Dynamic Core Provisioning for Quantitative Differentiated Services

Raymond R.-F. Liao, *Member, IEEE,* and Andrew T. Campbell, *Member, IEEE*

***Abstract*— Efficient network provisioning mechanisms that support service differentiation and automatic capacity dimensioning are essential to the realization of the Differentiated Services (DiffServ) Internet. Building on our prior work on edge provisioning, we propose a set of efficient dynamic node and core provisioning algorithms for interior nodes and core networks, respectively. The node provisioning algorithm prevents transient violations of service level agreements by predicting the onset of service level violations based on a multi-class virtual queue measurement technique, and by automatically adjusting the service weights of weighted fair queueing schedulers at core routers. Persistent service level violations are reported to the core provisioning algorithm, which dimensions traffic aggregates at the network ingress edge. The core provisioning algorithm is designed to address the difficult problem of provisioning DiffServ traffic aggregates (i.e., rate-control can only be exerted at the root of any traffic distribution tree) by taking into account fairness issues not only across different traffic aggregates but also within the same aggregate whose packets take different routes through a core IP network. We demonstrate through analysis and simulation that the proposed dynamic provisioning model is superior to static provisioning for DiffServ in providing quantitative delay bounds with differentiated loss across per-aggregate service classes under persistent congestion and device failure conditions when observed in core networks.**

***Index Terms*— Virtual Queue, Point-to-Multipoint Congestion, Service Differentiation, Capacity Dimension.**

I. Introduction

EFFICIENT capacity provisioning for the Differentiated Services (DiffServ) Internet [1] appears more challenging than in circuit-based networks such as the Asynchronous Transfer Mode (ATM) networks for two reasons. First, there is a lack of detailed control information (e.g., per-flow states) and support mechanisms (e.g., per-flow queueing) in the network. Second, there is a need to provide increased levels of service differentiation over a single global IP infrastructure. In traditional telecommunication networks, where traffic characteristics are well understood and well controlled, long-term capacity planning can be effectively applied. We argue, however, that in a DiffServ Internet more dynamic forms of control will be required to compensate for coarser-grained state information and the lack of network controllability, if service differentiation is to be realistically delivered.

There exists a trade-off intrinsic to the DiffServ service model (i.e., qualitative vs. quantitative control). DiffServ aims to simplify the resource management problem thereby gaining architectural scalability through provisioning the network on a per-aggregate basis, which results in some level of service differentiation between service classes that is *qualitative* in nature. Although under normal conditions, the combination of DiffServ router mechanisms and edge regulation of service level agreements (SLA) could plausibly be sufficient for service differentiation in an over-provisioned Internet backbone, network practitioners need to use *quantitative* provisioning rules to automatically re-dimension a network that experiences persistent congestion or device failure while attempting to maintain service differentiation [2], [3]. Therefore, a key challenge for the emerging DiffServ Internet is to develop solutions that can deliver suitable network control granularity with scalable and efficient network state management.

In this paper, we propose an approach to provisioning quantitative differential services within a service provider's network (i.e., the intra-domain aspect of the provisioning problem). Our SLA provides quantitative per-class delay guarantees with differentiated loss bounds across core IP networks. We introduce a distributed *node provisioning algorithm* that works with class-based weighted fair (WFQ) schedulers and queue management schemes. This algorithm prevents transient service level violations by adjusting the service weights for different classes after detecting the onset of SLA violations. The algorithm uses a simple but effective approach (i.e., the virtual queue method proposed in [4], [5]) to predict persistent SLA violations from measurement data and sends alarm signals to our network *core provisioning algorithm*. Our stress test results for both bursty On-Off and TCP application traffic show that the node provisioning algorithm alone can guarantee the delay and loss bounds when there is a low frequency (below 10%) of alarms raised. When there is a SLA violation, the algorithm will first meet the delay bound sacrificing the loss bound. For adaptive applications such as TCP which respond to packet losses, this approach has shown to be effective even without the involvement of core provisioning algorithms.

One challenge facing DiffServ network provisioning is the rate control of traffic aggregates that comprise flows exiting the core network at different network egress points. A rate control function includes traffic policing (i.e., packet dropping) and/or traffic shaping. This problem occurs when rate control can only be exerted on a per traffic aggregate basis, (i.e., at the root of a traffic aggregate's point-to-multipoint distribution tree). Under such conditions, any rate reduction of an aggregate would penalizes traffic flowing along branches of the point-to-multipoint distribution tree that are not congested. We

Raymond Liao is with Siemens TTB, 1995 University Ave., Suite 375, Berkeley, CA 94704, USA. This research was conducted while he was with the Dept. of Electrical Engineering, Columbia University. Andrew T. Campbell is with the Dept. of Electrical Engineering, Columbia University, New York, NY10027, USA. E-mail: {liao, campbell}@comet.columbia.edu

call such a penalty *branch-penalty*. Branch-penalty exists in DiffServ networks because rate control is performed at the ingress edge of the network instead of in the core of the network. Existing flow control algorithms have focused on fairness across different traffic aggregates while overlooking the effect of branch-penalty, which can lead to severe bandwidth reduction on traffic aggregates whose portion of traffic flowing through a congested link is small, and resulting in unnecessary under-utilization of network links that are not congested. Our approach, in contrast, comprises a suite of policies that minimize branch-penalty, deliver fairness with equal reduction across traffic aggregates, or extend the max-min fairness for point-to-multipoint traffic aggregates.

In summary, this paper makes two contributions. First, our node provisioning algorithm prevents transient service level violations by dynamically adjusting the service weights of a weighted fair queueing scheduler. The algorithm is measurement-based and effectively uses the multi-class virtual queue technique to predict the onset of SLA violations. Second, our core provisioning algorithm is designed to address the unique difficulty of provisioning DiffServ traffic aggregates. We proposed an algorithm that balances the trade-off between fairness and minimizing the branch-penalty. Collectively, these algorithms contribute toward a more quantitative differentiated service Internet, supporting per-class delay guarantees with differentiated loss bounds across core IP networks.

This paper is structured as follows. In Section II, we discuss related work. In Section III, we introduce a dynamic provisioning architecture and service model. Following this, in Section IV, we present our dynamic node provisioning mechanism, which monitors buffer occupancy, self-adjusts scheduler service weights and packet dropping thresholds at core routers. In Section V, we describe our core provisioning algorithm, which dimensions bandwidth at ingress traffic conditioners located at edge routers taking into account the fairness issue of point-to-multipoint traffic aggregates and SLAs. In Section VI, we discuss our simulation results demonstrating that the proposed algorithms are capable of supporting the dynamic provisioning of SLAs with guaranteed delay, differential loss and bandwidth prioritization across per-aggregate service classes. We also verify the effect of rate allocation policies on traffic aggregates. Finally, in Section VII, we present some concluding remarks.

II. Related Work

Dynamic network provisioning algorithms are complementary to scheduling and admission control algorithms. The provisioning algorithms introduced in this paper operate on a medium time scale, as illustrated in Fig. 1. In contrast, packet scheduling and flow control operate on fast time scales (i.e., sub-second time scales); admission control and dynamic provisioning operate on medium time scales in the range of seconds to minutes; and traffic engineering, including rerouting and capacity planning, operate on slower time scales on the order of hours to months. Significant progress has been made in the area of scheduling and flow control, (e.g., dynamic packet state and its derivatives [6], [7]). In the area of traffic engineering, solutions for circuit-based networks have been widely investigated in literature (e.g., [8], [9]). There has been recent progress on developing measurement techniques for IP networks [10]–[12]. In contrast, for the medium time scale mechanisms, most research effort has been focused on admission control issues including edge [13] and end host based admission control [14]. However, these algorithms do not provide fast mechanisms that are capable of reacting to sudden traffic pattern changes. Our dynamic provisioning algorithms are capable of quickly restoring service differentiation under severely congested and device failure conditions.

Delivering quantitative service differentiation for the DiffServ service model in a scalable manner has attracted a lot of attentions recently. A number of researchers have proposed effective scheduling algorithms. Stoica et. al. propose the Dynamic Packet State [6] to maintain per-flow rate information in packet headers leading to fine-grained per-flow packet-dropping that is locally fair (i.e., at a local switch). However, this scheme is not max-min fair due to the fact that any packet drops inside the core network wastes upstream link bandwidth that otherwise could be utilized. In [7], Stoica and Zhang extend the solution of [6] to support per-flow delay guarantees in a DiffServ network. Our work operates on top of per-class schedulers with emphasis on bandwidth allocation and the maintenance of service differentiation and network-wide fairness properties. The proportional delay differentiation scheme [15] defines a new qualitative "relative differentiation service" as oppose to quantifying "absolute differentiated services". The node provisioning algorithm presented in this paper also adopts a self-adaptive mechanism to adjust service weights at core routers. However, our service model differs from [15] by providing delay guarantees across a core network while maintaining relative loss differentiation. The work discussed in [16] has similar objectives to our node provisioning algorithm. However, it is motivated by a more comprehensive set of objectives in comparison to our work because it attempts to support optimization objectives that include multiple constraints for both relative and absolute loss and delay differentiation.

The idea of using virtual queues in scheduler design is a well accepted technique. For example, in [17] a duplicate queue is constructed to support two "Alternative Best-Effort" services (viz. low delay vs. high throughput). In our work, we use virtual queues to predict the onset of SLA violations. The idea was originally proposed in [4], [5] as a good traffic prediction technique for traffic with complex characteristics, such as, self similarity, because its stochastic properties share the same dominant time scale with the original queue. Our algorithm extends this work by dynamically adjusting the virtual queue scaling parameter with respect to queueing conditions.

Our approach to dynamic provisioning is complementary to the work on edge/end-host based admission control [13], [14], with admission control at the edge of core networks and provisioning algorithms operating inside core networks. An alternative approach that solely uses admission control for a DiffServ network can support stricter QoS guarantee but also lead to more complexity in the QoS control plane. For example, in [18] a complex bandwidth broker algorithm is

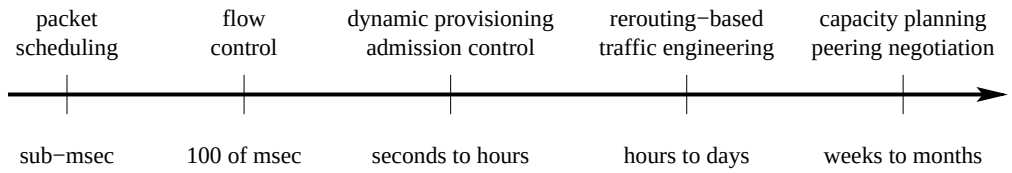


Fig. 1. Network Provisioning Time Scale

presented to maintain the control states of core routers and perform admission control for the whole network. In contrast, our provisioning algorithm uses a distributed node algorithm to detect and signal the need for bandwidth re-allocation. The centralized core algorithm only maintains the network load matrix and coordinates the allocation algorithm for fairness purposes.

One could argue that this problem could be resolved by breaking down a customer's traffic aggregates into per ingress-egress pairs and provisioning these pairs in a similar manner to circuit-based Multi-Protocol Label Switching (MPLS) [19] tunnels. However, such an approach would only work if the tunnel topology of virtual private networks (VPN) is a mesh. It would not work if a more scaleable hub-and-spoke topology is used for deploying VPNs because hub-and-spoke topologies lead to point-to-multipoint distribution trees. In addition, this approach would not work when the number of tunnels exceeds the number of shaper queues supported in edge routers. Our approach does not exclude support for MPLS tunnels, but benefits from any availability of MPLS tunnels because MPLS per-tunnel traffic accounting statistics will improve the measurement accuracy of our traffic matrix, as discussed in Section V-A. As a result, our approach improves the scalability of per-MPLS-tunnel traffic shaping by supporting traffic regulation for MPLS aggregates.

Currently network service providers use rerouting based traffic engineering approaches to cope with network traffic dynamics on slow time-scales. In the inter-domain case where one provider has no direct control of its peering networks, absence of direct control leads to the use of intra-domain routing policy as the only viable technique, with potential solutions ranging from optimal planning of routes for circuits/virtual paths [20], to traffic measurement based adjustment on OSPF weights and BGP route policies [10]. In the intra-domain case where direct control is possible, dynamic provisioning can offer faster response to service degradation.

Our provisioning method bears similarity to the work on edge-to-edge flow control [21] but differs in that we provide a solution for point-to-multipoint traffic aggregates unique to a DiffServ network rather than the point-to-point approach discussed in [21]. In addition, our emphasis is on the delivery of multiple levels of service differentiation.

III. Dynamic Network Provisioning Model

A. Architecture

We assume a DiffServ framework where edge traffic conditioners perform traffic policing/shaping. Nodes within the core network use a class-based weighted fair (WFQ) scheduler and various queue management schemes for dropping packets that overflow queue thresholds.

The dynamic capacity provisioning architecture illustrated in Fig. 2 comprises dynamic core and node provisioning modules for bandwidth brokers and core routers, respectively, as well as the edge provisioning modules that are located at access and peering routers. The edge provisioning module [22] performs ingress link sharing at access routers, and egress capacity dimensioning at peering routers.

B. Control Messaging

Dynamic core provisioning sets appropriate ingress traffic conditioners located at access routers by utilizing a *core traffic load matrix* to apply rate-reduction (via a *Regulate_Ingress Down* signal) at ingress conditioners, as shown in Fig. 2. Ingress conditioners are periodically invoked (via the *Regulate_Ingress Up* signal) over longer restoration time scales to increase bandwidth allocation restoring the max-min bandwidth allocation when resources become available. The core traffic load matrix maintains network state information. The matrix is periodically updated (via *LinkState_Update* signal) with the measured per-class link load. In addition, when there is a significant change in the rate allocation at egress access routers, a core bandwidth broker uses a *SinkTree_Update* signal to notify egress dimensioning modules at peering routers when renegotiating bandwidth with peering networks, as shown in Fig. 2. We use the term "sink-tree" to refer to the topological relationship between a single egress link (representing the root of a sink-tree) and two or more ingress links (representing the leaves of a sink-tree) that contribute traffic to the egress point.

Dynamic core provisioning is triggered by *dynamic node provisioning* (via a *Congestion_Alarm* signal as illustrated in Fig. 2) when a node persistently experiences congestion for a particular service class. This is typically the result of some local threshold being violated. Dynamic node provisioning adjusts service weights of per-class weighted schedulers and queue dropping thresholds at local core routers with the goal of maintaining delay bounds and differential loss, and bandwidth priority assurances.

C. Service Model

The proportional delay differentiation service proposed in [15] defines the relative service differentiation of a single node and not a path through a core network. In contrast, our work produces service assurances that are quantitative in terms of delay bound and loss differentiation, and support bandwidth allocation priorities across service classes within a DiffServ core network.

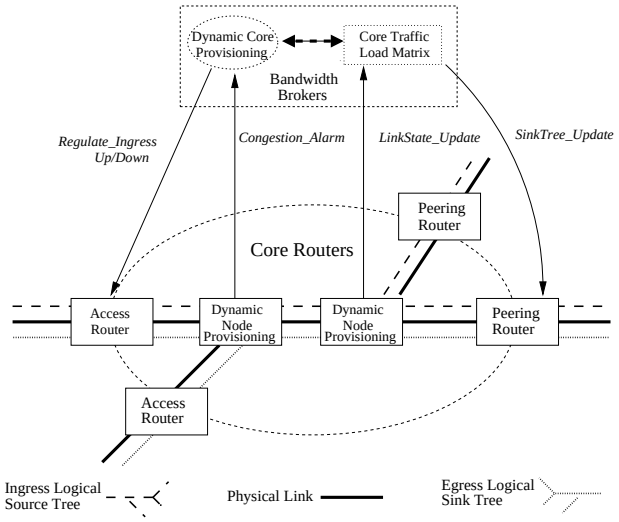


Fig. 2. Dynamic Capacity Provisioning Model

Our SLA comprises:

- a *delay guarantee*: where any packet delivered through the core network (not including the shaping delay of edge traffic conditioners) has a delay bound of D_i for network service class i;
- a *differentiated loss assurance*: where network service classes are loss differentiated, that is, for traffic routed through the same path in a core network, the long-term average loss rate experienced by class i is no larger than $P^*_{loss,i}$. The thresholds $\{P^*_{loss,i}\}$ are differentiated, i.e., $P^*_{loss,(i-1)} < P^*_{loss,i}$;
- a *delay bound precedence over loss bound*: when both the delay and loss bounds can not be maintained for class i, the loss bound will be revoked first before the delay bound;
- a *bandwidth allocation priority*: where the traffic of class j never affects the bandwidth/buffer allocation of class $i, i < j$, that is, the delay and loss bounds of class i will be revoked only after there is no bandwidth available (excluding the minimum bandwidth for each class) in classes $j, j > i$; and
- a *bandwidth utility function*: which provides an application programming interface (API) for edge service differentiation. The utility function serves as a user-approved per-class QoS degradation trajectory used by network provisioning algorithms under network congestion or failure conditions.

We design the service model such that maintaining a quantitative delay bound takes precedence over maintaining the packet loss bound. This precedence helps to simplify the complexity of jointly maintaining both loss and delay bounds at the same time. In addition, such a service is suitable for TCP applications that need packet loss as an indicator for flow control while guaranteed delay performance can support real-time applications. The precedence to delay bound does not mean that the loss bound will be ignored. For a service class with higher bandwidth allocation priority, its loss bound will be maintained at the cost of violating lower priority classes' loss and delay bounds.

In addition, the Congestion_Alarm signal from the node provisioning algorithm will give an early warning to the core provisioning algorithm, which can work with the admission control algorithm and edge-based traffic regulation algorithm to remove congestion inside the core network. One benefit of our dynamic provisioning algorithm is its ability to maintain service differentiation under unavoidable prediction errors made by the admission control algorithm.

The granularity of per-node delay bounds D_i is limited by the nature of slow time scale aggregate provisioning. The choice of D_i has to take into consideration the sum of a single packet transmission time at the link rate and a single packet service time through various fair queue schedulers [23]. This is in addition to the queueing delays due to traffic aggregates inside the core network.

The choice of the loss threshold $P^*_{loss,i}$ in an SLA also needs to consider the application behavior. For example, a service class intended for data applications should not specify a loss threshold that can impact steady-state TCP behavior. Studies [24] indicate that the packet drop threshold $P^*_{loss,i}$ should not exceed 0.01 for data applications to avoid the penalty of retransmission timeouts.

We define a service model for the core network that includes a number of algorithms. A node provisioning algorithm enforces delay guarantees by dropping packets and adjusting service weights accordingly. A core provisioning algorithm maintains the dropping-rate differentiation by dimensioning the network ingress bandwidth. Edge provisioning modules perform rate regulation based on utility functions. Even though these algorithms are not the only solution to supporting the proposed SLA, their design is tailored toward delivering quantitative differentiation in the SLA with minimum complexity.

Note that utility function based edge dimensioning has been investigated in our prior work [22]. In the remaining part of this paper we focus on core network provisioning algorithms that are complementary components to the edge algorithms of our dynamic provisioning architecture shown in Fig. 2.

IV. Dynamic Node Provisioning

The design of the node provisioning algorithm follows the typical logic of measurement based closed-loop control. The algorithm is responsible for two tasks: (i) to predict SLA violations from traffic measurements; and (ii) to respond to potential violations with local reconfiguration. If violations are severe and persistent, then reports are sent to the core provisioning modules to regulate ingress conditioners, as shown in Fig. 2.

The detection of SLA violation is triggered by the virtual queue method proposed in [4], [5]. A virtual queue has exactly the same incoming traffic as its corresponding real queue but with both the service rate and buffer size scaled down by a factor of $\kappa \in (0, 1)$. The virtual queue technique offers a generic and robust traffic control mechanism without assuming any traffic model (e.g., the Poisson arrivals, etc.). It performs well under complex traffic arrival processes including self similarity [5]. In our node provisioning algorithm, we extend this technique to queues with multiple classes served by a weighted fair queueing scheduler by dynamically adjusting the scaling parameter κ_i for each class.

The algorithm is invoked either by the event of detecting the onset of an SLA violation, or periodically over an $update\_interval$ interval. The value of the $update\_interval$ does not effect the detection of SLA violations because the virtual queue mechanism can trigger the algorithm execution immediately without the constraint of the $update\_interval$. However, the $update\_interval$ will effect the speed to detect the system under-load, and the measurement of traffic statistics. In Section VI-B.2, we investigate the appropriate choice of the $update\_interval$ value.

The SLA service model introduced in Section III-C is intended to be simple for ease of implementation. However, it still requires non-trivial joint control of both service weight allocation and buffer dimensioning to maintain the delay and loss bounds D_i and $P^*_{loss,i}$, respectively.

A. Loss Measurement

When $P^*_{loss,i}$ is small, solely counting rare packet loss events can introduce a large bias. Instead, the algorithm works with the inverse of the loss rate which essentially tracks the number of consecutively accepted packets. For each class, a target loss control variable $lossfree\_cnt_i$ is measured upon each update epoch t_n. Denote $cnt_{accepted}$ the number of accepted packets during the interval $(t_{n-1}, t_n]$, and $cnt_{dropped}$ the number of dropped packets in the same interval, then we have

$$lossfree\_cnt_i(t_n) = (cnt_{dropped} + 1)/P^*_{loss,i} - cnt_{accepted}. \tag{1}$$

In other words, $lossfree\_cnt_i$ represents the number of packets that have to be accepted consecutively under the $P^*_{loss,i}$ bound before the next packet drop. $lossfree\_cnt_i \leq 0$ signifies that the $P_{loss,i}$ bound is not violated; $lossfree\_cnt_i > 1/P^*_{loss,i}$ indicates the opposite; while $lossfree\_cnt_i \in (0 \quad 1/P^*_{loss,i}]$ indicates that there have not been sufficient packet arrivals yet.

The measurement of $cnt_{accepted}$ and $cnt_{dropped}$ uses a measurement window τ_l, which is one order of magnitude larger than the product of $1/P^*_{loss,i}$ and the mean packet transmission time in order to have a statistically accurate calculation of the packet loss rate. In the simulation section, we use $\tau_l \geq 10\,\text{s}$. However, a large τ_l means that a currently partial measurement sample has to be considered for the instantaneous packet loss. To improve statistical reliability, we also use the complete sample in the preceding window for calculation, that is:

$$\begin{aligned} cnt_{accepted} &= accept\_count(prev) + \\ &\quad accept\_count\_partial(now) \\ cnt_{dropped} &= drop\_count(prev) + \\ &\quad drop\_count\_partial(now). \end{aligned} \tag{2}$$

B. Delay Constraint

Our algorithm controls delay by buffer dimensioning and service weight adjustment. Exact calculation of the maximum delay of all enqueued packets is expensive since it requires tracking the queueing delay incurred by every enqueued packet. Instead, we calculate the current maximum queueing delay with its upper bound:

$$d_i \leq \bar{d}_i \triangleq d_i(HOL) + N_q/\mu_i, \tag{3}$$

where $d_i(HOL)$ is the queue delay of the head-of-line (HOL) packet, N_q is the queue size, and μ_i is the lower bound of the packet service rate calculated from the proportion of service weights in a WFQ scheduler. μ_i is a lower bound because the actual service rate will be higher when some of the other class queues are idle. The benefit of Eq. 3 is that we only need to calculate the delay of the HOL packet. The downside of this is that $\bar{d}_i$ becomes an approximation of the current maximum queueing delay. In fact, it represents an upper bound of the current maximum queueing delay because the first portion of Eq. 3 represents the maximum queueing delay incurred by any of the enqueued packets handled so far. The bound can be reached when all the enqueued packets arrived at the same time. Note that the same technique is used in [16] to measure the maximum queueing delay.

Now with $\bar{d}_i \leq D_i$, and Ineq. 3, we obtain a lower bound for the service rate μ_i:

$$\mu_i(new) \geq N_q/(D_i - d_i(HOL)). \tag{4}$$

This means that $\mu_i(new)$ needs to be above the lower bound in order to meet the delay bound of the enqueued packets. Subsequently, the dimensioning of buffer size Q_i for the ith class queue can be derived as:

$$Q_i(new) = \tilde{D}_i, \; where$$

$$\tilde{D}_i = \begin{cases} D_i - d_i(HOL) & if\ D_i > d_i(HOL), \\ D_i & otherwise, \text{ delay bound violated} \end{cases} \tag{5}$$

C. Virtual Queue Scaling

The virtual queue technique proposed in [4], [5] needs to be extended for a WFQ scheduler with multiple queues. Denote w_i the service weight of class i, then the minimum service rate is:

$$\mu_i = \frac{w_i}{\sum_i w_i}\, linerate. \tag{6}$$

Denote κ_i the scaling parameter for the ith queue, then the buffer size of each class queue is scaled down by κ_i. For the total service rate of the WFQ scheduler, we have:

$$linerate_{VQ} = \sum_i \kappa_i\, \mu_i = \frac{\sum_i \kappa_i\, w_i}{\sum_i w_i}\, linerate. \tag{7}$$

The scaling parameter for the total service rate is $\sum_i \kappa_i \;\; w_i / \sum_i w_i$, which is the weighted average of the individual scaling parameters.

The setting of κ_i takes into consideration the speed mismatch between the instantaneous arrival rate and service rate, and the response time of the queueing system to the adjustment of service weights. The purpose is to choose κ_i such that the early warning generated from the virtual queue will give enough time for the WFQ scheduler to react.

Since the node provisioning algorithm targets operating at the buffer half-full point to counter both queue under-load and overload, we can assume that the available buffer space at the beginning of an $update\_interval$ is $Q_i/2$. In addition, we focus on the case where the traffic load $\rho_i \stackrel{\triangle}{=} \lambda_i/\mu_i > 1$, which represents the extend of the rate mismatch between queue arrival and departure. Therefore, the time that it takes to fill the real queue buffer is:

$$t_{RQ} = \frac{Q_i/2}{(\rho_i - 1)\mu_i}. \tag{8}$$

For the virtual queue, with κ_i scaling down Q_i and μ_i, we have the time that takes to fill the virtual queue buffer as:

$$t_{VQ} = \frac{\kappa_i\, Q_i/2}{(\rho_i - \kappa_i)\mu_i}. \tag{9}$$

For a WFQ style (e.g., Weight Round Robin) scheduler, we estimate the system response time to the change in service weights as i/λ_i; that is, the response time is proportional to the number of queueing classes that have higher or equal allocation priority than i, and inversely proportional to the line rate. Here we use λ_i to approximate the line-rate. Therefore, we have the following inequality in order to achieve the early warning of buffer overflow:

$$t_{RQ} - t_{VQ} = \frac{Q_i}{2\mu_i}\frac{\rho_i(1-\kappa_i)}{(\rho_i - 1)(\rho_i - \kappa_i)} \geq \frac{i}{\rho_i\, \mu_i}. \tag{10}$$

Solving this inequality, we have the upper bound for setting κ as:

$$\kappa_i = \frac{\frac{Q_i}{2i}\rho_i^2 - \rho_i(\rho_i - 1)}{\frac{Q_i}{2i}\rho_i^2 - (\rho_i - 1)}. \tag{11}$$

Fig. 3 shows some typical values of κ as a function of ρ_i, Q_i and i. The value of κ_i is sensitive to the buffer size Q_i and the number of higher or equal priority queueing classes i. However, the value of κ does not vary much for large values of ρ_i, which represents extremely bursty traffic conditions. This indicates that the dynamic adjustment of the virtual queue scaling parameter is applicable to a wide range of traffic conditions. Indeed, taking the limit of ρ_i in Eq. 11, we have:

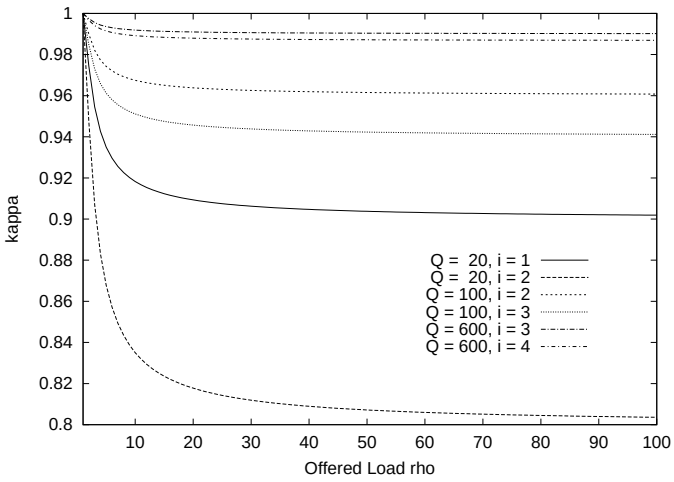


Fig. 3. Example of κ Values

$$\lim_{\rho_i \to \infty} \kappa_i = 1 - \frac{2i}{Q_i}. \tag{12}$$

This limit is also the lower bound of κ_i. It is desirable to keep the scaling parameter of a virtual queue not too small otherwise the virtual queue will generate a lot of false positive alarms. That is, $\frac{2i}{Q_i}$ should remain close to zero. Because $\frac{2i}{Q_i}$ increases as Q_i decreases, a small $Q_i \approx \mu_i\, D_i$ also means smaller delay requirements usually for higher allocation priority classes, therefore i is necessarily small as well. As a result, κ_i will stay away from values close to zero.

D. Control Action

The control action is to adjust the service rate (weight) as well as buffer size based on the short-term measurement of the traffic arrival rate $\bar{\lambda}_i$ and the queue length $\bar{N}_{q,i}$. The measurement method is the same as the dual-window averaging method used for loss measurement in Section IV-A, except that the window size is much smaller, set to be the same as the $update\_interval$ (i.e., the samples are averaged over an interval between 1 to 2 times the $update\_interval$). We find that this dual-window measurement is better than the widely used exponentially-weighted moving-average method for closely tracking the short-term variations in the sampled statistics.

The baseline assignment of the service rate uses the measured arrival rate $\mu_i(new) = \bar{\lambda}_i$. In addition to this, we decrease/increase the service rate based on the under-load/overload conditions, respectively.

We determine that a queue is overloaded when the $lossfree\_cnt_i > -burst\_loss/p^*_{loss,i}$. Here the meaning of a negative target loss-free count $-loss\_burst/P^*_{loss,i}$ provides an early response when the loss rate is within an additional $burst\_loss$ packet drops away from $P^*_{loss,i}$. In this work, we set $burst\_loss = 5$ to account for simultaneous packet drops

resulting from simultaneous arrivals at a full queue. In the case of queue overload, $\mu_i(new)$ has an additional increment from queue-length adjustment: $\left(\frac{\bar{N}_{q,i}-Q_i/2}{update\_interval}\right)^+$. The purpose of this is to use an additional workload to bring the queue length down to the half-point of the buffer size when $\bar{N}_{q,i} > Q_i/2$. After replacing Q_i with $\mu_i(new)\,\tilde{D}_i$ based on Eq. 5, we have:

$$\mu_i(new) = \bar{\lambda}_i + \left(\frac{\bar{N}_{q,i} - \mu_i(new)\,\tilde{D}_i/2}{update\_interval}\right)^+ \quad (13)$$

The solution is:

$$\mu_i(new) = \begin{cases} \frac{\bar{\lambda}_i + \bar{N}_{q,i}/update\_interval}{1+\tilde{D}_i/(2\,update\_interval)} & \text{if } \bar{N}_{q,i} \geq \bar{\lambda}_i\tilde{D}_i/2 \\ \bar{\lambda}_i & \text{otherwise} \end{cases} \quad (14)$$

Similarly, we determine a queue is under-loaded when $lossfree\_cnt_i \leq -burst\_loss/p^*_{loss,i}$. In this case, we set

$$\mu_i(new) = \max\{\mu_i(perv)\,,\, \bar{\lambda}_i\}, \quad (15)$$

The calculated $\mu_i(new)$ is then checked against the constraint of Eq. 4 and we have:

$$\mu_i(new) = \max\{\mu_i(new)\,,\, \bar{N}_q/\tilde{D}_i,\, \mu_{\min}\}, \quad (16)$$

where $\mu_{\min}$ is the minimum service rate reserved for each class to avoid starving a traffic class particularly when it transitions from idle to active.

The service rate $\mu_i(new)$ is then converted to service weight $w_i(new)$ for a WFQ scheduler. Note that $\mu_i(new)$ is the minimum service rate in a WFQ style scheduler because the unused service rate (weight) for some temporally idle classes will be proportionally allocated to busy classes. When there is congestion, i.e., not enough bandwidth to satisfy every $\mu_i(new)$, we use a strict priority in the service weight allocation procedure; that is, higher priority classes can "steal" service weights from lower priority classes until the service weight of a lower priority class reaches its minimum ($\mu_i(\min)$). We always change local service weights first before sending a Congestion_Alarm signal to the core provisioning module (discussed in Section V) to reduce the arrival rate which would require a network-wide adjustment of ingress traffic conditioners at edge nodes.

Similarly, when there is a persistent under-load in the queues, an increasing arrival rate is signaled (via the LinkState_Update signal) to the core provisioning module. An increase in the arrival rate is deferred to a periodic network-wide rate re-alignment algorithm which operates over longer time scales. In other words, the control system's response to rate reduction is immediate, while, on the other hand, its response to rate increase to improve utilization is delayed to limit any oscillation in rate allocation. In general, the timescale of changing ingress router bandwidth should be one order of magnitude greater than the maximum round trip delay across the core network in order to smooth out the traffic variations due to the transport protocol's flow control algorithm. Therefore, we introduce two control hystereses to the dynamic adjustment algorithm (Fig. 4 line (18)), in the form of a 10% bandwidth threshold and a 5 s delay.

The pseudo code for the node algorithm is detailed in Fig. 4.

dynamic adjustment algorithm

```
(1)  upon the expiration of the update_interval
        timer or the arrival of early warning
        events from the virtual queues:
(2)  IF early warning event
(3)     reset update_interval timer
(4)  ENDIF
(5)  FOR all classes 1,···,n
(6)     retrieve measurement: λ̄_i and lossfree_cnt_i
(7)     IF lossfree_cnt_i > −burst_loss/p*_loss,i   //overload
(8)        use Eq. 14 to calculate service weight
(9)     ELSE //under-load
(10)       use Eq. 15 to calculate service weight
(11)    ENDIF
(12)    use Eq. 16 to enforce lower bound on μ(new)
(13)    IF remaining service bandwidth < μ_i(new)
(14)       adjust μ_i(new) and set all μ_j(new), j > i
              to μ_min
(15)       send Congestion_Alarm signal
(16)       RETURN
(17)    ENDIF
(18)    adjust buffer size based on Eq. 5
(19)    calculate κ_i for virtual queue with Eq. 11
(20)    scale virtual queue service rate to
           κ_i μ_i(new), and buffer size to κ_i Q_i(new)
(21) END FOR
(22) IF remaining service bandwidth > 10% linerate
     for a duration > 5 s
(23)    send LinkState_Update signal to increase λ_i
(24) ENDIF
(25) RETURN
```

virtual queue prediction algorithm

```
(1)  upon the arrival of class i packets:
(2)  IF lossfree_cnt_i(now) > 1/P*_loss,i
        AND lossfree_cnt_i(now) > lossfree_cnt_i(prev)
        AND Congestion_Alarm signal not present
           for classes j ≤ i
(3)     invoke the dynamic adjustment algorithm
(4)     lossfree_cnt_i(prev) = lossfree_cnt_i(now)
(5)  ENDIF
(6)  RETURN
```

Fig. 4. Node Provisioning Algorithm Pseudo-Code

V. Dynamic Core Provisioning

Our core provisioning algorithm has two functions: to reduce edge bandwidth immediately after receiving a *Congestion_Alarm* signal from a node provisioning module, and to provide periodic bandwidth re-alignment to establish a modified max-min bandwidth allocation for traffic aggregates. We will focus on the first function and discuss the latter function in Section V-C.

A. Core Traffic Load Matrix

We consider a core network with a set $\mathcal{L} \triangleq \{1, 2, \cdots, L\}$ of link identifiers of unidirectional links. Let c_l be the finite capacity of link $l, l \in \mathcal{L}$.

A core network traffic load distribution consists of a matrix $\mathbf{A} = \{a_{l,i}\}$ that models per-DiffServ-aggregate traffic distribution on links $l \in \mathcal{L}$, where the value of $a_{l,i}$ indicates the portion of the ith traffic aggregate that passes link l. Let the link load vector be $\mathbf{c}$ and ingress traffic vector be $\mathbf{u}$, whose coefficient u_i denotes a traffic aggregate of one service class at one ingress point. Note that a network customer may contribute traffic to multiple u_i for multiple service classes

and at multiple network access points. This matrix formulation also supports multiple service classes. Let J be the total number of service classes. Without loss of generality, we can rearrange the columns of $\mathbf{A}$ into J sub-matrices, one for each class, which is: $\mathbf{A} = [\mathbf{A}(1)\vdots\mathbf{A}(2)\vdots\cdots\vdots\mathbf{A}(J)]$. Similarly, $\mathbf{u} = [\mathbf{u}(1)\vdots\mathbf{u}(2)\vdots\cdots\vdots\mathbf{u}(J)]$.

The constraint of link capacity leads to: $\mathbf{A}\mathbf{u}^{\mathbf{T}} \leq \mathbf{c}$. Fig. 5 illustrates an example network topology and its corresponding traffic matrix. In this figure, node 1, 2, 3, and 4 are edge nodes, while node 5 and 6 are core nodes. All the links are unidirectional. To better explain the construct of the traffic load matrix, we use the construct of the third column of the matrix A: $\mathbf{a}_{\cdot,\mathbf{3}}$ as an example. $\mathbf{a}_{\cdot,\mathbf{3}}$ represents the traffic distribution tree rooted at node 3, which is highlighted in the figure. Each entry $a_{l,3}$ represents the portion of node 3's incoming traffic that passes link l. For example, since 100% of node 3's incoming traffic passes through link 8, $a_{8,3} = 1$. Then at node 6, node 3's traffic is split between link 6 and 9 with a ratio of $7:3$, therefore $a_{6,3} = 0.7$, and $a_{9,3} = 0.3$. The 70% of traffic on link 6 is further split between link 2 and 3 with a ratio of $6:1$, as a result, we have $a_{2,3} = 0.6$, and $a_{3,3} = 0.1$. All the other entries in $\mathbf{a}_{\cdot,\mathbf{3}}$ are zero since they model the reserve links.

The construction of matrix $\mathbf{A}$ is based on the measurement of its column vectors $\mathbf{a}_{\cdot,\mathbf{i}}$, each represents the traffic distribution of an ingress aggregate u_i over the set of links $\mathcal{L}$. The measurement of u_i gives the trend of external traffic demands. In a DiffServ network, ingress traffic conditioners need to perform per-profile (usually per customer) policing or shaping. Therefore, traffic conditioners can also provide per-profile packet counting measurements without any additional operational cost. This alleviates the need to place measurement mechanisms at customer premises. We adopt this simple approach to measurement, which is advocated in [11] and measure both u_i and $\mathbf{a}_{\cdot,\mathbf{i}}$ at the ingress points of a core network rather than measuring at the egress points which is more challenging. The external traffic demands u_i is simply measured by packet counting at profile meters using ingress traffic conditioners. The traffic vector $\mathbf{a}_{\cdot,\mathbf{i}}$ is *inferred* from the flow-level packet statistics collected at a profile meter. Some additional packet probing (e.g., traceroute) or sampling (e.g., see [25]) methods can be used to improve the measurement accuracy of intra-domain traffic matrix. Last, with the addition of MPLS tunnels, fine granularity traffic measurement data is available for each tunnel. In this case, the calculation of the traffic matrix can be made more accurate. For example, in Fig. 5, if there is an MPLS tunnel from node 3 to node 1 to accurate report the traffic volume, $a_{2,3}$ can be calculated exactly, and the inference of $a_{9,3}$, $a_{6,3}$, and $a_{3,3}$ can also be more accurately determined after knowing the value of $a_{2,3}$.

B. Edge Rate Reduction Policy

Given the measured traffic load matrix $\mathbf{A}$ and the required bandwidth reduction $\{-c_l^\delta(i)\}$ at link l for class i, the allocation procedure *Regulate_Ingress_Down()* needs to find the edge bandwidth reduction vector $-\mathbf{u}^\delta = -[\mathbf{u}^\delta(1)\vdots\mathbf{u}^\delta(2)\vdots\cdots\vdots\mathbf{u}^\delta(J)]$ such that: $\mathbf{a}_{l,\cdot}(j) * \mathbf{u}^\delta(j)^T c_l^\delta(j)$, where $0 \leq u_i^\delta \leq u_i$.

When $\mathbf{a}_{l,\cdot}$ has more than one nonzero coefficient, there is an infinite number of solutions satisfying the above equation. In what follows, we investigate two distinctly different optimization policies for edge rate reduction: fairness and minimizing the impact on other traffic. For clarity, we drop the class (j) notation since the operations are the same for all classes.

1) Equal Reduction: Equal reduction minimizes the variance of rate reduction among various traffic aggregates, i.e.,

$$\min_i \left\{ \sum_{i=1}^{n} \left(u_i^\delta - \frac{\sum_{i=1}^n u_i^\delta}{n} \right)^2 \right\} \tag{17}$$

with constraints $0 \leq u_i^\delta \leq u_i$ and $\sum_{i=1}^n a_{l,i} u_i^\delta = c_l^\delta$. Using Kuhn-Tucker condition [26], we have:

Proposition 1: The solution to the problem of minimizing the variance of rate reductions comprises three parts:

$$\forall i \ \text{ with } \ a_{l,i} = 0, \ \text{ we have } \ u_i^\delta = 0; \tag{18}$$

then for notation simplicity, we re-number the remaining indices with positive $a_{l,i}$ as $1, 2, \cdots, n$; and

$$u_{\sigma(1)}^\delta = u_{\sigma(1)}, \cdots, u_{\sigma(k-1)}^\delta = u_{\sigma(k-1)}; \text{ and} \tag{19}$$

$$u_{\sigma(k)}^\delta = \cdots = u_{\sigma(n)}^\delta \frac{c_l^\delta - \sum_{i=1}^{k-1} a_{l,\sigma(i)} u_{\sigma(i)}}{\sum_{i=k}^{n} a_{l,\sigma(i)}}, \tag{20}$$

where $\{\sigma(1), \sigma(2), \cdots, \sigma(n)\}$ is a permutation of $\{1, 2, \cdots, n\}$ such that $u_{\sigma(i)}$ is sorted in increasing order, and k is chosen such that:

$$c_{eq}(k-1) < c_l^\delta \leq c_{eq}(k), \tag{21}$$

where $c_{eq}(k) = \sum_{i=1}^{k} a_{l,\sigma(i)} u_{\sigma(i)} + u_{\sigma(k)} \sum_{i=k+1}^{n} a_{l,\sigma(i)}$. Equal reduction gives each traffic aggregate the same amount of rate reduction until the rate of a traffic aggregate reaches zero.

Remark: A variation of the equal reduction policy is proportional reduction: to reduce each of the aggregates contributing traffic to bottleneck link l by an amount proportional to its total bandwidth. In particular, with $\alpha = c_l^\delta / \left(\sum_{\forall i, a_{l,i} > 0} a_{l,i} u_i \right)$, we have:

$$u_i^\delta = \begin{cases} 0 & \forall i \ \text{ with } \ a_{l,i} = 0 \\ \alpha u_i & \text{else.} \end{cases} \tag{22}$$

2) Minimal Branch-Penalty Reduction: A concern that is unique to DiffServ provisioning is to minimize the penalty on traffic belonging to the same regulated traffic aggregate that passes through non-congested branches of the routing tree. We call this effect the "branch-penalty", which is caused by policing/shaping traffic aggregates at an ingress router. For example, in Fig. 5, if link 7 is congested, the traffic aggregate #1 is reduced before entering link 1. Hence penalizing a portion of traffic aggregate #1 that passes through link 3 and 9.

The total amount of branch-penalty is $\sum_{i=1}^{n} (1 - a_{l,i}) u_i^\delta$ since $(1 - a_{l,i})$ is the proportion of traffic not passing through the congested link. Because of the constraint that $\sum_{i=1}^n a_{l,i} u_i^\delta = c_l^\delta$, we have $\sum_{i=1}^n (1 - a_{l,i}) u_i^\delta = \sum_{i=1}^n u_i^\delta - c_l^\delta$.

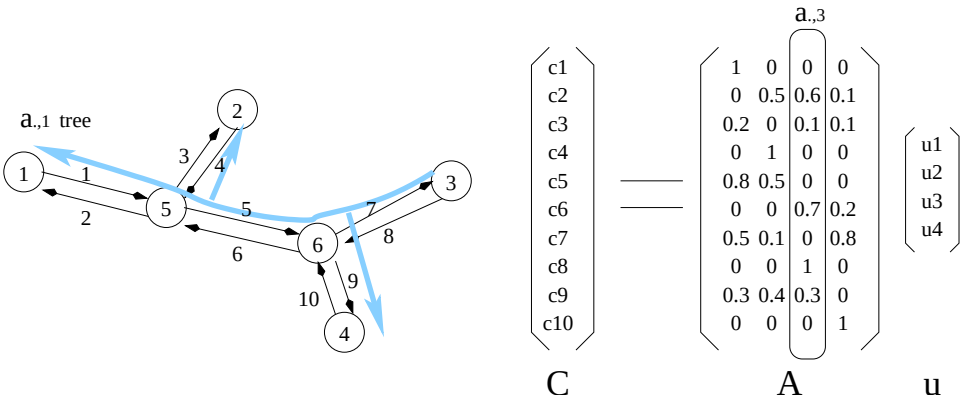


Fig. 5. An Example of a Network Topology and its Traffic Matrix

Therefore, minimizing the branch-penalty is equivalent to minimizing the total bandwidth reduction, that is:

$$\min \sum_{i=1}^{n}(1-a_{l,i})u_i^\delta \Longleftrightarrow \min \sum_{i=1}^{n} u_i^\delta \tag{23}$$

with constraints $0 \le u_i^\delta \le u_i$ and $\sum_{i=1}^{n} a_{l,i}u_i^\delta c_l^\delta$.

Proposition 2: The solution to the minimizing branch-penalty problem comprises three parts:

$$u_{\sigma(1)}^\delta = u_{\sigma(1)}, \cdots, u_{\sigma(k-1)}^\delta = u_{\sigma(k-1)}; \tag{24}$$

$$u_{\sigma(k)}^\delta = \frac{c_l^\delta - \sum_{i=1}^{k-1} a_{l,\sigma(i)}u_{\sigma(i)}}{a_{l,\sigma(k)}}; \text{ and} \tag{25}$$

$$u_{\sigma(k)}^\delta = \cdots = u_{\sigma(n)}^\delta = 0, \tag{26}$$

where $\{\sigma(1),\sigma(2),\cdots,\sigma(n)\}$ is a permutation of $\{1,2,\cdots,n\}$ such that $a_{l,\sigma(i)}$ is sorted in decreasing order, and k is chosen such that:

$$c_{br}(k-1) < c_l^\delta \le c_{br}(k), \tag{27}$$

where $c_{br}(k) = \sum_{i=1}^{k} a_{l,\sigma(i)}u_{\sigma(i)}$.

Proof: A straightforward proof by contradiction can be constructed as follows:

Let's assume that there is another rate reduction vector $\mathbf{v}^\delta \neq \mathbf{u}^\delta$ such that $\mathbf{v}^\delta$ minimizes the objective function (23), that is $\sum_{i=1}^{n} v_i^\delta < \sum_{i=1}^{n} u_i^\delta$. This inequality, together with the fact that $u_{\sigma(i)}^\delta$ $(\forall i < k)$ reaches the maximum possible value, lead to the existence of at least one pair of indices j and m, where $j < k$ and $m \ge k$, such that $a_{l,j} > a_{l,m} > 0$; $v_{\sigma(j)}^\delta < u_{\sigma(j)}^\delta$ and $v_{\sigma(m)}^\delta > u_{\sigma(m)}^\delta$. Now we can construct a third vector $\mathbf{w}^\delta$ as follows: $w_{\sigma(i)}^\delta = v_{\sigma(i)}^\delta$, $i \neq j,m$, $w_{\sigma(j)}^\delta = v_{\sigma(j)}^\delta + \epsilon/a_{l,\sigma(j)}$, and $w_{\sigma(m)}^\delta = v_{\sigma(m)}^\delta - \epsilon/a_{l,\sigma(m)}$. Here $0 < \epsilon < \min\left\{a_{l,\sigma(j)}\left(v_{\sigma(j)} - v_{\sigma(j)}^\delta\right),\ a_{l,\sigma(m)}v_{\sigma(m)}^\delta\right\}$ so that both $w_{\sigma(j)}^\delta$ and $w_{\sigma(m)}^\delta$ are positive. It is clear that $\sum_{i=1}^{n} a_{l,i}w_i^\delta \sum_{i=1}^{n} a_{l,i}v_i^\delta = c_l^\delta$. However, because $a_{l,\sigma(j)} > a_{l,\sigma(m)}$, we have $\sum_{i=1}^{n} w_i^\delta = \sum_{i=1}^{n} v_i^\delta - \epsilon(1/a_{l,\sigma(m)} - 1/a_{l,\sigma(j)}) < \sum_{i=1}^{n} v_i^\delta$. This contradicts the assumption that $\mathbf{v}^\delta$ minimizes the objective function (23). ■

The solution is to sequentially reduce the u_i with the largest $a_{l,i}$ to zero, and then move on to the u_i with the second largest $a_{l,i}$ until the sum of reductions amounts to c_l^δ.

Remark: A variation of the minimal branch-penalty solution is to sort based on $a_{l,\sigma(i)}u_{\sigma(i)}$ rather than $a_{l,\sigma(i)}$. This approach first penalizes the aggregates with the largest volume across the link (i.e., the "elephants"). This solution minimizes the number of traffic aggregates affected by the rate reduction procedure.

3) Penrose-Moore Inverse Reduction: It is clear that equal reduction and minimizing the branch-penalty have conflicting objectives. Equal reduction attempts to provide the same amount of reduction to all traffic aggregates. In contrast, minimal branch-penalty reduction always depletes the bandwidth associated with the traffic aggregate with the largest portion of traffic passing through the congested link. To balance these two competing optimization objectives, we propose a new policy that minimizes the Euclidean distance of the rate reduction vector u^δ:

$$\min\left\{\sum_{i=1}^{n}(u_i^\delta)^2\right\}, \tag{28}$$

with constraints $0 \le u_i^\delta \le u_i$ and $\sum_{i=1}^{n} a_{l,i}u_i^\delta c_l^\delta$.

Similar to the solution of the minimizing variance problem in the equal reduction case, we have:

Proposition 3: The solution to the problem of minimizing the Euclidean distance of the rate reduction vector comprises three parts:

$$\forall i \text{ with } a_{l,i} = 0, \text{ we have } u_i^\delta = 0; \tag{29}$$

then for notation simplicity, we re-number the remaining indices with positive $a_{l,i}$ as $1,2,\cdots,n$; and

$$u_{\sigma(1)}^\delta = u_{\sigma(1)}, \cdots, u_{\sigma(k-1)}^\delta = u_{\sigma(k-1)}; \text{ and} \tag{30}$$

$$\frac{u_{\sigma(k)}^\delta}{a_{l,\sigma(k)}} = \cdots = \frac{u_{\sigma(n)}^\delta}{a_{l,\sigma(n)}} = \frac{c_l^\delta - \sum_{i=1}^{k-1} a_{l,\sigma(i)}u_{\sigma(i)}}{\sum_{i=k}^{n} a_{l,\sigma(i)}^2}, \tag{31}$$

where $\{\sigma(1),\sigma(2),\cdots,\sigma(n)\}$ is a permutation of $\{1,2,\cdots,n\}$ such that $u_{\sigma(i)}/a_{l,\sigma(i)}$ is sorted in increasing order, and k is chosen such that:

$$c_{pm}(k-1) < c_l^\delta \le c_{pm}(k), \tag{32}$$

where $c_{pm}(k) = \sum_{i=1}^{k} a_{l,\sigma(i)}u_{\sigma(i)} + (u_{\sigma(k)}/a_{l,\sigma(k)})\sum_{i=k+1}^{n} a_{l,\sigma(i)}^2$.

```
(1)  sort the indices i of traffic aggregates
     based on :
          the increasing order of u_i for ER,
          the decreasing order of a_{l,i} for BR,
          the increasing order of u_i/a_{l,i} for PM;
(2)  locate the index k in the sorted index list
     based on :
          Ineq. 21 for ER,
          Ineq. 27 for BR,
          Ineq. 32 for PM;
(3)  calculate reduction based on:
          Eq. 18 - Eq. 20 for ER,
          Eq. 24 - Eq. 26 for BR,
          Eq. 29 - Eq. 31 for PM.
```

Fig. 6. Edge Rate Reduction Algorithm Pseudo-Code

Eq. 31 is equivalent to the Penrose-Moore (P-M) matrix inverse [27], in the form of

$$[u^{\delta}_{\sigma(k)}\, u^{\delta}_{\sigma(k+1)} \cdots u^{\delta}_{\sigma(n)}]^T$$

$$[a_{l,\sigma(k)}\, a_{l,\sigma(k+1)} \cdots a_{l,\sigma(n)}]^{+} * (c^{\delta}_{l} - \sum_{i=1}^{k-1} a_{l,\sigma(i)} u_{\sigma(i)}), \quad (33)$$

where $[\cdots]^{+}$ is the P-M matrix inverse. In particular, for an $n \times 1$ vector $\mathbf{a}_{l,\cdot}$, the P-M inverse is a $1 \times n$ vector $\mathbf{a}^{+}_{l,\cdot}$ where $a^{+}_{l,i} = a_{l,i}/(\sum_{i=1}^{n} a^{2}_{l,i})$.

We name this policy as the "P-M inverse reduction" because of the property of P-M matrix inverse. The P-M matrix inverse always exists and is unique, and gives the least Euclidean distance among all possible solution satisfying the optimization constraint.

Proposition 4: The performance of the P-M inverse reduction lies between the equal reduction and minimal branch-penalty reduction. In terms of fairness, it is better than the minimal branch-penalty reduction and in terms of minimizing branch-penalty, it is better than the equal reduction.

Proof: By simple manipulation, the minimization objective of P-M inverse is equivalent to the following:

$$\min \left\{ \sum_{i=1}^{n} \left(u^{\delta}_{i} - (\sum_{i=1}^{n} u^{\delta}_{i})/n \right)^2 + \left(\sum_{i=1}^{n} u^{\delta}_{i} \right)^2 /n \right\}. \quad (34)$$

The first part of this formula is the optimization objective of the equal reduction policy. The second part of formula (34) is scaled from the optimization objective of the minimizing branch penalty policy by squaring and division to be comparable to the objective function of equal reduction; that is, the P-M inverse method minimizes the sum of the objective functions minimized by the equal reduction and minimal branch penalty methods, respectively. Therefore, the P-M inverse policy has a smaller value in the first part of formula (34) than what the minimal branch penalty policy has; and a smaller value in the second part of formula (34) than the corresponding value the equal reduction policy has. Hence, the P-M inverse method balances the trade-off between equal reduction and minimal branch penalty. ■

It is noted that the P-M inverse reduction policy is not the only method that balances the optimization objectives of fairness and minimizing branch penalty. However, we choose it because of its clear geometric meaning (i.e., minimizing the Euclidean distance) and its simple closed-form formula.

4) Algorithm Implementation: The implementation complexity of the preceding three reduction algorithms lies in the boundary conditions where the rates of some traffic aggregates are reduced to zero. Because all three algorithms have similar structure, we can show the procedure of these algorithms in a coherent manner, as shown in Fig. 6.

```
(1)  identify the most loaded link l in the set
     of non-saturated links:
     l = arg min_{j∈L^u} { x_j = (c_j − allocated_capacity) / Σ_{i∈P} a_{j,i} };
(2)  increase allocation to all ingress
     aggregates in P by x_l, and update the
     allocated_capacity for links in L^u;
(3)  remove ingress aggregates passing l from P,
     and remove link l from L^u;
(4)  if P is empty, then stop; else go to (1).
```

Fig. 7. Edge Rate Alignment Algorithm Pseudo-Code

C. Edge Rate Alignment

Unlike edge rate reduction, which is triggered locally by a link scheduler that needs to limit the impact on ingress traffic aggregates, the design goal for the periodic rate alignment algorithm is to re-align the bandwidth distribution across the network for various classes of traffic aggregates and to re-establish the ideal max-min fairness property.

However, we need to extend the max-min fair allocation algorithm given in [28] to reflect the point-to-multipoint topology of a DiffServ traffic aggregate. Let $\mathcal{L}^u$ denote the set of links that are not saturated and $\mathcal{P}$ be the set of ingress aggregates that are *not bottlenecked*, (i.e., have no branch of traffic passing a saturated link). Then the procedure is given as in Fig. 7.

Our modification of step (1) changes the calculation of remaining capacity from $(c_l - allocated\_capacity)/||\mathcal{P}||$ to $(c_l - allocated\_capacity)/\sum_{i\in\mathcal{P}} a_{l,i}$.

Remark: The convergence speed of the max-min allocation for point-to-multipoint traffic aggregates is faster than for point-to-point aggregate because it is more likely that two traffic aggregates send traffic over the same congested link. In the extreme case, when all the traffic aggregates have portions of traffic over all the congested links, these aggregates are only constrained by the most congested bottleneck link. In this case, the algorithm takes one round to finish, and the allocation effect is equivalent to the equal reduction (in this case, "equal allocation") method with respect to the capacity of the most congested bottleneck link.

The edge rate alignment algorithm involves increasing edge bandwidth, which makes the operation fundamentally more difficult than the reduction operation. The problem is essentially the same as that found in multi-class admission control because we need to calculate the amount of offered bandwidth $c_l(i)$ at each link for every service class. Rather than calculate

$c_l(i)$ simultaneously for all the classes, we take a sequential allocation approach. In this case, the algorithm waits for an interval after bandwidth allocation for a higher priority. This allows the lower priority queues to take measurements on the impact of the changes, and to invoke Regulate_Down() if rate reduction is needed. The procedure is on a per class basis and follows the decreasing order of allocation priority.

VI. Simulation Results

A. Simulation Setup

We evaluate our algorithms by simulation using the ns-2 simulator [29]. Unless otherwise stated, we use the default values in the standard ns-2 release for the simulation parameters.

We use the Weighted-Round-Robin scheduler which is a variant of the WFQ algorithm. In our simulation, we consider the performance of four service classes that loosely correspond to the DiffServ Expedited Forwarding (EF), Assured Forward (AF1, and AF2), and best-effort (BE) classes. The order above represents the priority for bandwidth allocation. The initial service weights for the four class queues are 30, 30, 30 and 10, respectively, with a fixed total of 100. The minimum service weight $w_i(\min)$ for each class is 1. The initial buffer size is 30 packets for the EF class queue, 100 packets each of the AF1 and AF2 class queues, respectively, and 200 packets for the BE class queue.

The simulation network comprises eight nodes with traffic conditioners at the edge, as shown in Fig. 8. The backbone links are configured with 6 Mb/s capacity with a propagation delay of 1 ms. The three backbone links (C1, C2 and C3) highlighted in the figure are overloaded in various test cases to represent the focus of our traffic overload study. The access links leading to the congested link have 5 Mb/s with a 0.1 ms propagation delay. The ingress traffic conditioners serve the purpose of ingress edge routers. Each conditioner is configured with one profile for each traffic source. The EF profile has a default peak rate of 500 Kb/s and a bucket size of 10 Kb. The AF profile has a default peak rate of 1 Mb/s and a token bucket of 80 Kb. For simplicity, we program the conditioners to drop packets that are not conforming to the leaky-bucket profile. The core provisioning algorithm will regulate the ingress traffic rates by changing the profiles in the traffic conditioners.

A combination of Constant-Bit-Rate (CBR), Pareto On-Off and Exponential On-Off traffic sources are used in the simulation, as well as applications including a large number of greedy FTP sessions and HTTP transactions. The starting time of all sources is a random variable uniformly distributed in [0 5 s]. During the simulations, we vary the peak rate or the number of sources to simulate different traffic load conditions. Except where specifically noted, we use the default values for all ns simulation parameters.

Throughout the simulations, we use the same set of DiffServ SLAs:

- for the EF class, the delay bound $D_1 = 0.1$ s, the loss bound $P_1^* = 5 * 10^{-5}$;
- for the AF1 class, the delay bound $D_2 = 0.5$ s, the loss bound $P_2^* = 5 * 10^{-4}$;

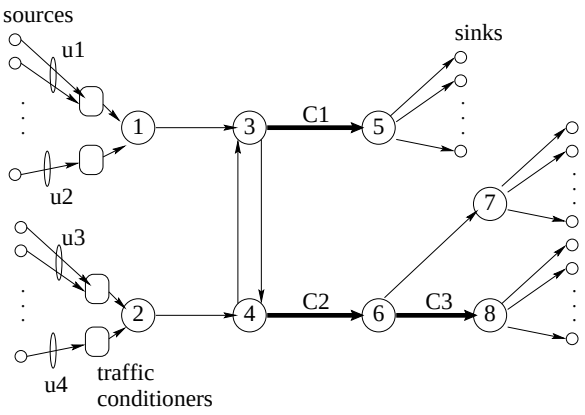


Fig. 8. Simulated Network Topology

- for the AF2 class, the delay bound $D_3 = 1$ s, the loss bound $P_3^* = 5 * 10^{-3}$.

For the BE class, there is no SLA that needs to be supported.

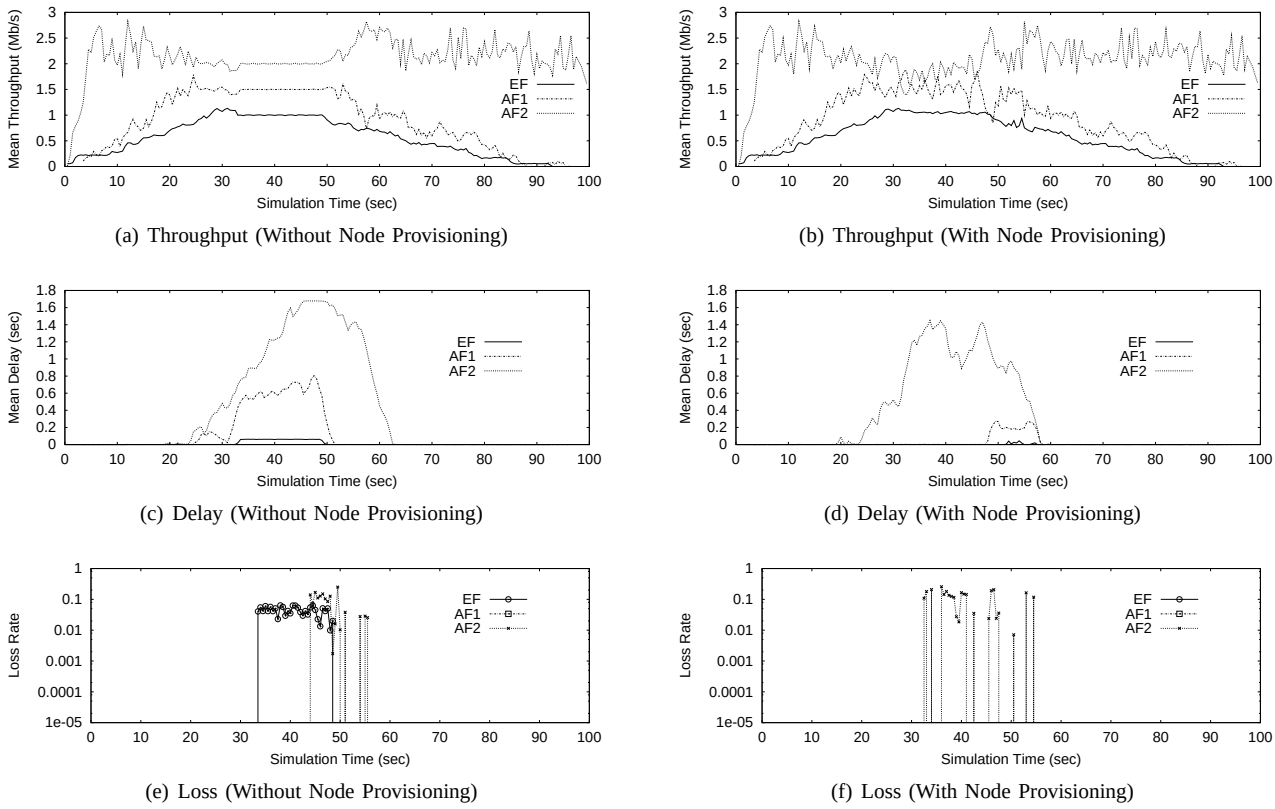
B. Dynamic Node Provisioning

The dynamic node provisioning algorithm interacts with the core provisioning algorithm via the Congestion_Alarm and LinkState_Update signals. To better stress test the node provisioning algorithm, we disable the alarm and update signals to the core provisioning algorithms in the simulations described in this section. In addition, we simplify the network shown in Fig. 8 into a dumb-bell topology (by combining nodes 1 to 4 into one node, and nodes 5 to 8 into another node). The 5 Mb/s link between these two "super" nodes will be the focus of simulations in this sub-section.

1) Service Differentiation Effect: We first use traces to highlight the impact of enabling and disabling the node provisioning algorithm on our service model. We compare the results where the algorithm is enabled and disabled.

We use 100 traffic sources: 20 CBR sources for the EF class; 30 Pareto On-Off sources for the AF1 class; and 40 and 10 Exponential On-Off sources for the AF2 and BE classes, respectively. Each source has the same average rate of 55 Kb/s, which translates into an average of a 110% load on the 5 Mb/s target link when all the sources are active. The simulation trace lasts 100 s. To simulate the dynamics of traffic overload, we activate and stop the EF and AF1 class sources in a slow-start manner, i.e., the activation time for the EF and AF1 traffic sources is uniformly distributed over the first 30 s. The stop time for the EF and AF1 sources is uniformly distributed over the last 40 s. With respect to the AF2 and BE sources, their slow-start activation time lies within the first 5 s, and their stop time is at the end of the simulation period. As a result, congestion occurs between 30 and 60 s in the trace. The node provisioning algorithm $update\_interval$ is set to a value of 200 ms.

Accurately setting the service weights is very important to the operation of the scheduler in the case where the node provisioning algorithm is disabled because its service weights are not adjusted during the simulation. We use the exact information of the traffic load mixture to set the service



(a) Throughput (Without Node Provisioning)

(b) Throughput (With Node Provisioning)

(c) Delay (Without Node Provisioning)

(d) Delay (With Node Provisioning)

(e) Loss (Without Node Provisioning)

(f) Loss (With Node Provisioning)

Fig. 9. Node Provisioning Service Differentiation Effect

weights to 23, 33, 43 and 1 for the EF, AF1, AF2, and BE classes, respectively. These settings yield a traffic intensity of 96%, 100% and 102% for the EF, AF1 and AF2 queues, respectively, while leaving $w_{\min} = 1$ for the BE traffic during the congestion interval. These setting represent the best-case scenario for the scheduler (in the case where the node provisioning algorithm is disabled) to maintain service differentiation for services classes that have SLA concerns. We note that in practice, however, there is no prior knowledge of traffic load during congestion. Therefore, the setting of service weights in practice would be less ideal when comparing the performance of the scheduler in a system where the node provisioning algorithm is disabled. As we will show later, even with such a best-case advantage, the scheduler still underperforms the node provisioning algorithm in both delay and loss performance because a fixed set of service weights can not deal with the varying mixture of traffic loads from different classes.

The statistical traces collected in this simulation are end-to-end throughput, packet loss rate, and mean delay for all the classes except BE. Each sample is averaged over a window of 0.5 s from the per-packet samples.

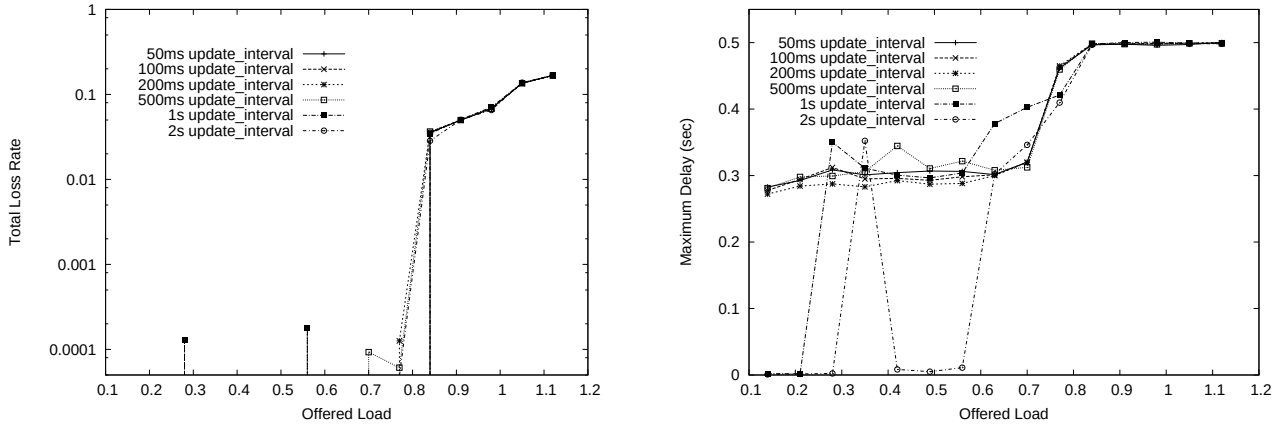
Fig. 9(a) and (b) show the throughput trace. When the system is not overloaded, both plots exhibit the same shape of curve. During congestion between 30 and 60 s into the trace, however, the plot with node provisioning disabled (Fig. 9(a)) shows almost flat throughput curves for the EF, AF1 and AF2 classes, with a ratio of 2:3:4 matching the service weight settings, respectively. In contrast, significant variations occur for the results with the node provisioning algorithm enabled, as shown in Fig. 9(b).

The effect of the node provisioning algorithm can be clearly observed in the delay plots of Fig. 9(c) and (d). Unlike Fig. 9(c) where both AF1 and AF2 delays exceed their bound of 0.5 s and 1 s, respectively, Fig. 9(d) shows that only the AF2 class exceeds its delay bound. In addition, the delay values for all three classes are smaller than the results shown in Fig. 9(c).

In the packet loss comparison, the lack of loss differentiation is clearly evident in Fig. 9(e), where both EF and AF2 classes have the same magnitude of loss rate of approximately 10%. In contrast, in Fig. 9(f) with node provisioning enabled, only AF2 has packet loss and the loss rate is comparable to the result shown in Fig. 9(f).

2) Update Interval: In this set of simulation, we investigate the appropriate value for the $update\_interval$ when invoking the node provisioning algorithm. An $update\_interval$ that is too small, increases the variations in the measured traffic arrival rate and leads to frequent oscillations in bandwidth allocation. In contrast, an $update\_interval$ that is too large, delays the detection of under-load in some traffic classes and hurts service differentiation.

We experiment with five different values of $update\_interval$: 50 ms, 100 ms, 200 ms, 500 ms, 1 s and 2 s. There are a total 70 traffic sources, with 20% for the EF class, 30% for the AF1 class, 40% for the AF2 class and 10% for the BE class. The EF source is CBR with a peak rate of 100 Kb/s. The AF1 and AF2 sources are Pareto On-Off sources with default ns values: an average 0.5 s for the on



(a) Loss Rate (b) Maximum Delay

Fig. 10. Node Provisioning Sensitivity to $update\_interval$, AF1 Class with Pareto On-Off Traffic

and off intervals, and a shape parameter with the value of 1.5. The AF2 sources have a peak rate of 200 Kb/s. The BE class sources are CBR with 100 Kb/s rate. We vary the peak rate for the AF1 class to change the offered load. The offered load is calculated as the ratio between the total arrival rate of the AF1 class and the available bandwidth to AF1 (which is the link capacity subtracted by the total EF traffic arrival rate).

Extensive statistics (e.g., delay, loss, service rate, arrival rate, etc.) are collected for each queueing classes at each network node, and for each flow from end-to-end. Most samples are collected when the node provisioning algorithm is invoked. Therefore, the maximum sampling interval is the $update\_interval$. The collected samples are then consolidated by the time-weighted average for the statistics requiring averaging (e.g., traffic load, mean delay, and loss rate, etc.). Statistics like maximum delay are calculated from the maximum of all the collected samples. The loss rate samples are accumulated using the dual-window approach described in Section IV, with the measurement window τ_l set to 30 s for the EF class and 10 s for all the other classes. The collected samples are then consolidated by time-weighted average for statistics including loss rate, mean delay, and arrival rate.

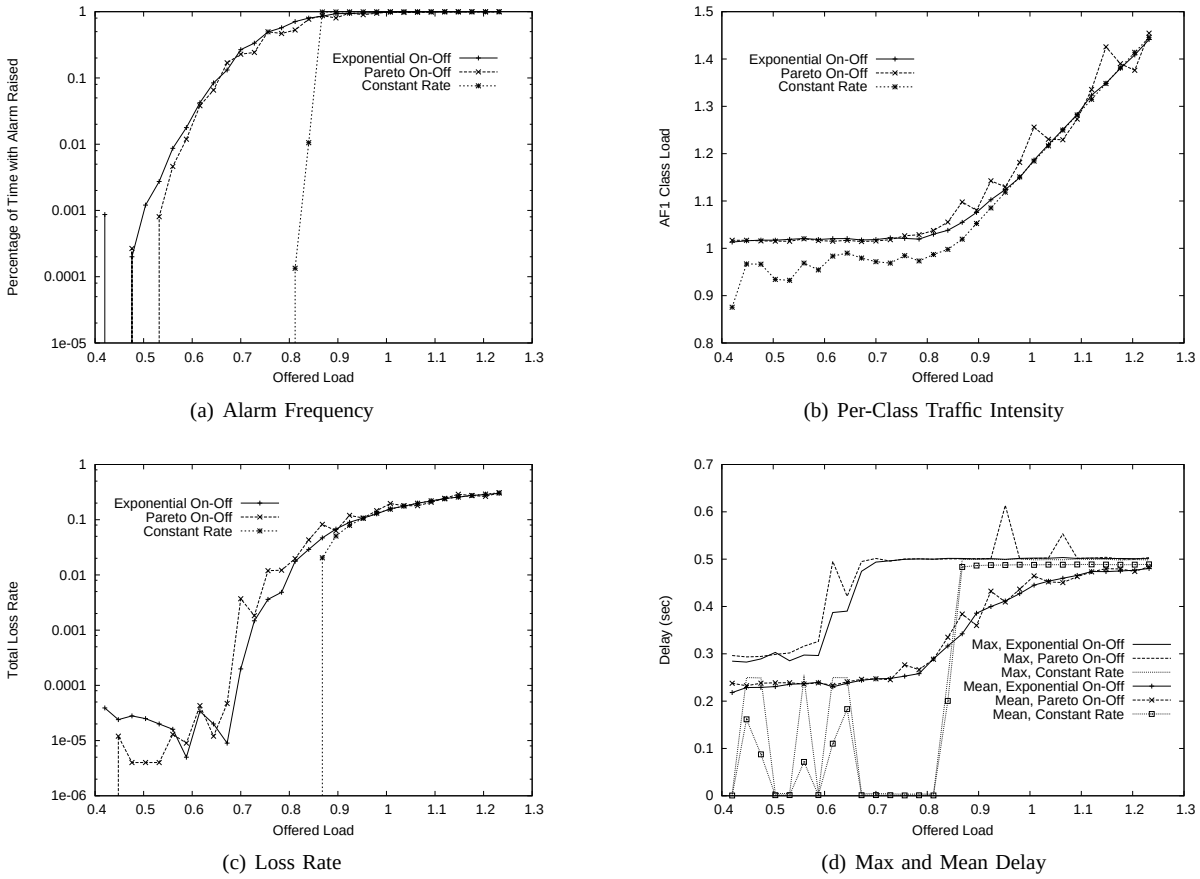
Fig. 10 shows both the packet loss and maximum delay performance. For the purpose of clarity, we only show the results for the AF1 class. Each sample point on the plot is a simulation run of 100 s. In general, the algorithm performance is not very sensitive to the value of the $update\_interval$. This is expected because the node provisioning algorithm can also be invoked by the virtual queues detecting an onset of SLA violation. Among the small differences, we observe that the $update\_interval \geq 1$ s is not good because it has packet losses and large variation of the maximum delay under low offered load. In addition, we observe that an $update\_interval$ value of 200 ms achieves low maximum delay relative to the other curves. This is consistently observed across the whole range of offered loads below 80%. When the offered load increases beyond 80% the system becomes over-loaded and the impact of a different $update\_interval$ becomes negligible. In what follows, we will use an $update\_interval = 200$ ms for all the simulations.

It is also interesting to observe one feature of the node provisioning algorithm: namely the algorithm always tries to guarantee the delay bound first. We observe that beyond 80% load the loss rate starts to exceed the $5*10^{-4}$ bound, while the delay bound of 0.5 s is always maintained even for an offered load exceeding 1.

3) Stress Test Under Bursty Traffic: We continue the preceding simulation runs with different traffic sources for the AF1 classes, including Pareto On-Off, Exponential On-Off and CBR traffic sources. Each sample point represents a simulation run of 1000 s. We use the CBR traffic source to provide a baseline reference for the two bursty On-Off traffic types.

Fig. 11 presents four sets of consolidated statistics for comparison. Fig. 11(a) plots the percentage of time that the Congestion_Alarm is raised for the AF1 class. Since we disable the dynamic core provisioning algorithm to stress test the node algorithm, the alarm frequency becomes a good indicator of the node algorithm's capability of handling bursty traffic. It is also a convenient indicator of the performance boundary below which the delay bound $D_2 = 0.5$ s and loss bound $P^*_{loss,2} = 5 * 10^{-4}$ should hold and above which the loss rate and maximum delay will grow to exceed these bounds. We observe that the algorithm performs equally well for both Pareto and Exponential On-Off sources, even though the Pareto source is heavy-tailed and more bursty. It is clear that the algorithm can handle up to 70% load for both the Pareto and Exponential On-Off traffic under the D_2 and $P^*_{loss,2}$ bounds. For the CBR traffic, the sustainable load reaches 85% as observed from the loss and delay measurements in Fig. 11(c) and 11(d), respectively. This falls short of 100% because the CBR traffic is also bursty being an aggregate of 21 individual CBR sources.

Fig. 11(b) shows the measured traffic intensity in the AF1 queue. Even though measuring the arrival rate is trivial, measuring the per-class service time is not easy for a multi-class queueing system. In the simulations, we use the sum of the per-packet transmission time and the Head-of-the-Line



(a) Alarm Frequency

(b) Per-Class Traffic Intensity

(c) Loss Rate

(d) Max and Mean Delay

Fig. 11. Node Provisioning Algorithm Performance, AF1 Class with Bursty Traffic Load

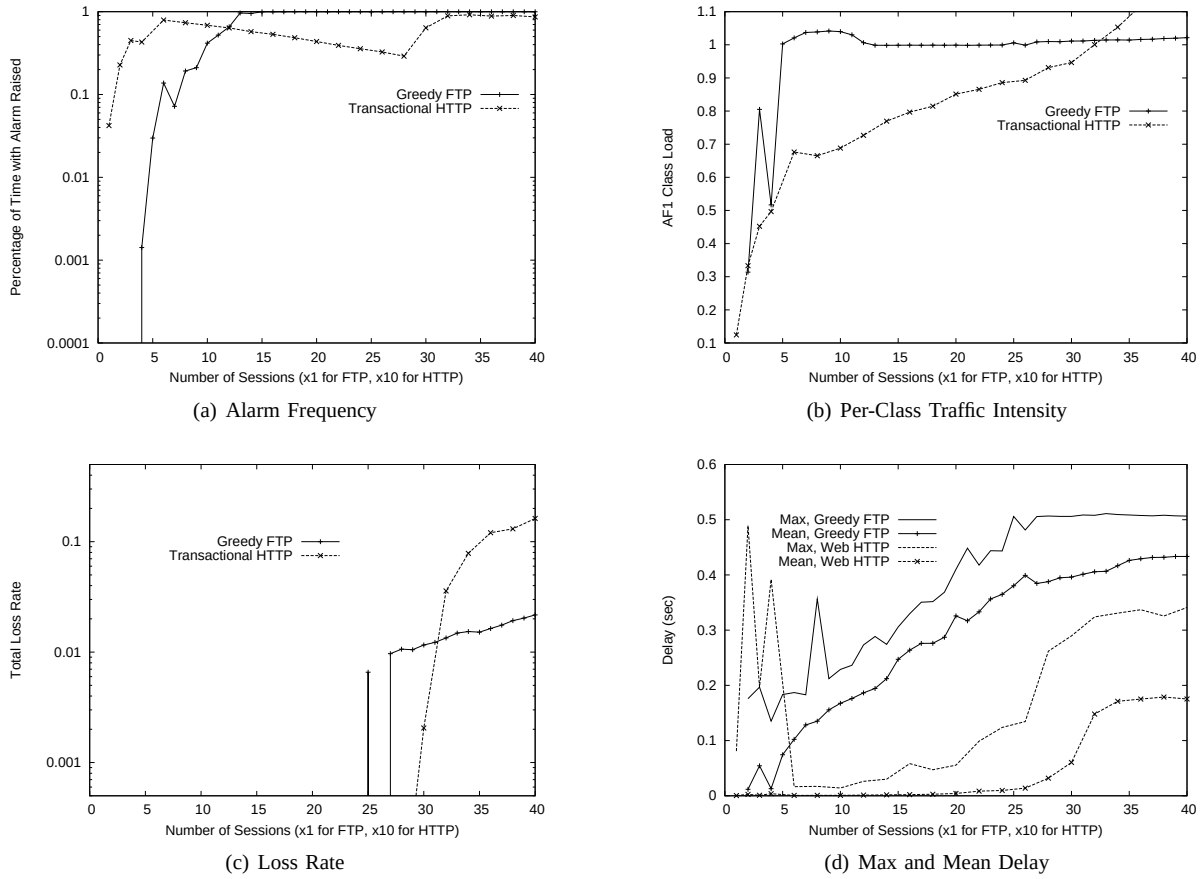
(HOL) waiting time as the total service time. The HOL waiting time is the time after a packet enters the HOL position of the queue, waiting for scheduler to finish serving the HOL packets of other queues. From this plot we can observe the algorithm's efficiency in allocating bandwidth. For the CBR traffic, the service bandwidth utilization remains at 100% until the incoming traffic exceeds the maximum service capability. For the Pareto and Exponential On-Off traffic, the utilization stays at 100% until the offered load reaches 50%. After that the utilization dips by about 10%. This is the amount of over allocation necessary to maintain the SLA.

Fig. 11(c) and 11(d) plot the loss rate and maximum delay measured at this AF1 class queue, respectively. The results verify that when the alarm signal is not raised, the system performance will remain below the SLA bounds. Once again we observe that the algorithm gives precedence in guaranteeing the delay bound first. Except two spikes for the Pareto source, all the maximum delay curves are below the 0.5 s bound. In addition, one can also discover the fact that only when the alarm frequencies exceed 10%, the loss rate will exceed the loss bound of $5 * 10^{-4}$. This is true for both the Pareto and Exponential On-Off sources, where the 10% alarm frequency corresponds well to the 70% maximum sustainable load, and for the CBR source, where the 10% alarm frequency matches the 85% maximum sustainable load. This information is important for the core provisioning algorithm as it allows the core algorithm to gauge the overload severity from the frequency of Congestion_Alarm signals sent by the node provisioning algorithms.

4) Scalability with Adaptive Applications: We further test our scheme with TCP applications including greedy FTP and transactional HTTP applications. Because TCP congestion control reacts to packet loss, the packet dropping action alone is also effective in reducing congestion for TCP. However, the adaptive flow control of TCP also will push the traffic load to 100% even with a small number of sources. To test our algorithm's performance in supporting a large number of TCP sources, we repeat the above test but instead of varying the peak rate of each source, we vary the number of TCP applications that are connected to the target node.

The results are shown in Fig. 12 in the same setting as Fig. 11. The traffic load for the EF, AF2 and BE classes remain the same as in the previous tests. We vary the number of the AF1 sessions: from 2 to 40 for greedy FTP traffic, and from 20 to 400 for web traffic. To better understand these results, we plot the FTP and HTTP results with a corresponding 1:10 ratio in the number of sessions on the x axis.

The web traffic is simulated using the ns-2 "PagePoolWebTraf" module. The parameters for the web traffic are set to increase the traffic volume of each web session so that on the target link of 5 Mb/s, queueing overload can occur. The inter-session time is exponentially distributed with a mean of



(a) Alarm Frequency

(b) Per-Class Traffic Intensity

(c) Loss Rate

(d) Max and Mean Delay

Fig. 12. Node Provisioning Algorithm Performance, AF1 Class with TCP Applications

0.1 s. Each session size is a constant of 100 pages. The inter-page time is also exponentially distributed but with a mean of 5 s. Each page size is a constant of 5 objects, while the inter-object time is exponentially distributed with a mean of 0.05 s. Last, the object size has a distribution of Pareto of the Second Kind (also know as the Lomax distribution) with a shape value of 1.2 and average size of 12 packets (which is 12 KB).

In Fig. 12(a), for both traffic sources, the alarm frequency rises above 10% for a small number of sessions, i.e., 5 sessions for FTP and 20 sessions for HTTP, respectively. The average traffic intensity (Fig. 12(b)), however, shows a difference. The FTP traffic intensity increases quickly to 100% and then stays at 100% after 5 sessions, while the HTTP traffic intensity increases gradually and reaches 100% much later at 220 sessions. These two plots indicate that the HTTP traffic is more bursty than the FTP traffic because for the HTTP traffic, its alarm frequency rises quicker while its average traffic intensity rises much slower than the FTP traffic. The FTP traffic, on the other hand, is less bursty only because its average load reaches 100% for most of the cases. However, even with a large value of alarm frequency, the system perform well for a wide range of number of sessions. The loss rate exceeds $5 * 10^{-4}$ at 25 FTP sessions or 300 HTTP sessions. The delay bound of 0.5 s is always met for the HTTP traffic. For the FTP traffic, because of the heavy traffic load, the delay bound is first violated at 25 FTP sessions, but is not exceeded much after that point (Fig. 12(d)).

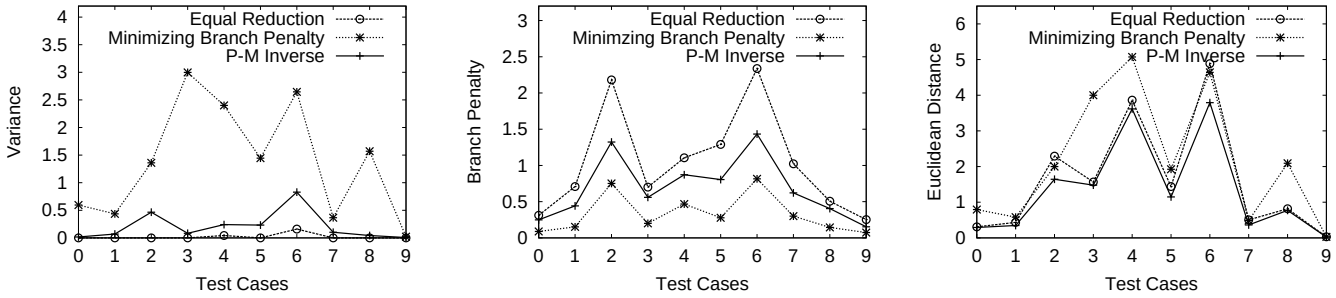
In summary, the stress test results from both bursty On-Off and TCP application traffic have shown that the node provisioning algorithm will guarantee the delay and loss bounds when there is no alarm raised, and also with a alarm frequency below 10%. When there is a SLA violation, the algorithm will first meet the delay bound sacrificing the loss bound. For adaptive applications like TCP which respond to packet loss, this approach has shown to be effective even without the involvement of core provisioning algorithms.

C. Dynamic Core Provisioning

1) Effect of Rate Control Policy: In this section, we use test scenarios to verify the effect of different rate control policies in our core provisioning algorithm. We only use CBR traffic sources in the following tests to focus on the effect of these policies.

Table I gives the initial traffic distribution of the four EF aggregates comprising only CBR flows in the simulation network, as shown in Fig. 8. For clarity, we only show the distribution over the three highlighted links (C1, C2 and C3). The first three data-rows form the traffic load matrix $\mathbf{A}$, and the last data-row is the input vector $\mathbf{u}$.

In Fig. 13, we compare the metrics for equal reduction, minimal branch-penalty and the P-M inverse reduction under



(a) Variance of Reductions (b) Branch Penalty of Reductions (c) Euclidean Distance of Reductions

Fig. 13. Reduction Policy Comparison (Ten Independent Tests)

ten randomly generated test cases. Each test case starts with the same initial load condition, as given in Table I. The change is introduced by reducing the capacity of one backbone link to cause congestion which subsequently triggers rate reduction.

Fig. 13(a) shows the fairness metric: the variance of rate reduction vector $\mathbf{u}^\delta$. The equal reduction policy always generates the smallest variance, in most of the cases the variances are zero, and the non-zero variance cases are caused by the boundary conditions where some of the traffic aggregates have their rates reduced to zero. Here we observe that the P-M inverse method always gives a variance value between those of equal reduction and minimizing the branch penalty. Similarly, Fig. 13(b) illustrates the branch penalty metric: $\sum_i (1-a_{l,i})u_i^\delta$. In this case, the minimizing branch penalty method consistently has the lowest branch penalty values, followed by the P-M inverse method. The last figure, Fig. 13(c), shows the Euclidean distance of $\mathbf{u}^\delta$, i.e., $\sum_i (u_i^\delta)^2$. In this case, the P-M inverse method always has the lowest values, while there is no clear winner between the equal reduction and minimizing branch penalty methods.

The results support our assertion that the P-M Inverse method balances the trade-off between equal reduction and minimal branch penalty.

In Fig. 14, we plot the time sequence of rate-regulating results using the default policies of our core provisioning algorithm, i.e., the P-M inverse method for rate reduction and the modified max-min fair rate alignment method for rate realignment. The traffic dynamics are introduced by sequentially changing link capacity of C_1, C_2 and C_3 as follows:

1) at 100 s into the trace, C_2 capacity is reduced to 3 Mb/s and requires a bandwidth reduction of 0.8 Mb/s from ingress traffic conditioners
2) at 200 s into the trace, C_3 capacity is reduced to 2 Mb/s, and requires a bandwidth reduction of 0.1 Mb/s,
3) at 300 s into the trace, C_1 capacity is reduced to 0.5 Mb/s, and requires a bandwidth reduction of 0.6 Mb/s, and
4) at 400 s into the trace, C_1 notices a capacity increase to 6 Mb/s, which leaves C_3 the only bottleneck.

TABLE I
TRAFFIC DISTRIBUTION MATRIX

Bottleneck Link	User Traffic Aggregates			
	U_1	U_2	U_3	U_4
C_1	0.20	0.25	0.57	0.10
C_2	0.80	0.75	0.43	0.90
C_3	0.40	0.50	0.15	0.80
Load (Mb/s)	1.0	0.8	1.4	2.0

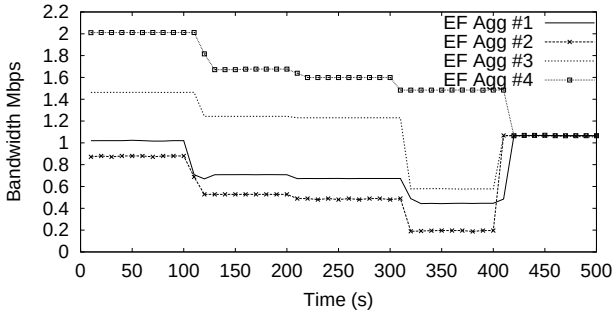


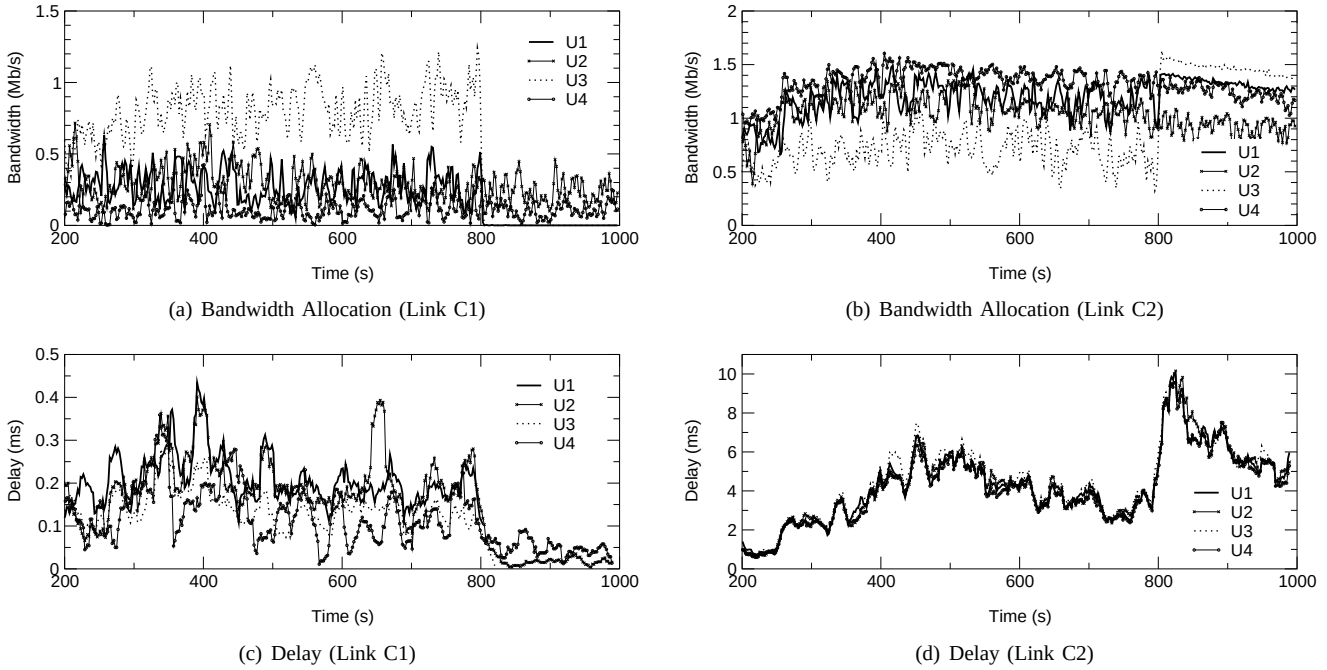
Fig. 14. Core Provisioning Allocation Result, Default Policies

The first three cases of reduction are also the first three test cases used in Fig. 13[1]. The last case invokes a bandwidth increment rather than a reduction. In this case, we use the modified max-min fair allocation algorithm to re-align the bandwidth allocation of all ingress aggregates. The allocation effect is the same as "equal allocation" because all the traffic aggregates share all the congested links.

2) Responsiveness to Network Dynamics: We use a combination of CBR and FTP sources to study the joint effect of our dynamic core provisioning algorithm (i.e., the P-M Inverse method for rate reduction and max-min fair for rate alignment) and our node provisioning algorithm. Periodic edge rate alignment is invoked every 60 s. We use CBR and FTP sources for EF and AF1 traffic aggregates, respectively. Each traffic class comprises four traffic aggregates entering the network in the same manner, as shown in Fig. 8. A large number (50) of FTP sessions are used in each AF1 aggregate to simulate a continuously bursty traffic demand. The distribution of the AF1 traffic across the network is the same as shown in Table I.

The number of CBR flows in each aggregate varies to

[1]We note that it does not make sense to plot the performance metrics shown in Fig. 13 in the same time sequence style as that of Fig. 14. The reason is that in a time-sequenced test, after the first test case, the load conditions prior to each rate reduction could be different for different allocation methods, and the results from the comparison metrics would not be comparable.



(a) Bandwidth Allocation (Link C1)

(b) Bandwidth Allocation (Link C2)

(c) Delay (Link C1)

(d) Delay (Link C2)

Fig. 15. Average Bandwidth Allocation and Delay Traces for AF1 Aggregates

simulate the effect of varying bandwidth availability for the AF1 class (which could be caused in reality by changes in traffic load, route, and/or network topology). The changes in the available bandwidth for the AF1 class includes: at time 400 s into the trace, $C2$ (the available bandwidth at link 2) is reduced to 2 Mb/s; at 500 s into the trace, $C3$ is reduced to 0.5 Mb/s; and at 700 s into the trace, $C3$ is increased to 3 Mb/s. In addition, at time 800 s into the trace, we simulate the effect of a route change, specifically, all packets from traffic aggregate $u1$ and $u3$ to node 5 are rerouted to node 8, while the routing for $u2$ and $u4$ remains intact.

Fig. 15 illustrates the allocation and delay results for the four AF1 aggregates. We observe that not every injected change of bandwidth availability triggers an edge rate reduction; however, in such a case it does cause changes in packet delay. Since the measured delay is within the performance bound, the node provisioning algorithm does not generate *Congestion_Alarm* signals to the core provisioning module, hence, rate reduction is not invoked. In most cases, edge rate alignment does not take effect either because the node provisioning algorithm does not report the need for an edge rate increase. Both phenomena demonstrate the robustness of our control system.

The system correctly responds to route changes because the core provisioning algorithm continuously measures the traffic load matrix. As shown in Fig. 15(a) and 15(b), after time 800 s into the trace, the allocation of $u1$ and $u3$ at link C1 drops to zero, while the corresponding allocation at link C2 increases to accommodate the surging traffic demand.

VII. Conclusion

This paper makes two contributions. First, our node provisioning algorithm prevents transient service level violations by dynamically adjusting the service weights of a weighted fair queueing scheduler. The algorithm is measurement-based and effectively uses a multi-class virtual queue technique to predict the onset of SLA violations. Second, our core provisioning algorithm is designed to address the unique difficulty of provisioning DiffServ traffic aggregates where rate-control can only be exerted at the root of traffic distribution trees. We proposed the Penrose-Moore (P-M) Inverse Method for edge rate reduction which balances the trade-off between fairness and minimizing the branch-penalty. We also extended max-min fair allocation for edge rate alignment and demonstrated its convergence property.

Collectively, these algorithms contribute toward a more quantitative differentiated service Internet, supporting per-class delay guarantees with differentiated loss bounds across core IP networks. We have argued that such an approach to dynamic provisioning is superior to static provisioning for DiffServ because it affords network mechanisms the flexibility to regulate edge traffic, maintaining service differentiation under persistent congestion and device failure conditions when observed in core networks.

Our service model uses two priority orders in QoS provisioning; that is, the relaxation of the loss bound in favor of a delay bound, and a static order of relaxation among service classes. The preference of a delay bound instead of a loss bound is intended to better support TCP applications with reduced round trip delays and early congestion notification through packet drops. However, under high loss rate conditions, low-priority flows would be starved due to the interaction of a high loss rate and TCP congestion control algorithms. Therefore, it is important for the core provisioning algorithm to prevent severe congestion from happening by regulating traffic at the edges. In this paper, we have shown that the

node provisioning algorithm can provide reliable early warning signals using a virtual queue technique, which does not require prior knowledge of traffic characteristics. We are currently studying how to extend the core provisioning algorithm to also provide loss guarantees across traffic classes. This problem bears similarity to the measurement based admission control algorithms discussed in the related work section (Section II).

The complexity of the proposed algorithms mainly resides in the node provisioning algorithm, which is distributed across core routers and is scalable to large network configurations. The challenge of implementing the centralized core provisioning algorithm lies in the continuous monitoring of the traffic matrix across the core network. To improve scalability, we a studying approaches that can enlarge the monitoring granularity and time scale; for example, focusing on a few potential bottleneck links instead of every internal link in the network, or, increasing the $update\_interval$ provisioning time scale. Recent work on network measurement [11], [12], [25] of the AT&T backbone network provides valuable insights and directions on how we could scale the monitoring process up to handle large networks. The centralized approach to the current design of the core provisioning algorithm provides a better response time to sudden changes in network traffic overloads. To improve survivability against network failures (e.g., outages, DDoS), fault tolerant practices in network management can be used to deploy redundant core provisioning algorithms. We are currently studying a fully distributed core provisioning algorithm that removes the single point of failure presented by the existing centralized scheme. A key challenge is to design and analyze the convergence and stability properties of a distributed solution in order to recover in a timely fashion after network failure. We plan to develop some form of analytical proof or argument guaranteeing the stability of such a scheme subject to the perturbations unbounded in magnitude, but bounded in time.

ACKNOWLEDGMENTS

This work is funded in part by a grant from the Intel Corporation for "Signaling Engine for Programmable IXA Networks". We would also like to thank the anonymous reviewers and the editor for their constructive comments to improve the quality of this paper.

REFERENCES

[1] S. B. et. al., "An Architecture for Differentiated Services," IETF RFC 2475, December 1998, http://www.ietf.org/rfc/rfc2475.txt.

[2] B. Teitelbaum, S. Hares, L. Dunn, R. Neilson, V. Narayan, and F. Reichmeyer, "Internet2 QBone: Building a Testbed for Differentiated Services," *IEEE Network Mag.*, pp. 8–16, September/October 1999.

[3] R. Rajan, D. Verma, S. Kamat, E. Felstaine, and S. Herzog, "A Policy Framework for Integrated and Differentiated Services in the Internet," *IEEE Network Mag.*, pp. 36–41, September/October 1999.

[4] R. J. Gibbens and F. P. Kelly, "Distributed Connection Acceptance Control for a Connectionless Network," in *Proc. IEE Int'l Teletraffic Congress (ITC)*. Edinburgh, UK: Elsevier Science Publishers B.V., June 1999.

[5] F. P. Kelly, P. B. Key, and S. Zachary, "Distributed Admission Control," *IEEE J. Select. Areas Commun.*, vol. 18, no. 12, pp. 2617–2628, December 2000, special Issue on QOS in the Internet.

[6] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks," in *Proc. Annual Conference of the ACM Special Interest Group on Data Commun. (SIGCOMM)*, September 1998.

[7] I. Stoica and H. Zhang, "Providing Guaranteed Services Without Per Flow Management," in *Proc. Annual Conference of the ACM Special Interest Group on Data Commun. (SIGCOMM)*, September 1999.

[8] D. Mitra, J. Morrison, and K. G. Ramakrishnan, "Virtual Private Networks: Joint Resource Allocation and Routing Design," in *Proc. IEEE Annual Conference on Computer Commun. (INFOCOM)*, New York City, March 1999.

[9] F. P. Kelly, "Routing in Circuit-switched Networks: Optimization, Shadow Price and Decentralization," *Adv. Appl. Prob.*, vol. 20, pp. 112–144, 1988.

[10] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "NetScope: Traffic Engineering for IP Networks," *IEEE Network Mag.*, March/April 2000.

[11] ——, "Deriving Traffic Demands for Operational IP Networks: Methodology and Expeience," in *Proc. Annual Conference of the ACM Special Interest Group on Data Commun. (SIGCOMM)*, September 2000.

[12] N. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe, "A Flexible Model for Resource Management in Virtual Private Networks," in *Proc. Annual Conference of the ACM Special Interest Group on Data Commun. (SIGCOMM)*, September 1999.

[13] C. Cetinkaya and E. Knightly, "Egress Admission Control," in *Proc. IEEE Annual Conference on Computer Commun. (INFOCOM)*, Tel Aviv, Israel, March 2000.

[14] L. Breslau, E. Knightly, S. Shenker, I. Stoica, and H. Zhang, "Endpoint Admission Control: Architectural Issues and Perormance," in *Proc. Annual Conference of the ACM Special Interest Group on Data Commun. (SIGCOMM)*, Stockholm, Sweden, September 2000.

[15] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," in *Proc. Annual Conference of the ACM Special Interest Group on Data Commun. (SIGCOMM)*, September 1999.

[16] J. Liebeherr and N. Christin, "JoBS: Joint Buffer Management and Scheduling for Differentiated Services," in *Proc. IEEE/IFIP Int'l Workshop on Quality of Service (IWQOS)*, Karlsruhe, Germany, June 2001.

[17] P. Hurley, M. Kara, J.-Y. L. Boudec, and P. Thiran, "A Novel Scheduler for a Low Delay Service within Best-Effort," in *Proc. IEEE/IFIP Int'l Workshop on Quality of Service (IWQOS)*, Karlsruhe, Germany, June 2001.

[18] Z. Zhang, Z. Duan, L. Gao, and Y. Hou, "Decoupling QOS Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services," in *Proc. Annual Conference of the ACM Special Interest Group on Data Commun. (SIGCOMM)*, September 2000.

[19] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching (MPLS) Architecture," IETF RFC 3031, January 2001.

[20] E. Bouillet, D. Mitra, and K. G. Ramakrishnan, "Design-Assisted, Real Time, Measurement-Based Network Controls for Management of Service Level Agreements," in *Proceedings of EURANDOM Workshop on Stochastics of Integrated-Services Comm. Networks*, Eindhoven, The Netherlands, November 15-19 1999.

[21] C. Albuquerque, B. J. Vickers, and T. Suda, "Network Border Patrol," in *Proc. IEEE Annual Conference on Computer Commun. (INFOCOM)*, Tel Aviv, Israel, March 2000.

[22] R. R.-F. Liao and A. T. Campbell, "Dynamic Edge Provisioning for Core Networks," in *Proc. IEEE/IFIP Int'l Workshop on Quality of Service (IWQOS)*, Pittsburgh, USA, June 2000.

[23] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Reading, Mass.: Addison-Wesley, 1997.

[24] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *ACM Comput. Commun. Review*, vol. 27, 1997.

[25] N. G. Duffield and M. Grossglauser, "Trajectory Sampling for Direct Traffic Observation," in *Proc. Annual Conference of the ACM Special Interest Group on Data Commun. (SIGCOMM)*, September 2000.

[26] H. W. Kuhn and A. W. Tucker, "Non-linear Programming," in *Proc. 2^{nd} Berkeley Symp. on Mathematical Statistics and Probability*. Univ. Calif. Press, 1951, pp. 481–492.

[27] S. Campbell and C. M. Jr, *Generalized Inverses of Linear Transformations*. London, UK: Pitman, 1979.

[28] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[29] UCB/LBNL/VINT, "Network Simulator - ns, DiffServ Module," www.isi.edu/nsnam/ns/ns-contributed.html.

Raymond R.-F. Liao is currently working on technology transfer of part of his Ph.D. research at Siemens TTB in Berkeley, CA. Raymond received his Ph.D. from Columbia University in Dec. 2002. His thesis won the Eliah I. Jury award from the Electrical Engineering Department. He received the M.A.Sc. degree in 1993 and the B.E. degree in 1990. Between 1993 and 1996, Raymond worked at Newbridge Networks Corp. on ATM network traffic management. Raymond's research interests include performance analysis, network adaptive services and incentive engineering. He has (co)authored two dozen publications and several patents in these areas.

Andrew T. Campbell is an Associate Professor of Electrical Engineering at Columbia University and a member of the COMET Group. He is working on emerging architectures and programmability for wireless networks. Andrew received his Ph.D. in Computer Science in 1996 and the NSF CAREER Award for his research in programmable mobile networking in 1999.