# Traffic Distribution over Equal-Cost-Multi-Paths

Tat Wing Chim and Kwan L. Yeung
Dept. of Electrical and Electronic Engineering
The University of Hong Kong
Pokfulam Road, Hong Kong
E-mail: {twchim, kyeung}@eee.hku.hk

*Abstract*—**To effectively manage the traffic distribution inside a network, traffic splitting is needed for load sharing over a set of equal-cost-multi-paths (ECMPs). In this paper, a new traffic splitting algorithm, called Table-based Hashing with Reassignments (THR), is proposed. Based on the load sharing statistics collected, THR selectively reassigns some active flows from the over-utilized paths to under-utilized paths. The reassignment process takes place in such a way that the packet out-of-order problem is minimized. As compared with the existing traffic splitting algorithms, THR provides close-to-optimal load balancing performance, less than 2% of packets arrived out-of-order, and a very small end-to-end packet delay performance. Although additional traffic monitoring function is needed by THR, we show that the extra complexity incurred is marginal.**

***Keywords-Traffic splitting; ECMPs; Packet Reordering.***

## I. Introduction

The main objective of traffic engineering is to reduce congestion hotspots and improve resource utilization across the network by carefully managing the traffic distribution inside a network. In today's Internet, IP routing is destination-based and forwarding paths are calculated based on shortest-path-first algorithm. Though IP routing is scalable, it lacks the capability of explicitly controlling the traffic distribution inside a network.

To achieve this, two models can be followed, *overlay model* and *peer model*. With overlay model, a fully-meshed logical topology based on, e.g. MPLS LSPs [1], can be constructed over the network's physical topology. Traffic can then be explicitly routed using logical paths. This model is known to suffer from the N-square problem [2].

Using the peer model [3][4], balanced traffic distribution can be achieved by manipulating the link weights in the Open Shortest Path First (OSPF) routing protocol. Once the link weights are properly set, the network can operate as it does today: the OSPF protocol calculates the forwarding paths based on the shortest-path-first computation. Compared with the overlay model, peer model is more scalable, does not require any changes to the basic IP routing architecture, and can be readily deployed.

For traffic engineering, the problem of finding the optimal paths for traffic demands is usually solved by formulating it as a linear programming problem [5]. The optimal solution usually requires splitting the traffic demand between two nodes over multiple paths. Besides the extra processing overheads, traffic splitting must be done carefully so that the packets from the same flow are not sent over different paths. Otherwise, different delays may cause packet reordering in TCP flows. This in turn degrades the TCP performance. Since the peer model relies on the basic IP routing architecture, the problems caused by traffic splitting tend to be more pronounced. Under OSPF routing protocol, if there are more than one equal-cost path (i.e. equal-cost-multi-paths, or ECMPs) towards a destination, traffic going to that destination will be distributed among all ECMPs for load balancing. However, the detailed traffic splitting mechanisms are subject to individual implementations.

A number of traffic splitting algorithms have been proposed. The simplest approach is to split the traffic over ECMPs in a packet-by-packet round robin fashion [6]. We call it *Packet-by-Packet* (PBP) splitting. With PBP, the traffic load can be perfectly balanced among all ECMPs but serious packet out-of-order problem will be induced as the delays on different ECMPs are different. To address this problem, per-flow forwarding is usually adopted at the expenses of less optimal load balancing. Three per-flow traffic splitting algorithms have been proposed, Direct Hashing, Table-based Hashing [6], and Fast Switching [7]. Please refer to the next section for details.

In this paper, we propose a new traffic splitting algorithm, called *Table-based Hashing with Reassignments* (THR). Using THR, the actual load sharing statistics among all ECMPs are monitored. This can be easily implemented by incrementing an associated counter at each ECMP upon a packet departure. For every pre-determined time interval (which is adjustable), the counter values are collected and compared to identify the most-utilized as well as the least-utilized ECMPs. Then we can select a flow from the most-utilized-path and reassign it to the least-utilized-path. The reassignment process is carefully designed so that the packet out-of-order events can be minimized. Inherent to its design, THR algorithm is capable of handling both equal and unequal traffic splitting. Our simulation results show that THR gives a near-optimal load balancing performance as that of Packet-by-Packet (PBP) algorithm, whereas the packet out-of-order events induced by THR are 96.6% less than PBP. Because of better splitting performance, THR also gives a lower average packet delay performance than all other traffic splitting algorithms.

This paper is organized as follows. In the next section, three existing flow-based traffic splitting algorithms are reviewed. In Section III, our feedback-based traffic splitting algorithm THR is introduced. In Section IV, THR is compared with other splitting algorithms via simulations. Finally, we conclude the paper in Section V.

## II. Existing Traffic Splitting Algorithms

Since packet-by-packet splitting introduces serious packet out-of-order problem, per-flow forwarding is usually adopted but at the expenses of less optimal load balancing. Up to our knowledge, three per-flow traffic splitting algorithms have been proposed, Direct Hashing, Table-based Hashing [6], and Fast Switching [7].

The first two are hash-based algorithms. A hashing value is obtained from a hashing function $H$, or,

Hash value = $H$ (Header invariant fields) modulo $x$.

The input to the hashing function is usually the four packet header fields, source IP address, destination IP address, source port number and destination port number. In Direct Hashing (Fig. 1), $x$ is the number of ECMPs $k$. For Table-based Hashing (Fig. 2), $x$ is replaced by the number of bins $b$. Note that a bin is an intermediate pointer that points to a selected ECMP. Direct Hashing uses the hash value directly to assign a flow to an ECMP, whereas Table-based Hashing uses the hash value to determine an intermediate bin, which in turn points to a selected ECMP for the flow. As compared with the packet-by-packet algorithm, the packet out-of-order problem becomes less serious. But a poorer load balancing performance is also expected.

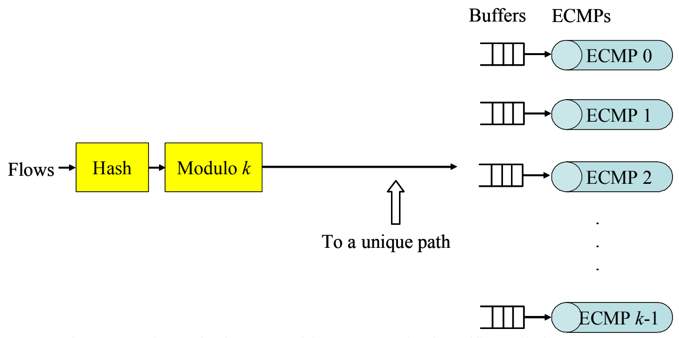

Fig. 1 Design of Direct Hashing at a typical traffic splitting router

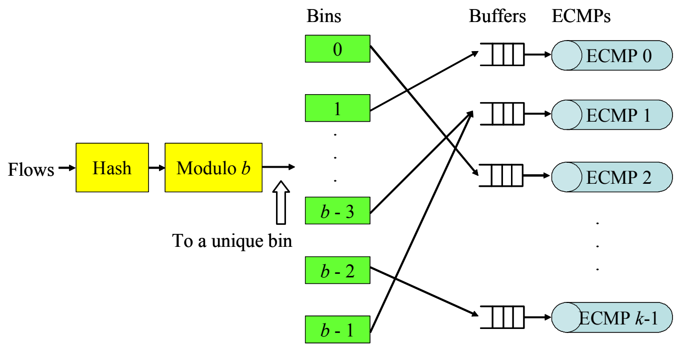

Fig. 2 Design of Table-based Hashing at a typical traffic splitting router

Fast Switching [7] is proposed by Cisco. It has a cache which stores the ECMPs assigned to all recently seen flows. When the traffic belongs to the same flow arrives, it is always assigned to the same ECMP. When all memory allocated for the cache is used up and a new cache entry needs to be created, the oldest entry is deleted in favor of the new one. When a packet/flow arrives and no matching with the cache can be found, the flow is assigned to the next ECMP in a round robin fashion. This flow ID as well as the path assigned to it are then stored in the cache.

The performance of Fast Switching is affected by its cache size. With small cache size, most packets cannot find a matching in the cache. They will always be assigned to the next ECMP in a round robin fashion. The performance under this situation is similar to packet-by-packet algorithm – the load can be nicely balanced but a serious packet-out-of-order problem will be experienced. With large cache size, most flows and their assigned ECMPs can be cached without being replaced/flushed. As a result, packets belong to the same flow can always be assigned to the same ECMP. The performance under this situation is similar to the two earlier hash-based algorithms – the load is not so nicely balanced but the packet-out-of-order problem is minor.

Besides equal traffic splitting, unequal traffic splitting among ECMPs is also very important. Among the four traffic splitting algorithms mentioned above, except Direct Hashing, both equal and unequal traffic splitting can be supported. Based on the assumption that *equal* traffic splitting is easier to implement (which we tend to disagree), three heuristic algorithms for emulating the unequal traffic splitting by equal traffic splitting algorithms are proposed [5] recently.

## III. Table-based Hashing with Reassignments

### A. *THR Algorithm*

Without loss of generality, Fig. 3 outlines the design of our proposed Table-based Hashing with Reassignments (THR) algorithm. The source IP address, destination IP address, source port number and destination port number fields of a packet are input to the hash function to produce a hash value, which is then divided by the number of bins. The remainder of the division points to one of the bins and this bin in turn determines the ECMP to which the packet should be assigned. Unlike the original Table-based Hashing scheme, the connections between the bins and ECMPs can be adjusted/reassigned according to traffic loads.

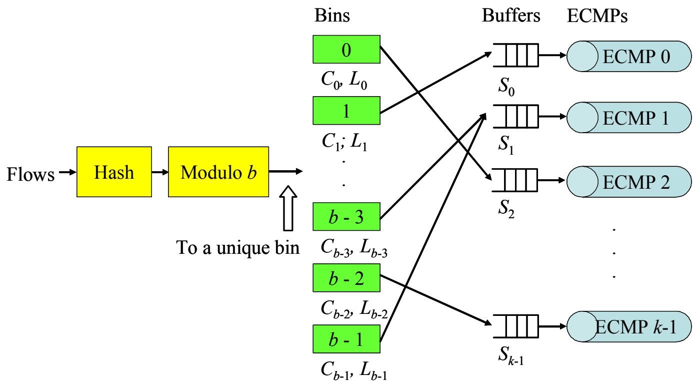

Fig. 3 Design of THR at a typical traffic splitting router

Packets assigned to an ECMP will wait in the corresponding output buffer for transmission. Without loss of generality, let hash bins 0, 1, ..., $m$-1 point to output buffer $i$ of ECMP $i$. At output buffer $i$, $m$ packet counters $C_0$, $C_1$, ..., $C_{m-1}$ are maintained to record the number of packets passing through, one for each bin. For each counter $C_x$, another parameter $L_x$ is used to store the last-modified time of $C_x$. So the total number of packets passing through ECMP $i$ in a predefined time interval of $T$ is $CurrS_i = C_0 + C_1 + ... + C_{m-1}$. Let the moving average of the number of packets passing through ECMP $i$ for each time interval of $T$ be $S_i$. $S_i$ can be found using an exponentially weighted moving average function as follows:

$$S_i = (1-\alpha)S_i^{'} + \alpha\, CurrS_i\,, \qquad (1)$$

where $S'_i$ is the moving average calculated in the previous interval $T$, and $\alpha$ is the smoothing constant ($0 \le \alpha \le 1$).

The value of $T$ determines the frequency for updating bin allocations. A small value gives a finer adjustment (and thus better load balancing performance) but a higher probability of packet-out-of-order. We recommend $T$ should be comparable to the average round trip time for all flows the router carried. In Section IV, the impact of using different values of $T$ will be examined by simulations.

Assume the desired traffic splitting ratio among the $k$ ECMPs is $x_0{:}x_1{:}\ldots{:}x_{k\text{-}1}$. The algorithm for THR is summarized in Fig. 4.

```
At each interval of $T$,
1. Calculate the moving average $S_i$ from Eqn.(1) for every ECMP.
2. Find ECMP $p$ such that $\frac{S_p}{\sum_{i=0}^{k-1} S_i} - \frac{x_p}{\sum_{i=0}^{k-1} x_i}$ is maximized.
3. Find ECMP $q$ from the set of ECMPs whose buffer occupancy is less
   than a pre-defined threshold value $\gamma$ such that $\frac{x_q}{\sum_{i=0}^{k-1} x_i} - \frac{S_q}{\sum_{i=0}^{k-1} S_i}$ is
   maximized.
4. If ECMP $q$ is found, go to Step 5. Otherwise, go to Step 6.
5. Among the $m$ bins connected to ECMP $p$, select bin $r$ such that $C_r + \beta(CurrTime - L_r)$ is maximized. Reassign bin $r$ from ECMP $p$ to ECMP $q$.
6. Reset all packet counters $C_x$ to 0 and the last modified time $L_x$ to $CurrTime$.
```

Fig. 4 Algorithm for THR

Step 1 calculates the moving average $S_i$ for every ECMP. Step 2 selects the most-utilized ECMP among all ECMPs. Step 3 selects the least-utilized ECMP. Note that the buffer occupancy of the selected ECMP should not exceed $\gamma$ (which is set to 0.8 in our simulations). This helps to avoid transferring traffic to a full or nearly-full output port buffer. If the buffer occupancies on all ECMPs exceed $\gamma$, no flow reassignment is needed.

If a least-utilized path is identified in Step 4, Step 5 selects an appropriate bin pointing to the most-utilized path and reassigns it to the least-utilized path found. In particular, the bin with the maximum value of $C_x + \beta(CurrTime - L_x)$ will be chosen, where $\beta$ is a weighting factor for combining the two components $C_x$ and $(CurrTime - L_x)$ together. A small value of $\beta$ implies that we focus more on the first component $C_x$. If $\beta = 0$, we always redirect the largest traffic flow (on the most-utilized path) to the least-utilized path. This tends to give a better load balancing performance. On the other hand, a large value of $\beta$ implies that we focus more on the second component $(CurrTime - L_x)$, which indicates the duration that bin $x$ does not have any packets arrived. If this duration is long enough and the corresponding bin is selected for reassignment, then by the time new packets arrived at this bin (which will follow the new path), the previous packets should have already been on its way (along the old path) to the destination if not already arrived. Therefore, a large $\beta$ has the effect of minimizing the packet out-of-order problem. In the next section, the value of $\beta$ is varied for investigating its impact on the load balancing performance and the packet out-of-order problem. Finally, Step 6 sets the parameters for the next update period $T$ no matter whether a reassignment is done in the current update period or not.

Note that if packets with variable sizes are used, counters $C_0$, $C_1$, ..., $C_{m-1}$ and $CurrS_i$ in Fig. 4 should be modified to function as byte counters instead of packet counters.

### *B. Complexities*

From Fig. 4, we can find that the worst-case time complexity of the Table-based Hashing with Reassignments (THR) algorithm is $O(b)$, where $b$ is the total number of bins. The number of executions of the THR algorithm per unit time is inversely proportional to the bin allocation update interval $T$. We have recommended that the value of $T$ should be comparable to the average round trip time of all the flows the router carried. In the next section, we will show that even this $T$ value is larger than the recommended value, the performance degradation is minimal.

Regarding the extra hardware required by the THR algorithm, $(2b + k)$ registers are required for implementing counters $C_x$, $L_x$ and $S_i$, where $b$ is the number of bins and $k$ is the number of ECMPs. The arithmetic operations conducted on those counters are very simple. In fact, such extra complexity is much lower than those required by Fast Switching, in which a reasonably large cache size is required and searching through the cache is much more time consuming. In short, we believe that with today's router technologies, the extra complexity of implementing THR algorithm at a router is marginal.

## IV. Simulation Results

### *A. Simulation Model*

Fig. 5 shows the network topology we adopted. Traffic is generated from two subnets, 0 and 1, each represents 300 traffic-generating hosts. Traffic generated from subnet 1 all goes to node 15. There are 8 equal-cost-multi-paths (ECMPs) between subnet 1 and node 15, via routers 3 to 10. Traffic generated by subnet 0 serves as the background uni-path traffic, which is uniformly distributed to nodes 3 to 10. We set the buffer size of routers 2 to 14 to 220 packets, which is sufficiently large to fully utilize the available network bandwidths when TCP is used. The bandwidth of each link is 44.736 Mbps (DS3). The number next to each link in Fig. 5 represents its propagation delay in milliseconds.

Our simulations are based on NS-2 [8] and TCP Reno is used. The simulation time is 30 seconds and the statistics collected in the initial 10 seconds are ignored. For Table-based Hashing (TH) and our THR, bin size is set to $b$ = 1024. The adopted hash function is CRC-32 [9], which operates on the four packet header fields, source IP address, destination IP address, source port number and destination port number. For our THR, the smoothing constant $\alpha$ in Eqn. (1) is set to 0.5. For Fast Switching (FS), a cache size of 3000 flows (that is about 30% of the total number of flows simulated) is used. This size can give a balanced performance of low probability of packet-out-of-order and good traffic splitting performance.

The Internet traffic is dominated [10] by a very small percentage of large volume flows, such as FTP. In our simulations, we assume 2% of the traffic flows is of FTP type while 98% of the traffic flows is of other types, as detailed below. We also assume that at each host the flow arrival follows a Poisson process.

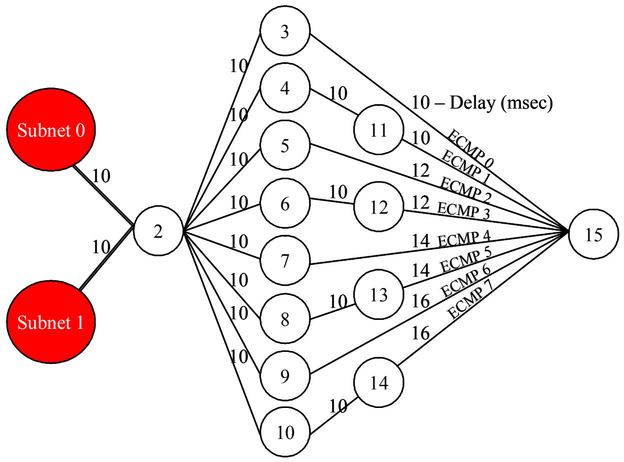

Fig. 5 Simulated network topology

- 2% of FTP: The amount of data for each flow, or flow volume, follows Pareto distribution with mean of 5000 KB and standard deviation of 500 KB.
- 32% of HTTP: The flow volume follows Pareto distribution with mean of 10 KB and standard deviation of 1 KB.
- 33% of Email: Pareto distribution with mean of 3 KB and standard deviation of 0.3 KB.
- 33% of DB: Pareto distribution with mean of 2 KB and standard deviation of 0.2 KB.

## *B. Equal Traffic Splitting Performance*

Refer to Fig. 5, we focus on the traffic flows destined to node 15, which can be split among the 8 available ECMPs. Assume the desired traffic splitting ratio among the 8 ECMPs is 1:1:1:1:1:1:1:1. Initially, we set the bin updating interval $T$=80 msec (the longest round trip time in the network), queue occupancy threshold $\gamma = 0.8$ and weighting factor $\beta = 100$.

For different traffic splitting algorithms, Fig. 6 shows the load balancing performance on 8 ECMPs. The numbers in each column indicate the share (i.e. the percentage of the total load) carried by the individual paths, from ECMP 0 at the bottom to ECMP 7 at the top. The total difference between the actual share and the ideal share of 12.5% over all 8 paths are summarized by the second row "Deviation" in TABLE I. The third row "Percentage" shows the percentage of packets arrived at the receiver (node 15) out-of-order. The last row "Delay" corresponds to the average end-to-end packet delay. As expected, Packet-by-Packet (PBP) algorithm gives the perfect load balancing performance but at the expenses of about 43% of packets arrived out-of-order. On the other hand, our proposed THR algorithm gives a load balancing performance very close to PBP (< 0.4% deviation), and with less than 2% of packets arrived out-of-order.

In principle, packets should experience no out-of-order problem under Direct Hashing (DH) and Table-based Hashing (TH) algorithms. But in our implementation, sequence number gaps (caused by network congestion) in the received packet stream are also counted as packet out-of-order events. This explains the observation of about 1% of packets arrived out-of-order under these two algorithms.

From TABLE I, we can also see that if the load balancing performance is good, the average end-to-end packet delay is small. As expected, THR provides the next-to-the-lowest end-to-end delay performance.

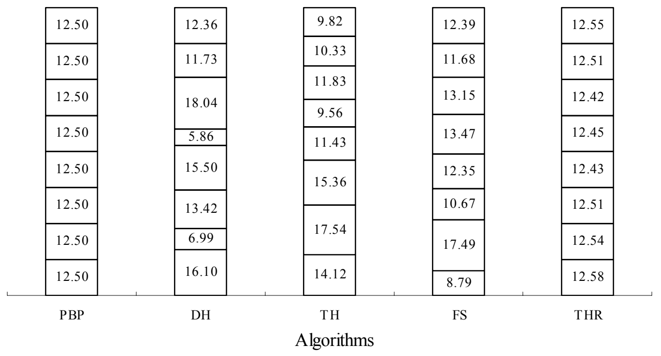

Fig. 6 Traffic equal-splitting using different algorithms

TABLE I LOAD BALANCING, PACKET OUT-OF-ORDER AND END-TO-END DELAY PERFORMANCE FOR EQUAL TRAFFIC SPLITTING

| **Algorithms** | **PBP** | **DH** | **TH** | **FS** | **THR** |
|---|---|---|---|---|---|
| **Deviation** | 0% | 26.2% | 19.05% | 20.14% | 0.39% |
| **Percentage** | 43.01% | 0.94% | 0.97% | 1.55% | 1.91% |
| **Delay (ms)** | 53.4 | 64.2 | 63.0 | 60.1 | 59.7 |

Fig. 7 shows the goodput at node 15 and the total goodput of the whole network, which is obtained by adding the goodput at all individual receivers. We can see that using multiple paths can generally provide higher goodput than using a single path (SP). The goodput performance of our THR is comparable to that of DH, TH and FS.

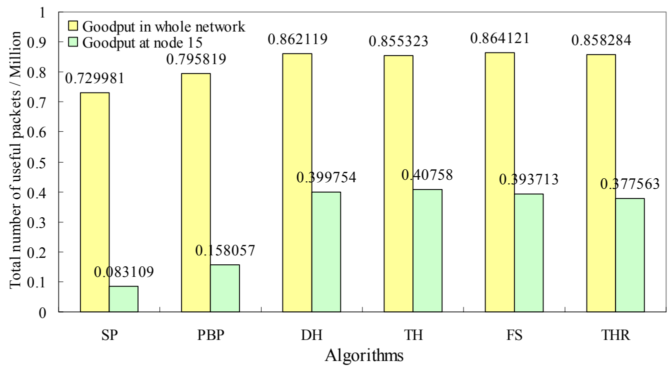

Fig. 7 Number of useful packets

Next we change the update interval $T$ of our THR algorithm to 500 msec and 1 sec for investigating its impact on load balancing performance. The total deviation from the desired share of 12.5% over all 8 ECMPs is summarized in TABLE II. Although the load balancing performance is poorer with larger update interval, compared with TABLE I, we can see that THR still outperforms the other three traffic splitting algorithms DH, TH and FS, by a large margin.

TABLE II Effect of update interval

| *T* (sec) | 0.08 | 0.5 | 1 |
|---|---|---|---|
| **Deviation** | 0.39% | 2.43% | 4.02% |

Finally, we vary the value of $\beta$ in THR to study its impact on both load balancing performance and packet out-of-order problem. TABLE III summarizes these results. As expected, as the value of $\beta$ increases, the load balancing performance becomes poorer (but still much better than DH, TH and FS) and the packet out-of-order problem becomes less serious.

TABLE III Effect of $\beta$

| $\beta$ | 100 | 300 | 500 |
|---|---|---|---|
| **Deviation** | 0.39% | 0.81% | 1.53% |
| **Out-of-order packets** | 1.91% | 1.71% | 1.57% |

## C. Unequal Traffic Splitting

Refer to Fig. 5, assume the bandwidth of all links on ECMP 0 and ECMP 1 are doubled while the bandwidth of all links on ECMP 6 and ECMP 7 are halved. Let the desired traffic splitting ratio among the 8 ECMPs be 2:2:1:1:1:1:0:0. For Packet-by-Packet (PBP) and Fast Switching (FS) algorithms, unequal-splitting is achieved by assigning a weight of 2 to the first two ECMPs, a weight of 0 to the last two ECMPs and a weight of 1 to all other ECMPs. For Table-based Hashing (TH), the hash bins are allocated such that 25% of bins are associated to each of ECMPs 0 & 1, and 12.5% to each of ECMPs 2 to 5.

For all four packet splitting algorithms, Fig. 8 shows the load balancing performance on the 8 ECMPs. TABLE IV summarizes the total deviation from the desired splitting ratio, the percentage of packets arrived out-of-order, and the average end-to-end packet delay. Again, we can see that our THR algorithm gives the overall best performance.

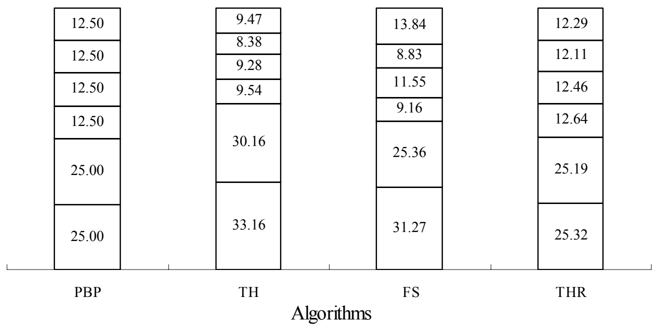

Fig. 8 Traffic unequal-splitting using differenct algorithms

TABLE IV Load balancing, packet out-of-order and end-to-end delay performance for unequal traffic splitting

| Algorithms | PBP | TH | FS | THR |
|---|---|---|---|---|
| **Deviation** | 0% | 26.64% | 15.92% | 1.29% |
| **Percentage** | 37.31% | 0.09% | 0.35% | 0.62% |
| **Delay (ms)** | 44.8 | 60.9 | 60.7 | 58.2 |

## V. Conclusions

In this paper, a new traffic splitting algorithm, called Table-based Hashing with Reassignments (THR), was proposed for load sharing over the set of equal-cost-multi-paths (ECMPs). As the traffic load in the network is time-varying, THR selectively reassigns some active flows from the over-utilized paths to under-utilized paths based on the load sharing statistics collected from each ECMP. The reassignment process is carefully designed to minimize the packet out-of-order problem. The effectiveness of THR was studied for equal and unequal traffic splitting scenarios. As compared with the existing traffic splitting algorithms, we found that THR provides a close-to-optimal load balancing performance, less than 2% of packets arrived out-of-order, and one of the lowest average end-to-end packet delays. Although additional traffic monitoring function is needed by THR, we argued that the extra complexity incurred is marginal.

## References

[1] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell and J. McManus, "Requirements for Traffic Engineering Over MPLS," RFC2702, Sep. 1999.

[2] Z. Wang, "Internet QoS, Architectures and Mechanisms for Quality of Service," Lucent Technologies, 2001.

[3] B. Fortz and M. Thorup, "Internet Traffic Engineering by Optimizing OSPF Weights," *Proceedings of IEEE INFOCOM*, vol. 2, pp. 519 – 528, March 2000.

[4] Y. Wang, Z. Wang and L. Zhang, "Internet Traffic Engineering without Full Mesh Overlaying," *Proceedings of IEEE INFOCOM*, vol. 1, pp. 565 – 571, April 2001.

[5] A. Sridharan, R. Guerin and C. Diot, "Achieving Near-Optimal Traffic Engineering Solutions for Current OSPF/IS-IS Networks," *Proceedings of IEEE INFOCOM*, vol. 2, pp. 1167 – 1177, March 2003.

[6] C. Villamizar, "OSPF optimized multipath OSPF-OMP," INTERNET-DRAFT, October 1998, (work in progress)

[7] A. Zinin, "Cisco IP Routing, Packet Forwarding and Intra-domain Routing Protocols," Addison Wesley, Section 5.5.1, 2002.

[8] http://www.isi.edu/nsnam/ns/

[9] International Organization for Standardization, "ISO Information Processing Systems - Data Communication High-Level Data Link Control Procedure - Frame Structure", IS 3309, October 1984, 3rd Edition.

[10] J. Y. Jo, Y. Kim and F. L. Merat, "Internet Traffic Distribution over Multilink Where High Bandwidth Scalable Switch Port Aggregates Multiple Physical Links," *Proceedings of IEEE ICC*, May 2003.