Stable Load Control with Load Prediction in Multipath Packet Forwarding

IlKyu Park*, Youngseok Lee, Yanghee Choi

School of Computer Science and Engineering

Seoul National University

San 56-1, Shinlim-dong, KwanAk-Ku, Seoul, Korea

TEL: +82-2-876-7170, FAX: +82-2-876-7171

{xiao,yslee,yhchoi}@mmlab.snu.ac.kr

August 14, 2000

Abstract

With the invention of MPLS, complex traffic engineering and multipath packet forwarding has become a reality. To use multipath packet forwarding, efficient load control between paths is necessary. Furthermore, dynamic load control is needed to optimize load mapping between paths depending on the network state. Unfortunately, distributed dynamic load control algorithms, if not carefully designed, can make the load mapping unstable, thus making many packets dropped due to unbalanced mapping. In this paper, we propose a dynamic load control scheme which makes the network efficient while keeping load mapping stable.

Keywords: Dynamic Load Control, Load Balancing, Multipath Forwarding, Hashing, MPLS

Designated technical subject : Communication Protocols and Architecture

*Please send all correspondence to IlKyu Park

1 Introduction

As the Internet has become the most important technology of today, it is experiencing rapid changes and developments. One of the most amazing improvement is the invention of Multi Protocol Label Switching(MPLS). With MPLS, packets can be forwarded at the speed only achievable with a hardware switch. It is also important for traffic engineering, as it provides explicit label switched path, and a set of attributes can be associated with LSPs. It is easy to integrate a constraint based routing with MPLS.

With multipath forwarding, a router can have many paths to a destination. This paths can be used simultaneously to provide more bandwidth than the bandwidth of a single path. Additionally, multiple paths can be used as backup paths for one primary path. If the primary path goes down, other paths can quickly be deployed.

Although MPLS enabled simultaneous use of multiple paths, it is not sufficient to effectively distribute network loads. There should be an efficient scheme for finding the paths from a source to a destination. Also, an algorithm to distribute the traffic among the selected paths is needed. If load control should be done according to the current network state, there should be a link-state exchange protocol to make the state of a link known to a remote node.

In this paper, we focus on dynamic load control considering network state. The goal of load control is to provide more network resources to endpoints while make the network efficient by reducing idle links. With dynamic load control, routers consider network state to adjust the mapping from input traffic into paths (LSPs). These paths are provided with MPLS. Depending on the load level of each link, dynamic load control algorithm distributes the load so that packet loss due to congestion becomes minimal, and the traffic is evenly distributed over possible paths. While dynamic load control provides more bandwidth for endpoints and makes the network efficient, it tends to be unstable because of its dynamic nature. Load control is performed in distributed manner, so each router calculates the usable or residual bandwidth of every links in the path. This decision is made independantly, so

when a link becomes underloaded, every router places more load on that link. At next stage, it becomes overloaded, so every router moves the load into other link. This cycle is repeated and make the network unstable, resulting every link congested after every other link state exchange. The instability ranges from a temporal instability to traffic oscillation in which no stable state is reached. We propose to use observed load change speed for guessing the number of paths used in the network, and control the load considering the change speed. We make prediction of the next load level, and modifies load mapping at an ingress router.

This paper is organized as follows. In section 2 the related works concerning dynamic load control is explained. In section 3 we define the problem and the environment which we assumed in solving the problem. Following section 4 explains the algorithm we propose, and section 5 defines the algorithm. Simulation result is presented in section 6, and the conclusion is given in section 7.

2 Related Works

There have been many studies on multipath forwarding issue. [1] analyzed the path disruption effects in multipath forwarding, but load control is not considered. In [6], authors proposed a dynamic load control scheme by separating long-lived flows from short-lived flows. [7] proposes a name-based hashing in which client requests are mapped to the identical servers. For multiple path finding algorithm, [9] proposed an efficient algorithm for constructing short disjoint paths. [10] presented a distributed algorithm for finding two disjoint paths. Also, traffic engineering based on multipath routing in the IP network are proposed in [11, 12]. This research is focused on local optimization. In [5], hashing based load distribution method is described, but how to adjust the load depending on the network state is not presented.

3 Problem Definition

3.1 Network Assumption

In this section, we explain what is necessary to deploy multipath forwarding with dynamic load control. First, an algorithm to find paths for forwarding is required. Several algorithms are suggested for multipath construction including [9, 10].

To use these paths, forwarding mechanism with explicit path is needed. Equal cost multipath (ECMP) or MPLS can be used for multipath forwarding. If ECMP is used, path selection is restricted to those having equal costs. Using MPLS, we can make use of arbitrary paths, with additional advantage of fast packet forwarding. In this paper, we assume that MPLS is used for explicit path forwarding and use the term Label Switched Path(LSP) to denote the path.

Another requirement is link-state information exchange protocol. To perform dynamic load control which controls the load according to the state of each path, a router must know the state of each link on that path. This information can be obtained by link-state routing protocols like OSPF[3], with QoS extension[4]. The link state information exchanged should include the load on each link and residual bandwidth of the link, or the load and physical bandwidth of the link. The load on the link is the amount of packets arrived to the link including packets dropped at that link. This value can also be calculated from throughput, which is the amount of data passed through that link, and loss rate of the link.

Finally, a mechanism to map input traffic into one of the possible LSPs is required. This mapping should be done on flow level, as mapping each packet to different LSP can result in many out-of-order packets, which have a negative effect on TCP performance. This mapping technique should allow change the mapping so that load on an LSP can be moved to other LSP. Hashing-based packet mapping described in [5] can be used for this purpose.

3.2 Load control according to the dynamic network state : Problem Definition

Load control problem is the problem of mapping input load into each possible LSP. The mapping determines how the network link will be used, thus determines network throughput. Loads must be mapped so that the number of congested links and idle links is minimized. The difficulty is these algorithm must be distributed, so that each router can determine their mapping without much control overhead, and thus be scalable. Distributed means their information about remote link is not accurate, and this difference can make load control inefficient, and may collapse from unstable condition. So, the problem is to make an algorithm stable enough but maximize network throughput using multipath.

3.3 Instability of Dynamic Load Control

In a network with multiple paths from an ingress router to some egress router, network bandwidth can be used more effectively by adjusting loads between several possible paths. By adjusting loads between paths, congestion on some link can be mitigated. To move loads and to balance loads, every router must know the state of network, and figure out which links are underutilized and which links are congested. This requires that each router must report its state to other routers. Unfortunately, state report consumes network resource, and the state information cannot be exchanged frequently enough for every router to know each other exactly.

The period between link state information exchange makes the state information inaccurate. If the period is longer, then the information exchanged will be more inaccurate, due to the difference of time when the information is calculated and when it is used. We can make it more accurate by exchange the link state information more frequently. But, if the error is accumulated, changing the frequency only slows down the problem, not eliminates the problem entirely.

In a naive dynamic load control method in which a router distributes the traffic into each path depending on the load of path, the inaccuracy resulting from link-state exchange is significant. In a

naive dynamic load control, each LSP to a destination is given the load according to the remaining bandwidth of that LSP. When a link is reported to be underutilized, every router that has one or more path through that link will map more traffic into that link. At next link state exchange, the link will certainly be reported to be overloaded, and again every router decreases the traffic through that link. So, the error is accumulated, making the link oscillating between underutilized state and congested state. This behavior had been shown in [6].

4 Stable Load Control with Prediction

This section describes the idea in the proposed algorithm. The instability presented in previous section results from the inaccuracy of link-state information, especially from the difference between current state and previous exchanged state. With the naive dynamic load control, each router places more load on a underloaded link, and decreases the load on the congested link, hence the underloaded link becomes congested, and congested link becomes underloaded. This cycle make the load difference between underloaded and congested links, resulting unstable load mapping. Hence, predicting the current network state is required to eliminate load oscillation.

In our approach, we propose to use previous observed increasing speed of link load. The increasing speed of the link load is proportional to the number of paths through the observed link, and in turn, the difference between current states of links and previous link-state update will be proportional to the number of paths through the link. Hence, the present inaccuracy can be estimated using the previous increasing speed. We denote this speed as s, and maintain the mean $\overline{s}$.

To make the prediction accurate, we propose that every router have the same criteria for the decision of increasing/decreasing the load on a given link, by deploying a threshold T. If the link is congested such that the load is greater than T percent of the link bandwidth, every router lowers the load on that link. If the link load is under T percent of the link bandwidth, then the link is considered underutilized, and each router deploys more traffic on the link. By this, we can predict

that the load will increase or decrease, and can use this as an estimation of the paths through the load. The threshold T on each link should be same or be advertised using link-state exchange.

We assumed that this algorithm is run in a core network, where traffic is greatly aggregated, and does not change abruptly. Our prediction is concentrated on the variation of load from load control. The proposed algorithm cannot discriminate between the variation from load control and the variation from source traffic change. If there is frequent change in the source traffic, we can make it smooth by extending link-state exchange period.

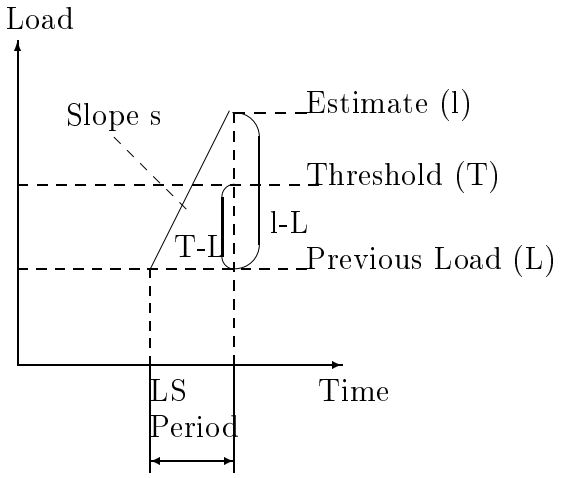


Figure 1: Estimation

After prediction, we apply the predicted result into load control. Here, we apply proportional decrease to keep the link near or under congested state. With the predicted load level $l(= CurrentLoad + LinkStateExchangePeriod * \overline{s})$, threshold T and link load L of previous link-state exchange, excessive load is $l - T$, and load increase due to the load control is $l - L$, as shown in Fig. 1. Therefore, we need to suppress the load increase by the factor of $(T - L)/(l - L)$, which will make the load level at next period around the threshold.

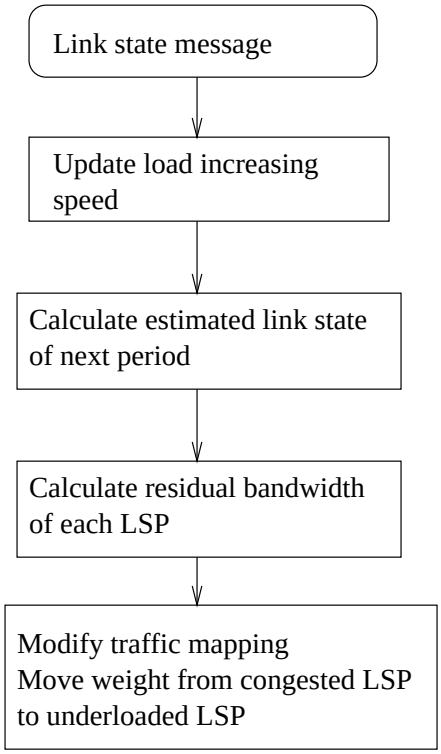


Figure 2: Flowchart of proposed algorithm

5 Algorithm

To perform dynamic load control, each router must keep track of past traffic trends, estimate next states of links, distribute input load into each output LSPs. This section presents these algorithms.

Flowchart of our algorithm is given in Fig. 2. Upon receiving a link-state update message, the router updates the speed at which load of a link increase. Then, it estimates the link load at next period, calculate residual bandwidth of each LSP using the estimated link load. With the residual bandwidth of each LSP, traffic mapping into paths is modified.

First, when a router receives link-state update message containing the link load, it checks if the link load has increased. If it has increased, then the router updates the mean value of increasing speed. This value is kept as a moving average with the parameter α. PL and CL in this algorithm means previous load and current load, respectively, and 'i' represents link id.

Second, upon every link-state update message, the router estimates the state of links at next

Algorithm 1 procedure $LinkStateUpdate(i, Load)$

1: **if** $(PL(i) < Threshold) and (PL(i) < CL(i))$ **then**
2: $\quad \overline{s(i)} = \overline{s(i)} * (1 - \alpha) + (CL(i) - PL(i))/UpdatePeriod * \alpha$
3: **end if**
4: PL(i) := CL(i)

period. Estimation is based on the current load level and the speed at which the link load changes. In our scheme, if the load of a link is higher than the threshold, routers do not assign more traffic into the LSP whose bottleneck is the link. So, we apply estimation to the links with load level less than threshold. When the load is less than the threshold, we estimate the load as $CL(i) + \overline{s} * UpdatePeriod$. In this algorithm, EL means Estimated Load, CL means Current Load.

Algorithm 2 procedure $LoadEstimation(i)$

1: **if** $CL(i) < Threshold$ **then**
2: $\quad EL(i) := CL(i) + \overline{s(i)} * UpdatePeriod$
3: **else**
4: $\quad EL(i) := CL(i)$
5: **end if**

With estimated load, we control the load between LSPs by adjusting path load ratio. We define 'path load ratio' as the ratio of input traffic with same destination and the amount of traffic the router forwards to the path. For example, if there are two paths to destination A, and path load ratio of path 1 is 30%, then the router forwards 30% of input traffic for A using path 1. The bandwidth of LSP is defined as the residual bandwidth of the bottleneck link in the path. While finding the bandwidth of LSP, each LSP is marked if its bottleneck is a link with load level higher than threshold.

After LSP bandwidth is calculated, load distribution is performed by moving path load ratio. First, the total residual bandwidth for a destination is calculated by summing up all residual bandwidth of paths for same destination. Then, each paths with load level higher than threshold is considered. As its load is high, the path load ratio for this link should be moved into other link, and to make its load level around threshold, resulting path load ratio should be set to (current load ratio)*(threshold)/(load level). Before moving the excessive load, the total residual bandwidth is checked if it can accomodate the excess load. If it can, then the path load ratio is decreased by the

factor of (threshold/load level). If it cannot, then current load level is preserved. This step is repeated for all paths with load level higher than threshold, and total load move is calculated. Finally, the decreased load (total excess load) is added to the paths with load level lower than threshold, proportional to the residual bandwidth of respective paths.

Algorithm 3 procedure *AssignWeight*(d)

```
1: G := LSPs with destination d
2: for all LSP in G do
3:    RB(LSP) := Smallest residual bandwidth value of the links in LSP, using EstimateLoad(i)
4:    BW(LSP) := Physical bandwidth at the bottleneck
5:    PRA(LSP) := Previous Load Ratio of this LSP
6:    Mark if the bottleneck is heavily loaded (more than threshold)
7: end for
8: RES_BW_RATIO_SUM := (Sum of residual bandwidths of paths with load level lower than
   threshold)/(Previous Input Load)
9: MOVE_BW_RATIO_SUM := 0
10: H := LSPs in G, with 1-RB(LSP)/BW(LSP) > Threshold
11: for all LSP in H do
12:    NEXT_RATIO = PRA(LSP) * (BW(LSP) * Threshold) / (BW(LSP) - RB(LSP))
13:    MOVE_RATIO = PRA(LSP) - NEXT_RATIO
14:    if MOVE_RATIO + MOVE_BW_RATIO_SUM < RES_BW_RATIO_SUM then
15:       SetRatio(LSP, NEXT_RATIO)
16:       MOVE_BW_RATIO_SUM = MOVE_BW_RATIO_SUM + MOVE_RATIO
17:    end if
18: end for
19: I := G - H
20: PartialSum := 0
21: for all LSP in I do
22:    if (BW(LSP)-RB(LSP))/BW(LSP) > Threshold then
23:       RB(LSP) := RB(LSP) * (RB(LSP) - CL(LSP))/(Threshold - CL(LSP)/BW(LSP))
24:       PartialSum := PartialSum + RB(LSP)
25:    end if
26: end for
27: for all LSP in I do
28:    SetRatio(LSP, PRA(LSP) + MOVE_BW_RATIO_SUM * RB(LSP) / PartialSum)
29: end for
```

6 Simulation Results

6.1 Simulation Environment

We have implemented multipath forwarding scheme in NS-2([8]). We also implemented link-state information exchange protocol. We used MPLS module for NS2 for multipath forwarding. We made a simple network(Fig. 3), and attached 6 CBR connections. The network consists of 6 ingress routers, A-F, and 6 egress routers A'-F'. For every ingress-egress pair, 4 LSPs are constructed, using 4 different links at the center of network. LSPs for different ingress-egress pairs may overlap at the center of network. The logical topology is shown at Fig. 4.

We performed simulation for various load. The bandwidth of total connections ranges from 6Mbps(100% of total physical bandwidth) to 4.2Mbps(70% of total bandwidth). There are 4 LSP paths for each connection. As we have assumed a core network where the traffic is aggregated much and several alternate path is possible, our model fits the real network.

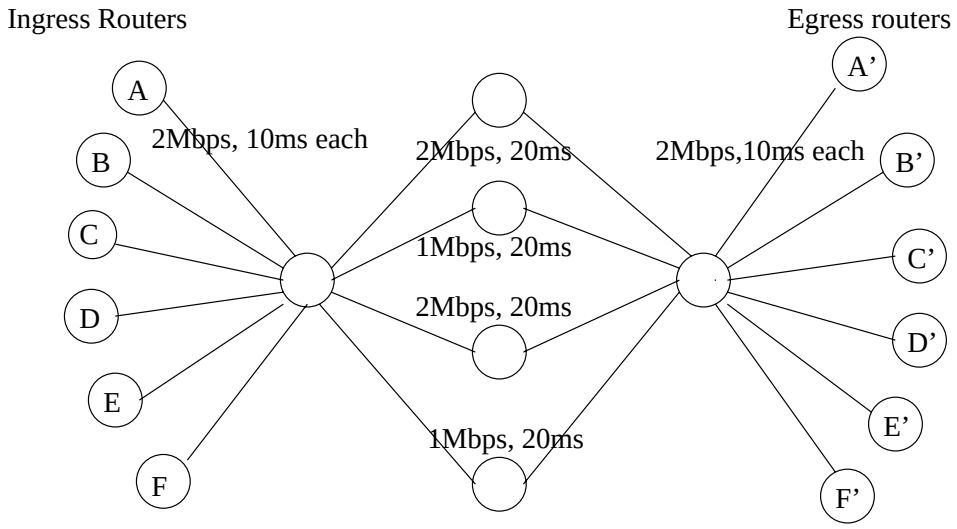


Figure 3: Simulation Topology

We also implemented a simple dynamic load control scheme. In this scheme, input traffic is mapped to each LSP according to their current residual bandwidth. We made simulation result with

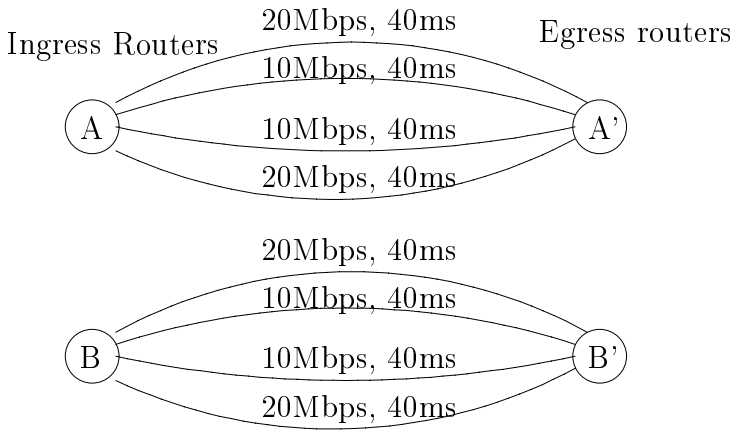


Figure 4: Logical Topology (others are the same)

each scheme, with different network load and compared them.

6.2 Simulation Result

Fig. 5 compares the case for simple, no estimation scheme and the case for proposed prediction scheme. Total load on the network varies from 6Mbps to 4.8Mbps. Under high load level (6Mbps and 5.6Mbps), the simple load control algorithm works well, and shows stable load distribution. But under lower load level, the load distribution soon become unstable and oscillate between links. The result of high load level is because, as there are not much load left, load control simple didn't worked and current level is usually next load level. But, under light load, there are much bandwidth left for load control, and the simple load control becomes unstable.

Right column of Fig. 5 shows the result of our scheme. Simulation settings are the same as the previous one. Under high load, there is not much difference from the case of simple load control. Both load control algorithm works well. Under light load, the proposed algorithm shows stable load level, and does not suffer from load oscillation.

Fig. 6 compares packet drop rate with each scheme. These figures show the ratio of lost bytes to the total load. There are not much difference when the load level is high, because every link is congested and load control does not work. As the load level is lowered, loss rate for proposed

algorithm slowly decreases, while loss rate for simple algorithm varies randomly. In the case of simple algorithm, the loss rate shows periodical high points aroune 0.25 and 0.3. At the load level of 70%, simple algorithm still shows high loss rate, while proposed algorithm shows few and stable loss trend.

In Fig. 7, the load differences between links are examined. Link load ratio is defined as the ratio of link load to its physical bandwidth. We examined the four links at the center of network, and calculated their standard deviation at each period. In the graph, we can see that there is not much difference when the load is high, but when the load is below 70%, the difference between links becomes quite large for simple algorithm. Deviation of links for proposed algorithm remains almost the same.

Fig. 8,9,10 shows the results when there is a change in source traffic. Fig. 8 (a) and (c) show the load level for each link when the load is changed from 4.8Mbps to 3.6Mbps at time 15, and returned to 4.8Mbps at time 30. Fig. 8 (b) and (d) show the load level when the load is changed from 3.6Mbps to 4.8Mbps at time 15, and returned to 3.6Mbps at 30. With the proposed algorithm, the load at each link remains still even when there is a change in input traffic (Fig. 8 (a) (b)). In simple algorithm, the load level starts fluctuating when there is abrupt change in input traffic (Fig. 8 (c) (d)).

Fig. 9 shows the deviation between loads of 4 links at the center. The simple algorithm is affected by the change of load, and the deviation increases quite high after the load change. Proposed algorithm shows stable graph at the event of change. Fig. 10 compares the drop rate. It also shows high drop rate for simple algorithm when there is load change, but the proposed algorithm's drop rate was not affected by the change of load.

7 Conclusion

In this paper, we proposed a dynamic load control scheme which does not suffer from instability. Proposed algorithm successfully distributed traffic into multiple paths, and made efficient use of network resources by considering load on each path. Proposed algorithm used estimation of network

load to prevent the link-state errors and oscillations resulting from the difference between current network state and previous link-state exchange.

In the simulation result, we have shown that the proposed algorithm works well in distributing traffic and does not suffer from instability as in simple dynamic load control. We have also shown that the proposed algorithm works when there are abrupt load change.

References

[1] D. Thaler, C. Hopps, "Multipath Issues in Unicast and Multicast NextHop Selection," Internet Draft, draft-thaler-multipath-05.txt, Feb. 2000

[2] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture", work in progress, draft-ietf-mpls-arch-06.txt, August 1999.

[3] J. Moy, "OSPF Version 2," RFC2328, April 1998.

[4] R. Guerin et al., "QoS Routing Mechanisms and OSPF Extensions," RFC 2676, August 1999.

[5] Zhiruo Cao, Zheng Wang, Ellen Zegura, "Performance of Hashing-Based Schemes for Internet Load Balancing," INFOCOM 2000

[6] A. Shaikh, J. Rexford, and K. G. Shin, "Load-Sensitive Routing of Long-Lived IP Flows," SIGCOMM'99, 1999

[7] D. Thaler, and C.V. Ravishankar, "Using Name-Based Mappings to Increase Hit Rates," IEEE/ACM Transactions on Networking, February 1998.

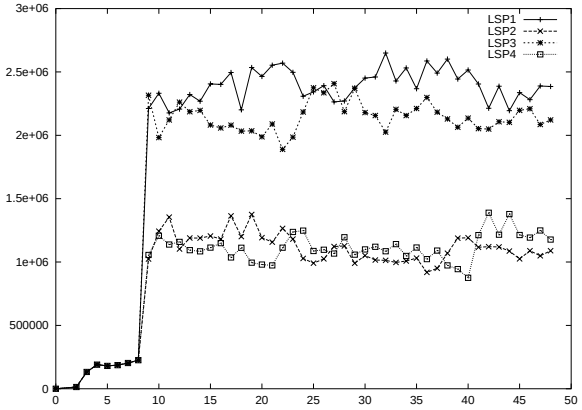
[8] UCB/LBNL/VINT Network Simulator - ns, http://www.isi.edu/nsnam/ns/

[9] Don Torrieri, "Algorithms for Finding an Optimal Set of Short Disjoint Paths in a Communication Network," IEEE Transactions on Communications, Vol. 40, No. 11, November 1992

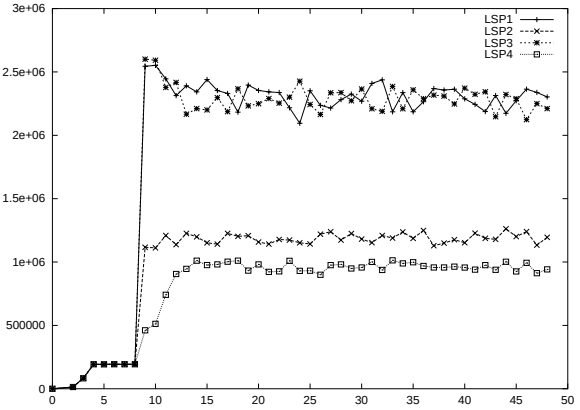
[10] Richard G. Ogier, Vlad Rutenburg, Nachum Shacham, "Distributed Algorithms for Computing Shortest Pairs of Disjoint Paths," IEEE Transactions on Information Theory, Vol. 39, No. 2, March 1993

[11] C. Villamizar, MPLS Optimized Multi-path(MPLS-OMP), IETF Internet Draft(Work in Progress), Feb. 1999

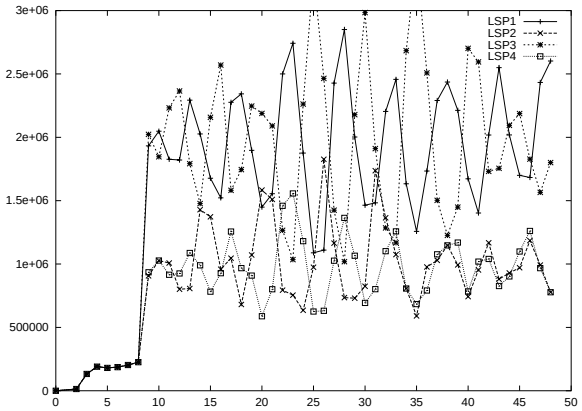
[12] C. Villamizar, OSPF Optimized Multi-path(OSPF-OMP), IETF Internet Draft(Work in Progress), Feb. 1999



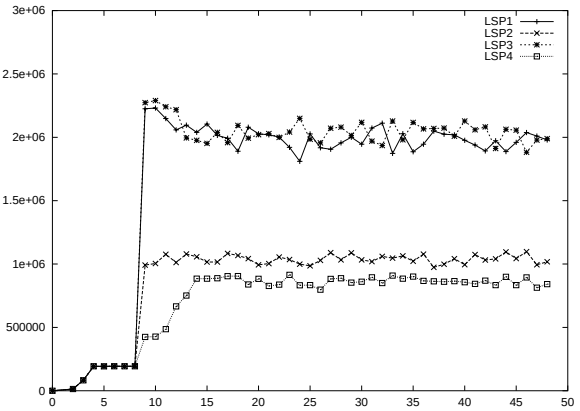
(a) Simple Load Control, 6Mbps



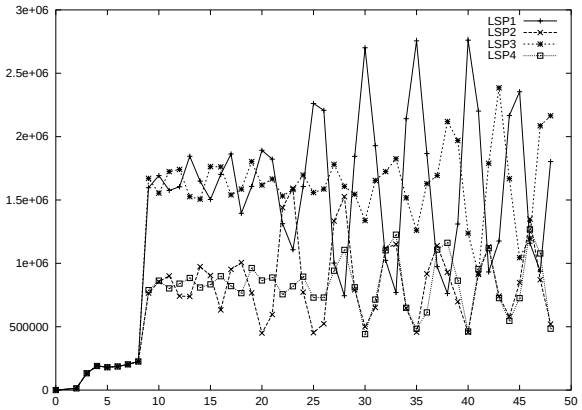
(b) Proposed Load Control, 6Mbps



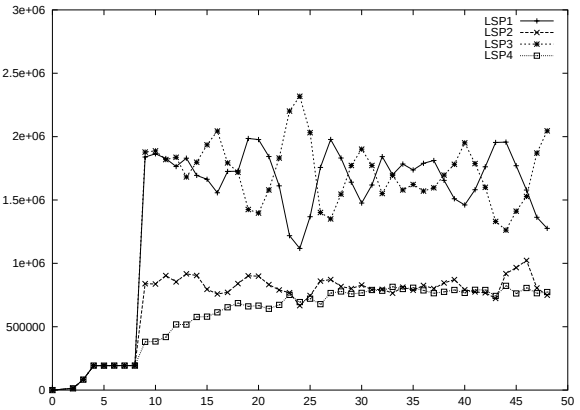
(c) Simple Load Control, 5.2Mbps



(d) Proposed Load Control, 5.2Mbps

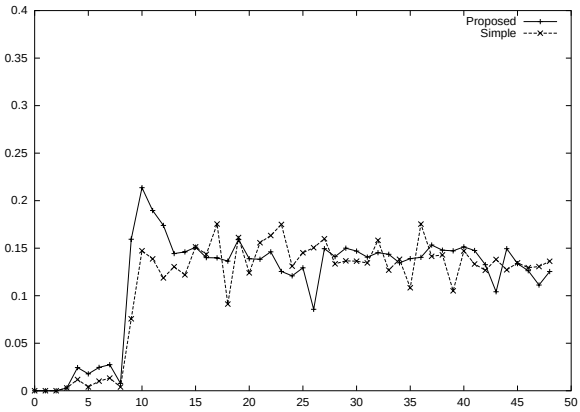


(e) Simple Load Control, 4.2Mbps

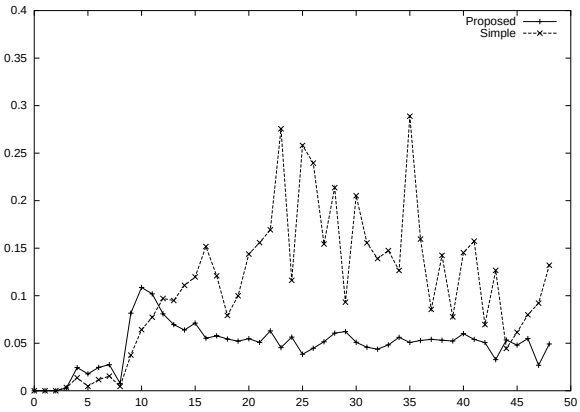


(f) Proposed Load Control, 4.2Mbps

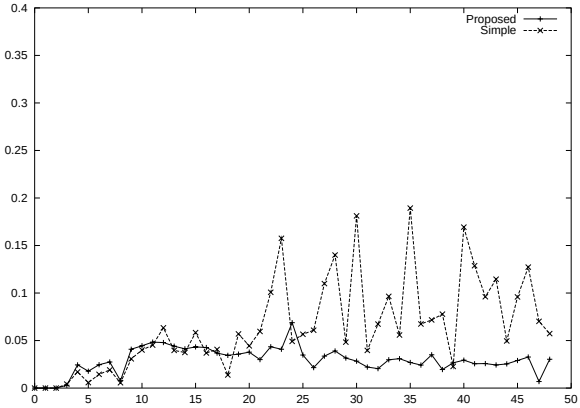
Figure 5: Comparison of Load Level



(a) Load = 6Mbps

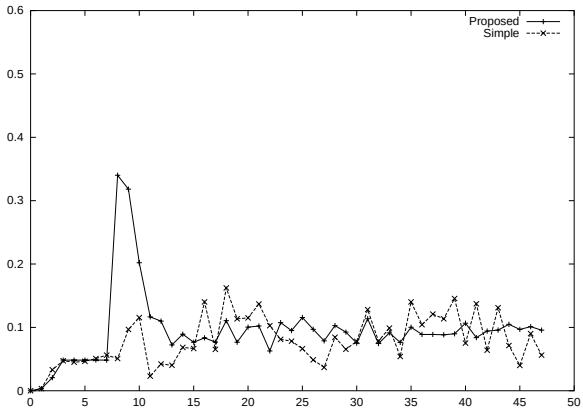


(b) Load = 5.2Mbps

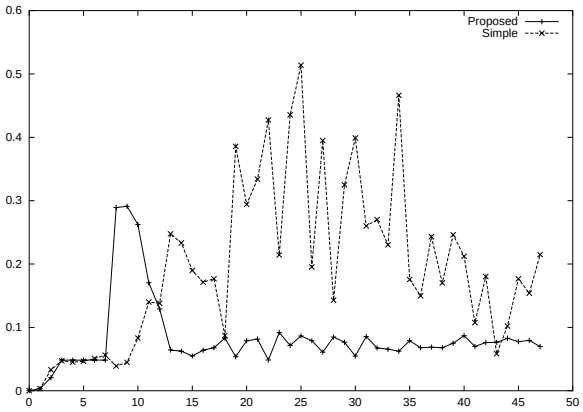


(c) Load = 4.2Mbps

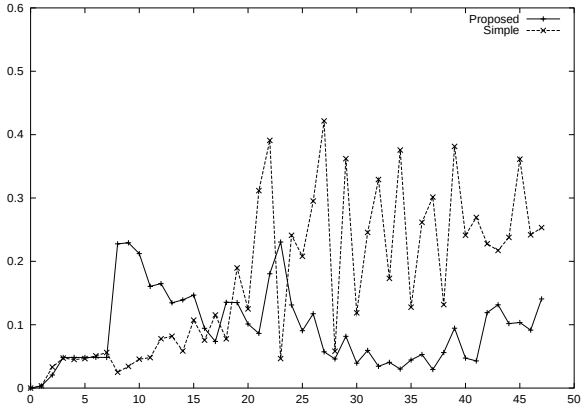
Figure 6: Comparison of Drop Rate



(a) Load = 6Mbps

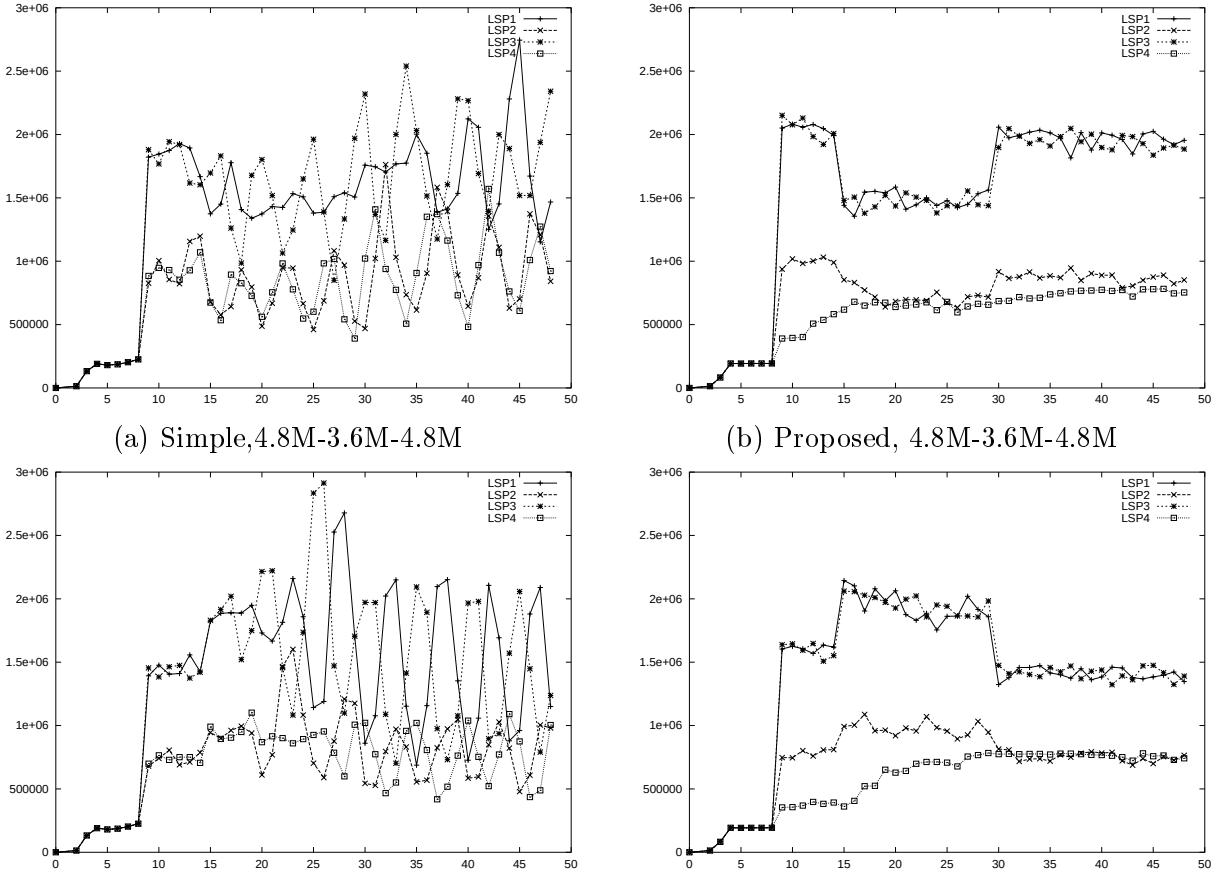


(b) Load = 5.2Mbps



(c) Load = 4.2Mbps

Figure 7: Deviation of load between links



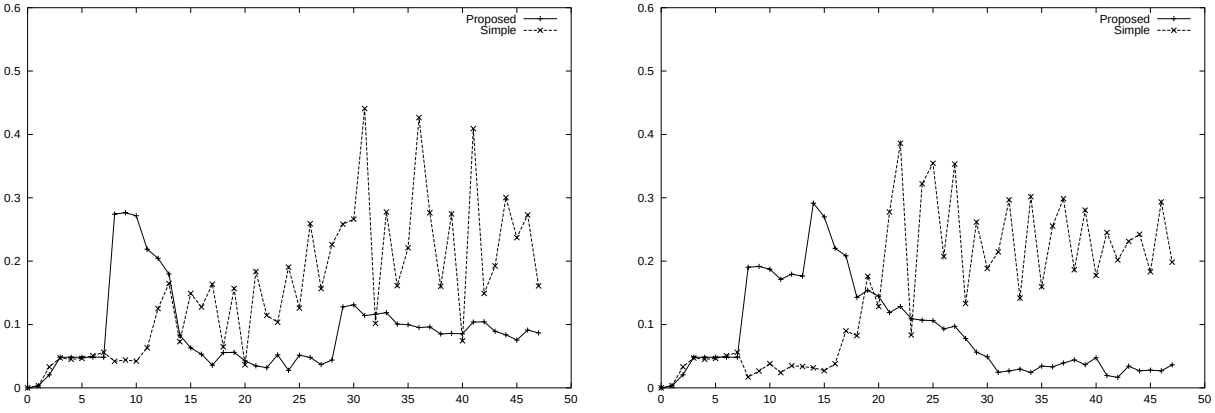
(a) Simple,4.8M-3.6M-4.8M

(b) Proposed, 4.8M-3.6M-4.8M

(c) Simple, 3.6M-4.8M-3.6M

(d) Proposed, 3.6M-4.8M-3.6M

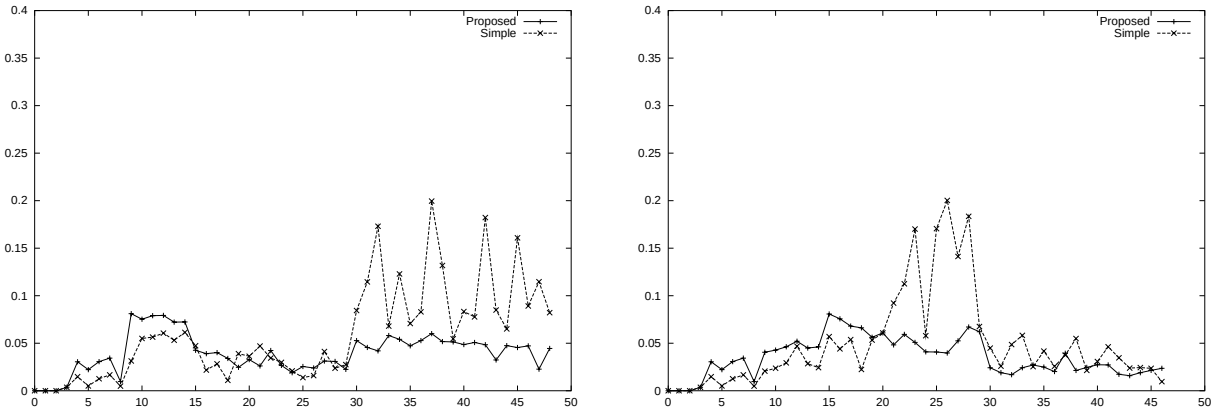
Figure 8: Load level, with a step change in source traffic



(a) 4.8M-3.6M-4.8M

(b) 3.6M-4.8M-3.6M

Figure 9: Deviation of load between links, with a step change in source traffic



(a) 4.8M-3.6M-4.8M

(b) 3.6M-4.8M-3.6M

Figure 10: Drop rate, with a step change in source traffic