Dynamic Traffic Conditioning for a DiffServ domain using Feedback and Topology Techniques

Athina Markopoulou
Stanford University
amarko@leland.stanford.edu

Barani Subiah
3Com Corporation
barani_subbiah@3com.com

In this paper, we propose and evaluate a dynamic traffic conditioning framework for a Differentiated Services (DS) domain that improves the resource efficiency and the stringency of the QoS guarantee. The DiffServ architecture provides services across a domain, by combining a small number of aggregate Per Hop Behaviors (PHBs) at the interior nodes, with appropriate traffic conditioning at the boundary of the domain. It is not obvious how to appropriately combine the above two elements in order to provide the QoS guarantees to already admitted flows, despite aggregation of flows or burstiness of traffic. We propose a framework that enables (i) the interior nodes to monitor the congestion and report it back to the boundary nodes and (ii) the ingress nodes to appropriately adjust their traffic conditioning function to follow the dynamics of the domain. We show that discarding packets at the boundary nodes than at interior nodes improve the overall utilization of the domain and helps to preserve QoS guarantees to already admitted traffic.

1. Differentiated Services

DiffServ neither provide better service, i.e. higher quality, to all users nor always strict end-to-end guarantees [1]. However, DiffServ does provide different levels of QoS to different applications. There are two main motivations behind it. The first one is economics-driven: differentiated charging can only be justified by providing differentiated service. The second motivation is that different applications require different types and levels of guarantees. For example, file transfer requires 100% reliability but may tolerate delay. Voice on the other hand is delay sensitive.

The strength of DiffServ is its scalability and its simplicity. By having the interior nodes keeping state only for a few classes, as opposed to the IntServ approach that kept state for all flows, DiffServ provides differentiated QoS while maintaining the stateless and scaleable properties of the IP architecture [2]. However, the fundamental problem of DiffServ is that the desired behavior of an application is specified per flow end-to-end, while the DiffServ mechanisms are defined per hop. PHB determines the treatment for a class at one node. It is not obvious how PHBs combined with traffic conditioning at the edge can construct predictable end-to-end to services, even inside one DS domain. First, the service provided at a DS node to a class is often qualitative, e.g Assured Forwarding (AF) [3] class, so the end-to-end service cannot be but relative. Secondly, flows of the same class aggregate in unpredictable ways inside the network. The final result depends on the aggregate stream load of a class arriving at a node. The problem and solution we explore in the following sections, that is how to monitor and report the congestion level in a DS domain and how to adapt the conditioning at the boundary so that it follows the dynamics of the domain, is crucial for the end-to-end success of the DiffServ architecture.

2. Problem definition

The fundamental problem in the DiffServ approach is that the Traffic conditioning (TC) at border nodes is static in a sense that TC does not adapt to the state of the interior nodes. Forwarding resources, buffer and bandwidth, are allocated for each class at interior nodes usually by the network operator. Border nodes perform appropriate conditioning on the incoming flows so that the arrival rate of aggregated flows of same class at each interior node will not be more than expected. However, network dynamics cause traffic belonging to the same class to be directed via a particular interior node thus causing congestion. Since TCs at border nodes does not adapt to congestion information, more traffic is admitted which leads to more packet loss inside a DiffServ domain. Since packet loss within one class is random, there is no guarantee that packets from one source is favored than other sources. For instance, a packet arrived through many hops, using up stream resources, may be dropped than a packet arrived from a single hop at a down stream node. This problem is further illustrated in Figure 1. When there is a congestion at node 3, which is shared by traffic flows AA' and CC', packets belong to both flows are dropped at random. However, it is evident that a packet loss of AA' flow has more damage than CC' since upstream resources used by AA' have gone waste. If border node 1 could some how control the AA' flow to match the drop probability at interior node 3 then traffic flow BB' would achieve a higher throughput.

The DiffServ framework as of today, does not address this problem. It just provides the building blocks, PHBs and traffic conditioning (TC) capabilities, and says that they should be appropriately combined to provide the desired end-to-end result [1]. We claim that if (i) we had some feedback information about congestion in the interior of the DS domain and (ii) the traffic conditioning at the edges used this information and adapt, then we would be able to avoid the above described problem. We would be able to drop/buffer packets at the border of the DS domain, rather than admit them at the first place, let them use/waste network resources, that could be used otherwise, and drop them later. In summary, the problem we are studying in the DiffServ context is the following: how to keep TC and PHBs matched, in order to improve network efficiency and provide the service guarantees across a DiffServ domain.

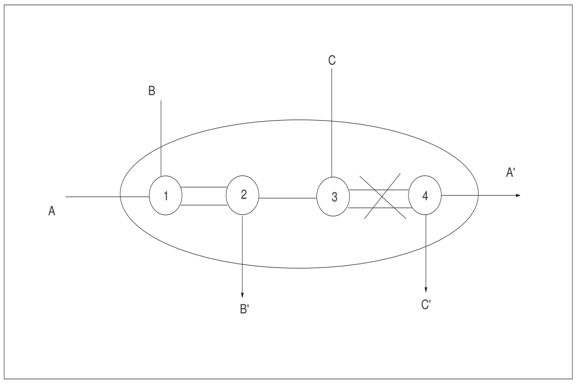

Figure 1: Example 1

3. Dynamic traffic conditioning framework

The mechanism proposed in this paper to solve the above problem consists of two key concepts. The first concept defines a way to propagate the congestion information from an interior node to the border nodes, while the second concept involves how the border nodes react to the congestion information. The functionalities needed at an interior node include monitoring the average queue size of a single traffic class, determining the congestion criteria and notifying border nodes within the DiffServ domain. In our approach, we rely on RED [4] mechanisms that compare a smoothed average queue size to some thresholds to decide if there is congestion. Once congestion is detected, notification sent from an interior node to the boundary node, may include various information. A simple case may be that an interior node sets a single bit flag to indicate congestion. More complex notification messages could include congestion level (eg. low, medium, or high average queue size), higher order statistics (eg. growth rate), drop probability value as calculated by RED. In our implementation, interior routers use ECN [5] capable RED, and they detect congestion whenever RED would normally set the ECN bit.

Once an interior node detects congestion, it constructs an Internet Control Message Protocol (ICMP) packet, containing its node ID (IP address) as well as congestion indication information using one of the schemes as described previously, ie. set a single bit flag. There are several ways an interior node can inform border nodes. One method is that the interior node copies the destination IP address, found on the last IP packet, onto the ICMP packet and then forwards it as normal. In this scheme, all ICMP packets received by an egress border node are analyzed. If the ICMP is a congestion indication ICMP packet then border node processes it and informs other border nodes. Alternatively, interior nodes could maintain a list of (one, some or all) border nodes and notify them once congestion is detected, probably using multicast. The set of border nodes to be notified, could be manually configured or dynamically discovered by an interior node. Once border nodes are notified about the congestion, they should decide on what actions should be taken to alleviate the congestion. In our approach, only one egress node, the default one, is notified about congestion, and then it notifies all ingress nodes.

Congestion notification from interior nodes to border nodes has few disadvantages. Bandwidth wasted by ICMP congestion packets is one of the primary concerns. However, limiting the frequency of congestion notification will lower the bandwidth inefficiency to certain extend. In our framework, we propose that congestion notification from an interior node must be distributed to all border nodes. This is to forewarn a border node that is about to send traffic to a congested node so as to avoid further congestion. Another disadvantage with any feedback scheme is the network oscillation. If feedback is frequent then border nodes react often, causing network oscillation. This is a serious problem and should be avoided in the first place. It is intuitive that frequent feedback will result in a better decision making, but it will decrease the confidence level of the feedback. We limit the frequency of congestion notification by using a timer, which is set to Round Trip Time (RTT) of diameter of the DiffServ domain. An interior node will notify only once with in the timer expiry even when congestion persists. We believe that the timer will suffice the time needed by the border nodes to receive the congestion indication and react accordingly. Congestion notification is re-sent only when the congestion continues beyond the timer expiry. Psuedocode of our monitoring framework is shown in Figure 2.

```
While () {
   Monitor the average queue size (RED mechanisms)
   If Congestion (if RED would normally set ECN bit)
   Create an ICMP message with destination address of last seen IP packet
   Indicate congestion information
   Forward it to the next hop
   Wait until Timer T1 expires (T1 == RTT of diamter of the DS domain)
}
```

Figure 2: Algorithm at the interior node

The second important aspect of our framework is how the ingress border nodes react to the congestion indication. Before a border node react, it has to verify that any of its own traffic passes through the congested node. This is determined by the topology information stored at border nodes. If the DiffServ domain uses Open Shortest Path First (OSPF) then we suggest that the OSPF topology table be used. In other cases, we propose that the topology information of a DiffServ domain is stored at border nodes. As soon as a border node receives a congestion notification from an interior node, it verifies which micro-flows pass through the congested node. For this purpose, we propose that a border node pre-calculate the paths using the topology table for all micro-flows and store them in its cache. If there is a topology change then the paths have to be re-calculated. Since all paths are in cache, it is faster to find which of the micro-flows that passes through a congested node. Once the border node identifies which micro-flows are passing through a congested node, it can react to the congestion in one of many ways as described below. A border node can drop packets, re-mark the packets to a lower priority, or delay the packets. Regardless of the method applied by a borer node, the biggest question is how border nodes co-ordinate among themselves so that congestion is alleviated at the interior node and it does not re-appear as soon as the they return to their full traffic rate.

In the first method, all border nodes that are currently sending traffic through the congested node lower their rate by certain percentage for a pre-specified period of time. The time period is determined by the round trip time between a border node and the congested node. We assume that round trip time can be calculated by using ping command and stored at cache in advance. Once the time period is elapsed, border node can resume its full traffic rate. In the second method, border nodes use flip a coin scheme to determine whether it should reduce the rate or continue as normal. If all border nodes reduce their traffic at once then it is possible that the traffic rate at the interior node could be lower than its capacity thus wasting network resources. With the use of flip a coin scheme, it is probable that half of the nodes will reduce the rate and other half of them will send traffic as usual. It is likely that the lower input rate by half of the border nodes would help alleviate the congestion at an intermediate node. The nodes that decided to lower the traffic can restart at full rate after the RTT timer expiry. In the third method, border nodes can use number of hops between itself and the congested node as a parameter to decide how much of the traffic rate can be lowered. Higher number of hop count often indicates that the upper stream resources are shared by many flows. Based on the number of hops, a node can determine what percentage of traffic could be lowered.

After a set of ingress nodes agree to lower their traffic based on one of the above methods, it is also important that these nodes return to their normal state preferably after the congestion at the interior node is disappeared. It is also necessary that these border nodes do not return to their full traffic load at the same time, else it is possible that congestion will re-appear at the same interior node and could lead to a cycle. In order to avoid this problem, it is proposed that border nodes return to their normal state at different intervals. One way to implement this scheme is to calculate the round trip time between a border node and the congested node. If the round trip times can be calculated and kept in border node's cache in advance, then it is easy to use this value during the congestion phase. Given that round trip time between border nodes and a congested node varies, it is likely that different border node returns to normal rate at different intervals. RTT delay is also beneficial since it allows a border node to be sure that there is no further congestion indication from the same interior node. Our dynamic traffic conditioning framework is described as a pseudocode in Figure 3.

```
While () {
   Receive congestion notification and determine which interior node
   Check the cache topology table if any traffic sent to congested node
   If TRUE {
       Group all micro flows towards the bottleneck
       Determine the overall rate of all micro flows
       Reduce the rate "appropriatelly"
       Get the RTT to the Interior node from cache
       Wait until the timer expires
       If no further congestion indication {
         Increase traffic gradually
       }
   }
   Wait until next congestion indication
}
```

Figure 3: Algorithm at the border node

4. Simulation

We used Berkeley Network Simulator (NS) [6] to simulate our framework and the results are presented in section 5. In this section, we present our assumptions, the schemes we chose and the values we selected for the parameters. We considered a single DS class traffic only, which is why we implemented a single queue at interior nodes and used the existing RED to detect congestion. We implemented few additions to RED, named

"RED/ECN", so that it generates an ICMP notification packet, whenever RED would normally set the ECN bit. At this moment, we just report a single flag, together with the congested node's ID, as opposed to more complex congestion information discussed in Section 3. For this simulation, we let the congested nodes report the congestion directly to ingress nodes that send traffic towards them. To identify such ingress nodes, we assume knowledge of topology and traffic paths. As we mentioned earlier, high frequency of congestion notification can lead to additional congestion and network oscillation. The timer to limit the notification frequency was set to twice the diameter of the DiffServ domain.

Topology information of the DS domain is stored in a cache at all border nodes. In our simulation, we consider a static routing topology at border nodes. The traffic sources are CBR streams at this moment (VBR to be added). The reaction algorithm is also a simple one at this moment, implementing multiplicative decrease, additive increase of the rates. Our studies are limited to congestion control across one DS domain only. We have not studied the interaction of our approach with end-to-end congestion control, such as TCP. The reaction heavily depends on the terms of Traffic Level Agreements (TLA) are expressed. Our algorithm assumes an agreement based on constant rate. However realistic agreements should be expressed in terms of average or peak rates, burst and average rates.

In particular, we carried out our simulation using a simple topology as shown in Figure 4. There are two CBR flows: S1-R1-R2-S3 and S2-R1-R2-S4, 10Mbits each. These flows should be thought as aggregate of flows. The bandwidth of each link is shown in the figure and the delay is 20msec for all of them. The CBR packet size is 210 bytes. Routers R1, R2 implement "RED/ECN". Think of R1, R2 as output buffer switches, where each buffer is a RED queue. Based on topology and traffic knowledge, R1 notifies both S1 and S2 about congestion on link R1-R2 because they are both responsible for it. R2 notifies only ingress S2 about congestion on link R2-S4 because S2 is the one responsible for it. There is no congestion on the other links, so notification is not needed. R1 does not send more than one notification per 50msec, that is a bit more often than per RTT=2*20msec to the ingress nodes. R2 does not send more than one notification per 90msec, that is slightly above RTT=2*40msec to the ingress nodes. The purpose of these timers is to give ample time to the ingress nodes to react to the congestion. The timers a bit more than the RTTs, because there are some queuing and processing delays.

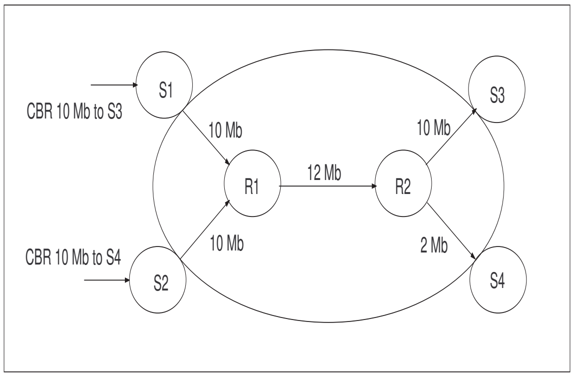

Figure 4: Simulation Model

Ingress nodes are able to change the rate of the flows according to the notification messages they receive. When ingress S1 and S2 receive these notification messages, they decrease the rate of the flow going though them towards the bottleneck, by half. S1 does not react more often than once per 50msec (slightly more than the RTT to R1, the furthest router affected by the S1-S3 flow) and S2 does not react more often than once per 90msec (slightly more than the RTT to R2, the further router affected). Each ingress also checks periodically (same time outs as above: 50msec and 90msec) itself. If it has not received any congestion messages during this interval, then it assumes that there is no further congestion downstream and increases the rate of the flows by a fixed percentage, e.g.. 20% or 50%.

Without any scheme, both flows would suffer equal losses on the shared RED link R1-R2, that is 6Mbps would pass from each flow. The lower flow S2-S4, used upstream shared resources (link R1-R2) although it got dropped anyway later. These shared resources (link R1-R2) could have been used to serve the upper flow S1-S3 instead. The ideal case here would be to admit just 2Mbps of the flow S2-S4 (that can pass through the downstream bottleneck link too) and all 10Mbps of the flow S1-S3. The following test cases will show how our scheme, implemented in NS, improves the network utilization.

5. Experimental results

We used the topology of Figure 4 for two scenarios. CBR traffic is going from S1 to S3 and from S2 to S4. Congestion indication is reported back to source S1, S2 which adjust accordingly the rates of the admitted traffic, in order to alleviate congestion. Figures 5 and 6 show that using our adaptive conditioning scheme, the rates of admitted traffic converge indeed to their optimal values. For test case 1, Table 1 shows the decrease of drops inside the domain and the improvement in utilization of the domain, with the proposed dynamic TC. Similarly, Table 2 shows the improvement in efficent use of the domain resources for test case 2.

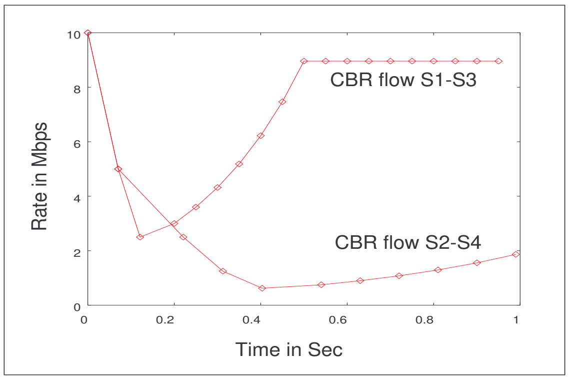

Figure 5: Traffic convergence rate - Test case 1

Table 1: Test Case 1

Link	% loss no TC	% loss with TC
In s1- Eg s3	39.1735	2.9481
In s2- Eg s4	79.6356	43.5056
Link s1-r1	0.0000	0.0000
Link s2-r1	0.0000	0.0000
Shared Link r1-r2	39.8791	5.6065
Link r2-s3:	0.0000	0.0000
Bottleneck Link r2-s4	65.8750	34.7054

5.1 Test case 1

As mentioned above, test case 1 was simulated with both traffic sources started at the beginning of the simulation. Since link R2-S4 does not have enough bandwidth to accommodate the flow S2-S4, packets are dropped at node R2. We studied two cases, first with out our scheme (no congestion control) and then with our scheme.

The traffic rate of both flows during the congestion was monitored and plotted in a graph which is shown in Figure 5. It can be seen that both flows S1-S3 and S2-S4 started at time 0.0 and their rates converged to their optimal value, for this topology, at 0.55sec. The final rates for flow S1-S3 is 8.96 Mbps and for flow S2-S4 is 1.87 Mbps, which are close to the optimal values. They are not exactly 2Mbps and 10Mbps, because of the coarse step of the reaction algorithm(rate adjustment). The finer the step we use, the closer to the optimal values, but the longer the convergence time. Another improvement we are considering is to use variable steps in our INCREASE/DECREASE procedures. During the simulation, we observed the number of congestion notification messages on all the links and the results have shown that the congestion notification overhead is negligible.

The results show that the final rates achieved with our scheme are close to the optimal ones, 2Mbps and 10Mbps. The loss on the upper flow is practically eliminated thus the overall network utilization has improved from a network providers view. The drops on the shared link R1-R2 are mainly experienced by the lower flow that will be anyway dropped downstream. The reaction time is short: approximately the RTT and beyond the reaction transient period the scheme converges quickly to the final rates and remains stable [CBR streams].

Table 2: Test case 2

Link	% loss no TC	%loss with TC
In s1-Eg s3	19.5364	5.0112
In s2- Eg s4	79.6624	52.5783
Link s1-r1	0.0000	0.0000
Link s2-r1	0.0000	0.0000
Shared link r1-r2	19.8723	8.0183
Link r2-s3:	0.0000	0.0000
Bottleneck Link r2-s4	74.6316	44.1227

5.2 Test case 2

This test was conducted with the 2 aggregate flows start and finish at different intervals of the simulation. Flow S1-S3 starts at 0.0 and finishes at 1.0sec while flow S2-S4 starts at 0.5 and finishes at 1.5sec.

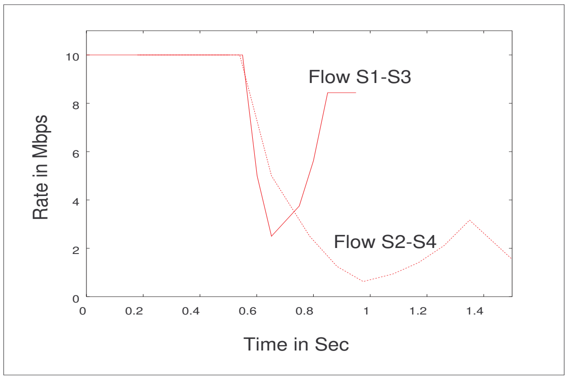

Figure 6: Traffic convergence rate - Test case 2

The final rates achieved by our scheme, are close to the optimal ones in test case 2 also. The plot as shown in Figure 6 indicates that the reaction and the convergence times are short in this case too. Congestion first appears around 0.5sec when the flow S2-S4 starts. The same plot shows that the fixed step increase in our algorithm has the following tradeoff: a large step leads to fast convergence but a small step leads to accuracy in achieving the desired rates. The messages overhead is still negligible. The final rates for flow S1-S3 is 8.44 Mbps and for flow S2-S4 is 1.58 Mbps.

6. Conclusion

In this paper, we proposed a framework in which (i) interior nodes monitor themselves and send feedback to the boundary nodes concerning their congestion level and (ii) ingress nodes react to congestion by adapting their traffic conditioning, based on the feedback received as well as knowledge of the DS domain topology. This way, domain resources are utilized effectively and predictable performance is guaranteed to already admitted traffic. Most of the complexity is delegated to the border nodes while interior nodes are kept simple, thus adhering to the DiffServ philosophy. The idea of using topology information to identify flows that causes congestion enables an ingress node to control traffic admittance accurately while maintaining fairness.

The purpose of the simple scenarios presented in this paper, was to demonstrate the value of the framework; to show that loss across the whole domain is reduced, better performance is provided to admitted traffic and this is done with minimal overhead and reasonable reaction times. We should, and we are indeed, experimenting further with different DS domain topologies, congestion reaction algorithms, parameters choices, and various traffic sources. Another open issue is the interaction with the TCP congestion control, which is anyway an issue in the DiffServ context.

References

[1] Y. Bernet, et.al, A framework for differentiated services, Internet draft March 1999

[2] Differentiated Services working group, http://www.ietf.org/html-charters/diff-charter.html

[3] J. Heinanen, e.a. Assured Forwarding PHB group, [draft-ietf-diffserv-phb-af-06.txt], work in progress

[4] S.Floyd e.a.: http://www.aciri.org/floyd/red.html

[5] S.Floyd e.a.: http://www.aciri.org/floyd/ecn.html

[6] Network Simulator, http://www-mash.cs.berkeley.edu/ns/