Identifying long-term high-bandwidth flows at a router *

Smitha, Inkoo Kim, and A. L. Narasimha Reddy

Texas A & M University, reddy@ee.tamu.edu

Abstract. In this paper, we propose a novel approach for identifying long-term high-rate flows at a router. Identifying the long-term high-rate flows allows a router to regulate flows intelligently in times of congestion. We employ a limited amount of state to record the arrivals of packets of individual flows. The state is managed as an LRU cache. The LRU cache self converges to hold only the flows that are high rate. The size of the cache need not depend on the number of flows at the router. We analyze a number of internet packet traces to show the effectiveness of this LRU-based cache. It is shown that this scheme is highly scalable since a few flows contribute a significant fraction of the traffic at a router.

1 Background & Motivation

Recently, there has been much interest in developing resource management techniques that can effectively control nonresponsive applications. It has been shown that nonresponsive applications can effectively claim most of the bandwidth at a router while starving other applications that respond to congestion [1]. This has motivated a number of recent proposals at novel buffer management techniques that allow different droprates for different flows. In order to provide different treatment to different flows at the time of congestion, it is necessary to identify flows that are significant contributors to traffic at the router. Similarly, to prevent nonresponsive applications from monopolizing the resources, it is first necessary to identify the flows that are not responding to congestion. In this paper, we propose a simple scheme to identify long-term high-rate flows and nonresponsive flows. The proposed scheme is independent of the actual resource management technique that may be applied to control the traffic.

Current Internet traffic is heterogeneous. Most of the bytes, typically, are transferred by a small number of flows (like ftp) while a large number of flows (like HTTP) do not contribute much traffic in bytes. In such an environment, flow based schemes tend to be inefficient as the work done to establish state may not be useful for most short-lived flows. RED[2] and CHOKe[4], even though increase drop rates for high bandwidth flows, do not work well as the number of high bandwidth flows increases. With the growing use of multimedia (audio & video) applications, it is expected that traffic due to unresponsive flows will increase in the future. Hence, it is important to find mechanisms that do not employ per-flow state, yet are effective in controlling several high bandwidth (unresponsive) flows at the router.

* This work was supported by an NSF Career Award.

Most schemes try to achieve fairness by estimating the number of flows that are present in the system. This is a difficult thing to do. With a great number of short lived flows in the system that are idle for longer periods than active, it is hard to estimate the number of flows accurately. There could be several short-lived flows that would require only a few KB of the bandwidth, but a few long lived flows that would need more bandwidth than their fair share. It would be inappropriate in this context to base the evaluation criterion on fairness. Achieving max-min fairness would not be possible since the demand of the flows is not known apriori.

Ideally, one would want to keep track of the long-lived flows and drop packets from these flows in situations of congestion, rather than drop packets from the short-lived flows. The rationale behind this is that dropping packets from short-lived flows may not reduce congestion as they are typically not the source of the congestion. As a result even if they reduce their rate, it would not make a significant difference to the network status. More importantly these flows may belong to HTTP traffic or Telnet like interactive traffic that are sensitive to delays. Dropping a packet from these flows would significantly affect the perceived network QOS without improving the network congestion. Whereas, dropping packets from flows that are causing the congestion to happen, viz., the high bandwidth flows, would make a great deal of difference. These high bandwidth flows may reduce their rate when they notice congestion and therefore experience less drops or they may not respond and continue experiencing high drops.

In this paper, we propose a simple method to identify high bandwidth flows at a router. The proposed method is totally decoupled from the underlying buffer management scheme in the routers i.e., it can be employed irrespective of the kind of the buffer management scheme used. The proposed method can be coupled with a resource management scheme to achieve the specific goals at the router.

2 Overview of the scheme

We consider various types of flows in this scheme viz., long-term high bandwidth flows (referred to as high-BW flows), short-lived flows, and low bandwidth flows. Flows that pump data at a rate that is greater than acceptable to the network (this is typically decided by the ISP) over a period of time are high-BW flows. Flows that pump bursts of data over a short period and stay idle for some period and continue this process are short-lived flows. The others are classified as low bandwidth flows simply because they do not violate the rate limit. Among the high-BW flows we identify two classes, one that reduces its rate when congestion is indicated and the second class that are non-responsive to congestion. Ftp applications are typical examples of responsive high-BW flows. UDP sources pumping data at high rates with no congestion control mechanism built into them can be classified as nonresponsive high-BW flows. HTTP transfers over the Internet can be classified as short-lived flows. UDP sources that send at a low rate and telnet type interactive applications can be classified as low bandwidth flows.

We propose a scheme that can be used by a router to recognize high-BW flows. Also, our scheme can distinguish between responsive and non-responsive high-BW flows.

2.1 Identifying high bandwidth flows

Packets from high-BW flows will be seen at the router more often than the other flows. Short-lived flows, characterized by HTTP transfers, are typically the ON-OFF type and send data intermittently. Thus, packets from such flows are not seen at a constant rate at the router. When they are seen, the data is much less than that of high bandwidth flows. So, by observing the arrivals of packets for a period of time, the router can distinguish between high-BW and low-BW flows.

In order to identify high-BW flows at the router, we employ an LRU (Least Recently Used) cache. This cache is of a fixed pre-determined size, 'S'. In an LRU cache every new entry is placed at the topmost (front) position in the cache. The entry that was the least recently used is at the bottom. This is chosen to be replaced when a new entry has to be added and there is not enough space in the cache. This mechanism ensures that the recently used entries remain in the cache. The objective is to store state information for only long-term high bandwidth flows in the LRU cache.

With a cache of limited size, a flow has to arrive at the router frequently enough to remain in the cache. Short-term flows or low-BW flows are likely to be replaced by other flows fairly soon. These flows do not pump packets fast enough to keep their cache entries at the top of the LRU list and hence become candidates for replacement. High-BW flows are expected to retain their entries in the LRU cache for long periods of time.

Every time the router sees a packet, it searches the cache to check if that flow's entry exists in the cache. If yes (no), we say that a *hit* (*miss*) for that flow has occurred. On a miss, the flow is added to the cache if there is space in the cache. If there is no space in the cache, it replaces the least recently seen entry. It adds this entry in the topmost position in the cache. On a hit, the router updates the position of the entry in the cache (brings it to the topmost position). The scheme employed in SRED [5] based on "zombie list" is similar to this approach, but it does not work well in the presence of many short-lived http flows.

The use of LRU can be justified by the following data obtained from a network trace. A complete distribution of packet counts across the flows in one network trace is given in Table 1. This distribution shows that most of the flows send very few packets and that most of the bytes are contributed by very few flows. This seems to suggest that limited state caches may be successful in identifying the high-BW flows.

To allow for burstiness of flows, we employ a 'threshold' below which a flow is not considered high bandwidth, even if its entry is in the cache. For each flow in the LRU, we keep track of its 'packet count' seen at the router. This count is updated on each packet arrival. Only when this count exceeds the 'threshold', a flow is regarded as a high-BW flow. Short-term flows and low-BW flows are likely to be replaced from the cache before they accumulate a count of 'threshold'.

Table 1. Packet distribution per Flow for 12NCS-953135966 Trace

n (number of packets)	number of flows having less than n packets
2	1610
5	719
10	408
20	239
50	171
100	94
200	70
300	62
400	29
500	25
700	24
1000	18

Packet sizes can be taken into account in determining the probability with which a flow is admitted into the cache. In order to keep the discussion simple, in the rest of the paper, we consider packets of the same size.

The LRU is implemented as a doubly linked list. Each node contains an entry for the flow id and the packet count. In order to make the search into the linked list easy, it is indexed by a hash table.

Cost Analysis The LRU when implemented as a doubly linked list, insertion and deletion of a flow takes O(1) time. Searching for a flow in the linked list would take linear time if it were a simple doubly linked list. A hash table is used to make the search O(1). Every time a new flow is added to the LRU, a hash table entry is made corresponding to this so that a search would take O(1) time. The memory cost is proportional to the size 'S' of the cache.

2.2 Performance measures

Typically, hit ratio is used as a performance measure of a cache. However, in our case, we need a different measure, because recording more hits is not our goal. Our goal is to identify the long term high rate flows. We ran the algorithm on the traces from NLANR, and obtained the list of flows which were identified by the LRU cache as exceeding a threshold value of hit count. We also collected the statistics about all the flows in the trace, and sorted all the flows in descending order of number of packets. We will call it 'sorted list'. We define 2 performance measures. **Measure 1:** If the cache identified m flows, ideally, the m flows should exactly match the m flows at the top of the sorted list. However, this was not always the case. We let the number of matching flows in the sorted list with m top entries be n. Then we define measure 1 to be

$$\text{measure1} = \frac{n}{m} \tag{1}$$

Measure 2: If the cache identified m flows, and not all of the m flows are in the top m of the sorted list, then some flows have escaped the filter mechanism. Assuming the least high packet count flow identified by the filter is the kth element on the sorted list, we define measure 2 to be

$$\text{measure2} = \frac{m}{k} \tag{2}$$

Based on these definitions, we present the results of employing an LRU cache in identifying the high-BW flows.

3 Trace analysis & LRU Performance

Fig. 1 shows the plots for measures 1 and 2, grouped by cache sizes and by threshold values. We can clearly see that larger caches improve the accuracy of identifying the high-BW flows for all threshold values. We can also see that small values for the threshold parameter increase inaccuracy because flows with small number of packets (but still exceeding the threshold) will be classified as high-BW flows. It is to be noted that we have used the entire length of the trace (roughly 95 seconds) as a measurement window. The identification accuracy is likely to be higher over shorter periods of time.

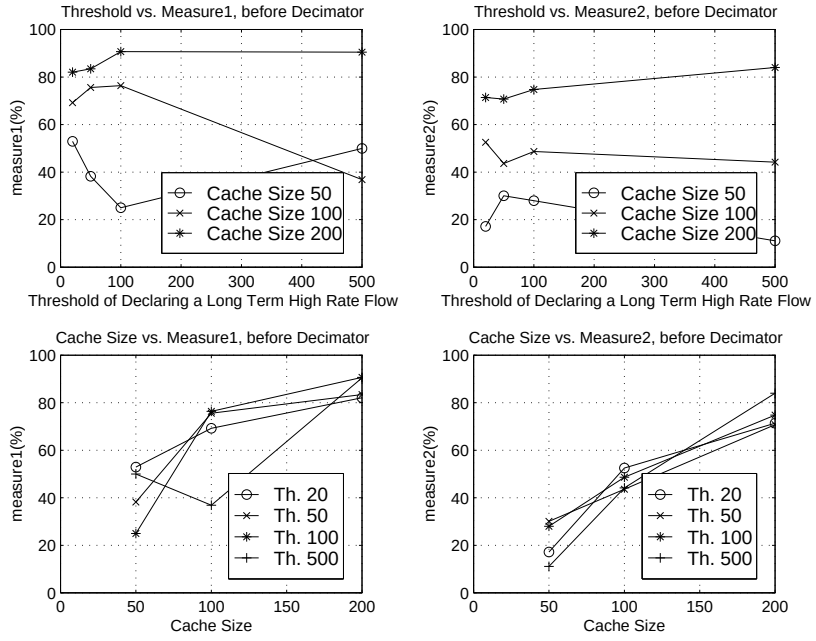

Fig. 1. Measure 1 and Measure 2 Plots for 12SDC-953145115 Trace

Table 2 shows the performance by measures 1 and 2 across four different traces. Whenever a flow's hit count reaches the threshold value, we declare the

flow 'identified', and record it in a list. The third and fourth columns in the table indicate the number of such identified flows and those that correctly belong to so many top flows. For example, with a cache size of 200 and threshold of 500, there were 42 flows that recorded 500 or more hits. Among them, 38 flows correctly belonged to the top 42 flows (as sorted by the number of packets). So by measure 1, the accuracy is 90.47% which is shown in the fifth column. The sixth column is the rank of the identified flow which sent the least number of packets. That is, among the 42 identified flows, the one with the least number of packets is found at the $50th$ position in the sorted flow list. So ideally, a total of 50 flows would have been eligible for identification by this cache, but $50 - 42 = 8$ flows escaped it. We identified 42 from the 50 flows, so measure 2 is 84.00%.

Table 2. Performance Measures for different traces, before Decimation

Trace	cache size	threshold	total identified	correct	measure1 %	eligible	measure2
Trace1	50	500	4	2	50.00	36	11.11
Trace2	50	500	8	4	50.00	31	25.80
Trace3	50	500	6	6	100.00	6.00	100.00
Trace4	50	500	14	8	57.14	24	58.33
Trace1	100	500	19	7	36.84	43	44.18
Trace2	100	500	31	25	80.64	38	81.57
Trace3	100	500	8	8	100.00	8.00	100.00
Trace4	100	500	21	19	90.47	24	87.50
Trace1	200	500	42	38	90.47	50	84.00
Trace2	200	500	40	39	97.50	42	95.23
Trace3	200	500	9	9	100.00	9.00	100.00
Trace4	200	500	23	22	95.65	24	95.83

As can be observed, the success rate depended on the cache size. At a cache size of 200, the identification is quite successful across all the traces. Since the traces consist of different mix of flows, in terms of life spans, pause periods or burstiness, or phases of flow arrivals, the performance is expected to be different across the traces. By both measures, the LRU based identification was quite successful. We discuss an improved algorithm in the next section.

4 Revised Algorithm - Random Decimator

One factor that decreases the accuracy of the above-described algorithm in holding the high-BW flows in the cache is the effect of single-packet, or short-lived flows passing through the LRU filter. For example, when a single packet flow arrives, it is put at the top of the filter. Even though there's no other packet from this flow, this entry pushes an already existing entry (possibly high-BW flow's) out of the cache. For our purpose, state information for low-BW flows is not needed, we want to filter them out. They just act as noise. In order to admit

a single packet flow into the filter (since we do not know if it's a single packet flow at the time of the first packet's arrival), we may have to drop a flow whose hit counts are several hundreds if it was somehow the least recently seen. We need to correct this problem.

The solution we adopted is putting a random decimator before the filter. A flow is admitted into the cache with a probability p. Hence, on an average, $1/p$ packets of a flow have to arrive at the router before that flow is given space in the cache. As a result, most low-BW packets do not disturb the state of the cache. By adjusting p, we can control the rate at which we allocate space in the cache. Smaller the p, the larger number of packets it takes to admit a flow. This is expected to help retain more high-BW flows in the cache.

4.1 Performance -After Random Decimator

Fig. 2 shows the performance of Trace 1 at different cache sizes and different thresholds after decimation with $p = 5\%$. The accuracy of identifying high-BW flows has improved significantly compared to the results in 1. The performance at the lower cache sizes has improved more significantly than at higher cache sizes. This is to be expected since the noisy affect of the low-BW flows on the cache state is felt more at smaller cache sizes. The results also indicate that at fairly small cache sizes, 100-200 entries, it is possible to achieve very high accuracies.

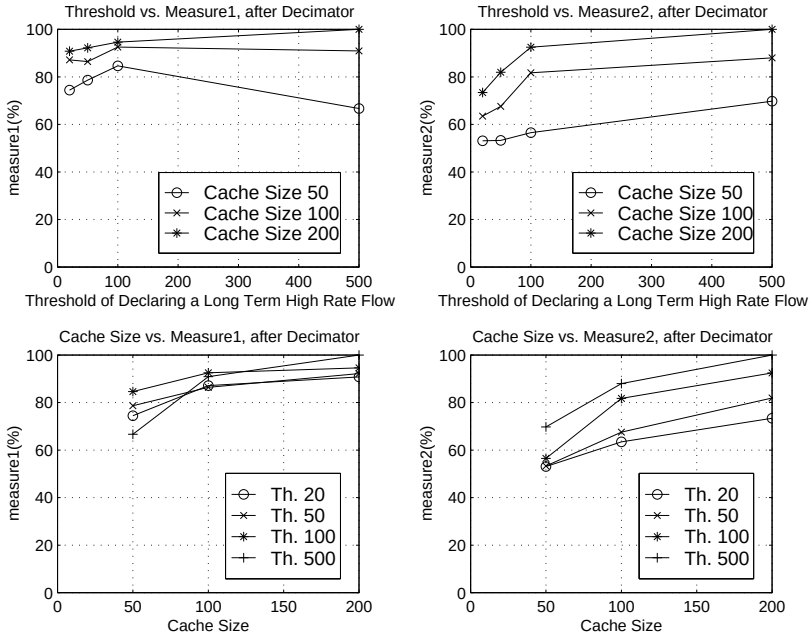

Fig. 2. Measure 1 and Measure 2 Plots for 12SDC-953145115 Trace

Table 3 show the performance of the LRU caches after decimation across four different traces, where we used p=5%. The performance has improved considerably, compared to the results before decimation. For example, with a cache size of 100 and threshold of 500, measure 2 performance for Trace 1 was 11.11% before decimation and is now 69.76% after decimation. For the same cache size and threshold, the measure 1 performance for Trace 4 increases from 57.14% to 100.00%. All the traces show similar improvement.

Table 3. Performance Measures across different traces, after Decimation

Trace	cache size	threshold	total identified	correct	measure1 %	eligible	measure2
Trace1	50	500	30	20	66.66	43	69.76
Trace2	50	500	31	26	83.87	38	81.57
Trace3	50	500	23	22	95.65	24	95.83
Trace4	50	500	8	8	100.00	8	100.00
Trace1	100	500	44	40	90.90	50	88.00
Trace2	100	500	41	40	97.56	42	97.61
Trace3	100	500	23	22	95.65	24	95.83
Trace4	100	500	8	8	100.00	8	100.00
Trace1	200	500	50	50	100.00	50	100.00
Trace2	200	500	42	42	100.00	42	100.00
Trace3	200	500	25	25	100.00	25	100.00
Trace4	200	500	8	8	100.00	8	100.00

4.2 Responsive versus nonresponsive flows

In order to demonstrate the effectiveness of the LRU policy in identifying nonresponsive flows, we carred out an ns-2 [7] based simulation experiment consisting of 20 TCP, 20 UDP and 300 HTTP flows. The cache occupancy times of different flows with a 30-entry cache are shown in Fig. 3. It is evident from figure 3 that the LRU cache was able to hold the UDP flows for a longer period (500 seconds -length of the simulation) than the TCP flows (less than a second). This data shows that nonresponsive flows are likely to be retained longer and hence can be clearly identified.

5 Scalability

To answer the scalability questions, we have looked at different traces from NLANR [8]. We collected six traces representing traffic at different times on different days. We reran our simulation with the traces to see the effectiveness of a small amount of state.

The results of caching with a 100-flow cache are shown in Table 4. The results show that a small cache of 100 entries can monitor slightly larger number of flows (189 to 327) over the simulation time since the captured flows are not active during the entire 95 second period of the trace. Even though this represents only

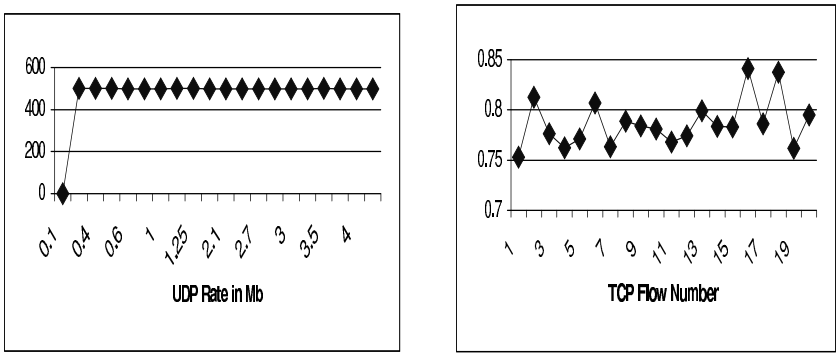

Fig. 3. Cache Occupancy

Table 4. Performance of a 100-flow cache on different traces

	Flows			Packets			Bytes		
Trace	Total Flows	Cached Flows	Hit Ratio	Total Packets	Cached Packets	Hit Ratio	Total Bytes(MB)	Cached Bytes(MB)	Hit Ratio
Trace1	4,245	327	7.70	131,044	61,794	47.16	43.58	10.22	23.45
Trace2	5,870	218	3.71	442,310	167,174	37.80	174.79	64.72	37.03
Trace3	9,059	207	2.29	458,691	240,922	52.52	185.86	104.54	56.25
Trace4	16,112	222	1.38	573,382	224,851	39.21	287.31	127.40	44.34
Trace5	21,348	257	1.20	557,001	147,179	26.42	236.19	80.71	34.17
Trace6	28,585	189	0.66	835,526	365,505	43.75	407.36	215.10	52.81

about 0.66% to 7.70% of the flows, the packet monitoring rate (or packet hit rate) is much higher, ranging from 26.42% to 52.52%. Similarly the byte monitoring rate (or byte hit rate) is much higher, ranging from 23.45% to 56.25%. This shows that the cache policy resulted in most of the higher-rate flows being captured and monitored in the cache. This is a direct result of the nature of the network trace where a few flows contribute to most of the packets and bytes transferred through the link. This heavy-tailed property has been extensively reported in the literature [10].

We ran a second experiment with Trace5 to see the impact of the cache size. The results are shown in Table 5. As the cache is varied from 50 flows to 1000 flows, the packet hit ratio in the cache goes up from 21.63% to 54.31% and the byte hit ratio goes up from 31.16% to 59.04%. This shows that with modest size caches (1000 flows), it is possible to observe and monitor a significant (about 60%) fraction of the traffic. As the hardware improves, more state may be deployed, and thus allowing greater service from this approach.

The results show that as the cache size increases, more traffic can be monitored and controlled without requiring the network elements to establish complete state for all the flows through the link.

Table 5. Impact of cache size on Trace5

	Flows			Packets			Bytes		
Cache Size	Total Flows	Cached Flows	Hit Ratio	Total Packets	Cached Packets	Hit Ratio	Total Bytes(MB)	Cached Bytes(MB)	Hit Ratio
50	21,348	120	0.56	557,001	120,491	21.63	236.19	73.60	31.16
100	21,348	257	1.20	557,001	147,179	26.42	236.19	80.71	34.17
200	21,348	557	2.61	557,001	204,740	36.76	236.19	107.53	45.53
500	21,348	1,290	6.04	557,001	248,742	44.66	236.19	119.00	50.38
1000	21,348	2,175	10.19	557,001	302,496	54.31	236.19	139.44	59.04

6 Conclusion

In this paper, we have analyzed the Internet traffic traces and proposed an efficient method to identify long-term high-rate flows. We argued that it is effective and efficient to identify a few high rate flows which can significantly contribute to congestion resolution. We proposed an LRU cache method to identify the important flows. The proposed method is self-driven and employs fixed amount of partial state that does not depend on the number of flows at the router. In order to reduce the noise from short-lived flows, we introduced a random decimator and accepted a new flow probabilistically. We simulated the algorithm on actual traces from NLANR and demonstrated that the algorithm is effective in identifying the long term high rate flows. Possible improvements to our algorithm include preferential decimation based on port/protocol information in the packet header, and decimation of short-lived flows by delay.

References

1. S.Floyd and K.Fall, *Promoting the use of end-to-end congestion control in the internet*, IEEE-ACM Transactions on Networking, pp. 458-472, August 1999.
2. S.Floyd and V.Jacobson, *Random early detection gateways for congestion avoidance*, IEEE-ACM Transactions on Networking, pp. 397-413, August 1993.
3. B. Suter, T.V. Lakshman, D. Stiliadis and A K. Choudhary, *Design Considerations for supporting TCP with per-flow queueing*, INFOCOMM'98.
4. Rong Pan, Balaji Prabhakar and Konstantinos Psounis. *CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation* in IEEE INFOCOMM, March 2000.
5. T.J. Ott, T.V.Lakshman, and L.H. Wong, *SRED: Stabilized RED* in Proceedings of IEEE INFOCOM, pp. 1346 -1355, March 1999.
6. Inkoo Kim *Analyzing Network Traces to Identify Long-Term High-Rate Flows*, Master's thesis, Technical Report, Texas A & M University, May 2001.
7. S. Floyd, *NS network simulator*, www.isi.edu/nsnam.
8. National Laboratory for Applied Network Research (NLANR), measurement and analysis team (MOAT), http://moat.nlanr.net/, Sept. 2000.
9. K. Claffy, G. Polyzos and H. W. Braun *A parameterizable methodology for Internet traffic flow profiling* in IEEE JSAC, pp. 1481-1494, March 1995.
10. M. Crovella and A. Bestavros *Self-Similarity in World Wide Web traffic: Evidence and possible causes* in IEEE/ACM Trans. on Networking, 1997.