실시간 네트워크에서 개선된 분산 QoS 알고리듬

서희종

An Improved Algorithm of Distributed QoS in Real-time Networks

Hee-Jong Suh

요 약

본 논문에서는 실시간 네트워크에서 사용될 개선된 분산 QoS 알고리듬을 제안한다. 이 알고리듬은 DCUR(delay-constrained unicast routing) 알고리듬과 같이 LC(least-cost) 패스나 LD(least-delay) 패스를 사용하고 있지만, 루프를 처리할 때에 DCUR 알고리듬보다도 더 효율적으로 처리한다. 본 알고리듬은 모든 패스를 더욱 더 효율적으로 찾을 수 있다.

ABSTRACT

In this paper, an improved algorithm of distributed QoS is proposed for real-time networks. This algorithm like a delay-constrained unicast routing(DCUR) algorithm uses either least-cost(LC) path or least-delay(LD) path of an active node, but when there is a loop, this algorithm is quite different from DCUR in choosing the link between the active node and the previous node to replace the original loop path. And this algorithm makes the construction of the paths more efficiently.

키워드

delay-constrained, Distributed QoS, Unicast Routing
딜레이-컨스트렌인드, 분산 QoS, 단방향 라우팅

Ⅰ. Introduction

The growing popularity of real-time and multimedia applications over the Internet has stimulated strong interest in extending QoS(quality of service) support to existing routing protocols(e.g. OSPF, BGP)[1,2]. The QoS requirements of these applications raise new challenges for the development of integrated-service network systems. One of the key issues is QoS routing. The goal of routing solutions is twofold: (a) selecting network routes that have sufficient resources to meet the QoS requirements of every admitted connection and (b)achieving global efficiency in resource utilization.

Routing consists of two basic tasks. The first task is to collect the state information and keep it up-to-date. the second task is to find a satisfactory path for a new connection based on the collected information. Most published routing algorithms [3,4,5,6] require every node to maintain a global network state either by a distance-vector protocol or by a link-state protocol[7]. However, such a

* 전남대학교 전자통신공학과(hjsuh@chonnam.ac.kr)
접수일자 : 2011. 11. 11 심사(수정)일자 : 2011. 12. 05 게재확정일자 : 2011. 12. 22

global state is consequentially imprecise in a dynamic network where the traffic load changes constantly. The imprecision is especially noticeable in large wide area networks due to the following reasons. First, it takes some propagation delay for a local state change to be broadcasted to other nodes. Second, a distance-vector(or link-state) protocol updates the state information periodically or upon triggering when significant state change is detected. There exists a tradeoff between the update frequency and the involved overhead. For large scale networks, the excessive communication overhead often makes it impractical for the update frequency to be high enough to cope with the dynamics of network parameters such as bandwidth and delay. Third, the hierarchical approach is likely to be used to solve the scalability problem of routing in large network[8]. And the state aggregation in hierarchical routing increases the level of imprecision[9].

There are two phases involved in handling a real-time channel: channel establishment[10] and run-time scheduling of packet[10,11]. The channel establishment phase involves the selection of a qualified route for the channel satisfying traffic characteristic and QoS requirements of the call request, without compromising the guarantees of the existing channels. Although several channel establishment schemes have been proposed, very few of them have explicitly addressed the issue of route selection, despite its importance in the channel establishment phase.

Unicast routing protocols can be classified into two categories: distance-vector protocols, e.g. the routing information protocol(RIP)[12], and link-state protocols, e.g. the open shortest path first protocol(OSPF)[13]. Distance-vector protocols are based on a distributed version of Bellman- Ford's shortest path(SP) algorithm[14]. Considering the message complexity, distance-vector routing protocols scale well to large networks, because each node send periodically topology update messages only to its direct neighbors. Each node maintains only limited information about the shortest paths to all other nodes in the network. Due to their distributed nature, distance-vector protocols may suffer from looping problems when the network is not in steady state. In link-state protocols, on the other hand, each node maintains complete information about the network topology, and uses this information to compute the shortest path to a given destination centrally using Dijkstra's algorithm[15]. Link-state protocols have limited scalability, because flooding is used to update the nodes' topology information. They do not suffer from looping problems, however, because of their centralized nature. In 1995, Garcia Luna Aceves and Behrens proposed a distributed protocol, based on link vectors, that avoids looping problems and scales well to large networks[16,17,18,19].

Both Bellman-Ford's and Dijkstra's SP algorithms are exact and run in polynomial time. As the name indicates, an SP algorithm minimizes the sum of the lengths of the individual links on the path from source to destination. If the length of a link is the measure of the delay on that link, then an SP algorithm computes the least-delay(LD) path, and if the link length is set equal to the link cost, then an SP algorithm computes the least-cost(LC) path.

Reeves and Salama study the problem of unicast routing of real-time traffic subject to an end-to-end delay constraint in connection-oriented networks[5]. They formulate the problem as a delay-constrained LC(DCLC) path problem. This problem is NP-hard. Therefore, they propose a distributed heuristic solution, which is the delay-constrained unicast routing(DCUR) algorithm.

In this paper, we first discuss DCUR algorithm, and then we propose a improved algorithm about DCUR algorithm. In DCUR algorithm, the active node chooses either LC path or LD path, when a

loop is detected, the active node sends a message(*remove_loop*) traversing the loop backwards to delete all links between the active node and the node whose routing table entry's flag is set to LC path(LCPATH), and find another path at this node towards the destination. But the proposed algorithm is quite different from DCUR, because the node will firstly consider to choose the link between the active node and the previous node to replace the original loop path when a node detect a loop.

This paper is organised as follows. In Section II, the DCUR algorithm is discussed. In Section III, an improved algorithm is proposed based on DCUR algorithm. Section IV concludes the paper.

II. Description of DCUR algorithm

1. Problem formulation

A point-to-point communication network is represented as a directed, simple, and connected network $N=(V,E)$, where V is a set of nodes and E is a set of directed links. A directed link $e=(u,v)\in E$ has a cost $C(e)$ and a delay $D(e)$ associated with it. $C(e)$ and $D(e)$ may take any non-negative real values. The link delay $D(e)$ is a measure of the link's utilization. For a given source node $s\in V$ and destination node $d\in V$, there is a set of possible paths p_1, p_2, ..., p_m from s to d, e.g. $P=\{p_1, p_2, \ldots p_m\}$, the cost of a path p_i is defined as:

$$cost(p_i)=\sum_{e\in p_i} C(e) \tag{1}$$

similarly, the end-to-end delay along the path p_i is defined as:

$$Delay(p_i)=\sum_{e\in p_i} D(e) \tag{2}$$

The DCLC problem finds the LC path from a source node s to a destination node d such that the delay along that path does not exceed a delay constraint σ. It is a constrained minimization problem that can be formulated as follows.

Delay-constrained least-cost(DCLC) path problem: Given a directed network $N=(V,E)$, a non-negative cost $C(e)$ for each $e\in E$, a source node $s\in V$, a destination node $s\in V$, and a positive delay constraint σ, the constrained minimization problem is:

$$\min_{p_i\in p'(s,d)} Cost(p_i) \tag{3}$$

where $p'(s,d)$ is the set of paths from s to d for which the end-to-end delay is bounded by σ. Therefore $p'(s,d)\subseteq p(s,d)$. If $p_i\subseteq p(s,d)$ then $p_i\subseteq p'(s,d)$ iff

$$delay(p_i)\le \sigma \tag{4}$$

2. Routing Information

When executing DCUR, we need some routing information which is kept at any node. Every node $v\in V$ must have the following information available during the computation of the delay-constrained path: the costs of all outgoing links, the delays of all outgoing links, a cost vector, a delay vector, and a routing table. The cost vector and delay vector structure are presented below, and the routing table structure will be described in the next section.

The cost vector at node x consists of $|V|$ entries, one entry for each node y in the network. Each entry in the cost vector holds following information:

1) the destination node ID, y,
2) the cost of the LC path from x to y, denoted $lc\_value(x,y)$
3) the ID of the next hop node on the LC path from x to y, $lc\_nhop(x,y)$

Similarly, the delay vector at node x has one entry for each node y in the network. Each entry in the delay vector holds:

1) the destination node ID, y,
2) the total end-to-end delay of the LD path from x to y, $ld\_value(x,y)$, and
3) the ID of the next hop node on the LD path from x to y, $ld\_nhop(x,y)$.

3. The Delay-constrained Unicast Routing (DCUR) Algorithm

DCUR is a source-initiated algorithm that constructs a delay-constrained path connecting source node s to destination node d. The path is constructed from the source to the destination. The procedure constructing path is as follows:

1) the source node s initiates path construction by looking up the $ld\_value(x,y)$ from its delay vector, and check whether

$$ld\_value(x,y) > \sigma \text{ or } ld\_value(x,y) \leq \sigma \quad (5)$$

if the former is satisfied, then delay-constrained path do not exist between s and d, and DCUR fails and stops. if, however, delay-constrained path do exist, i.e.,

$$ld\_value(x,y) \leq \sigma \quad (6)$$

the algorithm proceeds. Then source s becomes the current active node, denoted $active\_node$. At all times there exist only one active node at the end of partially constructed path. At source s, the variable $delay\_so\_far$ is set to 0, and the variable $previous\_active\_node$ is set to null.

2) The $active\_node$ reads the ID of the next hop node on the LC path towards d, lc_nop, from its cost vector. Then $active\_node$ sends a query message to lc_nop, requesting the LD value from $active\_node$ to d. $active\_node$ looks up the requested value ld_value (lc_nhop, d), from its delay vector, and sends a response message back to $active\_node$ with this information. After $active\_node$ receives the response message, it checks if

$$delay\_so\_far + delay\_value(active\_node, lc\_nhop) + ld\_value(lc\_nhop,d) \leq \sigma \quad (7)$$

if the inequality is satisfied, then there exist delay-constrained paths from $active\_node$ to d along the link($active\_node$,lc_nop),and $active\_node$ selects the direction of the LC path towards d. If the inequality is not satisfied, then $active\_node$ selects the direction of the LD path towards d.

3) after deciding which direction to follow, $active\_node$ creates a routing table entry with the following information;
   - the ID of s,
   - the ID of d,
   - $previous\_node$=ID of the $previous\_active\_node$,
   - $next\_node = \begin{cases} lc\_nop & \text{if } LC\ path\ direction\ is\ chosen, \\ ld\_nop & \text{if } LD\ path\ direction\ is\ chosen \end{cases}$

4) $active\_node$ adds delay($active\_node$, $next\_node$) to the variable $delay\_so\_far$. Then the $active\_node$ sends a $construct\_path$ message to $next\_node$ that contains: the ID of the source s, the ID of the destination d, the value of the delay constraint σ, and the updated value of $delay\_so\_far$ which represents the delay along the already constructed path from s to $next\_node$. After sending out $construct\_path$ message, $active\_node$ becomes inactive.
5) when a node $x \neq d$ receives a $construct\_path$ message, it becomes the new $active\_node$. The

new *active_node* sets *previous_active_node* to be the ID of the node which sent it a *construct_path* message. Then the new *active_node* executes the same procedure just described. When the destination node d receives a *construct_path* message, it records the ID of the node which sent the message. d creates a routing table entry, with the following values: ID of the source s, ID of the destination d, *previous_node* =*previous_active_node*, *next_node*=null, and *previous_delay*=*delay_so_far*. Then the destination sends an acknowledgement back to the source. When the source receives the acknowledgement message, it signals to the application that the path construction has been successfully completed, and traffic can be transmitted along the path.

III. Improved DCUR Algorithm(I-DCUR)

From above description of DCUR algorithm, we can see that, in DCUR, each node involved in the path construction operation selects either the LC path direction or the LD path direction. This means that some nodes choose the LC path direction while others choose the LD path direction. In this condition, loops occur. We will give an example to show how a loop occurs, and how DCUR detects the loop, and then we propose an improvement algorithm. Figure 1 shows a scenario that results in a loop. The source node s initiates the construction of a path towards the destination node d with an imposed delay constraint value of 8. Figure 1(a), 1(b),and 1(c) show successive stages of path construction until a loop is created. The source s looks up its LD vector and LC vector, apparently the LC path is $s \rightarrow u \rightarrow d$, but the total delay on the path is equal to 11, which is greater than the constraint value 8. Then the source s follows the LD path direction towards the destination d($s \rightarrow v \rightarrow d$), and link($s$, v) becomes the first link in the path.

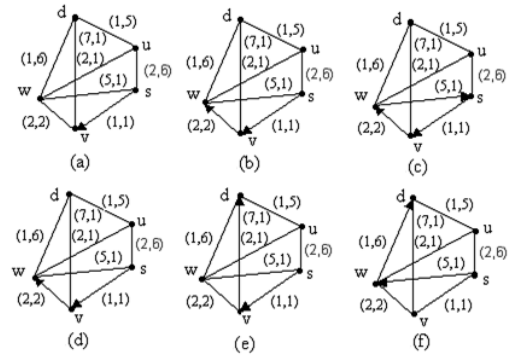

Fig. 1 Example of a loop scenario. s is the source and d the destination. link costs and link delay are shown next to each link as (cost, delay), $\sigma = 8$.

Node v follows the LC path direction towards d ($v \rightarrow w \rightarrow d$) and adds link($v$, w) to the path. Node w follows the LD path direction and adds link(w, s) to the path. This creates a the loop $\{s \rightarrow v \rightarrow w \rightarrow s\}$, as shown in Figure 1(c). When node s receives a *construct_path* message, it searches its routing table. Because a *construct_path* message includes the ID of the source, and the ID of the destination, node s can detect the loop if a routing table entry already exists for the source-destination pair specified in the *construct_path* message. It reacts by sending a *remove_loop* message that traverses the loop backwards. Node w receives the *remove_loop* message from node s, but node w is already following the LD path direction towards the destination, so all it does is to send the *remove_loop* message further backwards to v, and to delete its routing table entry, thereby removing link(s, w) from the path(Figure 1(d)). Node v receives the *remove_loop* message. It is following the LC path direction towards the destination, so it decides to follow the LD path direction instead, and

modifies its routing table entry accordingly. Thus removing link(v, w) from the path and adding link (v, d) instead. The final delay-constrained path from s to d is the one shown in Figure 1(e).

This improved algorithm is different from DCUR mainly in the case when a loop occurs, it is as follows: when a node y receives a $construct\_path$ message from node x, and becomes current $active\_node$,and node x becomes $previous\_active\_node$. If no loop is detected at node y, it executes the same procedure described above. If, however, a loop is detected, node y firstly looks for its $least\_delay$ vector to read the delay value between node y and x which sent the $construct\_path$ message. In DCUR procedure described above, the $previous\_active\_node$ follows the LD path direction to add link(x,y) to the path. It is easy to certify that link(x,y) must be the LD path between node x and node y, too. So node y must have kept the delay value between x and node y, D(x, y), in its LD path vector. Then node y sends back a $find\_loop$ message that contains node y's $delay\_so\_far(y)$ and D(x,y). After receiving the $find\_loop$ message, node x reads the ID of the next hop node on the LC path towards d, lc_nop, from its cost vector. Then x sends a query message to lc_nhop, requesting the LD value from lc_nhop to d. lc_nhop looks up the requested value, $ld\_value(lc\_nhop, d)$, from its delay vector, and sends a response message back to node x with this information. After node x receives the Response message, it checks if

$$delay\_so\_far(y) + D(x,y) + D(x,lc\_nhop) + ld\_value(lc\_nhop,d) \leq \sigma \quad (8)$$

If the inequality is satisfied, then there exist delay-constrained path from source s to destination d which use the path(s, y),the link(y, x), and the link(x, lc_nhop). Then x sends a $remove\_path$ message to $previous\_node$ to remove the path from node y to x originally constructed by DCUR before a loop was detected, and lets

$$delay\_so\_far = delay\_so\_far(y) + D(x,y) \quad (9)$$

and modifies the routing entry made while it was the active node with the following information:

- the ID of s,
- the ID of d,
- $previous\_node$=the ID of the node which sent the $find\_loop$ message
- $next\_node$=lc_nhop
- $previous\_delay$=$delay\_so\_far$
- flag=LCPATH

Then x adds D(x,$next\_node$)to the variable $delay\_so\_far$, and sends a $construct\_path$ message to $next\_node$, the procedure is the same as DCUR.

If inequality (8) is not satisfied, node x executes the same procedure when a loop is found in DCUR algorithm.

Ⅳ. Conclusion

An improved algorithm I-DCUR is proposed, which is based on DCUR algorithm. In DCUR algorithm, when a loop is detected, the $active\_node$ directly sends a $remove\_loop$ message to the previous node on the loop, and the previous node will proceed sending the $remove\_loop$ message backwards, removing routing table entries until it finds a node whose routing table entry's flag is set to LCPATH, indicating that this node is following the LC path direction towards the destination. In this algorithm, when a loop is found, maybe there have been lots of links involved between the active node and the node whose routing table entry's flag

is set to LCPATH, and this will prolong the average path constructing time. But in the I-DCUR algorithm, when a loop is detected, the active node sends a $find\_loop$ message to tell the previous node that a loop has been found, then the previous node will first consider whether it can choose the link($active\_node$, $previous\_node$) to replace the another path between the same two nodes or not. So, the link($active\_node$, $previous\_node$) will be added to the path, and the another path is deleted, then the procedure continues. If it cannot do that, it will do just like DCUR. Therefore I-DCUR can provide more chances in choosing a path, and make path construction more efficiently. From Figure 1(c), 1(d) and 1(e), according to DCUR, when node s finds a loop, the $remove\_loop$ message will traverse the loop backwards to node v to remove link(s, w), link(w, v) from the path, and adds link (v, d) to the path. The final path constructed is $s \rightarrow v \rightarrow d$, and the total cost is 8. With the I-DCUR, the previous node w will first check whether link (s, w) can replace the another path($s \rightarrow v \rightarrow w$). In the example, the link(s, w) is added to the path, and the path ($s \rightarrow v \rightarrow w$) is deleted, and the final path from source node s to destination node d is ($s \rightarrow w \rightarrow d$) whose total cost is 6.

References

[1] G. Apostolopoulos et al., "QoS routing mechanism and OSPF extensions," IETF RFC2676, Aug. 1999.

[2] E.Crawly,R.Nair,B.Rajagopalan,and H.Sandick, "A framework for QoS-based routing in the Internet," Internet Engineering Task Force RFC2386(Information), Aug. 1998.

[3] Shigang Chen; Nahrstedt, K. ''On finding multi-constrained paths,'' IEEE International Conference on Communications, pp. 874-879, Jun. 1998.

[4] Q. Ma and P. Steenkiste, ''Quality-of-service routing with performance guarantees,'' Proceeding of the 4th international IFIP Workshop on quality of service, May 1997.

[5] Reeves, D. S., Salama, H. F. "A distributed algorithm for Delay-Constrained unicast routing," IEEE Trans. on networking. Vol. 8, No. 2, pp. 239 - 250, Apr. 2000.

[6] Z. Wang and J. Crowcroft, ''Qos routing for supporting resource reservation,'' JSAC, Sep. 1996.

[7] James F. kurose and Keith W.ross, ''computer networking,'' second edition, pp.301-317, 2003.

[8] A. Forum, ''Private network interface(pnni),'' v1.0 specifications, May 1996.

[9] Guerin, R., Orda, A ''Qos-based routing in networks with inaccurate information: Theory and algorithms,'' Infocom'97, Japan, pp. 75-83, Apr. 1997.

[10] D. D. Kanklur, K, G. Shain, and D. Frerrari, "Real-time communication in multihop networks," IEEE Trans. Parallel Distrib. Syst, Vol. 5, pp. 1044-1056, Oct. 1994.

[11] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," Proc. IEEE, Vol. 83, pp.1374-1396, Oct. 1995.

[12] C. Hedrick, "Routing information Protocol," Internet RFC 1058, http://ds.internic.net /rfc/rfc1058.txt, Jun. 1988.

[13] J. Moy, "OSPF Version 2,'' Internet RFC 1583, http://ds.internic.net/rfc/rfc1583.txt, March 1988.

[14] D. Bertsekas and R. Gallager, ''Data Networks,'' Prentice-Hall, 2nd ed., 1992

[15] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, "Data Structures and Algorithm,"

[16] Garcia-Luna-Aceves, J. J., Behrens, J., "Distributed, Scalable Routing Based on Vectors of Link States," IEEE Journal on Selected Areas in Communications, Vol. 13, No. 8, pp. 1383-1395, Oct. 1995.

[17] 윤성하, 손희배, 이영철, "차세대 자동차 통합스마트 모니터 시스템에 관한 연구", 한국전자통신학회논문지, 6권, 3호, pp. 439-444, 2011.

[18] 김분희, "메뉴와 소셜 네트워크 공동구매 정보 동시제공 P2P 시스템", 한국전자통신학회논문지, 6권, 3호, pp. 445-450, 2011.

[19] 홍완표, "데이터통신 전송효율과 ASCII 부호체게 고찰", 한국전자통신학회논문지, 6권, 5호, pp. 657-664, 2011.

저자 소개

서희종(Hee-Jong Suh)

1975년 한국항공대학교 졸업(공학사)
1996년 중앙대학교 대학원 졸업(공학박사)
현재 전남대학교 교수
※ 관심분야 : 네트워크, 그래프이론