The Performance Evaluation of KLONE

Yao-Nan Lien, Yu-Sheng Huang and Kuo-Chan Huang National Chengchi University, Taiwan,
lien@cs.nccu.edu.tw

Abstract—**In a high-speed packet switching network, the processing time in a node becomes an important delay component that must be taken into account in designing routing algorithms for time sensitive services. To illustrate this hypothesis, we propose a new delay sensitive routing algorithm, KLONE, that takes both link and node delay time into consideration. Our simulation study shows that our algorithm could easily outperform traditional routing algorithms that only consider link delay time.**

I. INTRODUCTION

Providing time sensitive services becomes an essential task for some packet-switching networks such as All-IP networks [1], which have to carry all types of traffics currently supported by both circuit-switching and packet-switching networks. Since routing is a critical task to select path to deliver a packet in a packet-switching network, such a network requires a delay sensitive routing mechanism to provide time sensitive services with QoS guarantee. However, most traditional routing algorithms do not take delay time as a major concern. Only a few are designed for time sensitive services [7,9,10,11]. These time sensitive routing algorithms were designed at the time when networks were slow and link bandwidth was the scarcest resource. As link bandwidth grows rapidly in recent years due to the advance of optical communication technologies, link bandwidth is no longer the only scarce resource. The processing time in a node, e.g. router, becomes another critical source of time delay. It must be taken into account in designing adequate routing algorithms for time sensitive services.

In this paper, we designed a new flow-based routing algorithm, KLONE, which takes average delay time as its minimization objective and both node and link as delay components. Through an intensive evaluation using simulation method, we demonstrate that the KLONE algorithm outperforms the traditional OSPF algorithm [6] by about 30%.

A. Path Delay Time

The delay time of a packet traveled along a path, referred to as *path delay time* in this paper, can be divided into three components: link delay, node delay, and end-host delay. Among them, end-host delay, which is occurred at the hosts of both ends, can be ignored in the design of routing algorithms since it is independent of the path it travels.

B. Link Delay Time

Link delay time contains three components, the queuing delay, the propagation delay and the transmission delay. Propagation delay is the time of an electrical or optical signal transmitted along a specific link. It depends on the length of the link as well as the physical characteristics of the media and the signal. Transmission delay is the time of a data unit being transmitted alone a specific link with the propagation delay time ignored. For example, it will take 82ms to transmit 128k bits data along a T1 link.

C. Node Delay Time

Node delay is the delay time occurs in a node (e.g., a router). It contains two components: the processing delay and the queuing delay for a processor. The tasks perform in a router and the processing power of the router determines its delay time. Queuing delay is the time of a packet waiting in a queue before being processed.

As link bandwidth grows rapidly and is getting closer to node processing capacity, node delay is increasing its share in a path delay time making itself a notable

component in network performance. For instance, in 2001, the Code-Red worm quietly invaded numerous computers on Internet. In only caused little impact to the invaded hosts, but generated many small-sized packets to saturate numerous mediate routers and switches causing serious network performance degradation. These short packets did not occupy too much link bandwidth. Instead, they caused heavy loads on routers and switches and hence increased packet delay time tremendously. Although it is an extreme case, it shows that the delay time occurs in network nodes is increasing its weight in network performance. Node delay becomes a dominant part in a high-speed network.

D. The Myth of Bandwidth

In the beginning of router algorithm development, the tasks performed by a router are simple and the power of the processor within a router is much faster than the links in terms of processing or transmitting packets. The link delay was the major concern in designing a delay sensitive routing algorithm. In recently years, network operators start to deploy fiber optic networks with DWDM technique making a dramatic increase in network bandwidth. This fast growth in bandwidth makes link bandwidth closer to the node processing capacity and results in the increase of relative weigh of node delay. One way to reduce the delay time cause by routers is to embed higher layer protocols to lower layer equipments such as Layer 3 switch or MPLS [2]. Another way is to choose better routing algorithms that take node delay time into consideration.

E. An Illustration Example

In the following example, discovering a minimal delay path with and without taking node delay into account will be compared. As shown in Figure 1(a), a simple network is composed of eight nodes, A, B, C, …, H, and eight directed links. The weight of link $\overrightarrow{AD}$ is 2, and all the other links is 1. We assume the delay time caused by a node is proportional to the number of traffic flows passing that node. There is a unit traffic demand form A to F, from B to G and from C to H, respectively. Without considering node delay, the best routing algorithm will route all three traffic flows through node E, as shown in Figure 1(b). The delay time of each path will be 6.

Figure 1(c) shows the result of another possible routing that takes node delay time into account. One traffic flow will pass node D, instead of node E. The delay time is then 5 for each flow.

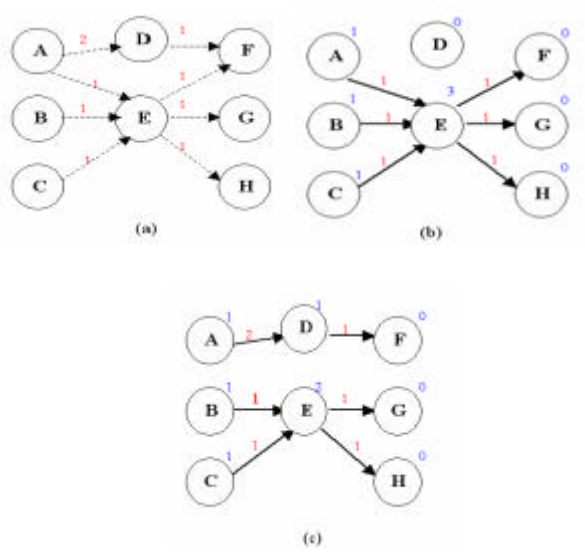


Fig. 1. Example routing with node delay considered.

Above example shows that, for high-speed networks, a routing algorithm that takes node delay time into account may obtain a better result as compared to the traditional routing algorithms that consider link delay time only.

II. RELATED WORK

A. Routing Approaches

1) Shortest Path Routing

Shortest path routing is to build up a shortest path tree to present the network topology, so then routes each of request traffics to its destination. The Dijkstra's shortest path algorithm [4,8] is a very famous example of shortest path routing, and it discovers a node's shortest paths to other nodes in $O(n \log n)$ with maintaining complete information about the network topology. For the knowledge of whole network topology, it is centralized in nature. Resulting from this centralized nature, it has a privilege, the loop-free feature. It is also a pre-computation approach, which executes the routing and set up the network before traffic actually occurs.

2) Flooding

Another routing algorithm is flooding [8], in which every incoming packet is sent out on every outgoing line except the one it arrived on. It generates vast numbers of duplicate packets. It is not practical in most applications, but it does have some uses.

3) Flow-Based Routing

Flow-based routing considers both the network topology and load [8]. The basic idea is that for a given line, if the capacity and average flow are known, it is possible to compute the mean packet delay on that line from queueing theory. The routing problem then reduces to finding the routing algorithm that produces the minimum average delay for the subnet.

4) Distance Vector Routing

Distance vector routing estimates the distance from source to destination by certain approaches [8,11]. They are often referred to as "Bellman-Ford" protocols because they are based on a shortest path computation algorithm. Distance vector routing protocols periodically send information to its neighbor nodes. Each node could estimate the distance to other nodes according to the number of intermediate passed nodes. If the network is fixed, the nodes will compute a converged result. The updated messages are only periodically sent to their neighbors, and the update messages do not cause a heavy load to the network compared to other protocols.

5) Link State Routing

Link state routing focuses on the states of links [11]. Link state routing transforms the link states into some mathematical expressions to choose proper paths. Open Shortest Path First (OSPF[6]) is a widely used example of link state routing and is used with RIP in parallel, which is a distance vector routing.

6) QoS Routing

QoS routing selects routes based on flow QoS requirements and network resource availability. QoS routing determines feasible paths satisfying QoS requirements, while optimizing resource usage and degrading gracefully during periods of heavy load [9,11]. Example QoS routing categories are: bandwidth-bounded routing; delay-bounded routing; bandwidth-bounded, delay-bounded routing; bandwidth-optimized, delay-optimized routing; bandwidth-bounded, cost-bounded routing; delay-bounded, cost-optimized routing; delay-optimized, bandwidth-optimized routing; delay-bounded, cost-optimized routing; delay-bounded, jitter-bounded routing.

B. Estimation of Delay Time

To execute a centralized pre-computation routing algorithm, we need to know the delay time that may occur on links and nodes. A packet switching networks works like a network of queues. Once the operating mechanisms of links and routers are known, we may be able to estimate their delay time. However, when the delay time is dependent on the load, the estimation may be difficult until the load is clearly known.

An approach to deal with unknown delay time in networks is sending probes through the network and measures its traveling time to estimate the delay time of a path. Christophe Beaujean[13] proposed a method that floods probes to deicide the shortest delay path. However, the characteristics of the probe may be much different from the request traffic, for example: size, and it might lead to incorrect information and then sender may make a sub-optimal route.

C. The KLONE Approach

The networks that are to support time sensitive services could choose to use a proper QoS routing. However, existing routing algorithms only consider link delays such that may not be adequate for high speed networks as explained in Section I As well as the section I, we model the problem as a flow-based routing problem with link and node delay dependent on their loads. An iterative approach is taken to cope with the difficulty of estimating load dependent delay time on links and nodes. We use a transformation to convert node delay into link delay such that intermediate problem of each iteration can be solved using a traditional routing algorithm. The proposed algorithm is evaluated by comparing with the OSPF algorithm and the KLONE without include node in Section III using average path delay time and goodput ratio as performance metrics.

III. ROUTING WITH NODE DELAY

A. Routing Problem Model

Given a directed graph *G(V, E)*, with |*V*| nodes and |*E*| links, the propagation delay time and bandwidth of each link, and the processing capacity of each node, the problem is to find a set of paths for a given set of traffic demands and the delay bound such that the total delay time is minimized. Given and derived parameters are listed in Table 1 and 2 respectively.

TABLE 1
NOTATION OF INPUT PARAMETERS.

G(V,E)	a directed graph, G, with set of nodes V and set of directed link E
v_i	a node; $v_i \in V$
e_k	a directed link $e_k = (v_x, v_y) \in E$, v_x is the start node, v_y is the end node of link e_k.; also denoted as e_{xy}
λ_{ij}	volume of traffic requests from v_i to v_j
Λ_k	volume of traffic requests from v_k to all other nodes. $\Lambda_k=\{\lambda_{ki}, \mid i=1,\ldots,\mid v\mid\}$
Λ	set of λ_{ij}, $\Lambda=\{\lambda_{ij}, \mid v_i, v_j \in V\}$
D	allowable delay time to transmit a packet from source to destination
$b(e_k)$	bandwidth of link e_k
$t(e_k)$	propagation delay time of link e_k
$p(v_k)$	processing capacity of node v_k

TABLE 2
NOTATION OF DERIVED PARAMETERS AND ROUTING RESULTS.

M_k	$\sum_{1}^{\mid V\mid} l_{ki}$, total volume of requested traffics starting from node v_k; the traffic volume of $\mid \Lambda_k \mid$
$S_k^{(n)}$	the selected routing path set(slice) of iteration n, corresponding to the request Λ_k
$P^{(n)}$	the selected routing path set(pasta) of iteration n, set of $S_k^{(n)}$.
μ_h	volume of traffics passing through link e_h, starting from v_x, and ending at v_y, also denoted as μ_{xy}
σ_k	volume of traffic passing through node v_k
U	set of μ_h, $U=\{\mu_h\}$
f_{ij}	the selected path for λ_{ij}, by the routing algorithm; $f_{ij}=v_i,e_{ii+1},v_{i+1},e_{i+1\ i+2},\cdots,e_{j-1j},v_j$
?	set of f_{ij}
$d(v_k)$	delay time caused by node v_k
$d(e_h)$	delay time caused by link e_h
$d(\Phi_{ij})$	total delay time along path set Φ_{ij}, $d(f_{ij}) = \sum_{e\in f_{ij}} d(e) + \sum_{v\in f_{ij}} d(v)$

The problem is then formulated as follows:

Find ?

$$\ni \min \sum_{f_{ij}\in\Phi} d(f_{ij}),$$

$$\text{s.t. } d(f_{ij}) < D, \forall f_{ij} \in \Phi . \qquad (1)$$

The delay bound of each traffic flow, D, could be variant without incurring a significant impact to the model. $d(\Phi_{ij})$ is the total delay time for a traffic flow; it is an accumulation of the delay time on all links and nodes along all the selected paths. The delay time occurred in the components of a real network is actually dependent on the stochastic behavior of the traffic and routing process. In reality, it is extremely difficult to solve a routing problem that takes stochastic behavior into account. Therefore, we take a compromised approach by relaxing the stochastic property of traffic and routing process in estimating of delay time on links and nodes.

We assume all traffics are of Constant Bit Rate (CBR) type and all resources (processing and link bandwidth) are proportionally shared by all traffic flows passing through. In this way, the load of each resource can be computed based on the total amount of traffics passing through that resource. Although this is a compromised model, it is more realistic as compared to the traditional fixed weight model. We hope it is a good approximation of a real network.

The problem can be easily proved NP-hard by reducing into a 0-1 knapsack problem. Therefore, we do not expect to find a polynomial-time optimal algorithm for

it. Instead, we designed a heuristic algorithm to find sub-optimal solutions. Furthermore, both objective and constraints are not simple functions of given parameters (delay time). Instead, they are result dependent variables. This makes the problem much more complicated.

B. Iterative Solution Approach

An iterative approach is taken to cope with the difficulty of load dependent delay time on nodes and links. In the first iteration, the delay time of every node is set to zero and the delay time of every link is set to its propagation delay time. In other iterations, the delay time of nodes and links are computed based on the result obtained in the previous iteration.

For convenience, the result obtained in an iteration is called a *pasta*. In each iteration, the problem is further divided into some number of sub-problems. Each routing solution (pasta) can be decomposed into a number of single root flow trees, named a *slice*. In such a tree, the root node is any node and the tree presents the flows generated from that root node and are forwarded to all other nodes. In each incremental step within an iteration, a slice corresponding to a request set $|\Lambda_k|$ is extracted from the pasta, recomputed using an algorithm similar to Dijkstra's, and superimposed back to the pasta. Thus, a pasta, the result of an iteration, is recomputed incrementally slice by slice.

After some number of iterations, hopefully, the delay time of each network component will be stabilized, and the routing solution obtained will be a stable solution. Termination is triggered in two conditions: when average path delay of two consecutive iterations is close within the predetermined value ε; or the number of iterations exceeds a given number. In the first condition, ε is defined as the difference of two consecutive iterations divided by the total path delay time of previous iteration as shown in Eq. 2:

$$e = \left[\sum d(f_{ij}) - \sum d(f_{lm})\right] \Big/ \sum d(f_{ij}) \quad | f_{ij} \in S_k^{(n)}, f_{lm} \in S_{k+1}^{(n)}. \quad (2)$$

We denote the result (pasta) obtained in the n-th iteration as $P^{(n)}$, the single root path tree (slice) corresponding to the Λ_k in the n-th iteration as $S_k^{(n)}$, and $P^{(n)} = \{S_1^{(n)} \oplus S_2^{(n)} \ldots \oplus S_k^{(n)}\}$, where $\oplus$ denotes a superposition. The iterative procedure is summarized in the followings:

(I) Initial condition

for all nodes and links, $\sigma = 0$, $\mu = 0$, $d(v)=0$, $d(e)=t(e)$;

$S_1^{(0)} = S_2^{(0)} = S_3^{(0)}, \ldots, = S_{|V|}^{(0)} = \{\}$; *//empty set*

$P^{(0)} = S_1^{(0)} \oplus S_2^{(0)} \oplus \cdots S_{|V|}^{(0)}$;

//$\oplus$ denotes superimposing a slice into a pasta

//Θ denotes removing a slice from a pasta

(II) First iteration

$P^{(1)} = \{\}$;

route Λ_1 based on ($P^{(0)} \Theta S_1^{(0)}$), to obtain $S_1^{(1)}$;

$P^{(1)} = P^{(1)} \oplus S_1^{(1)}$;

route Λ_2 based on ($P^{(0)} \Theta S_1^{(0)} \Theta S_2^{(0)} \oplus S_1^{(1)}$), to obtain $S_2^{(1)}$;

$P^{(1)} = P^{(1)} \oplus S_2^{(1)}$;

.

.

.

route $\Lambda_{|V|}$ based on ($P^{(0)} \Theta S_1^{(0)} \cdots \Theta S_{|V|}^{(0)} \oplus \cdots \oplus S_{|V|-1}^{(1)}$), to obtain $S_{|V|}^{(0)}$;

$P^{(1)} = P^{(1)} \oplus S_{|V|}^{(1)}$;

(III) On the k-th iteration:

$S^{(k)} = \{\}$;

for j? 1 to $|V|$

{

route Λ_j based on ($P^{(k-1)} \Theta \cdots S_2^{(k-1)} \cdots \Theta S_j^{(k-1)} \oplus \cdots \oplus S_{j-1}^{(k)}$), to obtain $S_j^{(k)}$;

$P^{(k)} = P^{(k)} \oplus S_j^{(k)}$;

}

(IV) Termination Conditions

When $P^{(M)} \approx P^{(M+1)}$ or the number of iteration exceeds a given number, terminate;

C. Estimation of Path Delay Time

The delay time of path f_{ij}, $d(f_{ij})$, consists of the delay time on all nodes and links in a path, which is $d(v_i)+d(e_{i\ i+1})+d(v_{i+1})+d(e_{i+1\ i+2})+\cdots+d(e_{j-1\ j})+d(v_j)$. μ_h and σ_k, are defined as the total volume of traffic flows passing through a link e_h and a node v_k, respectively and can be computed as follows:

$$m_h = \sum_{\substack{f_{ij}\in\Phi \\ e_h\in f_{ij}}} l_{ij}, \text{ and} \tag{3}$$

$$s_k = \sum_{\substack{f_{ij}\in\Phi \\ v_k\in f_{ij} \\ v_k\neq v_j}} l_{ij}. \tag{4}$$

1) Link Delay Time

$d(e_h)$ is the delay time of a flow of packets passing through link e_k, including transmission delay and propagation delay. μ_h is the total flows passing through e_h. As mentioned in Section III.A, we assume every traffic flow is a CBR and the bandwidth of a link is shared by all the traffic flows passing through that link. The queuing delay on the link, thus, can be ignored. Therefore, the delay time on a link for a flow is approximately the propagation delay time plus the total traffic flows divided by the bandwidth of that link, as shown in E.q. 5,

$$d(e_h) = m_h / b(e_h) + t(e_h) = (\sum_{\substack{f_{ij}\in\Phi \\ e_h\in f_{ij}}} l_{ij}) / b(e_h) + t(e_h) \tag{5}$$

Notice that the delay time of a link, $d(e_h)$, is independent of the size of the flow passing that link. All traffic flows passing the same link are delayed by the same amount of time.

2) Node Delay Time

$d(v_k)$ is the delay time caused by a node, v_k. Again, to simplify the delay time model, we assume all traffic flows passing a node are processed in time-sharing fashion, such that the $d(v_k)$ can be estimated as the total volume of traffics divided by the processing capacity of that node, as shown in E.q. 6:

$$d(v_k) = s_k / p(v_k) = (\sum_{\substack{f_{ij}\in\Phi \\ v_k\in f_{ij} \\ v_k\neq v_j}} l_{ij}) / p(v_k). \tag{6}$$

3) Path Delay Time

Thus, the delay time of a path Φ_{ij} is

$$d(f_{ij}) = \sum_{e_h\in f_{ij}} d(e_h) + \sum_{v_k\in f_{ij}} d(v_k). \tag{7}$$

4) Node Delay to Link Delay Conversion

We need an efficient algorithm to solve a single-source shortest path routing problem to obtain a *slice*. Unfortunately, current shortest path algorithms all assume zero weight on nodes such that they are not adequate to solve this problem even though the delay time of network components are all constant within each iteration. There are two approaches to solve this problem: either to develop a new algorithm that considers both node and link delay together or to convert node delays into link delays, and then apply a conventional shortest path algorithm to solve this problem. We choose the second approach for simplicity. In the rest of this section, the conversion of node weight into link weight will be illustrated. The transformed graph will be equivalent to the given graph in the sense of path delay time.

The delay time of a node is computed based on E.q. 6 in Section III.C.2, where the total traffic passing through a node is obtained by the summation of its outgoing traffic flows. Node delay time is added to propagation delay time of each incoming link. By doing so, we obtain another graph that has weights on its links only and is equivalent to the original graph with respect to the path delay time, as shown in the remaining of this section.

Given a node with a weight of m, that has two incoming links of weight w_1 and w_2, as well as two outgoing links of weight w_3 and w_4, as shown in Figure 2(a). We can transform the graph by connecting the incoming links to the outgoing links, with four internal links of weight m. When a traffic flow passes this node, no matter which incoming link it comes from or which outgoing link it selects to leave, it should suffer from the same delay time of m, as shown in Figure 2(b).

Consider a traffic flow passing through the node, the total weight sum is either $w_1 + m+w_3$, $w_1 + m+w_4$, or w_2+m+w_4, $w_2 + m+w_3$. Since weight m appears in all possible paths, it can be treated as a common link, where the incoming and outgoing links connect together. This is shown in Figure 2(c). Finally, we shift the node weight (m) to incoming links. The weights of incoming links are then changed to w_1+m and w_2+m respectively. The graph is then transformed into Figure 2 (d), where the weight of node is shifted to links.

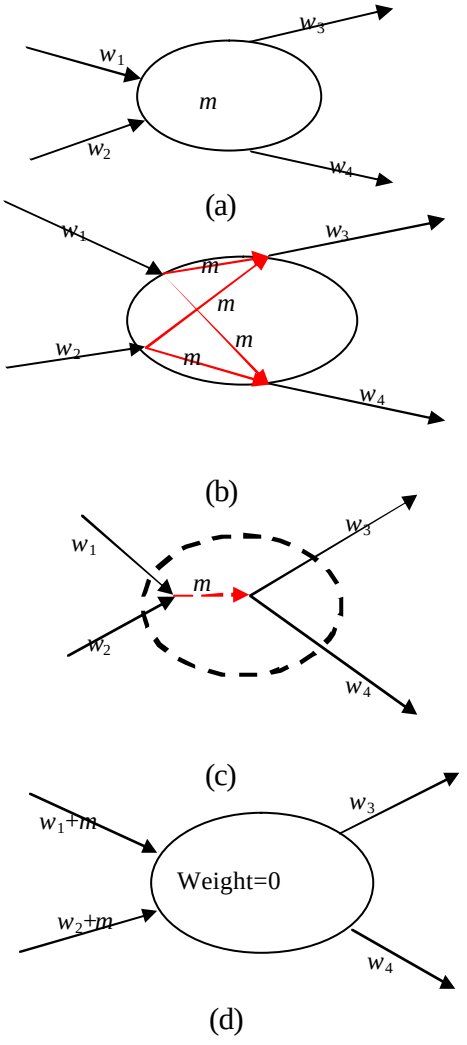


Fig. 2. Transformation of node delay to link delay.

IV. PERFORMANCE EVALUATION

It can be easily proved that the complexity of the KLONE algorithm is $N^4 \log N$ in the worst case. It is further evaluated by comparing with the traditional OSPF routing algorithm using a numerical simulation. Performance metrics are convergence speed, average path delay time, and goodput ratio. Convergence speed is evaluated by two different values: the number of iterations/slices when the convergence occurs (the average path delay time of two consecutive pastas differ by a predefined value, ε) divided by the total number of nodes (K_1/N and K_2/N). Goodput ratio is the ratio of total satisfied traffic requests to the total traffic requests. Average path delay time is the average time for a unit of request traffic passing through the network. It is computed as summation of the size of a traffic which be multiplied by the delay time on the selected path and then divided by the size of total traffic.

$$\frac{\sum [\,|l_{ij}| * d(f_{ij})\,]}{\sum |l_{ij}|} . \qquad (8)$$

A. Design of Experiments

We compared KLONE algorithm and OSPF algorithm in 64,000 different test instances, in the combinations of different number of nodes, network connectivity ratio, and different link bandwidth/processing capacity ratio. The range of link bandwidth was set from 0 to 400 Gbps, and propagation delays stayed below 20 ms. The number of nodes was set from 10 to 100 with a processing capacity in the range of Gbps. Connectivity is defined as $P/N*(N-1)$, where P is the number of links, and N is the number of nodes. The range of connectivity was set from 0 to 100 percents. The *BP ratio* is defined as *b(e)/p(v),* where *b(e)* is the link bandwidth and *p(v)* is the node processing capacity. We varied it from 1/300 to 1/1. The traffic coming into an edge node is assumed in an aggregated form. For a graph of N nodes, there are N*(N-1) requests, one from each node to every other node. The upper bound of delay time of all paths is set to D and D varies from 100 to 2000 ms.

B. Experiments and Results

The experiments and results will be presented in this section. Only a small portion of figures was shown due to the limit of the space. Most of the figures shown in this section are for the networks of size 50.

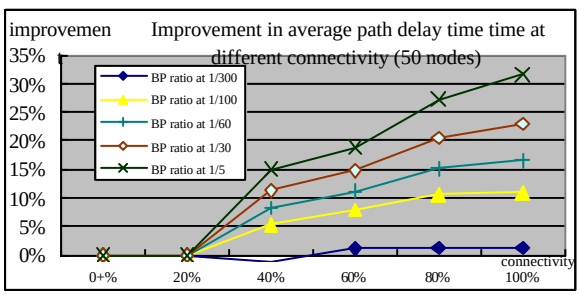
1) Exp-1: Convergence Test

We adjusted the following three parameters in the experiment to study their impact to the convergence speed: the BP ratio, the number of nodes and the connectivity. The results show that neither BP ratio nor the number of nodes has any impact to the convergence speed. On the other hand, we found that the convergence speed is dependent on the connectivity. This may be caused by two different reasons. First, higher connectivity may make a request easier to find a very good satisfied path, and then there is a higher probability to choose the same path in the succeeding iteration. On the other hand, lower connectivity may make a request having fewer paths to choose, so that the solution domain is much smaller and thus the convergence speed is faster. In more than 90% of the test instances, we found that the lines of both average path delay time and goodput ratio become stable after the $K_1=2/N$ and $K_2=2$.

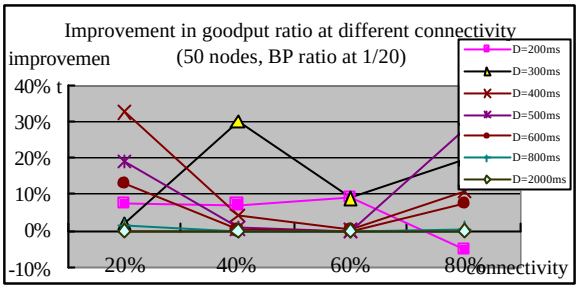
2) Exp-2: Sensitivity to Connectivity

Intuitively, higher connectivity implies more available paths between nodes. We studied the dependency between the connectivity and the two performance metrics: average path delay time and goodput ratio. We varied connectivity from 0% to 100% to see how average path delay time and goodput ratio are influenced.

In this experiment, we found that, at the same number of nodes, the average path delay time becomes smaller as the connectivity increases, as shown in Figure 3(a). The average path delay time improvement is defined as $(T_2-T_1)/T_2$, where T1 and T_2 are the average path delay time of KLONE algorithm and OSPF algorithm, respectively. The larger the value, the better the KLONE algorithm. In Figure 3(b), we show that at higher connectivity, KLONE algorithm has a higher goodput ratio than OSPF algorithm.



(a)

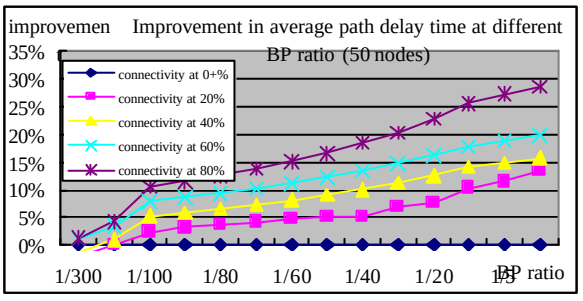


(b)

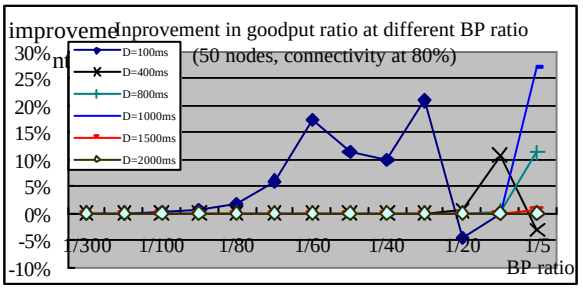
Fig. 3. Sensitivity to connectivity.

3) Exp-3: Sensitivity to BP Ratio

We varied the BP ratio from 1/300 to 1 to see the dependency between the BP ratio and the two performance metrics. We found that when the BP ratio increases, the improvement of average path delay time increases, as shown in Figure 4(a). This is consistent with our hypothesis that when the speed of links increases, an algorithm that concerns both link and node delay times might have a better performance than OSPF, which only concerns links delay times. We also compared the goodput ratio of KLONE algorithm and OSPF algorithm. We found that at different BP ratios, goodput ratio of KLONE algorithm is usually better than OSPF algorithm, as shown in Figure 4(b).



(a)

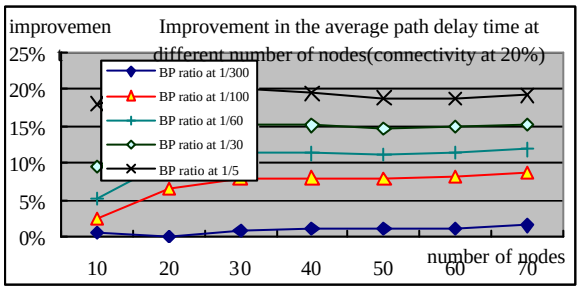


(b)

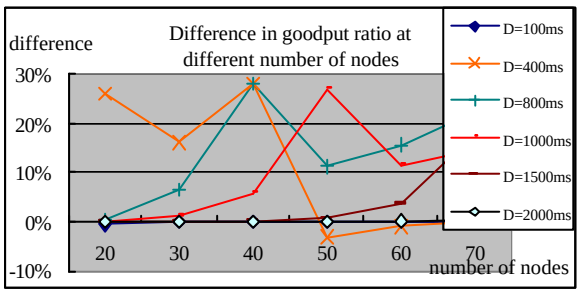
Fig. 4. Sensitivity to BP ratio.

4) Exp-4: Sensitivity to Number of Nodes

This experiment studied the dependency between the number of nodes and the delay time improvement. The number of nodes was varied from 20 to 70 in this experiment to see how it affects the performance. The performance improvement, which is defined in Section IV.B.2, is shown in Figure 5(a) in which the connectivity is 20%. The improvement increases as the number of nodes increases. Increasing the number of nodes will also increase the goodput ratio at the same delay bound, D, as shown in Figure 5(b). At different number of nodes, KLONE algorithm has a better goodput ratio than OSPF algorithm.



(a)

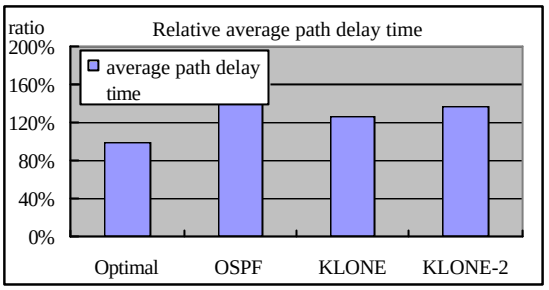


(b)

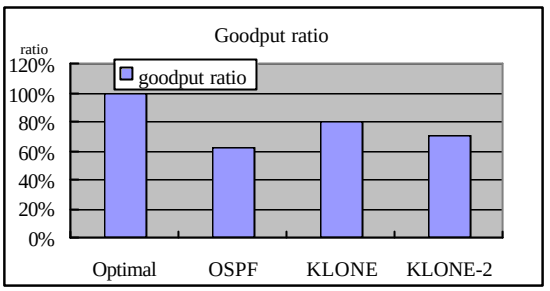
Fig. 5. Sensitivity to the number of nodes.

5) Comparison with Optimal Solution

In order to estimate the absolute performance of KLONE algorithm, we compared both algorithms with the optimal solution in a very small scale test instance, as shown in Figure 6, in which the number of nodes was set to 10, connectivity was set to 20%, BP ratio was set at 1/10. Figure 6 shows the comparison in the average path delay time. And we can know goodput ratio is not large obviously in the last between the KLONE with node ignored (KLONE-2) and original one.



(a)

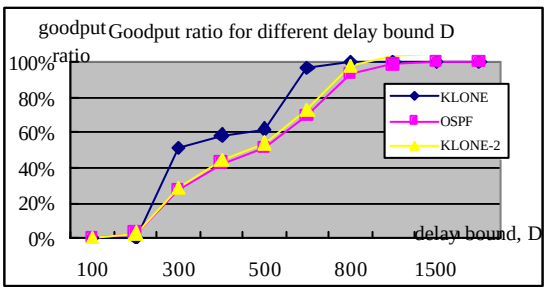


(b)

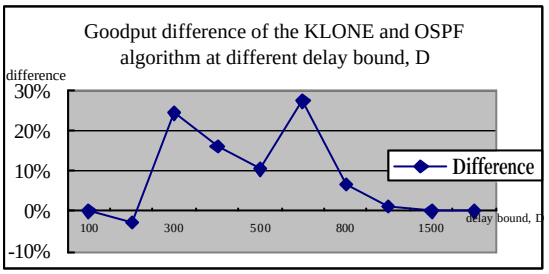
Fig. 6. Comparison with optimal solution.

6) Exceptions in Low Delay Bound

In the comparison of goodput ratio at different delay bound D, KLONE algorithm is generally better than OSPF and KLONE-2 algorithm. However, at some special range, such as small delay bound D, OSPF algorithm may have a better performance. This phenomenon may be caused by the fact that the path set generated by OSPF algorithm is less balance than that by KLONE algorithm. An unbalanced path set may have more low delay paths (and more long delay paths) such that have more paths satisfying a very low delay bound. This is shown in Figure 7, in which the number of nodes is 30, the connectivity is 60%, and the BP ratio is 1/5. In such instances, OSPF algorithm performs better than KLONE algorithm.



(a)

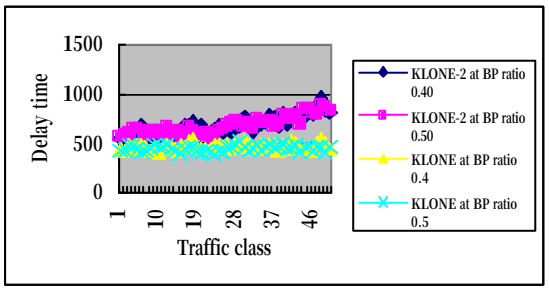


(b)

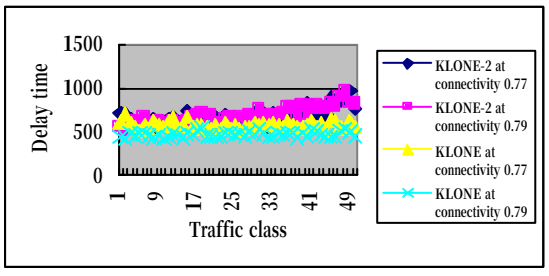
Fig. 7. An example of KLONE weakness in low delay bound.

7) Comparison of KLONE and KLONE-2

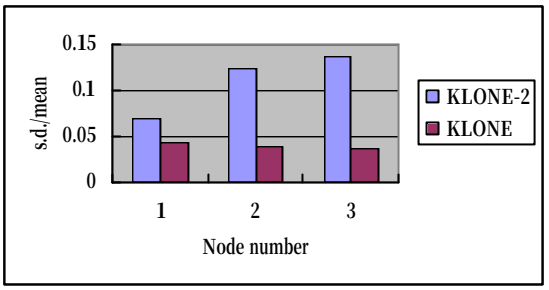
This phase shows the comparing result of the KLONE and KLONE-2. The delay time of traffic in the KLONE is small than KLONE-2 when small value of BP ratio and is small than another when traffic volume is large in the network environment of number of nodes is 50, the connectivity is nearing 78% and the BP ratio is 1/2. We shown in the (a) and (b) of the Figure8. Another experiment expresses when node tiny variation we can knowKLONE-2 would be large sensitivity in the same delay mean using relative dispersion to compare each other in the (c) of Figure8. The environment is in the number bode 50, 60 and 70, BP ratio at 1/2 and connectivity is nearing 77%.



(a)



(b)

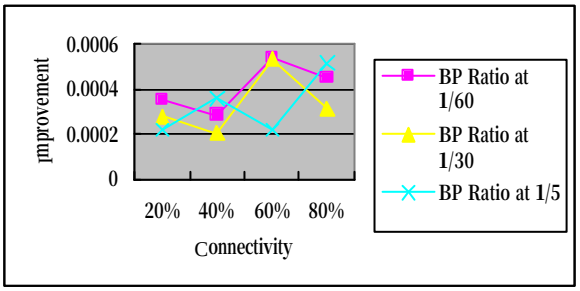


(c)

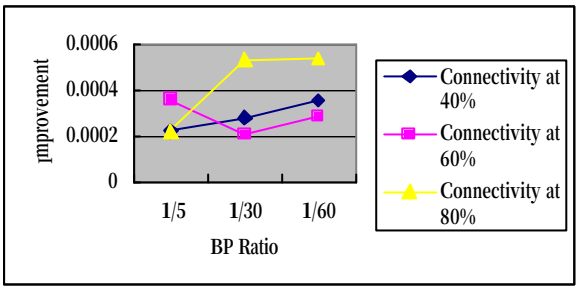
Fig. 8. Comparing the KLONE and itself without node.

8) Comparison of improvement between KLONE and KLONE-2

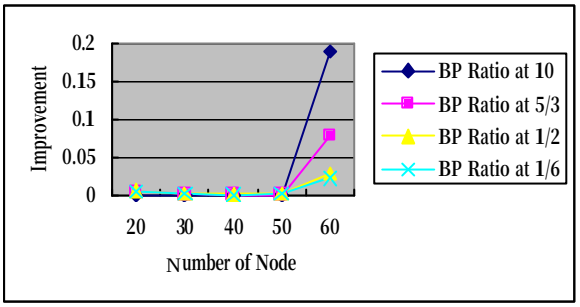
In this section we use Figure 9 to express the improvement between KLONE and KLONE-2 in comparing improvement. The definition of improvement is based on the aforementioned in the Exp2 and the BP ratio in Figure 9 is inversion of the aforementioned BP ration. In Figure 9 (a), we can find out the improvement will be obviously enhance when the delay generated from node is large. The situation when large improvement in the large BP ratio had been happened. The large difference is meaning that the path when selecting has number of nodes and KLONE-2 has not considering in. In (b) and (c), the improvement will be clearly in the large connectivity and plenty of node number in large BP ratio. To summary this section, when the environment has sufficient node number and the path includes many nodes would be used efficiently in KLONE.



(a)



(b)



(c)

Fig. 9. Improvement of the KLONE compared with KLONE-2.

V. CONCLUDING REMARK

With an intensive evaluation, we demonstrate the importance of the nodes delay in the routing path for high-speed packet switching networks. We hypothesized that a routing algorithm that considers the delay time of both nodes and links may have a better performance than that only considers link delay time in supporting delay sensitive services. We developed a flow-based routing algorithm, KLONE algorithm, which considers both link delay time and node delay time. The results of the evaluation show that KLONE algorithm could have a better performance than OSPF and KLONE-2 algorithm in most cases, with only a few exceptions. The hypothesis that considering node delay is important in high-speed packet-switching network is thus demonstrated.

This algorithm still has some weak points. First, KLONE algorithm may have worse goodput than OSPF algorithm when the delay bound is very low. Secondly, it does not support multi-path routing for the same traffic stream yet. Furthermore, a distributed version is needed in order to apply it onto real networks. In estimating the delay time of nodes and links, the traffic model should also be more realistic to include different traffic types, such as VBR, and in difference priorities.

REFERENCES

[1] 3rd Generation Partnership Project, ``Technical Specification Group Services and Systems Aspects; Architecture for an All IP network", 3GPP TR 23.922 version 1.0.0., October 1999.

[2] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell and J. McManus, ``Requirements for Traffic Engineering Over MPLS", RFC 2702, September 1999.

[3] Christophe Beaujean, ``Delay-Based Routing Issues in IP Networks", *contact GRADIENT CR/98/148*, May 2000.

[4] Dijkstra, E.W., ``A Note on Two Problems in Connection with Graphs", Numerische Math, vol. 1, March 1959, pp. 269-271.

[5] C. Hedrick, ``Routing Information Protocol", RFC 1058, June 1988.

[6] J. Moy, ``OSPF version 2", RFC 1583, March 1994.

[7] Douglas S. Reeves and Hussein F. Salama, ``A Distributed Algorithm for Delay-Constrained Unicast Routing", *IEEE Transaction on Network*, April 2000.

[8] A. S. Tanenbaum, ``Computer Networks, Third Edition", Prentice Hall, March 1996, pp. 345-366.

[9] Z. Wang and J. Crowcroft, ``Quality of Service Routing for Supporting Multimedia Applications", *IEEE Select on Communication*, September 1996.

[10] R. Wideyono, ``The Design and Evaluation of Routing Algorithms for Real-Time Channels", *International Computer Science Institute, Univ. of California at Berkeley, Tech Rep. ISCI TR-94-024*, June 1994.

[11] Ossama Younis and Sonia Fahmy, `` Constraint-Based Routing in the Internet: Basic Principles and Recent Research", *IEEE Communications Society Surveys & Tutorials*, vol 5, no. 1, 3Q 2003.