IPHV7I20034X
International Journal Of Advanced Research and Innovation -Vol.7, Issue .II
ISSN Online: 2319 – 9253
Print: 2319 – 9245

CONTROL METHODS FOR COMMUNICATION CONGESTION IN PACKET SWITCHING NETWORKS

[#1]Pratima Patil -M.Tech Pursuing,
[#2]P.Hari Krishna -Asst.Professor,
Department of Computer Science & Engineering,
Malla Reddy College Of Engineering & Technology, Hyderabad.

Abstract: *Presently the Internet accommodates simultaneous audio, video, and data traffic. This requires the Internet to guarantee the packet loss which at its turn depends very much on congestion control. A series of protocols have been introduced to supplement the insufficient TCP mechanism controlling the network congestion. CSFQ was designed as an open-loop controller to provide the fair best effort service for supervising the per-flow bandwidth consumption and has become helpless when the P2P flows started to dominate the traffic of the Internet.*

Token-Based Congestion Control (TBCC) is based on a closed-loop congestion control principle, which restricts token resources consumed by an end-user and provides the fair best effort service with O(1) complexity. As Self-Verifying CSFQ and Re-feedback, it experiences a heavy load by policing inter-domain traffic for lack of trust. In this paper, Stable Token-Limited Congestion Control (STLCC) is introduced as new protocols which appends inter-domain congestion control to TBCC and make the congestion control system to be stable. STLCC is able to shape output and input traffic at the inter-domain link with O(1) complexity. STLCC produces a congestion index, pushes the packet loss to the network edge and improves the network performance. Finally, the simple version of STLCC is introduced. This version is deployable in the Internet without any IP protocols modifications and preserves also the packet datagram. Although routers equipped with Active Queue Management such as RED [2] can improve transport performance, they are neither able to prevent congestion collapse nor provide fairness to competing flows.

In order to enhance fairness in high speed networks, Core-Stateless Fair Queuing (CSFQ) [3] set up an open- loop control system at the network layer, which inserts the label of the flow arrival rate onto the packet header at edge routers and drops the packet at core routers based on the rate label if congestion happens. CSFQ is the first to achieve approximate fair bandwidth allocation among flows with O (1) complexity at core routers.

Index Terms: - Policing, congestion, QoS, characterization, incentives.

1 INTRODUCTION:

The current Internet architecture trusts hosts to respond voluntarily to congestion; a feature commonly put down to The environment of mutual trust in which these algorithms emerged. Limited evidence shows that the large majority of end-points on the Internet comply with a TCP-friendly response to congestion. But if they didn't, it would be hard to force them to, given path congestion is only known at the last egress of

an internetwork, but policing is most useful at the first ingress.

Without knowing what makes the current co-operative consensus stable, we may unwittingly destabilize it. At the most alarmist, if this was to lead to congestion collapse [7] there would be no obvious way back. But even now, applications that need to be unresponsive to congestion can effectively steal whatever share of bottleneck resources they want from responsive flows. Whether or not such free riding is common, inability to prevent it increases the risk of poor returns, leading to under-investment incapacity.

To alleviate scalability problems that have plagued per flow solutions such as Fair Queueing [1] and Intserv [2], two new architectures have been recently proposed: Differentiated Services (Diffserv)[3], [4] and Stateless Core (SCORE) [5], [6], [7], [8]. A key common feature of these architectures is the distinction between edge and core routers in a trusted network domain (see Figure 1). The scalability is achieved by not requiring core routers to maintain any per flow state, and by keeping per flow or per aggregate state *only* at the edge routers.

In particular, with Diffserv, edge routers maintain per aggregate state [3], [9], [4], [10]. Core routers maintain state only for a very small number of traffic classes; they do not maintain any fine grained state about the traffic. With SCORE, edge routers maintain per flow state and use this state to label the packets [5], [6], [7], [8]. Core routers maintain no per flow state. Instead they process packets based on the state carried by the packets, and based on some internal aggregate state.

While Diffserv and SCORE have many advantages over traditional state full solutions, they also have serious limitations. The main source of these limitations is the implicit assumption that the information carried by the packets inside the network domain is "correct". This assumption is needed because core routers process packets based on this information. As a result, Diffserv and SCORE *cannot* transcend trust boundaries. This introduces three significant limitations.

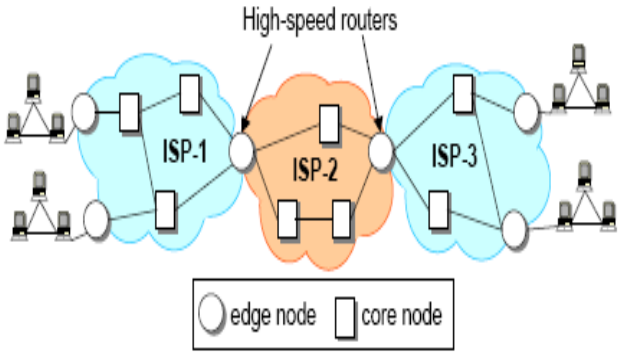

Fig. 1. Diffserv and SCORE network architectures. Edge routers maintain per aggregate or per flow state. Core routers maintain no per flow state. Highspeed boundary routers between distinct trusted domains are edge routers.

First, these architectures are not very robust because core routers have to trust information carried by the arriving packets, such as flow rate estimates or the DS field content. This enables a single faulty router to disrupt the service in an entire core merely by inserting incorrect state.1 Second, precisely because malfunctioning routers can have such a severe impact it is unlikely that cores will transcend trust boundaries. In particular, for reasons of trust the high-speed border routers between ISPs will be edge, not core, routers (see Figure 1). Thus, many of the claimed scaling advantages for core-stateless architectures will not be realized because these high-speed border routers will be required to perform per flow operations and keep per flow state.

Third, the deployment and configuration of routers must be done with extreme care to ensure there is a well-defined set of edge

IPHV7I20034X
International Journal Of Advanced Research and Innovation -Vol.7, Issue .II
ISSN Online: 2319 – 9253
Print: 2319 – 9245

routers surrounding the core. As noted above, a misconfigured router can have serious consequences at the level of the entire domain.

The goal of this paper is to address these limitations in the case of SCORE solutions. The main idea is to use *statistical verification* to identify and contain the flows whose packets carry incorrect information. To make this idea concrete, in this paper we focus on the case of CSFQ. In particular, we propose a design called *Self-Verifying* CSFQ (SV-CSFQ) that uses statistical It is important to note that what makes our approach possible in the case of SCORE solutions is that the state carried by flow verification to check the rate estimates. Routers no longer blindly trust the incoming rate estimates, instead they statistically verify and *contain* flows whose packets are incorrectly labeled. Thus, flows with incorrect estimates cannot seriously impact other traffic. Trust boundaries no longer require special treatment, and this allows us to move the burden of rate estimation all the way to the end-hosts. As a result, the design is robust, fully scalable, involves no configuration into separate core and peripheral regions.

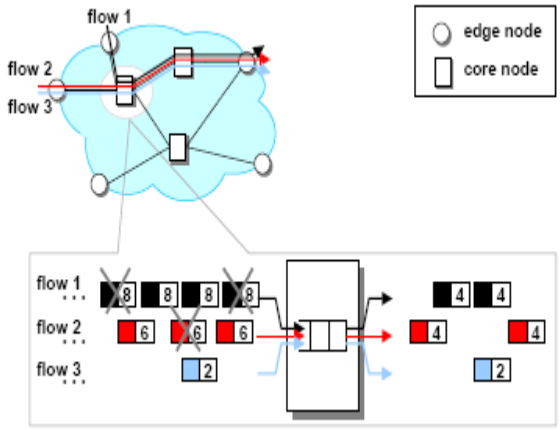

Fig2. Example illustrating the CSFQ algorithm at a core router. The crosses indicate dropped packets.

2. RELATED WORKS:

This section gives an overview of CSFQ, and discusses the impact of incorrectly labeled packets on other traffic.

A. Core-Stateless Fair Queuing (CSFQ)

CSFQ [7] and its variants [5], [6], [12] aim to achieve fair bandwidth allocation in a highly scalable fashion. To achieve this goal CSFQ identifies an *island* of routers that represents a contiguous and trusted region of network, such as an ISP domain, and distinguishes between the edge and the core of the domain. Edge routers compute per-flow rate estimates and *label* the arriving packets by inserting these estimates into each packet header. Core routers use FIFO queueing and keep no perflow state. They employ a probabilistic dropping algorithm that uses the information in the packet labels along with the router's own measurement of the aggregate traffic. Upon the arrival of a packet p, a core router enqueues the packet with probability

and drops it with probability = 1-, where *p.label* denotes the packet label, and _ represents the max-min fair rate on the output link. If all packets of flow i are labeled with the flow's rate, ri, flow i will receive on the average min(ri,α) bandwidth, which is exactly the fair bandwidth allocation of the flow. Given n flows that traverse a congested link of capacity, C, the fair rate α is defined by.

$$\sum_{i=1}^{n} \min(r_i, \alpha) = C \text{ [7].}$$

Figure 2 shows an example in which three flows with incoming rates of 8, 6, and 2, respectively, share a link of capacity 10. Note that in this case _ = 4. From Eq. (1), it follows that the forwarding probabilities of the three flows are 0.5, 0.66, and 1,

IPHV7I20034X
International Journal Of Advanced Research and Innovation -Vol.7, Issue .II
ISSN Online: 2319 – 9253
Print: 2319 – 9245

respectively. Thus, flows 1 and 2 are allocated a rate of 4, and flow 1 is allocated its incoming rate of 2. To reflect the change of flow rates due to packet dropping, the router updates the rate estimates in the packet headers of flows 1 and 2 to 4.

$$p_{enq} = \min(1, \alpha / p.label),$$

Fig3. Flow 1 is consistent, flow 2 is downward-inconsistent, and flow 3 is upward-inconsistent.

B. The Impact of Inconsistent Flows: An Example

This section discusses the impact of a downward-inconsistent flow on the consistent traffic, and the impact of a misbehaving router on the traffic in the entire domain. Consider the example in Figure 2 with the difference that the 8 Mbps flow is downward-inconsistent, i.e., its packets carry an inconsistent rate of 1 Mbps instead of 8 Mbps. As a result, the inconsistent flow will get an unfair advantage by receiving 8 Mbps, eight times more than the other two flows! Intuitively, this is because a core router cannot differentiate – based only on packet labels – between the case of an 8 Mbps inconsistent flow, and the case of 8 consistent that it generates. In contrast, in a stateful network, in which each router implements Fair Queueing, a misbehaving router can hurt *only* the flows it forwards. in which each router implements Fair Queueing, a misbehaving router can hurt *only* the flows it forwards.

C. Inconsistent and Consistent Flows

A flow whose packets carry inconsistent state is called *inconsistent*. A packet is said to carry inconsistent state if its label is different from the flow's rate. There are two types of

flows sending at 1 Mbps each. CSFQ implicitly assumes that the information carried by all packets is consistent, and, as a result, it will compute a fair rate of 1 Mbps instead of 4 Mbps, the inconsistent flow receiving 8 Mbps. a misbehaving router will affect not only the flows that traverses it, but *all* flows that share paths with the inconsistent flows inconsistent flows. If the label of a packet is larger than the actual flow rate, the flow is *upward-inconsistent*, and if the label is smaller than the flow rate, the flow is *downard-inconsistent* (see Figure 3). As we will show in the next section, of the two types of inconsistent flows, the downward-inconsistent ones are more dangerous as they can steal bandwidth from the consistent flows. In contrast, upward-inconsistent flows can only hurt themselves.

3. SELF-VERIF YING CSFQ (SV-CSFQ)

This section presents a solution that addresses the limitations of CSFQ, called *self-verifying* CSFQ (SV-CSFQ). At the heart of our solution is the router ability to *identify* and *contain* inconsistent flows. Since only downward-inconsistent flows can deny service to other flows, our focus is to identify those flows. Unless otherwise specified, in the remainder of this paper, we will refer to downward-inconsistent flows simply as *inconsistent* flows.

The ability of identifying inconsistent flows removes the need of trusting the state in the packet headers. Thus, the edge is no longer confined to the trusted boundaries. At the limit, the edge can be pushed all the way to the end-hosts. In this case, our solution no longer differentiates between edge and core routers, so it eliminates the need for router configuration. In addition, the ability to transcend trusted boundaries makes our solution highly scalable, while the flow identification and containment makes it robust.

To identify inconsistent flows, SV-CSFQ routers implement *flow verification*. Flow verification consists of three phases: (1) randomly select a flow, (2) re-estimate the flow's rate, and (3) check whether the re-estimated rate is within a predefined threshold from flow's packet labels. If yes, the flow is declared consistent; otherwise the flow is declared inconsistent. If a flow is declared inconsistent, the router starts to contain that flow. Flow containment involves relabeling the flow packets with a label that exceeds the actual flow rate. This guarantees that a contained flow will receive no more than its fair rate. Containing a flow has two immediate implications. First, the flow will no longer affect consistent traffic at downstream routers. Second, if the flow was contained by a downstream router, that router will eventually release the flow once it notices that the flow has became consistent. As a result, in steady state, only access (edge) routers will maintain state for *inconsistent* flows. To see why, consider an inconsistent flow that traverses two routers A and B, in this order. Assume that B identifies the flow first. After some time, A also identifies the flow and contains it. By containing the inconsistent flow, A transforms the inconsistent flow into a consistent one. In turn, as soon as it notices that the flow is no longer inconsistent, B will release the state of the flow. Thus, in the end, A will be the only one to contain and maintain state for the inconsistent flow. The pseudo-code of the flow verification algorithm is shown in Figure 4. Each router maintains two separate tables per output interface: one for the verified flows, called (*VerifyTable*), and one for the contained flows, called (*ContainTable*). Upon a packet arrival, the router checks whether the packet belongs to a flow that is already being verified or contained. If not, and if there is room in *VerifyTable*, the router decides to start verifying the flow with probability p. In our implementation, we set p to 0:5. In practice, every router can choose p independently. This randomness makes it very difficult, if not impossible, for an attacker to evade the verification algorithm. If the router decides to verify a flow, it creates an entry for that flow in *VerifyTable*.

If the flow is already being verified, the router checks whether the flow is consistent. If yes, the flow is removed from *VerifyTable*. If the flow is inconsistent, the flow is transferred from *VerifyTable* to *ContainTable*.4 Once a flow is contained, the router limits its service rate, and updates the labels in its packets. If a contained flow becomes consistent again, the flow

IPHV7I20034X
International Journal Of Advanced Research and Innovation -Vol.7, Issue .II
ISSN Online: 2319 – 9253
Print: 2319 – 9245

is removed from *ContainTable* after some period of time.

```
1. on receiving packet p
2.   f = get_flow_filter(p);
3.   if (f ∉ VerifyTable and f ∉ ContainTable)
4.     // if still room in VerifyTable, select f randomly
5.     if (!full(VerifyTable) and (random(0,1) < p))
6.       insert(VerifyTable, f);
7.       initialize state of flow f;
8.   else // f is either verified or contained
9.     f.rate = estimate_rate(f.rate, p);
10.    if (f ∈ VerifyTable)
11.      if (is_consistent(f))
12.        remove(VerifyTable, f);
13.      if (is_inconsistent(f))
14.        remove(VerifyTable, f);
15.        insert(ContainTable, f);
16.        initialize state of flow f;
17.    else // f ∈ ContainTable
18.      if (is_consistent(f))
19.        remove(ContainTable, f);
20.      else
21.          contain_flow(f);
22.          // make the flow consistent
23.          p.label = update_rate(f.rate, p);
24.          return;
25.   // CSFQ code executed by core routers (see [7])
26.   p_drop = max(0, 1 − α/p.label);
```

Fig. 4. Pseudo-code of SV-CSFQ.

The next two sections discuss the two building blocks of SVCSFQ: flow verification and flow containment. We emphasize that here that we discuss *one* possible realization of the pseudo code in Figure 4. We expect that the verification and containment algorithms to significantly improve over time. However, as suggested by the simulation results even in their current form, these algorithms are effective in protecting the consistent traffic.

4.CONCLUSION

While SCORE based solutions such as CSFQ [7], TUF [6], and [8] are able to provide services equivalent to the ones implemented by stateful solutions in a more scalable fashion, they still suffer from robustness and scalability limitations. In particular, the scalability is hampered because the network core cannot transcend trust boundaries (such as the ISP-ISP boundaries), and therefore high-speed routers on these boundaries must be stateful edge routers. The lack of robustness is because the malfunctioning of a single edge or core router could severely impact the performance of the entire network, by inserting inconsistent state in the packet headers.

In this paper, we have proposed an approach to overcome these limitations. To achieve scalability we push the complexity all the way to the end-hosts. To address the trust and robustness issues, all routers statistically verify whether the incoming packets carry consistent state. This approach enables routers to discover and isolate misbehaving flows and routers. The key technique we use to implement this approach is flow verification that allows the identification of packets carrying inconsistent state.

REFERENCES

[1] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Journal of Internetworking Research and Experience*, Oct. 1990, pp. 3– 26.
[2] S. Shenker R. Braden, D. Clark, "Integrated services in the Internet

architecture: An overview," June 1994, Internet RFC 1633.
[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," Dec. 1998, Internet RFC 2475.
[4] Juha Heinanen, F. Baker, W.Weiss, and J. Wroclawski, "Assured forwarding PHB group," June 1999, Internet RFC 2597.
[5] Z. Cao, Z.Wang, and E. Zegura, "Rainbow fair queueing: Fair bandwidth sharing without per-flow state," in *Proceedings of INFOCOM'99*, Tel- Aviv, Israel, Mar. 2000, pp. 922–931.
[6] A. Clerget and W. Dabbous, "Tag-based fair bandwidth sharing for responsive and unresponsive flows," in *Proceedings of INFOCOM'01*, Anchorage, AK, Apr. 2001.
[7] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks," in *Proceedings ACM SIGCOMM'98*, Vancouver, Sept. 1998, pp. 118–130.
[8] I. Stoica and H. Zhang, "Providing guaranteed services without per flow management," in *Proceedings of ACM SIGCOMM'99*, Cambridge, MA, Sept. 1999, pp. 81–94.
[9] D. Clark and J. Wroclawski, "An approach to service allocation in the Internet," July 1997, Internet Draft, http://diffserv.lcs.mit.edu/draft-clarkdiff- svc-alloc-00.txt.
[10] K. Nichols, S. Blake, F. Baker, and D. L. Black, "Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers," Dec. 1998, Internet RFC 2474.
[11] Van Jacobson and K. Poduri K. Nichols, "An expedited forwarding PHB," June 1999, Internet RFC 2598.
[12] N. Venkitaraman, J. Mysore, R. Srikant, and R. Barnes, "Stateless prioritized fair queuing," Aug. 2000, Internet Draft, draft-venkitaramandiffserv- spfq-00.txt.
[13] C. Fraleigh, S. Moon, C. Diot, B. Lyles, and F. Tobagi, "Architecture for a passive monitoring system for backbone ip networks," Oct. 2000, (submitted for publication).
[14] R. Mahajan, S. Floyd, and D. Whaterall, "Controlling high-bandwidth flows at the congested route," in *Proceedings of IEEE ICNP'01*, Riverside, CA, Nov. 2001.
[15] I. Stoica, H. Zhang, and S. Shenker, "Self-verifying CSFQ," http://www.cs.berkeley.edu/˜istoica/svcsfq-tr.pdf.

AUTHOR'S PROFILE:

[1]. PRATIMA PATIL, Pursuing M.Tech in Department of Computer Science & Engineering at Malla Reddy College Of Engineering & Technology, Hyderabad,Telangana.India.

[2].P.HARI KRISHNA , working as Asst.Professor in Department of Computer Science & Engineering, Malla Reddy College Of Engineering & Technology, Hyderabad, Telangana.India.