline that enforces the PHB for the behavior aggregate dispatches the packets among a set of inter-dependent queues. In the sense, PHB is a means by which a node allocates resources to behavior aggregates, and it is on top of this basic hop-by-hop resource allocation mechanism that useful differentiated services may be constructed. PHBs are implemented in nodes by means of some buffer management and packet scheduling mechanisms and the parameters associated with the mechanisms are closely related to those of traffic conditioning.

A general architecture of a Diffserv node is depicted in Figure 1 showing that packet classification and traffic conditioning are performed at the ingress interface before the forwarding engine while queuing discipline and possibly classification and

This work was done when Tianji Jiang worked as a summer intern in the NEC C&C Research Labs, San Jose, CA in summer 1999. At that time, Dr.Longsong Lin was also with the NEC C&C Research Labs, San Jose, CA

traffic conditioning are performed at egress interface after route lookups. The picture enumerates different classifiers, which include, to support multiple customers per interface, a combination of a MFC and a BAC to distinguish traffic based on customer as well as service level. For a DS interior node, there might not need conditioning before the forwarding process and packets with DS bytes set are passed to the BA classifier of a outgoing interface directly.

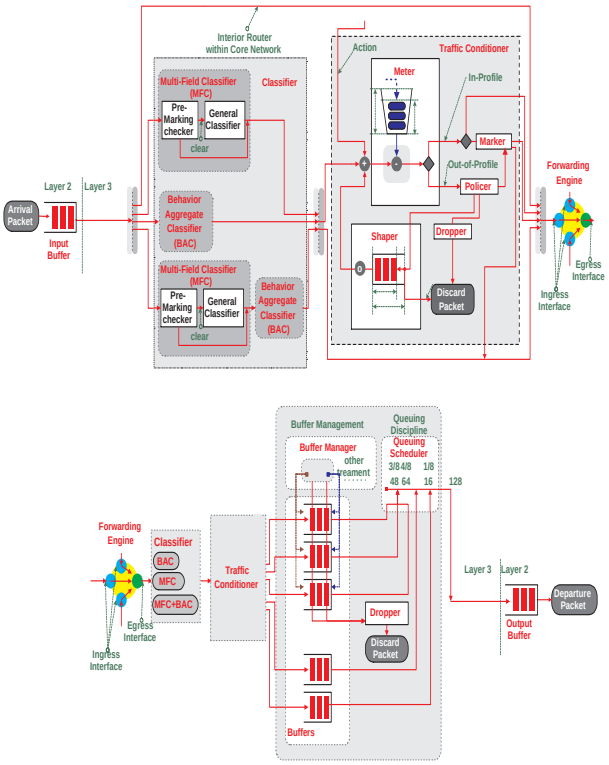

Fig. 1. A general architecture of a Diffserv node

Diffserv to date has offered three groups of PHB: Class Selector (CS), Expedited Forwarding (EF) and Assured Forwarding (AF) [5][6][7]. Without per-flow states in the backbone nodes and with optional signaling protocol for admission control, EF PHB promises to deliver a "virtual lease line(VLL)"-like end-to-end service with a low loss, low latency, low jitter, assured bandwidth through DS domains. AF PHB group provides assurance of quality according to the relative ordering between classes, rather than absolute service level for each class. Class Selctor PHBs subsumes the Precedence bits of the IP packet header to codepoints in order to preserve the backward compatibility with most of the deployed forwarding treatments selected by the IP Precedence field. For different PHBs, there are different constraints and requirements that must be fulfilled; hence, it often requires the supports of certain traffic conditioning functions. For example, to realize EF PHB, it is desirable to configure the PHB mechanisms to guarantee a specified rate for an aggregate and condition the aggregate so that its arrival rate at any node is always less than that node's configured minimum departure rate.

The main theme of this paper is to investigate the traffic conditioning components and then propose a generic type of traffic conditioning block called generic traffic conditioner (GTC) to support the standard PHBs.. There have been many traffic meters proposed for specific standard PHB [8][9] [10]. Our goal is to propose a single traffic conditioning architecture that can be configured to support the various PHBs. To show this capability of the GTC, we will show how to configure the traffic conditioning components to serve different PHBs. In particular, we shall demonstrate, through analysis and evaluation, how the GTC can support the PHB to guarantee low delay, jitter and loss as defined in EF PHB. An example of application of such a PHB is on the type of applications in which packet delay and jitter are critical while loss can be amortized by certain encoding techniques. Such applications could use services provided by EF PHB or AF PHB group with the support of traffic shaping and packet re-marking. In such case, we could target at minimizing the packet loss, given delay and jitter constraints. We will present the GTC algorithm, proofs of theoretical bounds for jitter and delay, and buffer requirement at interface. We will show the implementation of our design in Linux platforms and collect experimental data to validate our analytical results.

II. General Model for Traffic Conditioning

In general, a traffic conditioner block may contain different combinations of meter, marker (re-marker), shaper, dropper, and policer. The proposed traffic conditioner is generic in the sense that the components of its architecture can be enabled and configured to support currently defined PHB groups.

The essence of the GTC architecture is that it embodies shaping and policing mechanisms that can be configured to meet the requirement of any PHB. As meter and marker are two common denominators for all the PHBs, they play passive role in a traffic conditioning block. A meter only gauge the characteristic of the traffic, it should not change any nature of the traffic. If there is any change made to the traffic, it must be caused by the shaper. The marker in the GTC marks packet in accordance with the outcome of meter and policy components. Of course, the GTC can be configured to bypass the marking process if it is not necessary, which could occur when the packet is pre-marked by and submitted from a trusted party. In contrast, a shaper actively rectifies the pattern of the traffic, attempting to fit it to the committed traffic profile. Hence, an out-of-profile packet can be re-shaped as in-profile packet only through a shaper, rather than meter or elsewhere.

A common practice of meter consists of a token bucket parameterized by an average token generating rate and bucket size. Packets are stored in the buffer of an ingress interface before being measured by the token bucket. The buffer in front of the token bucket actually acts as a shaping buffer as incoming packets will pass if the bucket has enough tokens, or queued otherwise, or dropped if the buffer is full. Hence the buffer and token bucket are tied together closely to function as meter and shaper. An example of such implementation can be found in [2] where a queue is enlisted in front of the meter and packets are buffered there before submitting them for measurement. However, the shaper could be a separate component from the meter and its realization is very much implementation dependent. In principle, the shaper processes only out-of-profile packets and outputs them in certain sequence by a sequencer for measurement later. Such a generalization is reflected in the GTC architecture, as shown in

Figure 2 where a generic shaper outputs out-of-profile packets to meter for measurement depending on the policy. The interface queue buffers packets for measurement and packets overflowing the queue size will be dropped. The GTC also drops packets overflowing shaping buffer. In GTC, dropping could be a result of either policing or buffer overflow. As a comparison, a simple FIFO queue in front of meter may drop overflowing packets and queues non-conformant packets, but can not support dropping due to shaping or policing.

Furthermore, the GTC contains a policing component in which various service policy can be realized. The policer determines the packet treatment depending on the service policy. It could possibly change the forwarding behavior of an out-of-profile packet by re-marking or just dropping the packet. For example, a policer can dispose an out-of-profile EF packet to AF PHB by re-marking the packet to one of the AF classes. It could also enforce, for example, a low-loss service policy by submitting the packet to a shaper, where the packet waiting for being re-measured as in-profile. The policing component is general enough to embrace different complexity of service policy. It is the most distinguishing building block that differs Diffserv from Intsev. Although both can enforce service policy in multi-field classifier, Intserv could not enforce the policy to change the existing forwarding behavior of a packet. These two intrinsic features are manifested in Figure 2.

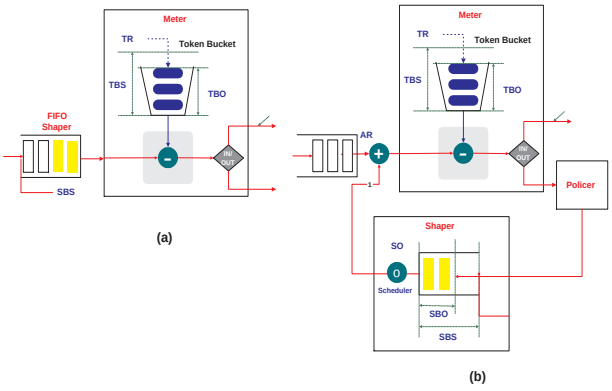

Fig. 2. A comparison of FIFO shaper and general shaper

It should be emphasized that a shaper usually delays the packets in sync with the parameters set at the meter in order to regulate the packets to be in-profile. The delay along with shaping could be harmful to the overall performance of the traffic. For example, for applications that requires particular bounds on low jitter, delay but tolerable low loss, a shaper should be designed without causing any violation of the bounds while trying to maintain low packet loss. Also, since shaping a packet will impede the newly incoming packets from being measured immediately, the buffer at ingress interface may get overflow quickly. Furthermore, when performing shaping, care should be taken as not to cause re-ordering of the original packet stream. This requires prioritizing the measurement order of the packets from shaping buffer over the newly incoming packets. The architecture of the generic traffic conditioning block is depicted in Figure 3.

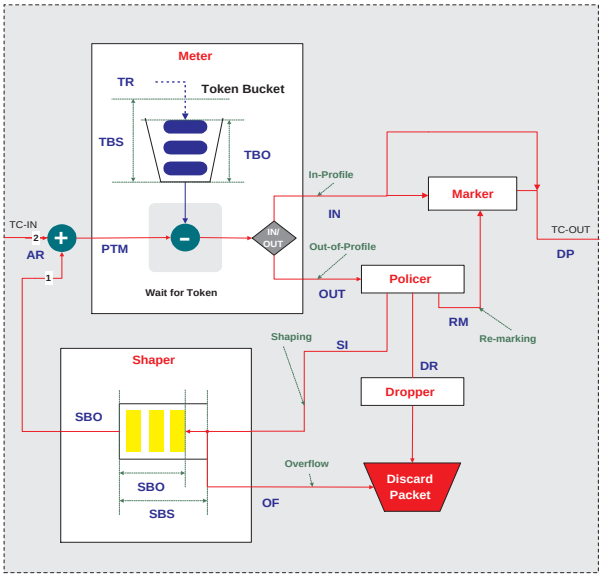

Fig. 3. The architecture of the generic traffic conditioner.

A. The GTC Algorithm

Given a traffic profile, GTC starts to operate as packets arrive at the meter and outputs marked packets from the maker. The meter picks up packets to measure first from the shaper and then from the input queue to GTC. The meter measures the packets and recognizes each of them as IN-profile (IN) or Out-of-Profile (OUT). The GTC uses a token bucket to measure packets. Tokens are generated in an average rate TR up to the bucket size TBS. An arrival of a packet triggers the measurement based on the current token bucket occupancy and the size of the arrival packet. Consequently, packet that is in-profile will be marked with corresponding DSCP in the marker. For out-of-profile packets, the policy determines whether they should be shaped, re-marked, or dropped depending on the service policy.

If shaping is required, these packets are re-queued in a shaping buffer waiting for meter's tokens. The shaping buffer can store up to SBS packets and packets overflowing the buffer occupancy will be discarded. The shaper is a simple FIFO queue that delays the out-of-profile packets for a certain time before submitting them to the meter. Although the shaper used in GTC is a simple FIFO, it does not preclude the use of other shaping mechanisms that schedule the sequence of submitting packets to the meter. It is also possible to enforce some drop preference in shaping buffer in case that the shaper needs to selectively drop packets to maintain the quality of packet in terms of delay and jitter. In that case, the drop preference could be assigned in the policer.

If the policer decides to change the PHB of the packet, it notifies the marker to re-mark the packet into other codepoint. Changing the PHB of a packet means either downgrading or upgrading the service class of the packet. However, changing codepoint does not necessarily mean to change a PHB. For EF PHB, packets may be re-marked at a DS boundary to only other codepoints that satisfy the EF PHB. This often occurs at the boundary between DS domains where different DSCPs have an equivalent PHB implementation. A DS boundary node may use a MF classifier to identify a micro-flow and configures a GTC with

a meter, policer, and shaper to regulate the packets of the flow to be in-profile and mark them with a corresponding codepoint. A DS interior node may use a BA classifier to identify an behavior aggregate. Then, the node can configure a GTC with a meter to measure its rate and a marker to re-mark the EF codepoint of the behavior aggregate to an PHB-equivalent codepoint.

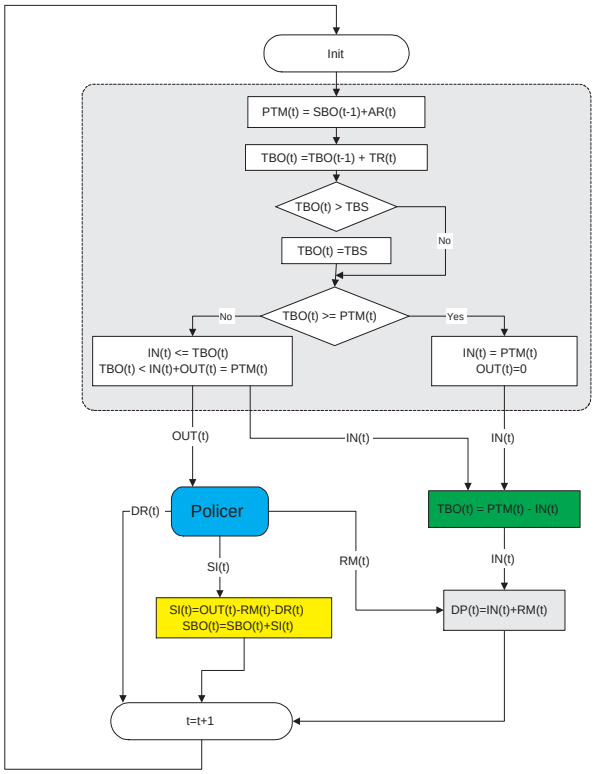

Fig. 4. Algorithm of the traffic conditioner

The token bucket is initially (at time 0) full, i.e., the token bucket occupancy $TBO(0) = TBS$, while the shaping buffer is initially empty, i.e., the shaping buffer occupancy $SBO(0) = 0$.

The TBO is increased by TR bytes per unit time. Hence the total available token for the data stream at time t is

$TBO(t) = TBS$, for $t = 0$.
$TBO(t) = TBO(t-1) + TR(t)$, for $t > 0$.
If $TBO(t) > TBS$, then $TBO(t) = TBS$.

The packets to be measured (PTM) by token bucket come from two sources: one from the input arrival at the traffic conditioner (AR) and the other from the feedback from the output of shaping buffer (SBO), i.e.,

$$PTM(t) = SBO(t-1) + AR(t).$$

If $TBO(t) < PTM(t)$, then the total number of bytes in the packets that are in-profile (IN) will be $IN(t) <= TBO(t)$, and the total number of bytes in the packets that are out-of-profile (OUT) should satisfy $TBO(t) < IN(t) + OUT(t) = PTM(t)$. Else, if $TBO(t) >= PTM(t)$, then all the packets are in-profile, i.e. $IN(t) = PTM(t)$ and $OUT(t) = 0$.

The policer operates on each of the $OUT(t)$ packets in turn. It decides how each packet is treated depending on the present occupancy of the shaping buffer as well as service policy. Shaping can only be applied if not violating the specified per-hop behavior. Otherwise, packets should be dropped. In the current implementation of policy, a packet whose addition will overflow the shaping buffer size should be dropped. If not dropped, the packet will be en-queued in the shaping buffer for later re-measurement or directly re-marked to a code-point corresponding to other classes of service, resulting in demoting or promoting to other PHBs. In this regard, re-marking as a result of policing may change the forwarding behavior of the packet. This may cause out of order delivery of the packet to the destination, if even only demoting is allowed when the codepoint changes. However, if packets are re-marked only to the codepoints of the same class as they were, e.g., AF1x, and then forwarded to the PHB queue with respect to that class, then re-ordering will not occur. In the regard, re-marking changes the forwarding behavior by differentiating the preferences of packet treatment, for example, drop probability, in the PHB queue. Specifically, for a out-of- profile packet with length P,

If $SBO(t) + P > SBS$, then
  drop the packet, $DR(t) += Pi$,
Else
  shape the packet, $SI(t) += Pi$, or
  re-mark the packet, $RM(t) += Pi$.

At any time instance, the token bucket is updated by $TBO(t) = TBO(t) - IN(t)$. Similarly, the shaping buffer is updated by $SBO(t) = SBO(t-1) + SI(t)$, where $SI(t)$ is the result of subtracting the packet being re-marked and dropped from the $OUT(t)$ packets in the time instance. That is, $SI(t) = OUT(t) - RM(t) - DR(t)$. In case that $SBO(t) > SBS$, then discard the overflow packets (OF) and $SBO(t) = SBO(t) - OF(t)$. Figure 5 shows a minimum configuration for a policer in which the criterion for dropping a packet is when adding of the packet overflows shaping buffer. The shaping buffer is managed under a minimum configuration that the buffer should not be overflowed. It is possible to exercise more sophisticated methods of buffer management, such as RIO or RED to selectively drop packets. In that case, the policer should enforce the stamping of drop preference for each packet passing by.

III. Delay and Jitter Analysis

In this section, we shall analyze the per-hop delay and jitter behavior due to the traffic conditioning. Our goal is to appropriate parameters for each component in order to achieve the specified PHB requirements. Then, we focus our discussion on provisioning the EF PHB by allocating bandwidth in a DS-domain. We shall derive a necessary condition of bandwidth allocation for EF PHB.

A. Traffic Conditioning Delay

A shaper that regulates non-conformant packets into conformant is the major factor causing delay and jitter of a packet. A meter ideally should not cause any delay of a packet; however, its size affects burst and delay jitter of a packet. In fact, the larger the size of a meter buffer, the greater the burst and jitter a packet may experience. For example, if the token bucket is set to the size of the largest possible IP packet (MTU) in a stream,

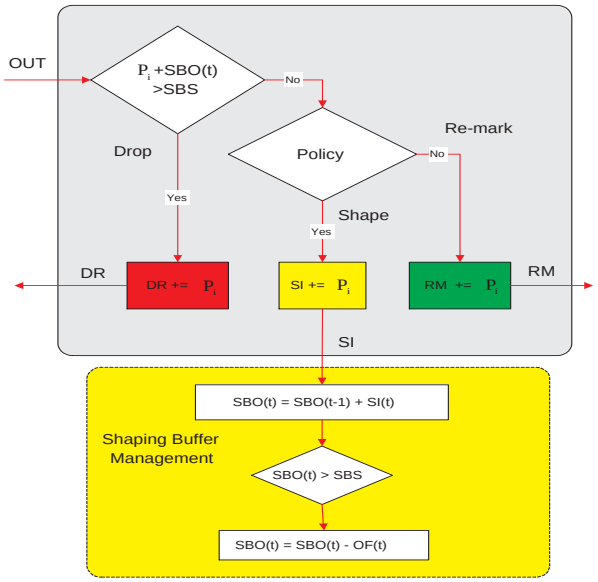

Fig. 5. A minimum configuration of policing and shaping components.

the largest jitter a packet can expect is the size of shaping buffer in packets times MTU. The situation could occur when a packet comes to the meter that has an empty bucket and the shaper is full of packets of size MTU, including this out-of-profile packet. Double the bucket size will double the jitter. In words, the size of shaping buffer affects delay and jitter while the size of meter bucket affects the burst and jitter. Hence, it is important to configure the SBS and TBS according to the requirement of delay and jitter for each node.

Assume that a packet of length P ($\leq TBS$) arrives to an interface device at time t_a and at that time, the packet sees the token bucket occupancy of $TBO(t_a)$ in meter. Moreover, also assume the packet is fully received in the queue of an ingress interface at time t_b. If the line rate of the physical device is LR, then the complete packet should be received in the interface buffer at time $t_b = P/LR + t_a$. If the number of tokens is less than P at time t_b, then by the time

$$t_c = \frac{P - TBO(t_b)}{TR} + t_b,$$

the token bucket should have enough tokens for the packet, i.e., the packet is conformant. Therefore, the delay experienced by the packet is

$$t_c - t_b = \frac{P - TBO(t_b)}{TR}.$$

If t_c is not the time the packet can be measured, then up to the time

$$t_m = \frac{TBS - TBO(t_b)}{TR} + t_b,$$

the bucket will be full, and the packet should be recognized as conformant. Thus, the delay experienced by a packet of size P due to waiting for tokens at meter is

$$\frac{P - TBO(t_b)}{TR} \quad (1)$$

which is *at most*

$$\frac{min(TBS, MTU) - TBO(t_b)}{TR}$$

Note that here we assume the size of a packet is less than or equal to min(TBS, MTU). Otherwise, the packet will be discarded anyway.

Following above derivation, if there are *no* packets in shaping buffer when the packet arrives to the interface device, then the traffic conditioning delay, d_{tc}, incurred by a packet of size P is given as Equation (1). On the other hand, if there are $k-1$ more packets in shaping buffer when the packet arrives to the interface device, then the traffic conditioning delay, d_{tc}, experienced by the packet, due to its waiting for tokens in meter as well as shaping other $k-1$ packets in shaper is given as

$$d_{tc} = \sum_{i=1,\cdots,k} \{\frac{P_i - TBO(t_b, i)}{TR}\}. \quad (2)$$

where P_i is the size of ith packet and $TBO(t_b, i))$ is the token bucket occupancy when the ith packet is about to be measured by the meter. In the worst case where all the packets are of size equal to min(MTU,TBS) and token bucket occupancy is always empty, the *maximum delay* incurred by a packet due to traffic conditioning is

$$k * \min(TBS, MTU) * \frac{1}{TR}. \quad (3)$$

According to the delay constraint a packet may tolerate at a node, we will show how to determine the number k in next section.

B. Per-hop Delay

For each packet, the delay incurred at each hop generally comes from four major sources: traffic conditioning delay d_{tc}, packet scheduling delay d_s , packet transmission delay d_t and packet classification & forwarding lookup delay d_{cf}. Usually delay d_{cf} is a small constant value and we assume it to be *zero* in the following analysis. Thus, the per-hop delay of a packet is equal to

$$d_{tc} + d_s + d_t$$

The traffic conditioning delay d_{tc} of a packet is shown as Eq.(2). The packet transmission delay d_t of a packet with length P and having physical output line rate LR [1] is equal to P/LR. However, the packet scheduling delay d_s of a packet is dependent on the queuing discipline used by a scheduling mechanism. For example, for Weighted Fair Queuing (WFQ), d_s is P/r, where P is packet size and r represents the bandwidth that a flow is entitled to receive.

Based on above description and Equation.(2), the per-hop delay of a packet is given as

$$\sum_{i=1,\cdots,k} [\frac{P_i - TBO(t_b, i)}{TR}] + d_s + \frac{P_k}{LR} \quad (4)$$

where the new arriving packet is considered as the k th packet and P_k is its length in bytes. In the worst case where all the packets are of size equal to min(MTU,TBS) and token bucket occupancy is always empty, the *maximum per-hop delay* incurred by a packet is

$$k * \min(TBS, MTU) * \frac{1}{TR} + d_s + \frac{\min(TBS, MTU)}{LR} \quad (5)$$

[1]We assume that the line rates at Ingress and Egress ports are same.

C. Determining Shaping Buffer Size (SBS)

Generally, according to the specification of a traffic aggregate, the average per-hop delay of a packet can be derived and it should be maintained within a certain bound. Suppose the bound is D, then from Eq.(5), it should have

$$k * \min(TBS, MTU) * \frac{1}{TR} + d_s + \frac{\min(TBS, MTU)}{LR} \leq D$$

and then we obtain the worst-case number of packets K_{max} that the shaper may contain without violating the delay bound requirement of any packet

$$K_{max} = \lfloor \frac{D - d_s - \min(TBS, MTU)/LR}{\min(TBS, MTU)/TR} \rfloor \qquad (6)$$

where $\lfloor x \rfloor$ means the largest integer that is less than or equal to x. Therefore, given the delay bound of a packet, we can obtain the maximum size of the shaping buffer, termed as SBS, in bytes

$$\begin{aligned} SBS &= K_{max} * \min(TBS, MTU) \\ &= \lfloor \frac{D - d_s - \min(TBS, MTU)/LR}{\min(TBS, MTU)/TR} \rfloor * \min(TBS, MTU) \end{aligned}$$

We will deploy this SBS measure in the implementation discussed later.

D. Per-hop Delay-Jitter and Token Bucket Size

In general, jitter, or variance of delay, that a packet can experience in a node is constrained by the maximum delay the packet may encounter. Specifically per-hop jitter is defined as the difference of per-hop delays between two consecutive packets of a stream. While end-to-end delay of a packet can be derived straightly from per-hop delay, end-to-end jitter of a packet is not just a summation of all per-hop jitters. In fact, the jitter of a packet at one node could be amortized at another node, resulting in a net jitter to be bounded.

For boundary node i merging n EF classes, the delay-jitter the node experience could be

$$\frac{min(TBS, MTU)}{TR} + \sum_{j=1,\ldots,n, j \neq i}^{n} w_j \frac{\min(TBS, MTU)}{LR/n} \qquad (7)$$

where w_j the allocation of bandwidth to the micro-flow j. The first term accounts for the jitter incurred due to conditioning while the second term due to scheduling. From the equation, the jitter of a node increases as its share of bandwidth allocation decreases or the number of merging classes increases.

$w_j \frac{\min(TBS, MTU)}{LR/n}$ represents the portion of the period allocated to the micro-flow j. In case that $w_i = w_j = 1/n$ for all i, j, it becomes

$$\frac{min(TBS, MTU)}{TR} + (n-1)\frac{\min(TBS, MTU)}{LR}$$

For interior node with no shaper, the delay-jitter possibly caused from merging another n EF classes of traffic is

$$n\frac{MTU}{LR}. \qquad (8)$$

The goal of conditioning traffic to low jitter is to configure the token bucket size in such a way that packets output from the traffic conditioner are spaced uniformly in token bucket rate TR. Hence, by shaping, an aggregate in boundary node shall be shaped so as to minimize its jitter, but the jitters for individual micro-flows could be intensified. Hence, each microflow may need be to further classified and conditioned at the egress boundary node. For interior nodes, jitters can be reduced by increasing the line rate.

From the above equations, per-hop jitter of a packet is largely affected by the size of token bucket. In theory, larger token bucket size results in greater variance of delay. This is because of its allowance for multiple packets to go out of the meter as these packets arrive and see a full bucket. For example, one bucket with two-MTU token bucket size (TBS) can output two packets at a time, while another with one-MTU TBS can output only one packet. The latter will space the incoming packet uniformly only if the token bucket is implemented in packet basis. Otherwise, if the bucket is implemented in bytes, which is the most often seen in practice, then it is possible that the outgoing traffic pattern is the same as the incoming traffic pattern. For instance, if a burst of four packets with one-quarter MTU each arrives at the meter and sees a full bucket, these packets will go out in the same pattern as their input's. After a certain amount of time, another burst of four packets of the same size comes and again these packets leave the meter in the same fashion. In this case, the jitter can not be well controlled in intermediate nodes of a DS domain.

E. Minimum Interface Queue Size

While minimum shaping buffer size is constrained by the delay requirement, the minimum ingress interface buffer size is determined by the line rate and meter parameters. Since a packet in the head of the shaping buffer need wait at most

$$\frac{MTU}{TR}$$

time in order to make itself conformant, the interface buffer could accumulate up to

$$LR\frac{MTU}{TR}$$

bytes of packet. This is the minimum requirement of the interface buffer for storing packets ready for measurement. As soon as there is one whole packet in the interface buffer, the meter is triggered to measure that packet. If the packet is determined to be non-conformant, then it is up to policer to decide whether the packet should be shaped or dropped. If shaped, the packet is enqueued into the shaping buffer, if there is enough capacity to accommodate it. Otherwise, the packet is dropped.

IV. Implementation and Evaluation

In this section, an implementation of the GTC is presented, followed by the experiments to evaluate the performance of the implementation, in terms of delay, jitter, and loss. The implementation is based on Linux platform.

A. Linux Network Implementation

Recent Linux kernels offer a wide variety of traffic control functions. ICA [11] implemented traffic control in the kernel and provided a set of interfaces between kernel traffic control element and user space programs. KIDS [12] used the concept of device to implement the DS functions in the kernel and implemented driver to control the DS functions. The IP network implementation in Linux is shown in Figure 6. There is only one central queue `backlog` that contains packets received from all interfaces. The `net_bh()` set a pointer in the `sk_buff` structure to the beginning of the protocol packet, after the Ethernet header. Hence the `ip_rcv` can access the IP header via this pointer. The IP header is then passed to the `ip_forwarding()` for header analysis which decide the interface the packet will be sent through. If a route is found, `Ip_queue_xmit()` is invoked to establishes the final IP protocol head and passes the packet to the `dev_queue_xmit()`. This function will actually extracts the packet's payload from the `sk_buff` in the `backlog` list and enqueues the packet with the new IP header into output queue of the corresponding network device. At this point, a special queueing discipline can in invoked. The enqueue function of the queueing discipline will invoke the classifier function to examine the packet header using a list of filters and associated traffic conditioner, and finally enqueue the packet on a specific queue of the interface.

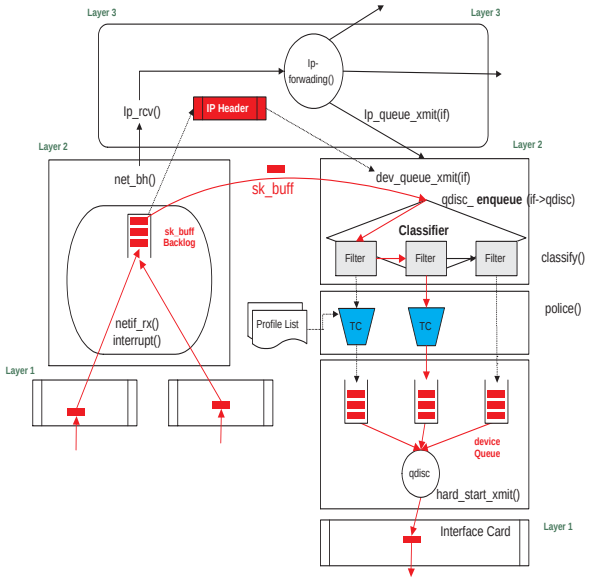

Fig. 6. Linux implementation of network and traffic conditioning

Our GTC implementation is based on the ICA Diffserve kernel; however, we found that the ICA implementation of Diffserve traffic control is not generic; hence, substantial modification is made. First, their traffic control is implemented at egress interface, including the classification. According to the Diffserv model, the classification traffic control should be generic and can be located in both ingress and egress. Secondly, the classifier is tightly coupled with policer such that if a packet requires reshaping, it needs to be classified again. We found this approach greatly degrades the system performance due to the passing of a large number of filters. Those drawbacks are remedied in our implementation. In addition, we have enhanced the traffic shaping function in the module.

Our implementation separates the classification function from the traffic conditioning function. A packet once classified will never be re-classified again. If the packet is out-of-profile and entitled to be shaped, it is stored in a shaping buffer. Any newly arrived packet will trigger a measurement on the packets in the shaping buffer first and then the new packet will becomes out-of-profile. Whether the out-of-profile packet will be discarded or shaped is subject to the policy that are configured in the traffic conditioner.

B. Experiment Results

The experimentation setup consists of two sources sending traffics to an edge node, which then route the traffics to an interior node, connecting the receiving host. The edge node, a DS-boundary node, performs micro-flow classification and traffic conditioning whereas the interior node only performs aggregate classification. The queuing discipline used for both nodes is class-based queuing. Four types of traffic are considered in the experiments: a HTTP traffic representing bursty TCP traffic, a CBR traffic with variable-size UDP data flow, a UDP traffic with bursty fixed-size UDP data flow, and a VoIP traffic representing constant bit rate UDP voice traffic. We will measure the packet loss, packets that have been discarded in the HTTP, jitter and delay for CBR and VoIP/UDP traffics.

In the first experiment, we compare the loss in the boundary router with and without the GTC. For the EF PHB, we set the TBS to 1 KBytes (MTU) and TR to 100 Kbps and increase the SBS from one to fifteen MTUs. To collect the packet loss data, we send 10 subsequent HTTP requests to the CNN web site via this EF PHB. Each HTTP session is a type of short-lived TCP connection. For each HTTP response, many requests will be generated and sent in a burst to the HTTP server again. Packet loss at the IP output queue is observed for each burst, resulting in retransmissions. However, with the GTC, the out-of-profile packets is stored in the shaping buffer, no loss occurs as the size of shaping buffer is sufficient to amortize the burst. The result is shown in Figure 7. It shows that at the boundary node without GTC, the packet loss is much greater than that with the GTC and as the SBS increases up to 6, no packet loss occurs.

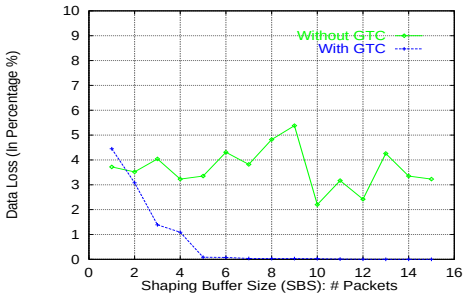

Fig. 7. A comparison of packet loss for HTTP traffic with respect to with GTC and without GTC

Next, we configure EF PHB with $min(TBS, MTU) = 1$ Kbytes, line rate $LR = 10$ Mbps, and token bucket parameters $TR = 200$ Kbps, $TBS = 1$ Kbytes. Given a delay bound D of 0.15 second, we calculate the optimal shaping buffer size

according to the Equation 6 as 3.73. Hence, we set the SBS equal to 3, and set a CBR with constant rate 100 Kbps and a background UDP data traffic with average rate 100 Kbps, both using EF PHB. From the Figure 8, we can find the delay of almost all packets is bounded below 0.15 seconds (i.e., the given delay bound). Also, the delay-jitter is found within 0.05 seconds.

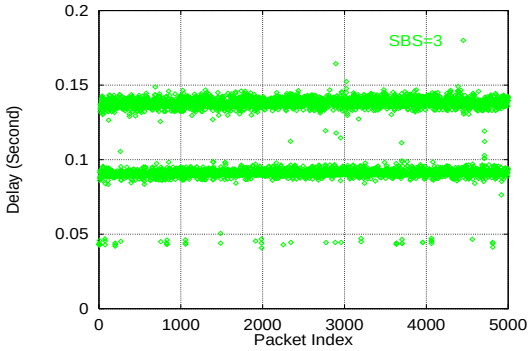

Fig. 8. Edge node produces bounded delay for CBR traffic with variable packet size

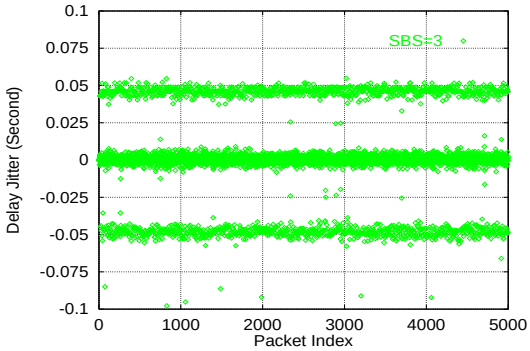

Fig. 9. Edge node produces bounded jitter for CBR traffic with variable packet size

To measure the jitter and delay due to the size of meter buffer, we configure EF PHB with token bucket rate $TR = 200$ Kbps with variable TBS and a fixed shaping buffer size of 3. The traffic model combines a VoIP traffic with a bursty UDP data traffic. The VoIP traffic sends at 40 Kbps and packet length 100 bytes (one 72-byte UDP data sent in every 20 ms) whereas the bursty UDP data traffic sends at average rate 151 Kbps with packet length 540 bytes (seven 512-byte UDP data sent in every 200 ms). Then, we vary the TBS from 600 Kbytes to 1500 Kbytes to observe the effect of TBS on jitter and delay. We measure the jitter and delay for aggregate as well as for microflow. In Figures 10 and 11, the jitter incurred due to the TC for both the VoIP flow and the aggregate with UDP data flow are presented. The figures are plotted with one point presenting the average of 20 measurement results. Because the token bucket meter for EF actually multiplexes two microflows into one aggregate, the individual microflow will experience substantial jitter introduced by the shaping. After shaping, the aggregate leaves the node in average at token bucket rate configured for the EF PHB so that the interior node needs to shape the aggregate again. Apparently, the jitter in the aggregate is smaller than that in the VoIP microflow. Also, from Figure 11, we see when the token bucket size change from the VoIP MTU to the UDP data MTU, the jitter become larger. The delay due to traffic conditioning is presented in Figure 12. It shows that the per-hop delay for VoIP traffic when merging with other bursty UDP traffic, is still bounded under 40 ms with average 20 ms, independent of other traffics on the same EF PHB, and the size of token bucket is almost irrelevant to the per-hop delay.

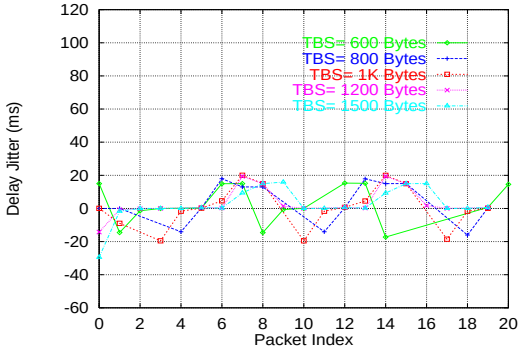

Fig. 10. Jitter of aggregate in boundary node for VoIP and UDP

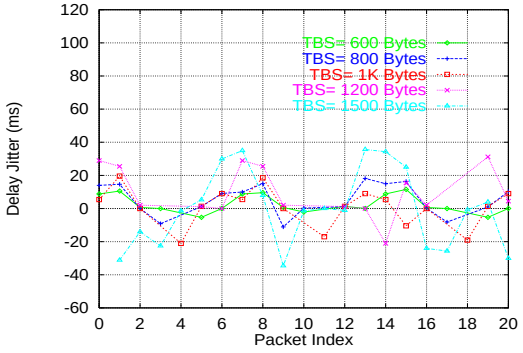

Fig. 11. Jitter of microflow in boundary node for VoIP and UDP

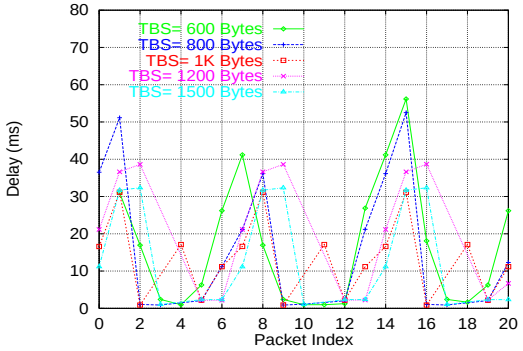

Fig. 12. Delay of microflow in boundary node for VoIP and UDP

V. Concluding Remarks

We have proposed a generic traffic conditioning architecture and implemented it in a Linux kernel. The GTC is configurable to support various forwarding PHBs. In particular, we have shown how to configure the GTC to serve for EF PHB that provides guaranteed delay and jitter bounds and low packet loss. The two important features of the GTC, shaping and policing, enable the boundary router of a DS domain to control the jitter and delay within the required bound while trying to minimize the loss of packet. We have derived mathematical expressions for per-hop delay and jitter bounds. To verify these theoretical results,

we have conducted experiments on the Linux implemented with GTC by injecting different traffic models, including HTTP, CBR, VoIP, and UDP, and observed the performance impact due to the traffic conditioning. We have shown that the GTC, when used for EF PHB, have achieved bounded delay and jitter as well as minimal packet loss.

REFERENCES

[1] S. Shenker, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Service," *RFC 2212*, September 1996.

[2] M. Carson, W. Weiss, and et al., "An Architecture for Differentiated Services," *RFC 2475*, December 1999.

[3] E. Davies, S. Keshav, and et al., "A Framework for Differentiated Services," *Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-diffserv-framework-02.txt*, February 1999.

[4] P. White, "RSVP and Integrated Services in the Internet: A Tutorial," *IEEE Communications Magazine*, May 1997.

[5] K. Nichols, S. Blake, and et al., "Integrated Services Operation over Diffserv Networks," *RFC 2474*, December 1998.

[6] V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB," *RFC 2598*, June, 1999.

[7] J. Heinanen, F. Baker, and et al., "Assured Forwarding PHB Group," *RFC 2597*, June, 1999.

[8] J. Heinanen and R. Guerin, "A Single Rate Three Color Marker," *Internet Draft, http://www.ietf.org/internet-drafts/draft-heinanen-diffserv-srtcm-01.txt*, May, 1999.

[9] J. Heinanen and R. Guerin, "A Two Rate Three Color Marker," *Internet Draft, http://www.ietf.org/internet-drafts/heinanen-diffserv-trtcm-01.txt*, May, 1999.

[10] L. Lin, J. Lo, and F. Ou, "A Generic Traffic Conditioner," *Internet Draft, http://www.ietf.org/internet-drafts/draft-lin-diffserv-gtc-01.txt*, August, 1999.

[11] R. Bless and K. Wehrle, "Evaluation of Differentiated Services using an Implementation under Linux," *http://www.icawww1.epfl.ch/linux-diffserv/*.

[12] W. Almesberger and A. Kuznetsov, "Differentiated Services on Linux," *http://www.telematik.informatik.uni-karlsruhe.de/forschung/diffserv/KIDS*, March, 1999.