Influence of the discarding the out-of-profile packets policy on TCP traffic

Vasil Hnatyshin, Joshua Ogren, Jonathan Pucci, and Christopher Clement.
hnatyshin@rowan.edu, {ogrenj54, puccij25, clement55}@students.rowan.edu
Computer Science Department
Rowan University
Glassboro, NJ 08071

Abstract

In the recent year a variety of approaches for QoS support in the Internet such as Integrated Services [1], Differentiated Services [2], Bandwidth Distribution Scheme (BDS) [3], and others have been proposed. Most of the proposed approaches rely on the traffic policing unit, which may mark, delay, or drop packets that arrive above their reserved rate. The packets that arrive above their reserved rate are often referred to as out-of-profile packets. Often the policer limits the transmission rate of the flows that enter the network by dropping all out-of-profile packets. The Transport Control Protocol (TCP) [4] treats packet loss as an indication of congestion and reduces the flow's congestion window. Reducing the flow's congestion window effectively reduces the transmission rate of that flow. Thus, dropping out-of-profile packet may have a double effect on the transmission rate of the flow: the flow's rate is reduced by the traffic policer and by the congestion control mechanism of TCP. In this paper we examine the influence of the traffic policer that drops out-of-profile packets on the TCP traffic. We examine the performance of the traffic policer within the BDS framework using the OPNET Modeler network simulator [5].

Keywords: Bandwidth distribution, traffic policing, token bucket, TCP.

1. Introduction

As the Internet expands and diversifies, the current best effort approach to Quality of Service (QoS) in the Internet is no longer able to address the growing needs of the new emerging applications. To solve the problem of providing Quality of Service in the Internet, a number of service differentiation models have been proposed. The Integrated Services (IntServ) model[1], the Differentiated Service (DiffServ) approach[2], and the Bandwidth Distribution Scheme (BDS) [3] are just examples of the few among many approaches that have been introduced in the last two decades. Generally, to guarantee a particular level of service the network negotiates a traffic profile with the user and then enforces it in the network routers. Often the traffic profile is included as a part of the Service Level Agreement (SLA).

When a packet arrives at the router it is being classified to identify the SLA the packet belongs to, metered to determine if the packet conforms to its profile, and then policed according to the rules specified in the SLA. The traffic meter differentiates between the packets that conform to their profiles (these packets are called in-profile) and the packets that do not conform to their profile (these packets are called out-of-profile). The traffic policer forwards all in-profile packets to their corresponding outgoing interfaces and "punishes" all the out-of-profile packets. The "punishment" of an out-of-profile packet may include shaping the packet (e.g. delaying the packet until it becomes in-profile), marking the packet to indicate that it arrived at the rate outside of the negotiated profile, or simply dropping the packet.

The policy of dropping out-of-profile packets may have a negative effect on the TCP traffic because TCP treats packet loss as an indication of congestion. When TCP perceives that the packet was lost it reduces the congestion window of the flow, which effectively lowers the transmission rate of that flow. Thus, discarding the out-of-profile packets may have a double effect on the flows: the transmission rates of the flows are reduced by the traffic policer and by the congestion control mechanism of TCP [4].

This paper investigates the effects of the policy of dropping the out-of-profile packets on the TCP traffic. We perform this study within the BDS framework that employs traffic policer at the network edges. In the BDS approach the traffic rates are dynamically adjusted based on the network characteristics and are strictly enforced at the network edges: the edge routers discard all out-of-profile packets. The rest of the paper is organized as follows. Section 2 presents an overview of the Bandwidth Distribution Scheme and a summary of TCP congestion control. Section 3 describes the simulation set-up and collected results. Finally, Section 4 analyzes collected results and provides summary and conclusions.

2. The overview of the Bandwidth Distribution Scheme

The Bandwidth Distribution Scheme[3] was developed to provide per-flow bandwidth guarantees in a scalable manner. The BDS core routers do not maintain per-flow information (e.g. bandwidth requirements of individual flows); instead core routers keep aggregate flow requirements. The amount of information kept in the network core is proportional not to the number of flows but to the number of edge routers, which we believe does not raise scalability concerns. The edge nodes maintain per-flow information and fairly allocate network resources (e.g. bandwidth) among individual flows according to the flow requirements and resource availability. The dynamic resource allocation at the edge routers is enabled by the network feedback which consists of periodic path probing and explicit congestion notifications. Overall, the BDS architecture consists of: the admission control mechanism, which determines if a new flow can be admitted into the network, the resource management mechanism, which fairly distributes available bandwidth among individual flows, and the protocol for distribution of the aggregate flow requirements, which provides feedback to the network routers about the changes of network characteristics.

In the BDS approach, it is assumed that each user-flow negotiates (e.g. requests) the bandwidth range called the Requested Bandwidth Range (RBR). The RBR of flow f, RBR^f, consists of two values: a minimum rate, b^f, below which the flow cannot operate normally, and the maximum rate, B^f, that the flow can utilize.

$$RBR^f = [b^f, B^f] \qquad (1)$$

The BDS approach guarantees that each flow would receive at least its minimum requested rate b^f. The excess bandwidth, (e.g. what is left after each flow is allocated its minimum requested rate) is fairly distributed among the flows that may benefit from it. The BDS resource management mechanism computes the transmission rate of the flow *f* that travels on the path *P* with the bottleneck link *l* as follows:

$$R_P^f = \min\left(B^f, b^f + \varphi_l^f\left(C_l - b_l\right)\right) \qquad (2)$$

In equation (2), C_l is the capacity of bottleneck link *l*, b_l is the aggregate of the minimum requested rates of the flows that travel through *l*, and φ_l^f is the fair share of flow *f* on *l*. The fair share of flow *f* on link *l* is usually computed as the ratio between the RBR of flow *f* and the aggregate RBR on *l*.

The BDS edge routers classify, meter, and then police all incoming traffic. By default, the edge routers drop all the packets that arrive above the flow's transmission rate R_P^f computed by equation (2). The BDS edge routers implement the per-flow traffic policers as an array of token buckets (e.g. one token bucket per flow). The token bucket is configured with the token generation rate $R_{TB}^{f} = R_{P}^{f}$ and the token bucket size S_{TB}. If the BDS edge router receives packet *p* of flow *f* at time T_i and the previous packet of flow *f* arrived at time T_{i-1} then the edge router will discard packet *p* of flow *f* only if:

$$R_{TB}^{f}(T_i - T_{i-1}) + S_{TB} < size(p) \qquad (3)$$

In this paper we examine how the above BDS resource management mechanisms influence the TCP traffic. The TCP congestion control employs so called "additive increase – multiplicative decrease" approach. In the absence of congestion, TCP increments transmission rates of individual flows (congestion window) by one full-sized segment per round-trip time (RTT) (linear increase during congestion avoidance). However, when congestion occurs, TCP cuts the flow transmission rates in half [4]. TCP treats each packet loss as an indication of congestion which may conflict with the discard all out-of-profile policy of the BDS traffic policer.

3. Simulation configuration and results

3.1. Topology Configuration

In this study we use the "dumbbell"-like topology shown in Figure 1. In this topology source nodes *client_1, client_2,* and *client_3* send traffic to destination nodes *server_1, server_2,* and *server_3*. The generated traffic traverses the bottleneck link *router_0 – router _1*. We examine performance of the BDS discarding policy using UDP and TCP transport protocols. In particular, we examine how the BDS approach distributes available bandwidth among individual flows. It should be noted, that the value of the link capacity is not the same as the amount of bandwidth provisioned for BDS traffic. In this study we assume that the network administrator provisions a portion of the link capacity for the BDS traffic. Thus, the BDS attempts to utilize all the bandwidth allocated for its traffic.

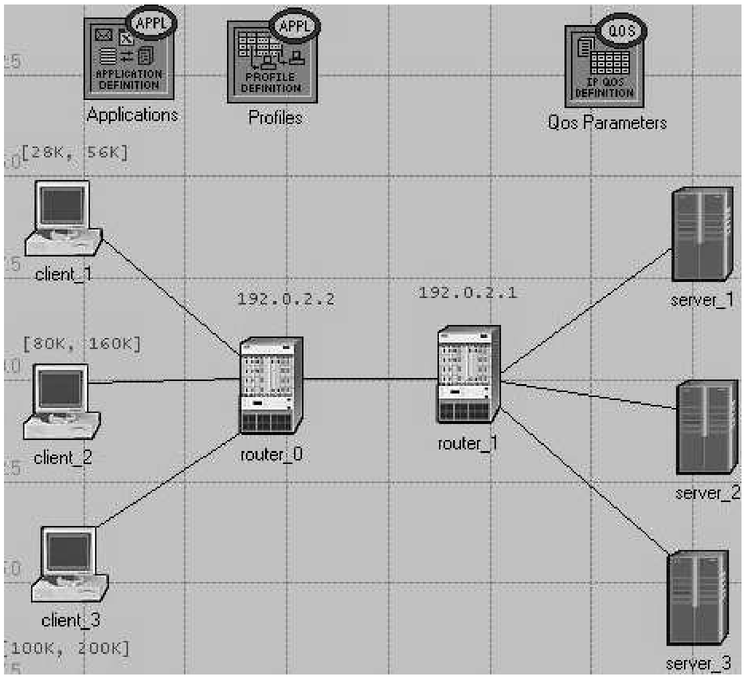

Figure1. Simulation Topology.

3.2. UDP scenario

In this set of experiments the client nodes were configured to establish video conferencing sessions with their corresponding server nodes. The video traffic has strict Quality of Service requirements in terms of throughput and jitter and because of that it uses UDP as its transport protocol. In this scenario the bottleneck link between *router_0* and *router_1* was provisioned with *4 Mbps* of bandwidth for BDS traffic. Each flow was transmitting traffic at the constant rate that was slightly above the flow's maximum RBR. We ran simulation for 100 seconds. Table 1 shows the summary of scenario's configuration.

Source	**Destination**	**Start Time**	**End Time**	**Min RBR**	**Max RBR**
client_1	*server_1*	10 sec	100 sec	500 Kbps	1000 Kbps
client_2	*server_2*	10 sec	85 sec	1000 Kbps	1500 Kbps
client_3	*server_3*	10 sec	60 sec	2000 Kbps	2500 Kbps

Table 1. Configuration of UDP scenario.

UDP is an unreliable transport protocol and does not react to the data losses. That is why we expected that each flow will receive exactly its fair share as computed by the BDS resource management mechanism. Table 2 shows the computed fair share for the initial [10 sec, 60 sec] time period. Figure 2 illustrates achieved individual flow throughput for UDP scenario. As expected, during the initial 60 seconds the traffic flows observed the throughput that was equal to their fair shares as computed in Table 2. Such bandwidth distribution consumed 100% of the bandwidth allocated for the BDS traffic.

When flow *client_3 – server_3* terminated at time 60 seconds, the aggregated maximum requested rate of remaining BDS flows was larger than the BDS capacity. According to equation (2), the BDS flows cannot transmit above their maximum requested rates. Thus, after termination of flow *client_3 – server_3*, the remaining flows *client_1 – server_1* and *client_2 – server_2* were allocated the amount of bandwidth equal to their corresponding maximum requested rates of 1000 Kbps and 1500 Kbps, respectively.

Source	**Destination**	**Bandwidth Share**
client_1	*server_1*	500 + (4000 – 3500) * (1000 – 500)/(5000 – 3500) = **633 Kbps**
client_2	*server_2*	1000 + (4000 – 3500) * (1500 – 1000)/(5000 – 3500) = **1133 Kbps**
client_3	*server_3*	2000 + (4000 – 3500) * (2500 – 2000)/(5000 – 3500) = **2133 Kbps**

Table 2. Computed flow fair shares during [10 sec, 60 sec] time period.

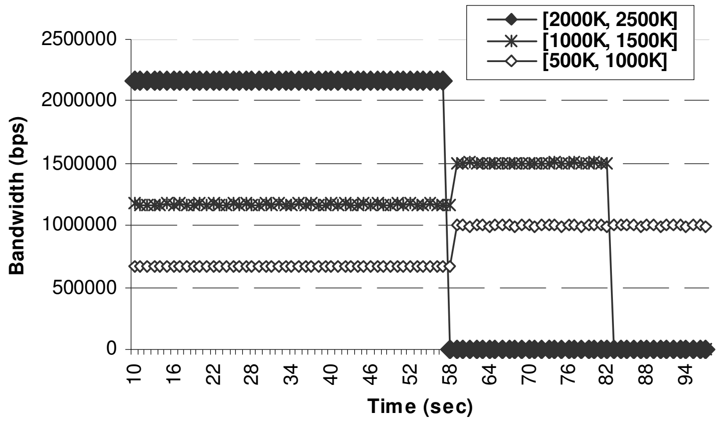

Figure 2. Flow throughput for UDP scenario.

This simple scenario shows that when using UDP transport protocol the policy of discarding the out-of-profile traffic does not interfere with the BDS resource distribution mechanism and allows each flow to achieve the throughput that corresponds to its fair share.

3.3. TCP scenario configuration

In this scenario the client nodes were configured to upload a large file to their corresponding server using the FTP application. Each flow transmitted traffic until the FTP upload was completed. Simulations were run for 300 seconds. FTP applications are very sensitive to loss and thus require reliable data transfer offered by TCP transport protocol. Since FTP applications do not require as much bandwidth as video traffic we adjusted the RBR of individual flows as well as the amount of bandwidth provisioned for BDS traffic in the network. We allocated 300 Kbps of available bandwidth for the BDS traffic on the bottleneck link. We set the individual client RBR to roughly correspond to the current connection types. Table 3 summarizes the flow configuration for TCP scenario.

Source	**Destination**	**FTP Upload**	**Min RBR**	**Max RBR**	**Connection Type**
client_1	*server_1*	1 MByte	28 Kbps	56 Kbps	Dial-up
client_2	*server_2*	2 MByte	80 Kbps	160 Kbps	DSL
client_3	*server_3*	4 MByte	100 Kbps	200 Kbps	Cable

Table 3. Flow configuration of TCP scenario.

3.4. Flow throughput vs. link capacity

First, we examined the performance of the FTP applications for two different BDS capacity settings: 300 Kbps and 1000 Kbps. In this simulation we used the *New Reno* flavor of TCP. Table 4 briefly summarized configuration of *New Reno* and other TCP flavors used in this study.

TCP Flavor	**MTU (bytes)**	**Receiver buffer (bytes)**	**Fast Retransmit**	**Fast Recovery**	**Selective Acknowledgement**
Tahoe	512	64 K	OFF	OFF	OFF
Reno	512	64 K	ON	Reno	OFF
New Reno	512	64 K	ON	New Reno	OFF
SACK	512	64 K	ON	New Reno	ON

Table 4. Summary of TCP flavor settings.

Table 5 presents the results collected in this scenario. The results showed that regardless of the capacity allocated for the BDS traffic, the actual throughput achieved by individual FTP applications was below their allocated fair shares. In some cases the simulation reported that the flow throughput was below the flow's minimum requested rate.

Source	**RBR (Kbps)**	**Capacity = 300 Kbps**		**Capacity = 1000 Kbps**	
		Fair Share	**Throughput**	**Fair Share**	**Throughput**
client_1	[28, 56]	40.38 Kbps	18.82 Kbps	56 Kbps	20.95 Kbps
client_2	[80, 160]	115.4 Kbps	79.22 Kbps	160 Kbps	98.84 Kbps
client_3	[100, 200]	144.2 Kbps	76.68 Kbps	200 Kbps	120.36 Kbps

Table 5. Fair Shares and Achieved Throughput for TCP flows.

3.5. Flow throughput vs. TCP flavor

In this section we examined the throughput achieved by individual flows when using different TCP flavors. In particular, we examine achieved throughput when using the following TCP flavors: Tahoe, Reno, New Reno, and Selective Acknowledgement (SACK).

The Tahoe flavor of TCP does not implement the fast recovery/fast retransmit mechanism. TCP Tahoe identifies the TCP segment loss based on the expiration of the retransmission timer. TCP Reno implements the fast recovery/fast retransmit mechanism and identifies the segment loss as follows. When the sender receives *three* consecutive acknowledgements (ACK) for the same segment then it assumes that the segment was lost and immediately re-transmits that segment without waiting for the timeout.

TCP New Reno also implements the fast recovery/fast retransmit mechanism. However, if during the fast-retransmit procedure another segment loss occurs, the fast-retransmit procedure is NOT terminated and restarted as in TCP Reno. If a partial segment acknowledgement (e.g. an ACK for a recently lost and re-transmitted packet arrives but not for subsequent packets) then TCP assumes that the following packet was also lost. This packet is being retransmitted without waiting for 3 duplicate ACKs and without restarting fast retransmit. Finally, the SACK flavor of TCP uses the same fast recovery/fast retransmit mechanism as TCP New Reno but instead of returning a cumulative ACK for each received TCP segment, it returns an ACK that lists the sequence numbers of successfully received TCP segment within its window. TCP SACK allows faster identification of the packet and thus faster response to congestion.

Table 6 display the achieved application throughput for different TCP flavors. These results were collected using flow configuration of Table 3 and BDS capacity of 300 Kbps. We ran each simulation five time and averaged the results.

Source	**RBR (Kbps)**	**Fair Share (Kbps)**	**Flow Throughput (Kbps)**			
			Tahoe	**Reno**	**New Reno**	**SACK**
client_1	[28, 56]	40.38	15.67	15.23	16.31	65.00
client_2	[80, 160]	115.4	60.72	49.81	52.35	122.90
client_3	[100, 200]	144.2	80.43	73.39	80.08	150.54

Table 6. Fair Shares and Achieved Throughput for different TCP flavors.

The collected results show that when using TCP Tahoe, Reno, or New Reno, the average flow throughput is significantly smaller than the corresponding flow fair share. However, when using TCP SACK, the average flow throughput exceeded the corresponding flow fair share. We were not able to discover the reasons for such behavior and currently investigating this phenomenon.

3.6. Flow throughput vs. TCP receiver window

Finally, we conducted a study that examines the achieved flow throughput for different TCP receiver window sizes. Reducing the TCP window size causes the actual TCP transmission rate to decrease. This, in turn reduces the number of TCP segments dropped by the BDS traffic policer and also limits the frequency of invocation the fast recovery/fast retransmit mechanism. Table 7 shows collected results for scenario flow configuration of Table 3 and BDS capacity of 300 Kbps. We ran each simulation five time and averaged the results. The collected results shown that regardless of the TCP receiver window size the flows were not able to achieve the throughput level that corresponds to their fair shares.

Source	**RBR (Kbps)**	**Fair Share (Kbps)**	**Flow Throughput (Kbps)**				
			8 Kbytes	**16 Kbytes**	**32 Kbytes**	**64 Kbytes**	**128 Kbytes**
client_1	[28, 56]	40.38	12.55	10.28	12.29	16.035	20.91
client_2	[80, 160]	115.4	54.17	65.56	83.84	64.00	46.14
client_3	[100, 200]	144.2	75.93	75.12	76.03	67.50	71.35

Figure 7. Fair Shares and Achieved Throughput for different Receiver window sizes

4. Analysis and conclusions

In this study we examined how the policy of discarding the out-of-profile traffic influences the throughput of TCP and UDP flows. We used the BDS token bucket traffic policer to discard the out-of-profile packets. When using UDP as transport protocol all the flows were able to achieve the throughput that corresponds to the flow's fair share. The reason for this behavior is the "non-responsiveness" of the UDP. UDP ignores packet losses and continues to transmit traffic at the same rate.

On the other hand, TCP reduces transmission rate of a flow whenever the flow experiences the packet loss. The TCP behavior causes data packets to arrive in bunches which causes the token bucket to discard consecutive packets. As the result, when token bucket drops multiple consecutive packets, the TCP congestion control mechanism reduces the TCP congestion window to 1 which slows down the flow's transmission rate to almost a complete halt. This process repeats multiple times until TCP converges to the "optimal" window size. Unfortunately, most of the TCP flows are short lived (e.g. E-mail,. Telnet, HTTP, etc) and often terminate well before TCP converges, which prevents them from achieving their allocated fair share. In addition, TCP experiences segment loss even after it converged, causing the window size and the transmission rate of the flow to fluctuate and go below the flow's allocated fair share. These facts explain why the TCP flows experience the throughout that is significantly lower than the flow fair share allocated by the BDS resource management mechanisms.

In conclusion, this study showed that the policy of discarding the out-of-profile packets does not work well with TCP flows. Such policy has double negative effect on the TCP flows: first, the flow's rate is reduced by the token bucket that drops all out-of-profile packets and then by the TCP's congestion control that reduces transmission rate upon each packet loss. As the result, the throughput of the TCP flows is significantly lower than the corresponding rate limit enforced by token bucket. This study was the first step in out investigation of the packet discarding policies and their influence on TCP traffic. Currently, we plan to investigate other means for punishing the out-of-profile traffic. In particular, we plan to study mechanisms where the packets are delayed instead of being dropped, the edge routers notify TCP directly about the need to adjust transmission rates, and the traffic policer avoids discarding consecutive packets and distributes the packet loss of individual flows over a period of time.

References:

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

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

[3] V. Hnatyshin and A.S. Sethi, "*Scalable Architecture for Providing Per-flow Bandwidth Guarantees*," Proceedings of the IASTED conference on Communications, Internet and Information Technology (CIIT 2004), St. Thomas, Virgin Isles, November 2004.

[4] Stallings, William. *Computer Networking with Internet Protocols and Technology,* Pearson Prentice Hall, Upper Saddle River New Jersey, 2004, p246-263.

[5] Opnet Modeler, version 10.5. http://www.opnet.com/