# A heuristic algorithm for bandwidth delay constrained routing

Cao Thai Phuong Thanh
Saigon University
Vietnam
Email: ctpthanh@sgu.edu.vn

Ha Hai Nam, Tran Cong Hung
Post & Telecommunications Institute of Technology
Vietnam
Email: namhh@ptit.edu.vn, conghung@ptithcm.edu.vn

***Abstract*—This paper focuses on routing algorithm with two quality of service constraints: bandwidth and delay. The former is a concave constraint while the later is additive. The goal of algorithm is to accept as many routing requests as possible. Therefore, the proposed algorithm reactively calculates link weights based on link bandwidths. Then, a heuristic idea is applied to Dijkstra in order to find a path satisfying delay constraint and having as little weight as possible. Simulated experiments demonstrate that our proposal have better performance than existing solutions in term of both acceptance ratio and computing time.**

***Keywords*—traffic engineering; bandwidth delay constrained routing; heuristic algorithm**

## I. Introduction

Network applications, especially multimedia, increasingly require quality of service (QoS). Those requirements are defined as QoS constraints in service-level agreements between customers and network providers. The providers aim to not only fulfil the agreements but also to use their network infrastructures effectively. Therefore, traffic engineering routing with QoS constraints has received much attention. The main objective of such routing algorithm is to accept as many routing requests as possible on the condition that each accepted route guarantees certain QoS metrics.

A popular QoS routing metric is bandwidth constraint addressed in [1], [2], [3], [4], [5]. In general, the bandwidth guaranteed routing algorithms dynamically calculate link weights for each requests. Then links that cannot satisfy the bandwidth requirement are removed before applying Dijkstra to select the least weight path. Different methods to assign link weights lead to different path decisions; i.e. different routing performances. However, bandwidth constraint checking is the same for all algorithms; and it is simple because bandwidth is a concave link metric. On the other hand, delay is an additive constraint which requires checking against the sum of delays of links along the path. As a result, the original Dijkstra cannot find the least weight path that also guarantees delay. In order to solve this problem, an extended Dijkstra algorithm [6] which finds the shortest path under another additive constraint can be applied. For instance, authors of [7] calculate link weights based on both bandwidth and delay, then use the extended Dijkstra to select routes. Inspired by work of [6] and [7], we: (a) present a simple but effective link weight , (b) modify the Dijkstra algorithm to find a bandwidth delay constrained path.

The rest of the paper is organized as follows. Section 2 defines and notates the routing problem as well as briefly describe existing solutions. Section 3 proposes a novel algorithm called Heuristic Routing Algorithm with Bandwidth Delay Constraints (HRABDC). Performance of HRABCDC is evaluated against other methods in section 4 by two metrics acceptance ratio and computing time. Finally, section 5 draws conclusions and outlines future work.

## II. Problem definition and related work

### A. Problem definition

A network is modelled as a graph $G(N, L)$. $N$ is a set of network nodes and $L$ is a set of links. Each link $l \in L$ has three properties: capacity ($c(l)$), residual bandwidth ($b(l)$) and delay ($d(l)$). Link capacity and delay are usually fixed, while residual bandwidth is varied based on traffic on that link. A routing request denoted as $r(i, e, \beta, \delta)$ demands a path from an ingress node $i$ to an egress node $e$ ($p_{ie}$) where all links must have residual bandwidths equal or greater than $\beta$ and sum of all link delays is equal or less than $\delta$. Moreover, if the request is accepted, it will reserve $\beta$ amount of bandwidth along the path $p_{ie}$. Consequently, link residual bandwidth is updated as the difference between the capacity and the sum of the bandwidth constraints of all paths assigned to that link.

It is assumed that routing requests arrive one at a time and network information such as residual bandwidth is available to algorithms. Moreover, ingress egress pairs are also assumed to be known. The routing objective is to maximize the number of accepted requests. Therefore, acceptance ratio is used to evaluate routing algorithms.

$$\text{acceptance ratio} = \frac{\text{number of accepted requests}}{\text{total number of requests}} \tag{1}$$

### B. Related work

[8] presents a simple and straightforward method namely Least Delay Algorithm (LDA). For each bandwidth delay constrained request, they remove unsatisfied bandwidth links and apply Dijkstra on link delays to find the least delay path. This path delay is then compared to delay constraints to decide whether the request is accepted or not. Although LDA is simple, it always selects the same path (the least delay one) for an ingress egress pair. Consequently, some parts of the network become under-utilized which in turn decrease the acceptance ratio.

In order to utilize network resources, dynamic link weights are calculated for each routing request. Maximum Delay

Weighted Capacity Routing Algorithm (MDWCRA) and Modified Maximum Delay Weighted Capacity Routing Algorithm (M-MDWCRA) [7] are examples of such reactive routing strategies. Those algorithms define delay-weighted capacity (DWC) as a measurement of how much request an ingress egress pair can satisfy.

$$DWC_{ie} = \sum_{LP^t_{ie} \in LP_{ie}} \frac{B^t_{ie}}{D^t_{ie}}$$

Where $LP_{ie} = \{LP^1_{ie}, .., LP^t_{ie}, .., LP^k_{ie}\}$ is a set of least delay paths of an ingress egress pair $ie$. In MDWCRA, a $t^{th}$ least delay path $LP^t_{ie}$ is computed after all links of $LP^1_{ie}, .., LP^{t-1}_{ie}$ are removed from the graph. Whereas in M-MDWCRA, only bottleneck links - links having the smallest residual bandwidths in the paths - are removed. The bottleneck bandwidth in $LP^t_{ie}$ is counted as the path bandwidth $B^t_{ie}$ and the sum of link delays is the path delay $D^t_{ie}$. Authors of [7] debate that the greater the DWC values are, the more requests can be satisfied, so the algorithms try to maximize DWC while selecting paths. Therefore, bottleneck links of the least delay paths are considered critical and assigned more weights because their usage decreases path bandwidths (i.e. decreases DWC). Furthermore, MDWCRA and M-MDWCRA apply extended Dijkstra [6] to compute the least weight path that also satisfies delay constraints.

The extended Dijkstra only works with positive integer delay values. Every node has a list of $(\delta + 1)$ entries $EN_n = \{en^0_n, .., en^\delta_n\}$. Each entry $en^k_n = \{d^k_n, w^k_n, m^k_n\}$ includes the indicator of previous node ($m^k_n$), weight ($w^k_n$) and delay ($d^k_n$). Moreover, entry delays have pre-assigned values $d^k_n = k, k = [0, 1, .., \delta]$, i.e. delays are also indexes of the list. Fig. 1 describes the extended Dijkstra algorithm used in MDWCRA and M-MDWCRA.

## III. Proposed algorithm

In reactive TE routing algorithms, purpose of dynamic link weights is to select different paths based on network states. Considering that the most important and changing state is link residual bandwidth, we proposes a simple weight calculation as follows:

$$w(l) = \frac{\text{used bandwidth}}{\text{residual bandwidth}} = \frac{c(l) - b(l)}{b(l)} \qquad (2)$$

The less bandwidth a link has, the more weight it is assigned so that the algorithm can avoid creating bottleneck. For instance, if two links have the same capacity, a link remaining less bandwidth (which leads to more weight) is likely to be avoided. Otherwise, if two links have the same residual, a link with higher capacity (which also leads to more weight) is protected because it has already been used more than the other.

In addition to link weight, the shortest path algorithm Dijkstra is modified to heuristically find path satisfying delay constraint and having as little weight as possible. It is applicable because the strict condition of the problem is delay whereas weights are used for dynamic route selection. Particularly, the algorithm called heuristic Dijkstra maintains the least delay values from all nodes to every egress node. Each node has one entry instead of $(\delta + 1)$ as the extended Dijkstra. An entry of node $n$, $en_n = \{d_n, w_n, m_n\}$, is updated on two conditions.

```
Input:
    A network $G(N, L)$
    A routing request $r(i, e, \beta, \delta)$
Output:
    The least weight path satisfied $\delta$ delay constraint
    or reject the request

function EXTENDED_DIJKSTRA($G, r$)
    INIT($G$)
    $Q = \{en^k_n | n \in N, k \in [0, \delta]\}$ // $en^k_n$ is an $k^{th}$ entry of node $n$
    while $Q \neq \emptyset$ do
        find $en^k_n \in Q$ such that $w^k_n = \min_{en \in Q}\{w\}$
        $Q = Q - en^k_n$
        if $w^k_n = \infty$ then
            break
        end if
        for each link $l$ from node $n$ to adjacent node $o$ do
            RELAX($Q, en^k_n, l, o, \delta$)
        end for
    end while
    GETPATH($G$)
end function

function INIT($G$)
    for each node $n$ $inN$ do
        for each entry $en^k_n | k \in [0..\delta]$ do
            $d^k_n = k$ // delay
            $w^k_n = \infty$ // weight
            $m^k_n = NULL$ // previous node
        end for
    end for
    for each entry $en^k_i | k \in [0..\delta]$ do // the ingress node $i$
        $w^k_i = 0$
    end for
end function
function RELAX($Q, en^k_n, l, o, \delta$)
    $new\_d = d^k_n + d(l)$
    if $en^{new\_d}_o \in Q$ and $new\_d \leq \delta$ then
        $new\_w = w^k_n + w(l)$
        if $new\_w < w^{new\_d}_o$ then
            $w^{new\_d}_o = new\_w$
            $m^{new\_d}_o = n$
        end if
    end if
end function
function GETPATH($G$)
    // consider entries of the egress node $e$
    find entry $en^k_e$ such that $w^k_e \neq \infty$ and $w^k_e = \min_{h \in [0, \delta]}\{w^h_e\}$
    if found $en^k_e$ then
        return path traced from $m^k_e$
    else
        reject the request
    end if
end function
```

Fig. 1: Extended Dijkstra algorithm for delay constrained routing [6], [7]

First, its weight must be greater than the sum of weight of the previous node entry and weight of the connecting link. Second, the delay constraint must be less or equal than the sum of three values: the previous node entry delay, the link delay and the least delay from $n$ to the egress node. The second condition assures that the to-be-updated path through node $n$ can satisfy the delay. The algorithm cannot guarantee the least weight path because a node entry only keeps the smallest weight path of the current iteration, but alternatives which might be better later are not traced. Figure 2 presents the heuristic Dijkstra.

```
Input:
    A network $G(N, L)$
    A routing request $r(i, e, \beta, \delta)$
    Least delays from all node to the egress $ld_{n\_e} | n \in N$
Output:
    A path satisfied $\delta$ delay constraint
    or reject the request

 1: function HEURISTIC_DIJKSTRA($G, r$)
 2:     INIT($G$)
 3:     $Q = \{en_n | n \in N\}$ // $en_n$ is an entry of node $n$
 4:     while $Q \neq \emptyset$ do
 5:         find $en_n \in Q$ such that $w_n = \min_{en \in Q} \{w\}$
 6:         $Q = Q - en_n$
 7:         if $w_n = \infty$ then
 8:             break
 9:         end if
10:         for each link $l$ from node $n$ to adjacent node $o$ do
11:             RELAX($Q, en_n, l, o, \delta, ld_{o\_e}$)
12:         end for
13:     end while
14:     GETPATH($G$)
15: end function

16: function INIT($G$)
17:     for each node $n$ $inN$ do
18:         $d_n = k$ // delay
19:         $w_n = \infty$ // weight
20:         $m_n = NULL$ // previous node
21:     end for
22:     $w_i = d_i = 0$ // the ingress node $i$
23: end function
24: function RELAX($Q, en_n, l, o, \delta, ld_{o\_e}$)
25:     $new\_w = w_n + w(l)$
26:     if $new\_w < w_o$ then
27:         $new\_d = d_n + d(l)$
28:         if $en_o \in Q$ and $(new\_d + ld_{o\_e}) \leq \delta$ then
29:             $w_o = new\_w$
30:             $d_o = new\_d$
31:             $m_o = n$
32:         end if
33:     end if
34: end function
35: function GETPATH($G$)
36:     if $m_e \neq NULL$ then
37:         return path traced from $m_e$
38:     else
39:         reject the request
40:     end if
41: end function
```

Fig. 2: Heuristic Dijkstra algorithm for delay constrained routing

General steps of the proposed algorithm namely Heuristic Routing Algorithm with Bandwidth Delay Constraints (HRABDC) are described in table I. There are offline and online phases. The offline phase is independent of routing requests and is conducted once the network topology changes. On the other hand, the online phase is the process of route selection upon receiving requests.

## IV. EVALUATION

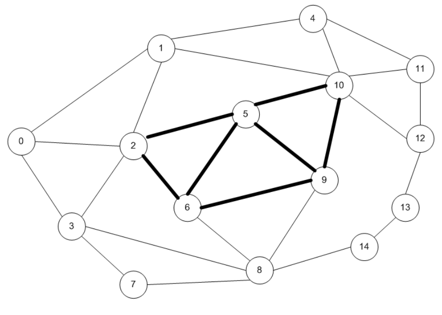
### A. Simulation setup

The proposed algorithm is evaluated against the Least Delay Algorithm (LDA) [8] and the Modified Maximum Delay Weighted Capacity Routing Algorithm (M-MDWCRA) [7]. According to the problem definition, evaluation metric is the acceptance ratio (equation (1)). Obviously, the higher the acceptance ratio is, the better the algorithm performs. The second evaluation metric is average computing time. A computing time of one request is computed when it arrives until it is accepted or rejected. Therefore, average computing time of routing algorithm should be low.

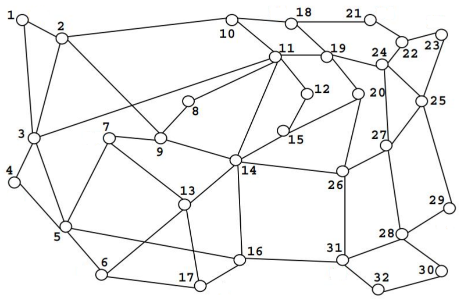
TABLE I: Heuristic Routing Algorithm with Bandwidth Delay Constraints (HRABDC)

| | |
|---|---|
| **Input** | A network $G(N, L)$<br>Traffic request $r(i, e, \beta, \delta)$ |
| **Output** | A selected path $p_{ie}$<br>Or reject the requests (i.e. no path) |
| **Offline phase** | 1. For each egress node $e$, calculate least delays from every node to $e$ ($ld_{n\_e}, n \in N$) |
| **Online phase** | For each $r(i, e, \beta, \delta)$:<br>1. Temporarily remove links whose $b(l) < \beta$<br>2. Calculate link weights $w(l)$ using equation (2)<br>3. Apply the heuristic Dijkstra algorithm to find the path from $i$ to $e$ with delay constraint $\delta$<br>If found: return the path $p_{ie}$ and update necessary information.<br>Else: return no path. |

In addition, we use the same network configuration as [7]. Particularly, figure 3(a) shows the first topology called MIRA with two types of capacities: 4800 (thick links) and 1200 (thin ones) bandwidth units. Meanwhile, figure 3(b) shows the second ANSNET topology with a link capacity of 1200 units. MIRA network has four ingress egress pairs: (0, 12), (4, 8), (3, 1), (4, 14) and ANSNET has five ones: (1, 29), (18, 6), (3, 23), (7, 31), (21, 17). Link delays of both topologies are uniformly distributed within $[5, 11]$ time units.



(a) MIRA topology



(b) ASNET topology

Fig. 3: Simulation network topologies

Routing test sets follow scenarios of static and dynamic requests. In the former scenario, requests arrive at the same rate and will statically hold bandwidth resource if they are accepted. On the other hand, arrival times of dynamic requests are randomized according to the Possion distribution with mean $\lambda$ requests per time unit. Moreover, if accepted they will hold bandwidth in mean $\mu$ time units of Exponential distribution. Each static and dynamic test set has 700 and 2000 requests respectively.

### *B. Experimental results & Analysis*

Routing requests are randomly generated with various bandwidth and delay constraints. Table II presents performance results of three evaluated algorithm (LDA, M-MDWCRA, and HRABDC) with following specification:

- On MIRA topology, bandwidth is required between four values of 10, 15, 20, and 30 units. Delay constraint is generated in range $[36, 70]$ which is identified based on mean and standard deviation of all path of ingress egress pairs. Moreover, dynamic requests have arrival times with Possion $\lambda = 60$ and holding times with Exponential $\mu = 20$.
- On ANSNET topology, required bandwidth values and delay range are $\{10, 20, 30, 40, 50\}$ and $[100, 155]$ respectively. Time parameters of dynamic requests are $\lambda = 40$ and $\mu = 10$.

TABLE II: Results of acceptance ratio (in percent) - computation time (in miliseconds) subject to number of requests (NoR)

(a) Static scenario on MIRA

| NoR | **LDA** | **M-MDWCRA** | **HRABDC** |
|---|---|---|---|
| **100** | 100- 0.15 | 100- 11.91 | 100- 0.38 |
| **500** | 84.60- 0.08 | 87.20- 10.66 | 86.80- 0.24 |
| **700** | 60.43- 0.08 | 68.43- 10.55 | 67.57- 0.23 |

(b) Dynamic scenario on MIRA

| NoR | **LDA** | **M-MDWCRA** | **HRABDC** |
|---|---|---|---|
| **100** | 100- 0.15 | 100- 11.52 | 100- 0.36 |
| **1000** | 75.00- 0.08 | 75.00- 11.34 | 78.90- 0.24 |
| **2000** | 60.50- 0.07 | 64.20- 11.78 | 64.65- 0.23 |

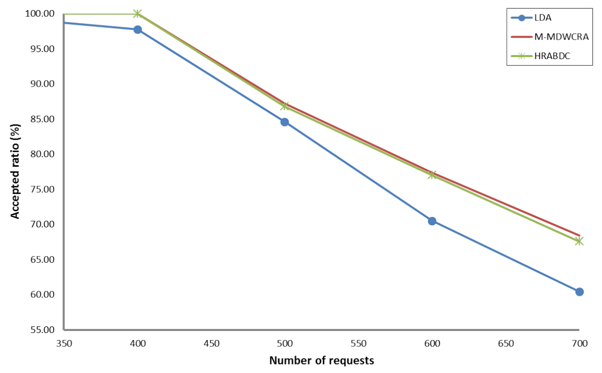
(c) Static scenario on ANSNET

| NoR | **LDA** | **M-MDWCRA** | **HRABDC** |
|---|---|---|---|
| **100** | 100- 0.24 | 100- 257.90 | 100- 0.55 |
| **500** | 62.80- 0.17 | 61.80- 243.71 | 64.80- 0.44 |
| **700** | 44.86- 0.16 | 44.14- 242.20 | 46.20- 0.42 |

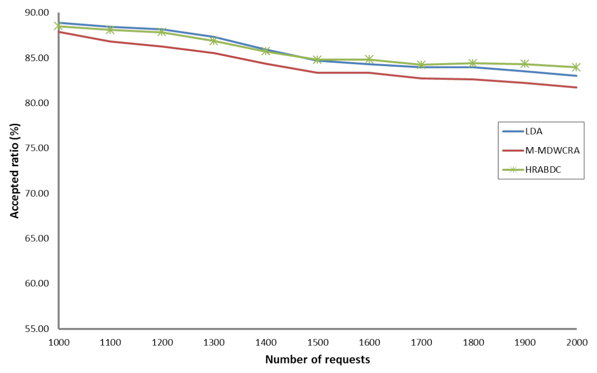
(d) Dynamic scenario on ANSNET

| NoR | **LDA** | **M-MDWCRA** | **HRABDC** |
|---|---|---|---|
| **100** | 100- 0.23 | 100- 251.84 | 100- 0.55 |
| **1000** | 88.90- 0.15 | 87.00- 245.89 | 88.50- 0.43 |
| **2000** | 83.00- 0.15 | 81.75- 248.52 | 83.95- 0.42 |

Figure 1 presents a comparison of acceptance ratios. LDA accepts much less requests than the others (e.g. 60.43% compares to 68.43% of M-MDWCRA and 67.57% of HRABDC - figure 4(a)) on MIRA topology. However, LDA performance is improved and even better than M-MDWCRA in some experiments on ANSNET. For instance, figure 4(b) shows that acceptance ratio of LDA is 1.25% higher than result of M-MDWCRA. It is because MIRA topo has few links and ingress egress pairs share common paths that the least delay selection of LDA considerably reduces the performance. Otherwise, there are many links and paths on ANSNET so LDA performs well. Nevertheless, the proposed algorithm HRABDC gains the highest acceptance ratio in many experiments such as results in table II(b), II(c), and II(d). In other cases, performance of HRABDC is also close to the best one. Table II(a) is an example where HRABDC accepts only 0.86% less than M-MDWCRA.



(a) Static scenario on MIRA

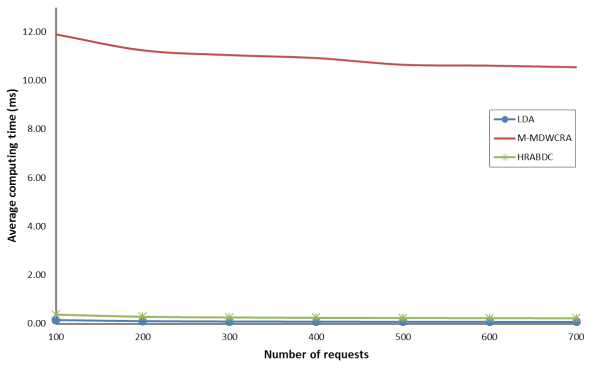


(b) Dynamic scenario on ANSNET

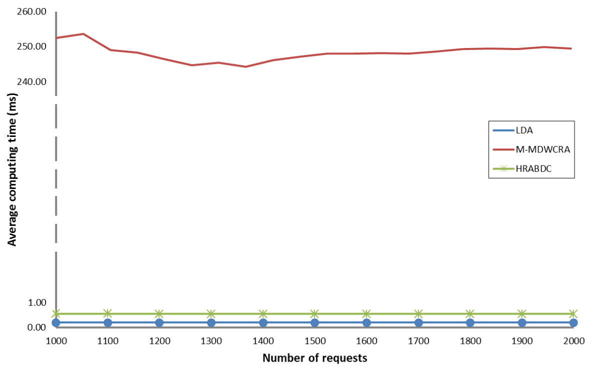
Fig. 4: Compare of acceptance ratio

Moving to complexities of the algorithms, LDA has two steps of browsing links for bandwidth constraint and applying Dijkstra for delay constraint. Therefore, LDA has complexity of $O(|L|+|N|^2)$ where $|L|$ is the number of links and $|N|$ is the number of network nodes. In addition to checking bandwidth, HRABDC calculates link weight by capacity and residual of the link itself. The heuristic Dijkstra in HRABDC also consider one entry per node so the complexity of HRABDC is $O(2.|L|+|N|^2)$. However, M-MDWCRA has high complexity of $O(|N|^2.|IE|.\log|N| + \delta^2.|N|^2)$ where $|IE|$ is the number of ingress egress pairs (details are presented in [7]). Differences between algorithm complexities are clearly demonstrated by average computing time.

Figure 5 plots the average computing time as a function of the number of requests. On MIRA topology (figure 5(a)), M-MDWCRA takes average of 10.55 ms to compute paths. It is about 45 times longer than 0.23 ms results of HRABDC. Whereas, HRABDC is slightly slower than LDA whose average computing time is 0.08 ms. On ANSNET which has more links than MIRA topology, computing time of HRABDC is also closely compatible with LDA's. On the other hand, M-MDWCRA becomes extremely slow due to the large number of paths and especially the large values of delay constraint. In figure 5(b), LDA has the lowest average computing time around 0.15 ms, next is HRABDC with about 0.42 ms. M-MDWCRA takes an average of 250 ms which is dramatically higher than computing times of the others.



(a) Static scenario on MIRA

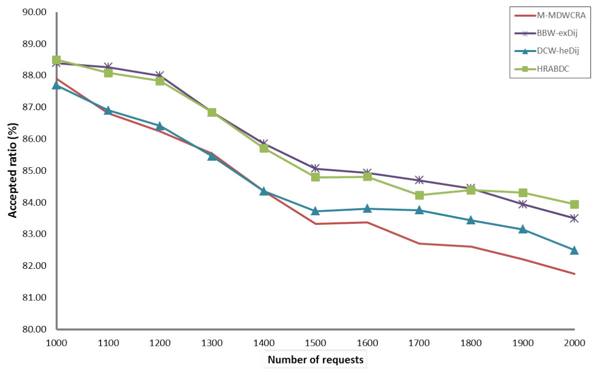


(b) Dynamic scenario on ANSNET

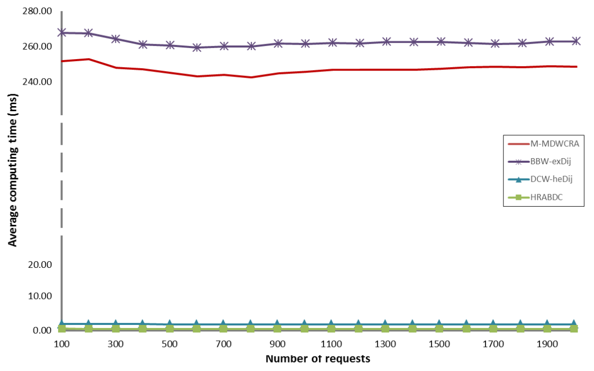
Fig. 5: Compare of average computing time

Further experiments are conducted to investigate dynamic weights and the path finding method of HRABDC. In particular, bandwidth-based link weight of HRABDC is combined to the extended Dijkstra so the combination is called BBW-exDij. Vice versa, the heuristic Dijkstra is also applied on the DWC-based weight of M-MDWCRA to create a routing strategy named DCW-heDij. Figure 6 shows a comparison of acceptance ratio and average computing time of M-MDWCRA, BBW-exDij, DCW-heDij, and HRABDC from a dynamic test on ANSNET topology.

The most noticeable outcome is that the extended Dijkstra consumes much more time than the heuristic one (more than hundreds times longer as shown in figure 6(b)). It is because the extended Dijksktra considers $(\delta + 1)$ entries for every node while there is only 1 entry per node in case of the heuristic method. Moreover, although the heuristic Dijkstra cannot guarantee to find the least weight path, it does not decrease the acceptance ratio. For example, figure 6(a) shows that M-MDWCRA and DCW-heDij which use the same DWC-based weights but different path finding methods have similar acceptance ratios (81.75% of M-MDWCRA and 82.30% of DCW-heDij). On the other hand, the bandwidth-based link weight of HRABDC is not only simple and easy to determine but also very helpful for dynamic path selection. Specifically, HRABDC and BBW-exDij are the best and the second best algorithms with respect to acceptance ratio in figure 6(a).



(a) Acceptance ratio of a dynamic test



(b) Average computing time of the above test

Fig. 6: Performance result of the combinations between weight calculation and path finding methods on ANSNET

## V. Conclusion

This paper proposes a novel heuristic routing algorithm with bandwidth and delay constraints (HRABDC). When a routing request arrives, link weight is simply calculated based on the capacity and residual bandwidth of the link. The purpose of dynamic link weights is to reactively select paths so that there is suffice resource for future requests. Considering that the least weight path is not the compulsory condition of the problem, the shortest path Dijkstra algorithm is modified to determine the path satisfied delay constraint with as least weight as possible. Experiments on different network topologies with various test sets demonstrate that our proposal achieves not only high acceptance ratio but also low computing time. Future work will investigate performance of the proposed algorithm in case of network changes. Furthermore, the heuristic ideas of weight calculation and path finding could be improved to satisfy more requests.

## References

[1] K. Kar, M. Kodialam, and T. Lakshman, "Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 12, pp. 2566–2579, 2000.

[2] R. Boutaba, W. Szeto, and Y. Iraqi, "DORA: efficient routing for MPLS traffic engineering," *Journal of Network and Systems Management*, vol. 10, pp. 309–325, 2002.

[3] B. J. Oommen, S. Misra, and O. Granmo, "Routing Bandwidth-Guaranteed paths in MPLS traffic engineering: A multiple race track learning approach," *IEEE Transactions on Computers*, vol. 56, no. 7, pp. 959–976, 2007.

[4] A. Alidadi, M. Mahdavi, and M. R. Hashmi, "A new low-complexity QoS routing algorithm for MPLS traffic engineering," in *2009 IEEE 9th Malaysia International Conference on Communications (MICC)*, Kuala Lumpur, Malaysia, 2009, pp. 205–210.

[5] C. T. P. Thanh, H. H. Nam, and T. C. Hung, "Low complexity bandwidth guaranteed routing algorithms using path holding time," in *Proceedings of the 2013 5th International Conference of Soft Computing and Pattern Recognition (SoCPaR 2013)*. Ha Noi, Vietnam: IEEE, 2013, pp. 92–97.

[6] S. Chen, "Routing support for providing guaranteed end-to-end quality-of-service," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Champaign, May 1999.

[7] Y. Yang, L. Zhang, J. K. Muppala, and S. T. Chanson, "Bandwidth delay constrained routing algorithms," *Computer Networks*, vol. 42, no. 4, pp. 503 – 520, 2003.

[8] Z. Wang and J. Crowcroft, "Bandwidth-delay based routing algorithms," in *Global Telecommunications Conference, 1995. GLOBECOM '95., IEEE*, vol. 3, Nov 1995, pp. 2129–2133.