A Heuristic for Global Coordination in *MPLS* Bandwidth Constrained Path Selection

Ning Wang [a], Yin Lu [b], George Pavlou [a]
a: Center for Communication Systems Research,
University of Surrey, Guildford, United Kingdom
b: Dept.of Computer Science and Engineering,
Southeast University, Nanjing, P.R. China
{N.Wang, G.Pavlou}@eim.surrey.ac.uk, luyinn@sina.com

Abstract
Multi-Protocol Label Switching (*MPLS*) has been considered to be a promising solution to achieve end-to-end *QoS* guarantees in Differentiated Services (*DiffServ*) domains [1]. Based on the Service Level Specification (*SLS*) between customers and the ISP, traffic forecast mechanism is able to predict traffic demands between ingress-egress routers, and hence bandwidth guaranteed *LSPs* can be set up accordingly through the *DiffServ* domain. In this paper, we address the problem of computing multiple *LSPs* with heterogeneous bandwidth requirements, while the overall network link cost is optimized. We first prove that finding a set of feasible *LSPs* with bandwidth constrained is NP-complete, and then propose an efficient heuristic with global network resource coordination over individual traffic aggregates. By simulation we show that the proposed coordinated path section (*CPS*) scheme obtains better overall *LSP* cost and lower bandwidth consumption compared with existing bandwidth constrained routing algorithms.

1. Introduction

Differentiated Services (DiffServ) has been proposed and incrementally deployed on the Internet for *QoS* provisioning in a scalable fashion. In contrast with the scenario of Integrated Services (*IntServ*) and *RSVP*, data traffic in DiffServ networks is classified into finite class of services at the edge of the network, and packets belonged to the same *QoS* class are treated aggregately inside the domain based on Per Hop Behavior (*PHB*). On the other hand, how to achieve end-to-end *QoS* guarantee raises a new challenge for DiffServ technology. Recently some research work has been focusing on the deployment of Multi-Protocol Label Switching (*MPLS*) inside DiffServ networks to satisfy quantitative *QoS* guarantees. *MPLS* is a new switching technology to forward packets according to a short, fixed length label, and to set up explicit Label Switching Paths (*LSPs*) through the network. These *LSPs* carry flow aggregates based on Forwarding Equivalence Classes (*FEC*) generated by classifying the data packets at the ingress routers of the *MPLS* networks. This basic characteristic provides promising solutions for supporting quantitative *QoS* requirements in DiffServ networks by means of setting up explicit *LSP* for each traffic aggregate.

It has been argued that, if *QoS* constraints such as delay and loss probability are to be incorporated in Service Level Agreements (*SLAs*) in the DiffServ network, the most practical way is to convey them into effective bandwidth requirement for *LSPs* [2]. In this sense, the strategy of setting up explicit *LSPs* for aggregated traffic trunks can be modeled as the bandwidth constrained routing problem. Traditional Shorted Path (*SP*) algorithm based on Dijkstra's approach has been proved not always efficient in *QoS* routing semantics when additional metric constraints are involved, e.g., delay, loss probability etc. In [3] it has been proved that routing with two additive metrics is an *NP-complete* problem. On the other hand, routing with one additive metric and bandwidth constraint can be performed in polynomial time by eliminating network links that do not have sufficient bandwidth capacity. In literature, several classic heuristic algorithms have addressed the problem of bandwidth constrained routing.

A. Simple Bandwidth Guaranteed Routing [3]

This is the most straightforward approach among all the bandwidth constrained routing algorithms. The main idea is that, before setting up any paths, all the links that do not have bandwidth capacity as required by the traffic trunk are eliminated from the network. Thereafter, Dijkstra's algorithm is used to compute a feasible shortest path. Simple as it is, this algorithm only focuses on the additive metric, but does not provide any mechanism for bandwidth optimization purpose. In fact, if multiple paths are to be computed for different source-destination pairs, the overall performance regarding this additive metric might not be optimal, as it will be shown in an example in section 3.3.

B. Widest-Shortest Path (*WSP*) and Shorted Widest Path (*SWP*) [3, 4]

Both *WSP* and *SWP* try to optimize the residual bandwidth distribution so that the network resource can accommodate more incoming traffic trunks in future. In *WSP*, when there exist multiple shortest paths with equal number of hops between a source-destination pair, the one with the maximum "bottleneck" bandwidth will be selected. In contrast, *SWP* first explores a path with maximum bandwidth among all feasible paths, and if there exists a tie, the one with the minimum number of hops will be selected. It is noted that the metric of hop counts can be replaced with any additive metric, e.g., link cost (if the optimization objective is to minimize overall cost, and this is typically useful for multicast routing), or delay (if the optimization objective is focused on end-to-end delay).

C. Bandwidth-inversion Shortest Path (*BSP*) [5]

In the *BSP* algorithm, the Dijkstra's shortest path algorithm is still used, but the weight of each link (*i, j*) is defined as follows:

$$w(i, j) = \frac{1}{b_{ij}}$$

where b_{ij} is the residual bandwidth capacity of link (*i, j*). The total weight of a path *P* is expressed as

$$w(P < v_1, v_2 ... v_n >) = \sum_{i=1}^{n-1} \frac{1}{b_{i,i+1}}$$

It can be found that the *BSP* algorithm considers exclusively the link metric of bandwidth, and although traffic trunks can follow the lightest path, other network metrics will not be explored simultaneously.

D. Enhanced Bandwidth-inversion Shortest Path (*EBSP*) [6]

Having realized that the performance of the *BSP* algorithm is not stable, and the fact that wide paths but with very long distance might be introduced, Wang and Nahrstedt proposed an enhanced *BSP* algorithm to improve the situation. In *EBSP* the new weight of a path is changed into

$$w(P < v_1, v_2 ... v_n >) = \sum_{i=1}^{n-1} \frac{2^{i-1}}{b_{i-1,i}}$$

It is noted that the new weight function is no longer static, and this characteristic of dynamic metric according to hop counts reduces the probability of obtaining "longer" paths.

All the algorithms introduced above are from the viewpoint of individual source-destination pairs, i.e., they are a type of on-demand schemes that do not need any information on other traffic trunks passing through the network. Hence, none of the above algorithms consider a "global" coordination on bandwidth allocation among all of the concurrent traffic demands. In this scenario, individual so-called "optimized paths" are only seen by their own corresponding source-destination pairs. However, it is not always the case that individual traffic trunks are blind to each other so that global coordination is impossible. In DiffServ networks, Service Level Agreements are made between customers and *ISP*, and the corresponding *SLS* specifies the detailed parameters in the agreement, including the location of ingress and egress pair, aggregated bandwidth requirements etc. Based on these *SLSs*, the *ISP* is able to grasp a complete map of its required traffic demands at the edge of the DiffServ domain, and hence simultaneous computation of all the *LSPs* between ingress-egress routers becomes possible, due to the mechanism of Traffic Forecasting (*TF*) from the *SLAs* with customers.

When *m* ($m \geq 2$) bandwidth constrained *LSPs* are to be set up, network bandwidth can be allocated to individual traffic trunks with global resource coordination. Given the bandwidth capacity of the network, the problem of finding a feasible solution for setting up individual bandwidth guaranteed *LSPs* is *NP-complete*, and we name this problem Global Bandwidth Constrained Routing (*GBCR*). Given additional metrics such as link cost and delay, the problem of path computation will become more complicated. Since link metrics such as delay and loss probability can be conveyed into bandwidth requirement, we will focus on another important metric, namely overall link cost, which can be defined according to the need of the *ISP*. Typically, hop count is a special case for link cost by setting the cost of all links to 1. In this sense, the overall cost will directly reflect network resource consumption, since *LSPs* with minimum total number of hops consume least network bandwidth. In this paper we will propose a new offline heuristic algorithm for optimization of overall link cost among multiple *LSPs* through the DiffServ domain. Although we only consider setting up point-to-point *LSPs* for unicast routing, the proposed approach can also be adapted for multicast paradigms in the sense that individual paths are replaced with trees with multiple ending point at egress routers.

2. Problem Formulation

A DiffServ capable network can be modeled as a directed graph $G = (V, E)$, with a boundary node set *D* ($D \subset V$ and $|D| = m$). Two types of metrics are associated with each link (i, j) in the graph, namely link cost c_{ij} and bandwidth capacity b_{ij}. (c_{ij} can be the parameter such as administrative cost or monetary cost , or simply hop count) For simplicity we assume that cost is a symmetric parameter, i.e., $c_{ij} = c_{ji}$ for any link (i, j) in the network. Aggregated traffic demands for the network can be identified with a $m \times m$ sized traffic trunk matrix *T*, and each element $t_{uv} \in T$ stands for the traffic demand from the ingress node *u* to the ingress node *v*, where nodes $u, v \in D$. We suppose that the number of non-zero elements in *T* is *L*. Global Bandwidth Constrained Routing (*GBCR*) is to find a set of paths $\{P_1, P_2, ... P_L\}$, for each of the non-zero element $t_k \in T$, and satisfies the following requirements:

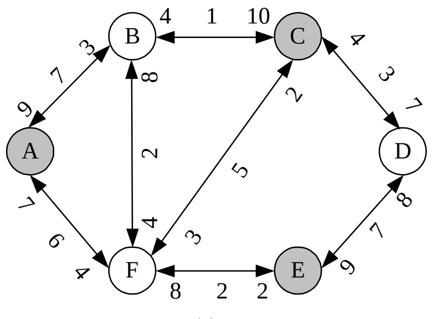
(1) Minimize $\sum_{k=1}^{L} \sum_{(i,j)\in E} c_{ij} X_{ij}^{k}$, $i \in V, j \in V$

(2) For each link $(i, j) \in E$, $\sum_{k=1}^{L} t_k X_{ij}^{k} \leq b_{ij}$

where

$$X_{ij}^{k} = \begin{cases} 1 & \text{if } (i, j) \in P_k \\ 0 & \text{otherwise} \end{cases}$$

The first constraint ensures that the total cost of the generated paths is minimized, while the second constraint is to ensure that the total bandwidth allocated to traffic trunks passing this link does not exceed its bandwidth capacity. Figure 1 shows a network model and a typical traffic matrix. In Figure 1(a), the number in the middle is the cost associated with the link and the numbers by the side is the bandwidth capacity along this direction, e.g., the bandwidth capacity of link (*A*, *B*) is 3 and that of link (*B*, *A*) is 9. The nodes filled with gray color are boundary nodes, and the traffic demands between each of them are shown in Figure 1(b). From the traffic matrix we can see that there are totally 5 traffic trunks between three boundary nodes, e.g., traffic with 3 units of bandwidth will be injected into the network at node *A* and eventually leave it at node *E*.



(a)

Traffic Matrix

	A	*C*	*E*
A	---	1	3
C	0	---	4
E	2	5	---

(b)

Figure 1 Network model and Traffic Matrix

A set of *LSPs* {P_1, P_2,…P_L} that satisfy constraint (2) is called a *feasible solution* to the Global Bandwidth Constraint Routing (*GBCR*) problem. A feasible solution to *GBCR* based on the network model and traffic matrix in Figure 1 is the path set {*Aà Bà C, Aà Fà Bà Cà Dà E, Cà Dà E, Eà Fà A, Eà Fà Bà C*} with total cost of 50. After setting up a set of paths for the traffic trunks, we find that the bandwidth of links (*C, D*) and (*F, B*) is decreased to 0, and we name this types of links *saturated links*. Now we give the proof that finding a feasible solution to *GBCR* is *NP-complete*. This indicates that there does not exist any algorithm that can compute a set of feasible path with the bandwidth constraint (2) in polynomial time.

Theorem 1: The problem of finding a feasible solution to *GBCR* is *NP-complete*.

Proof: In order to prove the problem to be *NP-complete*, we perform a bi-directional transformation between the PARTITION problem [7] and this problem in polynomial time.

First let *A* with elements { $a_1, a_{2,} ...a_k$ } be an instance of the PATITION problem. Each element a_i has a size $S(a_i)$. Let $S = \sum_{a\in A} S(a)$. We construct an instance of *GBCR* problem as shown in Figure 2. The directed network graph $G = (V, E)$ has a boundary node set $D = \{X_0, X_1, X_2, ...X_{k-1}, X_k\}$ and 5 additional internal nodes, namely R1 to R5. The directed link set *E* is as follows:

$$E = \{(X_i, R4) : i = 1..k)\} \cup \{(X_i, R5) : i = 1..k)\}$$
$$\cup \{(R4, R2), (R5, R3), (R2, R1), (R3, R1), (R1, X_0)\}$$

The directed link (R4, R2) and (R5, R3) have a bandwidth capacity of $S/2$ respectively. All the other links are assumed to have infinitive amount of bandwidth capacity. The traffic matrix contains *k* non-zero elements, namely from $X_i (i = 1...k)$ to X_0. The bandwidth demand of traffic trunk (X_i, X_0) is set to $S(a_i)$ units. It is easy to see that this transformation can be performed within polynomial time.

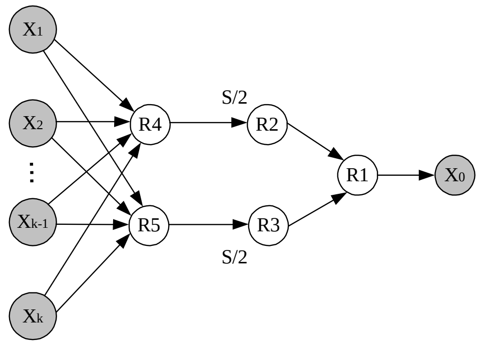


Figure 2 *NP-completeness* proof for finding feasible solutions to *GBCR*

Next we show that a feasible solution for *GBCR* exists if and only if set *A* has a partition.

(a) PARTITION à *GBCR*

First suppose we have a solution, say A_1, to the PARTITION problem, i.e.,

$$\sum_{a\in A_1} S(a) = \sum_{a\in A-A_1} S(a) = S/2$$

For each element $a_i \in A_1$, let the corresponding traffic trunk from ingress node X_i traverse the directed link (R4, R2) to reach the egress X_0. All other traffic trunks from ingress node X_i with the corresponding element $a_i \in A - A_1$ traverse the directed link (R5, R3) to reach the egress X_0. Then the sum of the bandwidth usage of the directed links (R4, R2) and (R5, R3) are exactly S/2.Thus we have obtained a feasible solution to the *GBCR* problem.

(b) GBCR à PARTITION

Next suppose that we have a feasible solution for *GBCR* based on the network graph in Figure 2. Note that each of the traffic trunks from ingress nodes $X_i (i = 1,2,...k)$ must either traverse directed link (R4, R2) or (R5, R3) to reach X_0. For each traffic trunk from X_i that traverses the link (R4, R2) to reach X_0, we let $X_i \in U, (U \subset D)$. Since the bandwidth of link (R4, R2) is S/2, we have:

$$\sum_{u\in U} S(u) \le S/2 \qquad (1)$$

Likewise,

$$\sum_{v\in D-U-\{X_0\}} S(v) \le S/2 \qquad (2)$$

But we note that

$$\sum_{u\in U} S(u) = \sum_{a\in A} S(a) - \sum_{v\in D-U-\{X_0\}} S(v) \ge \sum_{a\in A} S(a) - S/2 = S/2 \qquad (3)$$

Hence it follows from (1) and (3) we get

$$\sum_{u \in U} S(u) = S/2 \qquad (4)$$

This also implies that

$$\sum_{v \in D-U-\{X_0\}} S(v) = S/2 \qquad (5)$$

Equations (4) and (5) form a feasible solution to the PARTITION problem.

Theorem 2: The *GBCR* problem is *NP-complete*.
Proof: Being a generalization problem of finding feasible solution, which is an *NP-complete* one, the problem of *GBCR* itself is *NP-complete* with additional constraint of minimizing overall link cost.

3. Paths Coordination Heuristic

As we mentioned in section 1, from a viewpoint of an *ISP*, it is a valid assumption that all traffic demands between ingress-egress pairs can be obtained from traffic forecasting mechanism based on the *SLSs* between customers and the *ISP*. In this case, a set of *LSPs* can be set up simultaneously for delivering the corresponding traffic through the *MPLS* network. When multiple *LSPs* compete for some particular links that do not have sufficient bandwidth to support all of the traffic aggregates, our heurist will select part of the *LSPs* to traverse the link, while others are forced to explore alternative paths, with the objective that the overall path cost is maintained as low as possible. This type of path cost can directly reflect the overall bandwidth consumption and the overhead of setting up *LSPs*.
On the other hand, since our proposed heuristic for computing *LSPs* is an offline approach for network dimensioning, this implies that the forecasted traffic demands should be relatively static for a certain period of time. Whenever the initially computed *LSPs* cannot handle the network dynamics efficiently, online algorithms for local adjustment of *LSPs* will be triggered, and this will be our further research direction. Of course, accurate traffic forecasting as well as efficient network dimensioning algorithms from the *ISP* helps to reduce the frequency of performing online adjustments.

3.1 Terminologies Definition and Explanation

Before we present our proposed heuristic for coordinated path selection, we first make definitions and explanations of some terminologies to be used in the context.
(1) Cheapest Path (CP): The path with minimum overall link cost, and *CP* can also be computed by Dijkstra's shortest path algorithm regarding link cost. In this paper the term cheapest path is interchangeable with shortest path since the metric of delay is not within our concern.
(2) Saturated vs. Overloaded: As we mentioned in section 2, a *saturated* link refers to the one that has been nearly fully utilized and cannot afford bandwidth for any incoming *LSPs* under consideration. On the other hand, an *overloaded* link refers to the one whose actual allocated bandwidth has exceeded its bandwidth capacity, i.e., the corresponding utilization is greater than 100%. The objective of our algorithm is to minimize the overall network cost without incurring any overloaded link, but we do allow the existence of saturated links, though this is not desirable from the view point of load balancing.
(3) Traffic Trunk Profit (TTP): Suppose that the cheapest path *P* for the traffic trunk *T* from node *a* to *b* with total cost *C*(*P*) traverses a link *l*. If we prune *l* from the network and re-compute an alternative cheapest path *P'* for the same source-destination pair with total path cost *C*(*P'*), the Traffic Trunk Profit of *T* on link *l* is defined to be the difference between the total cost of *P* and *P'*, i.e.,

$$TTP_l^T = C(P') - C(P) \qquad (6)$$

Figure 3 presents the illustration on how *TTP* is defined for a traffic trunk *T* from ingress node *a* to egress node *b* on link *l*.

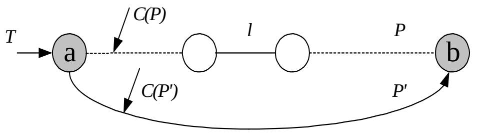


Figure 3 Definition of *Traffic Trunk Profit* (*TTP*)

(4) Profit-Expense Ratio (PER): Given a traffic trunk *T* from ingress node *a* to egress *b* and a particular link *l* on the corresponding cheapest path, The *Profit-Expense Ratio* (*PER*) of traffic trunk *T* on link *l* is defined to be:

$$PER_l^T = \frac{TTP_l^T}{B_T} = \frac{C(P') - C(P)}{B_T} \qquad (7)$$

where B_T is the bandwidth demand of traffic trunk *T*.

3.2 Coordinated Path Selection (CPS)

The main idea of our algorithm is to en-route all the traffic trunks through the network without incurring any overloaded links, and at the same time minimize the overall *LSP* cost. In the algorithms listed in section 1, individual *LSPs* are computed at the ingress router sequentially and independently with the consideration of local optimization. In contrast, in our heuristic when multiple *LSPs* need to traverse a cheap link that cannot satisfy all of the relevant demands, we will coordinate these *LSPs* in a global viewpoint, and allocate the link to those that can make most "profit" and least "expense" on that link. Specifically, according to the definition of Profit-Expense Ratio (*PER*), the limited bandwidth of the link will be allocated in non-ascent order to the *LSPs* with the largest value of *PER* as long as the link has not been overloaded. In effect, given a particular link, the *LSP* bandwidth allocation with cost optimization is modeled as the KNAPSACK problem [7] that is also *NP-complete*. The concept of *PER* is to guarantee that both profit (*LSP* cost) and expense (bandwidth constraint) are considered comprehensively. Figure 4 presents a scenario on the *PER* based bandwidth allocation mechanism in time of competitions for a particular link *l*. Suppose link *l* is on the cheapest paths for traffic trunks T_1 to T_i but the bandwidth capacity cannot support all of them. In this case, we prune *l* from the graph and reconstruct cheapest *LSPs* for all traffic trunks being involved. After computing the Profit-Expense Ratio for each traffic trunk on *l*, we sequentially allocate the bandwidth of link *l* to the *LSP* with the highest *PER* value. This procedure will terminate when the bandwidth of this link cannot support any traffic trunks that have not been considered. For all the traffic trunks that cannot traverse link *l*, they will follow the alternative cheapest paths that do not contain link *l*, namely P_i' in Figure 4. Specially, if there exist multiple paths with equal cost for a given traffic trunk (the corresponding value of *TTP* and *PER* is equal to 0), the algorithm is able to spread the traffic without even increasing the overall *LSP* cost. It should also be noted that, when computing alternative *LSPs*, no other overloaded links should be incurred. This path selection algorithm starts with the highest load link, and terminates when all the original overflowed links are eliminated from the network. The appendix presents the pseudo code of our coordinated path selection (*CPS*) algorithm.

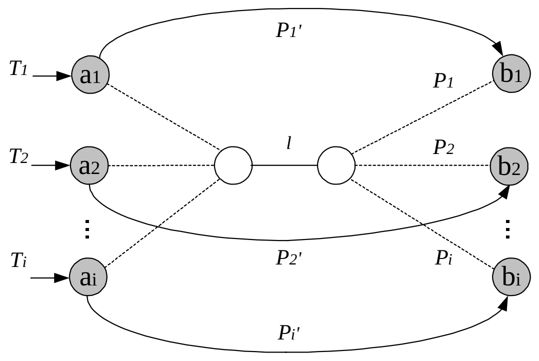


Figure 4 *PER* based bandwidth allocation

3.3. An Example

We use the network model in Figure 5 as an example for describing how the proposed heuristic works. Suppose three traffic trunks from S1 to S3 need to traverse the small network to reach the egress node R. The link metric in the figure has the same meaning with that in Figure 1.

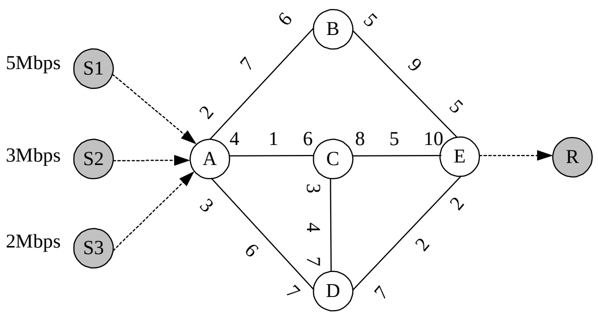


Figure 5 An example

Before using our proposed algorithm, we take the traditional approach [3] to compute *LSPs* without global coordination. First an *LSP* with cheapest cost is constructed for the traffic trunk from S1 with bandwidth demand of 5Mbps. The traffic trunk will take the path *Aà Cà E* to reach R, and after the allocation the residual bandwidth of link (*A*, *C*) and (*C, E*) is reduced 1Mbps and 5Mbps respectively, as shown in Figure 6(a). When we continue to compute *LSP*s for the rest of the two traffic trunks, we find link (*A*, *C*) has been saturated regarding their traffic demands. For the traffic trunk from S2, the cheapest path is *Aà Dà Cà E* with path cost 15, and for the traffic trunk from S3, the cheapest path is *Aà Dà E* with cost 8, as shown in Figure 6(b, c) respectively. The total path cost for the three traffic trunk is 6+15+8=29.

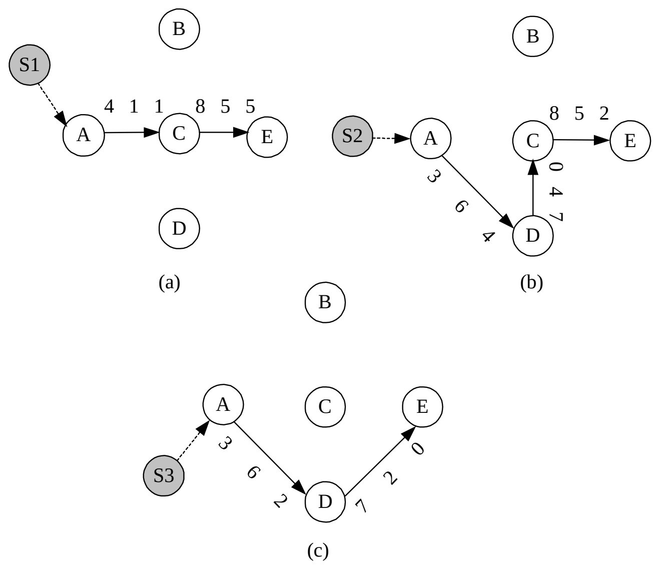


Figure 6 Traditional approach without coordination

Now we use our proposed heuristic for global path coordination to solve the problem again. From Figure 5 we find that the path *Aà Cà E* is the cheapest for all of the three traffic trunks, but unfortunately link (*A*, *C*) cannot afford its bandwidth capacity to satisfy all of them. According to our algorithm, Profit-Expense Ratio (*PER*) will be computed for the three traffic trunks regarding link (*A*, *C*), and then we can decide to which *LSP*(s) the bandwidth of the link should be allocated so that maximum profit will be obtained. If we prune link (*A*, *C*) from the graph and re-compute the three *LSPs*, the cheapest paths are: *Aà Bà E* for traffic trunk from S1 (path cost 16), *Aà Dà Cà E* for that from S2 (path cost 15), and *Aà Dà E* for that from S3 (path cost 8). In this case, the value of *PER* for the three traffic trunks on link (*A*, *C*) are 2.0, 3.0 and 1.0 respectively, according to formula (7) in section 3.1. Based on the obtained *PER* value, we decide that traffic from S2 has the highest priority to traverse link (*A*, *C*). After that we should consider to let the traffic trunk from S1 use link (*A*, *C*). However we find that the residual bandwidth is only 3Mbps and cannot afford the traffic demand of 5Mbps, and hence we will skip to the next traffic trunk, i.e., the one from S3. Since it has the bandwidth demand of 2Mbps, the traffic trunk can successfully traverse link (*A*, *C*). The resulted set of *LSPs* is given in Figure 7, with total path cost 28. It should be noted that if we sequentially compute *LSPs* for traffic trunks in an ascending order without global coordination in the graph in Figure 5, we are still able to obtain the same optimal result with that shown in Figure 7. However this is not always the case for other graph topologies, e.g., if we set the bandwidth capacity of link (*A*, *C*) to be 4Mbps instead of 6Mbps, the computation of *LSPs* in ascending order will not lead to the best solution.

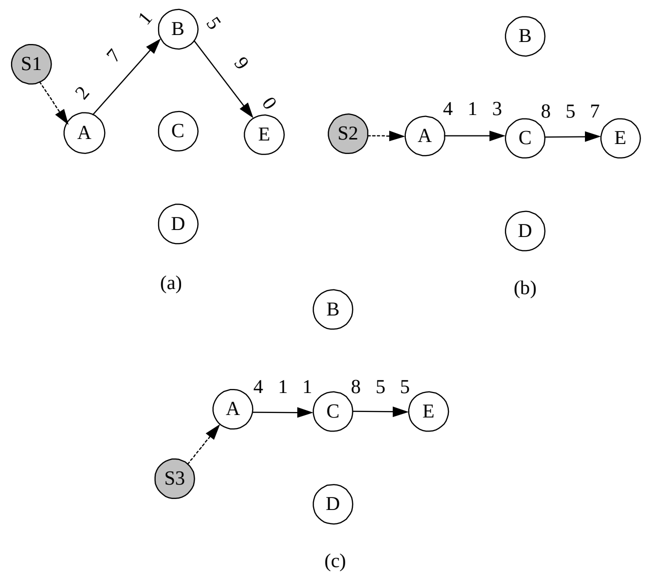


Figure 7 Result from coordinated path selection

4. Performance Study

In our simulation, we adopt the commonly used Waxman's random graph generator [8] in *GT-ITM* to create network topologies. Random network graphs with 100 nodes will be created, with the bandwidth capacity of each link (*u*, *v*) generated by the following function:

$$B(u,v) = B_m + r \times \% B_m \tag{8}$$

where B_m is the given minimum bandwidth capacity while *r* is a random number. Using this function no bandwidth with negative value will be generated and the bandwidth capacity of all links will range from B_m to $2\times B_m$ -1. The cost of each link is randomly selected between 1 and 100. On the other hand, a traffic matrix that specifies ingress-egress pair and the corresponding *LSP* bandwidth demand will also be created. In our simulation, we also use function (8) for deciding the average amount of bandwidth requirement from each traffic trunk. Altogether 30 traffic trunks are generated throughout the simulation, i.e., the traffic matrix contains 30 non-zero elements. The performance of the algorithm will be examined with the variance of minimum bandwidth capacity in the network. We will compare the performance of our proposed algorithm with *WSP* without coordination, cost-associate *BSP* (*CBSP*), i.e. we define the hybrid weight of each link (*u*, *v*) as:

$$w(u,v) = \frac{c_{uv}}{b_{uv}}$$

Figure 8 illustrates the performance of the overall path cost with the minimum bandwidth capacity B_m ranging from 30Mbps to 70Mbps in steps of 5Mbps. The average bandwidth requirement from each traffic trunk is fixed at 10Mbps. From the figure we notice that when the minimum bandwidth capacity increases the overall path cost achieved by all the three algorithms decreases. We can also find in Figure 8 that the proposed Coordinated Path Selection (*CPS*) algorithm achieves lower overall path cast compared with traditional approaches. Typically when the bandwidth capacity is relatively small, we notice that the corresponding gap is most significant. However, with the increase of the minimum bandwidth capacity, this gap becomes less obvious. This is because when there are less overflowed links in the network, both *CPS* and *WSP* will have a higher probability of selecting common paths for the given traffics aggregates.

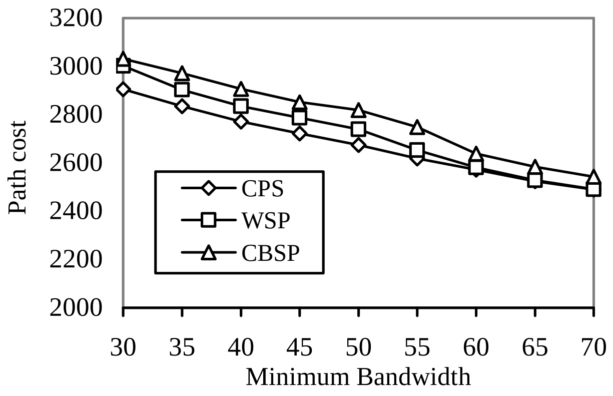


Figure 8 Path cost performance

Figure 9 presents the performance of the three algorithms in terms of link utilization. All the simulation conditions remain the same with those for Figure 8. We notice that the average link utilization decreases as the minimum bandwidth capacity is growing up in the network. Similar to the scenario in Figure 8, the performance of *CPS* is still the best amongst the three methods, i.e., it consumes least overall network bandwidth. While the bandwidth resource becomes ample, the advantage of *CPS* is less obvious compared with non-coordinated algorithms.

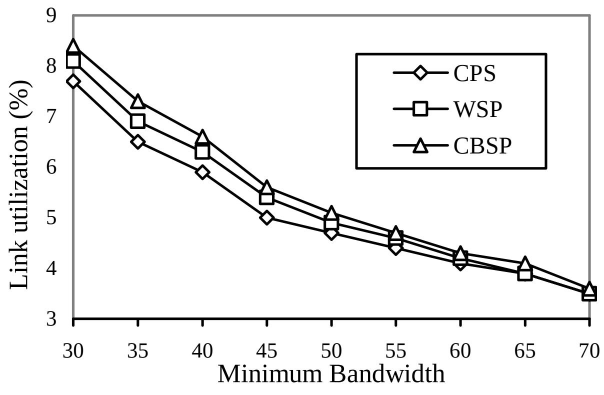


Figure 9 Link utilization performance

5. Summary

In this paper we proposed an efficient offline heuristic for coordinated path selection in *MPLS* bandwidth constrained routing. The motivation behind is to reduce the overall link cost and traffic loading by means of global resource coordination. Our simulation results show that the proposed algorithm provides better performance compared with traditional bandwidth associated *QoS* routing approaches. Our future research work will focus on efficient online adjustment of *LSPs* and extended algorithms on selecting paths for bandwidth guaranteed multicast traffic in DiffServ networks.

References

[1] Y. Lin *et al*, "*QoS Routing Granularity in MPLS Networks*", *IEEE* Communications, 40(6), 2002, pp 58-65
[2] M. Kodialam, T. V. Lakshman, "*Minimum Interference Routing with Applications to MPLS Traffic Engineering*", *IEEE INFOCOM* 2000, pp 884-893
[3] Z. Wang, J. Crowcroft, "*Quality-of-Service Routing for Supporting Multimedia Applications*", *IEEE JSAC*, Vol. 14, No. 7, 1996, pp 1228-1234
[4] G. Apostolopoulos *et al*, "*QoS Routing Mechanisms and OSPF Extensions*", *RFC* 2676, Aug. 1999
[5] Q. Ma, P. Steenkiste, "*On Path Selection for Traffic with Bandwidth Guarantees*", *IEEE ICNP'97*, pp 191-202
[6] J. Wang, K. Nahrstedt, "*Hop-by-Hop Routing Algorithms for Premium-class Traffic In DiffServ Networks*", *IEEE INFOCOM* 2002
[7] M. R. Garey *et al*, "*Computer And Intractability – A Guide to the Theory of NP-Completeness*", Freeman, 1979
[8] B. M. Waxman, "*Routing of multipoint connections*", *IEEE JSAC* 6(9) 1988, pp 1617-1622

Appendix: Pseudo code for *CPS*

```
Procedure CPS
  For each traffic trunk TT
    Compute cheapest path p using Dijkstra's algorithm
      without considering bandwidth constraints;
    Update traffic loading for each link;
  While there exist overflowed links do
      Select highest load link $\hat{l}$ ;
      Let P be the set of paths passing through $\hat{l}$ ;
      For each $p \in P$
        Let the original path cost be C(p);
        Tear down the original path p containing $\hat{l}$ ;
        Set the link cost of $\hat{l}$ to INFINITY;
        Compute alternative cheapest path p' without
          incurring new overflowed links;
        Compute PER = (C(p')-C(p))/B(TT(p));
      Sort P according to PER in non-ascend order;
     While $\hat{l}$ is not overflowed do
         If the residual bandwidth of $\hat{l}$ > B(TT(p))
           Allocate $\hat{l}$ to TT(p) and compute cheapest
             path p for TT(p);
         If there are no new overflowed links
           TT( p) takes link $\hat{l}$
         Else TT(p) follows alternative path p';
        Update traffic loading for each link;
        If overflowed links remain
          Error: Can't find feasible LSPs;
        Find next p in P;
     Select next highest load link $\hat{l}$ ;
End Procedure
```