Self-Tuning End-to-end QoS Internet

Zvi Rosberg

CSIRO ICT Centre, Sydney, NSW, Australia. Email: zvi.rosberg@csiro.au

***Abstract*—A novel QoS architecture along with scalable protocols for rate allocation in multi-service Internet is introduced. The protocols compute distributively, adaptively and asynchronously the target rates for aggregate flows and shape their traffic accordingly so as to meet multiple QoS requirements for packet delay, packet loss rate and minimum bandwidth. The target rates are optimal in sense that they are proportionally fair with regard to all rates satisfying the multi-service QoS requirements. The protocols also comprise a rule for a safe connection admission control. Protocols stability and effectiveness are demonstrated by a simulated test-bed of a large ISP network.**

I. Introduction

Today's Internet relies on the Transmission Control Protocol (TCP) to control congestion which could be satisfactory assuming: (1) proper bandwidth dimensioning; (2) data services use TCP while real-time services (e.g. streaming applications) use User Datagram Protocol (UDP); and (3) the wide majority of traffic remains TCP. Under this assumptions, TCP packets, which adapt their rate to congestion, can wait allowing the real-time UDP services, which do not adapt their rate to congestion, achieve acceptable quality. However, these assumptions are likely not to hold in the future.

Indeed, proper dimensioning is difficult given the bursty nature of Internet traffic and its unpredictable growth. The proportion of streaming applications increases and most of them "tunnel" their UDP datagrams through the HTTP open port (port 80) to bypass firewalls. Consequently, they receive best-effort service rather than appropriate quality of service (QoS). This necessitates a QoS solution at the IP packet level, regardless of the flow upper layer protocol (TCP or UDP). It also requires a connection admission control (CAC) for bandwidth guarantee. Current QoS provisions of DiffServ [2] and IntServ [3] have their drawbacks. Unfortunately, IntServ is not scalable and its QoS guarantee is limited to domain boundaries [16]. DiffServ, which is scalable, guarantees just per-hop behavior (PHB) and requires complex configuration that could be derived only by a network provisioning tool.

In recent years, fluid flow network models have been used to characterize fairness, stability and convergence of various flow control algorithms in large networks [4] [5] [6] [7] [8] [9] [10] [14]. The general approach of these fluid flow network models is to consider a constrained optimization problem of flow rates maximizing a separable network utility function subject to all link capacity constraints. Those models and the associated derived algorithms cannot provide a comprehensive solution for rate allocation in a multi-service Internet due to the following shortcomings.

A major shortcoming is that these flow controls are restricted to best-effort flows not addressing QoS requirements of multi-service flows running in current Internet.

Another shortcoming is that even for best-effort flows, fairness is not attained in practice. Indeed, streaming applications use configured rate-based flow controls and TCP use one of the proposed fair window-based flow controls (*max-min fair*, *proportional fair* [5] or $(\boldsymbol{p}, \boldsymbol{\alpha})$*-proportional fair* [10]). However, the fairness of these flow controls is guaranteed only if the non-TCP applications are controlled with the same mechanism, or if they are "TCP-friendly" [15]. Otherwise, the constraints of the optimization problem determining the fair rates are incomplete. Consequently, the resulting TCP connection rates are not optimal and therefore unfair.

Yet another shortcoming is that these flow controls lack a safe congestion avoidance mechanism. Marking-based congestion control proposed in [6] [8] and the rate-based control of [5] are exogenous and not well commanded by TCP flow control. They are based on rate estimation at the routers and affects the sources only in a probabilistic manner. As an exogenous mechanism it violates fairness since its rate reduction effect is not incorporated into the problem constraints.

This paper addresses the shortcomings above and derives a scalable protocol for end-to-end QoS guarantee with delay, packet loss and bandwidth requirements. Moreover, the protocol also achieve fairness. The paper extends the results of [13] where only delay requirements have been addressed. It also adds an adaptive mechanism for coping with traffic changes.

Flow control can be implemented in different OSI layers, e.g., in IP network layer or in various transport layers such as TCP, ATM and MPLS. Also, flow control can run in routers or in end user hosts. Since flow control aims at addressing overall network performance, it is advocated to implement it at the backbone network layer and at the routers. Additionally, for better scalability it is also advocated to control aggregate flows rather than individual application flows. Aggregate flow control also diminish the inefficiency of dealing with short-live connections. Furthermore, since TCP and UDP sockets are being used by virtually all applications, flow control at the IP network layer is preferred over flow control at a transport layer (i.e., in TCP, ATM or MPLS) since it provides a generic solution applicable across all networks and domains.

It worth noting that implementing a network layer flow control for multi-service flows will limit TCP flow control to the access links and access networks only. The main contributions of this paper are: (i) extension of fairness to multi-service flows

with multiple QoS requirements comprising packet delay, packet loss rates and bandwidth (Section II); and (ii) an adaptive scalable and converging procedure guaranteeing all QoS requirements along with $(\boldsymbol{p},\boldsymbol{\alpha})$-proportional fair rates (Section III). The stability and the effectiveness of the proposed solution is demonstrated by a test-bed simulation of an ISP network (Section IV).

II. Fairness for All

QoS requirements include maximum end-to-end packet delay, maximum packet loss rate and minimum rate allocation. Priority packet scheduling in current routers provide predictable performance means for controlling packet delay and the packet loss rate. For minimum rate guarantee, a reliable reservation protocol and a CAC protocol are also required.

The proposed architecture for QoS guarantee comprises: (i) packet classification at the edge routers of the Internet service providers (ISPs) along with marking in the *type of service* IP header field; (ii) CAC and anonymous bandwidth reservation; (iii) a rate management protocol along with network probing; (iv) prioritized packet scheduling at all routers; and (v) adaptive traffic shaping only at the edge routers.

Packet classification and marking are standard tasks in the current Internet infrastructure. Anonymous bandwidth reservation refers to a scalable reservation protocol not requiring connection identities. Rate management and network probing refer to a network layer protocol for computing target rates of aggregate flows and estimating their packet delays and packet losses. Prioritized packet scheduling is a standard scheduler existing in all modern routers. Finally, adaptive traffic shaping is an extension of the current token bucket shaping used by modern routers. The extension copes with the case where token generation rates are adapted to target rates which vary in time. The target rates are received from the rate management and network probing protocol jointly referred to as the *rate management protocol (RMP)*.

The RMP in this paper extends the RMP introduced in [13] by including an anonymous bandwidth reservation procedure, minimum bandwidth guarantee and network probing. The new RMP along with an adaptive token bucket shaper guarantee end-to-end QoS.

A. Multi-service $(\boldsymbol{p},\alpha)$-proportional fairness

Given a network topology, let $\mathcal{N}$ be the set of its unidirectional links labeled $1,\ldots,N$; and $\mathcal{I}$ be the set of one-way flows labeled $1,\ldots,I$. Each flow, i, is associated with a source rate, $x_i \geq 0$, and a fixed route. The routes are specified by an $I \times N$ zero-one matrix, $\boldsymbol{A}$, where $A_{i,n} = 1$ if and only if flow i traverses link n. Each link, n, has a capacity of c_n b/s and the entire network is associated with a utility function $f(\boldsymbol{x}) = \sum_{i=1}^{I} f_i(x_i)$, where $f_i(x)$, the utility of flow i, is a strictly concave increasing function.

A slightly more general definition of $(\boldsymbol{p},\alpha)$-proportional fairness than the one of [10] is given here, where the scalar α of [10] is replaced by a vector. Let $\boldsymbol{X}$ be a set of feasible rates, $\boldsymbol{p} = (p_1,\ldots,p_I)^T > \boldsymbol{0}$ be a set of *flow weights*, $\boldsymbol{\alpha} = (\alpha_1,\ldots,\alpha_I)^T \geq \boldsymbol{1}$ be a set of *fairness levels* and $\boldsymbol{c} = (c_1,\ldots,c_N)^T$ be the set of link capacities.

A vector of rates, $\boldsymbol{x}^* = (x_1^*,\ldots,x_I^*)^T$, is $(\boldsymbol{p},\boldsymbol{\alpha})$-proportionally fair if it is feasible and for any other feasible vector $\boldsymbol{x} = (x_1,\ldots,x_I)^T$,

$$\sum_{i=1}^{I} p_i \frac{(x_i - x_i^*)}{x_i^{*\alpha_i}} \leq 0. \tag{1}$$

The standard feasible set, $\boldsymbol{X}$, for single service flows is given by $\boldsymbol{A}^T\boldsymbol{x} \leq \boldsymbol{c}$ [5]. Proportional fairness [5] is obtained by setting $\alpha_i = 1$ and $p_i = 1$ for all i; max-min fairness is obtained by letting $\alpha_i \to \infty$ [10, Lemma 3]. The extension here, where α is flow dependent, enables a mixture of fairness levels in the same network.

The correspondence between $(\boldsymbol{p},\boldsymbol{\alpha})$-proportional fairness and optimal rates can be observed by considering the constrained optimization problem $\max_{\boldsymbol{x}\in\boldsymbol{X}} f(\boldsymbol{x})$, where $f(\boldsymbol{x}) = \sum_{i=1}^{I} f_i^{\alpha_i}(x_i)$ and

$$f_i^{\alpha_i}(x_i) = \begin{cases} p_i(1-\alpha_i)^{-1}x_i^{1-\alpha_i}, & \text{if } \alpha > 1; \\ \\ p_i \ln(x_i), & \text{if } \alpha = 1. \end{cases} \tag{2}$$

As pointed out in [13], a solution, $\boldsymbol{x}^*$, to this constrained optimization program is $(\boldsymbol{p},\boldsymbol{\alpha})$-proportional fair if and only if the constraint set is convex. That is, fairness follows merely from the structure of the utility function as long as the constraint set is convex. Hence, the constraints can be extended so as to cover QoS requirements without violating fairness.

The extension of $(\boldsymbol{p},\alpha)$-proportional fairness to multi-service flows is done by further constraining the feasible rates so as to guarantee all flow QoS requirements. Note that constraining the flow rates is 'dual' to bandwidth over-provisioning. The main difference being that it is more controlled and adaptive to traffic fluctuation. To guarantee all QoS requirements, the following constraints are introduced.

1) Delay and Packet Loss Constraints: Priority packet scheduling is a flexible way for addressing multiple QoS requirements. Suppose that each flow is classified into one out of M priority classes based on its end-to-end delay and packet loss requirements, where 1 labels the highest priority and M labels the lowest. For bounding the expected packet queueing time of m-priority flows in link n, an upper bound constraint of $\overline{q}_n(m)$ is imposed on $\sum_{j=1}^{m} E\big(Q_n(j)\big)$, where $Q_n(j)$ is the queue length of the j-priority flows in link n.

Assuming that the performance of each router can be approximated by an M/M/1 preemptive priority queue yields

$$\sum_{j=1}^{m} E\big(Q_n(j)\big) = \frac{\sum_{j=1}^{m}\rho_n(j)}{1-\sum_{j=1}^{m}\rho_n(j)}, \tag{3}$$

where $\rho_n(m) = [\boldsymbol{A}^T\boldsymbol{x}(m)]_n/c_n$, $\boldsymbol{x}(m) = \big(x_1(m),\ldots,x_I(m)\big)^T$ and $x_i(m) = x_i$ if flow i is of class m, and zero otherwise.

Thus, the upper bounds on queueing time are converted to upper bounds $\overline{\boldsymbol{q}}(m) = \big(\overline{q}_1(m),\ldots\overline{q}_N(m)\big)$ on the queue lengths $\sum_{j=1}^{m} E\big(Q_n(j)\big)$, $m = 1,\ldots,M$, given by the following M linear constraints presented in a matrix form:

$$\sum_{j=1}^{m} \boldsymbol{A}^T\boldsymbol{x}(j) \leq \frac{\overline{\boldsymbol{q}}(m)}{1+\overline{\boldsymbol{q}}(m)}\boldsymbol{c}, \quad m = 1,\ldots,M, \tag{4}$$

where $\boldsymbol{xy}$ is an element-wise product.

Since packet losses are mainly due to buffer overflow, controlling the average link queue lengths can guarantee also packet loss rate. This PHB is translated into an end-to-end guarantee by using an outer-loop protocol (defined below) which adapts the upper bounds $\{\overline{\boldsymbol{q}}(m)|m=1,\ldots,M\}$ based on estimated packet delays and losses.

2) Minimum Rate Constraints: For minimum rate requirements, the following constraints used in [11] are added:

$$\boldsymbol{x} \geq \underline{\boldsymbol{r}}, \tag{5}$$

where $\underline{\boldsymbol{r}} = (\underline{r}_1, \ldots, \underline{r}_I) \geq 0$.

3) Fairness and Optimality with Multi-service Flows: Considering the multi-service flows defined in Subsections II-A1 and II-A2, the set of feasible rates is now given by the convex set:

$$\boldsymbol{X} = \left\{ \boldsymbol{x} \geq \underline{\boldsymbol{r}};\ \sum_{j=1}^{m} \boldsymbol{A}^T \boldsymbol{x}(j) \leq \frac{\overline{\boldsymbol{q}}(m)}{1+\overline{\boldsymbol{q}}(m)} \boldsymbol{c},\ \forall\ m \right\}, \tag{6}$$

Consequently, $(\boldsymbol{p}, \boldsymbol{\alpha})$-proportional fairness with multi-service flows naturally extends by using the same traditional definition of (1) with the new feasible set of (6).

Thus, as discussed at the beginning of this section, $\boldsymbol{x}^*$ is $(\boldsymbol{p}, \boldsymbol{\alpha})$-proportionally fair with multi-service flows if and only if it is the optimal (unique) solution to

$$\max_{\boldsymbol{x} \in \boldsymbol{X}} \sum_{i=1}^{I} f_i^{\alpha_i}(x_i), \tag{7}$$

where $f_i^{\alpha_i}$ is defined in (2) and $\boldsymbol{X}$ is given in (6).

An important characteristic of multi-service fairness defined inhere is that the primary service requirements, e.g., packet delay, packet loss and minimum rates are placed in the constraint set along with the link capacity constraints. Then, as in the original fairness definition, a proper separable concave utility function is chosen to ensure that rates are allocated fairly subject to the constraints. The specific shape of each flow utility function determines the fairness level, which is subjective. Note that this fairness definition is across service classes and not within each service class.

B. Computing Fair Flows with Multiple Priority Classes

For end-to-end flow control, the algorithm of [5] is extended to multiple priority flows. Unlike the algorithm of [5], where routers measure link utilization, routers in the algorithm here receives the the transmission rates from the end nodes.

The extension here adds another element to the extension of [13], where $\underline{\boldsymbol{r}} = \boldsymbol{0}$, i.e., no minimum rate requirements. The extension for $\underline{\boldsymbol{r}} > \boldsymbol{0}$ becomes straightforward after shifting the rate variables to $\boldsymbol{y} = \boldsymbol{x} - \underline{\boldsymbol{r}}$.

Noting that minimum rate requirement translates into a reduction of the original available link capacities (in excess of the required minimum) from $\boldsymbol{c}$ to $\boldsymbol{c}' = \boldsymbol{c} - \boldsymbol{A}^T \underline{\boldsymbol{r}}$, the Lagrangian of problem (7) becomes

$$\begin{aligned} L(\boldsymbol{y}, \boldsymbol{\Lambda}) \quad &= \sum_{i=1}^{I} \sum_{m=1}^{M} p_i (1-\alpha_i)^{-1} y_i(m)^{1-\alpha_i} \\ &\quad + \sum_{n=1}^{N} \sum_{m=1}^{M} \lambda_n(m) \left(c_n(m) - \sum_{k=1}^{I} \sum_{j=1}^{m} A_{k,n} y_k(j) \right), \end{aligned}$$

where $\boldsymbol{\Lambda} = \{\lambda_i(m)\}$ are the Lagrange multipliers, $p_i > 0$, $\alpha_i \geq 1$ are given scalars and $c_n(m) = \overline{q}_n(m) c'_n / (1 + \overline{q}_n(m))$.

By equating the gradient of $L(\boldsymbol{y}, \boldsymbol{\Lambda})$ with respect to $\boldsymbol{y}$ to zero, it is simple to verify that for every non-negative Lagrange multipliers, $\max_{\boldsymbol{y} \geq \boldsymbol{0}} L(\boldsymbol{y}, \boldsymbol{\Lambda})$ is attained by

$$y_i(m) = \chi_i(m) \left[\frac{p_i}{\sum_{n=1}^{N} \sum_{j=m}^{M} A_{i,n} \lambda_n(j)} \right]^{1/\alpha_i}, \tag{8}$$

where $\chi_i(m) = 1$, if flow i is of priority m; and zero otherwise. Restricting attention to the case of $\alpha_i \equiv 1$ (proportional fairness), and removing terms not depending on the Lagrange multipliers, the dual problem becomes

$$\begin{aligned} \max_{\boldsymbol{\Lambda} \geq \boldsymbol{0}} \quad & \left\{ \sum_{i=1}^{I} \sum_{m=1}^{M} p_i \ln \left(\sum_{n=1}^{N} \sum_{j=m}^{M} A_{i,n} \lambda_n(j) \right) \right. \\ & \left. - \sum_{n=1}^{N} \sum_{m=1}^{M} \lambda_n(m) c'_n(m) \right\}. \end{aligned} \tag{9}$$

Observe that a single priority class optimization problem and the multiple priority classes problem (7) involve flow rates and capacity constraints, both having the same mathematical setting differing only by which flow rates are transparent to other flows. Following the derivation in [13], a synchronous discrete time algorithm for computing the fair rates, i.e. the optimal solution is given by the to following iterations.

Each flow i with priority m updates its target rate at time $t+1$, $x_i(t+1)$, using the source node iteration:

$$x_i(m, t+1) = \underline{r}_i + \frac{p_i}{\sum_{n=1}^{N} \sum_{j=1}^{m} A_{i,n} \lambda_n(t,j)}, \tag{10}$$

where $\lambda_n(t,j)$ is the feedback information received from link n computed by the following link iteration:

$$\lambda_n(t+1, m) = \left[\lambda_n(t,m) + \kappa \left(F(n,m,t) - \frac{c_n(m) \lambda_n(t,m)}{\lambda_n(t,m) + \epsilon} \right) \right]^+, \tag{11}$$

where κ is a positive constant, ϵ is an arbitrarily small number, $F(n,m,t) = \sum_{k=1}^{I} \sum_{j=1}^{m} A_{k,n} \big(x_k(j,t) - \underline{r}_k \big)$ and $[x]^+ = \max\{x, 0\}$.

Observe that $c_n(m)$ is a function of the total minimum rate required by all the flows traversing link n and the upper bound $\overline{q}_n(m)$. This information is delivered by the RMP from the edge routers to the respective core routers using an anonymous reservation protocol and an outer-loop protocol which adapts the bounds based on packet delay and packet loss probing.

The time lagged asynchronous distributed version of iterations (10) and (11) is defined as follows. Let $d^1_{k,n}$ and $d^2_{k,n}$ be non-negative integers, where $d^1_{k,n}$ is the delay between the rate change at the source of flow i (henceforth called: source i) and its reflection on link n; and $d^2_{k,n}$ is the delay between the feedback update at link n and its reflection on source i.

Flow i with priority m updates its target rate at $t+1$ by

$$x_i(m, t+1) = \underline{r}_i + \frac{p_i}{\sum_{n=1}^{N} \sum_{j=1}^{m} A_{i,n} \lambda_n(t - d^1_{i,n}, j)}, \tag{12}$$

where $\lambda_n(t', j)$ is the feedback information from link n retrieved at time t'.

The feedback information of each virtual link n_m at time $t+1$, $\lambda_n(t+1, m)$, is updated in its attached router by

$$\lambda_n(t+1, m) = \left[\lambda_n(t, m) + \kappa\left(F(n, m, t) - \frac{c_n(m)\lambda_n(t, m)}{\lambda_n(t, m) + \epsilon}\right)\right]^+, \quad (13)$$

where the total unreserved capacity of link n used by all flows with priorities $1, \ldots, m$, is given by

$$F(n, m, t) = \sum_{k=1}^{I}\sum_{j=1}^{m} A_{k,n}\big(x_k(t - d^2_{k,n}, j) - \underline{r}_k\big), \quad (14)$$

and $x_k(t', j)$ is the transmission rate of source k at time t'.

Applying the results of [5], the following theorem follows.

Theorem 1: If there is a feasible solution to (7), then for arbitrarily small values of κ and ϵ, iterations (12) and (13) stabilize around rates arbitrarily close to the proportionally fair rates. ■

Stabilization in Theorem 1 means that the variations from the proportionally fair rates are arbitrarily small.

III. A Rate Management Protocol (RMP)

The protocol here, which is part of the USPTO patent pending [12], extends the one of [13] by further addressing minimum rate allocation, flow admission control, and adaptive bounding of the link utilization. RMP sole objective is the computation of the target transmission rates; traffic shaping is done by token bucket.

RMP is a distributed, asynchronous, iterative and scalable protocol. If the network dynamics resembles the fluid network dynamics (which is observed in very large networks), the target rates converge to the fair rates satisfying the QoS requirements for minimum transmission rates, end-to-end packet delay, and packet loss rates. Theorem 1 and the simulation results reported in Section IV support this assertion.

RMP comprises two modules: an edge module executed in every edge router; and a core module executed in every IP router (edge and core). A flow in the protocol is an *aggregate flow* comprising multiple *application flows* with the following common characteristics: (i) core network route; (ii) weight; (iii) minimum rate requirement; (iv) end-to-end packet delay requirement; and (v) packet loss rate requirement. Characteristics (ii)-(v) determine the flow QoS class.

For each active (aggregate) flow, an RMP edge module generate special packets transmitted toward the flow destination edge modules. Except for bandwidth reservation/release messages requiring reliability, core modules are flow-unaware and process RMP messages in a stateless manner. Once the minimum bandwidth of a flow is reserved, its target rate computation does not require from the core modules to be flow-aware. Consequently, RMP is scalable as its core module processing complexity is the same as for IP packet switching.

For avoiding an obscure pseudocode, the RMP specification focuses on the target rate calculation. It is assumed that a router processes all packets (RMP and payload) and provides the RMP modules with the relevant triggering events described below. At the high level, RMP modules operate as follows:

- Each edge module maintains a a table of active aggregate flows originating from its edge router. It is assumed that the add/remove/update table operations are triggered by a router process based on application flow activities, i.e., payload transmission, arrivals and departures.
- There are two types of flows: with and without minimum transmission rate requirement. Flows with minimum transmission rates are subject to admission control. For admission, the new required delay and packet loss along with the potential deterioration of current packet delays and packet losses is first verified. Then, a reliable two-phase commit procedure is used for bandwidth reservation. A new flow with minimum transmission rate is accepted only if its packet delay, packet loss and minimum bandwidth can be met, and the packet loss and packet delay of existing flows are protected. Flows without minimum transmission rates are accepted on a 'best-effort-rate' basis, i.e., their packet delay and packet loss are guaranteed on the expense of their transmission rates.
- Each network link capacity is partitioned into reserve capacity and residual capacity. The latter is shared fairly amongst all active flows. Every change in the mixture of an aggregate flow with a minimum rate requirement triggers a *join* or an *exit* event followed by a reliable reserve/release procedure. When an aggregate flow becomes inactive (exiting), its fair share from the residual capacities is also released by a reliable procedure. Reliability is achieved by using message sequence numbers and timeout-triggered retransmissions.
- For every active aggregate flow, each edge module also transmits regularly a stream of *forward RMP messages* toward its edge router destination using an unreliable protocol. Each message contains the flow characteristics and control information. Regular RMP messages are dual-purpose; they serve as performance probes and as couriers for information exchanged amongst the edge and core modules.
- An RMP message contains the following information about its aggregate flow: (i) local identity along with its source IP address; (ii) a increnet/decrement to the previous transmission rate as computed by (12); (iii) a hint value for updating the link utilization upper bounds; (iv) a reserved link capacity update; (v) flow priority; (vi) message sequence number; and (vii) a network feedback information: a reservation response or a congestion penalty computed by (13).
- Forward RMP messages are processed by every core module they traverse toward their destined edge modules. The processing include updates of (in message fields and in core module tables): reserved capacity, utilized residual capacity, link utilization upper bound, and feedback information.
- When a forward RMP message arrives at the destined edge module, it is marked as a *backward RMP message* and sent back to its originating edge module. Backward RMP messages are just forwarded by the core modules without any processing.
- When a backward RMP message returns to the originating edge module its round trip time (RTT) updates the estimated

RTT and its sequence number is used for updating the estimated packet loss. Additionally, its feedback field is used for updating the transmission rate according to (12).

The pseudocode is given in the next subsection.

A. RMP with Multi-Service Flows

Let i be a unique local label of an aggregate flow at a given edge module and denote by $rmp(i)$ an RMP message associated with flow i. An $rmp(i)$ comprises of: implicit global id $rmp(i).id$ (determined by the flow source-destination IPs and its label i); $rmp(i).prio$ (flow priority); $rmp(i).\Delta r$ (different in residual capacity usage compared to previous usage); $rmp(i).\Delta \underline{r}$ (different in reserved rate compared to the previous rate); $rmp(i).utilHint$ (1,0,-1 for increment, leave as, decrement link utilization upper bound, respectively); $rmp(i).\lambda$ (network feedback data); $rmp(i).sn$ (message sequence number); $rmp(i).type$ (message type taking values from $\{reg, res, rel, reset\}$ standing for regular unreliable, reliable bandwidth reservation, reliable release, and reliable resetting residual capacity usage, respectively); $rmp(i).fwd$ (forward/backward packet); and $rmp(i).cmt$ (committal/noncommittal).

Each edge node module contains a flow table comprising the following information for each aggregate flow i: $\underline{r}'(i)$ (previous minimum required transmission rate); $\underline{r}(i)$ (current minimum required transmission rate); $r'(i)$ (previous residual capacity usage); $r(i)$ (current residual capacity usage); $prio(i)$ (flow priority); $weight(i)$ (flow weight as given by $p(i)$ defined in Section II-A); $\overline{loss}(i)$ (maximum required packet loss rate); $\overline{delay}(i)$ (maximum required packet round trip time); $delayEst(i)$ (estimated packet delay); $lossRate(i)$ (estimated packet loss rate); $sn(i)$ (the last transmitted sequence number).

Each core node module contains a link table comprising the following information for each output link n and packet priority m: c_n (capacity of link n); c'_n (unreserved capacity of link n); $F(n,m)$ (unreserved capacity of link n utilized by packets with priorities $\leq m$); $\overline{u}_n(m)$ (utilization upper bound on the unreserved capacity of link n for packets with priorities $\leq m$); $\lambda'_n(m)$ (previous penalty of link n for priorities $\leq m$); $\lambda_n(m)$ (current penalty of link n for priorities $\leq m$); $tuN_n(m)$ (total counter of rmp packets with priority m traversing link n since the last change of upper bound $\overline{u}_n(m)$); $cuN_n(m)$ (sum of increment minus decrement change requests from rmp packets with priority m traversing link n since the last change of upper bound $\overline{u}_n(m)$);

Note that the total transmission rate of each flow utilizes the minimum required reserved bandwidth (if specified) and a fair share from the residual unreserved bandwidth.

The specification of RMP is given in two set of modules, each comprising multiple processes. The first set specifies the edge and the core modules of the protocol part using regular unreliable transmission of RMP messages for each active flow that has been admitted to the network. The second set specifies the protocol part handling the arrivals and departures of application flows with minimum rate requirements, as well as the termination of an aggregate flow. This part uses a reliable protocol for guaranteeing consistent bandwidth reservation.

1) RMP basic edge and core modules: The pseudocode of the RMP edge and core modules for flows which have been admitted to the network is given in Figures 1-5. Flows are admitted to the network only after their reserved bandwidth in the network have been updated, and their requirement for packet delay and packet loss guarantee have been verified. This part of the protocol uses unreliable transmissions and computes the fair rate for each active flow so as to meet packet delay and packet loss.

Edge Process 1: NEWRMPMSG$(i, p, \Delta r, \Delta \underline{r}', sn)$
1. $rmp(i).\lambda \leftarrow 0$; /* reset feedback info */
2. $rmp(i).\Delta r \leftarrow \Delta r$;
3. $rmp(i).\Delta \underline{r} \leftarrow \Delta \underline{r}'$;
4. $rmp(i).prio \leftarrow p$;
5. $rmp(i).sn \leftarrow sn$;
6. $rmp(i).fwd \leftarrow$ **true** ;
7. $rmp(i).type \leftarrow$ **reg**;
8. $rmp(i).cmt \leftarrow 0$;
/* **compute utilization upper bound hint** */
/* $(\alpha_d^1, \alpha_d^2, \beta_l^1, \beta_l^2)$ are tuning parameters */
9. if $\Big((delayEst(i) < \overline{delay}(i) - \alpha_d^1)$ **and** $(lossRate(i) < \overline{loss}(i) - \beta_l^1)\Big)$ **then** $rmp(i).utilHint \leftarrow 1$; /* relax bound */
10. else if $\Big((delayEst(i) > \overline{delay}(i) + \alpha_d^2)$ **or** $(lossRate(i) > \overline{loss}(i) + \beta_l^2)\Big)$ **then** $rmp(i).utilHint \leftarrow -1$; /* tighten bound */
11. else then $rmp(i).utilHint \leftarrow 0$; /* leave as is */
return $(rmp(i))$

Fig. 1. Edge module: new RMP message constaruction.

Edge Process 2: UPDATEFLOWTABLE$(event, i, prio, w, \underline{r}, \overline{d}, \overline{l})$
if $(event = new)$ **then**
/* **a new accepted aggregate flow** */
/* **set variables in local table** */
1. $\underline{r}'(i) \leftarrow 0$; /* set previous min required rate */
2. $\underline{r}(i) \leftarrow \underline{r}$; /* set current min required rate */
3. $r'(i) \leftarrow 0$; /* set previous residual transmission rate */
4. $r(i) \leftarrow 0$; /* set current residual transmission rate */
5. $\overline{loss}(i) \leftarrow \overline{l}$; /* set max required loss */
6. $\overline{delay}(i) \leftarrow \overline{d}$; /* set max required delay */
7. $delayEst(i) \leftarrow 0$; /* set delay estimator */
8. $lossRate(i) \leftarrow 0$; /* set loss estimator */
9. $prio(i) \leftarrow prio$; /* set priority */
10. $weight(i) \leftarrow w$; /* set flow weigh (p_i in II-A) */
11. $sn(i) \leftarrow 0$; /* initialize sequence number */
else if $(event = join)$ **then**
/* **a new accepted application flow with min rate** */
12. $\underline{r}(i) \leftarrow \underline{r}(i) + \underline{r}$; /* update current min rate */
else if $(event = exit)$ **then**
/* **an application flow exit** */
if $(a\ flow\ with\ min\ rate)$ **then**
13. $\underline{r}(i) \leftarrow \underline{r}(i) - \underline{r}$; /* update current min rate */
14. if $(\underline{r}(i) = 0)$ **then** /* last appl flow with min rate left */ *"remove flow i from local table" ;*
else if $(r(i) = 0)$ **then** /* last appl flow w/o min rate left */
15. *"remove flow i from local table" ;*
return

Fig. 2. Edge module: active flow table update.

```
Edge Process 3: MAIN(0)
/* each active flow i has a retransmission timeout, RTO(i) */
wait (event)
  if (event = timeout(i) or new) then
      /* rmp(i) packet has not return within RTO(i) */
      /* or a new aggregate flow added to flow table */
      { /* nothing to update */
        /* send another rmp packet (below) */
  else if (event = fwd rmp(i) arrival) then
      { /* send it back to origin */
        1. rmp(i).fwd ← false ;
        2. send rmp(i) with priority p to the flow origin ;
        3. Goto wait(event) ;
  else if (event = bkwd rmp(i) arrival) then
      { /* update packet delay estimator */
        /* using damp factor 0 < γ < 1 */
        4. rtt ← rmp packet round-trip-time ; ;
        5. delayEst(i) ← γ × delayEst(i) + (1 − γ) × rtt ;
        /* update packet loss rate estimators */
        /* using damp factor 0 < η < 1 */
        6. L ← rmp(i).sn − previous-sn-received ;
        7. lossRate(i) ← η × lossRate(i) + (1 − η) × (L−1)/L ;
        /* update previous & current residual rates */
        8. r'(i) ← r(i) ;
        9. r(i) ← weight(i)/rmp(i).λ ;
    /* send another rmp packet */
    10. sn(i) ← sn(i) + 1 ;
    11. { p ← prio(i) ; sn ← sn(i) ;
          Δr = r(i) − r'(i) ; Δr = r(i) − r'(i)) ;
    12. rmp(i) ← NEWRMPMSG(i, p, Δr, Δr, sn(i))
    13. send rmp(i) with priority p to the flow destination ;
continue
```

Fig. 3. Edge module: main process.

```
Core Process 1: INITIALIZE(0)
 for each (m = 1, ..., M and output link n)
    initialize : c_n, c'_n, ū_n(m), F(n, m), c_n(m),
        λ'_n(m), λ_n(m), tuN_n(m), cuN_n(m)
 return
```

Fig. 4. Core module: initialization.

2) RMP reliable edge and core modules: Since RMP also handles flows with minimum rate requirements, a reliable procedure for bandwidth reservation/release is needed. Furthermore, for achieving flow-unaware operation in the core modules, each flow source updates the core modules about about its current residual capacity usage using rate-differences. Consequently, a reliable final update is needed upon flow exit.

Additionally, since bandwidth reservation is for multiple links and is done in sequence, a two-phase commit reservation procedure is required. Namely, during the first round-trip of a reservation RMP packet, each link registers the request; and during the second round-trip (if bandwidth is available at each route link), a reservation commit is performed. Flows with minimum rate requirements requesting bandwidth reservation/release, and flows without minimum rate requirements exiting the network are referred to as *flows in a request mode*.

Reliability is achieved by using sequence numbers and retransmissions of RMP packets along with a short-live table in each core module of flows in a request mode.

```
Core Process 2: MAIN(0)
wait (event)
  if (event = bkwd rmp(i) arrival) then
    0. forward rmp(i) to its destination ;
  else if (event = fwd rmp(i) arrival) then
      { 1. m ← rmp(i).prio ;
        2. n ← packet switching output link ;
        3. for (j = m, ..., M) do
             F(n, j) ← F(n, j) + rmp(i).Δr ;
        4. λ'_n(m) ← λ_n(m) ; /* update previous penalty */
        5. λ_n(m) ← Eq. (13) ; /* compute new penalty */
        6. tuN_n(m) ← tuN_n(m) + 1 ; /* update total counter */
        7. cuN_n(m) ← rmp(i).utilHint ; /* update change counter */
           /* update utilization upper bound */
           /* use 1 > th_1 ≫ 0.5 */
        8. if (cuN_n(m)/tuN_n(m) > th_1) then
            { /* relax upper bound */
              8.1. ū_n(m) ← min{1, ū_n(j) + δ} ;
              8.2. tuN_n(m) ← 0 ; cuN_n(m) ← 0 ;
              8.3. for (j = m + 1, ..., M) do
                   { update ū_n(j) to keep monotonicity ;
                     and reset tuN_n(j), cuN_n(j), if updated ;
           /* use 1 > th_2 ≫ 0.5 */
        9. else if (cuN_n(m)/tuN_n(m) < th_2) then
            { /* tighten upper bound */
              9.1. ū_n(m) ← ū_n(j)/2 ;
              9.2. tuN_n(m) ← 0 ; cuN_n(m) ← 0 ;
              9.3. for (j = 1, ..., m − 1) do
                   { update ū_n(j) to keep monotonicity ;
                     and reset tuN_n(j), cuN_n(j), if updated ;
        10. rmp(i).λ ← rmp(i).λ + λ_n(m) ;
        11. forward rmp(i) with priority p to its destination ;
continue
```

Fig. 5. Core module: main process.

The pseudocode of the RMP edge and core modules for flows for flows in a request mode is given in Figures 6-9.

```
Edge Process 4: MAINRELIABLE(0)
wait (event)
    1. if (event = none − reg fwd rmp(i) arrival) then
       { rmp(i).fwd ← false ;
         send rmp(i) with top priority back to source ;
    2. else if (event = release(i, r, r)) then RELBNDW(i, r, r) ;
    3. else if (event = reserve(i, m, r, d̄, l̄)) then
        { /* do preliminary check if network can accommodate */
          /* a new flow with prio m, where δ_d, δ_l are active flow */
          /* protection levels for delay and loss */
          if (delayEst of priority m > d̄ − δ_d) then
              reject flow i ;
          else if (lostEst of priority prio > l̄ − δ_l) then
              reject flow i ;
          else RESBNDW(i, r) ;
continue
```

Fig. 6. Edge module: main reliable process.

Remark 1: Regular unreliable RMP packets can be lost and overtake each other affecting RMP convergence. As long as these events are infrequent, their "noise" can be modeled as part of the asynchronous behavior which has been shown not affecting the convergence.

Remark 2: RMP is not a traffic shaper. The rates that are

Edge Process 5: RELBNDW($i, r, \underline{r}$)
/* If $\underline{r} > 0$, release reserved bandwidth allocated to aggregate */
/* flow i due to exit of an appl flow with min required rate $\underline{r}$ */
/* If $r > 0$, remove also the residual rate r used by */
/* aggregate flow i due to its exit */
/* Each release request uses a unique sequence no., sn_rel(i) */
1. $sn\_rel(i) \leftarrow sn\_rel(i) + 1$; **/*** update sequence number ***/**
2. $\begin{cases} p \leftarrow prio(i) \ ; sn \leftarrow sn\_rel(i) \ ; \\ \Delta r = -r \ ; \Delta \underline{r} = -\underline{r} \ ; \end{cases}$
3. $rmp(i) \leftarrow$ NEWRMPMSG$(i, p, \Delta r, \Delta \underline{r}, sn)$;
4. $rmp(i).type \leftarrow$ **rel** ;
5. $rmp(i).fwd \leftarrow$ **true** ;
6. *send rmp(i) with top priority to the flow destination ;*
/* Note: the global flow id is determined by i, p */
/* and source-destination IP addresses */
7. wait $(event)$
7.1. if $(event = timeout(i))$ **then**
/* rmp(i) packet has not returned within RTO(i) ***/**
/* resend the same rmp with the same rmp(i).sn ***/**
resend rmp(i) with top priority to flow destination ;
7.2. else if $(event = rmp(i)\ bkwd\ return)$ **return** ;
continue

Fig. 7. Edge module: release bandwidth process.

Edge Process 6: RESBNDW($i, \underline{r}$)
/* Reserve bandwidth $\underline{r}$ for flow i using a 2-phase commit */
/* Each reserve request uses a unique sequence no., sn_res(i) */
1. $sn\_res(i) \leftarrow sn\_rel(i) + 1$; **/*** update sequence number ***/**
2. $\begin{cases} p \leftarrow prio(i) \ ; sn \leftarrow sn\_res(i) \ ; \\ \Delta r = 0 \ ; \Delta \underline{r} = \underline{r} \ ; \end{cases}$
3. $rmp(i) \leftarrow$ NEWRMPMSG$(i, p, \Delta r, \Delta \underline{r}, sn)$;
4. $rmp(i).type \leftarrow$ **res** ;
5. $rmp(i).fwd \leftarrow$ **true** ;
6. $rmp(i).cmt \leftarrow 0$; **/*** phase 1 ***/**
7. *send rmp(i) with top priority to the flow destination ;*
/* Note: the global flow id is determined by i, p */
/* and source-destination IP addresses */
8. wait $(event)$
8.1. if $(event = timeout(i))$ **then**
/* rmp(i) packet has not returned within RTO(i) */
/* resend the same rmp with the same rmp(i).sn */
resend rmp(i) with top priority to flow destination ;
8.2. else if $(event = rmp(i)\ bkwd\ return)$
8.2.1. if $(rmp(i).\lambda = 1 \wedge rmp(i).cmt = 1)$ **then**
/* phase 2 reservation completed successfully */
accept flow i ;
return ;
8.2.2. else if $(rmp(i).\lambda = 1 \wedge rmp(i).cmt = 0)$ **then**
/* phase 1 reservation completed successfully */
/* send commit */
$rmp(i).cmt \leftarrow 1$;
send rmp(i) with top priority to flow destination ;
8.2.3. else if $(rmp(i).\lambda = 0$ **and** $rmp(i).cmt = 0)$
/* phase 1 reservation completed unsuccessfully */
reject flow i ;
return ;
continue

Fig. 8. Edge module: reserve bandwidth process.

iteratively updated at the edge modules are passed to a token bucket traffic shaper. The traffic shaper does not wait until RMP convergence, instead it shapes the traffic "on-the-fly" based on the most current rates. Such shaping is required in practice since the set of active flows vary in time.

Core Process 3: MAINRELIABLE(0)
wait $(event)$
if $(event = none - reg\ rmp(i)\ arrival)$ **then**
1. if $(rmp(i).fwd =$ **false** $)$ **then**
do nothing ;
2. else if $(rmp(i).fwd =$ **true** $)$ **then**
$m \leftarrow rmp(i).prio$;
$n \leftarrow$ *packet switching output link ;*
$id \leftarrow$ *flow global id ;*
$sn \leftarrow rmp(i).sn$; **/*** release-request unique sn ***/**
$\underline{r} \leftarrow rmp(i).\Delta \underline{r}$; **/* '-' of reserved capacity */**
$r \leftarrow rmp(i).\Delta r$; **/* '-' of residual capacity used */**
if $(rmp(i).type =$ **rel**$)$ **then**
/* capacity release request */
/* same release request may arrive more than once */
lookup(id,sn) $\in$ ***relTable*** *;*
if $((id, sn\_rel) \notin$ **relTable**$)$ **then**
$c'_n \leftarrow c'_n - \underline{r}$ **/*** increase residual capacity ***/**
for $(j = m, \ldots, M)$ **do**
/* decrease relevant flow rates */
$F(n, j) \leftarrow F(n, j) + r$;
add(id,sn) to ***relTable*** *and set its cleanTimer ;*
else if $(rmp(i).type =$ **res**$)$ **then**
if $(rmp(i).cmt = 0)$ **then**
/* Phase 1 of 2-phase commit: the same */
/* reserve request may arrive more than once */
lookup(id,sn) $\in$ ***res1Table*** *;*
if $((id, sn) \notin$ **res1Table**$)$ **then**
if $(\underline{r} < c'_n)$ **then**
$c'_n \leftarrow c'_n - \underline{r}$; **/*** conditional reserve ***/**
add(id,sn) to ***res1Table*** *& set its cleanTimer ;*
$rmp(i).\lambda \leftarrow (rmp(i).\lambda \wedge 1)$
else $rmp(i).\lambda \leftarrow (rmp(i).\lambda \wedge 0)$
else $rmp(i).\lambda \leftarrow (rmp(i).\lambda \wedge 1)$
else if $(rmp(i).cmt = 1)$ **then /*** phase commit ***/**
/* the same reserve request */
/* may arrive more than once */
lookup(id,sn) $\in$ ***res2Table*** *;*
if $((id, sn) \notin$ **res2Table**$)$ **then**
remove(id,sn) from ***res1Table*** *& reset cleanTimer ;*
add(id,sn) to ***res2Table*** *and set its cleanTimer ;*
$rmp(i).\lambda \leftarrow (rmp(i).\lambda \wedge 1)$
3. *forward rmp(i) with top priority to its destination ;*
else if $(event = cleanTimer(table, id, sn)$ **then**
remove(id,sn) from ***table*** *;*
if $(table = res1Table)$ **then** $c'_n \leftarrow c'_n + \underline{r}$; **/*** unreserve ***/**
continue

Fig. 9. Core module: main reliable process.

Remark 3: Since core node modules are not flow-aware (except for very short time intervals), RMP is scalable in the number of flows. The router processing, which is a function of its input packet rate, x, increases from an order of x packet/sec, to an order of $x + y$ packet/sec, where y is regulated by the timeout timer of Edge Process 3 in Figure 3.

IV. A PERFORMANCE STUDY

To verify and demonstrate the stability and the effectiveness of the self-tuning RMP, an efficient and fast simulator has been developed using Matlab. The fast simulator has been verified against a detailed NS2 simulator and exhibits excellent match. The simulator methodology and its verification results is described briefly in [13] and will be fully reported elsewhere.

The Matlab fast simulator is needed for large network simulation that could not be done with NS2 within a reasonable time fashion. Unlike NS2 simulation requiring about a week for a network of 40 half-duplex core links and 60 flows, Matlab simulation requires few minutes for the same task.

The USA Epoch ISP network [1] depicted in Figure 10 is used as the evaluation test bed. The network comprises 37 full-duplex core links, i.e., $N = 74$ half-duplex links, each of capacity 1 Gb/s. The propagation delay of a core link is a function of the link length and the propagation delays of each access link (not in the graph) is fixed at five nanosecond.

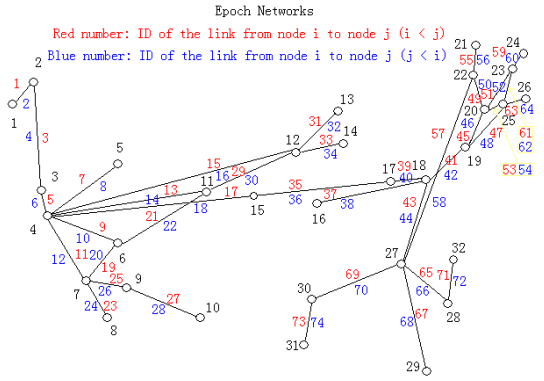

Fig. 10. Epoch ISP Network Topology, USA.

The core network serves 1536 one-way flows from 3 QoS classes, each comprising 512 flows. A QoS class is characterized by a maximum RTT packet delay, a maximum packet loss rate and a minimum rate. Each flow uses the shortest hop-count route to its destination and two dedicated access links of capacity 100 Mb/s for accessing the source edge router and the destination user node, respectively. It is assumed that packet lengths are fixed and equal 1500 bytes.

To neutralize the routing effect, the same set of routes are used for each QoS class. That is, every active source-destination pair has three flows, one from each QoS class, following the same shortest hop-count route.

A. Stability and effectiveness of RMP

RMP Stability and effectiveness are demonstrated in three simulated scenarios (S) summarized in Table I. A scenario is characterized by $B - Req(m)$, $D - Req(m)$ and $L - Req(m)$ denoting minimum rate, maximum packet RTT and maximum packet loss rate, respectively, of QoS class $1 \leq m \leq 3$.

S	B-Req (packets/sec)	D-Req (msec)	L-Req	RT-PD
1	(104.1667 10.4167 0)	(50 75 150)	(0.01 0.001 0.00)	$\leq$ DelayReq
2	(104.1667 10.4167 0)	(50 60 90)	(0.01 0.001 0.00)	$\leq$ DelayReq
3	(104.1667 10.4167 0)	(50 75 150)	(0.01 0.001 0.00)	$\geq$ DelayReq

TABLE I
SIMULATION SCENARIOS

In the first scenario, flows require a minimum rate not generating excess queueing time. Additionally, their packet RTT requirements are feasible, i.e., they are greater than their corresponding RT-PD. The scenario represents a case where a safe CAC is applied and RMP is expected to allocate and shape the flow rates so as to match all QoS requirements.

The second scenario is similar to the first one differing only by more stringent RTT requirements. It represents another case of safe CAC, where RMP is also expected to match all QoS requirements. However, the link upper bounds, $\{u_n(m)\}$, are expected to be lower compared to the first scenario.

In the third scenario, flows also require minimum rate but the RTTs required by some of the flows are smaller than their corresponding RT-PD, i.e., infeasible. The scenario represents a case where CAC is unsafe admitting flow with infeasible QoS requirements. It examines how RMP copes with infeasible QoS requirements which may occur in practice.

Clearly, to avoid situations where QoS requirements are infeasible, a safe CAC must apply a rule where new flows with infeasible QoS requirements or new flows which may deteriorate the performance of the exiting flows below their QoS requirements, are rejected. Since the effect of a newly admitted flow can only be estimated, a completely safe admission control is infeasible in practice. Consequently, the manner by which RMP copes with the third scenario is important.

The simulation results are summarized in Table II. Beside the adaptive RMP, a non-adaptive RMP with fixed bounds of $\big(u_n(1), u_n(2), u_n(3)\big) = (0.25, 0.50, 0.75)$ is also used for comparison.

The cells of Table II contain triplets (a_1, a_2, a_3), where a_m is a value corresponding to flows from QoS class m. The first three rows contain the proportions of flows with RTT less than or equal the RTT requirements within an error margin of $0,5\%$ and 10%, respectively. Similarly, rows four and five contain the proportions of flows satisfying the bandwidth and packet loss requirements, respectively. The overall average RTT, average packet loss and average transmission rate of the three QoS classes are given in rows six, seven and eight, respectively.

In scenarios $1, 2$, RMP attains all QoS requirements. In the first scenario, the bounds, $\{u_n(m)\}$, are driven rapidly to one. In the second scenario, $\{u_n(m)\}$ are driven to one except for the bounds of links $19, 20, 23, 24, 28, 29$, which are driven to $(0.28, 0.35, 0.77, 0.96, 0.36, 0.35)$ for $m = 1$, and $u_{20}(1) = 0.8$. As expected, the bounds in the second scenario are tighter than those of the first scenario. The upper bounds stabilize around those values with unnoticeable variation. Clearly, the non-adaptive case with $u_n(m) = 1$ for scenario 1 (omitted from the table) achieves the same performance as the adaptive case. For scenario 2, the non-adaptive case with $\big(u_n(1), u_n(2), u_n(3)\big) = (0.25, 0.50, 0.75)$ fail to achieve QoS requirements for QoS classes $2, 3$. Also, adaptive RMP attains all QoS requirements while achieving average transmission rates similar to those of the non-adaptive RMP.

In the infeasible scenario 3, adaptive RMP successfully drives the network to utilization uppers bounds where the feasible RTT requirements are fulfilled. It is more successful in satisfying the QoS requirements compared to the non-adaptive version. The success is achieved by a substantial reduction of transmission rates. Almost all link utilization bounds for QoS 1 are stabilizing below 0.003, except for 10 bounds stabilizing at one. For QoS 2, the bounds are stabilizing below 0.04, except for 10 bounds stabilizing at one. For QoS 3, half of the bounds are stabilizing below 0.20 and half at one.

Measure	Scenario 1	Scenario 2		Scenario 3	
	adapted	adapted	non-adapted	adapted	non-adapted
Prop $\leq$ DelayReq	(1 1 1)	(1 1 1)	(1 .90 .52)	(.65 .63 .99)	(.56 .48 .47)
Prop $\leq$ DelayReq + 5%	(1 1 1)	(1 1 1)	(1 .96 .53)	(.74 .70 1.0)	(.65 .49 .48)
Prop $\leq$ DelayReq + 10%	(1 1 1)	(1 1 1)	(1 .99 .55)	(.85 .78 1.0)	(.73 .50 .49)
Prop $\geq$ BndwReq	(1 1 1)	(1 1 1)	(1 1 1)	(1 1 1)	(1 1 1)
Prop $\leq$ LossReq	(1 1 1)	(1 1 1)	(1 1 1)	(1 1 1)	(1 1 1)
Avg RTT (msec)	(21 31 57)	(21 28 44)	(21 35 70)	(51 57 68)	(54 84 169)
Avg Loss	(0 0 0)	(0 0 0)	(0 0 0)	(0 0 0)	(0 0 0)
Avg Rate (packet/sec)	(1050 1064 1114)	(874 892 972)	(892 936 977)	(305 336 671)	(887 930 970)

TABLE II
SIMULATION RESULTS

The RTT of individual packets of three typical flows from QoS class 2 within a time window of 1500 iterations for scenarios $1-3$ are depicted in Figures 11–13. The figures for QoS class 1 are similar but with lower values and less variability. For QoS class 3, they are also similar but with higher values and higher variability. The RTT variation from one packet to another is contributed by the queueing time variation which increases when the RTT requirement becomes more stringent.

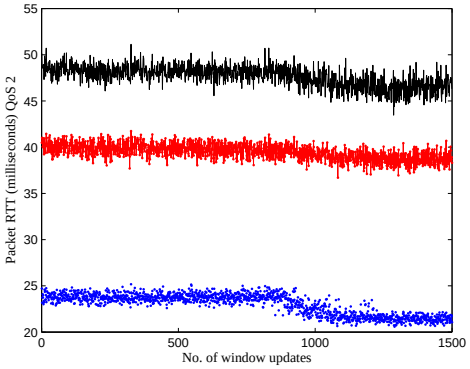

Fig. 11. Scenario 1: Packet RTT of 3 flows from QoS class 2.

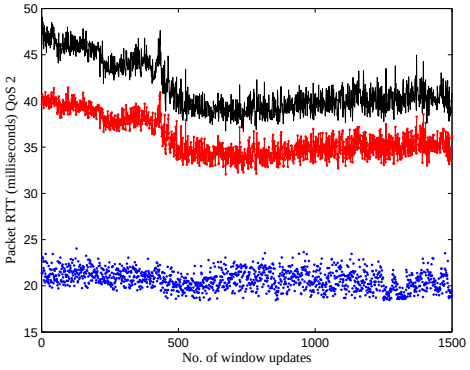

Fig. 12. Scenario 2: Packet RTT of 3 flows from QoS class 2.

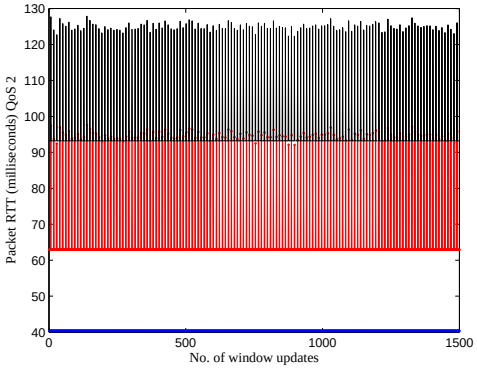

Fig. 13. Scenario 3: Packet RTT of 3 flows from QoS class 2.

REFERENCES

[1] Available at: http://www.isp-planet.com/profiles/epoch.html.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, "An Architecture for Differentiated Services," IETF RFC 2475, Dec. 1998.

[3] R. Braden, D. Clark and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," IETF RFC 1633, June 1994.

[4] K. Kar, S. Sarkar and L. Tassiulas "A Scalable Low-Overhead Rate Control Algorithm for Multirate Multicast Sessions," *IEEE JSAC*, vol. 20, no. 8, pp. 1541–1557, Oct. 2002.

[5] F. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: Shadow price proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, pp. 237-252, 1998.

[6] S. Kunniyur and R. Srikant, "End-to-End Congestion Control Schemes: Utility Functions, Random Losses and ECN Marks," *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 689–702, Oct. 2003.

[7] C. M. Lagoa, H. Che, and B. A. Movsichoff, "Adaptive Control Algorithms for Decentralized Optimal Traffic Engineering in the Internet," *IEEE/ACM Trans. Networking*, vol. 12, no. 3, pp. 415–428, June 2004.

[8] S. H. Low, "A Duality Model of TCP and Queue Management Algorithms," *IEEE/ACM Trans. Networking*, vol. 11, no. 4, pp. 525-536, Aug. 2003.

[9] L. Massouli and J. Roberts, "Bandwidth Sharing: Objectives and Algorithms," *IEEE/ACM Trans. Networking*, vol. 10, no. 3, pp. 320-328, June 2002.

[10] J. Mo and J. Walrand, "Fair End-to-End Window-Based Congestion Control," *IEEE/ACM Trans. Networking*, vol. 8, no. 5, pp. 556–567, Oct. 2000.

[11] G. Rogers, J. Chan, and D. Agahari, Rate Control of Elastic Traffic with QoS Guarantees: a Stability Analysis & Experimental Implementation, Proc. of IFIP/IEEE NOMS 2006, Apr. 2006.

[12] Z. Rosberg, United States Patent application 11,608,834 filed on December 12, 2006.

[13] Z. Rosberg and M. Zukerman, "Multi-Service Flow Control and Fairness for All," CSIRO ICT Centre Pub. no. 07/054, Feb. 2007.

[14] J. Wang, D. X. Wei and S. H. Low, "Modelling and Stability of FAST TCP," Proc. of IEEE INFOCOM 2005, Miami, FL, March 2005.

[15] J. Widmer, R. Denda and M. Mauve, M., "A survey on TCP-friendly congestion control," *IEEE Network*, vol. 15, no. 3, pp. 28–37, 2001.

[16] X. Xiao and L. M. Ni, "Internet QoS: a big picture, *IEEE Network*, vol. 13, no. 2, pp. 8–18, Mar/Apr 1999.