Uncooperative Congestion Control

K. Chandrayana *S. Kalyanaraman*

R.P.I., Troy, NY, USA.
chandk@rpi.edu,shivkuma@ecse.rpi.edu

Abstract— **Traditionally uncooperative rate control schemes have implied open loop protocols such as UDP, CBR. In this paper we show that closed loop uncooperative rate control schemes also exist and that the current AQM proposals cannot efficiently control their mis-behavior. Moreover, these proposals require that AQM be installed at all routers in the Internet which is not only expensive but requires significant network upgrade.**

In this paper we show that management of uncooperative flows need not be coupled with AQM design but can be viewed as edge based policing question. In this paper we propose an analytical model for managing uncooperative flows in the Internet by re-mapping their utility function to a target range of utlity functions. This mapping can be achieved by transparently manipulating congestion penalties conveyed to the uncooperative users.

The most interesting aspect of this research is that this task can be performed at the edge of the network with little state information about uncooperative flows. The proposed solution is independent of the buffer management algorithm deployed on the network. Thus the framework presented in this paper not only works on a network of Drop-Tail queues but also with any AQM scheme. We have analyzed the framework and evaluated it on various single and multi-bottleneck topologies with both Drop-Tail and RED. Our results show that the framework is robust and works well even in presence of background traffic and reverse path congestion.

I. Introduction

Over the years as the Internet has evolved TCP has formed the backbone of its stability. TCP placed the trust of responsive behavior, i.e. decrease rate if there is congestion, at the end-user and as a result the core network could be kept simple. However as the application needs changed newer rate control schemes were proposed. Moreover, new software advancements have also placed users in a position where they can change their congestion control schemes. As such we now have an Internet which operates with a spectrum of transport protocols, some of which don't even react to congestion indications. Thus, over the years, the trust placed in the end-system to react to congestion indications has been sufficiently weakened.

It has been widely reported that this breach of trust or presence of uncooperative users can lead to TCP unfriendliness and also cause congestion collapse [1], [6]. Moreover, as reported recently and further validated by our results, these uncooperative flows can also force a traffic based denial-of-service to their cooperative counterparts [9], [10]. Also as the network grows and the access pipes get bigger, uncooperative flows will pose a significant challenge before the network providers. This is because of uncooperative flows have the ability to monopolize bottleneck space and their disregard to appropriate congestion responses may cause congestion collapse thus effecting the stability of the Internet. Some architectural responses such as use of AQM schemes, schedulers and pricing mechanisms have been suggested to manage the uncooperative flows [6]. However, use of AQM and schedulers require deployment at all (bottleneck) routers in the network, which is not only expensive but also requires significant network upgrade. These deployment considerations coupled with presence of simple Drop-Tail queueing schemes at all routers in the Internet present us with an interesting question - *What are the appropriate alternate architectural responses for managing a network of un-cooperative users, such that it requires minimal network support ?*

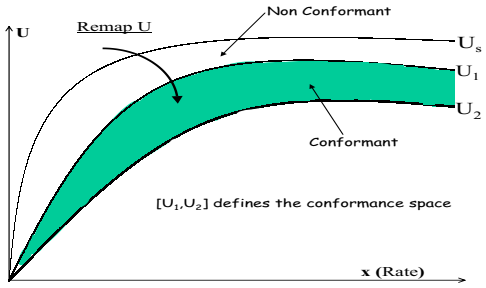


Fig. 1. Mapping a uncooperative user to a conformant space.

In this paper we explore architectural responses for managing the entire spectrum of uncooperative sources at the *edge of the network*. The biggest advantages of the framework presented in this paper are that it is independent of buffer management scheme deployed on the network and works equally well in a dropping or a marking based network. The framework presented in this paper can also be used to distribute rates amongst user's according to some a-priori fair rate allocation, while still allowing users to choose their rate control schemes. Thus this proposal can be used to enforce congestion response conformance e.g. TCP-Friendliness. Moreover, our framework allows for a enforcement of a broader range of congestion response conformance criteria.

The framework presented in this paper follows from the flow optimization model [11], [12], [14], specifically the duality framework of Low etal. [14]. The flow optimization framework is a network-based approach for modeling rate control schemes and computing average sending rates and end-to-end loss probabilities for users. In this paper we describe a user with his rate, x and a utility function, $U(x)$, while a network is identified with link capacities. Thereupon, the users try to maximize their utility functions subject to link capacity constraints and in the process we derive rate control schemes for the users and link price update mechanisms for the network.

In this paper we call users cooperative if their utility functions fall within some a-priori specified target range of utility functions. For example in Fig 1 $[U_1, U_2]$ defines the cooperation boundaries or the target range. We show that through a transparent penalty function transformation the network provider can *remap* the utility functions of the uncooperative users to a target range of utility functions, see Fig 1. Further, this *re-mapping* can be easily implemented at the edge of the network. Moreover, our framework allows users freedom to choose arbitrary concave utility functions or in other words they can pick any rate control scheme [11], [12], [14]. This solution presented in this

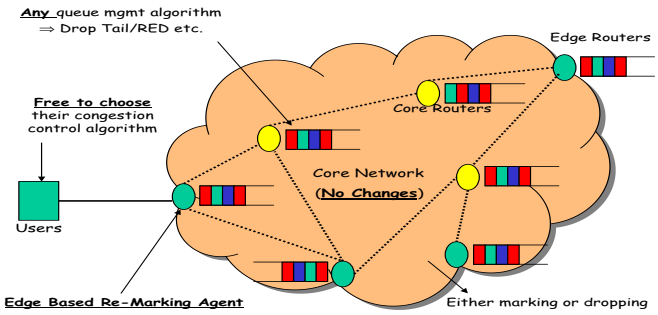


Fig. 2. Model for managing Uncooperative users at Network Edge

paper is attractive because it *does not require any upgrades in the routers* of the network, they function as usual, i.e. they may mark, or drop packets using any buffer management scheme (including Drop-Tail policy). Fig 2 shows the model for policing uncooperative users.

The problem of managing uncooperative users has been actively researched [1], [6], [5], [13], [17]. Router based mechanisms, such as Active Queue Management (AQM) schemes, schedulers and pricing mechanisms, have been suggested for managing uncooperative users in the network. However, use of AQM schemes, schedulers require deployment at all (bottleneck) routers in the network, which is not only expensive but also requires significant network upgrade. Moreover AQM schemes face configuration problems and also lack of deployment of ECN. As a result, they are not deployed and Internet works on simple Drop Tail queueing and the problems due to uncooperative flows persist.

This paper suggests that management of uncooperative flows need *not* be coupled with AQM design and can be simply viewed as an edge network based policing question. Our mechanisms may also be thought of as a new class of "traffic conditioning" techniques [4], where the "conditioning" can be achieved by manipulating either the feedback or packet stream. Moreover, since the users cannot always be trusted with their rate control schemes, the network has to enforce this trust and the network edge is the first place this trust is enforced. Additionally, this function can combined with the other edge based functions like preventing spams, denial-of-service attacks etc.

We have implemented this framework in NS-2 and evaluated it for various single and multi-bottleneck topologies, for both marking and dropping congestion notification policies and also with and without AQM schemes. Our results show that the framework can "re-map" any uncooperative user to co-operative user for a broad range of network scenario. Further, the framework is robust and works well even in the presence of background web-traffic and reverse-path congestion. However, for our our scheme to perform well we need to estimate user's utility function. Towards this end we also outline and evaluate Linear Least Squares Errors (LLSE) and Non-Linear LSE (Least Squared Error) methods for these purposes. Our initial results show that these methods are easy to implement and work well, even with a small sample set or in other words they can quickly characterize sources. The paper also presents results for simple differentiated services which can be derived from the model. Finally, we also compared the performance of CHOKe and BLUE in managing un-cooperative flows.

To summarize, the main contributions of this paper are:

- Proposes an edge-based model for managing uncooperative users.
- The framework is independent of AQM schemes, i.e., it works with both RED or other AQM scheme and Drop Tail queues.
- The proposed framework works well with both marking and dropping as congestion notification policies.
- The framework can also be thought of as a new class of traffic conditioning where conditioning can be achieved by manipulating either the ack or packet stream.
- It suggests that management of uncooperative flows need not be coupled with AQM design.
- The model presented in this paper can prevent traffic based denial of service attacks.

The rest of the paper is organized as follows. In Section II we first define cooperative behavior and then accordingly classify uncooperative flows as open and closed loop protocols. Section III further motivates the need for our work by evaluating the impact of uncooperative flows on AQM and Drop Tail queues. Thereafter, we present our utility function re-mapping model in Section IV. In Section V we describe our implementation and the simulation setups. Section VI presents the results. In Sections VII we describe the utility function estimation while Section VIII we present the effect of estimation errors on the model. We discuss the merits and drawbacks of the scheme in Section X while Appendix D shows how simple differentiated services can be obtained with out model. Finally we present the conclusions and future work in Section XI.

II. Classes of Uncooperative Flows

In this section we will define what we imply by cooperative and uncooperative flows. Using this definition we will classify uncooperative flows and finally we will introduce responsive uncooperative flows.

Shenker, Kelly etal have shown rate control schemes can be defined in terms of utility function [11], [12], [14], [23]. The biggest advantage of this method is that a broad class of rate control schemes can be defined by a single utility function, for e.g. all TCP-Friendly schemes can described by the utility function $U(x) = -1/x$, where $U(x)$ represents the utility function and x the rate.

Definition 1: Flows whose utility function lie within some a-priori specified target range are called cooperative.

For example in Fig 1 any rate control scheme whose utility function lies between $[U_1, U_2]$ is considered cooperative. TCP-Friendliness can be an example of cooperative regime where the target utility function range is $U_1 = U_2 = -1/x$. For reasons of simplicity, in this paper, *we choose TCP as our definition of cooperation.* TCP being the most widely used transport protocol in the Internet further rationalizes our choice. Thus uncooperative flows corresponds to any TCP unfriendly rate control schemes. However, we would like to point out that TCP-Friendliness is just one notion of cooperative flow under our model. We could easily define any other notion of cooperation by suitably choosing a single or range of objective utility functions.

Uncooperative flows can be classified into two categories:

- Unresponsive or Open-Loop flows
  - Do not react to congestion indication.
- Responsive or Closed-Loop flows
  - React to congestion indication by cutting down their rates.

Traditionally by uncooperative flows we have referred to UDP and CBR. These protocols always send data at a constant rate and since they do not use any feedback from the network are generally referred to as open loop protocols. Open loop schemes are used by most of the multimedia and gaming applications e.g. Real Audio, Internet telephony, Quake,Half life etc [8]. These unresponsive flows can be modeled by a constant utility function, i.e. U(x) = constant [23].

Responsive uncooperative flows encompass a larger range of mis-behaving scenarios. Their misbehavior can be defined on basis of their increase policy, (i.e. how they probe the network for available bandwidth) and their decrease policy (or how they respond to congestion indication). Further, these flows are commonly identified with concave utility functions, for e.g. TCP's utility function is $U(x) = -1/x$. Responsive uncooperative flows can also be divided into two sub-categories: one whose utility function does not change with time e.g. $U(x) = -1/x^a$ where a is a constant, while the other class has a time-varying utility function e.g. $U(x) = -1/x^{a(t)}$. In this section we will briefly present uncooperative closed loop schemes derived from time-invariant utility functions. The reader is referred to Appendix C for the time-varying utility function discussions.

Let *x* represent the rate and *R* the RTT. Then we could identify the utility function of any increase/decrease based rate (or window) control scheme with the following relationship

$$U'(x) = \frac{1}{Rxf(x)g(x)} : \ f(x) \geq 0, \ 0 \leq g(x) \leq 1 \qquad (1)$$

where the increase policy, I, and decrease policy are

$$I : \frac{1}{f(x)} \quad D : g(x) \qquad (2)$$

Also it can be easily shown that these utility functions are strictly concave. Utility function of TCP-Friendly is described by $U(x) = \frac{-1}{x}$, any scheme which has $f(x)g(x) \propto x$ will also be TCP-Friendly. Similar conclusions about TCP-Friendliness of these generalized schemes were shown by [16]. Binomial Congestion Control Scheme (BCCS) proposed in [2] is one special case of the above model with

$$f(x) = \frac{\alpha}{x^k}, \quad g(x) = \beta x^l \qquad (3)$$

where α, β, k, l are some constant. The utility function of binomial schemes is given as $U(x) \propto \frac{-1}{x^n}$ where $k + l = n$. TCP-Friendly schemes in BCCS can be defined by $k + l = 1$.

Since in this paper we have chosen TCP-Friendly schemes to be cooperative, the uncooperative BCCS schemes are defined by $k+l < 1$. Further, it can be shown that a more general definition of uncooperative flows would be sub-linearity or $f(x)g(x) < x$. Uncooperative flows can also be obtained by choosing a large value of α ($\alpha > 1$) or a small value of β ($\beta < 0.5$). However for the simulations reported in this paper we use the k and l values to generate uncooperative flows. This is because such flows can be obtained a simple tweaking of the TCP and thus are more likely to to be found on the Internet.

Open and closed loop rate control schemes can have very different impact on bottleneck sharing. While open loop schemes may not always shut-out cooperative flows their closed loop counterpart's affects may be more pronounced. Open loop protocols always send data at a fixed rate which may not always exceed the bottleneck capacity. Thus open loop protocols will always hog bandwidth equal to their sending rate leaving the rest for cooperative flows. On the contrary, closed loop protocols are always looking to absorb whatever capacity is available and as such if a cooperative flow cuts down it's rate these closed loop uncooperative flows will step in to claim that bandwidth.

III. Motivation: Impact of Uncooperative Flows on Existing Buffer Management Algorithms

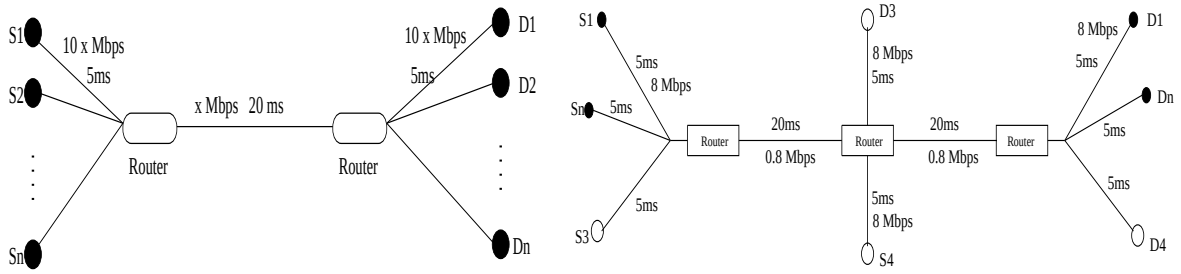
Though many AQM schemes have been proposed to manage uncooperative flows their deployment on the Internet has been lacking because of variety of reasons: configuration problem, lack of deployment of ECN and requirement of significant network upgrade. As a result of these deployment constraints, the present Internet works on simple Drop-Tail queueing. In this section we evaluate the effect on uncooperative flows on the buffer management schemes and motivate the need for our work.

A. Uncooperative Flows and AQM Schemes

Many AQM schemes have been proposed to limit the effect of uncooperative flows. These proposals can be broadly classified into two categories: state-full schemes like FRED [13] etc and stateless schemes like CHOKe [22], BLUE [5]. State-full schemes also include some partial state schemes like RED-PD [17] where states for only the mis-behaving sources are stored. Each of these proposals has it's own merits; stateless schemes are easy to manage while state-full schemes patrol uncooperative flows more efficiently but do not scale. However, given the number of AQM proposals it is beyond the scope of this paper to do an exhaustive performance evaluation across all schemes, hence we will only evaluate CHOKe and BLUE as they represent the stateless alternatives to this work.

We evaluated CHOKe and BLUE on NS-2 on various single and multi bottleneck topologies with different degrees of flow multiplexing. However, we will only present the results for multi-bottleneck scenario. The multi-bottleneck topology is shown in Fig 3 b). For this setup, we define long flow as a flow which traverses both the bottleneck, whereas the short flows are defined as flows traversing only one bottleneck. Since limitations of CHOKe with unresponsive flows has already been outlined in [17], for our simulations we will evaluate CHOKe (and BLUE) with responsive uncooperative flows. For our simulations these uncooperative flows were generated using BCCS with $k + l < 1$. There was one long flow and one short flow on each bottleneck and the short flows were mis-behaving, $k = 0, l = 0.5$. For the AQM settings we refer the reader to Section V.

Fig 4 (a)-(c) plots the throughput of each flow as well as the ideal share from each simulation while Fig 4 (e)-(g) shows the link utilization for the same simulation. Since we have chosen TCP-Friendliness as our definition of cooperation the ideal shares correspond the simulation where both the long and short flows were TCP flows. It can been seen from Fig 4 b) that CHOKe marginally improves the throughput of long flow as compared to that with RED, Fig 4 a). But more importantly this marginal improvement in performance of CHOKe comes at the expense of link utilization, i.e. the link utilization is almost 30% less with CHOKe (Fig 4 e, the thick curve in this plot is the average utilization). On the other hand, BLUE does even worse than RED and the long flow is further penalized as it's



(a) Single Bottleneck Topology (b) Multi-Bottleneck Topology

Fig. 3. Topologies used in the Simulations.

throughput goes down. Moreover BLUE also does not utilize the link efficiently, Fig 4 f), though it's better than CHOKe. Table I shows a similar results when the number of flows on each bottleneck was increased to 10 (5 long and 5 short flows), the bottleneck capacity increased to 10Mbps and a buffer of 150 packets. Again it can be seen that marginal improvement in performance of CHOKe comes at the expense of significantly low link utilization (of 70%). Figure 4 d) plots the results with our framework and shows that our framework can improve fair sharing of the bottleneck without compromising link utilization.

One of the reasons why CHOKe's performance suffers is because it has poor estimate for the aggressiveness of the uncooperative flow. For every incoming packet to the queue, CHOKe picks a random packet from the queue and matches it's header. If the headers match then CHOKe drops both the packets otherwise it probabilistically enques the incoming packet. Thus if the selfish behavior of the uncooperative flows can be classified properly then depending upon the aggressiveness CHOKe can pick n packets from the queue to match the header. Such a method will then greatly improve the fair sharing of the bottleneck. Our proposal does better precisely because of this reason. At the edge of the network we measure the loss probability and rate of the uncooperative users and use it to decide the penalty transformation for the uncooperative flow. In Section IV we will present these arguments in detail.

We also ran simulation with partial network upgrade, i.e. setups where CHOKe was turned on only one bottleneck router while the other bottleneck had Drop Tail queueing. We found performance of CHOKe in partial upgrade to be similar to that of CHOKe on both bottlenecks. However, on a single bottleneck topology CHOKe does remarkably well and the all flows share bandwidth fairly though link utilization remains poor. In yet another set of simulations we enabled ECN on the network and also modified CHOKe to mark packets instead of dropping them. Since our sources were closed loop schemes we expected CHOKe to limit the rates of uncooperative sources. However, the results were most surprising as CHOKe performed even worse than RED. Because of space constraints we are not presenting those results here.

In summary, CHOKe performs remarkably well in patrolling uncooperative users over single bottleneck scenarios. However, it's performance is only marginally better than RED on multi-bottleneck scenarios and it also results in poor link utilization. These wide fluctuations in link utilization suggests oscillations in the bottleneck queue size which in turn cause window (or rate) oscillations. These oscillations are considered harmful as they increase jitter and make any kind of buffer or resource provisioning harder. Thus CHOKe and BLUE cannot always patrol uncooperative flows, especially under multi-bottleneck scenarios and also result in poor link utilization.

Type	Ideal	RED	CHOKe	BLUE
Long Flow (S1-D1)	132	82	95	63
Short Flow (S3-D3)	340	390	300	430

TABLE I

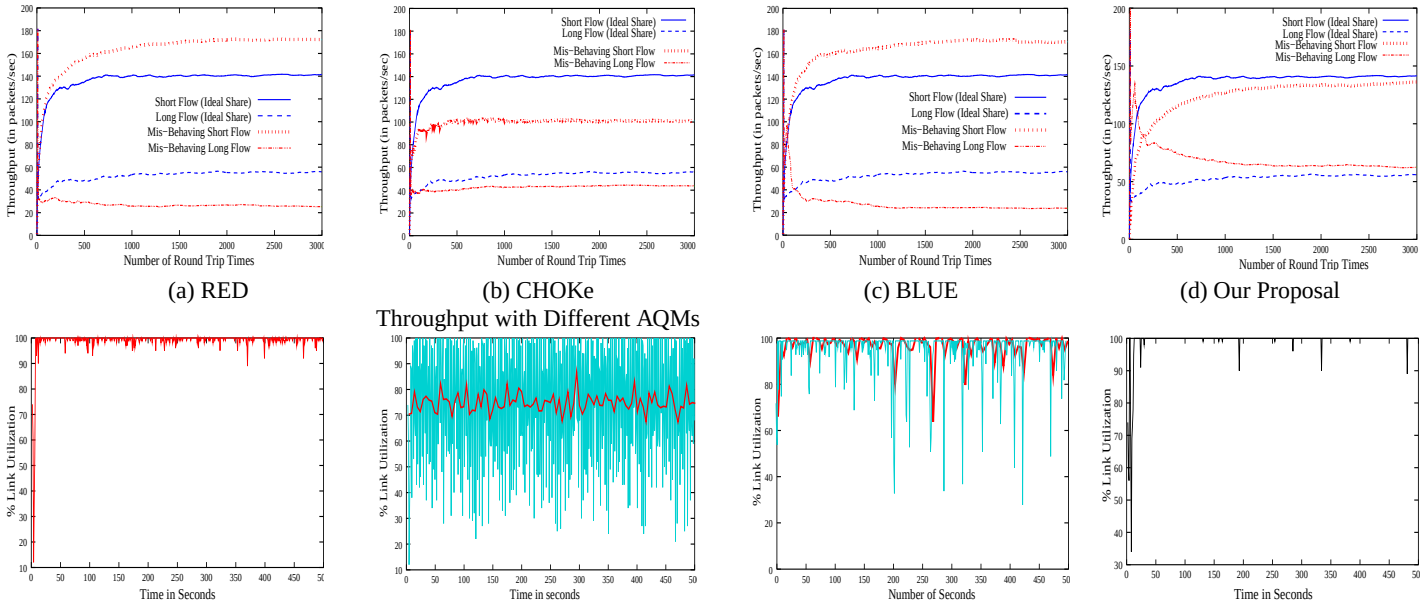
PERFORMANCE OF AQM SCHEMES: COMPARISON OF THROUGHPUT (PACKETS/SEC) OF DIFFERENT AQM SCHEMES ON A MULTI-BOTTLENECK TOPOLOGY WITH 10 FLOWS ON EACH BOTTLENECK.

B. Uncooperative Flows and Drop-Tail Queues

Since AQM schemes require significant network upgrade, network providers have not turned on these proposals on the routers. As a result, the present Internet still works with simple FIFO queuing. In this section we will present the impact of uncooperative flows on a network of Drop-Tail queues.

Fig 5 shows the shares of a long and short flow on a multi-bottleneck topology. The simulation set-up is similar to the one described above with one long and one short flow. It can be seen from figure 5 a) TCP-Friendly is almost shut out by the mis-behaving flows, who now get all the bandwidth. Not only is the TCP-Friendly flow is forced into multiple timeouts (23 for this case) but these timeouts occur with very small windows and are often back to back. This result is also indicative of traffic based denial-of-service attacks on cooperative users. Similar results were obtained with a higher multiplexing (of flows) and with single bottleneck scenarios but due to space constraints are not reported here.

To summarize, with DropTail queues uncooperative flows may get significant share of the bandwidth, almost to the extent of shutting out cooperative flows. This might also be construed as denial-of-service to the TCP flows [9], [10]. Thus given that AQM proposals are yet to be deployed on the network and presence of simple FIFO queueing uncooperative flows not only get more than their fair share but may also lead to denial-of-service to conformant flows. As such we are presented with the following question: *What are the appropriate alternate architectural responses for managing a network of un-cooperative users, such that it requires minimal network support ?* Moreover, *as ECN and AQMs are eventually deployed on the network, do these solutions still work ?* In the following section we present our framework which addresses these questions.



(a) RED (b) CHOKe (c) BLUE (d) Our Proposal

Throughput with Different AQMs

(e) RED (f) CHOKe (g) BLUE (h) Our Proposal

Link Utilization with Different AQMs

Fig. 4. **Multi Bottleneck**: Throughput of long TCP-Friendly flows and short uncooperative flows (k=0,l=0.5) flows with different buffer management schemes.

IV. Re-marking Framework for Managing Non-Conformant Users

Consider a user *s*, who is described with the help of his rate, x_s, a utility function U_s and the Set of links which he uses, *L(s)*. Let the network be identified with links *l* of capacity C_l and the set of users using a link, *l*, be given by *S(l)*. Further, assume that the rates are bounded and that the utility functions are *increasing* with rates and *strictly concave*. Then the flow optimization problem can be defined as users trying to maximize their individual utility functions and the network trying to maximize the resource allocation subject to link capacity constraints. The problem is formally defined as [14]:

$$maximize \quad \sum_{s \in S} U_s(x_s) \tag{4}$$

$$subject\ to \quad \sum_{s \in S(l)} x_s \quad \leq \quad C_l, \quad \forall l \tag{5}$$

for all $x_s \geq 0$. The solution to this problem is given by the following update rules

$$x_s(t) \quad = \quad U_s^{'-1}(\sum_l p_l) \tag{6}$$

$$p_l(t+1) \quad = \quad [p_l(t) + \gamma(\sum_{s \in S(l)} x_s - C_l)]^+ \tag{7}$$

where p_l are the dual variables of the problem and can be identified as penalties, price or link loss probability [14], [12], [11].

From the above update rules it follows that both the rate control algorithm and the equilibrium rate can be associated with the utility function user chooses to maximize (equation (6, 7)). However, given that the *same price* is being communicated by the network, the equilibrium rates can be different, but are still fair from the utility function perspective. Thus a bias in equilibrium rates can be created by choosing two different utility functions.

Lets assume that the users are maximizing the utility function U_s and that the network decides to map every user to U_{obj}. Then if this mapping were successful, the rate updation algorithm (and thus equilibrium rates) of the users would be

$$x_s(t) = U_{obj}^{'-1}(p^s)$$

where $p^s = \sum_{l \in L(s)} p_l$ or the end-to-end price. However, the actual rate control algorithm for users is still given by equation 6. Now suppose that instead of giving an end-to-end price of p^s the network gives the user s the price p^s_{new}, where

$$p^s_{new} = U_s^{'}(U_{obj}^{'-1}(\sum_{l \in L(s)} p_l)) \tag{8}$$

If the user s uses this transformed end-to-end price then his rate updation algorithm algorithm becomes

$$x_s \quad = \quad U_s^{'-1}(p^s_{new}) \tag{9}$$

$$\quad = \quad U_s^{'-1}[U_s^{'}(U_{obj}^{'-1}(p^s))] \tag{10}$$

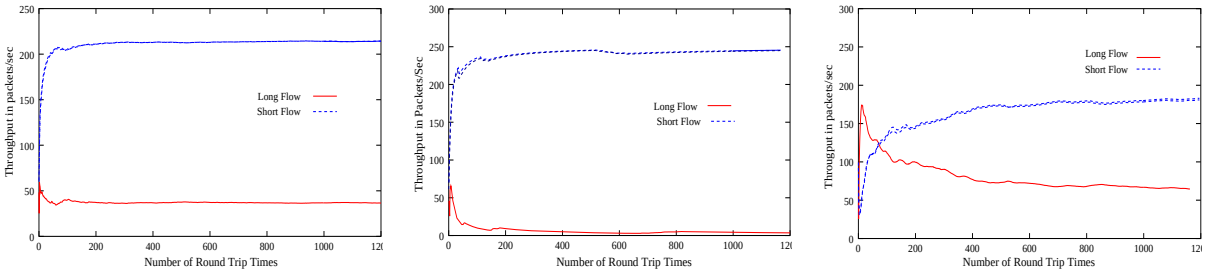
$$\quad = \quad U_{obj}^{'-1}(p^s) \tag{11}$$

Thus it follows from the above equation that by communicating a different price we have transformed the user's utility function from $U_s(x)$ to $U_{obj}(x)$. This transformation can be implemented using the following update rules

$$p_l(t+1) \quad = \quad [p_l(t) + \gamma(\sum_{s \in S(l)} x_s - C_l)]^+ \tag{12}$$

$$x_s(t+1) \quad = \quad U_s^{'-1}(p^s_{new}) \tag{13}$$

Theorem 1: Given the non-negativity constraint on x_s and p_l and strictly concave utility functions U_s and U_{obj}, the new update algorithm as defined in equations (12, 13) still converges to the optimal point.



(a) Ideal Share (b) Impact of Uncooperative Flow (c) With our Proposal

Fig. 5. **Performance of Drop-Tail Queueing**: Throughputs (in pkts/sec) for two competing flows on a multi-bottleneck setup, long flow is TCP Friendly while the short flows are uncooperative.

Proof: : See Appendix A ■

Since all the utility functions are strictly increasing and concave this update rule does not change the network objective, still minimizes the dual function and converges asymptotically. Moreover, the above update rule also does not change the core network, as we retain the price update rule as proposed in [14]. Further, the price being communicated to the user can be updated at the edge. We now state the algorithm for the edge re-marker as

Edge Marker's Algorithm:

- For each source, receive from the network the total price for the source's traffic as $p^s(t) = \sum_{l \in S(l)} p_l(t)$.
- Recalculate (or Re-mark) the new price for the source as

$$p^s_{new} = U_s^{'}(U_{obj}^{'-1}(\sum_{l \in L(s)} p_l)).$$

- Communicate this *re-marked* price to the source.

The update algorithm for the network and the source are given by equation (12) and (13) respectively. Finally, we will conclude this section by the following two theorems on the convergence of the the algorithm.

Theorem 2: Assume that utility functions, U_s, are increasing, strictly concave and continuously differentiable, and their curvature is bounded away from 0. Then starting from any initial rates in the interior of *X* and prices *p(0)* $\geq$ *0*, every accumulation point (x^*, p^*) of the sequence (*x(t)*, *p(t)*) generated by the above algorithm and equations (12,13) is primal dual optimal.

Proof: See Appendix B. ■

Theorem 3: The rate of convergence of the edge re-marker's algorithm is given by the smallest eigen vector of ABA^t where A is the routing matrix and B is diag$(U_{obj}^{'-1}(p^*))'$ and p^* is equilibrium price.

Proof: Will be outlined in a Tech-Report. ■

V. Implementation and Simulation Setup

We implemented the edge based re-marker in the NS (Network Simulator). For a marking based network, the edge based re-marker was placed on the reverse path (i.e. on the reverse access link of the user) and re-marked the ACKs. However, for a dropping based network, we configured the edge-based re-marker on the exit router in the forward path.

The edge re-marker also estimated the loss rate for each flow and subsequently used it to re-mark the ACKs. For the purposes of estimating losses, we used Exponential Weighted Moving Average (EWMA) and the Weighted Average Loss Indication (WALI) methods of Equation-based rate control algorithm [7]. We updated these loss indications every RTT and we have assumed that the network knows the RTT of the flows. We also assumed that we know the utility functions of all the flows. In this paper we present the results for EWMA based loss-estimator. Similar results were obtained with WALI based estimator. For EWMA based system we gave 60% weight to the history, while with the WALI based estimator we measured samples over 8 windows to estimate losses. A more detailed discussion on the merits and demerits of these schemes can be found in [7]. For our simulation we used the congestion control and loss recovery mechanisms of TCP New Reno. Also in this paper, we disabled the delayed acknowledgments option. The maximum advertised window is set sufficiently high so that it does not constrain the actual window. We plot the throughput of competing flows in packets/sec, averaged over 20 round-trip times.

Figure 3(a) shows the single bottleneck topology used in the simulations. The access links were configured at a rate 10 times greater than that of the bottleneck link. All the links use Random Early Drop (RED) queues with min thresh and max thresh set as buffer/3 and 0.8*buffer respectively, where buffer is the total bottleneck buffer length. Further, the weight was set as 0.002 and the marking probability for RED was set to 0.1. The RTT was 60ms and the packet size 500B. For simulations with BLUE, the probability increment and decrement were set to 0.0025 and 0.00025 respectively. Further the hold time was set to 100ms.

Figure 3(b) shows a multi-bottleneck topology used in the simulation. The bottleneck buffer was set to 25 packets. We also evaluated our framework for another multi-bottleneck setup of bottleneck link of 10 Mbps, access link of 100 Mbps and a buffer of 150 packets. The link delays were kept the same. RED minimum and maximum threshold settings were similar to those of single bottleneck. Also for all the simulation setups (single or multi-bottleneck) the access link rate are always 10 times greater than that of the bottleneck link.

VI. Results

In the following sections we present our simulation results. Our simulation objectives can be stated as

- Validate the model with single and multi-bottleneck topologies with varying degrees of (flow) multiplexing.

- Examine the robustness of the model in presence of background (web) traffic and reverse path congestion.
- Verify if the model works with dropping as a congestion notification mechanism.
- Validate if it can work with and without AQMs. Specifically, evaluate its performance on a network of Drop Tail queues.
- Substantiate and test how to estimate utility functions.
- Test the sensitivity of the model with respect to inaccurate RTT and utility function estimates.

In the results presented in this section the bias due to large RTT persists. Removing this bias was not a design goal of this paper. However, this is not a limitation of our work either. Removing bias against long RTT flows will imply a Max-Min sharing of the bottleneck. This can be achieved in our framework by simply choosing an objective function as $U_{obj}(x) = \lim_{N\to\infty} \frac{-1}{x^N}$. This essentially follows from the argument that a particular form of fairness is associated with every utility function, for e.g. $U(x) = log(x)$ represents proportional fairness. We refer the reader to [18] for more details.

A. ECN Enabled Network

In this section we will evaluate the performance of the framework when marking can be used a congestion indication. For the results presented in this section RED was configured on all routers and it marked packets. We extensively tested our framework with various single and multi-bottleneck topologies and with different kind of selfish schemes. However, due to space constraints, in this section we will only present some results with multi bottleneck topologies.

A.1 Multi Bottleneck Topology

Figure 3 b) shows the multi-bottleneck topology used for simulations reported in this section. We define long flow as a flow which traverses both the bottleneck, whereas the short flows are defined as flows traversing only one bottleneck. In this simulation setup (0.8Mbps, 25 packet buffer), we first measured the optimal rate allocations when all the flows (long and short) are TCP friendly and plot them in 6 a). As expected, the short flows grab more share of the bottleneck because they have smaller RTTs and go through a single bottleneck as compared to the long flow. We then changed the short flows to be uncooperative (k=0, l=0.5) and plot the result in 6 b). The effect of mis-behavior is more pronounced in this case as the uncooperative flows are trying to shut out the TCP friendly flow. However, when we used our model to re-mark the uncooperative flows we see that (figure 6 c)) the flows now share the bandwidth fairly. More importantly, we see that the result in figure 6 c) is very similar to 6 a), i.e., *we have successfully mapped the utility function of the non-cooperative flows*.

In figures 6 d), e) and f) we plot the results for a multi-bottleneck topology (10Mbps, 250 packets buffer) where on each bottleneck there are 5 TCP Friendly flows and 5 uncooperative flows (k=0, l=0.5). Figure 6 (d) plots the throughput of long and short flows, if they all were TCP Friendly. As expected the longer flows get a smaller share of the bottleneck than the shorter flows. In Figure 6 (e), we changed the shorter flows to act as uncooperative flows and plot the throughput, and it can be seen that the uncooperative shorter flows conveniently beat down the TCP friendly flows. However, in presence of re-marking, (Figure 6 (f)) the uncooperative flows are conveyed higher price by the edge-re-marker and thereby share the bottleneck more favorably with the longer flows. Once again, we see that *re-marking tends to achieve the same performance as those as if all the flows were TCP Friendly*.

A.2 Background Traffic

In this section we evaluate the framework in presence of noise-like mice traffic. HTTP sources were added to the persistent uncooperative and conformant sources. Each http page sends a single packet request to the destination, which then replies with a file of size which was exponentially distributed with 12 1Kb packets. After a source completes this transfer it waits for a random time, which was exponentially distributed with a mean of 1 second and then repeats the process.

We tested our framework for both single and multi-bottleneck simulation setups with different levels of noise where represents the bottleneck bandwidth occupied by the background (or http source in this case) flows. For both the setups we varied the noise level case from 15% to 65%.

Figure 7 shows the results of a multi-bottleneck simulation with 10 competing persistent flows and of these, 7 flows were TCP Friendly while the remaining 3 where uncooperative (k=0,l=0.5). The bottleneck bandwidth for this simulation was 10Mbps and a buffer of size 150 packets. 80 http sources were added to generate 65% noise (ie the http sources occupied 65% of the bottleneck bandwidth). shows where the noise traffic is 65%. From the results it can be concluded that re-marker still manages to efficiently patrol uncooperative users. This thus shows the robustness of the scheme, even when sufficiently high (65%) noise is present in the network.

A.3 Cross Traffic

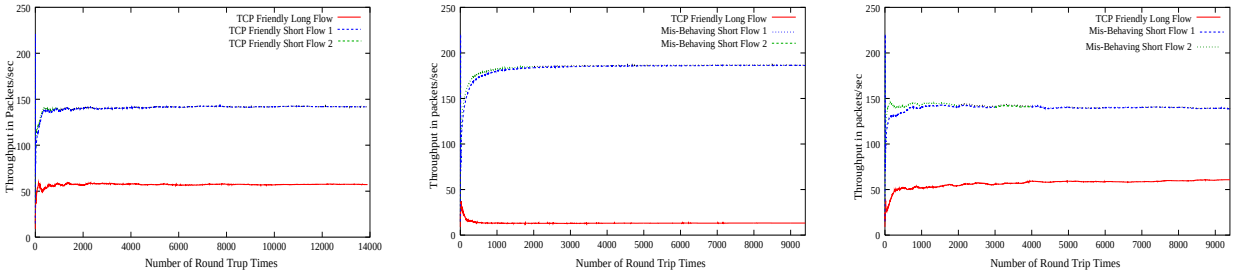
In this section we present the results for our penalty function transformer when two way traffic is present. We evaluate this scenario with the multi-bottleneck topology, where we have 5 TCP Friendly long flows and 5 uncooperative (k=0, l=0.5) short flows on each bottleneck. Additionally, on the reverse path, there are 5 TCP Reno flows on each bottleneck. The bottleneck bandwidth for this simulation was 10Mbps and a buffer of size 250 packets. Re-marking, once again achieves equitable sharing of the bottleneck (as shown in Figures 8 (a) and (b)).

B. Performance Evaluation on a Network of Drop-Tail Queues

Up till now we have discussed the uncooperative framework with re-marking, i.e., we have assumed that ECN support is available in the network. In this section, we look at the alternative scenario, when drops are used to convey congestion penalties. Further, we assume that the network operates with Drop-Tail queues only. Again, we evaluated the performance of the model with varying degrees of multiplexing but only present some results for both single and multi bottleneck topologies.

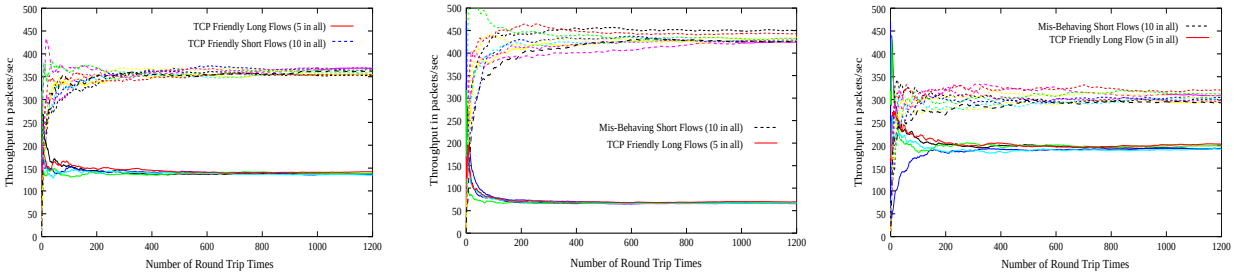
B.1 Single Bottleneck Topologies

We present the result with a single bottleneck of 0.8Mbps and access links of 8Mbps for 2 competing flows. One of the flows is TCP-Friendly while the other is misbehaving flow (with k=0, l=0.5). Both the flows have same RTT of 60ms. Figure 9 shows the results of with and without the re-marking framework. It can



(a) Ideal Share (b) No-Remarking (c) With Remarking

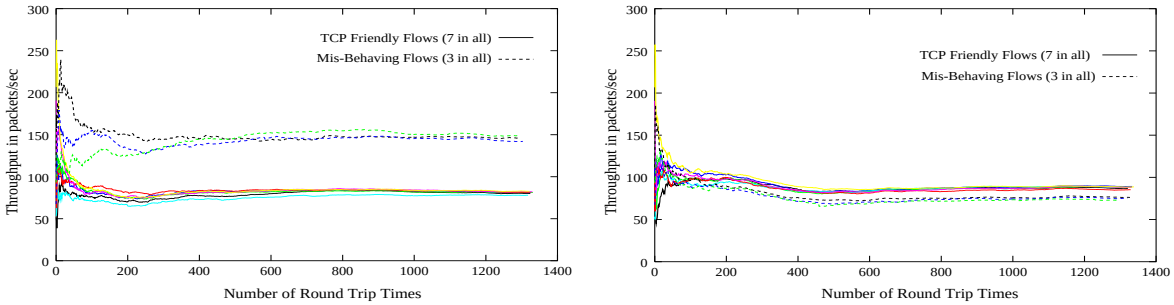
2 Long, 2 Short Flows on Each Bottleneck



(d) Ideal Share (e) No-Remarking (f) With Remarking

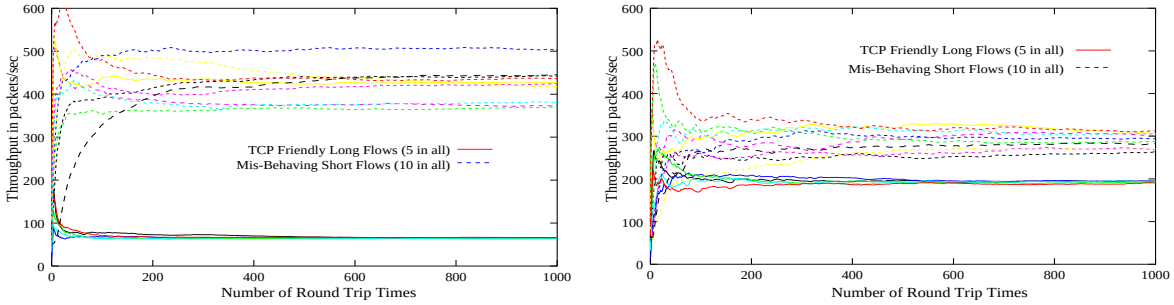
5 Long, 5 Short Flows on Each Bottleneck

Fig. 6. **Multi Bottleneck**: Throughputs (in pkts/sec) for competing flows (2 and 10), where the long flows are TCP Friendly while the short flows are Uncooperative with (k=0, l=0.5). Ideal bottleneck shares for the long and short flows for 2,10 competing flows are plotted in figures (a) and (d) respectively.



(a) No Re-Marking (b) Re-Marking

Fig. 7. **Background Traffic**:Throughputs (in pkts/sec) for 10 competing flows in a single bottleneck topology, where 7 flows are TCP Friendly while the other 3 are Uncooperative with (k=0, l=0.5) **with 65% noise**.



(a) No Re-Marking (b) With Re-Marking

Fig. 8. **Cross Traffic**: Throughputs (in pkts/sec) for 10 competing flows in a multi-bottleneck topology, where on each bottleneck there are 5 TCP Friendly flows and 5 Uncooperative with (k=0, l=0.5), with two-way traffic.

be seen from the figure 9 that in absence of re-marking the uncooperative flow gets most of the bottleneck share. However, when we start re-marking the misbehaving flows this bias against the TCP-Friendly is reversed. Interestingly, as seen from the figure 9, TCP-Friendly flow gets a better share of the bottleneck. This is because unlike marking, dropping is a stricter means to convey congestion notification as it can lead to uncooperative flow timing out and consequently the misbehaving flow suffers.

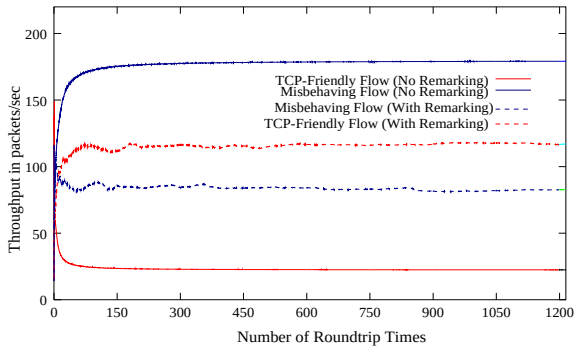


Fig. 9. **DropTail Queues**: Throughputs (in pkts/sec) for two competing flows in a single bottleneck scenario with DropTail queues throughout the network, with and without Re-Marking. One of the competing flows is TCP Friendly while the other is uncooperative.

B.2 Multi Bottleneck Topology

Fig 5 presents the results with a multi-bottleneck topology with two flows on each bottleneck. In this simulation the long flows were cooperative (or TCP-Friendly) while the short flows were uncooperative. Fig 5 (a) plots the ideal share, which corresponds to the scenario where both long and short flows are TCP-Friendly. When these short flows are replaced by uncooperative flows, $k-0, l = 0, 5$ it can be seen that these short flows shut out long flows, see Fig 5 (b). In other words uncooperative sources are creating a traffic based denial of service to the TCP flows. However, when we re-map the uncooperative flows, it can bee seen, Fig 5 (c) that the bottleneck is shared fairly now. Moreover, the long flows get more than their fair share, which is because dropping sometimes forces uncooperative flows into timeouts.

B.3 Background Traffic

C. Background Traffic

In this section we evaluate the robustness of our framework in presence of noise-like mice traffic. The reader is referred to VI-A.2 for details about mice traffic generation. We used a single bottleneck topology and used different level of flow multiplexing to evaluate the effect of background traffic on the performance of a DropTail queue network with and without remarking. However we only report the results for the case where there were 10 persistent and of these, 7 flows were TCP Friendly while the remaining 3 where uncooperative ($k = 0$,$l = 0.5$). The bottleneck bandwidth for this simulation was 10Mbps and a buffer of size 150 packets. Also in this setup we increased the noise sufficiently high to validate the robustness of the scheme in presence of many flows and noise. Figures 10 a) and 10 b) plot the results for the cases where the noise traffic is 65% (or 80 http sources), i.e. mice traffic occupied 65% of the bandwidth. Figure 10 b) shows the robustness of the scheme when sufficiently high (65%) noise is present in the network and the re-marker still manages to efficiently patrol uncooperative users.

VII. Estimating the Utility Function

The framework presented in this paper works well if the network has some information about the rate control scheme being used by the uncooperative user. Essentially what we need is a relationship between the rate and the loss probability. This is then used to compute the re-mapping or re-marking function, as specified in equation (6). Moreover, this relationship between rate and loss probability also quantifies the utility function of the user. With this introduction we will now briefly explain how we can estimate the utility function of a flow.

In this paper we have chosen BCCS schemes as uncooperative users. These schemes can be described by their exponent, *n*, by the following relationship

$$U(x) \propto \frac{-1}{x^n} \tag{14}$$

Thus for describing these class of uncooperative flows we just need to estimate the parameter, *n*. Consider the following relationship between rate and the loss probability

$$U_s^{'}(x_s) \propto p \tag{15}$$

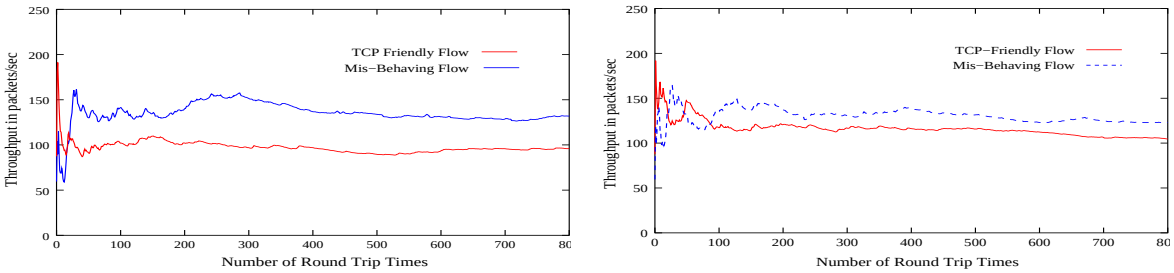
$$p \propto \frac{n}{x^{n+1}} \tag{16}$$

$$log(p) = log(nK) - (n+1)log(x) \tag{17}$$

where *K* is some constant. From equation (17), it follows that estimating the parameter *n* is nothing but a regression analysis. Thus to estimate the utility function all we need is measure of the throughput *x* and the loss probability *p*. These can be calculated by either sampling the packet stream (at the egress) or the ack-stream. Then, given the loss and throughput samples, Linear Least Squared Errors (LLSE) method could be applied to estimate *n*, which is nothing but the slope of the least-square fit. Moreover, the intercept of the least-squared fit gives us the ratio of the increase and decrease parameters. This is because the relationship between loss probability and rate represents the throughput formula. Since we already know the exponent, n, and supposing we know the RTT, R then it can be shown that for BCCS schemes $nK = \frac{\alpha}{\beta R}$. Thus this simple estimation technique can also be used to re-map uncooperative flows which are derived from changing the increase and decrease parameters.

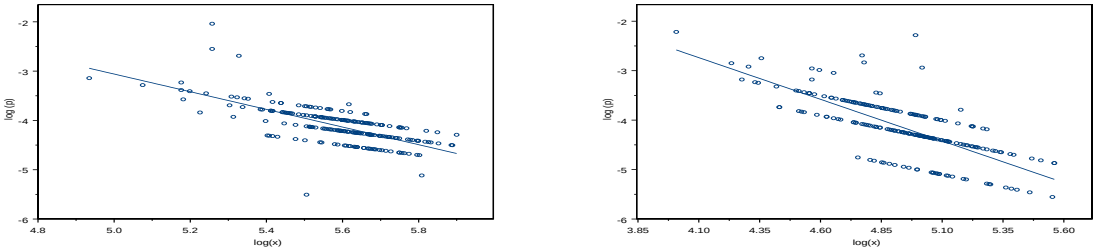
Now we elaborate on our efforts to estimate the utility function of the misbehaving user. We have assumed that the identity of misbehaving user is revealed to us. At the edge router, we can collect samples of throughput (packets sent over time) and loss rate and a time-series can be constructed. For estimating the utility function we constructed three time series with bin sizes of 0.5, 1.0 and 2 seconds, where in each bin we measured the number of packets sent and the loss rate for that bin. Figure 11 shows the results of a simulation of 2 flows, one TCP and the other a uncooperative flow with $k = 0, l = 0.5$ competing on a single bottleneck (see 3 a), with the bin size of 2 seconds. The bottleneck capacity was 0.8Mb, the access links of 8Mb and the bottleneck employs RED with a buffer size of 25 packets. Figure 11 a) and b) show the estimation results for the uncooperative and the TCP flow respectively. The slope of the graph in each case measures *-(n+1)*, where *n* is the exponent. For the uncooperative flow we estimated the exponent to be approximately 0.6 (the slope of the graph is 1.5) while the actual value of n was 0.5. Similarly for the TCP flow we estimated the exponent to be approximately 0.8 instead of 1.0. In the following section we will try to quantify the effects of inaccurate estimates. Specifically, we will use these estimated exponents to remark the uncooperative flows.

Although LLSE is simple and has faster convergence it suffers from implementational complexities. Specifically its time



(a) No Re-Marking (b) With Re-Marking

Fig. 10. **Background Traffic**:Throughputs (in pkts/sec) for 10 competing flows in a single bottleneck topology, where 7 flows are TCP Friendly while the other 3 are Non-Conformant with (k=0, l=0.5) **with 65% noise**.



(a) Estimation of Utility Function: Uncooperative Flow (b) Estimation of Utility Function: TCP Flow

Fig. 11. **Estimation of Utility Function** for 2 competing flows in a single bottleneck topology, where one flow is TCP Friendly flow while other is Uncooperative with (k=0, l=0.5).

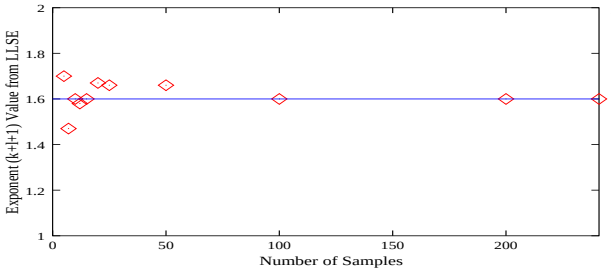


Fig. 12. Exponent Value Vs Sample Size: As the Sample Size increases estimation gets better. Even Smaller samples give good estimates. Motivates use of RLS.

complexity is $O(M^2)$, where M is the order of the filter [21]. Moreover, it needs the entire data set a-posteriori to estimate the parameters. However, there exist LLSE schemes which compromise the implementation complexity with convergence. Recursive Least Squares (RLS) [21] is one such scheme. It has a time complexity of $O(M)$ and it can recursively use new data with some incremental work. We will now motivate the need for using RLS and show that good estimates can be gathered with small sample set and then the estimates can be improved by further measurements. Moreover, with RLS the new measurements can be incrementally consumed.

A point of concern in estimation is - *how many samples are needed to characterize a source ?*. We will address this concern using the example presented above. We took the time-series used in previous examples and broke it into smaller series. This thus gives us the results for estimation with smaller sample space; the new sample sets corresponded to 5, 7, 10, 12, .., 250 samples. In Fig 12 we have plotted the exponent value versus number of samples for the uncooperative user. As the figure shows even with 5 samples the exponent, n, is detected to be 0.7 and as the sample size increases the exponent value fast approached the true value. This suggests that using estimation schemes like RLS will make the estimation task easier.

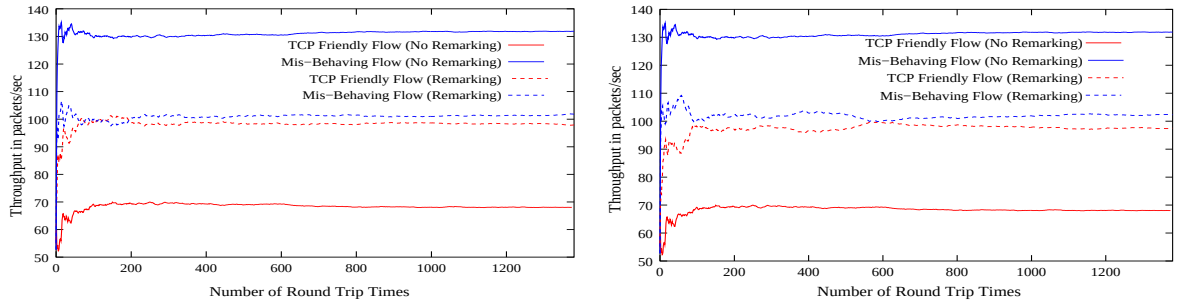
VIII. Sensitivity Analysis of Framework

In this section we investigate the effect of inaccurate estimation. Specifically we test the validity of the model in presence of inaccurate utility function and RTT estimates. RTT-estimation is needed for updating our congestion indication estimations (which is similar to the one presented in [7]) while utility function estimation is needed for re-mapping. Our simulation results suggests that the inaccurate RTT estimates don't have a pronounced effect on the re-mapping, at most they might slow the convergence (to the objective utility function). However, large errors in estimation of utility function may over-penalize the non-conformant sources.

A. Effect of Inaccurate RTT Estimate

In all our previous simulations we assumed that the network knows the RTT of the flows. We used these RTT estimates to update our congestion indication estimations. For the results presented in this section we looked at two cases, one when we under-estimated the RTT and the other when we over-estimated it. We present the results with a single-bottleneck of 0.8Mbps, 25 packet buffer and 2 competing flows.

Figure 13 a) shows the results when the RTT was under-estimated as 0.05 (instead of 0.06). Figure 13 b) shows similar results when we over-estimated the RTT as 0.07. The figures suggest that inaccuracy in RTT estimates alters the convergence speed to the optimal point; a larger value of RTT will slow down the convergence while a smaller value will increase the convergence. However, from both the results its easy to see that the



(a) Under-estimation of RTT (rtt=0.05s)

(b) Over-Estimation of RTT (rtt=0.07s)

Fig. 13. **Inaccurate RTT Estimates**: Throughputs (in pkts/sec) for 2 competing flows in a single-bottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with (k=0, l=0.5), when network has inaccurate RTT estimates.

effect of inaccurate RTT estimation is not pronounced and the model works well. We ran simulations with higher degree of multiplexing and came to a similar conclusion. However, we do not present those results here.

B. Effect of Inaccurate Utility Function Estimate

In this section we evaluate the model's sensitivity to utility functions; when the utility functions are under-estimated and when they are over-estimated. Under-estimation here refers to the case when we estimate the utility function to be less aggressive than it really is, i.e. when $k + l$ values are reported to be larger than the actual values. Over-estimation refers to the case where we report the flow to be more aggressive than it really is, i.e. $k+l$ values are reported to be smaller than the actual values. We present the results with a single-bottleneck topology (figure 3 a)) for 2 flows.

Figure 14 a) shows the results when the utility function was under-estimated as 0.6 (instead of 0.5). Figure 14 b) shows similar results when we over-estimated it as 0.4. It can be seen from the results that the model is sensitive to inaccurate estimate of utility functions. When we under-estimated the utility function ($k + l = 0.6$) the model didn't penalize the uncooperative flow much, and as such it still garners more bandwidth than the TCP flow. In the case of over-estimation ($k + l = 0.4$) we see that the network penalizes the uncooperative flow more and consequently brings it share down below the TCP Friendly flow.

However, the estimation errors pointed out in the simulation are large (the error is 20% since we estimate the $k + l$ values as 0.5±0.1). We evaluated the model for two other error estimates, 10% and 5%. As expected, as the estimation error decreases the model starts to get better. Further we found that for estimation errors of more than 10% the model does not penalize (or over penalizes) the uncooperative flow much and it consequently has a larger (or smaller) share at the bottleneck. We evaluated the model for different simulation setup, with 10 flows (5 uncooperative, 5 TCP-Friendly) and came to a similar conclusion.

IX. Unresponsive Flows

Unresponsive flows can be identified with constant utility function, i.e. $U(x) = constant$ [23]. However, to re-map a utility function we require of strictly concave utility function. This is because then these functions can be uniquely inverted and equation (8) can be applied to them. However unresponsive flows break this strict concavity requirement and as such our re-mapping does not apply to them.

In order to police these sources we suggest an approach similar to one described in [17]. Let us denote by R the target RTT and by p the loss rate seen at a router by the responsive flows. Then using an approximation for the TCP throughput formula the equivalent TCP steady state rate for a flow with round-trip R seconds and a loss rate of p is given by

$$x^* = \frac{1.5}{R\sqrt{p}} \tag{18}$$

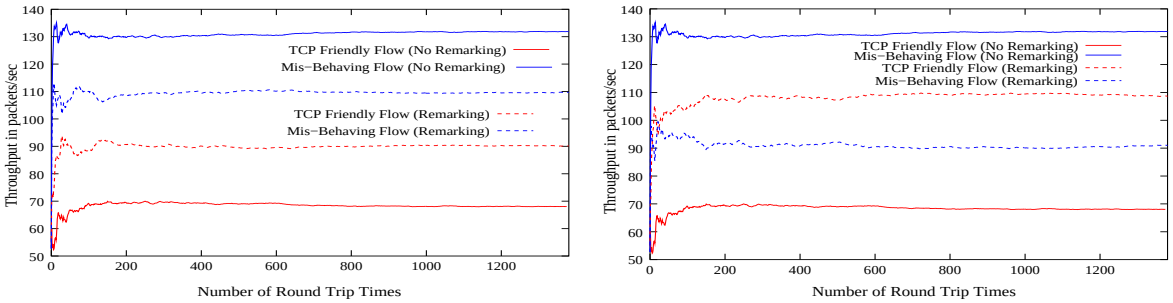
The network provider can use this target rate, x^*, as the fair rate allowed for the unresponsive flows and then can appropriately decide where to enforce this rate. Since unresponsive flows do not react to congestion indications we will need a traffic shaper to enforce the fair rate. We evaluated this proposal on a single bottleneck topology and our initial results suggest that for small number of flows in the system this method can lead to fair sharing of the bottleneck. However, because of space constraints we do not present the results here.

X. Limitations of the Model

In this paper we have proposed an abstract model for modeling and managing uncooperative flows. This paper suggests that management of mis-behaving flows need *not* be coupled with AQM design and can be simply viewed as an edge network based policing question. We believe that this design arguments has clear incentives and needs to be pursued further. In this section we will debate the merits and the limitations of the model.

The scheme proposed in this paper is sensitive to loss and utility function estimation. Techniques for loss estimation have been discussed in detail in [7]. However, these techniques are still empirical at best and need to to be evaluated further. Estimation of utility function also has a significant impact on the performance of the uncooperative congestion control framework. Though we have outlined and evaluate LLSE and Non-Linear LLSE methods for estimating the utility function we need to expand our work in this direction to include more efficient lattice based and recursive estimation techniques like RLS. For a more general utility function as defined in equation (1) we could use the Non-Linear Least Squared to detect a power-series in *x and n*. We are currently working on this estimation problem.

In this paper we do not present any methods for detecting uncooperative users, rather we assume that their identity is given to us. We believe this is a separate but very important issue. Recently various hashing, filtering and other similar proposals have been put forward for identifying misbehaving flows [19],



(a) Under-estimation of Utility Function: In-sufficient Re-marking, Misbehaving Flow Still Wins

(b) Over-Estimation of Utility Function: Excessive Re-marking, Misbehaving Over Penalized (Loses to TCP)

Fig. 14. **Inaccurate Utility Function Estimates, 20% Estimation Errors**: Throughputs for 2 competing flows in a single-bottleneck topology, where one flow is TCP Friendly flow while other is Non-Conformant with (k=0, l=0.5), when network has inaccurate estimates of source's utility function.

[5], [3]. Also the increasing use of Smart Sampling in Netflow benefits us as its puts a first level filter on high bandwidth flows. This then considerably reduces the number of flows to be monitored and thereafter we can apply any of the methods presented above to detect uncooperative flows. However, we would also like to point that the objective of this paper was not to identify uncooperative flows but how to manage them.

Path asymmetry is another issue which we need to look in detail. If a single exit router is used by the flow then the model is immune to path-asymmetry problems in the network. Both unique entry or exit routers is generally true in the present Internet. However, if different exit and entry routers are used for any flow then we need to study the effect of path-asymmetry.

Despite these limitations, we believe our work is a first step in modeling and managing uncooperative flows at the edge of the network by keeping state for only the mis-behaving flows. The incentives for such an approach are clearly high: the model proposed in this paper can be implemented at the network edges, works well with both dropping based or an ECN enabled network and more importantly is independent of the buffer management scheme deployed on the network. Further, this framework presented in this paper can be considered as a general traffic conditioner. Packeteer boxes, deployed widely on the Internet, already do a similar, though limited, traffic conditioning of reducing congestion by pacing the acks [20] and work well with large number of flows.

XI. Conclusions and Future Work

In this paper we have proposed an abstract model for modeling and managing uncooperative flows. We show that we can describe the uncooperative (as well as cooperative) flow by utility function. This utility function space can thus be partitioned to identify uncooperative flows. Within the flow optimization framework, we propose an edge based model to map a uncooperative user's utility function, U_s, to any target range of objective utility function, U_{obj}. This re-mapping can be carried by a transparent manipulation of the congestion penalties.

The model presented in the paper can be implemented at the network edge and as thus is incrementally deployable. Moreover, this model does not require any changes in the core routers. However, we do need to store some information about uncooperative sources at the network edge. The model presented in this paper is also independent of the buffer management algorithm, i.e. it works not only with any AQM scheme but also with Drop Tail queues. Also, the proposed solution can work in a dropping based network as well as in an ECN enabled network.

We have analyzed the framework and evaluated it for various single and multi-bottleneck scenarios with marking and dropping policies being used for congestion notification. Further we showed model is robust and works well even in presence of high background (web) traffic and reverse path congestion. In this paper we also presented a scheme to estimate the utility function of the uncooperative user. We have also evaluate the impact of uncooperative flows on existing AQM proposals and our results suggest that these schemes cannot always manage uncooperative behavior. We are currently working on ways to improve the utility function estimation schemes presented in this paper.

References

[1] A. Akella etal. Selfish Behavior and stability of the Internet: A Game Theoretic Analysis of TCP. *In Proc. of ACM Sigcomm*, Aug 2002.

[2] D. Bansal and H. Balakrishnan. Binomial Congestion Control Scheme. *Proc. of IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.

[3] F. Baboescu, G. Varghese. Scalable Packet Classification. *In Proc. of ACM Sigcomm*, San Diego, Aug 2001.

[4] S. Blake et. al. An Architecture for Differentiated Services. IETF RFC-2475.

[5] W. Feng et. al. Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness. *In Proc. of INFOCOM*, April 2001.

[6] S. Floyd and K. Fall. Promoting the Use of End-to-end Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458-472, 1999.

[7] S. Floyd, etal. Equation-Based Congestion Control for Unicast Applications. In *Proc. of ACM SIGCOMM*, Aug 2000.

[8] S. Floyd and M. Handley and E. Kohler. Problem Statement of DCP. http://www.icir.org/floyd/papers.html

[9] A. Kuzmanovic and E. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants). In *Proc. of ACM SIGCOMM*, Aug 2003.

[10] S. Gorinsky, S. Jain, H. Vin and Y. Zhang. Robustness to Inflated Subscription in Multicast Congestion Control. In *Proc. of ACM SIGCOMM*, Aug 2003.

[11] Frank Kelly, Aman Maulloo and David Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the O.R. Society*, 49 (1998) 237-252.

[12] S. Kunniyur and R. Srikant. End-To-End Congestion Control: Utility Functions, Random Losses and ECN Marks., *Proc. of IEEE INFOCOM* , Tel-Aviv, Israel, March 2000.

[13] D. Lin and R. Morris, "Dynamics of Random Early Detection, " *Proceedings of ACM SIGCOMM*, 1197.

[14] S. H. Low, D. E. Lapsley. Optimization Flow Control, I: Basic Algorithm and Convergence. *IEEE/ACM Transactions on Networking*, 7(6):861-75, 1999

[15] S. H. Low. A Duality Model of TCP and Queue Management Algorithms. *Proc. of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*, Sept, 2000.

[16] "Nishanth R. Sastry and Simon S. Lam. CYRF: A Framework for Window-based Unicast Congestion Control. *In Proc. of ICNP*, 2002.

[17] R. Mahajan and S. Floyd. Controlling High-Bandwidth Flows at the Congested Routers. *In ICNP 2001*.

[18] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Trans. on Networking*, 8(5):556–567, 2000.

[19] T.J. Ott, T.V. Lakshman, L.H. Wong, SRED: Stabilized RED. *In Proc. of IEEE INFOCOM*, 1999.

[20] Packeteer. http://www.packeteer.com
[21] J. G. Proakis et. al, "Algorithms for Statistical Signal Processing", Prentice Hall, 1st edition, Jan 2002.
[22] R. Pan, B. Prabhakar and K. Psounis. CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. *Proc of INFOCOM*, Mar 2000.
[23] S. Shenker. Fundamental Design Issues for the Future Internet. JSAC, vol 13(7), 1995.

APPENDIX

This section will not appear in the final paper but will be made available through a Tech-Report.

The assumptions used in the paper are as follows

- *A1:* The Utility functions are continuous, strictly concave and increasing in their arguments. Further the rates are bounded by *I: [m_s, M_s].*
- *A2:* The curvature of U_s are bounded away from 0 on *I*, i.e. $-U_s^{''}(x_s) \geq 1/\alpha_s > 0$.

I. PROOF FOR RE-MAPPING UNCOOPERATIVE FLOWS FRAMEWORK

Proposition 4: Given the non-negativity constraint on x_s and p_l and strictly concave utility functions U_s and U_{obj}, the new update algorithm as defined in equations (12, 13) still converges to the optimal point.

Proof: The re-mapping function, $U_s^{'}(U_{obj}^{'-1}(p))$, can also be explained as the solution to the following set of equations:

$$\sum_{s \in S(l)} x_s \leq C_l, \quad \forall l \tag{19}$$

$$p_l(\sum_{s \in S(l)} x_s - C_l) = 0 \tag{20}$$

$$U_s^{'}(x_s) = g(\sum_{l \in L(s)} p_l) \tag{21}$$

and $p, x \geq 0$, which are in turn the KKT conditions for the following strictly concave maximization problem

$$\underbrace{max}_{x} \quad \sum_{s \in S} \int_0^{x_s} F(U_s^{'}(y_s))\, dy_s \tag{22}$$

$$\sum_{s \in S(l)} x_s \leq C_l, \quad \forall l, \quad x \geq 0 \tag{23}$$

$$F = \left(U_s^{'}\left(U_{obj}^{'-1}(x_s)\right)\right)^{-1} \tag{24}$$

Now differentiating the objective function (as defined in equation (22) twice with respect to x_s we get

$$\frac{\partial^2 \int_0^{x_s} F(U_s^{'}(y_s))\, dy_s}{\partial x_s^2} = F'(U_s'(x_s))\, U_s^{''}(x_s) = U_{obj}^{''}(x_s) \tag{25}$$

Then using assumption we conclude that the objective function (equation 22) is indeed strictly concave. Further it can be concluded that the optimization problem is presented as if all flows have a utility function of U_{obj}. Since the objective utility functions are strictly increasing and concave it follows that the solution converges to the optimal point.

II. PROOF FOR THEOREM 2

In this section we will outline the proof for Theorem 2. However this proof requires some Lemmas. Here we will only state the lemmas and omit their proofs, for reasons of space constraints.

Let a function, $f(p_l) : f(p_l) \geq 0$, $\forall p_l$, *f(0) = 0* exists for all links and the following condition holds true

$$\sum_{l \in L(s)} f(p_l) = U_s^{'}(U_{obj}^{'-1}(\sum_{l \in L(s)} p_l)) = p_{new}^s \tag{26}$$

Then the utility function transformation outlined in equation (11) can be explained by the following modified dual

$$D(p) = \sum_{s \in S} U_s(x_s) - \sum_l f(p_l)(\sum_{s \in S(l)} x_s - C_l) \tag{27}$$

where $f(p_l)$ is defined by equation 26. Now if, $f(p_l)$ is an increasing function in p_l and is always greater than 0; then this function will not change the minima (because $f(p_l)$ satisfies all the properties of Lagrangian multipliers).

Lemma 5: Given the non-negativity constraint on x_s and p_l and strictly concave utility functions U_s and U_{obj}, the function p_{new}^s, $f(p_l)$ as defined in (26) are non-negative and strictly increasing in their argument.

Lemma 6: Under Assumptions (A1, A2) $\nabla D(p)$ is Lipschitz.

Lemma 7: Under assumption (A1, A2) D(p) is lower bounded, continuously differentiable and convex.

Theorem 8: Assume that utility functions, U_s, are increasing, strictly concave and continuously differentiable, and their curvature is bounded away from 0. Then starting from any initial rates in the interior of *X* and prices *p(0)* $\geq$ *0*, every accumulation point (x^*, p^*) of the sequence (*x(t), p(t)*) generated by the above algorithm and equations (12,13) is primal dual optimal.

Proof: By Lemma 6 and 7 the dual objective function *D(p)* is convex, lower bounded and $\nabla D(p)$ is Lipschitz, then any accumulation point p^* of the sequence {*p(t)*} generated by the gradient projection algorithm is dual optimal [14]. Moreover, the constraints are linear and the primal problem is strictly concave hence there is no duality gap. Therefore dual optimal is also primal optimal. ■

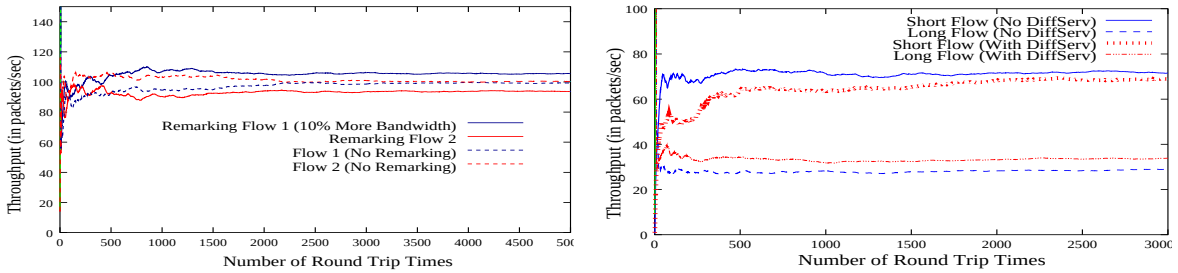
III. TIME VARYING UTILITY FUNCTIONS

Section II outlined some rate control schemes which can be obtained from time-invariant utility functions. There we only considered strictly concave utility function. This assumption implied that all the schemes derived from utility function which are strictly concave will result in stable schemes. In this section we will present some guidelines for deriving rate control schemes from time varying utility functions. Further, we will also put forward some conditions which make these rate control schemes Lyapunov stable.

Consider BCCS schemes where it's parameters α, β, k, l are allowed to vary with time. Then our analysis shows that for a scheme where only β varies with time, *either* of the following limitations on β result in a stable scheme (where stability is defined a Lyapunov stability).

$$|\dot{\beta}(t)| < \rho \frac{\beta(t)^2 R}{\alpha} \tag{28}$$

$$\dot{\beta_s} > 0 \tag{29}$$



(a) Single Bottleneck (b) Multi-Bottleneck

Fig. 15. **Differentiated Services:** Example illustrating how Simple Service Differentiations can be achieved. The objective here was to increase the share of one flow by 10%.

where ρ is some constant which depends on number of links traversed by the flow. The utility function of such a scheme would then be given by

$$U(x) = \frac{-\alpha}{R^2 x^n \beta(t)} \tag{30}$$

and uncooperative (with respect to TCP) flows are given by $|\beta(t)|_i 0$. Similarly, it can be shown that if only the exponents, i.e. k, l, are allowed to vary with time then the rate control schemes are guaranteed to be Lyapunov stable if

$$\dot{k}(t) + \dot{l}(t) = 0 \tag{31}$$

and the utility function of such a scheme would be

$$U(x) = \frac{-\alpha}{R^2 x^{n(t)} \beta} \tag{32}$$

where $n(t) = k(t) + l(t)$ and again uncooperative sources can be generated by choosing $n(t) < 1$.

IV. Differentiated Services

In this section we will briefly present how simple differentiated services can be obtained from our framework. As shown in Fig 1 any uncooperative user can be mapped to a conformant utility space. Exploit this mapping simple differentiated services can be obtained by re-mapping the utility function to a higher utility function curve, for example map U_2 to U_1 (Fig 1).

Though theoretically it is possible to map a utility function to a higher utility function, e.g. map U_2 to U_1, but in practice it implies reducing the end-to-end price for U_2. This clearly cannot work in a dropping based network. Moreover this line of direction is also flawed when applied to a marking based network. This is because a mark always represents a congestion state and by removing a mark would only delay the congestion indication, which is in turn more harmful for the source. Hence we need to take a slightly different approach. Suppose that there are two flows, F_1, F_2, in the network, and the utility function of both the flows is U_1. Further assume we need to provide differentiated services to F_1 such that it always receives 10% more bandwidth than F_2. This can be implemented in our framework by simply re-mapping the utility function of F_2 to U_2, such that the function $U_1 \overrightarrow{f(p)} U_2 \Rightarrow x_1 \overrightarrow{f(p)} x_2$ where $f(p)$ represents the remarking function which achieves the mapping of U_1 to U_2 and x_1, x_2 represents the steady state rates of F_1, F_2 respectively.

In Fig 15 a) we plot one such result for a single bottleneck topology, where both the flows use TCP and go over a bottleneck link of 0.8Mbps, the buffer size is 25 packets and the RTT is 60 ms. The aim of the simulation was to given one flow 10% more bandwidth than the other flow. As shown in the figure, by re-mapping one of the flows to a lower utility function we can achieve simple differentiated service. A similar result is plotted for a multi-bottleneck scenario in Fig 15 b) where the aim was to increase the share of the long flow by 10%. In this simulation the bottleneck capacity was again 0.8Mbps, buffer size of 25 packets, there was one long and one short flow on each bottleneck and all the flows used TCP.

Thus the framework presented in this paper can be extended to provide service differentiation. This solution is attractive because it can be achieved irrespective of the congestion control scheme employed by the user and also does not require any support from the core routers in the network. Moreover, it works with Drop-Tail queues.