A Core-Stateless Utility Based Rate Allocation Framework

Narayanan Venkitaraman, Jayanth P. Mysore, and Mike Needham

Networks and Infrastructure Research, Motorola Labs,
{venkitar,jayanth,needham}@labs.mot.com

Abstract. In this paper, we present a core-stateless framework for allocating bandwidth to flows based on their requirements which are expressed using utility functions. The framework inherently supports flows with adaptive resource requirements and intra-flow drop priorities. The edge routers implement a labeling algorithm which in effect embeds partial information from a flow's utility function in each packet. The core routers maintain no per-flow state. Forwarding decisions are based a packets label and on a threshold utility value that is dynamically computed. Thus the edge and core routers work in tandem to provide bandwidth allocations based on a flow's utility function. We show how the labeling algorithm can be tailored to provide different services like weighted fair rate allocations. We then show the performance of our approach using simulations.

1 Introduction

The Internet is being increasingly used for carrying multimedia streams that are sensitive to the end-to-end rate, delay and drop assurances they receive from the network. We are motivated by two key characteristics that a significant number of these flow share. First, multimedia flows are increasingly becoming adaptive and can adjust their level of performance based on the amount of resource available. The different levels of performance result in varying levels of satisfaction for the user. Another key characteristic is that most of them tend to be composed of packets which contribute varying amounts of utility to the flow they belong to. This intra-flow heterogeneity in packet utility could be caused due to the stream employing a hierarchical coding mechanism as in MPEG or layered multicast, or due to other reasons such as the specifics of a rate adaptation algorithm (as explained later for TCP). In either case, dropping the "wrong" packet(s) can significantly impact the qualitative and quantitative extent to which a flow is able to make use of the resources allocated to it. That being the case, the utility provided by a quantum of resource allocated to a flow depends on the value of the packets that use it. So, merely allocating a certain quantity of bandwidth to a flow does not always imply that the flow will be able to make optimal use of it at all times. As has been observed previously, applications don't care about

G. Carle and M. Zitterbart (Eds.): PfHSN 2002, LNCS 2334, pp. 1–16, 2002.

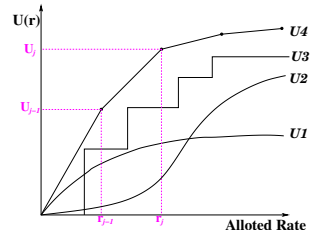


Fig. 1. Utility Functions

bandwidth, per se, except as a means to achieve user satisfaction. In summary, if optimizing the perceived quality of service is the end goal of an architecture, then it is important that we allocate resources according to user's preferences, and provide simple ways for a flow to make best use of its share.

Utility functions have long been recognized as an abstraction for a user to quantify the utility or satisfaction that (s)he derives when a flow is allocated a certain quantum of resource. It maps the range of operational points of a flow to the utility that a user derives at each point. Figure 1 shows some sample utility functions. Such an abstraction provides the necessary flexibility to express arbitrarily defined requirements. Also, it is now well established that different notions of fairness can be defined in terms of utility functions [4,7,8,10]. Partly motivated by recent work by Gibbens, Kelly and others [5,4,11,2], we use utility functions as an abstraction that is used to convey application/user level performance measures to the network. In this paper, we only concern ourselves with allocation of bandwidth as a resource. Therefore, we have used the terms resource and bandwidth interchangeably. We hope that the proposed framework will be a step toward a more general solution that can be used for allocation of other network resources such as those that impact end to end delay and jitter.

In this paper, we propose a scalable framework for allocating bandwidth to flows based on their utility functions. The architecture is characterized by its simplicity – only the edge routers maintain a limited amount of per flow state, and label the packets with some per-flow information. The forwarding behavior at a router is based on the state in the packet header. As the core routers do not perform flow classification and state management they can operate at very high speeds. Furthermore, the framework allows a flow to indicate the relative priority of packets within its stream. The dropping behavior of the system is such that for any flow lower priority packets are dropped preferentially over high priority packets of the flow.

We refer to our architecture as the *Stateless Utility based Resource allocation Framework*(SURF)[1]. The network objective and architecture are described in Section 2. The algorithms implemented by this architecture are described in Section 3. In Section 4, we present the performance of our architecture in a variety of scenarios. Section 5 discusses our implementation experience and some key issues. Section 6 discusses the related work and section 7 concludes the paper.

[1] We borrow the notion of stateless core from CSFQ [14]

2 System Architecture

2.1 Network Model

The approach that we propose is based on the same philosophy used in technologies like CSFQ [14] and Corelite [12]. The network's edge routers maintain per-flow state information, and label packets based on rate at which flows send packets. Core routers maintain no per-flow state. Forwarding decisions are based on the labels that packet carry and aggregate state information such as queue length. Thus the edge and the core routers work in tandem to provide per-flow allocations. We build on these principles to provide resource allocation based on utility functions.

2.2 Network Objective

Let us assume that all flows provide the network their utility functions. There are a variety of objective functions that can be used to accomplish different goals. In the following discussion we consider two possible objectives, provide the intuition behind them, and motivate our choice of one of them as an objective that we use in this paper.

A possible network objective is to maximize the aggregate utility at every link in the network. i.e., at every link in the network, maximize $\sum_{i=1}^{M} U_i(r_i)$, subject to the constraint $\sum_{i=1}^{M} r_i \leq C$, where M is the number of flows sharing the link, $U_i(r_i)$ is the utility derived by flow i for a allocation r_i and C is the total link capacity. Another potential objective is to maximize the aggregate system utility, i.e, maximize $\sum_{i=1}^{N} U_i(r_i)$, where N is the total number of flows in the network. For a network with just a single link both the objectives are identical. However, for a multi-hop network they are different. For instance, consider the example shown in Figure 2. Here, $f1$ is a high priority flow and hence has a larger incremental utility than that for $f2$ and $f3$. If the available bandwidth is two units, then if we use the first objective function we will allocate both units to $f1$ in both the links. This maximizes the utility at every link in the network(3.0 units at every link) and the resultant system utility is 3.0 because only $f1$ received a bandwidth allocation. However, if we use the second objective function, we will allocate two units to $f2$ and $f3$ in each of the links. Though the aggregate utility at any given link is only 2.0, the resultant aggregate system utility is 4.0. This difference in allocation results from different interpretations

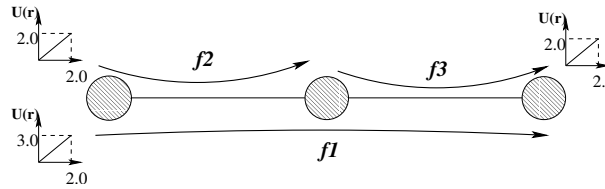


Fig. 2. Bandwidth Allocation Example

of the utility function. The first objective function treats utility functions from a user's perspective by collapsing the entire network into a single unified resource, neglecting the hop count. Thus, this interpretation has 2 key characteristics: (i) it is topology agnostic, i.e. a user does not have to be concerned with how many hops a flow traverses when specifying a utility value, and (ii) it maintains the relative importance of different flows as specified by the utility function across all links.

While the first objective function suites an user's perspective the second treats them from a resource pricing point of view. In this interpretation, the more hops a flow traverses the more resources the flow utilizes and the more a user should pay for comparable performance. Specifically, the utility functions can be viewed as quantifying a user's willingness to pay. Optimizing the second objective function can maximize the network operator's revenue. Networks which employ such an optimization criterion require a user to be cognizant of the hop count of the end to end path traversed by his flow and alter the utility function to get performance comparable to a case with a different number of hops.

Arguably, a case can be made in favor of either of the cases mentioned here or many other possible objectives. As our focus in this paper is to view utility function as a guide to user satisfaction, independent of network topology, we choose to focus on the former objective function.

3 Distributed Framework and Algorithms

In this section, we describe the distributed framework that provides rate allocations that approximate the desired network objective. A key characteristic of the framework is that only the routers at the edges of the network maintain per-flow state information and have access to the utility function of the flows. The core routers however, treat packets independent of each other. They do not perform any per-flow processing and have a simple forwarding behavior.

The framework has two primary concepts: First, an ingress edge router logically partitions a flow into *substreams*. The substreams correspond to different slopes in the utility function of the flow. Substreaming is done by appropriately labeling the headers of packets using incremental utilities. Second, a core router has no notion of a flow, and treats packets independent of each other. The forwarding decision at any router is solely based on the incremental utility labels on the packet headers. Routers do not drop a packet with a higher incremental utility label as long as a lower priority packet can instead be dropped. In other words, the core router attempts to provide the same forwarding behavior of a switch implementing a multi-priority queue by using instead a simple FIFO scheduling mechanism, eliminating any need for maintaining multiple queues or sorting the queue. For ease of explanation, in this paper, we describe the algorithms in the context of utility function U4 in Figure 1[2].

[2] Many utility functions such as U1 can be easily approximated to a piece-wise function similar U4. For functions such as U3 we are still working on appropriate labeling algorithms that provide the right allocation with least amount of oscillations

3.1 Substreaming at the Edge

Every ingress edge router maintains the utility function, $U(r)$, and the current sending rate, r, corresponding to every flow that it serves. The current sending rate of a flow can be estimated using an algorithm similar to the one described in CSFQ [14]. The edge router then uses a labeling algorithm to compute an incremental utility value, u_i, that should be marked on the packet header. The result of this procedure is that the flow is logically divided into k substreams of different incremental utilities, where k is the number of regions or steps[3] in the utility function from 0 to r. The u_i field is set to $(U(r_j) - U(r_{j-1}))/(r_j - r_{j-1})$ which represents the increment in utility that a flow derives per unit of bandwidth allocated to it, in the range (r_j, r_{j-1}). Thus all packets have a small piece of information based on the utility function of the flow embedded in them.

3.2 Maximizing Aggregate Utility

Routers accept packets such that a packet with a higher incremental utility value is not dropped as long as a packet with a lower incremental utility could instead be dropped. Such a policy ensures that in any given router, the sum of u_i of the accepted packets is maximized. There are many different ways by which such a dropping policy can be implemented in the router.

One solution is to maintain a queue in the decreasing order of priorities[4]. When the queue size reaches its maximum limit, q_{lim}, the lowest priority packet in the queue can readily be dropped and incoming packet can be inserted appropriately. This would provide the ideal result. But in a high speed router, even with a moderate queue size, such a solution will be inefficient as the processing time allowable for any given packet will be very small. In the following section, we propose an algorithm that approximates the behavior of such a dropping discipline using a simple FIFO queue, without the requirements of maintaining packets in a sorted order or managing per-flow or per-class information.

Priority Dropping with a FIFO Queue: The problem of dropping packets with lower incremental utility labels before packets with a higher incremental utility can be approximated to the problem of dynamically computing a minimum threshold value that a packet's label must have, in order for a router to forward it. We call this value the threshold utility, u_t. We define threshold utility as the minimum incremental utility that a packet must have for it to be accepted by the router. The two key constraints on u_t are that it must be maintained at a value which will (a) result in enough packets being accepted to fully utilize the link and (b) not cause buffer overflow at the router.

In Figure 3, $R(u)$ is a monotonically decreasing function of the incremental utility u. It represents the cumulative rate of all packets that are forwarded

[3] A step in refers to a contiguous region of resource values with the same slope

[4] This will be in addition to the FIFO queue, that is required to avoid any reordering of packets.

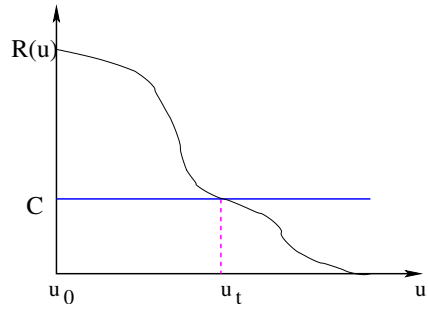


Fig. 3. Threshold Utility

through a link for a given threshold utility value, u_j. So $R(u_j) = \sum_{k=j}^{max} r(u_k)$, where $r(u_k)$ is the rate of packets entering an output link with an incremental utility label of u_k. The threshold utility, u_t is a value which satisfies the condition $R(u_t) = C$, where C is the capacity of the output link. Note that for a given $R(u)$, there may not exist a solution to $R(u) = C$ because of discontinuities in $R(u)$. Also the function $R(u)$ changes with time as flows with different utility functions enter and leave the system and when existing flows change their sending rates. Hence, tracking the function is not only expensive but may in fact be impossible. So in theory, an algorithm that uses the value of a threshold utility for making accept or drop decisions, cannot hope to replicate the result obtained by an approach that involves sorting using per-flow state information. Our objective is to obtain a reasonably close approximation of the threshold utility so that the sum of utilities of flows serviced by the link closely tracks the optimal value, while the capacity of the output link is fully utilized.

First, we give the intuition behind the algorithm that a router uses to maintain the threshold utility u_t for an output link and then provide the pseudo code for the algorithm. We then describe how it is used to make the forward or drop decision on a new incoming packet.

The objectives of the algorithm are (i) to maintain the value of u_t such that for the given link capacity the sum of the utilities of all the accepted packets is close to the maximum value possible, and (ii) to maintain the queue length around a specified lower and upper threshold values(q_{lth} and q_{uth}). There are three key components in the algorithm. (a) to decide whether to increase, decrease or maintain the current value of u_t, (b) to compute the quantum of change and (c) to decide how often u_t should be changed. The key factors that determine these decisions are avg_qlen, an average value of the queue length computed (well known methods for computing the average, like the exponential averaging technique can be used for this purpose) and q_{dif}, the difference between the virtual queue length value at the current time and when the threshold u_t was last updated. The virtual queue length is a value that is increased on an enqueue event by the size of the packet received if its $u_i \geq u_t$. The value is decreased by the size of the packet either on a deque or during a successful enque of a packet

with a label less than than the u_t[5]. The latter enables corrective action when packets are being dropped due to a incorrect(large) threshold. Thus, the virtual queue length is simply a value that increases and decreases without the physical constraints of a real queue. Maintaining a virtual queue length in this manner provides an accurate estimate of the state of congestion in the system. Note that even when the real queue overflows, the virtual queue length will increase, resulting in a positive q_{dif} reflecting the level of congestion. Similarly, when the u_t value is very large and no packets are being accepted, the virtual queue length will decrease, resulting in a negative value of q_{dif}. q_{dif} reflects the rate at which the length of the virtual queue is changing. When there is a sudden change in $R(u)$, q_{dif} provides a early warning signal which indicates that u_t may need to be modified. However, if the link state changes from uncongested to congested slowly, the absolute value of q_{dif} may remain small. But a value of avg_qlen that is beyond the specified queue thresholds indicates that u_t needs to be changed.

The quantum of change applied to u_t is based on the amount of buffer space left – given by the queue length, and the rate at which the system is changing – given by q_{dif}. Congestion build up, is equivalent to $R(u)$ in Fig. 3 shifting to the right. To increase the threshold, we use a heuristic to determine a target value u_{itgt} such that $R(u_{itgt}) < C$. This is used to significantly reduce the probability of tail drops. Currently this value of u_{itgt} is based on the average u_i values of all the accepted packets and the maximum u_i value seen in the last epoch. The value of u_t is then incremented in step sizes that are based on the estimated amount of time left before the buffer overflows. Similar computation is done to decrease the threshold where u_{dtgt} is based on the average u_i values of all packets dropped in the last epoch. The pseudocode for updating the threshold is as follows:

if $(avg\_qlen < q_{lth}) or (q_{dif} < -K_q)$
  $time\_left = avg\_qlen/q_{dif}$
  $change = (u_t - u_{dtgt})/time_left$
else if $(avg\_qlen > q_{uth}) or (q_{dif} > K_q)$
  $time\_left = (q_{lim} - avg\_qlen)/q_{dif}$
  $change = (u_{itgt} - u_t)/time\_left$
$u\_t+ = change$

There are two events that trigger a call to the update-threshold() function. They are (a) whenever $|q_{dif}| > K_q$ and (b) a periodic call at the end of a fixed size epoch. The first trigger ensures fast reaction. The value of K_q is a configurable parameter and is set such that we do not misinterpret a typical packet burst as congestion. Also, it provides a self-healing feedback loop. For instance, when congestion is receding, if we decrease u_t by steps that are smaller than optimal, this trigger will result in the change being applied more often. Case (b) ensures that during steady state, the value of u_t is adjusted so that the queue length is maintained within the specified queue thresholds.

[5] This case would occur when link is not congested but u_t is incorrectly large.

The forwarding algorithm is very simple. When the link is in a congested state($avg\_qlen > q_{lth}$), if $u_i \leq u_t$ the packet is dropped. Otherwise the packet is accepted.

3.3 Variants of SURF

The framework described above is flexible and can be tailored to specific needs by choosing appropriate utility functions. The labeling algorithms at the edge router can be tailored to label the incremental utilities for specific cases. The forwarding and threshold computation algorithms remain the same. This is a big advantage. In this section, we describe a few specific cases of the edge labeling algorithm and provide the pseudo-code.

Fair Bandwidth Allocation. A common notion of fair bandwidth allocation is one in which all flows that are bottle necked at a link get the same rate, called the fair share rate. To achieve such an allocation, all we need to do is assign identical utility functions with constantly decreasing incremental utilities to all flows. For ease of understanding we provide a labeling procedure for an idealized bit-by-bit fluid model. Let u_{max} be the maximum possible value of incremental utility.

label(pkt)
  $served+ = 1$
  $pkt.u_i = u_{max} - served$

where the value of *served* is reset to 0, after a fixed size labeling epoch, say 1 sec. Let us suppose that the rate at which each flow is sending bits is constant. The result of this labeling algorithm then is that during any given second, the bits from a flow sending at rate r bits per second are marked sequentially from u_{max} to $u_{max} - r$. The router in a bottleneck link will compute the threshold u_t and drop packets from all flows with $u_i < u_t$. This results in fair bandwidth allocation. This is an alternate implementation of CSFQ [14]. As we will see in the next section, a key advantage of this approach is that it allows us to convey rate information as well as intra-flow utility using the same field in the packet header.

Intra-flow Priorities. Consider a flow, i, which is sending packets with multiple priority levels at a cumulative rate r_i. For instance, the levels could be I, P and B frames in an MPEG video stream or layers in a layered multicast stream [9]. The utility function corresponding to the flow will be similar to U4 in Figure 1. Independent of the number of layers and rate allocated to other flows, if flow i's packets need to dropped, we would like the packets from layer $j+1$ to be dropped before layer j. To achieve such a dropping behavior, the end hosts must communicate the relative priority of a packet to the edge router. A simple mechanism to accomplish this could be in the form of a field in the packet header. The desired dropping behavior honoring intra-flow drop priorities can be

achieved in the proposed framework by using a labeling procedure similar to the pseudo-code given below. The forwarding and threshold computation algorithms remain unchanged.

label(pkt)
  $p = pkt.intraflow\_priority$
  $served[p] += pkt.size$
  $cum\_rate[p] = cum\_rate[p-1] + est\_rate[p]$
  if ($served[p] < cum\_rate[p-1]$) or
    ($served[p] \geq cum\_rate[p]$)
      $served[p] = cum\_rate[p-1]$
  $pkt.u_i = u(served[p])$

Figure 4 describes the above pseudo-code. In the code given above, $est\_rate[p]$ is the estimated rate at which a flow is sending packets of a certain priority level p and $cum\_rate[p]$ is simply $\sum_{i=1}^{p} est\_rate[p]$(assuming 1 to be the highest priority level). $est\_rate[p]$ can be computed using a method similar to the one used in [14]. $served[p]$ maps the packet received onto the appropriate region in its utility function. $u(r)$ gives the incremental utility of the region corresponding to rate r.

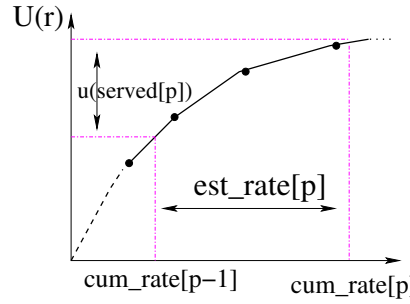


Fig. 4. Labeling Algorithm

Improving TCP Performance. The labeling algorithm used by the edge router can be tailored to improve the performance of TCP. Specifically, it can mitigate two primary causes of a reduction in throughput – (i) drop of a retransmitted packet (ii) back to back drops of multiple packets. To accomplish the former, the labeling procedure can label retransmitted packets with the highest allowable incremental utility for the flow; and to accomplish the latter, the labeling algorithm can assign consecutive packets to different priority levels thereby reducing the chances of back-to-back drops. Such an implicit assignment of interleaved priorities is also useful for audio streams, whose perceived quality improves when back-to-back drops are avoided.

4 Performance Evaluation

In this section, we evaluate the algorithms using simulations. We have implemented the algorithms in the *ns-2* simulator. We have used the two topologies. Topology 1 is a simple network with a single congested link which all flows share. We use this topology to first show that the system converges to the optimal rate allocation with both adaptive and non adaptive flows. We then use topology 2, shown in Figure 5, to confirm that the solution converges to the expected values in a multi hop network. We then take a specific variant of SURF discussed in 3.3, where all flows have the same utility function. Using TCP and CBR flows we show that fair allocation is achieved. Finally, using a flow with 3 different packet priority levels, we show that the proposed framework drops packets based on a flow's intra-flow priority. The allocated rate for CBR flows is measured by summing the number of bits received every 250 ms in the sinks, and that for adaptive flows is measured in the sinks using an exponential averaging routine(similar to the arrival rate estimation algorithm in the edge router).

All the simulations presented in this section use a fixed packet size of 1000 bytes, a maximum queue size of 100 packets, queue thresholds q_{uth} and q_{lth} of 10 and 50 packets respectively and a K_q of 10 packets. K_q and q_{lth} are chosen so that we do not interpret small bursts that typically occur at a router as congestion and q_{uth} is chosen so that there is sufficient time to adapt u_t and avoid tail drops. Flows use the utility functions shown in Figure 6. The 4 values given for each utility function are incremental utilities for the 4 rate regions. For instance, function U2 has an incremental utility of 0.45 for an allocation in the region 480-960Kbps.

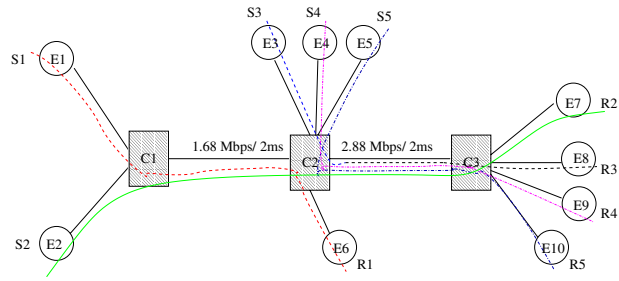


Fig. 5. Topology 2

4.1 Single Congested Link

In this scenario, Flows 1, 2 and 3 have utility functions $U1$, $U2$ and $U3$ (shown in Figure 6) respectively. For the first set of results, we consider 3 non-adaptive CBR sources, each sending at a rate of 1.92Mbps, the link capacity. To calculate the optimal allocations, we note that the incremental utilities for these flows are

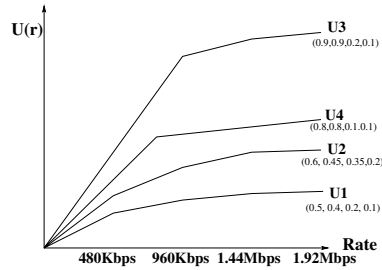
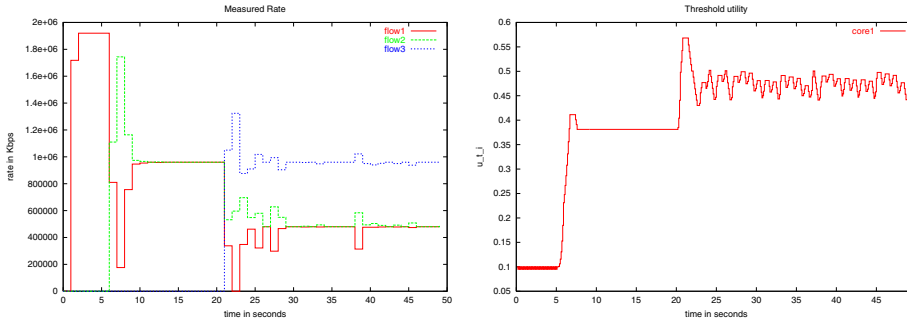


Fig. 6. Utility Functions

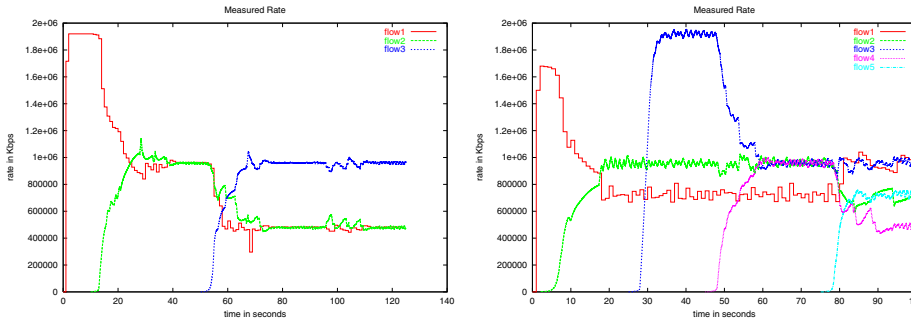


(a) (b)

Fig. 7. Single Link With Non Adaptive Flows

such that, $U3$ has the highest incremental utility in region 0-960Kbps. This is followed by $U2$ then $U1$ in the region 0-480Kbps, followed by $U2$ and $U1$ again in the region 480-1.44Mbps. So when flow1 and flow2 are sharing the channel, the optimal allocation would be to provide 960Kbps each to flow1 and flow2. Similarly when all 3 flows share the channel, the optimal allocation is to allocate 960Kbps to flow3 and 480Kbps each to flow1 and flow2.

In the simulation, flow 1, enters the network at time 0. As it is the only flow in the system, it is allocated the entire network bandwidth. 5 seconds after the start of the simulation, flow 2 enters the network. From Figure 7(a) we see that flow1 and flow2 are allocated and 960Kbps each, which is the optimal allocation. After 20 seconds, flow 3, enters the system. This flow has incremental utility of 0.9 (U3 of Figure 6) which is larger than that of the other two flows in the region 0-960Kbps. Consequently, the threshold utility, shown in Figure 7(b), momentarily shoots up. However, this results in increased packet drops, resulting in containing the queue size within acceptable limits. When the queue size drops, the threshold utility also drops to a lower value. It finally stabilizes at a value that results in the capacity being shared between Flows 1, 2 and 3 in the ratio 1:1:2. This conforms to the allocation that optimizes the aggregate utility of the flows sharing the link.



(a) (b)

Fig. 8. Adaptive and non-adaptive flows with SURF

We now present the results for the same scenario with adaptive flows. Flow 2 and 3 are adaptive. As these flows adjust their sending rates, the distribution of u_i(and hence, $R(u)$) will also change. The objective of this results is to show that even when $R(u)$ changes u_t converges to the correct value. Figure 8(a) shows the measured rate at the receivers of the flows.

4.2 Multiple Congested Links

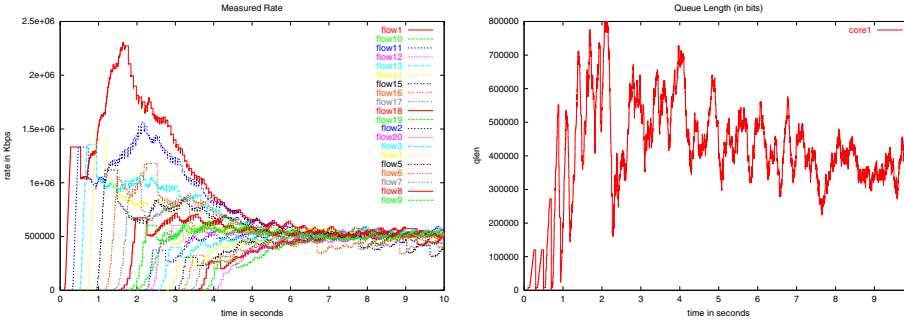
The results for this scenario are shown in Figure 8(b). In this case, there are five flows having utility functions U1,U2,U3,U1 and U4 respectively. The key objectives in this experiment are to show (1) that the computation of threshold utility settles about the correct value even with flows traversing more than just a single link, (2) even if there is no exact solution to $R(u) = C$, the value computed by the algorithm hovers around the right value.

Between time 0 to 25 seconds, only flow 1 and 2 are in the system. The optimal allocation will result in a ratio of 2:1.5 for flows 1 and 2. This is a case when there is no exact solution to the condition $R(u) = C$. At $u_t < 0.4$, the accepted rate will be only 1.44Mbps, where as at $u_t \geq 0.4$, the accepted rate will be 1.92Mbps. The results in Figures 8(b) show that the bandwidth distribution stabilizes around the correct values.

At time 25 seconds, flow 3 enters and occupies the entire remaining bandwidth in the link C2-C3. Its entry does not affect the allocations made to flow2. Flow4 then enters at time 45sec. It grabs its share of the two units of bandwidth at the expense of flow3 because flow 3 has a lower incremental utility in the region beyond 960Kbps, compared to flow 4 in the region 0-960Kbps. At time 75sec, flow 5 enters the network. It has a larger incremental utility till the rate 720Kbps, compared to flows 2 and 4. So it gets that share in preference to flows 2 and 4, which end up with allocations of 720Kbps and 480Kbps respectively.

4.3 Achieving Fair Allocation

The simulated network topology has a single link with 20 sources sharing a 10Mbps. The labeling epoch length in the edge routers is set to 250 ms. The edge routers implement the labeling algorithms that provide fair bandwidth allocation. The sources successively enter the system in 200 ms intervals. Figure 9(a) and (b) show the rate allocations and the profile of the queue length with TCP sources. The figures indicate that: (1) there is approximately fair sharing of the link bandwidth, (2) the queue length is controlled to a range close to the specified threshold values.



(a) (b)

Fig. 9. Providing Fair Allocation For TCP Flows

4.4 Honoring Intra-flow Priorities

We next perform a simple experiment to illustrate the support for intra-flow priority dropping provided by an island of SURF routers. In this experiment we have five UDP flows sharing a 2 Mbps link. Each of the flows is assumed to be encoded using a layered encoding scheme that has 3 levels of priority. A base layer of highest priority(priority 1), and two enhancement layers with priorities 2 and 3(priority 2 packets being more useful in reconstructing the content than those of priority 3). Further, the flows send a third of their packets in each layer, uniformly striating the flow across the three layers. Each flow sends data at a rate of 1 Mbps, resulting in a fair share of 400 Kbps. When using SURF, we observed that each flow was in fact allocated exactly 400 Kbps.

Priority	*total_sent*	*dropped*
1	785	3
2	785	618
3	785	761

The above table shows the dropping behavior of a network implementing SURF. The values shown are typical of any given flow. *total_sent* is the number

of packet sent by a flow. *dropped*, the number of packets dropped in different priority levels clearly indicates that the lower priority packets are dropped in preference to higher priority packets. During the initial phase of the simulation, when flows are introduced successively, all the layers are served. But as congestion builds up, all packets from priority 3 are dropped. The fair share of 400Kbps is shared between the entire 333Kbps of priority 1 packets and a small portion(66Kbps) of priority 2 packets. This simulation shows that in addition to providing utility based allocation of rate, SURF routers honor intra-flow priorities.

5 Discussion

5.1 Implementation Experience

We have implemented the framework described here in the linux kernel, as a queuing discipline that can be configured using *'tc'*, the traffic control program. The threshold computation, forwarding procedure and the different variants of SURF that we discussed before have been implemented. The results obtained using the simulations match closely with the results on our implementation testbed. We have used *vcr*, a client-server program that can stream mpeg audio-video files and play the same, to demonstrate the improvement in user experience. The only change that was required to the *vcr* program itself, was an addition to convey the relative priority of a packet. We use the standard '*setsockopt*()' system call before every '*send*()' operation to convey this information to the IP layer. The time and space complexity of implementing the operations of the core router are of the same order as RED. So we believe that the proposed schemes can be implemented without compromising the efficiency of high speed core routers.

5.2 Convergence Issues

One of the key issues is the stability and convergence of the threshold computation algorithm. Though in the simulations that we have performed the threshold converges to the correct value, an analytical evaluation of the convergence of the algorithm is essential. In [1], we have considered the specific case of section 3.3 where all flows have the same utility function. We consider the different cases with constant bit rate sources and TCP like rate adaptive sources and on-off sources. We have derived the conditions that are essential for the convergence of a threshold computation algorithm(or fair share estimation in the CSFQ). We show that both the CSFQ algorithm and the algorithm proposed in this paper satisfy those conditions. Further analysis with different types of utility function is part of ongoing work.

5.3 Pricing Issues

In a network that provides service differentiation based on utility functions, greedy users may be tempted to associate the highest utility value to any amount

of bandwidth allocated. Typically a pricing scheme is used to prevent users from lying about their utility functions. The proposed framework does not dictate the use of a specific pricing scheme. This framework can coexist with many different pricing schemes. In many cases, the pricing scheme can use some parameters provided by the framework to determine the cost of the resource. For example, in a simple spot-market pricing mechanism that is based on the current level of congestion, the threshold utility(u_t) can be used as an indicator of the current the level of demand of the resource. Also, note that the framework does not bind a flow to a particular utility function. It offers the flexibility to dynamically change the utility function during the lifetime of a flow. This is a very useful feature because, the utility that a user derives from a certain amount of resource allocated to a flow can change dynamically based on various circumstances.

6 Related Work

The concept of core-stateless networks was proposed by Stoica et al in [14]. They provide an architecture for approximating the behavior of a network of fair queuing routers. [14] achieves this by labeling packets with the estimated sending rate of a flow and probabilistically dropping packets based on the label and a dynamically computed fair share rate in the core. Our approach is partly motivated by their work. In [13], the authors propose a similar approach based on carrying flow state in packet headers to provide delay guarantees. RFQ [3], which was developed independent of our framework is an extension to CSFQ which provides support for intra-flow priorities by marking packet with different colors. It does not consider user-specified utility functions for labeling packets. In [11], Kunniyur and Srikant, describe a framework for designing end-to-end congestion control schemes where users can have different utility functions. While the framework proposed in this paper is network based, the framework in [11] depends on flows performing congestion control based on utility functions. The objective function they optimize is different from the one we elaborate in this paper. In fact, they maximize aggregate user satisfaction, rather than utility at every link. Also, being a purely end-to-end adaptation scheme with no router support, it does not provide for intra-flow drop priorities. [6] presents a framework for optimizing the aggregate user satisfaction of a network by performing congestion avoidance using the congestion price as the feedback parameter. They present an algorithm core routers can use for computing the congestion price (the shadow price), and a rate adaptation algorithm for end hosts. The rate adaptation algorithm adapts the rate of a flow based on a flows willingness to pay. They show that at the socially optimum allocation, the first derivative of a user's utility function exactly matches the sum of the shadow prices of all resources along the flows route. They propose changes to the use of the ECN mechanism in TCP, to implement their framework. [2] proposes a mechanism of utility max-min which tries to maximize the minimum utility received by applications sharing a bottleneck link. This is different from the objectives that we considered in section 2.

7 Conclusion

In this paper, we have presented *SURF*, a scalable architecture for providing bandwidth allocation to flows based on their utility functions. Our proposed architecture maximizes the aggregate utility at each link. Its key attributes are scalability, accomplished using a core-stateless architecture, and support for intra-flow priorities. By simply tailoring edge labeling algorithms, this framework can be leveraged to optimize the performance of a variety of flows. We described an algorithm for computing the threshold utility and forwarding packets based on the computed value. We then presented labeling algorithms that can be used to provide fair sharing of link bandwidth; and to optimize the perceived quality of layered streams while providing equal sharing of the bandwidth across flows. We presented a selected set of results to illustrate various facets of performance. As high speed networks become increasingly common, we believe that core-stateless schemes such as SURF will be useful in enabling flexible service models and optimizing user satisfaction in a unified, scalable framework.

References

1. R. Barnes, R. Srikant, J.P. Mysore, N. Venkitaraman "Analysis of Stateless Fair Queuing Algorithms", *to appear in Proc. of 35th Annual Conference on Information Sciences and Systems*, March 2001.
2. Z. Cao and E. Zegura. "Utility Max-Min: An application-Oriented Bandwidth Allocation Scheme", *Proc. of IEEE INFOCOM*, 1999.
3. Z. Cao, Z. Wang and E. Zegura. "Rainbow Fair Queuing: Fair Bandwidth Sharing Without Per-Flow State", *Proc. of IEEE INFOCOM*, 2000.
4. F.P. Kelly. "Charging and Rate Control for Elastic Traffic", *European Transactions on Telecommunications*, vol 8, 1997.
5. R.J. Gibbens and F.P. Kelly. "Distributed connection acceptance control for a connectionless network", *Proceedings of the 16th International Teletraffic Congress, Edinburg*, June 1999.
6. P. Key, D. McAuley, P. Barham. "Congestion Pricing for Congestion Avoidance", *Technical Report MSR-TR-99-15, Microsoft Research*, Febuary 1999.
7. L. Massoulie and J. Roberts. "Bandwidth Sharing: Objectives and Algorithms", *Proc. of INFOCOM*, March 1999.
8. J. Mo and J. Warlang. "Fair end-to-end Window based Congestion Control",
9. S. McCanne, V. Jacobson, and M. Vetterli. "Receiver-driven layered multicast", *Proc. SIGCOMM'96*, Stanford, CA, Aug. 1996, ACM, pp. 117-130.
10. S. Shenker "Fundamental Design Issues for the Future Internet", *IEEE JSAC. Vol 13, No. 7*, September 1995.
11. S. Kunniyur and R. Srikant "End-to-End Congestion Control Schemes: Utility Functions, Random Losses and ECN Marks", *Proc. of IEEE INFOCOM* 2000.
12. S. Raghupathy, T. Kim, N. Venkitaraman, and V. Bharghavan. "Providing a Flexible Service Model with a Stateless Core", *Proc. of ICDCS 2000.*
13. I. Stoica and H. Zhang. "Providing Guaranteed Services Without Per Flow Management", *Proceedings of the ACM SIGCOMM '99 Conference*, September 1999.
14. I. Stoica, S. Shenker and H. Zhang. " Core-Stateless Fair Queueing: Achieveing approximately Fair Bandwidth Allocations in High Speed Networks", *Proceedings of the ACM SIGCOMM '98 Conference*, September 1998.