Internet Traffic Load Balancing using Dynamic Hashing with Flow Volume

Ju-Yeon Jo[1], Yoohwan Kim[1], H. Jonathan Chao[2], and Frank Merat[1]

[1] Electrical Engineering and Computer Science Department
Case Western Reserve University, Cleveland, OH 44106

[2] Electrical and Computer Engineering Department
Polytechnic University, Brooklyn, NY 11201

ABSTRACT

Sending IP packets over multiple parallel links is in extensive use in today's Internet and its use is growing due to its scalability, reliability and cost-effectiveness. To maximize the efficiency of parallel links, load balancing is necessary among the links, but it may cause the problem of packet reordering. Since packet reordering impairs TCP performance, it is important to reduce the amount of reordering. Hashing offers a simple solution to keep the packet order by sending a flow over a unique link, but static hashing does not guarantee an even distribution of the traffic amount among the links, which could lead to packet loss under heavy load. Dynamic hashing offers some degree of load balancing but suffers from load fluctuations and excessive packet reordering. To overcome these shortcomings, we have enhanced the dynamic hashing algorithm to utilize the flow volume information in order to reassign only the appropriate flows. This new method, called dynamic hashing with flow volume (DHFV), eliminates unnecessary flow reassignments of small flows and achieves load balancing very quickly without load fluctuation by accurately predicting the amount of transferred load between the links. In this paper we provide the general framework of DHFV and address the challenges in implementing DHFV. We then introduce two algorithms of DHFV with different flow selection strategies and show their performances through simulation.

Keywords: Load balancing, Packet reordering, Multilink, Hashing, Flow analysis, Internet.

1 INTRODUCTION

Bundling multiple physical links to one logical link is very popular in today's Internet and its use is growing.[4] Packets on the logical links can be sent over any available physical sublink. This local parallelism offers scalability, reliability and cost-effectiveness. As the bandwidth demand grows, more sublinks can be added transparently without affecting the routing table. Recognizing these needs, there is commercial equipment available today to provide such functionality[2][3][11], and some proposals on logical link concept are being discussed in Internet standards community[13].

To achieve the goal of cost-effectiveness, packets must be evenly distributed over multiple links, otherwise packets may get concentrated in a small number of hot links, causing inefficient use of link capacities, long transmission delay, and possibly packet loss by buffer overflow. However, distributing packets over multiple links naturally causes out-of-order packet delivery. It has been known that such reordering results in many performance penalties and should be avoided as much as possible[4][23]. This problem of preserving the packet order while achieving load balancing is a difficult problem,[4] and has not yet been studied thoroughly enough to provide a practical solution. Nevertheless it is critical that we overcome the problem as we proceed with more parallel link deployment and advanced router development.

In this paper, we briefly survey the possible approaches and recommend a hash-based method as the most cost-effective general solution. In section 3 we take a deeper look at hash-based methods and introduce the inherent problem of load unbalancing. Then we show that the dynamic hashing is not an ideal solution and propose a new enhanced method, DHFV. We describe the algorithm in detail in section 4 and 5 and show the simulation results in section 6.

2 PACKET DISTRIBUTION METHODS

2.1 Arbitrary distribution

In this method, a packet is sent over any available link, preferably on the most lightly loaded link at the moment. It is easier to implement this method than other methods and it preserves a near-perfect load balancing, where the link queue lengths are always equalized by only one packet length difference. As long as there is any available bandwidth in any link, there is no packet loss. This method is acceptable in short-distance and slow-speed links with equal propagation delays, where the incoming packets can be retrieved in order without a special effort. But for high-speed links with different link characteristics, such as different link queue delay, different physical path, and different propagation delay, it will be a poor choice due to extreme packet reordering. So this method is not desirable in Internet.

2.2 Resequencing-based distribution

This is similar to the above method in its manner of packet distribution, but it has a provision to allow the receiver to resequence the packets in order. Typically, a sequence number is attached to each packet[6] or on a block of packets[18] or on a partial packet[1][22], so that the receiver can retrieve the sequence number and rearrange packets. In case of block, a sequence number flag is inserted between groups of packets or frames. In case of partial packet, the packets are divided into equal size cells and sent over all of the links in parallel. Since all the packets are always sent in order, although broken into smaller pieces, the receiver can reassemble the cells into a packet in order.

This method shows the near-perfect load balancing performance while keeping the packet order completely. However, there may be technical and economic barriers. Operations such as attaching and detaching serial numbers, resequencing packets, and breaking and reassembling packets, require substantial hardware support both for sender and receiver.[24] In high-speed backbone links, the implementation complexity is further increased by the speed. The difference in propagation delay time between links is significant, requiring large buffer space to compensate. Due to such costs, this method may not be as practical as a hash-based method described below.

2.3 Hash-based distribution

While the above method guarantees that no packets are reordered globally within the whole link, it is not necessary to keep the packet order between different flows. A flow is a stream of packets sent from an application at a source machine to another application at a destination machine. For the packets belonging to the same flow, there are invariant fields that can be considered a flow ID in IPv4. Hashing the flow ID results in a unique link number. This technique is commonly used for determining the flow ID[9][12][26], and distributing packets over multiple paths or servers.[10][14][27][28] Many commercial routers also utilize hashing technique for traffic distribution.[2][3][11]

Hash-based packet distribution is simple and easy to implement with a guarantee of no reordering while it does not require any processing on the receiver side. But there is a critical disadvantage for hashing, that is, a possibility of load unbalancing.[4][30][32] Although it has been implicitly assumed that the load balancing will be achieved statistically by large numbers of packets, it hasn't been proven yet. While load unbalancing is tolerable under a light traffic load, it is not any more under a heavy traffic load, especially with large number of parallel links. Despite of such a disadvantage, hash-based methods are more attractive than resequencing-based solutions due to easier implementation. With a provision for load balancing under heavy load, hash-based methods can be very practical for any traffic condition.

3 HASH-BASED PACKET DISTRIBUTION

3.1 The methods

We describe 3 hashing techniques, namely, static hashing, dynamic hashing, and dynamic hashing with flow volume. At the first step of all the methods, a hash value is generated as following.

Hash value = H (Header invariant fields) modulo k,
where H = hash function,
$k = n$ for static hashing, or $k = b$ for dynamic hashing
n = number of sublinks, b = number of bins in dynamic hashing

For H, we use CRC-32 hashing in all our simulation, which is known to have a very good performance[8][10][19], with 96-bit input as shown in Figure1.

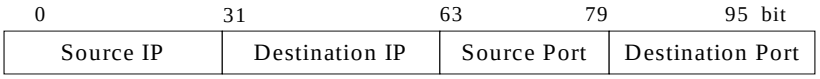

Figure 1: Hash Input Fields

Static hashing uses the hash value directly to assign a flow to a sublink. Dynamic hashing has an additional step where the hash value first determines an intermediate bin number, which in turn determines the sublink. The bin does not contain any data about packets or flows, it merely points the sublink for a packet to go. Dynamic hashing is also referred as Table-based hashing[8]. Typically there are a greater number of bins than the number of sublinks (at least more than 4 times) and a bin may be assigned to any of the sublinks. The *bin-to-sublink* assignment is not determined by hashing, but it is determined by a policy. The difference among the different dynamic hashing methods lies in what policy they adopt. For example, in Equal Cost Multipath Algorithm (ECMP)[17], the bins are equally divided among the sublinks, while in OSPF Optimized Multipath Algorithm (OSPF-OMP)[29], the bins are divided proportionally to the sublink bandwidths. Figure 2 and Figure 3 illustrates the difference between static and dynamic hashing.

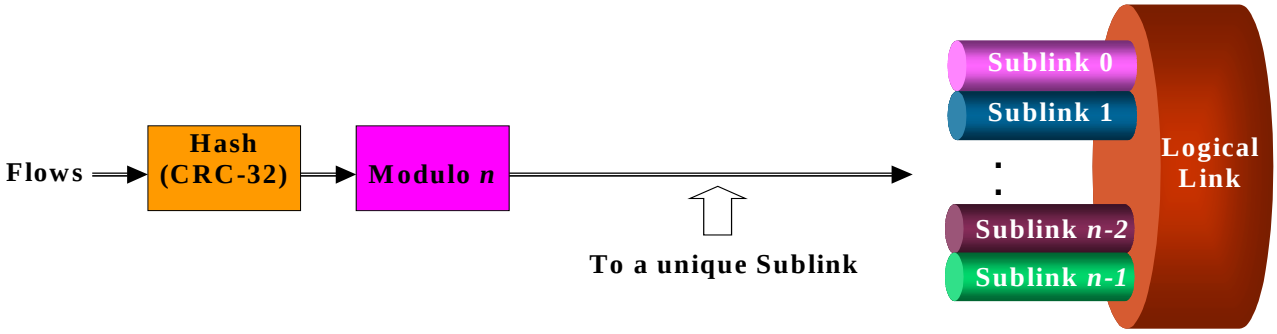

Figure 2: Static Hashing (1-Stage)

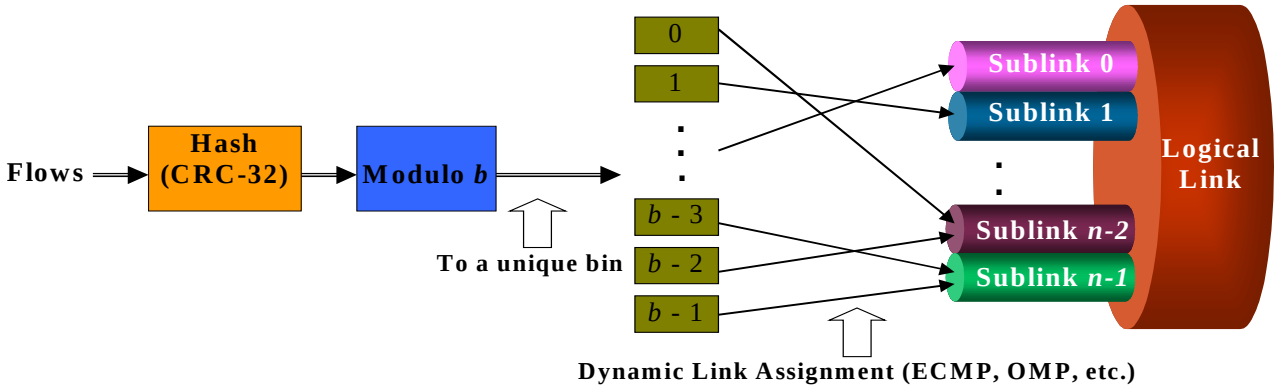

Figure 3: Dynamic Hashing (2-Stage)

Dynamic hashing offers the following benefits over static hashing because it can change the *bin-to-sublink* assignment at any time and by any policy.

- Unequal sublink bandwidth: When the sublink bandwidths are not equal, the bins can be divided proportionally to the sublink bandwidth.
- Load balancing: As the sublink status changes, bin(s) can be reassigned to another sublink.
- Fault tolerance: Upon a sublink failure, the bins for the failed sublink are simply reassigned to other sublinks.
- Scalability: Upon addition or removal of a sublink, only *bin-to-sublink* assignment is changed without changing the number *k* in the hashing procedure. The flows that were assigned to other sublinks are not affected. On the other hand, *k* should be changed in static hashing to reflect the new number of sublinks, forcing all other flows to be reassigned.

In this paper, we introduce a dynamic hashing method with an enhancement, called dynamic hashing with flow volume (DHFV), where we measure the flow volume for each bin and use the information to determine the *bin-to-sublink*

assignment more intelligently. Flow volume is the amount of traffic per unit period for a flow. With sufficiently large number of bins, a bin is likely to represent an individual flow, so each bin's unit traffic volume can be considered a pseudo-flow volume. For this reason we will use the terms *bin* and *flow* synonymously for the rest of the paper. In DHFV, bins contain the flow volume data in addition to the assigned sublink numbers but the framework of dynamic hashing is unchanged as illustrated in Figure 4.

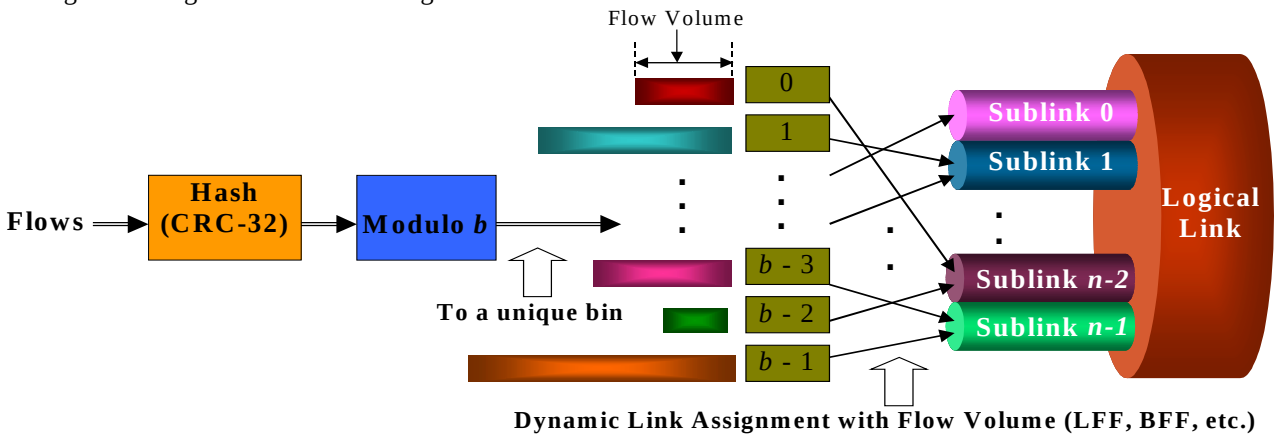

Figure 4: Dynamic Hashing with Flow Volume

3.2 Performance of static hashing

Now we examine how evenly the static hashing can distribute traffic over multiple sublinks. We have studied the Internet packet trace data that are publicly available from the NLANR project. While all of the data sets showed very similar results, the one presented in this paper[21] contains about 8.8M packets collected on an OC-12 link for 90 seconds in December 2001. Its traffic amount is about 5.2 GBytes, which is equivalent to 458 Mbps. In our simulation, when the trace data was fed to a single 500 Mbps link to simulate about 92% load, all the packets were smoothly drained without a packet loss using less than 25% of the 100-millisecond buffer. However, when the output link was divided into 8 equal bandwidth links and each flow was statistically assigned to an output link, the following result was observed.

SUBLINK NUMBER	NUMBER OF FLOWS	PERCENTAGE
Sublink 1	38,448	12.455%
Sublink 2	38,858	12.588%
Sublink 3	38,550	12.488%
Sublink 4	38,634	12.515%
Sublink 5	38,458	12.458%
Sublink 6	38,647	12.519%
Sublink 7	38,554	12.489%
Sublink 8	38,546	12.487%
Total	308,695	100%

Table 1: Flow Number Distribution by CRC-32 Hashing

SUBLINK NUMBER	NUMBER OF PACKETS	PERCENTAGE
Sublink 1	1,189,814	13.549%
Sublink 2	978,711	11.145%
Sublink 3	960,113	10.933%
Sublink 4	992,149	11.298%
Sublink 5	1,171,780	13.344%
Sublink 6	1,078,258	12.279%
Sublink 7	1,136,942	12.947%
Sublink 8	1,273,791	14.505%
Total	8,781,558	100%

Table 2: Packet Number Distribution by CRC-32 Hashing

SUBINK NUMBER	TRAFFIC VOLUME IN BYTES	PERCENTAGE OF BYTES	POTENTIAL OVERFLOW (BYTES)	POTENTIAL OVERFLOW PERCENTAGE	POTENTIAL SUBLINK UTILIZATION
Sublink 1	765,601,055	14.851%	62,476,055	1.212%	100.000%
Sublink 2	538,798,352	10.451%	0	0.000%	76.629%
Sublink 3	542,521,848	10.523%	0	0.000%	77.159%
Sublink 4	524,228,936	10.169%	0	0.000%	74.557%
Sublink 5	745,623,463	14.463%	42,498,463	0.824%	100.000%
Sublink 6	639,485,709	12.404%	0	0.000%	90.949%
Sublink 7	722,754,330	14.020%	19,629,330	0.381%	100.000%
Sublink 8	676,335,666	13.119%	0	0.000%	96.190%
Total	5,155,349,359	100%	124,603,848	2.417%	91.651%

Table 3: Traffic Volume Distribution by CRC-32 Hashing (Throughput per sublink for 90 seconds is 703 Mbytes**)**

From Table 1, we can immediately notice that hashing does an extremely good job in scattering flows. In fact, a load balancing system relying mainly on flow distribution, e.g., web servers [5], works very well with static hashing. But hashing is not so effective in distributing packets as shown in Table 2 and is even worse for distributing bytes as shown in Table 3 . We are more concerned about this byte distribution because the link load unbalancing occurs as a result of traffic amount, not by number of packets or flows.

More experiments show that hashing performs well in load balancing under a light load, but not so well at a heavy load. Usually the Internet backbone links are not that heavily loaded, but there are congested periods, and low-speed edge links are more susceptible to load fluctuations having more chance of heavy load. As problems tend to occur under heavy load, we are interested in the unbalancing problem under heavy load. As for the sublink numbers, the unbalancing is not a serious problem with a small number of links, i.e., less than 4, but as the number of links grows, the unbalancing becomes greater. As more parallel links are deployed to meet the growing bandwidth demand, the unbalancing will become more serious with pure static hashing. Installing more buffer memory on the link interface could be a temporary solution, but it may not be always possible due to hardware limitations and cost constraints.

This unbalancing occurs due to the Internet traffic characteristics where small number of large flows occupy big portion of the traffic.[15][20][31] In our trace data, only 0.1% of flows account for 53% of the traffic amount and less than 1% of flows occupied 80% as illustrated in Figure 5 and Figure 6. Unless such large flows are evenly distributed, the unbalancing will occur no matter how well all the flows are distributed. Although CRC-32 hashing was very effective, its effectiveness diminishes with much smaller flow number for the large flows.

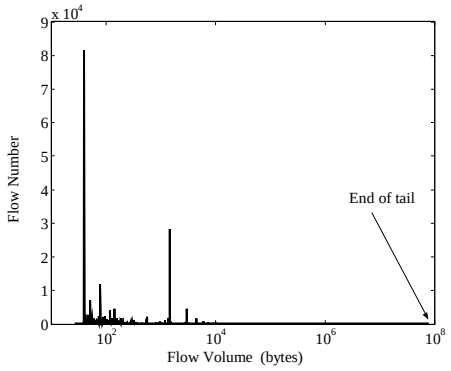

Figure 5: Flow Number Distribution by Flow Volume

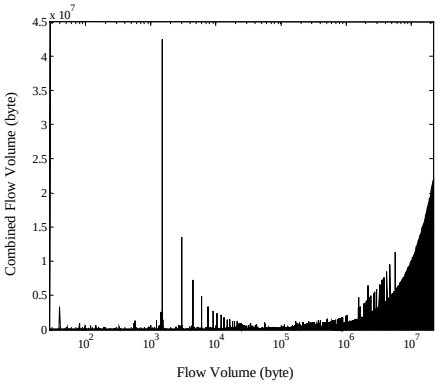

Figure 6: Combined Flow Volume Distribution

3.3 Performance of dynamic hashing

Dynamic hashing offers an intuitive load balancing solution. As the sublink load changes, we can reduce or increase the number of bins that are assigned to the sublink. However, we have two concerns: flow disruption and load fluctuation. Flow disruption occurs whenever a flow is reassigned in the middle of active transmission, which may cause packet reordering at the receiver because the buffer queue length and propagation delay on each sublink are different. Load fluctuation occurs as a result of moving too much traffic at once because the new sublink then becomes overloaded and sends the traffic back. To avoid it, the traffic must be reassigned slowly to monitor whether it creates any new overload. Because of this, there is a trade-off between adaptation time and load fluctuation. When the rate of reassignment is low, the adaptation time is long and there may be packet loss meanwhile. On the other hand, if the rate of reassignment is high, it could cause a load fluctuation between the sublinks, resulting in more reassignments. Therefore in dynamic hashing, the amount of fluctuation, convergence time, and amount of reassignment are important performance metrics as well as packet loss. Table 4 shows the simulation result of dynamic hashing from the same trace data and link environments as in static hashing. Several aspects affect the performance of dynamic hashing, such as the threshold to declare unbalancing, reassignment rate, strategies to choose the source and target sublinks, etc., so this result shows only a particular case. Nevertheless it illustrates the following points. Dynamic hashing achieves fairly good load balancing from the point of packet loss, but it causes a lot of reassignments. In the trace data, there are about 300,000 flows, so the reassignment may affect 1 to 5 % of flows. Secondly, different reassignment rates show the trade-off effect between packet loss and reassignment, suggesting that reducing both of them at the same time may be difficult.

REASSIGNMENT RATE	BUFFER SIZE	BYTES LOST	LOSS PERCENTAGE	NUMBER OF REASSIGNMENTS
High	100 ms	1,717,436	0.0333 %	14905
High	1 sec	0	0 %	2675
Low	100 ms	2,647,460	0.0514 %	8885
Low	1 sec	0	0 %	2195

Table 4: Results of Dynamic Hashing

3.4 Dynamic hashing with flow volume (DHFV)

Dynamic hashing relies only in the number of bins for load balancing. But a bin may contain no flow, many small flows or one jumbo flow, and without knowing such information, dynamic hashing must blindly choose the bins to reassign. If the combined traffic amount removed is too small, there is still overload and packet loss. On the other hand, if the combined traffic amount is too much, there is load fluctuation. Besides, small flows that do not cause the overload and would not contribute to the load balancing after reassignment are reassigned unnecessarily. With the flow volume information in DHFV, we can precisely select the flows that should be reassigned. Generally we are interested in reassigning large flows because

- It is much more effective to move one large flow than many small flows for load transfer.
- Moving the large flows immediately relieves the overload, and prevents packet loss.
- It reduces the number of overall reassignments
- It promotes fairness by punishing only the flows that caused overload.
- It is easier to detect large flows than small flows because large flows tends to sustain high bandwidth for long time

For DHFV method to be practical, several questions must be answered.

- There are a lot of flows in the logical link, how do we keep track of the large number of flows?
- Internet traffic is bursty, so the flow volume measured in one period may not be related with the next period. How strongly are they related?
- Which and how many flows do we reassign?
- How do we determine the duration of measurement unit period and bin numbers?
- Is it too complex to implement?
- Can it be done in real-time while the router is operating?
- And finally, does the new method perform better than conventional dynamic hashing?

In the rest of the paper, we address the above issues.

4 COLLECTING THE FLOW VOLUME

4.1 Flow identification by hashing

To measure traffic characteristics of the flow, first we need to separate the flows in a practical manner. Flow ID described in Figure 1 is very large (96 bits), so it cannot be used as is. To reduce it to a manageable size, hashing is used to convert the flow ID over a smaller range. In the case of CRC-32, the hash value is still 32 bits long, so it is reduced to smaller size bins, e.g., 1024 or 4096, by modulo operation. The result is the assigned bin number, which is a pseudo-flow ID.

It is true that there are millions of flows in a high-speed link, so how do we keep track of each flow? First, it is the fact that only a small portion of them is active momentarily[28] because a lot of them are short-lived, dormant or even single-packeted. Since we need to keep track of flows only during the short unit period, which is less than 1 second, the actual number of flows that we need to track is greatly reduced. In our experiment, there are 308,695 flows during 90 seconds, but there are typically about 2,000 active flows during any measurement unit period. Secondly, even if flows are combined into a bin after hashing, the performance is not affected in terms of load balancing, although it may increases the number of flows that are affected by reordering. With sufficiently large bin numbers, the collision rate can be very low. In our experiments, 4096 bins on OC-12 were more than adequate for acceptable flow separation.

4.2 Temporal locality of flow volume

Can the flow volume measured in one period be used for predicting next period flow volume? It is reported that Internet traffic has some temporal locality[20][31]. We further show that such locality is reliable enough for using in DHFV. Figure 7 shows a typical scatter chart showing the correlation between the flow volumes in one period (x-axis) to next period (y-axis). It was produced from the OC-12 trace data by dividing flows into 4096 bins and measuring the traffic volume in every 50 milliseconds. The dots in the region labeled *starting* indicate flows that are in transit to high bandwidth traffic from low bandwidth traffic, and the region labeled *ending* indicates the opposite. The area with our greatest interest is *sustained* region, where there is a great correlation between one period and the next period. As shown in Figure 7, the correlation is typically very high in high flow volume region. Although the correlation in small flow volume region is very low, it is not our concern since we are interested only in large flows. To enhance the predictability of large flows, two continuous periods may be monitored. By selecting only those flows that kept high flow volumes continuously, we can effectively eliminate the bursty flows.

The number of bins and length of measurement period affect the correlation. As measurement period gets longer, the correlation gets better because the fluctuations are removed and flow volumes approach their average value. In OC-12 link, a measurement period of as short as 50 milliseconds produced a meaningful correlation as in Figure 7. The correlation gets better too as the bin number is increased since each bin can represent each flow more accurately, but once the flows are well separated the correlation remains same. In our experiment, bin number of 4096 and 8192 produced almost the same correlation.

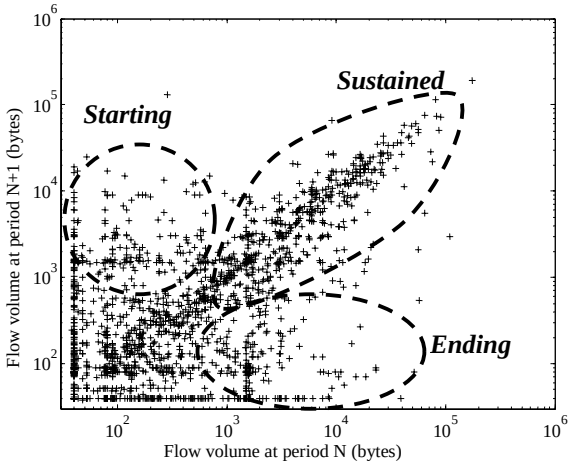

Figure 7: Flow Volume Correlation with 4096 bins and 50ms Measurement Unit Period

4.3 Updating the flow volume

Each bin is associated with a variable that keeps the flow volume. When a packet arrives, its destination bin number is determined first and the corresponding flow volume is incremented by the packet size that is available from the packet header. At the end of each measurement unit period, all the variables contain the flow volume for each bin that has arrived during the period. After the flow volumes are used for the load-balancing algorithm, the variables are reset and the updating process is repeated.

5 THE DHFV ALGORITHMS

As the conventional dynamic hashing have different methods depending on the *bin-to-sublink* assignment policy, DHFV may have several variations. In this paper, we consider two variations, Best-fit Flow First (BFF) algorithm and Largest Flow First (LFF) algorithm. Both of them share these policies.

- At the end of each period, the queue lengths for all the sublinks are measured to detect overloaded sublink(s). If any queue length exceeds a threshold, the dynamic hashing algorithm is triggered.
- Target sublink is selected among the lightly loaded sublinks, i.e., the sublinks with queue length under the threshold. If there is no such sublink, no action is taken.

- Flows from an overloaded link are reassigned to a single target sublink, not being distributed over multiple target sublinks.
- When there are multiple overloaded sublinks, each overloaded sublink will be assigned to a target sublink in the order of the degree of overload, that is, most heavily loaded sublink to most lightly loaded, 2nd heavy to 2nd light, and so on.
- For the sake of simplicity, we assume the sublink bandwidths are equal.

5.1 Best-fit Flow First (BFF)

In BFF, the flows are reassigned from the source to the target sublink so that the reassigned flow volumes fit best into the calculated traffic amount with minimum number of flows. The traffic amount to be moved is calculated according to the difference in queue lengths between the source and target sublinks. By that way, only the amount of traffic that can be accommodated by the target sublink is moved and load fluctuation can be avoided.

If an overload is detected in any of the sublinks, these actions are taken.

1. BFF searches for target sublink(s) to move the traffic load.
2. If all other sublinks are overloaded, do nothing; otherwise determine the target sublink(s).
3. At the same time, the flow volumes assigned to the source sublink are sorted in descending order.
4. The flows to reassign are chosen from the source sublink(s) to meet the criteria described below.
5. Change the *bin-to-sublink* assignment for the selected flows.

To describe how to choose the flows, we define following terms.

- Duration of measurement unit period = d
- Flow volumes = $f_1, f_2, .., f_b$ in descending order
- Sublink queue length for source sublink= q_{max}
- Sublink queue length for target sublink= q_{min}
- R = movement rate ($0 < R \leq 1$)
- Traffic amount to be moved, $M = (q_{max} - q_{min}) * R$

The flows volumes are tested if it fits into M from the largest one. If the largest flow volume is greater than M, then the second largest one is tested, and so on. After the largest flow volume f_i that is smaller than M is found, M is decreases by f_i , and second best-fit flow volume f_j is searched in a similar manner. The process is repeated until M becomes 0 or no non-zero flow volume is smaller than current M. BFF procedure is illustrated in Figure 8. At the end, we have the set of flows to reassign F of which sum of flow volumes are as close as possible to original M. That is,

$$F = \{f_i, f_j, \ldots.\} \text{ such that } \sum_{f_k \in F} f_k \approx M$$

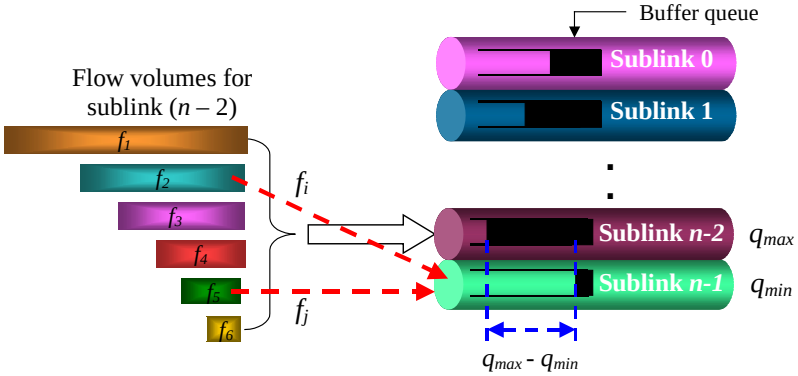

Figure 8: BFF Algorithm

By moving the traffic amount equal to M, it is likely that the queue length in the source link is decreased by M at the end of next period, if the traffic pattern remains same. And throughout the subsequent periods, the queue length is decreased further until the queue length becomes normal, reflecting only the temporary fluctuations rather than the overloading flows. To make the load balancing faster, the movement rate R may be increased either statically or dynamically. For example, R may be greater when q_{max} is closer to the buffer limit.

BFF guarantees that the minimum numbers of flows are reassigned to satisfy the need, thus preventing unnecessary reassignments as in conventional dynamic hashing. In BFF, flow volumes are sorted and the best-fit flow volumes are linearly searched, so the algorithmic complexity is *O (b * log b)* for sorting *O (b)* for linear search in worst case. Due to this overhead, BFF may not be suitable for real-time operation if the bin number *b* is large. This can be resolved by allowing a gap between each measurement periods so that sorting and selection can be performed during the gap time.

5.2 Largest Flow First (LFF)

Unlike BFF, LFF does not consider the absolute size of flow volume, but only considers the ranking of flow volume. LFF simply reassigns the flow with the largest flow volume from the overloaded sublink to the target sublink. In Figure 8, it would be always f_1. The largest flow can be determined in the following way.

1. For each sublink, there is a *MAX_VOLUME* variable that keeps the current largest flow volume, and a *MAX_BIN* that keeps the corresponding bin number.
2. When a packet arrives, LFF updates the associated flow volume and compares it with the *MAX_VOLUME*. If the new flow volume is greater than *MAX_VOLUME*, LFF updates *MAX_VOLUME* and *MAX_BIN* with the new one.
3. At the end of the period, *MAX_BIN* contains the bin number for the largest flow for the sublink.

If the largest flow alone is not large enough for proper load balancing, the queue length in source sublink does not get reduced. Then the LFF algorithm is repeated until the load balancing is done with the new largest flows at the subsequent periods.

To further enhance the predictability, we can use two periods to detect a flow with sustaining high bandwidth. In that case, the flow volumes for two periods are kept. The new flow volume is compared with the current *MAX_VOLUME*, only if its past period flow volume was large enough, e.g., the past flow volume was greater than 25% of past *MAX_VOLUME*. Otherwise the flow is considered a bursty flow and is not allowed to update the current *MAX_VOLUME*. LFF algorithm can be extended to reassign multiple largest flows instead of single largest flow simply by keeping multiple entries of *MAX_VOLUME* and *MAX_BIN*.

LFF is simpler than BFF because it does not require sorting or the complex flow selection process. LFF does not need the gap time between the measurement periods since the largest flow information is readily available at the end of each period.

6 SIMULATION RESULTS

The simulation was performed to compare the performance of the load balancing schemes under the same conditions as in section 3.2. The trace data was fed to a 500 Mbps logical link, one with 8 equally divided sublinks (Table 5) and another with 16 equally divided sublinks (Table 6). The buffer space of the sublinks was for storing 100 ms of traffic. The bin number for dynamic hashing was 4096 and flow volume was collected every 50 milliseconds. Dynamic hashing algorithms were activated when the buffer queue exceeded 50% of the buffer space. For BFF, *R* was 0.5, meaning that half of the difference in maximum and minimum queue length was transferred. Both BFF and LFF performed much better than conventional dynamic hashing with less packet loss and without many reassignments, indicating that they reassigned the overloading traffic quickly enough when the overload occurs. The actual instances of packet reordering may be different from the number of reassignments. In case of dynamic hashing, the amount of actual packet reordering should be less than the number of reassignments because some of the reassigned bins may not have an active flow, whereas the reassignment always results in reordering in BFF and LFF.

METHOD	LOST BYTES	PACKET OF LOST BYTES	NUMBER OF REASSIGNMENTS	PERCENT OF REASSIGNMENTS
Static Hashing	144,729,419	2.8074 %	0	0 %
Dynamic Hashing	2,647,460	0.0514 %	8,885	2.878 %
DHFV - BFF	46,436	0.0009 %	947	0.307 %
DHFV - LFF	0	0 %	250	0.081 %

Table 5: Results with 8 Sublinks

METHOD	LOST BYTES	PERCENT OF LOST BYTES	NUMBER OF REASSIGNMENTS	PERCENT OF REASSIGNMENTS
Static Hashing	270,855,666	5.2539 %	0	0 %
Dynamic Hashing	16,463,619	0.3194 %	35,690	11.5616 %
DHFV - BFF	8,147,904	0.1580 %	2,755	0.8925 %
DHFV - LFF	1,975,532	0.0383 %	1,565	0.5070 %

Table 6: Results with 16 Sublinks

Between the two methods of DHFV, LFF method shows better performance than BFF with less reassignments and packet loss. This is because;

1. LFF has better predictability for next period because high-bandwidth flows tend to be more stable with less fluctuation than lower bandwidth flows.
2. Low bandwidth flows tend to last shorter, so the load is not actually transferred to the target links in BFF
3. In BFF, big flows may escape the bin selection process when its flow volume is bigger than the movement amount *M*. It results in continuous growth of sublink queue and packet loss, and leads to more reassignments in subsequent periods.
4. In LFF, two period's measurements are used instead of one period in BFF, so the predictability was higher.
5. In BFF, multiple small flows are selected to make up for the movement size, increasing the number of reordering. In LFF, only one flow that has the highest impact is moved, thus reducing reassignment.

As a result, LFF can select the desired flows more accurately, and achieves quicker adaptation. Table 6 shows the performance for 16 sublinks. As the sublink number is increased, the unbalancing becomes stronger. So all the methods are negatively affected. Still the relative performance remains the same. The static hashing performs the worst, and the LFF shows the best performance of all, keeping the packet loss within an acceptable range (< 0.1%). The actual performance may vary depending on the parameter settings, and the traffic pattern, but we have observed that the relative performance is always preserved.

7 DISCUSSIONS

7.1 Packet Reordering

As we apply load-balancing mechanisms to the static hashing, packet reordering occurs. So we must choose either packet loss by doing static hashing or reordering by doing dynamic hashing under heavy load. Packet reordering is not new in the Internet and the effect of reordering is well studied[4][7][23], but we are not aware of any study comparing the impact of packet loss and the impact of reordering. Although it needs further study to determine which is preferable for the Internet traffic, our preliminary study shows that reordering has a less damage for TCP performance, specifically Reno TCP with SACK option, a currently popular TCP implementation. In addition, efforts are being made such as DSACK option to keep the TCP performance high in the presence of reordering [4][7][16]. The improvements in TCP will make packet reordering more preferable to packet loss because the receiver has at least a chance to resequence the received packets in case of reordering while only an expensive retransmission can recover the packet loss.

Still some types of penalty must be paid once packet reordering occurs, such as increased delay, larger buffer space, more host processing, etc., so reducing the amount of reordering is still important. Our proposed DHFV scheme reduces the reordering to an acceptable level. It reduces the number of reordering instances in the order of magnitude from the conventional dynamic hashing. Furthermore DHFV limits the reordering only to data traffic and does not affect low-bandwidth control traffic that has greater consequences by reordering. According to Bennett et al.[4], there are several disadvantages of reordered packets to TCP and some of them are caused by reordered ACKs. In DHFV, the reordering is limited to sustained bulk data transfer, so the problem of reordered ACK is avoided.

7.2 Fairness

In conventional dynamic hashing, many small flows are reassigned. This is not only ineffective because the small flows do not relieve the overload, but more importantly, they are reassigned unfairly because of other high-volume flows on the same sublink. It is totally unnecessary to reassign them, and DHFV method successfully prevents such small flows

from being reassigned, thus limiting the reordering only to those flows that created unbalancing. So DHFV is considered fairer than dynamic hashing.

7.3 Implementation complexity

Compared with static hashing, dynamic hashing requires more processing such as

- Additional memory space for keeping *bin-to-sublink* information
- Additional memory access to read *bin-to-sublink* information before sending a packet to a sublink
- Monitoring the sublink queue lengths for overload detection
- Executing the load balancing policies

In addition to the above dynamic hashing, enabling flow volume measurement requires additional processing such as

- Additional memory space for keeping the flow volumes
- Reading the size in packet header and adding it to the corresponding flow volume
- Comparing the flow volumes and updating the largest flow information (LFF)
- Sorting flow volumes (BFF) for source sublink

Obviously LFF is preferred because BFF method requires an expensive sorting process. We believe that LFF algorithm can be implemented with a reasonable effort by modifying the existing implementation of conventional dynamic hashing.

7.4 Related work

Cao et al. discuss the load balancing performances by different hashing methods[8], although the aspect of reordering was not considered. It shows a result of good load balancing over 10 links, however the trace period was rather short (14 sec) to detect the actual packet numbers per flow, so the result is somewhat biased by the low number of packets per flow.

Shaikh et al. suggest rerouting long-lived flows at the routing level.[25] This method has several different perspectives from ours. It handles the flows at coarse grain, on the order of minutes vs. the order of milliseconds in our case. The feedback by routing or congestion control mechanism is too late to prevent packet loss resulting from buffer overflow. In short, we are more interested in identifying the high-bandwidth flows within a few measurement periods rather than identifying absolutely long-lived flows over long periods, although they may be related. Secondly, at this coarse grain, the reordering is not a primary concern.

The load-balancing problem arises in parallel routing architectures, too. Wang et al.[30] use hash-based assignment with destination IP to assign a packet to an available routing agent. Uneven load balancing was observed in packet numbers, which was corrected by reassigning flows. The flows have difference priority for reassignments. Although TCP flows have lower priority for reassignment than UDP flows, the effect may be limited because TCP accounts for the majority of Internet traffic. So distinguishing flows based on the flow volume rather than flow types is considered more effective.

8 CONCLUSIONS

A logical link composed of multiple parallel links plays an important role in today's Internet and hashing is promising for packet distribution without packet reordering. However hashing may suffer from load unbalancing and packet loss under heavy load. Dynamic hashing offers a simple solution but may cause massive reordering. We have enhanced dynamic hashing to utilize flow volume information, where only largest flows are reassigned. The new method achieves very good load balancing while creating very low number of reordering instances. This method is practical and can be implemented with a small modification of dynamic hashing.

REFERENCES

[1] Hari Adiseshu, Guru Parulkar and George Varghese, "A Reliable and Scalable Striping Protocol," In *Proceedings of ACM SIGCOMM*, pp. 131-141, August 1996.

[2] Vadim Antonov and David Bernstein, "The Pluris Massively Parallel Router (MPR)," *Gigabit Networking Workshop*, 1998.

[3] Avici White Paper, "The Use of Composite Links to Facilitate the Creation of a New Optical Adaptation Layer," http://www.avici.com.

[4] Jon C. R. Bennett, Craig Partridge, and Nicholas Shectman, "Packet Reordering is Not Pathological Network Behavior," *IEEE/ACM Transactions on Networking*, vol. 7, No. 6, pp. 789-798, December 1999.

[5] Azer Bestavros, Mark Crovella, Jun Liu, and David Martin, “Distributed Packet Rewriting and its Application to Scalable Server Architectures,” In *Proceedings of the 1998 International Conference on Network Protocols (INCP ’98)*, Oct., 1998.
[6] Uyless Black, *PPP and L2TP: Remote Access Communications*, Prentice-Hall, Inc. Upper Saddle River, NJ 2000.
[7] Ethan Blanton and Mark Allman, “On Making TCP More Robust to Packet Reordering,” *ACM Computer Communication Review*, 32(1), January 2002.
[8] Zhiruo Cao, Zheng Wnag, and Ellen Zegura, "Performance of Hashing-Based Schemes for Internet Load Balancing," In *Proceedings of IEEE Infocom*, pp. 332-341, 2000.
[9] Zhiruo Cao and Zheng Wnag, "Flow Identification for Supporting Per-Flow Queueing," In *Proceedings of 9th International Conference on Computer Communications & Networks*, pp. 88-93, 2000.
[10] Girish Chandranmenon and George Varghese, “Trading Packet Headers for Packet Processing,” *IEEE/ACM Transactions on Networking*, vol. 4, No. 2, pp. 141-152, 1996.
[11] Cisco White Paper, “Load Balancing with Cisco Express Forwarding,” http://www.cisco.com.
[12] K. C. Claffy, Hans-Werner Braun, and George C. Polyzos, “A Parameterizable Methodology for Internet Traffic Flow Profiling,” *IEEE Journal on Selected Areas in Communications*, vol. 13, No. 8, pp. 1481-1494, 1995.
[13] R. Damle, Y. Lee, E. Brendel, R. Hartani, and V. Sharma, “Optical Channel Concatenation – Need and Requirements,” *Internet Draft, draft-damle-optical-channel-concatenation-00.txt*, June 2001.
[14] N. Duffield and M. Grossglauser, “Trajectory Sampling for Direct Traffic Observation,” *IEEE/ACM Trans. on Networking*, June 2001.
[15] Anja Feldmann, Jennifer Rexford, and Ramon Caceres, “Efficient policies for carrying Web traffic over flow-switched networks,” *IEEE/ACM Transactions*, Vol. 6, No 6, pp. 673-685, Dec. 1998.
[16] Sally Floyd, "A Report on Recent Developments in TCP Congestion Control," *IEEE Communications Magazine*, vol. 39, no. 4, Apr. 2001.
[17] Christian Hopps, “Analysis of an Equal-Cost Multi-Path Algorithm,” *Internet Draft, draft-hopps-ecmp-algo-analysis-04.txt*, Feb. 2000.
[18] IEEE Standard 802.3ad-2000, Link Aggregation (CSMA/CD: ETHERNET)
[19] Raj Jain, "A Comparison of Hashing Schemes for Address Lookup in Computer Networks," *IEEE Transactions on Communication*, vol. 40, pp. 1570-1573, Oct. 1992.
[20] Ratul Mahajan, Sally Floyd, and David Wetherall, “Controlling High-Bandwidth Flows at the Congested Router,” In Proceedings of International Conf. On Network Protocols (ICNP), November 2001.
[21] NATIONAL LABORATORY FOR APPLIED NETWORK RESEARCH (NLANR). Network traffic packet header traces. http://moat.nlanr.net/Traces/Traces/daily/20011204/IND-1007501453-1.tsh.
[22] Craig Partridge and Walter Milliken, “Method and Apparatus for Multiplexing Bytes Over Parallel Communications Links Using Data Slices”, *US Patent 6,160,819*, Dec. 12, 2000.
[23] Vern Paxson, “End-to-End Internet Packet Dynamics”, *IEEE/ACM Transactions on Networking*, vol. 7, No. 3, pp. 277-292, June 1999.
[24] N. T. Plotkin and P. P. Varaiya, "Performance Analysis of Parallel ATM Connections for Gigabit Speed Applications," In *Proceedings of IEEE Infocom*, pp. 1186-1193, March/April 1993.
[25] Anees Shaikh, Jennifer Rexford, and Kang Shin, "Load-sensitive routing of long-lived IP flows," In *Proceedings of ACM SIGCOMM*, September 1999, pp. 215-226.
[26] Alex Snoeren, Craig Partridge, Luis Sanchez, Christine Jones, Fabrice Tchakountio, Stephen Kent, and W. Timothy Strayer, "Hash-Based IP Traceback," In *Proceedings of ACM SIGCOMM*, pp. 3-14, Aug., 2001.
[27] Ion Stoica and Hui Zhang, "LIRA: An Approach for Service Differentiation in the Internet," In *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp. 115-128, Jul. 1998.
[28] Bernhard Suter, T. V. Lakshman, Dimitrios Stiliadis, and Abhijit K. Choudhury, "Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-Flow Queueing," *IEEE Journal on Selected Areas in Communications*, vol. 17, No. 6, pp. 1159-1169, June 1999.
[29] Curtis Villamizar, "OSPF Optimized Multipath (OSPF-OMP)," *IETF Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-ospf-omp-01.txt*, Oct. 1998.
[30] Jun Wang and Klara Nahrstedt, “Parallel IP Packet Forwarding for Tomorrow’s IP Routers”, In *Proceedings of IEEE Workshop on High Performance Switching & Routing*, pp. 353-357, 2001.
[31] Carey Williamson, “Internet Traffic Measurement,” *IEEE Internet Computing*, Nov./Dec. 2001, pp. 70-74.
[32] Kun-Lung Wu and Philip Yu, “Load Balancing and Hot Spot Relief for Hash Routing among a Collection of Proxy Caches,” In *Proceedings of 19th International Conference on Distributed Computing Systems*, pp. 536-543, 1999.