CERIAS Tech Report 2001-105
Unresponsive Flow Detection and Control in Differentiated Services Networks
by A Habib, B Bhargava
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Unresponsive Flow Detection and Control Using the Differentiated Services Framework*

AHSAN HABIB, BHARAT BHARGAVA
Center for Education and Research in Information Assurance and Security (CERIAS) and
Department of Computer Sciences Purdue University, West Lafayette, IN 47907-1398, USA
E-mail: {habib, bb}@cs.purdue.edu

ABSTRACT

During periods of congestion, TCP flows back off and adjust the sending rate. This behavior makes TCP a conservative protocol and helps to avoid congestion collapse. Flows, like UDP, do not respond to congestion and keep sending packets. This causes other TCP flows sharing the same link to back off. Unresponsive flows waste resources by taking their shares in the upstream and dropping packets later when the downstream is congested. We use the Differentiated Services (DiffServ) architecture to solve this problem. With the help of core routers of DiffServ networks, we detect congestion due to unresponsive flows and using edge routers we control/shape these flows. We describe how core routers detect congestion and inform edge routers about it. We design an algorithm to regulate unresponsive flows dynamically. Our rate control algorithm works well in a variety of situations. The goal of this work is to ensure that TCP does not starve due to unresponsive flows as well as to stop bandwidth waste in the upstream path when packets are dropped in the downstream because of unresponsive flows.

KEYWORDS

Unresponsive Flow, Differentiated Services, Quality of Service, Traffic Conditioner, Shaping.

1 Introduction

A flow is unresponsive if it fails to decrease its sending rate in response to congestion. During congestion, adaptive flows like TCP back off and reduce their sending rates. This behavior of TCP prevents congestion collapse. If all flows act in this manner, there should not be any unfairness as well as congestion collapse. But flows like UDP send at the same rate even when there exists congestion along the path, because UDP does not use any feedback mechanism and can not respond to congestion. This behavior may cause TCP flows to starve and introduces unfairness when various kinds of flows coexist at the same time in the Internet.

If a packet is dropped at the downstream path, it wastes resources already taken at the upstream. This behavior causes global max-min unfairness [1]. The packets dropped at the bottleneck link have already consumed resources from non-bottleneck links earlier along the path. The unresponsive flows cause this unfairness.

Congestion collapse can be mitigated using improved packet scheduling or active queue management [3, 11]. However these techniques can not solve the global max-min unfairness problem, because congestion can be far down along the path and the upstream queues do not know about this. To solve both problems, we need a mechanism to ensure that the rate at which packets are entering a network domain should be the same as packets are leaving the domain. We use the Differentiated Services (DiffServ) architecture [2] to address this issue. The DiffServ framework uses edge routers at the border of a network domain and core routers inside the domain. Traffic conditioners at the edges shape, mark, and if necessary drop traffic. In the core of the DiffServ network, Per Hop Behaviors (PHBs) are used to achieve service differentiation. The Assured Forwarding [8] PHB uses three priorities packets per class. Dropping highest priority packets of each class exihibits that the network is congested [14]. This congestion drop is sent to ingress routers to regulate unresponsive flows. The drops due to shaping at the ingress routers is propagated to egress routers of previous domain to regulate an unresponsive flow at the upstream path. We use this framework to control unresponsive flows and refer as Unresponsive Flow Control (UFC) scheme. We present a number of experiments to show the behavior of this framework.

The paper is structured as follows. Section 2 discusses related work. Section 3 discusses what modifications are needed at the core so that it can properly inform the edges about the congestion. We design a shaping algorithm in this section to regulate the unresponsive flows. Section 4 contains all the details of our simulation setup. Section 5 presents and discusses the measurements. We conclude with a summary and a discussion of future work.

2 Related Work

Floyd et al discuss congestion collapse from undelivered packets in [6]. This situation arises when bandwidth is continuously consumed by packets at the upstream that are dropped at the downstream. Several ways to detect unresponsive flows are presented. It is suggested that routers can monitor flows to detect whether flow is responsive to congestion or not. If a flow is not responsive to congestion, it can be penalized by discarding packets to a higher rate at the router. According to the authors there are some limitations of these tests to identify non-"TCP-friendly flow". It does not help to save bandwidth at the upstream if the flow

*This research is supported by the National Science Foundation CCR-001712 and CCR-001788, CERIAS, and IBM SUR grant.

sees the congestion at the downstream because this solution does not propagate the congestion information from downdtream to upstream.

Seddigh et al [12] suggest that if TCP and UDP are put into separate queues or Assured Forwarding classes, they may coexist fairly. This discrimination between TCP and UDP traffic may punish some well-behaved UDP flows. The core router does not know the profile of a flow and can not decide to allocate bandwidth to them fairly. The problem is associated with network load, capacity, and the reaction of different transport protocols to congestion. A dynamic control mechanism can solve this problem.

Albuquerque et al [1] propose congestion avoidance mechanism named Network Border Patrol. To detect congestion, it measures entering rate of traffic to a domain and the leaving rate from the domain. It detects and restricts unresponsive traffic flows and eliminates congestion collapse. The border routers monitor all flows, measure rates, and exchange this information with all edge routers periodically and this can be expensive. Moreover, TCP is responsive so we do not need control mechanism for TCP at the edges.

Chow et al [4] propose a framework where edge routers periodically obtain information from the core by probing and adjust the conditioner using the traffic dynamics. In this scheme, core needs to maintain all the state information. A simpler scheme can be employed where core sends packets to edge routers only at the time of congestion.

Wu et al propose Direct Congestion Control Scheme (DCCS)in [14]. In this scheme, they detect congestion by observing packet drops with *lowest priority to drop* at the core router. We follow the same rule in our research to detect congestion. Our core is simpler in the sense that it detects drops of only unresponsive flows. The main difference between our work and [14] is that we design the shaper at the edge that controls the unresponsive flow.

Recent work by Mahajan et al [9] uses Aggregate-based Congestion Control (ACC) to detect and control high bandwidth aggregate flows. They use the history of packet drops over a time interval and then the ACC agent matches prefix of IP destination addresses to detect flows going to the same destination address for Denial of Service(DoS) attacks. The ACC agent controls the flows using a rate-limiter and pushes status messages reporting the aggregate's arrival rate to the upstream routers. We use DiffServ architecture to detect and propagate messages. Our goal is to detect and control unresponsive flows but it can protect DoS attack by using their idea of prefix matching [9].

3 Framework of Congestion Control

This section describes the modifications on the DiffServ components are necessary to support Unresponsive Flow Control (UFC) scheme. We need to modify core routers so that they can inform the edge routers about the congestion. The edge router has a traffic conditioner that may re-mark a traffic stream or may discard or shape packets to bring the stream into compliance with a traffic profile specified by the network administrator [2]. We have to use a proper shaping algorithm that can control unresponsive flows at the time of congestion. One additional modification is to be done at the edge; the ingress router of one domain informs the egress router of the previous domain about the congestion. Thus congestion information is propagated to the upstream.

3.1 Modification at Core Router

Core does not store any per flow reservation information. This makes the DiffServ architecture more scalable. We make little modification at the core routers to inform edge routers about the congestion.

Wu et al [14] suggest that packets dropped at the core with lowest drop precedence, say DP0, indicates that there is a congestion in the network. We use a similar idea to identify the congestion. We detect congestion only for unresponsive flows using protocol information from transport layer. At the core, there is no way to classify packets as responsive or unresponsive. This idea of monitoring all flows vs. unresponsive flows is debatable. But it is true that responsive flows will back off just after one time-out period. So, the advantage of monitoring responsive flows is small comparing to the overhead of monitoring it.

The core stores {source addr, destination addr, source port, destination port, protocol, timestamp, outgoing_link_bw} about a dropped packet. The core sends this drop information periodically to the ingress routers when total drops exceeds a local threshold. The first five are necessary to identify a flow. The outgoing link bandwidth for a flow at the core helps to regulate the flow dynamically. The edge routers can be more aggressive if the core has a thin outgoing link. The edge can store the outgoing link information based on the core id. Core sends its id to mention the outgoing link the packet is traversing through if it has multiple of those. The modification at the core does not impose a lot of overheads on it because it stores/sends drop information only about unresponsive flows and only at the time of congestion.

3.2 Modification at Edge Router

There are two types of edge routers: ingress and egress. Same router can be configured to act as both. We present the modification on each of them separately.

Egress Router: We distinguish two types of drops at the edge routers. First one is a drop due to shaping at the edge, say *sdrop* and the other one is a drop due to congestion at the core/edge router, say *cdrop*. If there is a drop due to congestion, we use more information than just packets dropped to regulate conditioner. The egress router informs both drop information to the previous ingress router separately.

Ingress Router: The modification in the ingress router is to add/modify shaping algorithm. Ingress gets shaping drop, *sdrop*, from egress node and congestion drop, *cdrop*,

from cores and egress routers, which are used for shaping. For a particular flow, suppose, the bottleneck bandwidth is bb. The bandwidth of outgoing link of the flow at the edge bo. The flow has an original profile (target rate) of op and adjusted profile of ap. The weighted average rate for this flow is $wavg$. In case of *cdrop*, the profile of the flow is updated using following equations

$$adjust = cdrop \times packet\_size \times max(1, \gamma \frac{bo}{bb}) \quad (1)$$

$$ap = max(0, min(ap - adjust, wavg - adjust)) \quad (2)$$

where $0 < \gamma < 1$, γ is aggressiveness to congestion control. Higher value of γ helps to converge the drop adjustment faster.
In equation (2), adjusted profile is taking non-negative minimum value from current profile or from current average arrival rate. The arrival rate is calculated over a time frame using Time Sliding Window [5] algorithm.

For *sdrop*, the profile is adjusted using equation (3). The ap is initialized at the beginning with op. If the router does not receive any drop information during a time interval, it increases the adjusted profile using equation (4) periodically at a certain rate r, where r is initialized to a constant number of packets each time the router gets drop information. In absence of any drop, the rate r is increased using equation (5).

$$ap = max(0, ap - sdrop \times packet\_size) \quad (3)$$

$$ap = min(op, ap + r) \quad (4)$$

$$r = min(\frac{wavg}{f}, 2 \times r) \quad (5)$$

where f is a factor, which controls how fast the rate can be increased in the absence of any drop feedback. This rate should be bounded by the current average rate. This rate adjustment algorithm follows TCPs congestion control algorithm. The profile increment is doubled each time in absence of any drop until it hit a threshold $\frac{wavg}{f}$ and then it is increased linearly.

At the edge, shaping is done based on the current average rate and the adjusted profile using the algorithm below:

```
For each incoming flow
      if avg > ap
         /*we should drop some packets */
         drop next d = min(α, avg/ap) packets
         update average rate
         /*rate is decreasing over time*/
      else
         do regular marking
where (num_of_active_flows)/2 ≤ α < num_of_active_flows
```

The algorithm ensures dropping some packets when current rate is higher than the adjusted profile to reduce congestion. The chance of dropping all d packets from a particular flow is low. Too many packets should not be dropped at a time since it may deteriorate the application level quality of the flow.

Parameters	Value
Packet Size	1024 Bytes
TCP implementation	TCP new Reno
TCP window size	64
TSW window size	1 sec
weighted average w_q	0.002
RED parameters	$\{min_{th}, max_{th}, P_{max}\}$
DP0	{40,55,0.02}
DP1	{25,40,0.05}
DP2	{10,25,0.1}

Table 1: Simulation parameters

4 Simulation Setup

We use the **ns-2** simulator [10] for our experiments. For the standard DiffServ implementation, we use software developed at Nortel Networks [13]. We use the TSW tagger meter and TSW3CM marker in the edge device. It has three drop precedences DP0, DP1 and DP2. DP0 means lower precedence to drop and DP2 means higher. The marker uses a Committed Information Rate (CIR), target rate, and a Peak Information Rate (PIR). The packets are marked with a probability based on the current rate, CIR, and PIR.

The simple topology, shown in Figure 1, has two network domains. We can test both ingress and egress routers to control congestion. The complex topology is used later to simulate a more realistic situation. The edge devices implement traffic conditioning, while the core device implements the Assured Forwarding [8] PHB using three drop precedences. We use throughput and packet drop ratio as metrics to evaluate performance. The parameters for the simulation is shown in Table 1. RED [7] parameters in the table are set to get the service differentiation among different types of packets. We use the software implementation of DiffServ network developed at Nortel Networks [13]. The RED parameters are taken from their work.

Initailly, we use two aggregate flows. Flow 1-3 is from node $n1$ to $n3$ and Flow 2-4 is from node $n2$ to $n4$. The number of flows in each aggregate is varied. Normally, we use 10 TCP micro-flows, where a micro-flow represents a single TCP connection, as Flow1-3 and 10 UDP micro-flows as Flow2-4. Then we add background traffic (both TCP and UDP type) from node $n5$ to $n6$. We change the number of flows and RTTs in different experiments. The detail is mentioned in the corresponding experiments.

The metrics used to evaluate performance include:

1. **Throughput:** This denotes the average bytes received by the receiver application over simulation time. A higher throughput usually means better service for the application (e.g., smaller completion time for an FTP

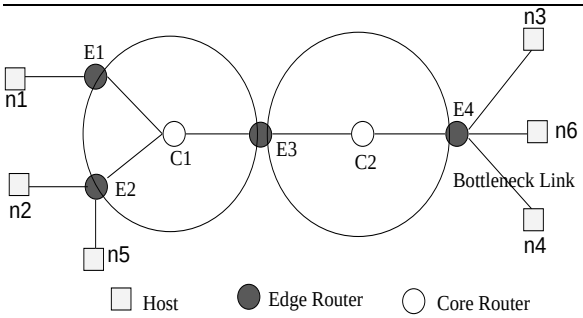


Figure 1: All links are 10 Mbps except $E4-n4$. The capacity of $E4-n4$ is varied to simulate bottleneck link for UDP traffic. Two aggregate flows between n1-n3 and n2-n4 and background traffic flows between n5-n6.

flow). For the ISP, higher throughput is preferable because this means that links are well-utilized.

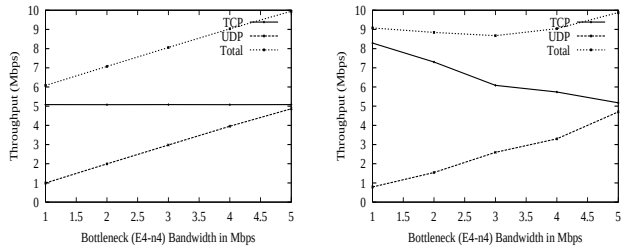
2. **Packet Drop :** We measure packets drop with the lowest precedence to drop, DP0, to show congestion and drops due to the shaping to show that shaping is done at different edges based on congestion information to improve the situation.

5 Smulation Results

We present a variety of scenarios to show that Unresponsive Flow Control (UFC) scheme works well. First, we show if there is no flow control, there is a chance for congestion collapse in the Internet. Next we show that the congestion collapse can be overcome with UFC. We impose both TCP and UDP type background traffic. We show the effect of RTT and number of flows on the flow control algorithm. Finally, we present simulation with complex topology and multiple cross traffic across the path of controlled flows.

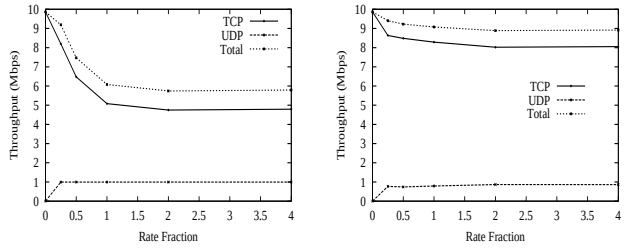
5.1 Congestion Collapse

First, we show the congestion collapse due to unresponsive flows. In Figure 1, there is an aggregate TCP flow with 10 micro-flows from host $n1$ to $n3$ and a UDP aggregate flow with 10 micro-flows from host $n2$-$n4$. Both flows have the same profile or target rate (5 Mbps). Figure 2 shows how TCP and UDP flows behave with respect to changing the bottleneck bandwidth (bb) from $1-5$ Mbps. The X-axis shows the bb and Y-axis shows the throughput achieved by both flows. Figure 2(a) shows that TCP flow gets its share of 5 Mbps all the time because it does not go through the congested link. When the bottleneck bandwidth is 1 Mbps, 4 Mbps bandwidth is wasted by UDP flows in the absence of the flow control. But if we use UFC scheme, it controls the UDP flow rate and makes the extra bandwidth available for TCP flow. Figure 2(b) shows that TCP flow get 8 Mbps when bb is 1 Mbps. UFC prevents the network from congestion collapse due to undelivered packets. It can not



(a) No Flow Control (b) With Flow Control

Figure 2: a. Without flow control and TCP gets only 5 Mbps when bottleneck bandwidth is 1 Mbps. b. With Flow control and now TCP gets 8 Mbps. Both flows have the same profile.



(a) No Flow Control (b) With Flow Control

Figure 3: UDP sending rate is varied using rate fraction, R_f. UDP sends as high as 20 Mbps (R_f=4), bottleneck ($E4-n4$) bandwidth is 1 Mbps.

achieve 100% link utilization. This can be achieved with proper tuning of parameters described in UFC algorithms.

To show how effectively flow control scheme works, we show the data in Figure 3. In this experiment, both TCP and UDP have the same profile but the sending rate of UDP is varied. We define a rate fraction, $R_f = \frac{SendingRate}{Profile}$. For example, $R_f = 0.5$ means that the flow is sending at a rate 50% of its own profile and $R_f = 4$ means the flow is sending at a rate four times of its own profile. The X-axis shows the rate fraction, R_f, of UDP and the Y-axis shows the bandwidth achieved by both flows. When UDP's sending rate is zero, TCP gets the whole 10 Mbps. If sending rate of UDP is very low and no packet is dropped, there is no shaping (shaping drop is zero) at the edge. Figure 3(a) shows that when sending rate is high enough to drop packet at the bottleneck link (bb= 1 Mbps), there is a congestion collapse in the network. TCP gets only 5 Mbps and the total is 6 Mbps. But with UFC scheme, Figure 3(b), the high sending rate of UDP does not affect the TCP flow to get extra bandwidth. The sending rate of UDP is increased as high as 4 times of its profile. The profile is 5 Mbps, i.e. UDP is sending at a rate of 20 Mbps and still there is no

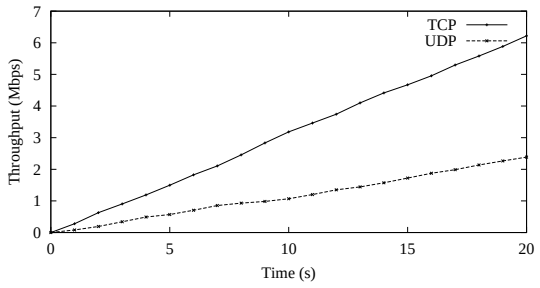


Figure 4: Cumulative receiving rate at the destination. There is no sharp drop during transmission.

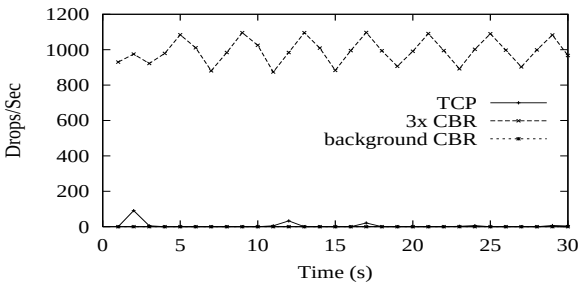
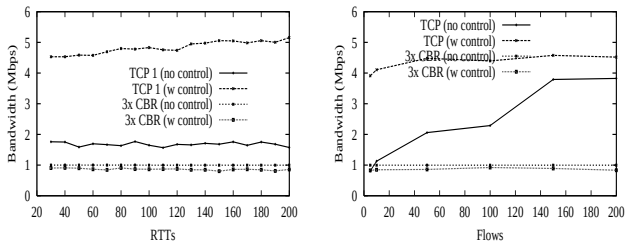


Figure 5: Drop rate of packets for different flows.



(a) Effect of RTTs (b) Changing number of Flow

Figure 6: RTTs and number of micro-flows per aggregate flow is varied for both TCP and UDP. Flow control works fine with varing RTT and with higher flow aggregation.

congestion collapse with UFC scheme.

Figure 4 shows the cumulative packets received successfully at the destination side. It shows that the curve is linear, which means the receiver gets packets at a constant rate. There is no large number of drop in the middle of the network, which may affect the application performance. The application level quality of UDP flows will deteriorate if there is a sharp drop of huge number of packets in the middle of a network.

We use a similar algorithm as TCP's congestion control to adjust the rate of an unresponsive flow. When there is a drop, the profile of a flow is adjusted temporarily and shaping is done based on the current average rate of a flow, number of active flows, and the adjusted profile. In the absence of any drop, the profile of the flow is increased periodically by adding a constant. The constant value is doubled (exponential increase) in each time interval until it hits a threshold, provided that there is no drop event (does not get any drop feedback from any core or edge routers). Then the adjusted profile is increased linearly. In Figure 5, the CBR is sending at a rate that is three times of its profile, $R_f = 3$. The packet drop rate is increased and decreased based on traffic changes (shaping information propagates with time and the rate is controlled accordingly). The TCP flow has only initial drop and then it does not see much drop. There is no drop with the background traffic because it does not see any bottleneck on its way. It takes a short period of time at the beginning to make the drop rate stable.

All of the above experiments are repeated with background traffic between host $n5$ and $n6$ of Figure 1. We use both TCP and UDP type of background traffic. We get the same outcome as mentioned above. Some results with background traffic is presented in the next sub section.

5.2 Effect of RTT and Multiple Flows

We show how stable is the flow control algorithm. We use aggregate TCP flows from $n1$ to $n3$, UDP flows from $n2$ to $n4$, TCP as well as UDP flow are used as background traffic from node $n5$ to $n6$ of Figure 1. The RTT is varied by changing the link delay. The link delay for the path of TCP is kept fixed while it is varied for both CBR and background traffics. First, we keep the number of flows fixed to 10 micro flows per aggregate flow and later we fix the RTT and vary the flows from 5-200.

Figure 6 (a) and (b) show the throughput achieved by TCP and CBR flows for varying RTT and varying flows respectively. Background traffic is not shown here. The flow control algorithm works nicely for different RTTs. The output does not change significantly. Figure 6(b) is more interesting because the bandwidth achievement changes with the number of active flows. When there is a lot of flows, TCP gets a good share even without having any flow control. It is because when many flows are present in a system, some flows starve and go for long time-out where as others still can get service. High volume of traffic makes it possible to increase the overall gain by the TCP flows. One interesting point we observed is that, the flow control algorithm needs a close approximation of active flows (see algorithm at the end of Section 3. If there are 200 active flows in the system, the algorithm works fine even if the approximation is 100 but fails to gain good performance if the approximation is 10. It is also true in the reverse manner, that is if there are 10 flows in the system and if the approximation is 200 then the performance of unresponsive flows deteriorate.

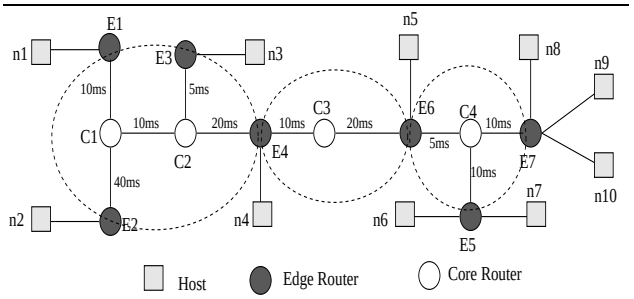
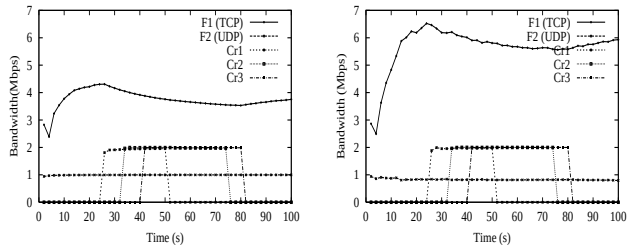


Figure 7: Complex topology with CBR cross traffic among n3-n4, n5-n6 and n7-n10.



(a) No Flow Control (b) With Flow Control

Figure 8: Dynamic adjustment of F2 flow works fine in presence of cross traffic. TCP flow (F1) gets more bandwidth with flow control scheme.

5.3 Simulation with Cross Traffic

We use more complex topology with multiple domains and with cross traffic to test our framework. The topology is shown in Figure 7. There are several aggregate flows present in this case such as TCP flow between $n1 - n8$, UDP flow between $n2 - n9$, $n3 - n4$, $n5 - n6$, and $n7 - n10$. We label the flows $F1$ between $n1 - n8$, $F2$ between $n2 - n9$ and $Cr1$, $Cr2$ and $Cr3$ flows between $n3 - n4$, $n5 - n6$, and $n7 - n10$ respectively. These Crs are used as cross traffic. We set the start and the finish time of these Crs flows differently to change the overall traffic situation over the path for the flows $F1$ and $F2$. There are 10 micro flows per aggregate in this setup. Flows $F1$ and $F2$ have same profile of target rate 5 Mbps and all cross traffic are sending at a rate of 2 Mbps.

Figure 8 shows the bandwidth achievement of all aggregate flows mentioned above with and without flow control. The cross traffic achieves the same target in both scheme. This cross traffic does not suffer because the flows do not send more than their profile and they do not see any bottleneck on their way. If there is no flow control, $F1$ (TCP) can not even get its target 5 Mbps, With flow control mechanism, $F1$ gets more than the target. It is because after controlling the UDP flow, TCP gets some unused bandwidth.

6 Conclusions and Future Work

We proposed and evaluated a simple way to detect and regulate unresponsive flows to prevent congestion collapse due to undelivered packets. Our scheme requires little modification at the core and does not introduce a lot of overhead. The modification at the edges (ingress and egress) is not major. The implementation is simple and the deployment will be easy. In our scheme, core/egress routers sends control packets only at the time of congestion. For this reason, this scheme will not introduce a lot of control packets into the network. We plan to implement our algorithm in Linux and setup a test-bed to obtain experimental results to complement and validate the simulation results of our framework as a future work.

Acknowledgements

The authors would like to thank Sonia Fahmy and Mohamed Hefeeda for their valuable suggestions.

References

[1] C. Albuquerque, B. Vickers, and T. Suda. Network Border Patrol. *IEEE INFOCOM 2000*, 2000.

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

[3] B. Braden and et al. Recommendations on queue management and congestion avoidance in the internet. RFC 2309, April 1998.

[4] H. Chow and Leon-Garcia A. A feedback control extension to differentiated services. Internet Draft, draft-chow-diffserv-fbctrl-00.pdf, March 1999.

[5] D.D. Clark and W. Fang. Explicit allocation of best effort packet delivery service. *IEEE/ACM Transactions on Networking*, 6, 4:362–374, 1998.

[6] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, August 1999.

[7] S. Floyd and V. Jacobson. Random Early Detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1, 4:397–413, 1993.

[8] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB group. RFC 2597, June, 1999.

[9] M Mahajan and et al. Controlling high bandwidth aggregates in the network. Technical Report, ACIRI, Feb 2001.

[10] S. McCane and S. Floyd. Network simulator ns-2. http://www.isi.edu/nsnam/ns/, 1997.

[11] T. Ott, T. Lakshman, and L. Wong. SRED: Stabilized RED. *In Proceedinds of the IEEE INFOCOM*, March 1999.

[12] N. Seddigh, B. Nandy, and P. Pieda. Study of TCP and UDP interaction for the AF PHB. Internet Draft, 1999.

[13] F. Shallwani, J. Ethridge, P. Pieda, and M. Baines. Diff-Serv implementation for ns. http://www7.nortel.com:8080/CTL/#software, 2000.

[14] H. Wu, K. Long, S. Cheng, and J. Ma. A Direct Congestion Control Scheme for Non-responsive Flow Control in Diff-Serv IP Networks. Internet Draft, draft-wuht-diffserv-dccs-00.txt, August 2000.