UNIVERSITY OF CALIFORNIA
Santa Barbara

"Early Regulation of Unresponsive Flows"

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Science

by

Anand Rangarajan

Technical Report TRCS99-26
Dept of Computer Science
University of California, Santa Barbara

July 1999

The thesis of Anand Rangarajan is approved

Committee Chairperson

June 1999

To my parents

ACKNOWLEDGMENTS

I thank my advisor, Prof. Anurag Acharya, for his guidance and support without which this thesis would have never been possible. He was always ready to discuss problems with me amidst his busy schedule and helped me with clever solutions to all kinds of problems that I stumbled across - from complex implementation issues to simple LaTeX tricks. I consider myself fortunate to be his student.

I'm grateful to Prof. John Bruno for his immense help in choosing my thesis topic and for his invaluable advise in the later stages.

Prof. Kevin Almeroth readily accepted unannounced drafts and provided highly useful comments and suggestions. His help is much appreciated.

I thank Mark Allman of Glenn Research Center, NASA for sparing his time to read through my thesis and providing useful feedback.

Many thanks to Sally Floyd of ACIRI. The information on her web site was a primary source of motivation for this work and her *ns* simulation scripts were very useful.

I appreciate all those on the *ns*-users' mailing list who replied to my questions and helped me solve problems related to the implementation in ns.

I would like to thank my family for their continuous support and encouragement.

Thanks to Max Ibel for helping me with LaTeX, lending his laptop, and most importantly, for coffee.

Thanks to Josep Blanquer and Rohit Grover for enlivening the lab with music.

Last but not the least, thanks to all those who kept asking me "Have you finished your thesis yet?". Without you, I would've taken a lot longer to finish this work being in a lovely place like Santa Barbara.

ABSTRACT

"Early Regulation of Unresponsive Flows"

by

Anand Rangarajan

In this thesis, we propose router mechanisms to regulate unresponsive best-effort traffic. By unresponsive traffic, we mean flows that do not reduce their sending rate in response to congestion. The goal of the proposed mechanisms is to drop undeliverable packets (packets that are dropped somewhere in the network before they reach their destination) as close to the periphery of the network as possible. The key ideas of our approach are: (1) edge routers keep track of incoming flows and their arrival rates; (2) non-edge routers use RED (Random Early Detection) for queue management and generate rate-limited source quenches on packet drops to advise sources to reduce their sending rates; and (3) edge routers snoop on source quenches passing through them and use them to control per-flow regulators. Regulators adjust their maximum sending rate using a multiplicative-decrease, additive-increase discipline. A decrease is triggered by the arrival of a source quench; an increase is triggered at the end of every estimated round-trip time period in which no source quench was received. We examine the impact of these mechanisms for a variety of simulated network topologies and traffic patterns.

Contents

Chapter 1

Introduction

Effective congestion control is one of the main reasons for the health of today's large, rapidly growing, best-effort service Internet. TCP has been, and is currently the predominant protocol used by Internet applications. TCP's congestion control algorithms [16, 32] have been hugely successful. However, the stability of the Internet is being threatened by an increase in the number of flows that do not use congestion control. A growing number of streaming-content providers use UDP-based delivery with little or no congestion control. For example, RealAudio, one of the major streaming applications, uses one of several fixed data rates depending on the bandwidth available when the flow begins; the Real Broadcast Network uses *Robust UDP* [14] (i.e., UDP with retransmissions but no congestion control). We call flows that do not reduce their sending rate upon notification of congestion or worse still, increase their sending rate [1] as *unresponsive.* Flows that respond to congestion in the same way as a standard TCP flow would, in the same circumstances are referred to as *responsive.* Flows that are more aggressive, in terms of end-to-end congestion control, than a standard TCP flow are referred to as *under-responsive.* In this thesis, we focus on problems due to unresponsive flows.

The problems due to increasing deployment of unresponsive traffic range from extreme unfairness against competing responsive traffic to the possibility of congestion collapse [11]. Congestion collapse is a situation where there is maximum utilization of network resources, but very little accomplishment of useful work. The primary reason for the potential congestion collapse due to unresponsive flows is the absence of end-to-end congestion control mechanisms for the unresponsive flows. To give an intuitive idea, consider a congested gateway through which a mixture of responsive and unresponsive flows pass. When the gateway drops packets (which could be from any flow), the responsive flows reduce their sending rate in response to packet drops, but the unresponsive ones do not. This process repeats till the responsive flows are completely shut off. Since the benefits of an uncongested (or under-congested) network are available only if all (or most) end-hosts cooperate in responding to congestion, growth in unresponsive traffic will provide an incentive for even more end-hosts to eliminate congestion control. As more of the traffic in the Internet becomes unresponsive, more gateways get congested and finally the network becomes chronically congested. As a result, a large fraction of the packets that enter the network are dropped, somewhere in the network, before they reach their destination. These dropped packets are called *undeliverable* packets. Undeliverable packets do not contribute to any useful work (since they do not reach their destination), but waste network resources (link bandwidth, buffers etc). This situation would not

[1] For example, "RealVideo uses forward error correction (FEC) techniques to allow complete reconstruction of image packets that may be lost. The likelihood of frame damage occurring is reduced dramatically, making video frames impervious to all but extremely high levels of packet loss. " [15]

have occurred if the end-systems used end-to-end congestion control and reduced their sending rate when their packets were dropped due to congestion.

Given the rapid expansion of the Internet and the motivation/pressure for commercial service providers to become unresponsive, it is no longer possible to rely on voluntary cooperation of the end-hosts' congestion control. To ensure the continued stability of the Internet, it is, therefore, important for the network to regulate unresponsive traffic. Recognizing this need, the authors of RFC 2309 [4] strongly urge *"research, engineering and measurement efforts contributing to the design of mechanisms to deal with flows that are unresponsive to congestion notification or are responsive but more aggressive than TCP."* Floyd&Fall [11] believe that growth of undeliverable packets generated by open-loop applications is the largest unresolved danger with respect to congestion collapse in the Internet today.

In this thesis, we propose and examine some router mechanisms to identify and regulate unresponsive best-effort traffic. The goal of the proposed approach is to drop undeliverable packets as as close to the periphery of the network as possible. Dropping undeliverable packets early frees network resources for other packets that can make better use of them.

We have evaluated the performance of our scheme for a variety of simulated network topologies and traffic patterns that include competing responsive and unresponsive flows.[2] The results, show that this scheme is able to move most of the packet drops from unresponsive flows to the periphery of the network. The resources freed by dropping these packets early allows other flows to achieve significant performance improvement.

[2] We have not considered *under-responsive* flows in our traffic patterns

Chapter 2

The Problem of Unresponsive Flows

The increasing deployment of unresponsive flows could lead to two classes of problems on the Internet: unfairness to competing responsive traffic and congestion collapse due to undeliverable packets. In this chapter, we present an overview of these problems and review related work. Our description of these problems is largely adapted from [11].

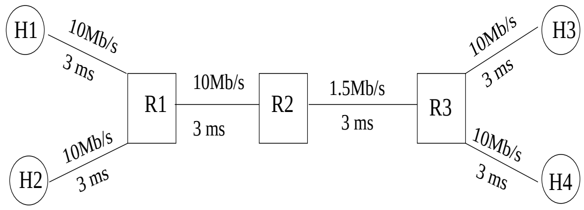


Figure 2.1: Simulation network for the Unfairness test

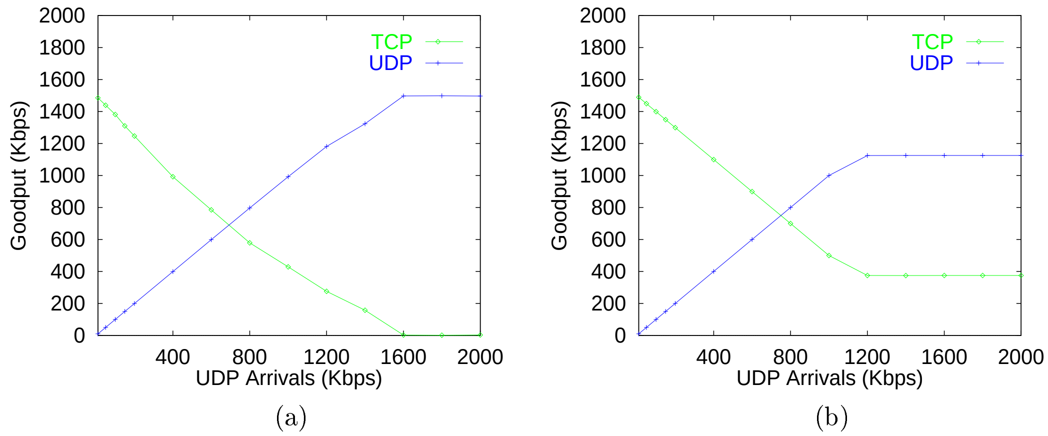


Figure 2.2: (a) Simulations with three UDP flows and one TCP flow competing for a link bandwidth of 1.5 Mbps and RED queue management - UDP flows are unfair to TCP. (b) Simulations with three UDP flows and one TCP flow competing for a link bandwidth of 1.5 Mbps and DRR queue management - UDP flows are NOT unfair to TCP

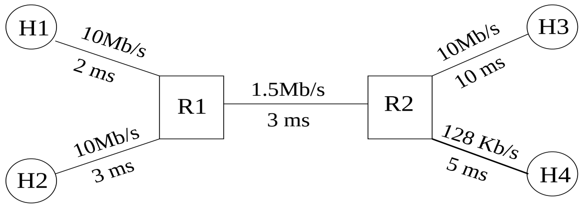


Figure 2.3: Simulation network for the test of congestion collapse

2.1 Unfairness

The test network used to demonstrate unfairness is shown in Figure 2.1. The topology corresponds to that of two LANs connected via a long-haul link. The traffic consists of one responsive TCP flow from H1 to H3 and three unresponsive constant-bit-rate (CBR) UDP flows from H2 to H4. This experiment illustrates the fact that when a mixture of responsive and unresponsive flows pass through a congested router, the unresponsive flows gain an unfair share of the resources at the congested router. The scheduling mechanism used in the link R1-R2 is Deficit Round Robin (DRR) [31] and the one used in the link R2-R3 is Random Early Detection (RED) [12]. A sequence of experiments were performed by varying the aggregate UDP sending rate from 10 Kbps to 2 Mbps. The performance metric used was the aggregate TCP and UDP goodput. By *goodput* we mean the bandwidth delivered to the receiver excluding duplicate packets. Figure 2.2 shows the results of the experiments. The x-axis shows the UDP sending rate. There are two points corresponding to each UDP sending rate - one representing the aggregate TCP goodput and the other the aggregate UDP goodput. As it can be seen from the graph, when the UDP sending rate is large, the UDP flows use almost all of the shared link's bandwidth. As the UDP sending rate increases, R2-R3 link gets congested. The congested router R2 starts dropping packets. The TCP flow responds to packet-drops by reducing its sending rate, but the UDP flows do not. When the UDP sending rate is more than the link bandwidth, the TCP flow gets nothing effectively.

Figure 2.2(b) presents results for simulations similar to those presented in Figure 2.2(a) except that the RED queue management in the link R2-R3 is replaced by DRR. The graph shows that the unfairness problem can be solved by using per-flow scheduling mechanisms at all the links in the path. Each of the four competing flows is restricted to about 25% of the link bandwidth. Since the TCP flow is guaranteed 25% of R2-R3 link bandwidth, it is unaffected by congestion.

2.2 Congestion Collapse

Congestion collapse is a situation where, the network resources are fully utilized, but no (or little) useful work is done. The following experiment illustrates congestion collapse caused due to undeliverable packets. It occurs when a source keeps sending packets into the network that are dropped somewhere before they reach their final destination. The undeliverable packets waste bandwidth in all the links in the path from the source till the bottleneck link where they are dropped. The wasted bandwidth could have been used by other competing traffic. A congestion collapse situation occurs due to increased deployment of open-loop applications that do not use end-to-end congestion control or in the worst case, increase their sending rate in response to increased packet drop rate. S.Floyd and K.Fall identify congestion collapse due to undeliverable packets as the largest unresolved danger in the Internet today [11].

Figure 2.3 shows the simulation network that is used to illustrate congestion collapse due to

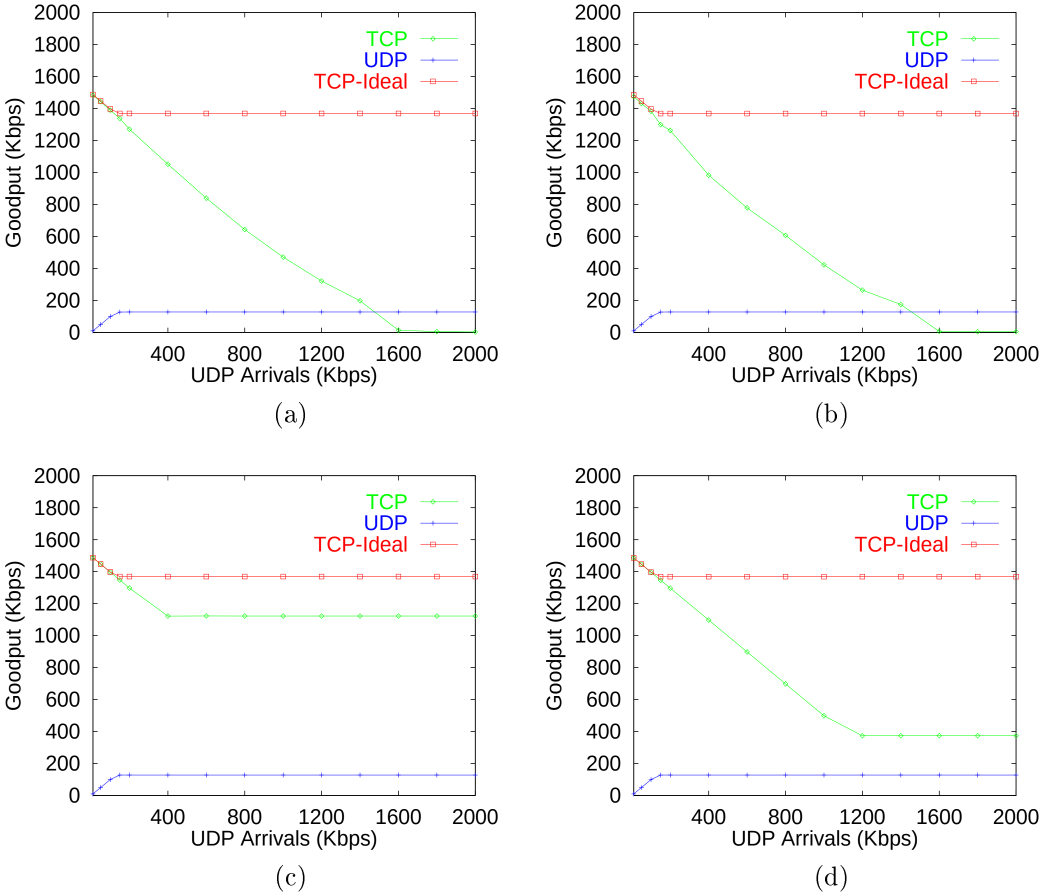


Figure 2.4: (a) Congestion collapse with 3 TCPs and 1 UDP, RED queue management (b) Congestion collapse with 1 TCPs and 3 UDP, RED queue management (c) No congestion collapse with 3 TCPs and 1 UDP, DRR scheduling (d) Congestion collapse with 1 TCPs and 3 UDP, DRR scheduling

undeliverable packets. The topology corresponds to two LANs connected via a long-haul link where one of the hosts is connected to its LAN via an ISDN line. The traffic consists of a mixture of competing TCP and constant-bit-rate UDP flows. The TCP flows are from H1 to H3 and the UDP flows are from H2 to H4. Note that the bottleneck link for the UDP flows is the 128 Kbps link attached to their destination H4. This corresponds to the "last mile" links that are used by a large number of hosts. As a result of the bottleneck, when the UDP sending rate exceeds 128 Kbps, most of the UDP packets are dropped in the router R2, unable to find space in the queue for the R2-H4 link. The tests involve a combination of two different traffic mixtures (three TCP flows, one UDP flow and one TCP flow, three UDP flows) and two different scheduling mechanisms (FIFO with RED queue management and DRR). The different mixture of traffic and scheduling mechanisms were chosen to illustrate that congestion collapse due to undeliverable packets can occur irrespective of the number of unresponsive flows and the kind of scheduling discipline used at the routers.

The simulation results are presented in Figures 2.4. The x-axis in each graph shows the aggregate UDP sending rate. For each UDP sending rate three points are marked, one for the observed aggregate TCP goodput, one for the aggregate UDP goodput and one for the aggregate TCP goodput in the ideal case (i.e., when the UDP source never sends more than 128 Kbps). Figures 2.4(a) and 2.4(b) illustrate that when FIFO with RED queue management is used, the TCP goodput falls almost linearly with increasing UDP sending rate, but the UDP goodput remains constant.

Figure 2.4(c) shows that per-flow scheduling mechanisms can prevent congestion collapse when *one* UDP flow is the cause. The UDP flow is restricted to about 25% of the link bandwidth and the three TCP flows together get about 75% of the link bandwidth. Note that the TCP flows could have achieved a higher goodput if the aggregate UDP flow's sending rate had not exceeded 128 Kbps (its maximum attainable goodput).

Using per-flow scheduling mechanisms is not enough in all cases. In the case illustrated by Figure 2.4(d) where one TCP flow competes with three UDP flows, the UDP flows together get about 75% of the shared link's bandwidth. But only a small fraction of the packets that used this bandwidth make it to their destination. The UDP packets that are dropped at the router R2 have wasted the bandwidth in the R1-R2 link that could have been used by the TCP flow.

The above simulations show that the essential factor behind congestion collapse due to undeliverable packets is neither the scheduling mechanism at the router nor the bandwidth used by a single UDP flow, but instead, the *absence of end-to-end congestion control for UDP flows.* The UDP sources do not care about the bottleneck in the path to the destination and keep sending packets into the network at a much larger rate. If the sources had adjusted their sending rate according to the bottleneck link, the TCP flows could have achieved much higher goodput.

2.3 Previous Work on Unresponsive Flows

Previous research into handling unresponsive flows has taken one of two approaches. The *allocation* approach isolates individual flows by performing a fair allocation of the available bandwidth between competing flows [2, 6, 13, 19, 23, 30, 31, 33]. Most allocation-based schemes require *all* routers to maintain state and perform all operations on a per-flow basis. Stoica et al [33] have proposed CSFQ, a *Core*-State-less allocation scheme, which maintains per-flow state only in edge routers. CSFQ makes a good tradeoff between implementation complexity and perfect fairness. In CSFQ, edge routers maintain per-flow information and estimate the arrival rate of each flow. Based on the this arrival rate estimate, they insert a label into the header of each of the flow's packet passing through. The core routers use FIFO packet scheduling mechanism and do not maintain per-flow information. Supplementary to the FIFO scheduling, is a probabilistic dropping algorithm which drops packets on input. The probability of dropping a packet when it arrives is a function of the arrival rate estimate carried in the packet header and the fair share rate at the core router which is estimated based on measurements of aggregate traffic. The design of our scheme which we refer to as ERUF, resembles that of CSFQ in that both of them try to move per-flow operations to the periphery of the network. However, CSFQ makes its decisions locally at each router and ignores congestion that may be occurring downstream. Therefore, it does not try to drop undeliverable packets as early as possible. Though the algorithm works well in guaranteeing fairness in most cases, it does not solve the problem of congestion collapse due to undeliverable packets.

In the *identification* approach, routers use a lightweight detection algorithm to identify unresponsive flows and then explicitly manage the bandwidth of these flows [10, 11]. Floyd&Fall [11] have proposed a technique which uses RED for queue management and uses the RED packet drop history to estimate arrival rates. Their technique performs tests that identifies flows that are *unresponsive*, *TCP-friendly*, or *high-bandwidth* and regulates them. The core routers perform two tasks: (1) classify an incoming packet as belonging to a regulated flow or an unregulated flow (2) use a scheduling mechanism to restrict the bandwidth usage of the flows that are regulated. Their technique provides an incentive for flows to use end-to-end congestion control as flows marked to be regulated are can be penalized. While ERUF shares their idea of regulating unresponsive traffic, there is one main difference. Packets of the unresponsive traffic are dropped at the edge routers in

ERUF, whereas in their technique, they are dropped at the congested router. Dropping packets at the edge is better because, it saves a lot of network resources from the edge router till the congested router that could be used by other flows that share the path. Dropping packets at the congested router wastes bandwidth from the source till the congestion point. Their technique does not completely solve the problem of congestion collapse due to undeliverable packets. Also, the complexity at the core router is increased due to the classification and complex scheduling to regulate flows.

A related research direction deals with development of UDP-based protocols that are "TCP-friendly" [20, 22, 28, 35, 37, 38]. These protocols use some form of end-to-end congestion control to try to limit the sending rate to a value close to that of a conformant TCP flow. We believe that this is an important research direction; since UDP-based protocols fill an obvious need, developing TCP-friendly versions of them helps reduce the danger of congestion collapse. However, some form of router-based regulation will always be necessary as we can no longer *depend* on cooperation of end-hosts.

Chapter 3

ERUF Architecture

3.1 ERUF Overview

The main goal of our approach, which we which we refer to as *Early Regulation of Unresponsive Flows (ERUF)*, is to drop undeliverable packets as early as possible in their respective paths. Undeliverable packets are those that are dropped somewhere in the network before they reach their destination. It is good to drop such packets early since it would save network resources which can the be gainfully utilized by other competing flows. We classify the routers in the Internet as edge routers - those that form the boundaries, and non-edge routers - those that form the interiors. ERUF has three key aspects. First, edge routers (see Figure 3.1) keep track of incoming flows, their arrival rates and average packet size. Second, non-edge routers use the Random Early Drop (RED) algorithm [12] for queue management and generate ICMP source quench messages on packet drops to advice sources to reduce their sending rates. We assume that the generation of source quenches is suitably rate-limited.[1] Finally, edge routers snoop on source quenches passing through them and use them to control flow-specific token-bucket-based regulators [36]. These regulators adjust their maximum sending rate, referred to as the *regulator bandwidth*, using a multiplicative-decrease/additive-increase discipline. A decrease is triggered by the arrival of one or more source quenches within a single round-trip time (RTT) estimate;[2] an increase is triggered at the end of every RTT only if a quench has not been received in that RTT period.

Since UDP-based streaming protocols usually don't acknowledge individual packets, there is no direct way to estimate round-trip time between the edge router regulating a flow and a congested router downstream. We consider two easy-to-compute approximations: (1) *nexthop*, which is two times the propagation delay on the next link towards the congested router; and (2) $allhops = numhops \times nexthop$, where *numhops* is the number of hops between the edge router and the congested router.

ERUF places the major burden of managing unresponsive flows on the edge routers. Edge routers need to classify incoming packets, compute flow-specific arrival rates and rate-limit individual flows. Since edge routers are typically not on high-speed backbone links and typically do not deal with very large number of flows, the resource requirements for keeping track of their information are likely to be tractable. This approach of moving flow-specific operations to the periphery of the network has been taken by other researchers as well as by several IETF working groups [3, 5, 33].

[1]RFC 1812 [1] proposes several techniques to limit the rate at which source quenches are generated.

[2]Only the first quench causes a decrease, subsequent quenches within a single RTT estimate are ignored.

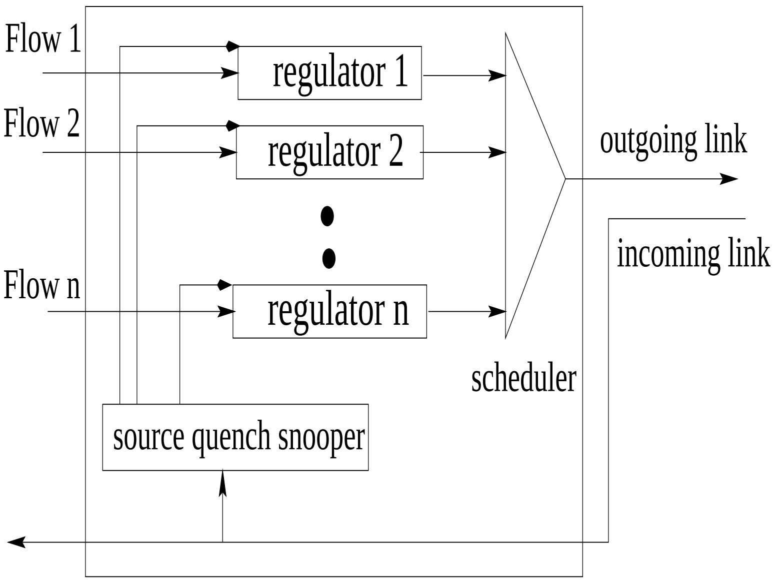


Figure 3.1: Schematic of one interface of an edge router

ERUF requires that congested routers generate (rate-limited) source quenches in response to packet drops. These source quenches are used by edge routers to detect congestion (or incipient congestion) further downstream. There are pros and cons to using source quenches for this purpose. Using source quenches for propagating congestion information to edge routers has three advantages over setting an *explicit congestion notification (ECN)* bit in the packet header (the other main explicit congestion notification technique proposed in the literature [9, 27, 26]). First, source quenches are protocol-independent and can be used for protocols which do not have acknowledgment (*ack*) packets flowing back to the source; ECN-bit-based approaches work only for protocols that include *ack* packets. Second, even for protocols with *ack* packets, using source quenches requires *no* cooperation from the end-hosts (recall that the regulation is done by the edge routers and not the end-hosts). On the other hand, an ECN-bit-based approach relies on cooperation from the receivers to copy the ECN bit into *ack* packets. Finally, in a source-quench-based approach, edge routers need to inspect the relatively rare ICMP packets to extract congestion notifications whereas extracting congestion information from ECN bits would require edge routers to inspect every *ack* packet flowing through them.

The primary argument against using source quenches is that it adds traffic in the reverse direction of the original flow and that this can increase the congestion along paths that have multiple congested routers in both directions. Citing studies done in late 80s [7, 21], the authors of RFC 1812 [1] recommend that routers should *not* generate source quenches. The cited studies, however, were done using routers with a Drop-Tail packet-dropping discipline which can generate a large number of source quenches in a short time. Along with others [9, 18], we believe that the introduction of active queue management techniques such as RED will limit the overhead of source quench messages. In Chapter 4, we present some simulation results for the impact of source quenches on congested reverse paths.

In the rest of this chapter, we describe the details of the ERUF architecture. We present algorithms for installing/maintaining/removing the flow-specific regulators. We would like to emphasize, however, that the main point of this thesis is the overall ERUF technique; while we believe the specific algorithms described work reasonably well for the wide variety of topologies and traffic patterns we have considered in this thesis, we expect them to evolve as the Internet community gains more experience with the dynamics of unresponsive flows.

3.2 Estimating Arrival Rate

Edge routers estimate the arrival rate for each flow. We currently use a simple window-based algorithm (see pseudo-code given below) to estimate the arrival rate of flows. The size of window is dynamically adjusted depending on the burstiness of the flow. The window size is initially set to one and is increased by one for every packet till the estimate of the arrival rate converges. This algorithm worked well (i.e., converged rapidly with windows with 4-7 packets) for all the network topologies and traffic patterns we used in our experiments. In the long run, we plan to use a less memory-intensive scheme - for example, the algorithm proposed by Stoica et al [33] which uses exponential averaging with variable weights.

```
// Pseudo-code of the algorithm used to calculate arrival rate

struct Window window[MAXWINSIZE];
// struct has packet size and packet arrival time
long byteCount = 0;
boolean stabilized = false;

initialize elements of window to 0;
double calculateArrivalRate(int packetSize, double packetArrivalTime)
  int end  = 0;
  double arrivalRate;

  if (stabilized) {
     byteCount = byteCount + packetSize - window[end].size;
     window[end].size = packetSize;
     window[end].time = packetArrivalTime;
     end = (end+1) % MAXWINSIZE
     arrivalRate = byteCount*8/(time-window[end].time);
     winsize = end;
     return arrivalRate;
  }
  else {
      byteCount += packetSize;
      arrivalRate = byteCount*8/(time-window[0].time);
      window[end].size = packetSize;
      window[end].time = packetArrivalTime;
      end = (end+1) % MAXWINSIZE;


      if (end == 0) {
         stabilized = true;
         winSize = MAXWINSIZE;
      }
```

```
        else if (arrival rate has stabilized) { // check for stability
            stabilized = true;
            winSize = end;
            end = 0;
        }

        return arrivalRate;
    }
}
```

3.3 Managing Flow Regulators

Edge routers use *Longest Queue Drop (LQD)* [34] for per-flow buffer management. The idea behind using LQD is that flows that get a higher share of bandwidth than other competing ones tend to have longer queues, and dropping packets from the longest queue should make sharing of bandwidth fairer (in comparison to using drop tail). Initially, no regulator is associated with a flow. As long as a flow receives no source quenches, it is allowed to continue unregulated. When an edge router receives a source quench for a presently unregulated flow, it imposes a token-bucket-based regulator on that flow with the regulator bandwidth value set to half the arrival rate of that flow at that point. The regulators adjust their bandwidth using a multiplicative-decrease/additive-increase discipline. A decrease is triggered by the arrival of one or more source quenches within a single round-trip time (RTT) estimate. When the regulator bandwidth is decreased, it is reduced by half. Within a single RTT, only the first quench causes a decrease in the regulator bandwidth; subsequent quenches are ignored. This protects the regulator from being rapidly driven down by a sequence of source quenches before the source has had a chance to respond to the congestion.[3] Once a regulator has been placed on a flow, a separate process increases the regulator bandwidth by the average packet size of the flow for every RTT in which no quench is received. When the regulator bandwidth becomes greater than or equal to the output link bandwidth, the regulator is removed and the flow is allowed to continue unimpeded.

Note that the algorithm used to tune the regulator bandwidth is similar to the one used to manage the TCP congestion window. However, in contrast to TCP, it is required to operate in an environment that has no acknowledgment (*ack*) packets. Therefore, to detect congestion (or incipient congestion), it uses source quenches. Furthermore, in the absence of *ack* packets, an edge router has no reliable way of determining the round trip time between itself and the congested router that does not involve injecting additional packets into the network. We consider two easy-to-compute approximations in this thesis: (1) *nexthop*, which is two times the propagation delay on the next link towards the congested router;[4] and (2) $allhops = numhops \times nexthop$, where *numhops* is the number of hops between the edge router and the congested router. The first approximation (*nexthop*) is a strict lower bound on the actual RTT and is likely to lead to the regulator bandwidth increasing faster than a conformant TCP would increase its congestion window. This can be exploited by some under-responsive flows to escape ERUF's regulation. This issue is further explored in Section 4.6. The second approximation (*allhops*) is neither a lower bound nor an

[3] A rapid sequence of source quenches can be generated either by a congested router with a Drop-Tail queue management discipline or (more rarely) by a RED router in the presence of extremely bursty traffic.

[4] This estimate has been previously used by Floyd&Fall [10] to compute the maximum sending rate of a TCP flow at the congested router that uses RED.

upper bound on the actual RTT.[5] However, for network topologies with long paths and terrestrial links, it might provide a better approximation to the actual RTT. We empirically investigate the behavior of these approximations in Section 4.2 by using them for a variety of network topologies and traffic patterns.

ERUF does not require fine-grain flow identification and can work with flows of any granularity. However, ERUF works best if a small number of source quenches can significantly reduce the amount of data being sent towards the congested router. ERUF does not work well for traffic patterns with a large number of small unaggregated flows since each source quench results in only a small reduction in the amount of data being sent towards the congested router - each quench can at best reduce the sending rate of a single small flow[6]. Given that routers limit the rate at which source quenches are generated (as they should), it may not be possible to simultaneously quench enough number of the flows. Note that this problem is not specific to ERUF. Traffic consisting of a large number of small unresponsive flows poses a problem for any congestion-management algorithm. Floyd&Fall [11] show that this traffic pattern causes congestion collapse even for scheduling techniques such as *weighted round-robin* which try to isolate individual flows (see Section 2.2). To handle this situation, we propose that edge routers use some degree of flow aggregation and identify flows either by source-destination pairs or by destination alone. Each aggregate is regulated as a whole and a source quench generated for *any* flow in an aggregate results in a decrease in the regulator bandwidth for the entire aggregate. We evaluate the impact of using aggregation in Section 4.4.

The ERUF algorithm is similar, in spirit, to the *SQuID* algorithm proposed by Prue&Postel in RFC 1016 [25]. SQuID specifies *end-host* response to source quenches. There are, however, several differences. First, the SQuID algorithm adjusts the per-packet regulator delay instead of the regulator bandwidth. For flows with variable packet size, this can lead to undesirable variations in the sending rate. Second, the SQuID algorithm uses a additive-increase/additive-decrease discipline for responding to source quenches. Subsequent research and experience has shown that multiplicative-decrease/additive-increase provides better stability [16, 17].

3.4 Implementation in ns-2

We used the ns-2 simulator[7] for our experiments. To simulate ERUF, we added a couple of features to ns-2. First, we implemented an ICMP agent to generate and accept ICMP messages. We defined a new kind of node that has an ICMP agent at port 0. We used this node for all our simulated topologies. By doing this, we ensured that each node always had an ICMP agent. This frees the user creating a topology from having to manually add an ICMP agent to specific nodes in the network. Using a fixed port for the ICMP agent makes it easy to locate the ICMP agent on any node if we know the corresponding node address [8]. For instance, let us assume router R1 has dropped a packet from source H1 and wants to send a source quench message to H1. All that R1 has to do is to find its own ICMP agent (it must have one) and instruct it to send a source quench message to host H1. R1 can easily fill the destination address of the ICMP message since it knows the node address of H1 from the dropped packet's header and since the ICMP agent's port number is fixed. We modified the RED implementation in ns-2 to use this mechanism to send source quench

[5] If the first link towards the congested router is a satellite link and all others are terrestrial, *allhops* can be an overestimate of the actual RTT (depending on queuing delays). If the satellite link is now moved to the second link towards the congested router, *allhops* is likely to be an underestimate of the actual RTT.

[6] By *small* flows, we mean flows whose sending rate is small compared to the bandwidth of the congested link.

[7] Available from *http://www-mash.cs.berkeley.edu/ns/*. We used version 2.1b5.

[8] The address of an agent in ns-2 is a combination of the node address and the port address.

packets.

Second, we implemented the algorithm we described in Section 3.2 to estimate arrival rates and average packet size of each flow. We used the source address and source port for flow identification. For some experiments where flow-aggregation was involved, we used the destination address for flow identification.

Third, we implemented mechanisms to install/maintain/remove regulators. We modified the DRR queuing mechanism in ns-2 to provide a separate queue for each flow identified. [9]

Each queue is associated with a regulator which may or may not be active. When a new flow is detected, its regulator is inactive. Its regulator becomes active when it receives a source quench. The regulation is based on a token-bucket scheme. The maximum sending rate of a regulator is referred to as the *regulator bandwidth*. Each regulator is associated with a token generator that generates tokens into a token pool at a rate defined by the regulator bandwidth. For our experiments, we assigned a value of 100 bytes to each token. Therefore, the token generator would generate a token every *800/(regulator bandwidth)* seconds. For example, if the regulator bandwidth was 1 Mbps, a token would be generated every 800 microseconds. When the regulator is active, packets in the corresponding queue cannot be transmitted unless they are marked "ready to transmit". A packet that is at the head of the packet queue is marked "ready to transmit" if and only if there are sufficient of tokens in the token pool to cover the packet size. For example, a 1500 byte packet at the head of the queue will be marked ready to transmit if and only if there are at least 15 tokens in the token pool. To control the amount of burstiness in flows, there is a limit on the maximum number of tokens that can be generated. This we refer to as the token pool size. For our experiments, we used a token pool size that was 10% larger than what the regulator bandwidth would allow. By varying the regulator bandwidth, we can control the maximum sending rate of the regulator.

Finally, we implemented a mechanism to snoop on ICMP source quench messages on a link and to notify the regulation mechanism. This is done by inserting an object we call the *Source Quench Snooper (SQSnooper)* into the Link object in ns-2. All packets traversing the link would pass through the *SQSnooper* which checks if the packet is an ICMP source quench message. If so, it notifies the regulation mechanism. The regulation mechanism checks if this router is the edge router for the destination of that source quench message.[10] If so, it adjusts the regulator of the associated flow (as described in Section 3.3). The source quench message is then forwarded to the next hop. The *SQSnooper* forwards all other packets without taking any action.

[9] The hashing function used for the DRR implementation in ns-2 can result in collisions when mapping flows onto queues. If collisions are not resolved, more than one flow would share the same regulator. This can cause over-regulation or under-regulation of some of the flows in the aggregate.

[10] For our experiments, we considered the the router one hop away from a host to be the edge router for that source.

Chapter 4

Experiments

We evaluated the performance of ERUF for a variety of simulated network topologies and traffic patterns. We ran each experiment for 500 seconds of simulation time. All of the experiments were done using the *nexthop* approximation unless specified otherwise. The Section 4.2 analyzes the differences between the *nexthop* and *allhops* approximations. For each experiment, we compared the performance achieved with and without ERUF. In the without ERUF case, edge routers use DRR and non-edge routers use RED in the downstream links of a flow. In cases where it mattered, we have compared the results with results obtained by using DRR in all the routers in the network (referred as all-DRR). For unresponsive flows, we used constant-bit-rate UDP flows as might be generated by streaming audio/video applications. For responsive flows, we used *ftp*-type long-term TCP flows. The results presented were obtained using TCP-SACK. We repeated the entire core set of experiments with TCP-Reno, TCP-Tahoe and TCP-Vegas but saw no significant difference in the results. We used 100 byte packets for the CBR flows and 1500 byte packets for the TCP flows.

For the core set of experiments, we simulated five network topologies corresponding to frequently occurring scenarios in the Internet today:

Dumbbell: This scenario corresponds to that of two LANs connected via a long-haul link. One of the hosts is connected to the LAN through a low-speed dial-up link. The traffic consists of a mixture of responsive and unresponsive flows. All the flows share the same source domain (or subnet) and sink domain (or subnet). This is the scenario which we used in Section 2.2 to demonstrate congestion collapse.

Multi-dumbbell: This scenario is a variant of *dumbbell*. The traffic consists of responsive and unresponsive flows. There are multiple unresponsive flows going to the same destination, but originating from different domains/hosts. A common occurrence of this scenario is that of a host receiving streaming content originating from different domains.

Cross-traffic: In the previous two scenarios, competing responsive and unresponsive flows were related in the sense that they shared either the source or the sink domain. In this scenario, the traffic consists of responsive flows and unresponsive flows that are completely unrelated; that is, the responsive flows share neither the source nor the sink domain with the unresponsive flows.

Waist: This is a variant of the *dumbbell* in which there are no low-speed dial-up links. This corresponds to a scenario in which two high-speed domains or subnets are connected via a long-haul link. The traffic consists of a mixture of responsive and unresponsive flows that share the same source domain and sink domain. This is the scenario that we used in Section 2.1 to demonstrate unfairness caused by unresponsive flows to competing responsive flows.

Video-server: The previous scenarios considered traffic patterns containing both responsive and unresponsive flows. This scenario represents situations where all the flows involved are unresponsive. Such a situation can occur for example, with a streaming content provider with multiple server machines. The topology is similar to that of the *dumbbell*.

To compare the two approximations proposed for the round-trip time (RTT) estimate between the edge router and the congested router, we repeated the entire core set of experiments with the *allhops* approximation. For all five topologies and traffic patterns, we found no significant difference between the two approximations (in either the packet drop-rates or the goodput of individual flows). To convince ourselves, we constructed two additional topologies involving links with long delays, which accentuated the differences between the two approximations.

To examine the impact of source quenches on congested reverse paths, we conducted a set of experiments with a network topology that is congested in both directions. Our goal was to determine how effective source quenches are in such a situation and what impact they might have on performance of competing traffic.

To evaluate the efficacy of ERUF for traffic patterns with a large number of *small* flows [1] and to examine the impact of flow aggregation, we simulated two congestion scenarios: one that has a varying number of flows with the same total sending rate and the other that aggregates all flows to the same destination.

To evaluate the performance of ERUF in the presence of multiple congested links we conducted a set of experiments that varied the number of congested links in a single chain.

It is important to ensure that a scheme to regulate unresponsive flows does not over-zealously limit responsive flows as well. To examine the the impact, if any, of the ERUF regulators on conformant TCP flows, we conducted experiments on two topologies with multiple competing TCP flows. We repeated these experiments with the different versions of TCP available in ns-2 - Reno, Tahoe, Vegas and SACK.

To evaluate the ability of ERUF to regulate unresponsive flows, we used three metrics: goodput, aggregate packet drop-rate at edge routers and aggregate packet drop-rate at non-edge routers. The goodput metric shows the performance improvements achieved by the network by dropping undeliverable packets as early as possible. It also shows how well ERUF is able to shield responsive flows from the impact of unresponsive ones. The other two metrics, aggregate packet drop-rate at edge routers and aggregate packet drop-rate at non-edge routers, jointly provide an indication of the resources conserved by dropping undeliverable packets early.

4.1 Core Experiments

4.1.1 Dumbbell scenario

This is the scenario that we used to illustrate congestion collapse due to undeliverable packets in Section 2.2. The topology, shown in Figure 4.1, corresponds to a long-haul link connecting two local-area networks (LAN). The host H4 corresponds to one that is connected to its ISP or corporate network via an ISDN line. In this scenario, the long-haul link is shared by one TCP flow (from H1 to H3) and three constant-bit-rate UDP flows (from H2 to H4). Figure 4.2 compares the goodput and drop-rate for the experiment with ERUF and without ERUF. The x-axis in the goodput graph shows the total sending rate of the three UDP flows from host H2 to host H4. The goodput graphs present two lines; one for the TCP goodput at host H3 and the other for the aggregate UDP goodput at host H4. Note that the graph for without-ERUF is exactly the same as the one in Section 2.2. At higher UDP arrival rates, the TCP flow gets about 25% of the R1-R2

[1]By *small* flows we mean flows whose individual sending rate is low.

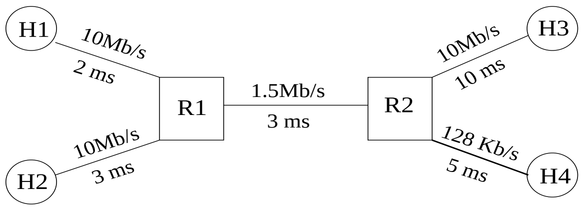


Figure 4.1: Simulation network for the *dumbbell* scenario. The traffic consists of one TCP flow from H1 to H3 and three equal CBR flows from H2 to H4. The total sending rate of the CBR flows is varied between 10 Kb/s to 2 Mb/s.

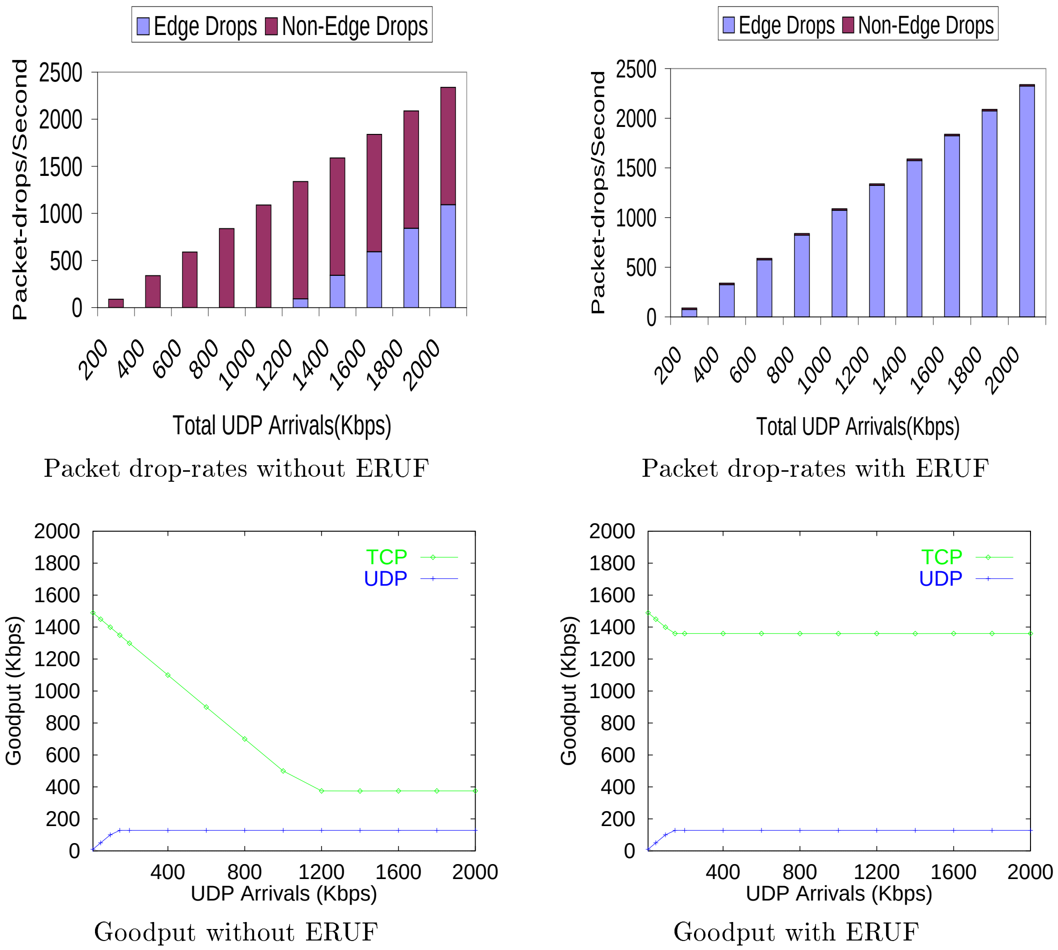
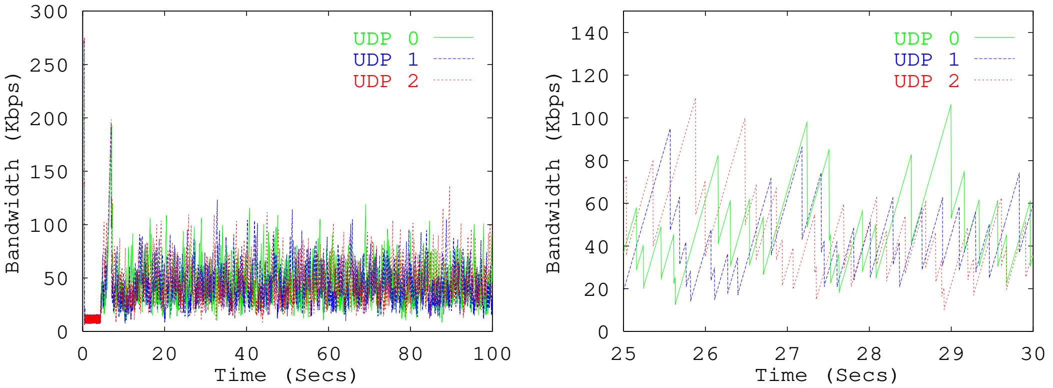


Figure 4.2: Goodput and packet drop-rate for the *dumbbell* scenario.



Variation over a period of 100 seconds   Variation over a period of 5 seconds

Figure 4.3: Regulator bandwidth variation for the *dumbbell* scenario

link bandwidth. Even though the UDP flows get about 75% of the R1-R2 link bandwidth, they are able to get a goodput of only 128 Kbps which is about just 8.5% of the bandwidth they use. This means that about 90% of the R1-R2 link bandwidth used by the UDP flows does not contribute to their goodput and is used by the packets that are dropped at the bottleneck link R2-S4. This is substantiated by the drop-rate graph. The The drop-rate graph shows the number of packets dropped per second at the router R1 and the router R2. The graph for without-ERUF shows that a large fraction of the packets that are dropped in this case are at the non-edge router R2. The edge drops increase mainly after the UDP sending rate becomes grater than the R1-R2 link bandwidth.

With ERUF, the regulation at the edge router results in a lot of packets of the UDP flows being dropped at the router R1. This saves a lot of bandwidth in the R1-R2 link that is used by the TCP to increase its goodput by a factor of up to 3.5. It should be noted that the UDP goodput has not decreased at all from the maximum achievable value. We note that the total number of packets dropped are about the same as the case without ERUF, but most of the packets are dropped at the edge router R1 closest to the source. The source quenches generated at the router R2 give valuable feedback to the edge router R1 to regulate the unresponsive UDP traffic.

To illustrate the effect of the regulator, Figure 4.3 shows the regulator bandwidth variation for the UDP flows when their aggregate sending rate is 1.6 Mbps. We see that the regulator bandwidth for each of the three UDP flows oscillate around 50 Kbps. The saw-tooth pattern seen in the figure is characteristic of multiplicative-decrease/additive-increase control mechanisms. A similar pattern is usually seen for TCP's congestion window variation. Also, it can be seen that the variation is not fair at a fine time granularity. At any particular instant of time, the regulator bandwidth for one of the three UDP flows is often more than the other two.

4.1.2 Multi-dumbbell scenario

In the *dumbbell* scenario, all three CBR flows being regulated originated from the same domain (or subnet). We used the *multi-dumbbell* scenario, shown in Figure 4.4, to evaluate the effectiveness of ERUF's regulation when there are multiple unresponsive flows originating from different domains (or subnets) going to the same destination. In this variant of the *dumbbell*, the traffic consists of two CBR flows from H2 to H4, one CBR flow from H5 to H4 and one TCP flow from H1 to H3.

Figure 4.5 shows the goodput and packet drop-rate for this experiment using under the three different mechanisms - without-ERUF, all-DRR and with-ERUF. The x-axes in both the graphs show the aggregate sending rate of all UDP flows into the network. Without ERUF, the TCP

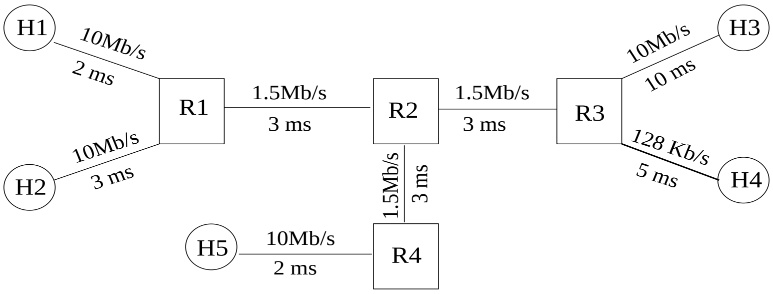


Figure 4.4: Simulation network for the *multi-dumbbell* scenario. The traffic consists of one TCP flow from H1 to H3, two CBR flows from H2 to H4 and one CBR flow from H5 to H4. All CBR flows have equal sending rates; the total sending rate of the CBR flows is varied between 10 Kb/s to 2 Mb/s.

goodput falls almost linearly with increasing UDP sending rate. When the UDP sending rate is more than 1.5 Mbps, the TCP goodput drops to zero. This is because, the router R2 uses RED and provides no bandwidth guarantee for the TCP flow. RED drops packets of both the UDP and TCP traffic, but only the TCP flow responds by reducing its sending rate. Since the congestion is persistent, always *some* of the TCP packets get dropped at router R2 and eventually, the TCP flow is effectively shut off. The packets dropped at router R2 and at router R3 (R3-R4 link) contribute to the non-edge drops shown in the drop-rate graph. There are very few edge drops because the total arrival rate of the flows (UDP and TCP) at individual edge routers are not high enough to cause buffer overflow (the edge drops do no show up in the graph because the non-edge drops are relatively very high).

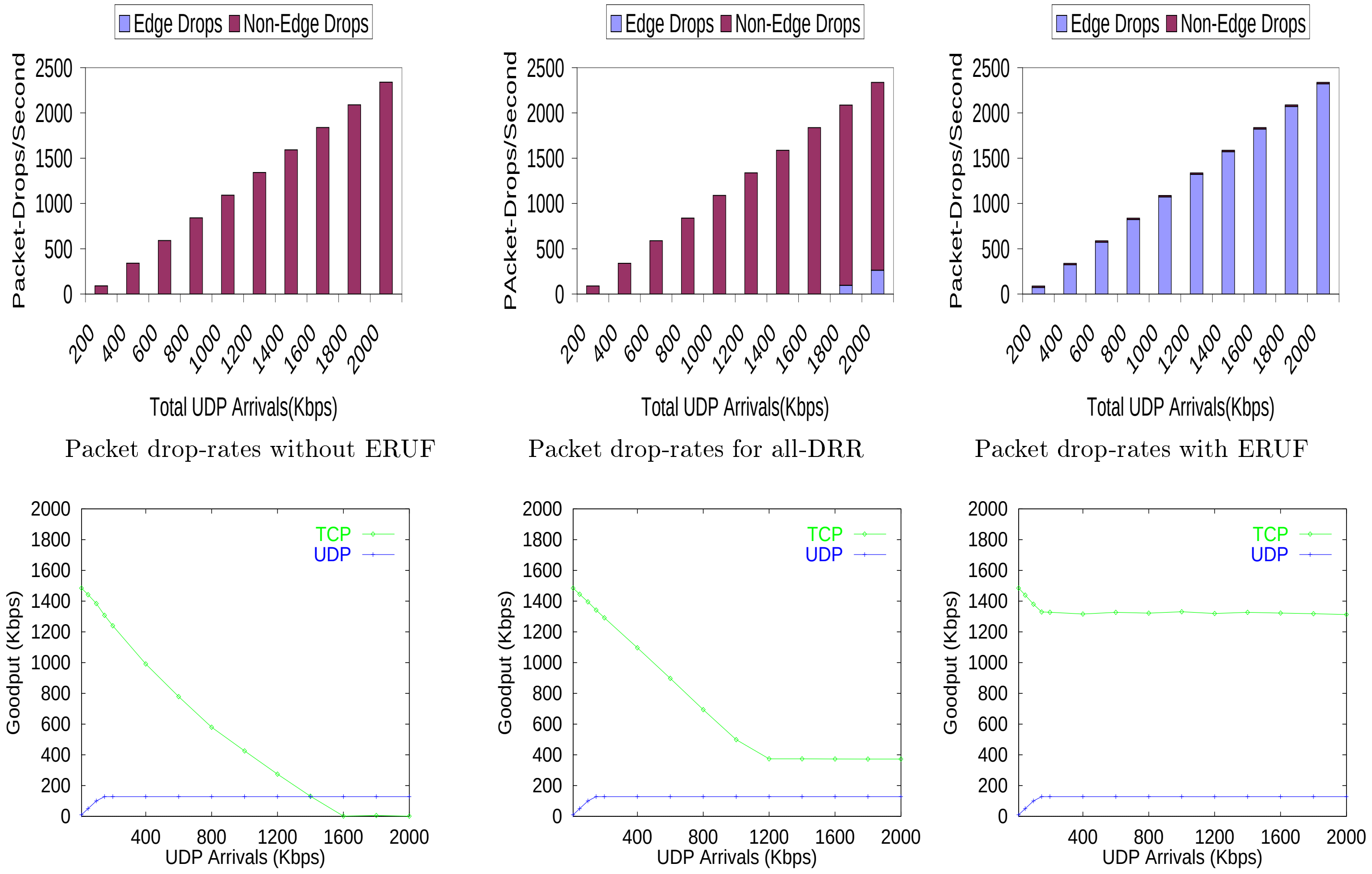
The goodput graph for all-DRR is similar to the one without ERUF until the UDP sending rate reaches 1.2 Mbps. After this, the TCP goodput flattens out at about 400 Kbps, roughly 25% of 1.5 Mbps. This is due to the fact that the DRR scheduler at router R2 guarantees the TCP flow 25% of the R2-R3 link bandwidth. The drop-rate graph is also similar to the one without ERUF except for a few edge-drops for the higher UDP sending rates. These edge-drops occur due to the increased total arrival rate for all flows at R1 since the TCP sending rate is more compared to without ERUF for the same UDP sending rates.

With ERUF, we see that the TCP flow achieves a near-ideal goodput. The aggregate TCP goodput is improved by a factor of 11 compared to without ERUF. This is because ERUF is able to drop undeliverable packets of the UDP flows at their respective edge routers.

Figure 4.6 shows the regulator bandwidth variation for the UDP flows. Similar to the *dumbbell* scenario, the regulator bandwidth oscillates around 50 Kbps. Regulation saves a lot of bandwidth in the congested link which is used by the TCP flow to achieve its performance. This is reflected in the drop-rate graph which shows that while the total number of packets dropped are the same for all three schemes, with-ERUF is able to drop almost all of the dropped packets at edge routers whereas most of the packets dropped for without ERUF and all-DRR occur inside the core network. There are very few non-edge drops. As in the *dumbbell* case, it can be noted that the total number of packet drops is almost the same in all cases and the UDP goodput is also at its maximum in all the cases.

4.1.3 Cross-traffic scenario

In the first two scenarios, the competing TCP and CBR flows were either sourced from the same domain or sink'ed in the same domain. This scenario, shown in Figure 4.7, represents cross-traffic



Packet drop-rates without ERUF

Packet drop-rates for all-DRR

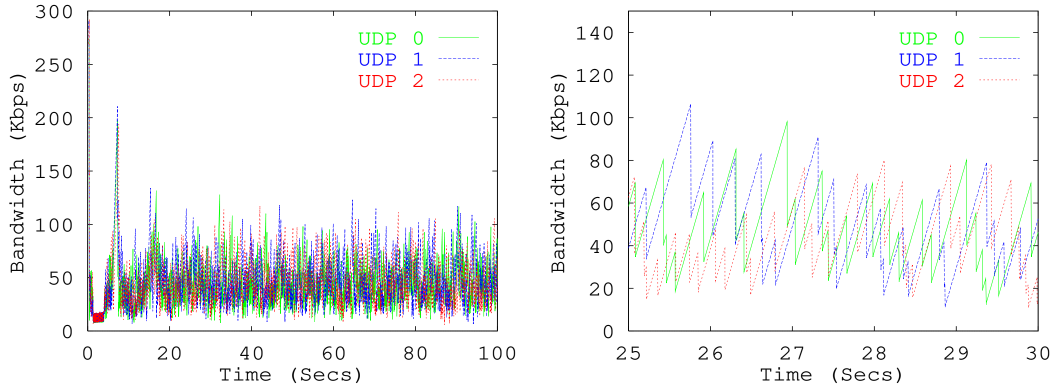
Packet drop-rates with ERUF

Goodput without ERUF

Goodput for all-DRR

Goodput with ERUF

Figure 4.5: Goodput and packet droprate for the *multi-dumbbell* topology.



Variation over a period of 100 seconds   Variation over a period of 5 seconds

Figure 4.6: Regulator bandwidth variation for the *multi-dumbbell* scenario

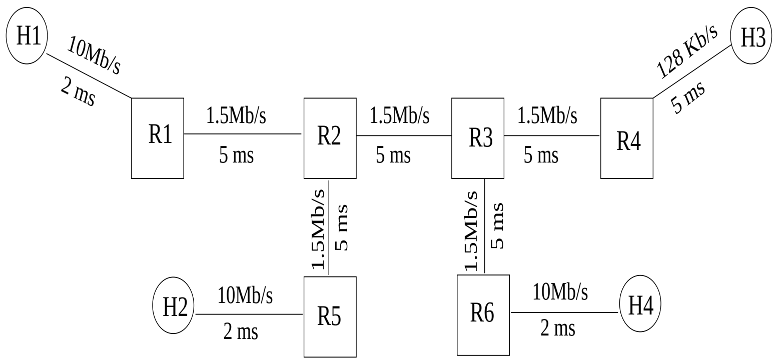
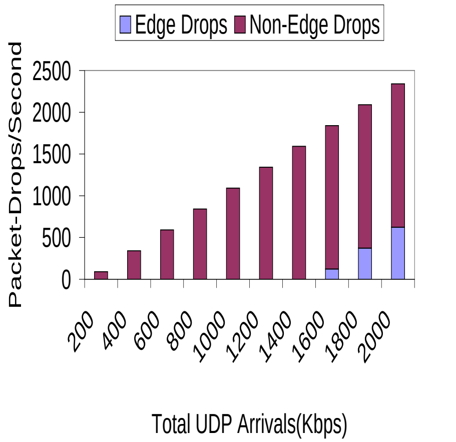
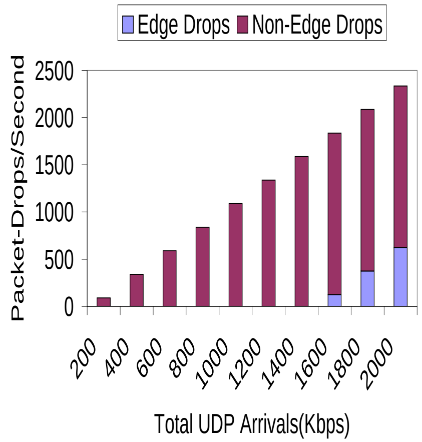


Figure 4.7: Simulation network for the *cross-traffic* scenario. The traffic consists of one TCP flow from H2 to H4 and three CBR flows from H1 to H3. All CBR flows have equal sending rate; the total sending rate of the CBR flows is varied between 10 Kb/s to 2 Mb/s.

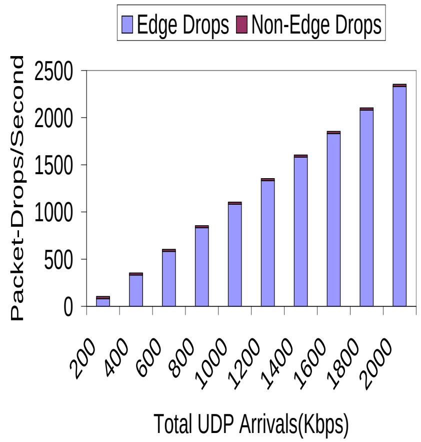
situations where competing flows share neither source nor sink. The traffic consists of three CBR flows from H1 to H3 and one TCP flow from H2 to H4. We chose this scenario to demonstrate that unresponsive traffic from one domain can affect responsive traffic from some other domain and to evaluate ERUF's performance in such cases. Figure 4.8 shows the drop-rate and goodput graphs without ERUF, for all-DRR and with ERUF. Without ERUF, we see that the TCP goodput falls rapidly as the UDP sending rate increases. This behavior is similar to that of the *multi-dumbbell* scenario. However, the performance of without-ERUF scenarios differ for higher UDP arrival rates. In this scenario, the TCP goodput stabilizes at about 70 Kbps whereas in the *multi-dumbbell* case, the TCP dwindles to nothing. Intuitively, this is due to the fact that the TCP flow is able to achieve a larger arrival rate at router R2 for *cross-traffic* than for *multi-dumbbell*. In the *multi-dumbbell* case, the TCP has to compete with the UDP flows at the edge router R1 for the R1-R2 link bandwidth. But in *cross-traffic*, TCP competes with no other flow at the edge. As a result, the TCP flow manages to achieve a little better goodput in this scenario than in the *multi-dumbbell*. There is also a difference in the drop-rate graphs. There are no edge drops in *multi-dumbbell* for without-ERUF and all-DRR, but both these cases result in some edge drops in *cross-traffic*. This is because the UDP arrivals into the network are split between router R1 and R4 in *multi-dumbbell*. As a result, the UDP arrival rate at neither of these routers is large enough to cause buffer overflow. But for *cross-traffic*, all three UDP flows compete for the R1-R2 link and their aggregate arrival



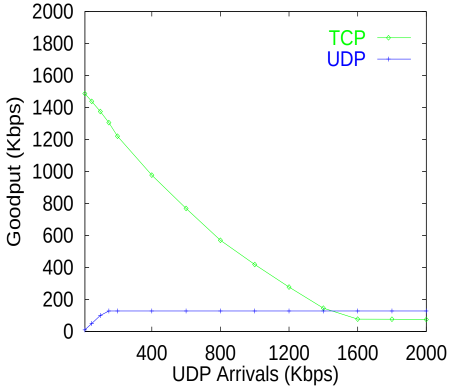
Packet drop-rates without ERUF



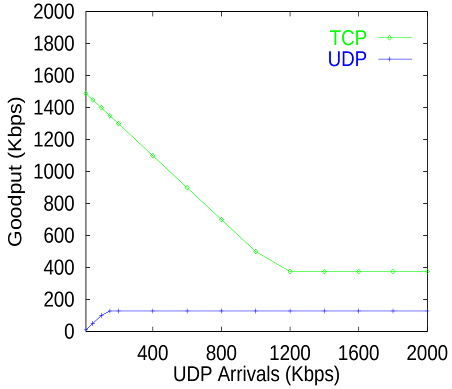
Packet drop-rates for all-DRR



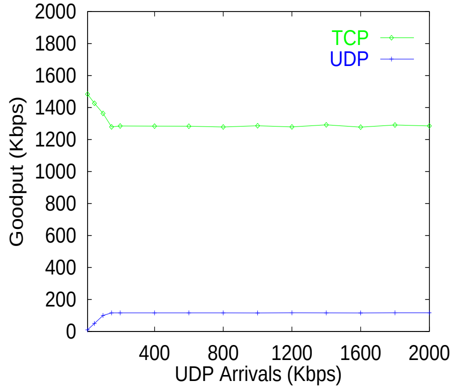
Packet drop-rates with ERUF



Goodput without ERUF

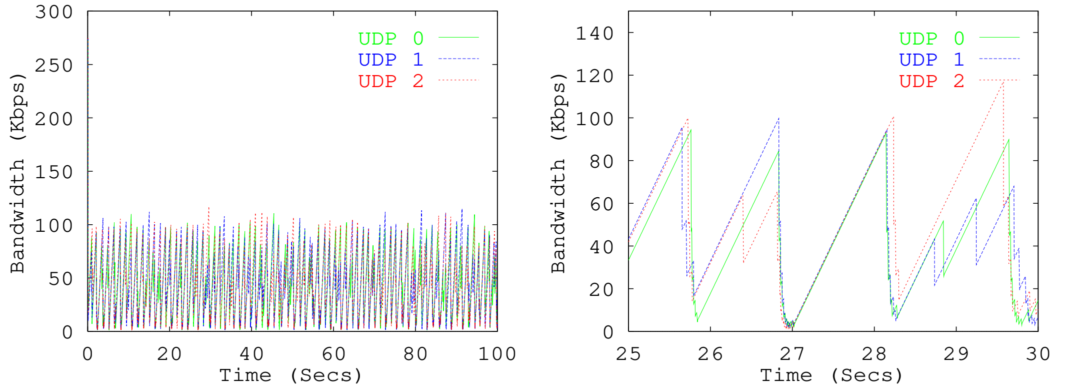


Goodput for all-DRR



Goodput with ERUF

Figure 4.8: Goodput and packet drop-rate for the *cross-traffic* scenario



Variation over a period of 100 seconds — Variation over a period of 5 seconds

Figure 4.9: Regulator bandwidth variation for the *cross-traffic* scenario

rate at router R1 is larger than the capacity at all times.

In the case of all-DRR, each of the four flows get about 25% of the R2-R3 link bandwidth. Therefore, at higher UDP arrival rates, the TCP goodput stabilizes at about 375 Kbps. This is exactly same as in the *multi-dumbbell* case. The drop-rate graph is similar to the case without ERUF. The number of edge drops is the same because the total arrival rate at the edge routers remains the same. The number of non-edge drops also remain almost the same because, the TCP drops for without ERUF are converted to UDP drops for all-DRR. Comparing the drop-rate graph with the corresponding graph for *multi-dumbbell*, we see that there are more edge-drops in *cross-traffic*. As in without ERUF, this is because the aggregate arrival rate at the edge router is more in the *cross-traffic* scenario than in the *multi-dumbbell*.

With ERUF, we see that most of the non-edge drops are converted to edge-drops. Consequently, the TCP gets more bandwidth in the R2-R3 link and improves its goodput (about three times more than all-DRR when the UDP rate is 2 Mbps). As for the previous scenarios the total number of drops remains the same in all the three scheduling mechanisms.

Figure 4.9 shows the regulator bandwidth variations for the UDP flows when their aggregate sending rate is 1.6 Mbps. As for the previous scenarios, the regulator bandwidth oscillates around 50 Kbps. However, the ramp-up is slower for this scenario since the R1-R2 link has a larger delay (5 ms) than the corresponding links for the previous scenarios (3 ms). Since the ramp-up is not fast, it is able to climb more (up to 100 Kbps) than the *dumbbell* and the *multi-dumbbell* cases. This is because a slower ramp-up results in a slower increase in average queue length and hence, fewer drops. But it is also driven down more (below 20 Kbps) by a train of source quenches. Comparing the goodput graphs for *cross-traffic* and *multi-dumbbell*, we see that the goodput of TCP and UDP for *cross-traffic* are a little less than in *multi-dumbbell*. This is because, the arrival rate of both the UDP and TCP traffic at the router R2 is more in *cross-traffic* than in the *multi-dumbbell* case. Consequently, there are more non-edge drops which generate more source quenches which in turn cause more regulation.

4.1.4 Waist scenario

This scenario, shown in Figure 4.10, is similar to the the topology we used in Section 2.1 to illustrate the problem of unfairness caused by unresponsive flows. Note that this scenario is similar to the *dumbbell* with the 128 Kbps link replaced by a 10 Mbps link. It corresponds to two local area networks or two domains connected via a long-haul network. The traffic pattern is the same

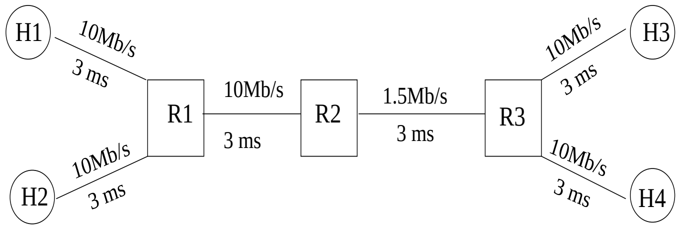


Figure 4.10: Simulation network for the *waist* scenario. The traffic consists of one TCP flow from H1 to H5, one CBR flow from H2 to H6,one CBR flow from H3 to H7 and one CBR flow from H4 to H8. All CBR flows have equal sending rates; the total sending rate of the CBR flows is varied between 10 Kb/s to 2 Mb/s.

as in the *dumbbell* - three UDP flows from H2 to H4 and one TCP flow from H1 to H3. Figure 4.11 shows the drop-rate and goodput graphs for the *waist* topology without ERUF, for all-DRR and with ERUF. The goodput graph for without-ERUF shows that at lower UDP arrival rates, the TCP flow gets more bandwidth and as the UDP arrival rate approaches the link bandwidth, the TCP flow gets lesser and lesser of the shared link's bandwidth. Once the UDP flows' sending rate increases beyond the shared link's capacity, the TCP flow is shut off. The non-edge drops shown in the drop-rate graph is due to the total arrival rate of the flows - TCP and UDP - at the router R2, which exceeds the capacity of the R2-R3 link. There are no edge-drops because the total arrival rate at R1 never exceeds the R1-R2 link capacity.

With all-DRR, each of the four flows (three UDP and one TCP) gets about 25% of the R2-R3 link bandwidth. Therefore, the TCP flow gets about 375 Kbps and the three UDP flows together get about 1125 Kbps of the shared link's bandwidth. We see non-edge drops once the aggregate UDP sending rate exceeds 1125 Kbps. Since the UDP flows are restricted to 3/4 of the R2-R3 link bandwidth, we see more non-edge drops than in the case without ERUF.

With ERUF, the non-edge drops generate source quench packets and the edge router R1 applies regulators on the corresponding flows. Figure 4.14 shows the regulator bandwidth variation for both the TCP and UDP flows.[2] We see that the UDP flows are regulated much more than the TCP. Since the UDP flows do not respond to congestion, they contribute to more non-edge drops and receive more source quenches. Therefore, a lot of packets of the UDP flows are dropped at the edge router R1 and the UDP flows get lesser than their fair share of 1125 Kbps. This is reflected by the drop-rate graph which shows more edge drops and relatively very few non-edge drops. The regulation of the UDP flows allows the TCP flow to perform better than in the all-DRR case. Note that ERUF does not guarantee fairness. The more a flow contributes to congestion, the more it is regulated. It should be kept in mind that even though all-DRR guarantees fairness in this case, it is very expensive to provide per-flow queuing at the non-edge routers at least with today's technology. Also, it is not clear if fairness is a good thing in this case.

4.1.5 Video-server scenario

The previous scenarios considered traffic patterns with competition between responsive and unresponsive flows. This scenario, shown in Figure 4.15, represents situations in which all the competing flows are unresponsive. Such a situation would occur, for example, with a streaming-content provider when multiple server machines are streaming data to many clients. In this scenario, the traffic pattern consists of two CBR flows from H1 to H3 and three CBR flows from H2 to H4. Note

[2]We did not show bandwidth variation for TCP in the previous scenarios because in those cases, the UDP flows were the only cause of the congestion collapse and their regulation is more significant than TCP's.

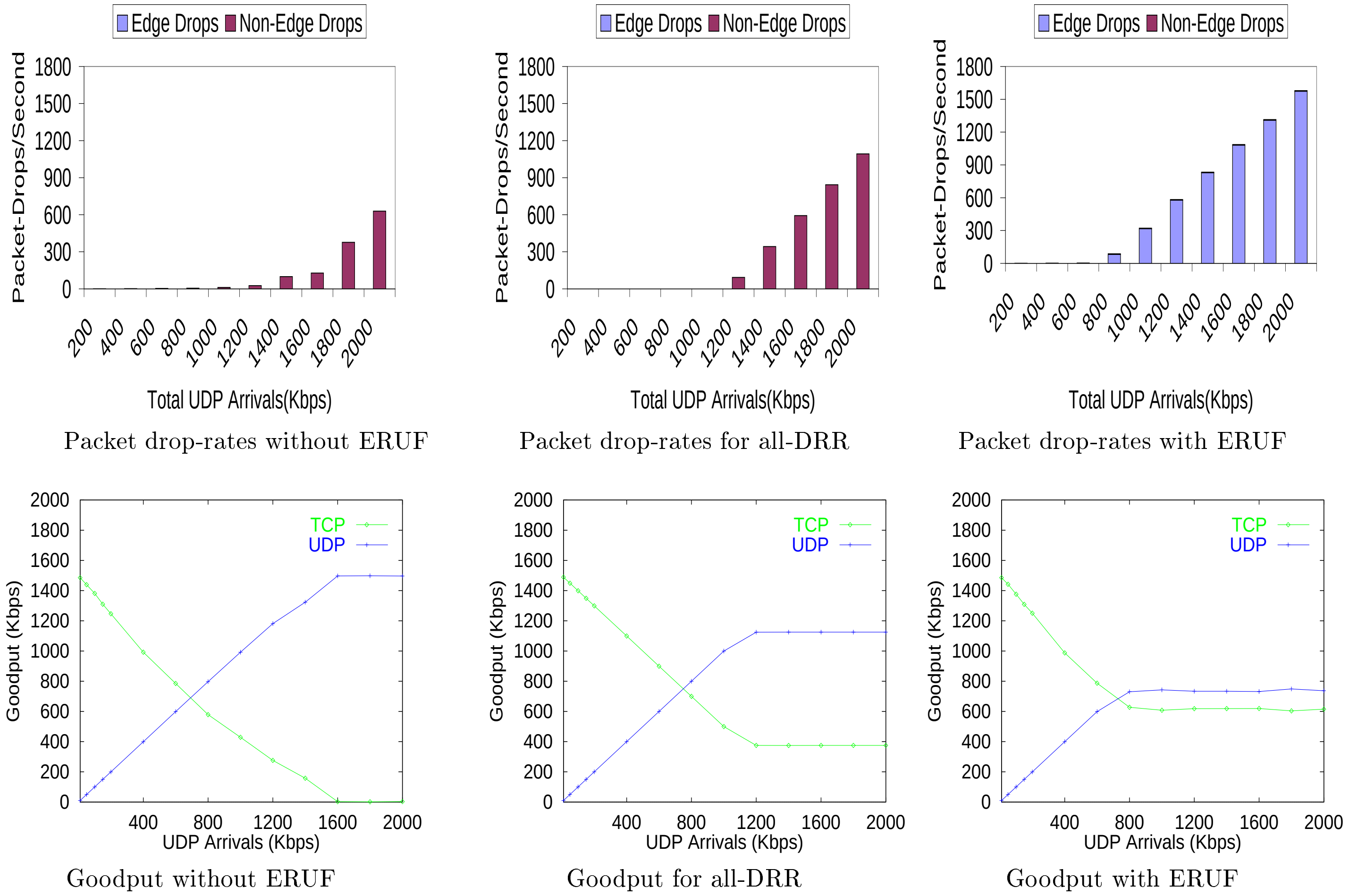


Figure 4.11: Goodput and packet drop-rate for the *waist* scenario

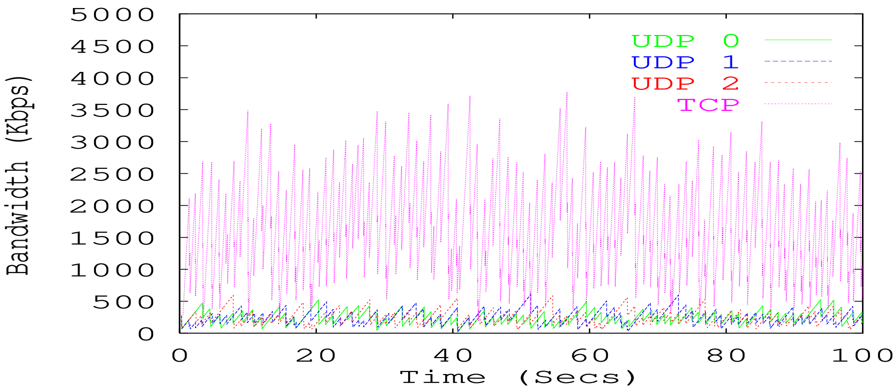


Figure 4.12: TCP/UDP regulator bandwidth variation for the *waist* scenario over a period of 100 seconds

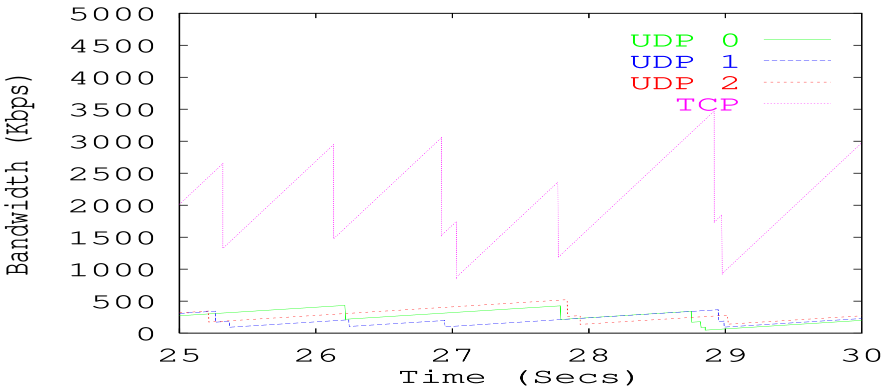


Figure 4.13: TCP/UDP regulator bandwidth variation for the *waist* scenario over a period of 5 seconds

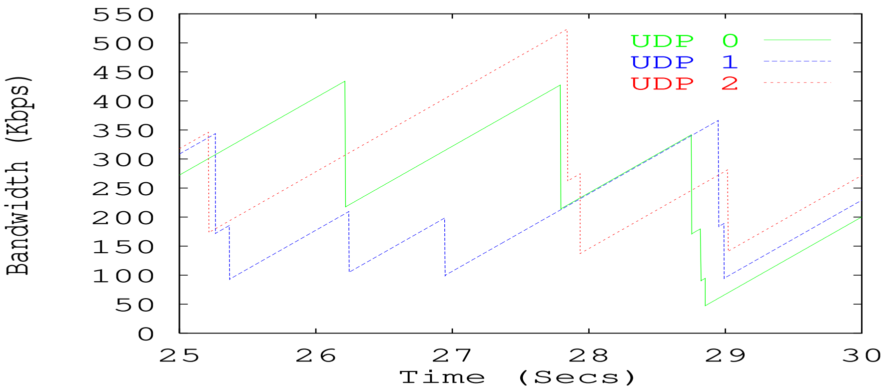


Figure 4.14: UDP regulator bandwidth variation for the *waist* scenario over a period of 5 seconds

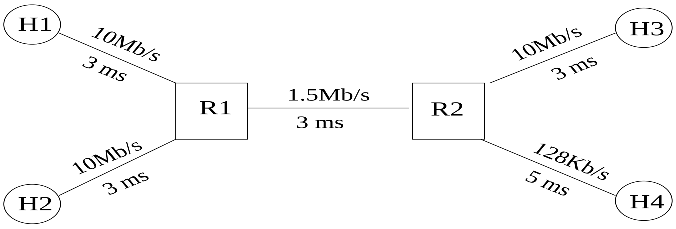
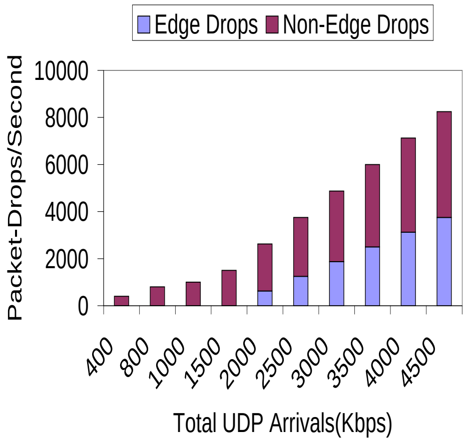
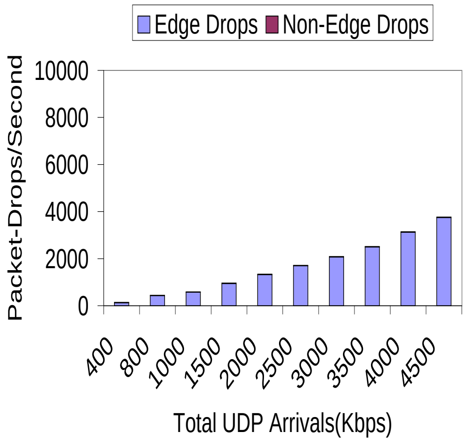


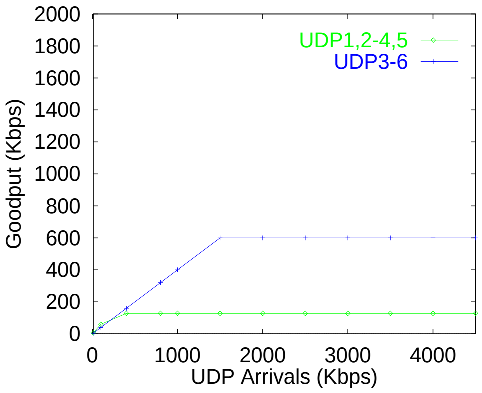
Figure 4.15: Simulation network for the *video-server* scenario. The traffic consists of two CBR flows from H1 to H3 and three CBR flows from H2 to H4. All CBR flows have equal sending rates; the total sending rate of the CBR flows is varied between 10 Kb/s to 4.5 Mb/s.



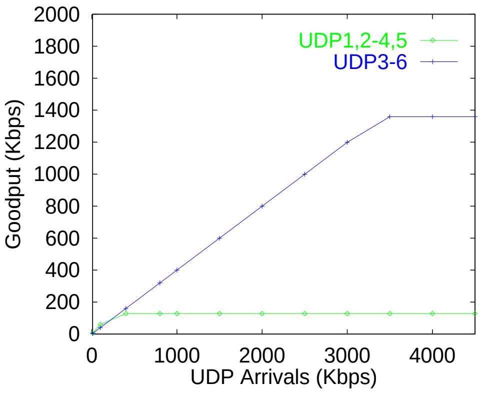
Packet drop-rates without ERUF



Packet drop-rates with ERUF

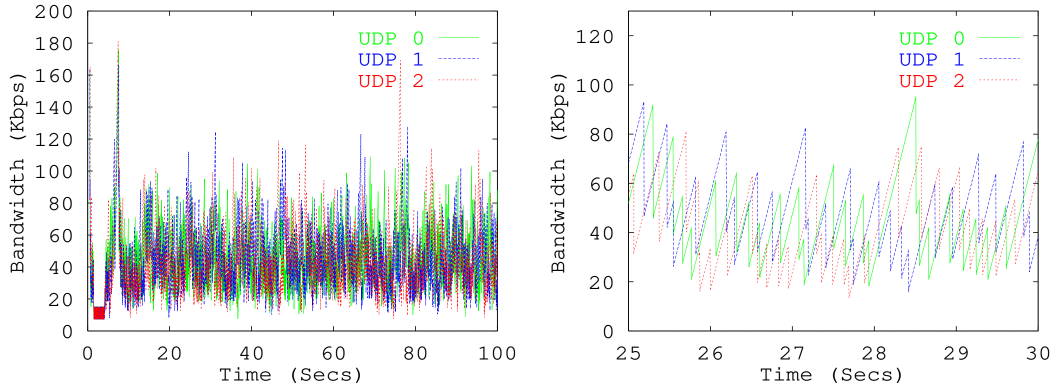


Goodput without ERUF



Goodput with ERUF

Figure 4.16: Goodput and packet drop-rate for the *video-server* scenario.



Variation over a period of 100 seconds   Variation over a period of 5 seconds

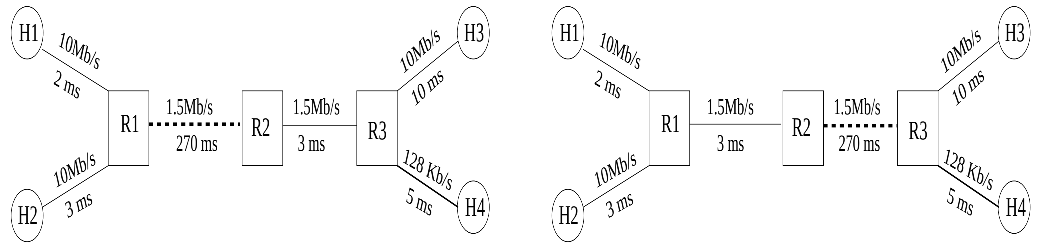
Figure 4.17: Regulator bandwidth variation for the *video-server* scenario

that this scenario is similar to the *dumbbell* scenario, the main difference being that all competing flows are unresponsive.

Figure 4.16 shows the packet drop-rate and goodput graphs for the *video-server* scenario. The goodput graph shows the aggregate goodput of the flows from H1 to H3 (referred hence as F_1^3) and the flows from H2 to H4 (referred hence as F_2^4) for different aggregate UDP arrival rates. Without ERUF, we see that the F_1^3 goodput flattens at about 600 Kbps. This is because, the DRR algorithm at router R1 gives an equal share of the R1-R2 link bandwidth among the five competing flows. Each flow gets about 300 Kbps of the R1-R2 link bandwidth. But this is more than necessary for the flows F_2^4 because their aggregate goodput cannot exceed 128 Kbps. The packets of F_2^4 that are dropped at router R2 just waste the bandwidth in the R1-R2 link. This can be seen from the drop-rate graph as well which shows an increasing number of non-edge drops as the UDP arrival rate at router R1 increases. On the other hand, with ERUF, we see very few non-edge drops. This is because the flows F_2^4 are regulated due to the source quenches generated by the packets that are dropped at the non-edge router R2. Figure 4.17 shows the regulator bandwidth variation for F_2^4. As expected, it oscillates around 50 Kbps. Regulation increases the bandwidth available for flow F_1^3 in the R1-R2 link and hence we see an increase in their goodput compared to the case without ERUF. It is interesting to note that the number of drops is lesser than without ERUF. This is different from the *dumbbell* case. The reason for this difference is that in the *dumbbell* scenario, the TCP flow's sending rate is higher compared to that of flows F_1^3 in this case. This causes more buffer overflows at the router R1. This is also why we varied the UDP arrival rate up to 4500 Kbps in this scenario (so that we can see when the goodput of flows F_1^3 flattens). Another point to be noted when comparing these results with that of *dumbbell* is that the x-axis in the *dumbbell* shows aggregate sending rate of F_2^4 only but in this case it shows the aggregate sending rate of F_2^4 and F_1^3.

4.2 Evaluating RTT Approximations

To compare the two approximations proposed for the round-trip time (RTT) between the edge router and the congested router, we repeated the entire core set of experiments with the *allhops* approximation. For all five topologies and traffic patterns, we found no significant difference between the two approximations (in either the packet drop-rates or the goodput of individual flows). To convince ourselves, we constructed two additional topologies involving links with long delays,



NearSat *FarSat*

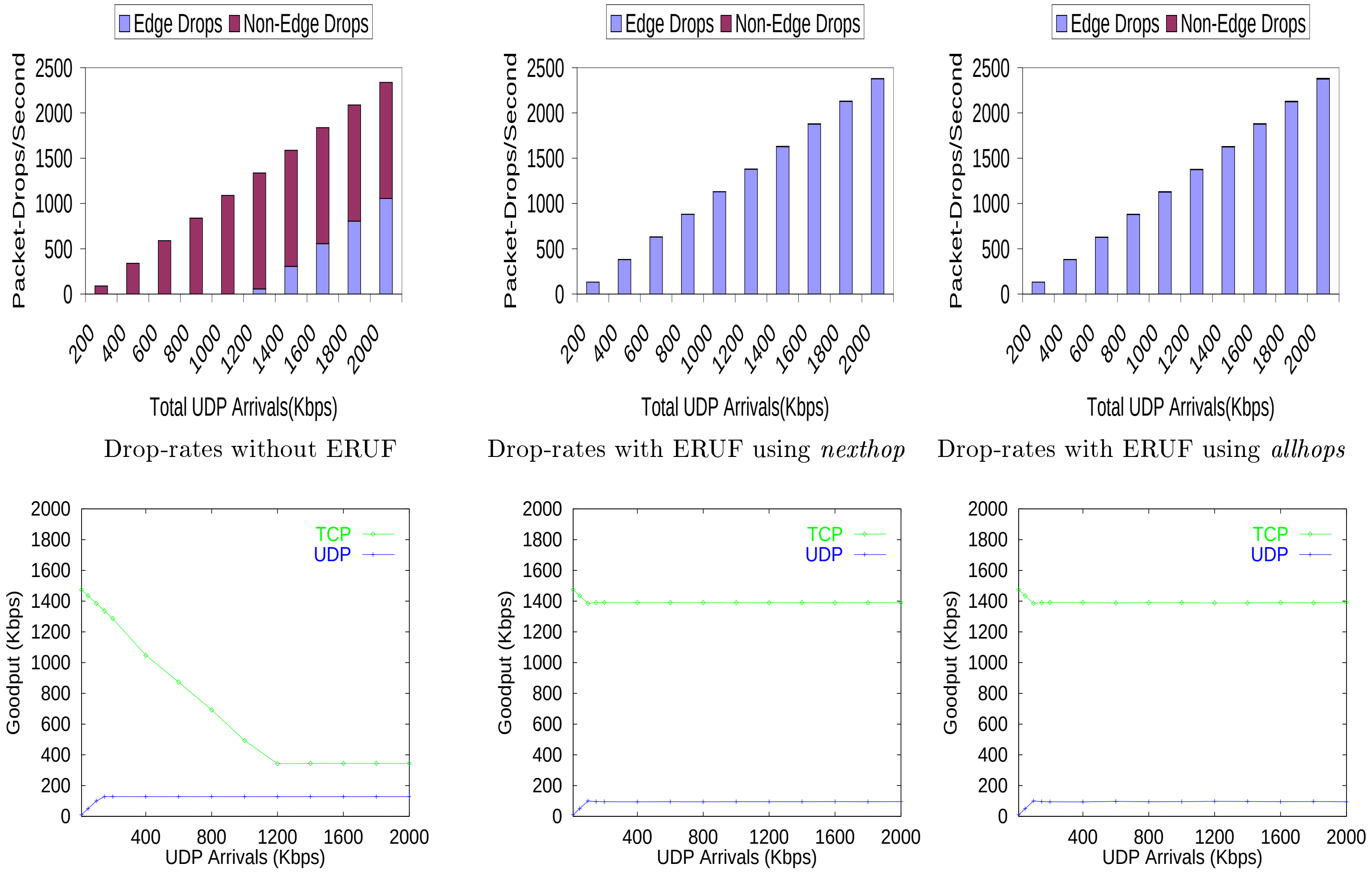
Figure 4.18: Simulation network for the *NearSat* and *FarSat* topologies. The satellite links are shown by dashed lines. The traffic consists of one TCP flow from H1 to H3, and three CBR flows from H2 to H4. All CBR flows have equal sending rates; the total sending rate of the CBR flows is varied between 10 Kb/s to 2 Mb/s.

which accentuated the differences between the two approximations. These topologies are variants of *dumbbell* with an additional satellite link inserted. The first topology, *NearSat*, has the satellite link next to the edge router and the other, *FarSat*, has the satellite link one hop away from the edge router. For *NearSat*, *nexthop* is a reasonable approximation for RTT and *allhops* is a loose over-estimate. For *FarSat*, both *nexthop* and *allhops* are loose under-estimates.

Figure 4.19 and Figure 4.20 show the results for these experiments. Without ERUF, the graphs are similar to the *dumbbell* scenario. With ERUF, in both the *NearSat* and *FarSat* cases, most of the undeliverable packets are dropped at the edge which results in a better performance for the TCP flow. However, there are some differences between the *NearSat* and the *FarSat* cases. The immediately visible difference in the graphs for the two topologies is that, with *NearSat*, TCP's goodput is better. This can be explained using the regulator bandwidth variation shown in Figure 4.22 for the UDP flows at the edge router R1. In the *NearSat* case, the regulator bandwidth ramps up much slower than in the *FarSat* case. This is expected because for*NearSat*, the estimated RTT is 540 ms using *nexthop* and 1080 ms using *allhops*. For *FarSat* the estimated RTT is 3 ms and 6 ms using *nexthop* and *allhops* respectively. The estimated RTT in the *NearSat* cases is much higher than the *FarSat* cases. The slower climb of the regulator bandwidth causes more number of edge drops for the UDP flows in the *NearSat* cases. Therefore the *NearSat* UDP flows use lesser bandwidth in the downstream links which gives the competing TCP flows more bandwidth. As a result, TCP achieves better goodput in the *NearSat* cases.

Now let us analyze the differences caused by the approximation techniques in each of the two cases. First, the *NearSat* case. We see from Figure 4.22 that the regulator bandwidth using *allhops* ramps up slower than using *nexthop* (as the RTT estimate is larger). But since the ramp-up is slower, RED at router R3 starts dropping packets a little later and hence the regulator bandwidth is driven down fewer number of times (by source quenches) than *nexthop*. Also, using *allhops*, more source quenches are ignored due to the larger RTT estimate. This means the regulator bandwidth will be driven down less frequently. This offsets for the slower ramp-up and hence we don't see much difference in the goodput graphs. In *FarSat*, in both cases, the pattern of the regulator bandwidth variation has high spikes, and between the spikes, the bandwidth oscillates around a lower value. In the *allhops* case, the regulator bandwidth stays below 50 Kbps for a longer time. In the *nexhop* case, between the spikes, the bandwidth climbs to a comparatively higher value due to the faster ramp up. This results in lower goodput. The variation pattern in both cases are determined by various constants in the topology - delays, buffer sizes, bandwidth etc.

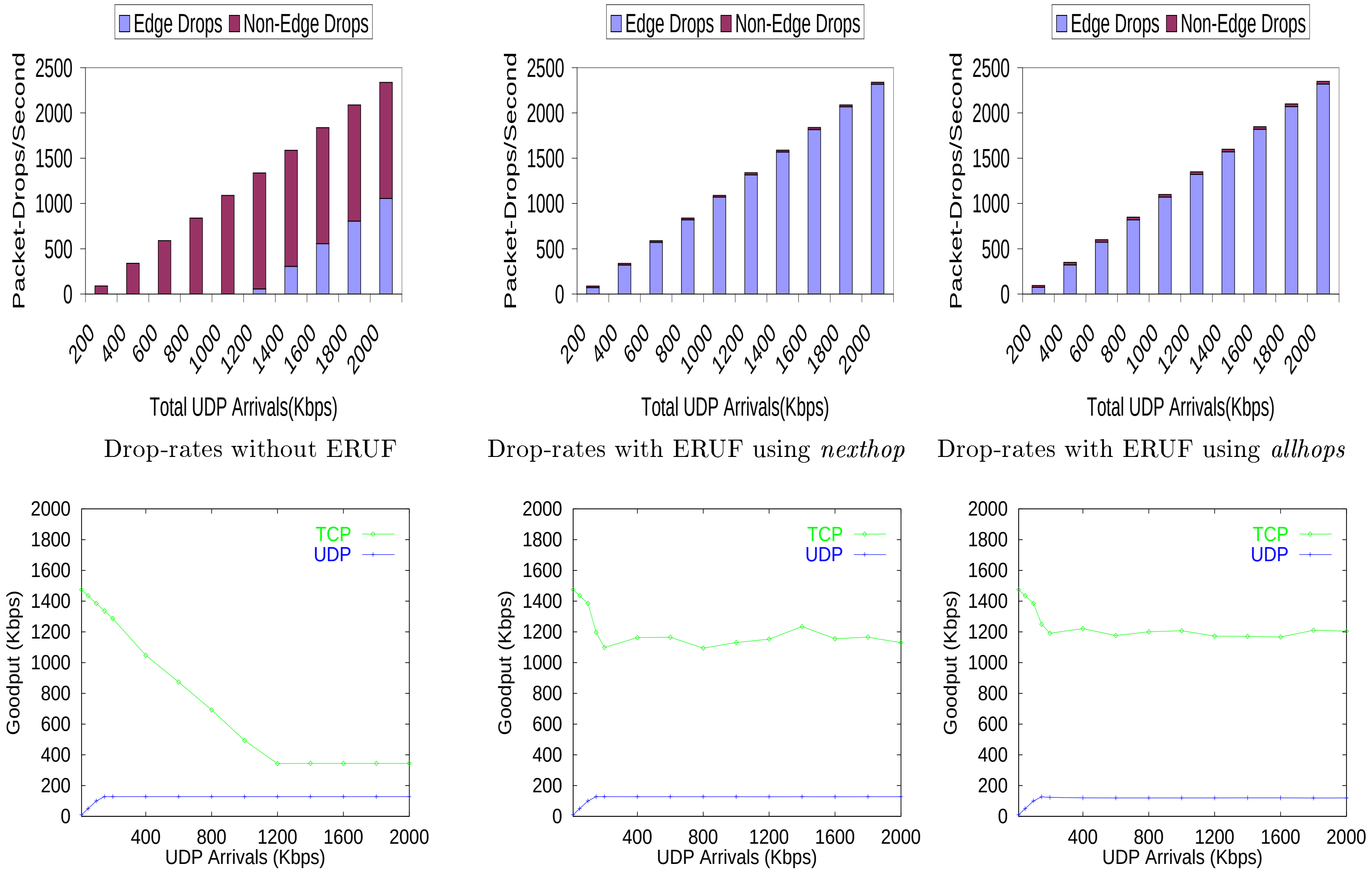
In conclusion, both the approximations for RTT estimates work well for topologies where the delays of the links involved do not differ by a large extent. In cases where the delays differ by a



Drop-rates without ERUF — Drop-rates with ERUF using *nexthop* — Drop-rates with ERUF using *allhops*

Goodput without ERUF — Goodput with ERUF using *nexthop* — Goodput with ERUF using *allhops*

Figure 4.19: Goodput and and packet drop-rate for the *NearSat* topology.



Drop-rates without ERUF

Drop-rates with ERUF using *nexthop*

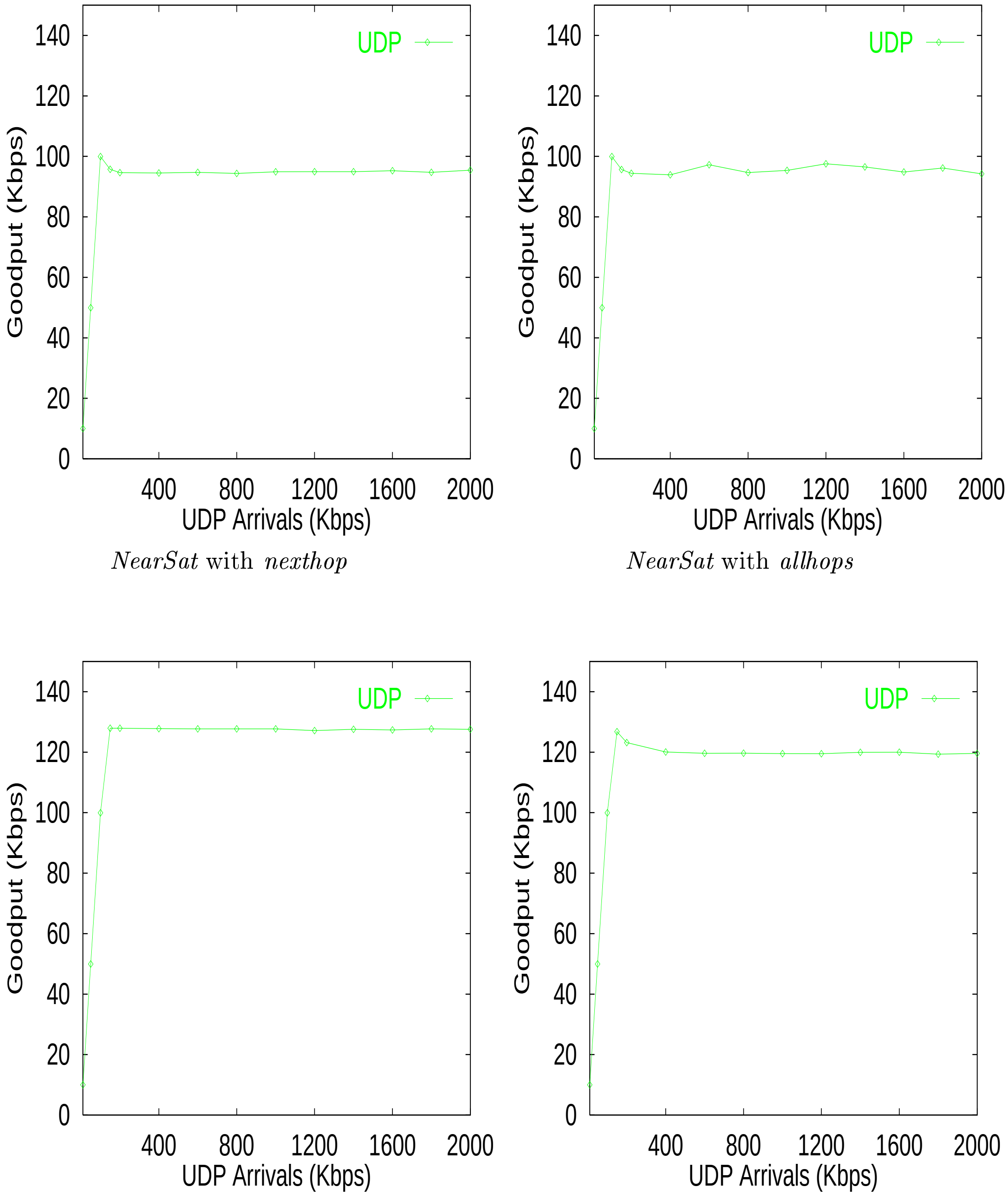
Drop-rates with ERUF using *allhops*

Goodput without ERUF

Goodput with ERUF using *nexthop*

Goodput with ERUF using *allhops*

Figure 4.20: Goodput and packet drop-rates for the *FurSat* topology.

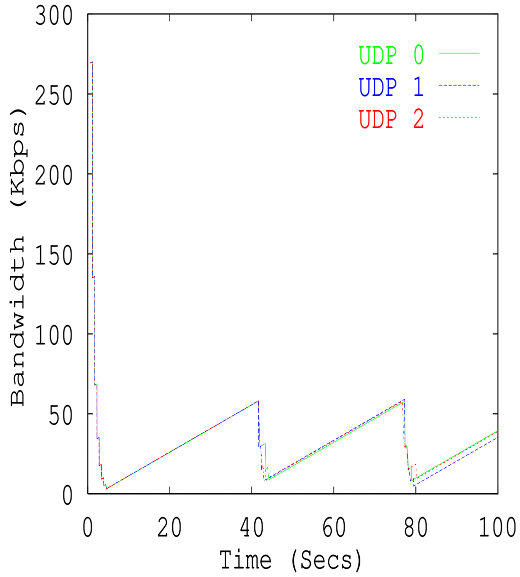


NearSat with *nexthop* | *NearSat* with *allhops*

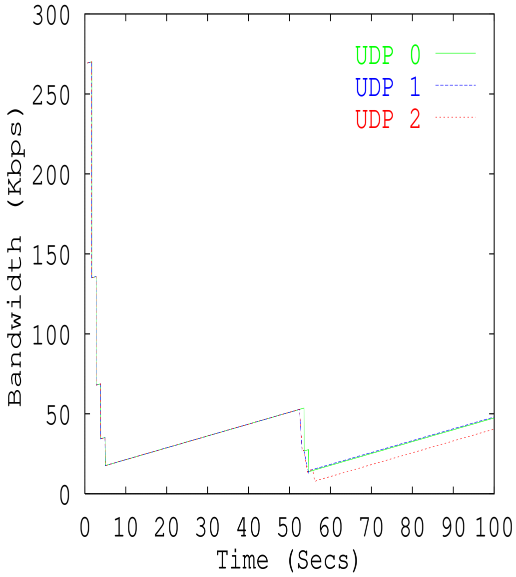
FarSat with *nexthop* | *FarSat* with *allhops*

Figure 4.21: UDP goodput for the *NearSat* and *FarSat* scenarios

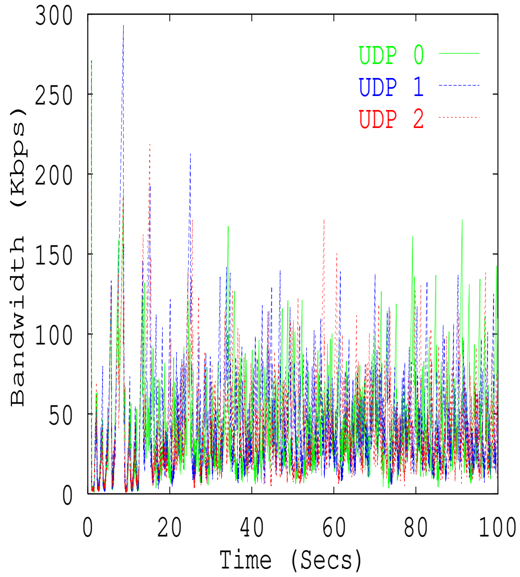
large extent, the approximations result in a moderate performance compared to the former case. Nevertheless, the performance is much better when compared to without ERUF. More research is required to estimate RTT in a better way in all kinds of topologies.



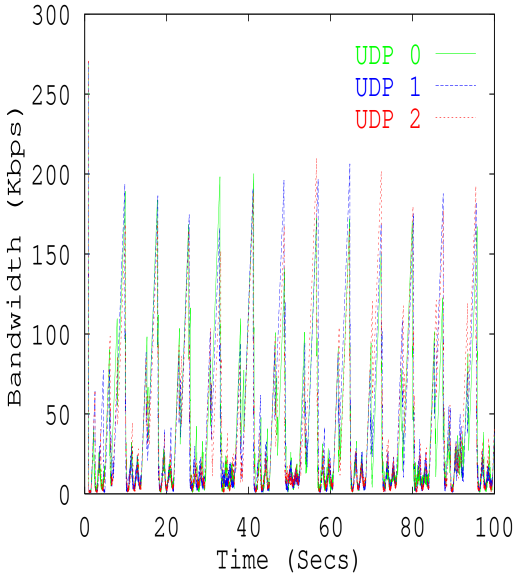
NearSat with *nexthop*



NearSat with *allhops*



FarSat with *nexthop*



FarSat with *allhops*

Figure 4.22: Regulator bandwidth variation for the *NearSat* and *FarSat* scenarios

4.3 Impact of Source Quenches on Congested Reverse Paths

To examine the impact of source quenches on congested reverse paths, we used a variant of the *dumbbell* scenario with additional traffic in the reverse direction (see Figure 4.23). The additional traffic is a single CBR flow from H5 to H6 whose sending rate is varied from 200 Kb/s to about 3 Mb/s. The other traffic consists of three CBR flows with an aggregate sending rate of 1 Mb/s from H2 to H4 and one TCP flow from H1 to H3. We conducted two experiments - one with ERUF regulation enabled in edge router R1 alone referred as forward-ERUF, and the other with ERUF

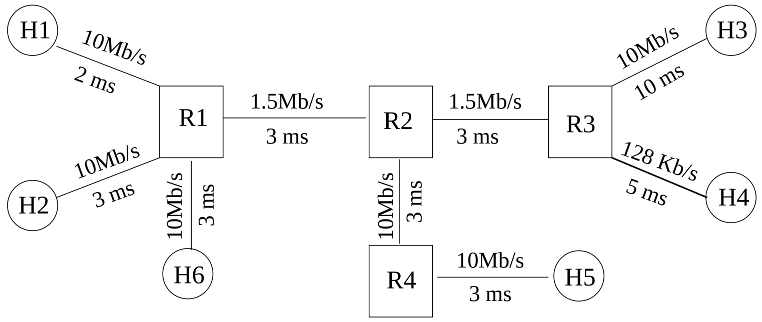


Figure 4.23: Simulation network to test the impact of source quenches on congested reverse paths. The traffic consists of one TCP flow from H1 to H3, three CBR flows with a total sending rate of 1 Mb/s and one CBR flow with different sending rates from H5 to H6.

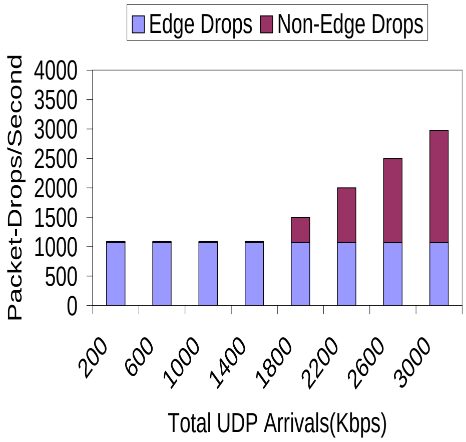
regulation enabled in all edge routers (routers R1 and R4) referred as all-ERUF. In forward-ERUF, there was regulation only for the forward traffic (H1/H2 to H3/H4) and *no* regulation for the reverse traffic (H5 to H6). The goal of the first experiment was to evaluate the performance of ERUF when source quench messages are delayed/dropped. The goal of the second experiment was to determine if regulation of the traffic in the reverse direction would improve overall performance.

Figure 4.24 shows the goodput and drop-rate graphs for this experiment. The x-axis of the goodput graph shows the UDP-reverse (H5 to H6) flow's arrival rate into the router R4. With forward-ERUF, we see that as long as the UDP-reverse flow's sending rate is less than or little over the R2-R1 link bandwidth (1.5 Mbps), the TCP flow gets near ideal goodput. This is because, the source quench messages that compete with UDP-reverse for the R2-R1 link bandwidth reach router R1 without any difficulty and the UDP-forward traffic is well regulated. As soon as the UDP-reverse flow's sending rate goes well over 1.5 Mbps, the source quench messages are delayed/dropped. Lack of feedback makes the regulation mechanism less effective. Consequently, the TCP goodput falls rapidly when the UDP-reverse flow's arrival rate is over 2.2 Mbps. This is substantiated by the drop-rate graph which shows that as the UDP-reverse flow's arrival rate increases beyond 1.5 Mbps, the number of non-edge drops also increase.

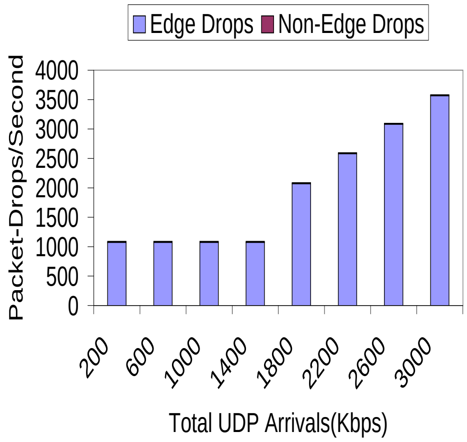
With all-ERUF, we see that the goodput graph is essentially the same as with forward-ERUF until the UDP-reverse flow's arrival rate reaches 1.5 Mbps. Once the arrival rate increase beyond 1.5 Mbps, UDP-reverse flow's packets are dropped at router R2 (R2-R1 link). This generates source quenches and router R4 starts regulating the UDP-reverse flow. Consequently, UDP-reverse's sending rate is limited at router R4 which reduces the congestion at the R2-R1 link. This makes it easier for the source quenches (for UDP-forward) from router R3 to reach R1. This in turn, leads to better regulation of UDP-forward and better performance of TCP. Note that UDP-reverse's goodput is less compared to forward-ERUF at UDP arrivals of 1.6 Mbps and higher. UDP-reverse is penalized for being unresponsive. But the aggregate goodput of all the flows into the system is definitely better than with forward-ERUF. The drop-rate graph shows that there are very few non-edge drops. This is mainly because, the non-edge drops with forward-ERUF (at R2-R1 and R3-H4) are converted to edge drops by enabling ERUF in all the routers.

4.4 Handling Large Number of Small Flows

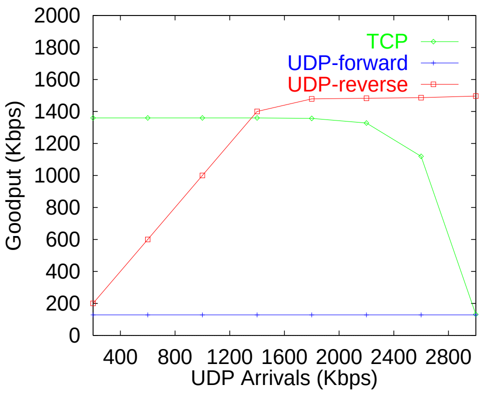
We conducted the following experiment to evaluate the efficacy of ERUF for the situation where there are a large number of unresponsive flows each with a low individual sending rate, but a



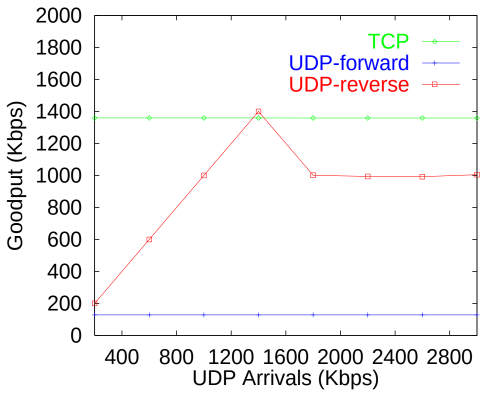
Packet drop-rates with forward-ERUF



Packet drop-rates with all-ERUF



Goodput with forward-ERUF



Goodput with all-ERUF

Figure 4.24: The impact of source quenches for networks that are congested in both directions.

high aggregate sending rate. We also evaluated the impact of ERUF with flow aggregation in this experiment.

For this scenario, we varied the number of unresponsive flows keeping their aggregate sending rate a constant (see Figure 4.25). The traffic consists of one TCP flow from H1 to H4 and 1-45 CBR flows from H2 to H3 (with an aggregate sending rate of 1.5 Mb/s). We conducted two experiments: one with fine-grain flow identification (the source address and source port were used to identify a flow) and the other with a flow aggregation scheme which aggregates all flows to the same destination. This experiment was designed to analyze the performance of ERUF when a large number of flows each with a small sending rate, but with a large aggregate sending rate cause congestion. Figure 4.26 shows the drop-rate and goodput graphs for this experiment without ERUF, for all-DRR, with ERUF and with ERUF+Aggregation.

Without ERUF, we find that the unresponsive flows use almost all of the bandwidth in the R2-R3 link no matter how many such flows there are. This behavior is very similar to the unfairness case explained in Section 2.1. Since the aggregate arrival rate of the unresponsive flows at the router R2 is the same irrespective of the number of flows, and since there is no flow identification/discrimination at R2, we see the same number of non-edge drops irrespective of the number

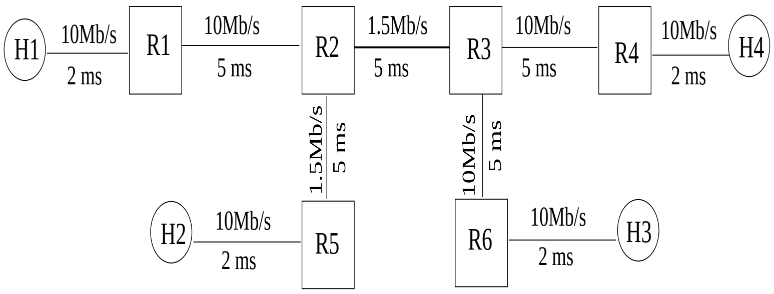
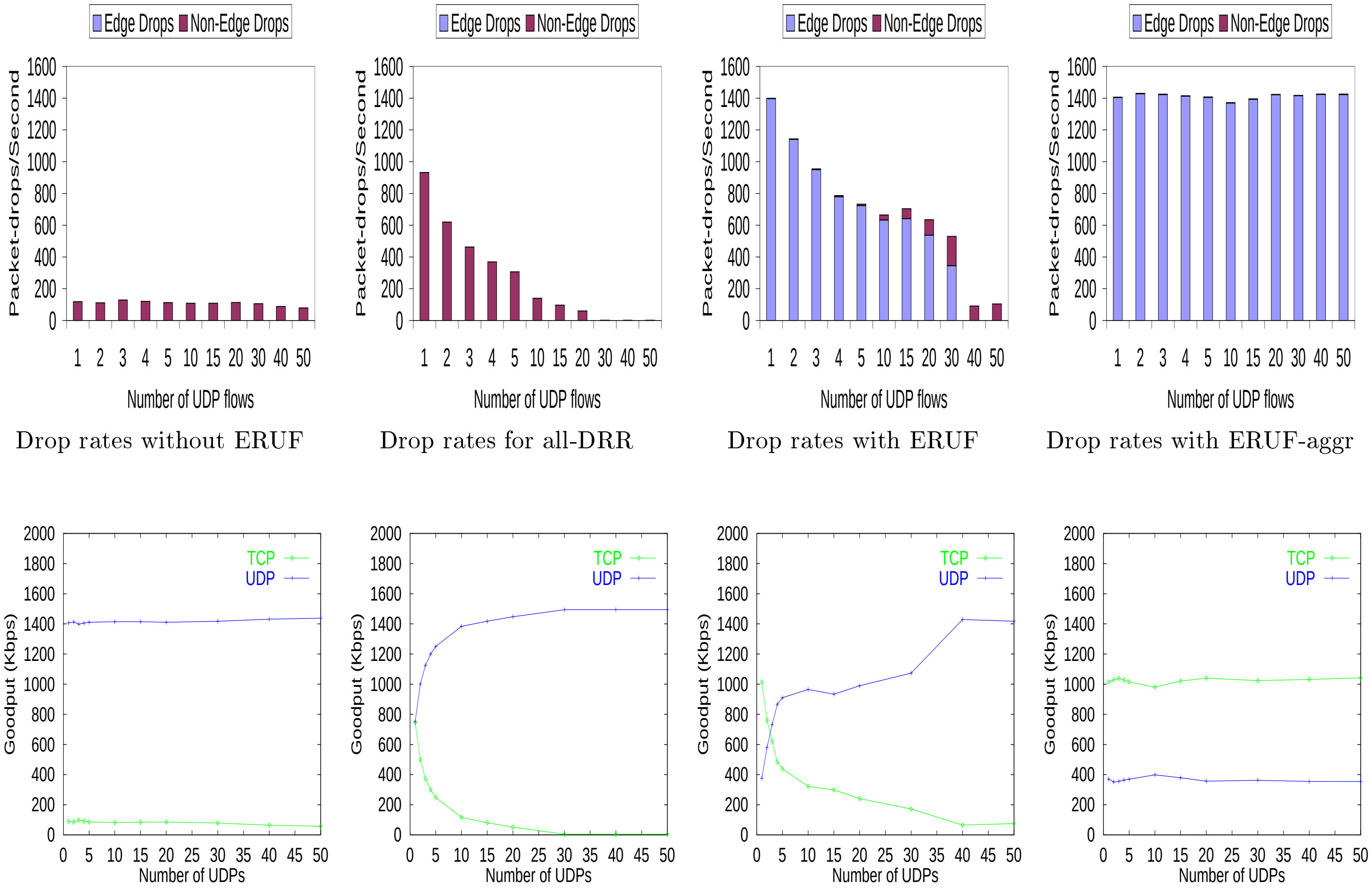


Figure 4.25: Simulation network to test the effect of ERUF when there are large number of small flows. The traffic consists of one TCP flow from H1 to H4, and 1-50 CBR flows from H2 to H3. The total sending rate of the CBR flows is 1.5 Mb/s for all configurations.

of unresponsive flows.

With all-DRR, we see that TCP gets a goodput of about $1/n$ of the R2-R3 bandwidth and the aggregate CBR goodput is about $(n-1)/n$ of the R2-R3 bandwidth, where n is the number of CBR flows. This is expected because DRR is a fair queuing algorithm and each of the n competing flows gets about $1/n$ of the share. It is interesting to note that the number of non-edge drops decrease as the number of CBR flows increase. This is because, as the number of flows increases, the TCP flow gets lesser share of the bandwidth. Therefore, the TCP flow's sending rate decreases as the number of flows increases. Since the aggregate arrival rate of the CBR flows is constant (1.5 Mbps), the aggregate arrival rate of all flows (UDP and TCP) decreases as the number of flows increases. Therefore, the number of drops at the R2-R3 link (non-edge drops) decreases.

With ERUF, we see that with smaller number of flows, ERUF is very effective in dropping packets early. As the number of unresponsive flows increases, the effectiveness of ERUF in protecting responsive flows decreases. As the number of CBR flows increases (and the sending rate of individual flows drops), the edge drops are distributed roughly evenly among all the unresponsive flows. As a result, one source quench has a smaller impact (as it results in halving a smaller flow). By the time a flow x gets quenched, another flow y which was quenched some time earlier would have gained sufficient regulator bandwidth. Figure 4.27 shows the regulator bandwidth variation for three UDP flows, for one UDP flow and for three UDP flows with aggregation. We find that with three UDP flows without aggregation, the bandwidth oscillates around 200 Kbps. With one flow, the oscillation is around 300 Kbps. The regulator bandwidth variation does not decrease proportionally to the number of UDP flows. Therefore the aggregate sending rate of the CBR flows at router R1 increases as the number of CBR flows increases. As a result, the total number of edge drops reduces, compared to the cases with fewer flows. Since the effectiveness of UDP regulation decreases with increase in number of UDP flows, the goodput of the TCP flow decreases. But we should note that with ERUF, TCP always achieves a better goodput compared to all-DRR. ERUF does not equally distribute the available bandwidth between competing flows. It rate-limits unresponsive flows and the responsive flows benefit from that. Figure 4.26 also shows the result of the same experiment with the use of aggregation. We see that when all the CBR flows between H2 and H3 are aggregated at router R1, ERUF achieves the same performance for this aggregate as it does for a single large flow. It can also be seen from the Figure 4.27 that with aggregation of the three UDP flows, the regulator bandwidth variation is close to the one with a single UDP flow. This result argues for flow-aggregation at edge routers. Several IETF working groups recommend a similar aggregate-at-edge-routers approach [3, 5].



Drop rates without ERUF

Drop rates for all-DRR

Drop rates with ERUF

Drop rates with ERUF-aggr

Goodput without ERUF

Goodput for all-DRR

Goodput with ERUF

Goodput with ERUF-aggr

Figure 4.26: Goodput and packet drop-rate for the large number of small flows scenario

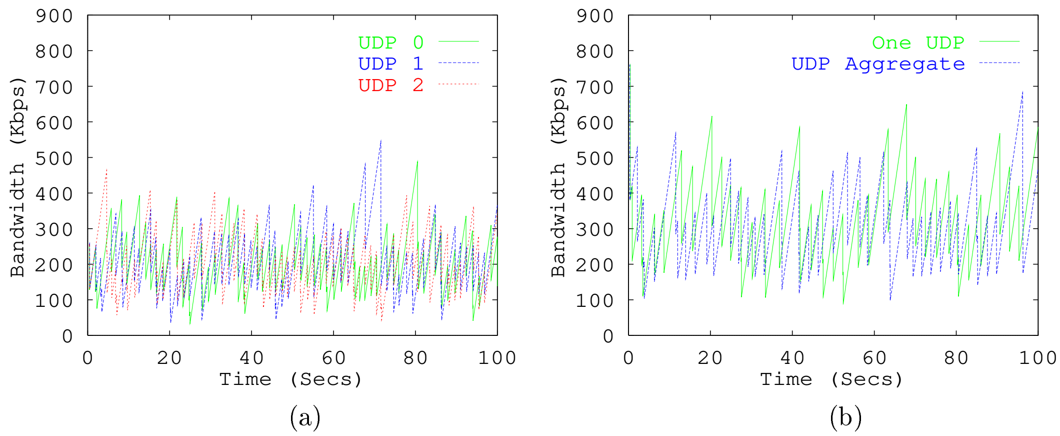


Figure 4.27: Regulator bandwidth variation for the scenario with large number of small flows: (a) Three UDP flows, no aggregation. (b) One UDP flow, no aggregation Vs Three UDP flows with aggregation

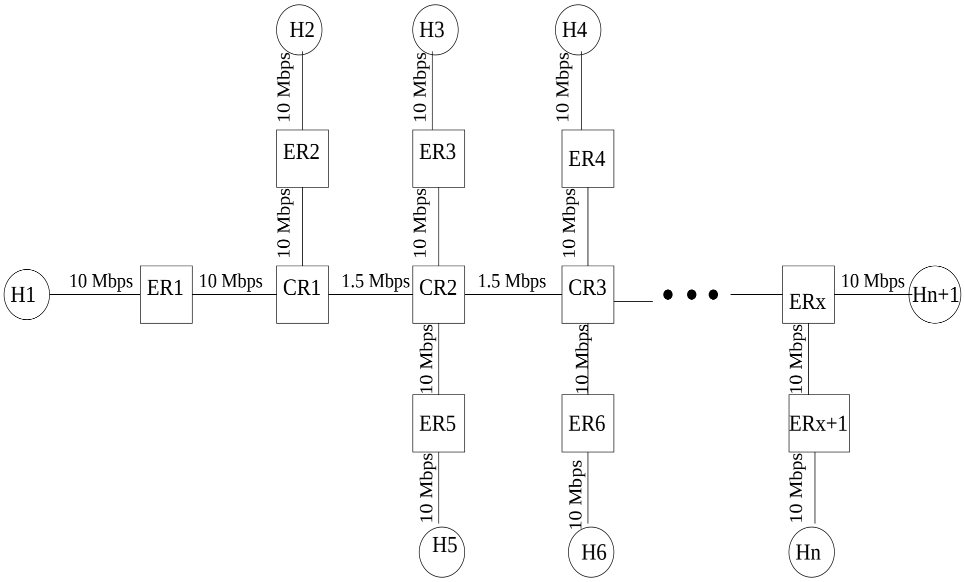
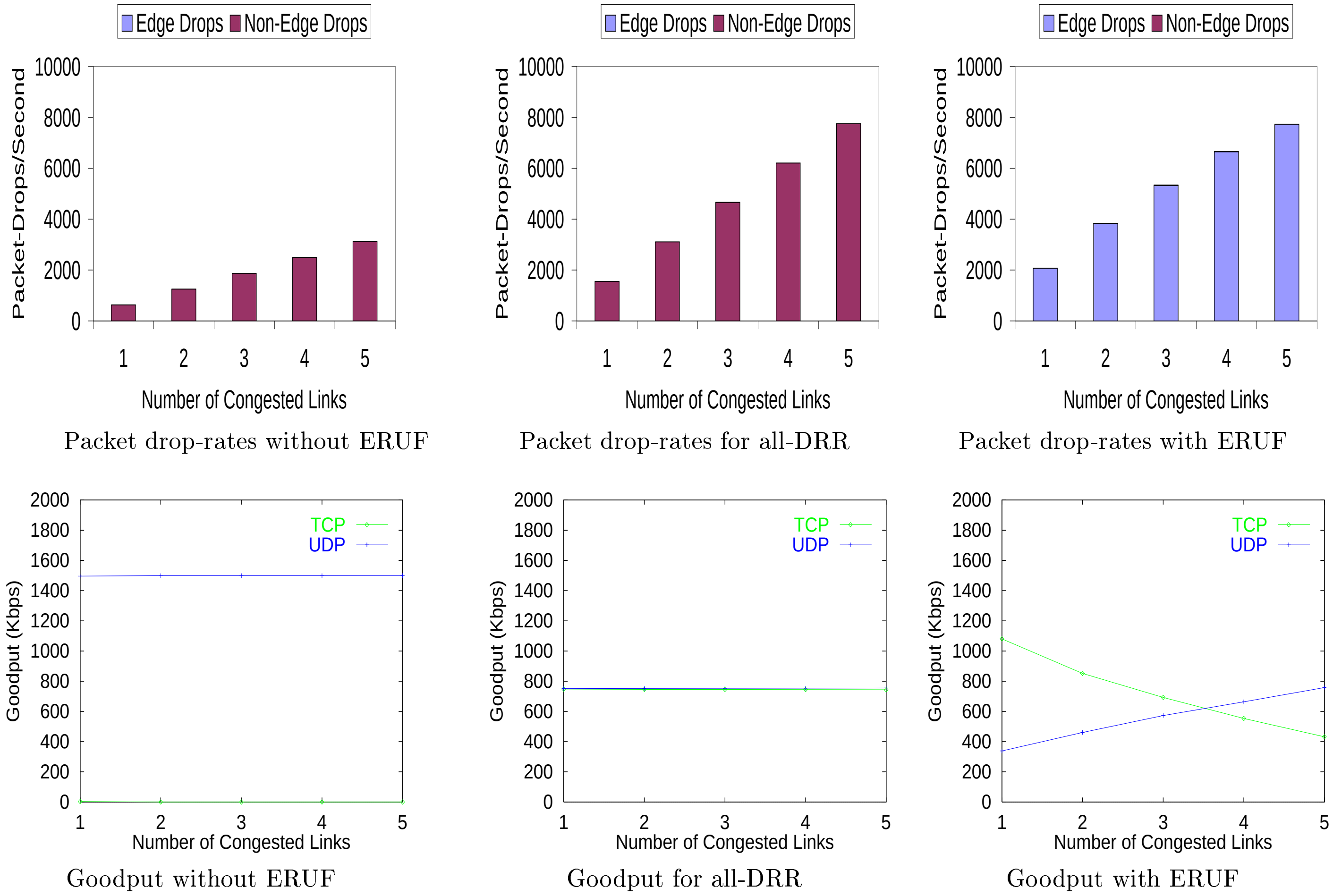


Figure 4.28: Simulation network for the *multiple congested links* scenario. The traffic consists of one TCP flow from H1 to Hn+4, and one CBR from Hi to Hi+3, $2 \leq i \leq n+1$, where n is the number of congested links. The sending rate of each CBR flow is 2 Mb/s.

4.5 Impact of Multiple Congested Links

In all of the previous experiments, the path of the TCP flow had only one congested link. We performed this experiment to analyze the performance of ERUF in the presence of multiple congested links in the TCP flow's path. Figure 4.28 shows the topology we used for this scenario. The traffic consists of one TCP flow from H1 to H$n+1$ and one UDP flow from Hi to Hi+3, $2 \leq i \leq n+1$, where n is the number of congested links. The sending rate of each UDP flow is 2 Mbps. We varied n from 1 to 5.

Figure 4.29 shows the packet drop-rate and goodput graphs for this scenario without ERUF, with all-DRR and with ERUF. The goodput graphs show the average goodput for the UDP flows and the goodput of the TCP flow for different number of congested links. Without ERUF, the



Packet drop-rates without ERUF
Packet drop-rates for all-DRR
Packet drop-rates with ERUF
Goodput without ERUF
Goodput for all-DRR
Goodput with ERUF

Figure 4.29: Goodput and packet droprate for the *multiple congested links* scenario

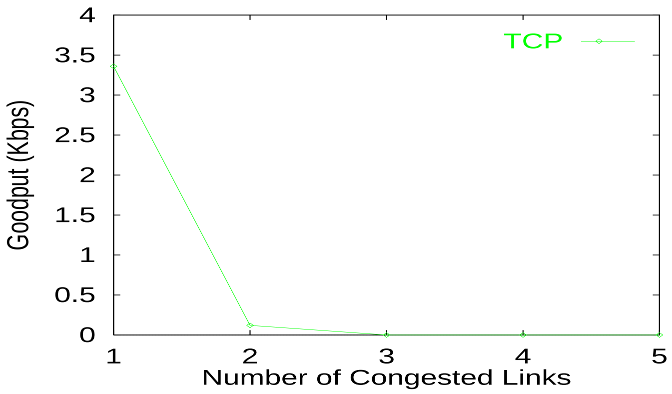


Figure 4.30: Goodput of TCP flow without ERUF in the *multiple congested links* scenario

TCP flow's goodput is almost always close to zero. Figure 4.30 shows the TCP goodput in a magnified scale. Since the links in the TCP flow's path are perennially congested, there is always a probability that the TCP flow's packets will be dropped. TCP, being a responsive flow, reduces its sending rate in response to packet drops. The probability of packet drop for the TCP flow increases with increase in the number of congested links. Since the TCP flow's packets almost always get dropped when trying to pass through multiple perennially congested links, the TCP source is shut off. The goodput falls to zero with increase in the number of congested links. The drop-rate graph shows no edge-drops since the links immediately downstream from the edge routers are sufficiently provisioned. Also, we can see that the number of non-edge drops increase linearly with the number of congested links.

With all-DRR, the congested links are shared equally among the competing TCP and UDP flows. Each congested link (1.5 Mbps) is shared by the TCP flow and one UDP flow. Therefore we see that average UDP goodput and the TCP goodput are almost always equal. There are more non-edge drops in this case compared to without ERUF because, the total arrival rate at the congested routers is more (since TCP's sending rate is more).

With ERUF, we see that there are very few non-edge drops. The edge routers use the source quenches generated by the non-edge routers as feedback and regulate the flows. As a result, the edge-drops increase and the non-edge drops decrease. The goodput graph shows that for a small number of congested links (< 4), the TCP flow gets more than the UDP and for more links, UDP flows get more. This can be explained as follows. The TCP flow's goodput depends, among other factors, on packet drops and round-trip time (RTT). With fewer congested links, the probability of packet drops is smaller and so is RTT. Therefore, the TCP flow achieves a higher goodput. For more congested links, packet drops are more frequent, and the RTT is large. Therefore, the TCP source's sending rate reduces as the number of congested links grows. Consequently, the UDP flows get more bandwidth. Note that for 5 congested links, the average UDP goodput is the same as in the all-DRR case, but the TCP goodput is far less. This is because, with ERUF, the congested routers use RED and when congestion occurs, *some* of the TCP packets are also dropped. But for all-DRR, when congestion occurs (primarily because of the UDP flow), packets are dropped from its the UDP flow's queue and the TCP flow is unaffected.

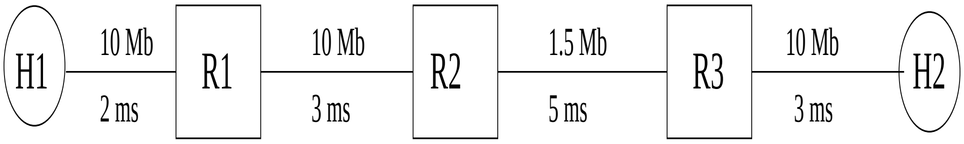


Figure 4.31: Simulation network to compare *cwnd* and regulator bandwidth variations

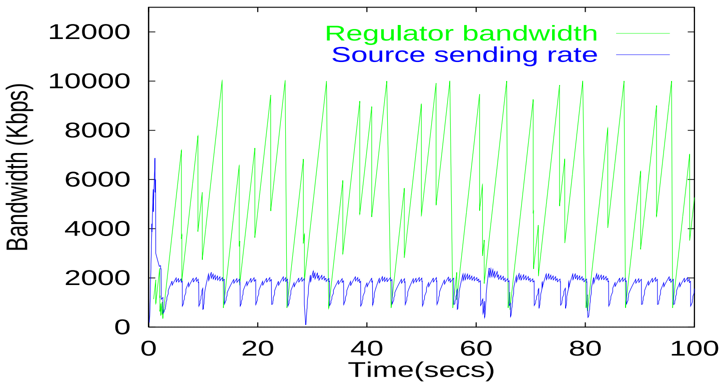
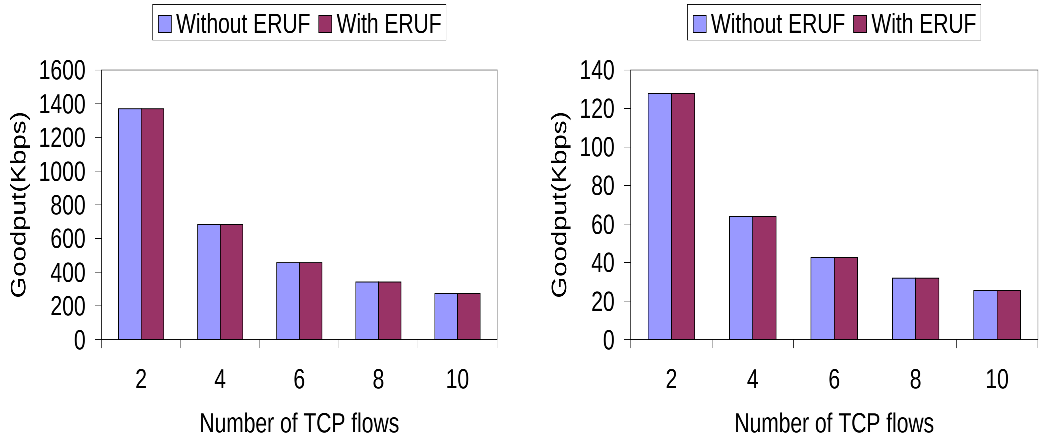


Figure 4.32: Regulator bandwidth and congestion window variation

4.6 Impact of the ERUF Regulators on TCP Flows

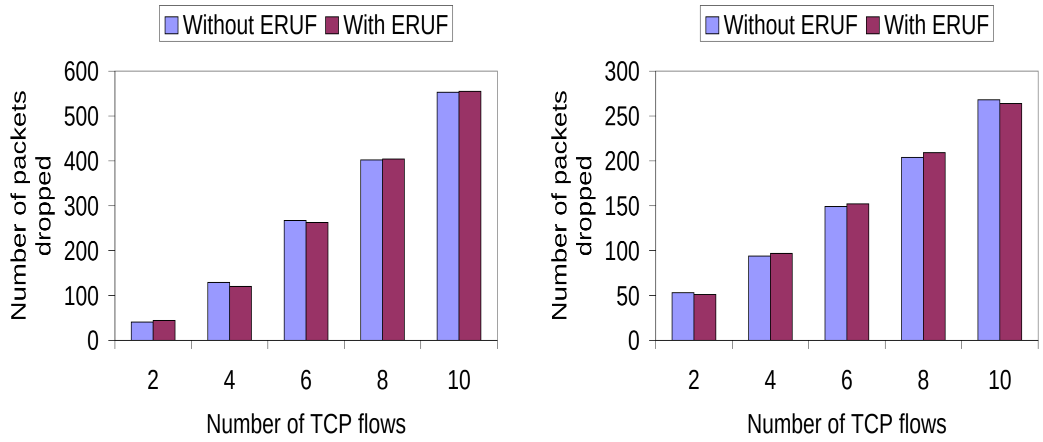
The regulator bandwidth at the edge routers using ERUF is varied according to a multiplicative decrease/additive increase discipline that is, the regulator bandwidth is decreased by half and increased by the average packet size of the flow (see Sections 3.3, 3.4 for detailed descriptions). This is similar to the standard TCP's congestion window (*cwnd*) variation. The extent to which the regulator's bandwidth accurately matches that of a standard TCP source's sending rate determines how well ERUF can detect and regulate unresponsive and under-responsive flows. It would also be good to ensure that responsive flows are not affected by regulation.

Intuitively, since ERUF's RTT estimate (using the *nexthop* approximation) is a lower bound on the end-to-end RTT used by TCP, over any give time interval t, the regulator bandwidth should ramp up faster than the standard TCP source's sending rate. To verify this, we used the topology shown in Figure 4.31 with traffic consisting of just one TCP (SACK) from H1 to H2. In Figure 4.32, H1's sending rate (determined by the congestion window) and the regulator bandwidth are plotted against time in the x-axis. H1's sending rate at any time t is calculated as $cwnd \times packetsize/srtt$ where *cwnd* is the congestion window, *packetsize* is the average packet size of the TCP flow and *srtt* is the smoothed RTT at time t. As expected, we see that the regulator bandwidth ramps up faster than the source's sending rate. Notice that both drop rapidly in response to packet drops. But the regulator bandwidth drops from a higher bandwidth than the source's sending rate. Therefore at the point where both start ramping up again, the regulator bandwidth is more than the source's sending rate. Another point to be noted is that, the source might go to slow-start (*cwnd* reset to one segment) when there are multiple packet drops from the same window whereas the regulator bandwidth will not. All of the factors mentioned above contribute to the regulator bandwidth always being higher than the standard TCP source's sending rate. This fact can be exploited by under-responsive flows whose sending rate is more than a responsive flow, but less than the regulator bandwidth to gain an unfair share of the bandwidth. The conclusion is that,



Goodput for TCP H1-H3 Goodput for TCP H2-H4

Figure 4.33: Goodput for TCP SACK in the *dumbbell* topology



Edge drops Non-edge drops

Figure 4.34: Packet-drops for TCP SACK in the *dumbbell* topology

while ERUF is successful in regulating flows that are unresponsive, it may not be as effective against flows that are under-responsive. For better regulation of under-responsive flows, we need to able to vary the regulator bandwidth as close as possible to that of a standard TCP flow. For this, we need to estimate RTT in a better way.

To quantitatively verify the effects of regulation on responsive flows, we performed two sets of experiments with competing responsive flows (TCP SACK) - one using the *dumbbell* topology and the other using the *waist* topology. We repeated the two sets with other versions of TCP - Reno, Tahoe and Vegas. In all the cases, the total number of TCP flows were equally distributed among the source-destination pairs H1-H3(F_1^3) and H2-H4(F_2^4). We measured the average TCP goodput and the total number of edge/non-edge drops. Figure 4.33 and 4.34 show the average goodput and packet drops respectively for the TCP flows F_1^3 and F_2^4 with and without ERUF for

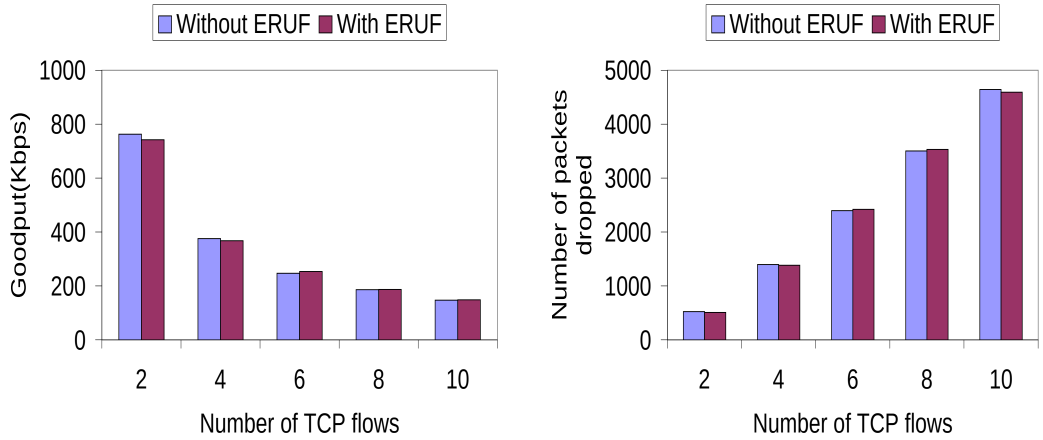


Figure 4.35: Goodput and packet-drops for TCP SACK in the *waist* topology

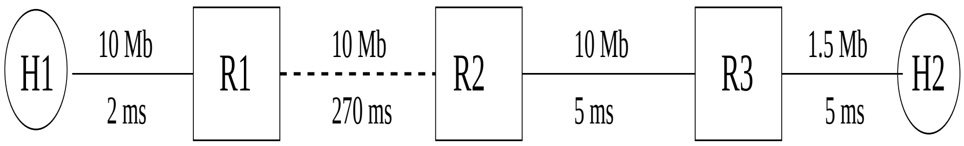


Figure 4.36: Simulation network for the case when ERUF over-estimates RTT by a lage extent.

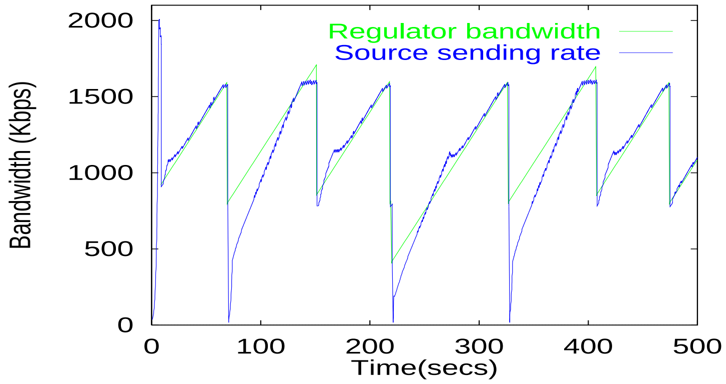


Figure 4.37: Regulator bandwidth and congestion window variation for the extreme case using *allhops* approximation. ERUF over-estimates RTT by a large extent.

the *dumbbell* topology. Figure 4.35 shows the average TCP goodput and non-edge drops with and without ERUF for the *waist* topology. Edge-drops are not shown for the *waist* topology because there were none. In all the cases, we see very little or no difference in performance of the TCP flows with and without ERUF. Results with other versions of TCP were similar. They are presented in Appendix A. The results establish the fact that the regulation does not usually have a significant effect on the performance of responsive flows.

To study the impact of ERUF-regulation on standard TCP flows in the extreme case when ERUF over-estimates the RTT by a large extent, we used the topology shown in Figure 4.36 and the *allhops* approximation to estimate RTT. The traffic consists of one TCP (SACK) from H1 to

H2. Figure 4.37 shows the regulator bandwidth and the TCP source's sending rate variation with time. Since the ERUF's estimate of RTT is more than the actual RTT, the regulator bandwidth ramps up slower than the source's sending rate (see for time $\approx$ 100, 250, 350 secs). Note that the x-axis varies from 0 to 500 seconds in this case, whereas in Figure 4.31 it varies from 0 to 100 seconds. Due to over-regulation, the TCP flows' packets spend a longer time at the edge router which causes the TCP sender's estimate of the end-to-end RTT to increase.

The conclusion is that the *nexthop* approximation works well in regulating unresponsive flows and *not* regulating responsive flows in commonly occuring scenarios. Also, since it is an under-estimate, it ramps up much faster than TCP. This could be exploited by some under-responsive flows to their benefit. The extreme case presented using *allhops* approximation is a rare occurence and is presented only for comparison.

Chapter 5

Conclusions and Open Questions

In this thesis, we have presented router mechanisms to regulate unresponsive best-effort traffic. The goal of the proposed mechanisms is to drop undeliverable packets as close to the periphery of the network as possible.

Simulation results for a variety of network topologies and traffic patterns show that the proposed mechanisms are able to move most packet drops to the periphery of the network. The resources freed by dropping these packets early allows competing responsive flows to achieve significant performance improvement. ERUF works best if a small number of source quenches can significantly reduce the amount of data being sent towards the congested router. To handle a large number of small bandwidth flows, we propose that edge routers aggregate flows.

In this thesis, we have focused on regulation of unresponsive flows. Many TCP implementors are developing "faster" TCP implementations that eliminate one or more of the congestion control mechanisms (e.g., [8, 29]). As these implementations proliferate, they too can cause congestion collapse. Since ERUF underestimates the round-trip time between the edge router and the congested router, it increases the regulator bandwidth faster than a conformant TCP would increase its congestion window. This leaves a window for non-conformant TCP implementations to exploit. We plan to extend ERUF to reduce this window. One of the alternatives we are considering is a hybrid of ERUF and the technique proposed by Floyd&Fall [11]. The idea is to estimate the maximum bandwidth for a TCP-conformant flow at the congested router (as proposed in [11]) and to piggyback this information on the source quench. This could allow the edge router to impose tighter regulators on errant flows. We also plan to study ERUF's performance when the traffic consists of non-TCP flows that are TCP-friendly.

More work needs to be done towards estimating RTT between the edge router and the congested router. Both the approximation techniques for estimating RTT that we suggested worked well in topologies involving links with small delays. However, when the topologies involve links with long delays like satellite links, the RTT estimation techniques cause erroneous regulation.

One possibility to estimate RTT in a better way is to use link state routing protocols like OSPF to gain knowledge about the whole topology. The edge routers can learn the the propagation delay of all the links in their Autonomous System (AS) using OSPF. This would help in better approximation for RTT since the edge router would know the exact path to the congested router and the propagation delay in each of the links in the path. The whole path can be known only if the congested router is in the same AS as the edge router. But even if the congested router is in a different AS, this mechanism would be a better estimate than *allhops* and *nexthop*.

Another possibility is that the edge router can periodically send ping packets to well known routers to estimate RTT. This scheme should be able to find the real RTT between the congested router and the edge router. Of course, it depends on whether the ping packets are reliable, congestion level in the network etc. Care should be taken so that we do not overwhelm a congested router with ping packets. It should also be noted that it is impossible to find the real RTT between the edge router and the core router without sending additional packets into the network.

This thesis focussed on evaluating ERUF's performance on traffic consisting of unicast flows only. We would like to extend ERUF or an ERUF-like scheme to handle multicast traffic. One obvious problem in multicast situations is that of source quench implosion. This is similar to the ack or nack implosion problems faced by reliable multicast protocols [24] and similar solutions (ack/nack aggregation) could be useful in this case too.

Bibliography

[1] F. Baker. Requirements for IP version 4 routers. RFC 1812, Network Working Group, Jun 1995.

[2] J. Bennett and H. Zhang. WF^{2}Q: Worst-case fair weighted fair queuing. In *Proc. of INFOCOM'96*, pages 120–8, 1996.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC 2475, Network Working Group, Dec 1998.

[4] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. RFC 2309, Network Working Group, April 1998.

[5] R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan. A framework for multiprotocol label switching. Network Working Group Internet Draft[1], Nov 1997.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. In *Proc. of SIGCOMM'89*, pages 3–12, 1989.

[7] G. Finn. A connectionless congestion control algorithm. *ACM Computer Communicatons Review*, 19(5), Oct 1989.

[8] Flash Networks' SatBooster Enhances Internet Communications up to Fivefold Over Satellite Links. Available at *http://www.flash-networks.com/press.htm*, 1997. Flash Networks Press Release.

[9] S. Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.

[10] S. Floyd and K. Fall. Router mechanisms to support end-to-end congestion control. Available at *http://www-nrg.ee.lbl.gov/floyd/papers.html*, Feb 1997.

[11] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. Submitted for publication, 1998.

[12] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug 1993.

[13] S. Golestani. A self-clocked fair queuing scheme for broad-band applications. In *Proc. of INFOCOM'94*, pages 636–46, 1994.

[14] RealNetworks Inc. RBN^{TM} (Real Broadcast Network) White Paper. Available at *http://www.real.com/-solutions/rbn/whitepaper.html*, Jan 1999.

[15] RealNetworks Inc. RealVideo Technical White Paper. Available at *http://www.real.com/devzone/-library/whitepapers/overview.html*, Jan 1999.

[16] V. Jacobson. Congestion avoidance and control. In *Proc. of SIGCOMM'88*, pages 314–29, 1988.

[1] Available at *http://www.ietf.org/internet-drafts/draft-ietf-mpls-framework-02.txt*

[17] R. Jain. A delay based approach for congestion avoidance in interconnected heterogenous networks. *ACM Computer Communication Review*, 19(5):56–71, Oct 1989.

[18] P. Karn. Email message to end2end mailing list *Re: Buffer size = BW-delay product*[2], Apr 7 1998.

[19] D. Lin and R. Morris. Dynamics of random early detection. In *Proc. of SIGCOMM'97*, 1997.

[20] J. Mahdavi and S. Floyd. TCP-friendly unicast rate-based flow control. Technical note sent to the end2end-interest mailing list[3], Jan 1997.

[21] A. Mankin, G. Hollingsworth, G. Reichlen, K. Thompson, R. Wilder, and R. Zahavi. Evaluation of internet performance - FY89. Technical Report MTR-89W00216, MITRE Corporation, Feb 1990.

[22] J. Padhye, Jim Kurose, D. Towsley, and R. Koodli. A TCP-Friendly Rate Adjustment Protocol for Continuous Media Flows over Best Effort Networks. Technical Report TR 98-04, Dept of Computer Science, University of Massachusetts, Amherst, Oct 1998. A brief version will appear in SIGMETRICS'99.

[23] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control – the single node case. In *Proc. of INFOCOM'92*, 1992.

[24] S. Paul, J. Lin, and S. Bhattacharyya. Reliable multicast transport protocol (rmtp). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, April 1997.

[25] W. Prue and J. Postel. Something a host could do with a source quench. RFC 1016, Jul 1987.

[26] K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481, Network Working Group, Jan 1999.

[27] K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems*, 8(2):158–81, 1990.

[28] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In *Proc. of IEEE INFOCOM'99*, 1999. To appear.

[29] E. Roberts. IP Acceleration Software: Torquing up TCP/IP. Available at *http://www.data.com/-hot_products/speedseeker.html*, Jan 1998. Data Communications Online.

[30] S. Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. In *Proc. of SIGCOMM'94*, pages 47–57, 1994.

[31] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *Proc. of SIGCOMM'95*, pages 47–57, 1995.

[32] R. Stevens. Tcp slow start,congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, Jan 1997.

[33] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high speed networks. In *Proc. of SIGCOMM'98*, 1998. 118-30.

[34] B. Suter, T. Lakshmanan, D. Stiliadis, and A. Choudhary. Design considerations for supporting TCP with per-flow queuing. In *Proc. of INFOCOM'98*, 1998.

[35] W. Tan and A. Zakhor. Internet video using error resilient scalable compression and cooperative transport protocol. In *Proc. of ICIP Volume 3*, pages 458–62, 1998.

[36] A. Tannenbaum. *Computer Networks*. Prentice-Hall, third edition, 1996.

[37] T. Turletti, S. Parisis, and J. Bolot. Experiments with a layered transmission scheme over the internet. Technical Report 3296, INRIA Sophia-Antipolis, Nov 1997.

[38] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proc. of INFOCOM'98*, 1998.

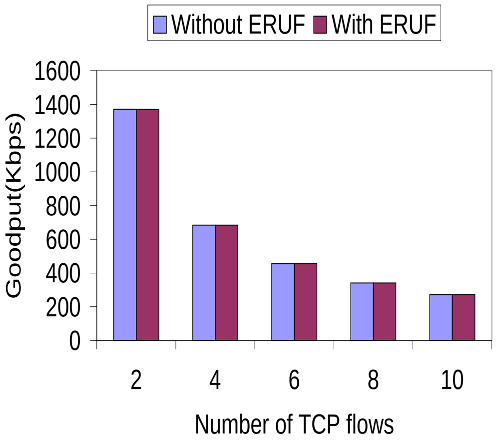
[2] Available at *ftp://ftp.isi.edu/end2end/*.

[3] Available at *http://www.psc.edu/networking/papers/tcp_friendly.html*

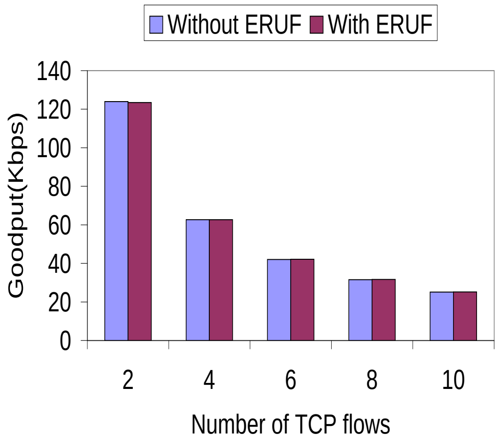
Appendix A

Impact of ERUF Regulators on Responsive flows

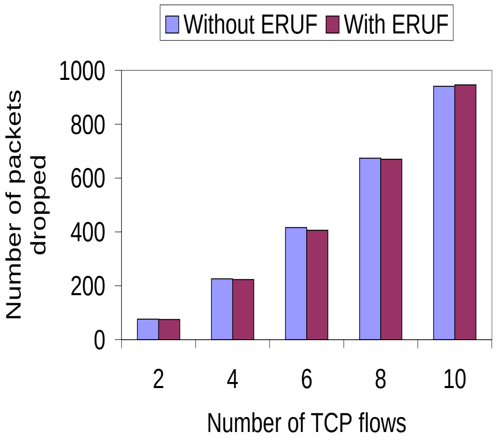
The impact of ERUF regulators on responsive flows was discussed in Section 4.6. To quantify the statement that ERUF regulators do not affect responsive flows, we performed a set of experiments. We used 2, 4, 6, 8 and 10 competing TCP-Reno flows as traffic in the *dumbbell* topology and the *waist* topology. We performed the experiments with ERUF and without ERUF and measured the average TCP goodput and number of edge-drops and non-edge drops in both cases. We then repeated the entire set of experiments using TCP-Tahoe and TCP-Vegas. The results are presented in Figures A.1, A.2, A.3, A.4. The results show that there are no differences between the with-ERUF and without-ERUF cases at least for these scenarios. In the case of *waist* topology, there were no edge drops for any of the experiments. Results for similar experiments using TCP-SACK were presented in Section 4.6.



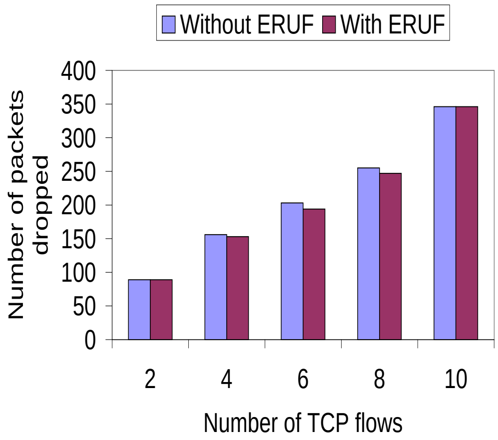
Goodput for TCP H1-H3



Goodput for TCP H2-H4

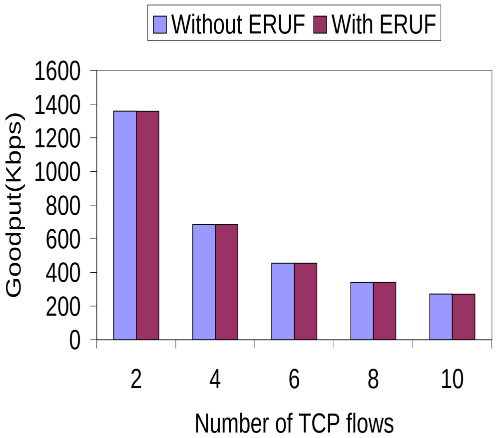


Edge drops

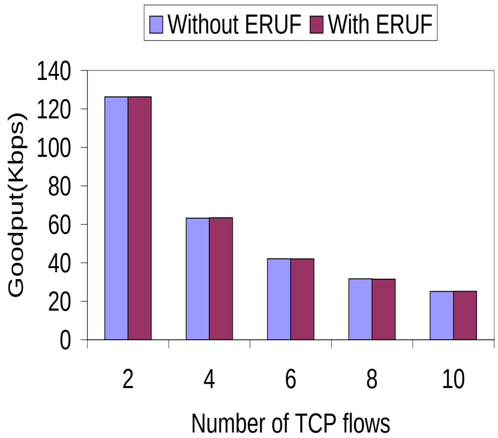


Non-edge drops

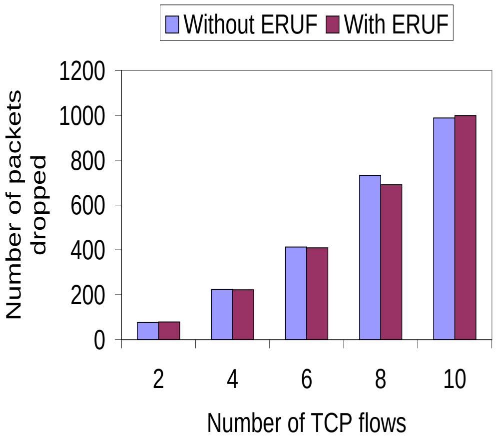
Figure A.1: Goodput and packet-drops for TCP Reno in the *dumbbell* topology



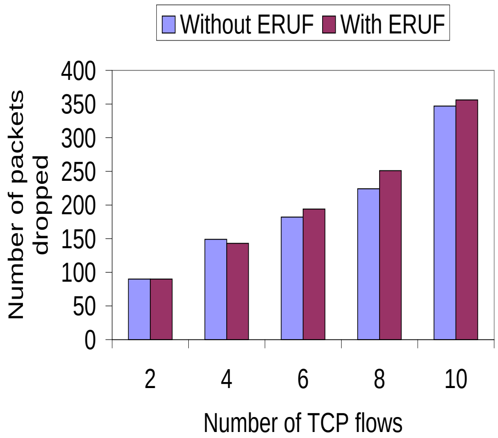
Goodput for TCP H1-H3



Goodput for TCP H2-H4

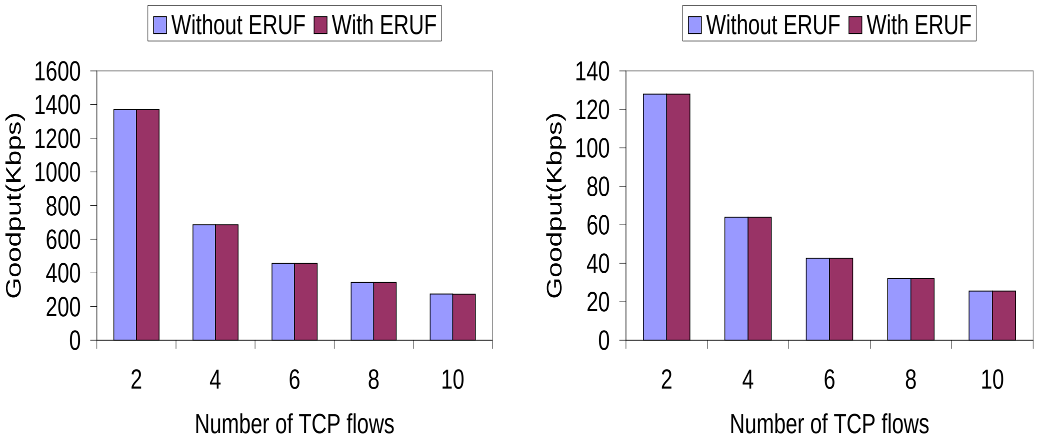


Edge drops for TCP H1-H3



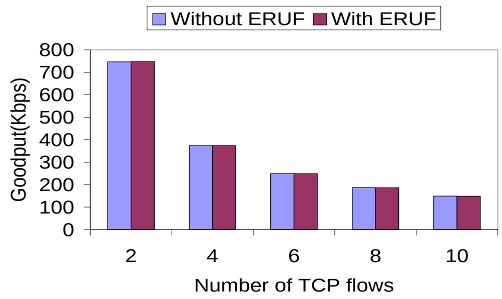
Non-edge drops for TCP H2-H4

Figure A.2: Goodput and packet-drops for TCP Tahoe in the *dumbbell* topology

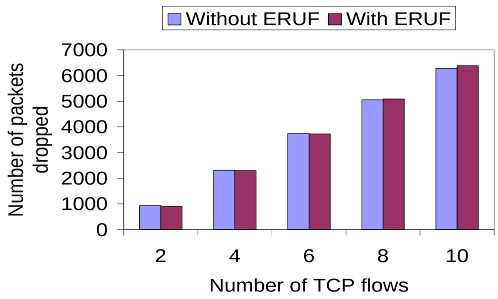


Goodput for TCP H1-H3 Goodput for TCP H2-H4

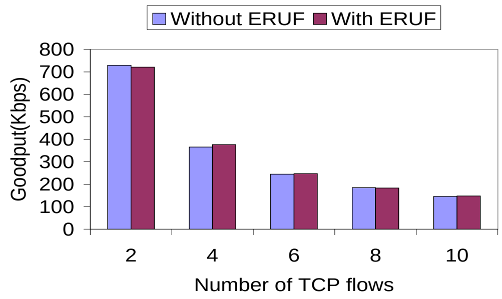
Figure A.3: Goodput graph for TCP Vegas in the *dumbbell* topology



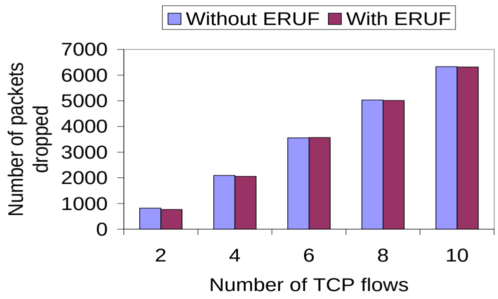
Goodput for TCP Reno



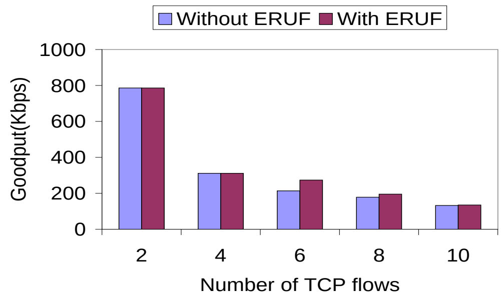
Packet drops for TCP Reno



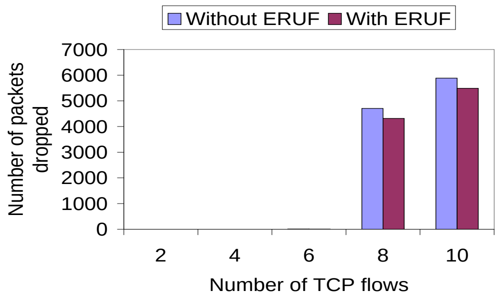
Goodput for TCP Tahoe



Packet drops for TCP Tahoe



Goodput for TCP Vegas



Packet drops for TCP Vegas

Figure A.4: Goodput and non-edge drops graph for TCP Reno, Tahoe and Vegas in the *waist* topology