A Scalable Distributed QoS Multicast Routing Protocol

Shigang Chen
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida 32611, USA
Email: {sgchen}@cise.ufl.edu

Yuval Shavitt
Department of Electrical Engineering - Systems
Tel-Aviv University, Ramat Aviv 69978, ISRAEL
Email: shavitt@eng.tau.ac.il

***Abstract*— Many Internet multicast applications such as teleconferencing and remote diagnosis have Quality-of-Service (QoS) requirements. It is a challenging task to build QoS constrained multicast trees with high performance, high success ratio, low overhead, and low system requirements. This paper presents a new scalable QoS multicast routing protocol (SoMR) that has very small communication overhead and requires no state outside the multicast tree. SoMR achieves the favorable tradeoff between routing performance and overhead by carefully selecting the network sub-graph in which it conducts the search for a path that can support the QoS requirement, and by auto-tuning the selection according to the current network conditions. Its early-warning mechanism helps to detect and route around the real bottlenecks in the network, which increases the chance of finding feasible paths for additive QoS requirements. SoMR minimizes the system requirements; it relies only on the local state stored at each router. The routing operations are completely decentralized.**

I. Introduction

Multicast is an efficient way to deliver content to a large group of receivers by using a tree structure embedded in the network. Given a QoS requirement such as bounded end-to-end delay, a *feasible multicast tree* is one that satisfies the requirement. A *feasible tree branch* (*path*) is a path that connects a new group member to a multicast tree and has the resources to support the required QoS. The task of QoS multicast routing is to find feasible tree branches for new group members. A survey in this research area can be found in [1]. Finding feasible tree branches is difficult in very large networks such as the Internet because it is impractical to maintain the global QoS state at any single node. A brute-force flooding algorithm that searches all possible paths in the network guarantees to find a feasible branch if one exists. However, the excessive overhead of full-scale flooding deems to be impractical for all but small networks. Thus, for applications that require QoS guarantees, recent research focuses on distributed multicast routing algorithms that search a selected subset of the network to find feasible tree branches for new group members [2], [3], [4], [5].

A good QoS routing protocol achieves a favorable tradeoff between the routing overhead and the ability of finding a feasible path (often quantified as *success ratio* or *success probability*). In addition, a good protocol exhibits or optimizes other characteristics, such as minimizing the extra state information the protocol maintains in the network; decentralizing the routing operations to spread the workload; and adapting the routing activity according to the current network condition and avoiding the area where traffic congestion occurs. QMRP [2] is a protocol that exhibits a very good tradeoff between the routing overhead and the success probability. In addition, QMRP has many of the good merits mentioned above. However, QMRP suffers from two probloms. Firstly, QMRP deposits temporary state in the routers for each join request. It is highly desired that the routers only maintain per group information but not per-group-per-join information. Secondly, QMRP was designed for applications with non-additive QoS requirements such as bandwidth and buffer space. It lacks the mechanism to handle additive QoS requirements such as delay. While spanning join [3] and QoSMIC [5] do not have the above problems, they have higher overhead and lower success probability [2]. Hence, a further study for a scalable, efficient QoS multicast routing protocol is under call.

This paper suggests a scalable QoS multicast routing protocol, SoMR, that shares the adaptive path-branching idea of QMRP, but eliminates the temporary use of per-group-per-join routing state. In QMRP, each new member initiates a search tree, which grows towards the multicast tree. The search tree is per join state information. In SoMR, the multicast tree grows towards the new members. The protocol does not require to store any extra routing state other than the multicast tree itself. It not only gets rid of per join routing state but also allows the dynamic aggregation of multiple join requests, where a single tree branch may grow toward multiple new members. By eliminating the search tree, SoMR removes the state machine in QMRP that governs the construction of the search tree, and therefore simplifies the implementation. SoMR uses a novel early-warning (EW) mechanism that takes the additive nature of the delay requirement into account and attempts to identify the most appropriate point to explore alternative paths in order to maximize the chance of success.

The rest of the paper is organized as follows. Section II presents our network model. Section III describes the routing protocol. Analysis and simulation results are provided in Section IV and Section V, respectively. Section VI draws the conclusion.

II. Network model

We make the following assumptions about the network.

1) There exists an underlying unicast routing protocol which can deliver a message between any two connected nodes in the network. A node knows the length (number of hops) of the unicast routing path to any destination. Many widely used unicast routing protocols such as RIP and OSPF provide this information.
2) Every node maintains its up-to-date local state, such as the delay of each outgoing link, which includes the processing time, buffering delay, and link propagation delay. Assume that once resources are committed, such delay can be assured during the lifetime of data communication. How to make resource reservation [6] and what packet scheduling algorithms are used [7] are out of the scope of this paper.

We assume that any new member is able to map a multicast group address to the root node of the tree on demand possibly by a query/response Session Directory [8].

We define the notations in the following. Let k and i be two on-tree nodes. The path in the multicast tree connecting them is called the *in-tree path*, denoted by $P_{k,i}$. The guaranteed delay bound of this path is called the *in-tree delay*, denoted by $delay(P_{k,i})$. Let T be the set of on-tree nodes and r be the root of the tree. A delay-constrained multicast tree satisfies that, $\forall i \in T,\ delay(P_{r,i}) \leq D$. We require each on-tree node i to know $delay(P_{r,i})$. In fact, our protocol makes sure that any node joining the tree will have this value.

We assume that each link (i, j) can ensure certain delay bound (which might be infinity) for the Class of Service (CoS) with which the multicast group is associated. This bound of the link delay is denoted as $delay(i, j)$.

III. A New QoS Multicast Routing Protocol

A. Protocol Description

SoMR consists of two phases. The first phase is similar to shortest path routing (SPR), in which a JOIN message is sent from the new member t to the root r along the unicast routing path. The JOIN message accumulates the path it traverses. It also accumulates the delay of the path in the reverse direction. When the JOIN message reaches an on-tree node k, if the accumulated delay plus the in-tree delay from r to k does not violate the delay requirement, a feasible tree branch is detected, which is the traversed unicast path. A CONSTRUCTION message is then sent back along the path (source routing) to construct a tree branch connecting the new member. Since the new member joins the tree successfully, the second phase will not be activated.

On the other hand, if the delay requirement is violated at k, the JOIN message continues travelling to the root r. When the root receives the message, it starts the second phase, which employs multi-path routing. The root sends GROW messages to its neighbors. These GROW messages will then travel along the unicast routing paths towards the new member. As they travel, they try to construct new tree branches hop by hop along the way. Each GROW message carries the delay requirement D. It also accumulates the delay of the constructed tree branch. Hence, when an intermediate node i receives GROW, it knows the in-tree delay from r to i, $delay(P_{r,i})$.

Below we describe the actions that i will take after receiving GROW. First, i does an EW (Early Warning) test to see how likely the proceeding unicast path will satisfy the delay requirement D. If the EW test passes, routing continues along the unicast path towards t; Otherwise, i attempts multi-path routing which may result in multiple downstream paths to be constructed. Let j be the next hop on the unicast path. The EW test accepts four parameters, D, $delay(P_{r,i})$, $delay(i, j)$, and l, which is the length of the unicast path from i to the new member t. The EW test is defined as follows.

```
**if** $delay(i, j) > (D - delay(P_{r,i}))/l$
**then** warning
**else** pass
```

$D - delay(P_{r,i})$ is the remaining slack of the delay requirement that further tree construction is allowed to have. $(D - delay(P_{r,i}))/l$ is the ”fair share” of this slack for each link on the path from i to t. The above EW test states that if the delay of the link is larger than the fair share, a warning should be triggered; otherwise, the test is passed. More sophisticated EW test can be used, but are not considered here. In our simulation, the above EW test worked well.

If the EW test is passed, which means that the proceeding path is likely to satisfy the QoS requirement, i adds link (i, j) into the multicast tree and forwards the GROW message to the next hop j. If every intermediate node passes the EW test, a feasible branch is established for the new member.

However, if the EW test warns that the proceeding path may violate the QoS requirement, extra effort needs to be taken. Searching multiple downstream paths will increase the chance of success. Namely, the tree construction needs to *branch out*. We call i a *branching point*. GROW messages are sent to a subset of adjacent nodes x that satisfy the following *QoS test*:

```
**if** $delay(i, x) > D - delay(P_{r,i})$
**then** fail
**else** pass
```

Apparently, x can be the node j that just failed the EW test, but x should not be the adjacent node from which the GROW was previously sent to i. For the purpose of overhead reduction, we also might select only some of the nodes that pass the QoS test (see section III-C).

If the QoS test is failed for every adjacent node, a BREAK message is sent back to trim the partially constructed tree branch. When a node k receives a BREAK message from a node i, it first deletes link (k, i) from the multicast tree, and then if k becomes a leaf node and is not a member of the multicast group, it will delete itself from the multicast tree and propagate the BREAK message to its parent node. As BREAK travels back to r, the new tree branch is deleted.

There is a difference between the EW test and the QoS test. The EW test tries to make early guess on whether the proceeding path is likely to satisfy the delay constraint. If it sees signs of trouble, it triggers branching to improve the chance of success. The QoS test is to check if the delay constraint has already been violated. If it is, no further construction will be done towards this direction.

At the beginning of the second routing phase, our protocol requires the root to be a mandatory branching point (an EW test is not necessary). Our simulations consistently show that SoMR performs better this way. The reason is that an early branching widens the search range and gives the subsequent tree construction more flexibility.

Whenever a GROW message reaches t, a feasible tree branch is successfully established. t may receive multiple GROW messages from different branches. In this case, it sends back BREAK messages to tear down all but the best branch. t can use the information (delay, bottleneck bandwidth, etc.) collected by the received GROW messages to determine which branch should be kept. Therefore, although multiple GROW messages can grow multiple tree branches temporarily for the same new member, only one branch will remain and the other extra ones will be detected and pruned automatically.

Although a single join can result in multiple temporary tree branches, one important point is that, *no matter how many simultaneous joins there are*, each node keeps only a multicast entry (one per group), and it does not keep any information about a particular join like the state machine in QMRP. So the maximum memory consumed by a multicast group on a router is constant (an entry in the multicast routing table), independent of the number of simultaneous joins.

B. Breaking loops

As tree branches are constructed towards the new member, loops may form in the multicast tree. Fig. 1 gives one example. Before we provide a general solution to the looping problem, we need to study GROW messages more closely. Consider a GROW message that constructs a tree branch along a unicast path P to the new member. Some of the links on P may be already in the multicast tree while the others are not. When a GROW message travels along a link that is already in the multicast tree, we assign a color of *blue* to the GROW message. When a GROW message travels along a link that is not in the multicast tree, we assign *green* to the message.[1] Only green-GROW messages may form loops, because green-GROW messages join new links to the tree while blue-GROW messages follow existing tree links.

The sender of a GROW message can determined the color of the message as follows. When a node i sends a GROW to an adjacent node j, if j is the parent or a child of i in the multicast tree, i marks the GROW to be blue; otherwise, it marks the GROW to be green.

Using the coloring scheme, loop detection is easy. When an on-tree node receives a green-GROW message, a loop is formed. In Fig. 1, the on-tree node i detects a loop when it receives a green-GROW message from b. A BREAK message is sent back to break the loop, while the GROW message continue constructing a tree branch towards the new member.

Arriving at i, the GROW message has the in-tree delay of the new tree branch ($r \rightarrow a \rightarrow b \rightarrow i$). i knows the in-tree delay of the old tree branch ($r \rightarrow a \rightarrow c \rightarrow i$). The BREAK message can be sent to tear down the new branch, in which case the in-tree delay in the GROW message needs to be updated to equal that of the old tree branch. Or the BREAK message can be sent to break the old branch based on certain optimization criteria (e.g., the in-tree delay of the new branch is smaller), and in this case the in-tree delay stored at i needs to be updated.

C. Optimization

Whenever the EW test generates a warning at an intermediate node, the node becomes a branching point and multiple tree branches may grow out from this node towards the new member.[2] The number of branching points, if not restricted, can potentially be large, which will result in large routing overhead. We define two protocol parameters that are used to restrict the number of constructed tree branches.

Maximum Branching Level (MBL): An easy way to control the number of branching points is to maintain an assertion: between the root and any on-tree node, there are at most m branching points. In other words, the maximum number of branching points is bounded by $\sum_{i=0}^{m-1}(d-1)^i$, where d is the maximum degree of a node. When $d = 2$, $\sum_{i=0}^{m-1}(d-1)^i = m$; when $d > 2$, $\sum_{i=0}^{m-1}(d-1)^i = \frac{(d-1)^m-1}{d-2}$. Such a restricted version of SoMR is denoted as SoMR-m. This number m is called the *maximum branching level*. An illustration of SoMR-3 is given in Fig. 2. SoMR-m can be easily implemented by augmenting the GROW messages with a counter.

Directivity can be implemented to discourage tree branches growing away from the new member t. When a GROW is sent from i to j, if the distance from j to t is not shorter than the distance from i to t, the counter for MBL is set to zero, indicating that there is no branching point allowed for this GROW message.

Maximum Branching Degree (MBD): A branching point may have a large number of adjacent links, which can also cause excessive overhead. SoMR-m can be further argumented with an additional parameter, *maximum branching degree*, which specifies the maximum number of GROW messages that are allowed to be sent by a branching point. If the maximum branching degree x is smaller than the node degree minus one,[3] the node selects x outgoing links (based on distance to the new member or randomly) from which GROW has not been received, and sends GROW messages out along these links.

We suggest both MDL and MBD to be implemented. With MDL = m and MBD = x, the maximum number of branching points is $\Sigma_{i=0}^{m-1}x^i = \frac{x^m-1}{x-1}$. Therefore, the overhead can be controlled by these two parameters.

[1] The green-GROW message will join this link to the multicast tree.

[2] The BREAK messages will cut all but one branch. Hence, there will be only one tree branch connecting the new member eventually.

[3] The node should not send GROW to a link from which a GROW message has been received previously.

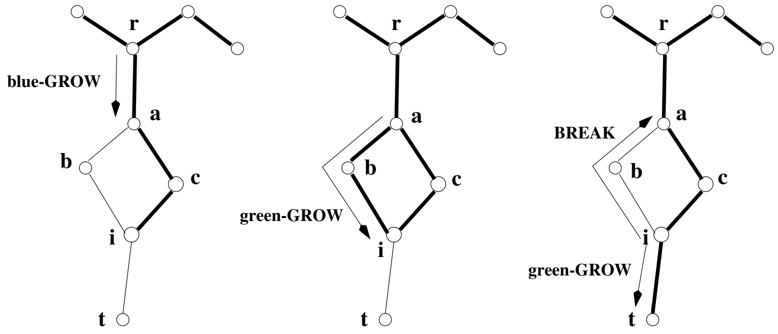

Fig. 1. Break loops

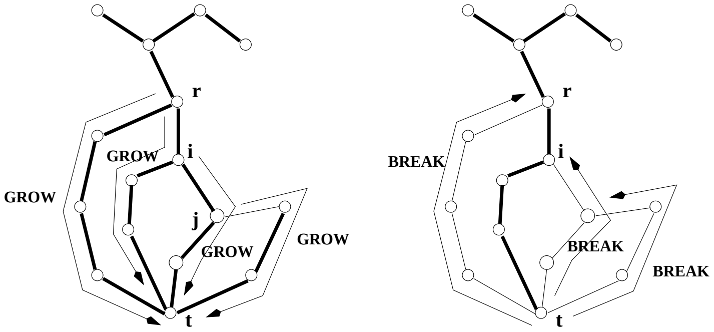

Fig. 2. an example of SoMR-3

IV. Analysis

Due to the space limitation, we omit the proof of the following theorems.

Theorem 1: SoMR never forms a persistent loop in the multicast tree.

Theorem 2: SoMR never partitions the tree.

Theorem 3: SoMR-m terminates in finite time.

Theorem 4: Suppose the unicast routing paths are the shortest paths in terms of hops. For SoMR-m, a tree branch from the root to a member is at most $2m$ hops longer than the shortest path. If the directivity is implemented, a tree branch is at most two hop longer than the shortest path.

In the following, we compare the worst case overhead of three protocols: spanning join, QoSMIC, and SoMR. To simplify the problem, we consider a network of n uniformly connected nodes. Let the diameter of the network be 2ω hops. Assume the number of nodes in the k-neighborhood of a node, $\mathcal{N}_k$, grows quadratically with k, i.e., $\mathcal{N}_k = \alpha k^2$. Thus, the diameter is given by $\alpha\omega^2 = n$. When the spanning join protocol broadcasts in a neighborhood with a radius of k hops, the number of messages sent is αk^2. Hence, in the worst case the total number of messages sent in consecutive broadcasts are

$$\sum_{k=1}^{\omega}(\alpha k^2) = \alpha\frac{\omega(\omega+1)(2\omega+1)}{6} \approx \frac{n(2\omega+1)}{6} \in O(n\omega)$$

The local search of QoSMIC broadcasts in a small neighborhood with a constant radius. The message overhead of this part can be considered as a constant. Let T be the size of the multicast tree. The worst case overhead of the tree search is $O(T)$. For a dense tree that populates the entire network, $O(T) = O(n)$.

Consider SoMR-m with MBD $= x$. The maximum number of branching points is $\frac{x^m-1}{x-1}$ (Section III-C). The maximum number of branches is $x\frac{x^m-1}{x-1}$. Note that x and m are both small constants. The length of any branch is bounded by $O(2\omega)$. Therefore, the total number of messages sent is bounded by $O(x\frac{x^m-1}{x-1}2\omega) = O(\omega)$ in the worst case. We shall do a more detailed study on overhead in Section V by simulation. What the above analysis tells us is that, as the network size increases, the worst case overhead of spanning join, QoSMIC, and SoMR increases in the order of $n\omega$, n, and ω, respectively. For a perfect uniformly-connected network with every node degree being d, $\omega = O(\sqrt[d/2]{n})$. Among the three, SoMR is the least sensitive to the size of the network, which means better scalability.

V. Simulation

Two performance metrics, *success ratio* and *average message overhead*, are defined as follows.

$$\text{success ratio} = \frac{\text{number of successful joins}}{\text{total number of join requests}}$$

$$\text{avg. msg. overhead} = \frac{\text{total number of messages sent}}{\text{total number of join requests}}$$

When the message overhead is calculated, sending a message over a path of l hops is counted as l messages.

Four multicast routing protocols were simulated: *SPR*, *SoMR-3*, *QoSMIC*, and *spanning-joins*. The maximum branching degree of SoMR-3 is 5, i.e., a branching point can send at most 5 GROW messages to its neighbors. For QoSMIC [5], the local search and the tree search are implemented as sequential procedures; the tree search is executed only when the local search fails. *Directivity*, *local minima*, and *fractional choice* [5] were also implemented. For spanning joins [3], we implemented its directed flooding version, called *directed spanning joins* [3].

The major advantage of SoMR over QMRP [2] is that SoMR does not maintain any per-group-per-join state information. Note that the multicast tree is per-group information and has to be there. QMRP needs to maintain the temporary search trees,

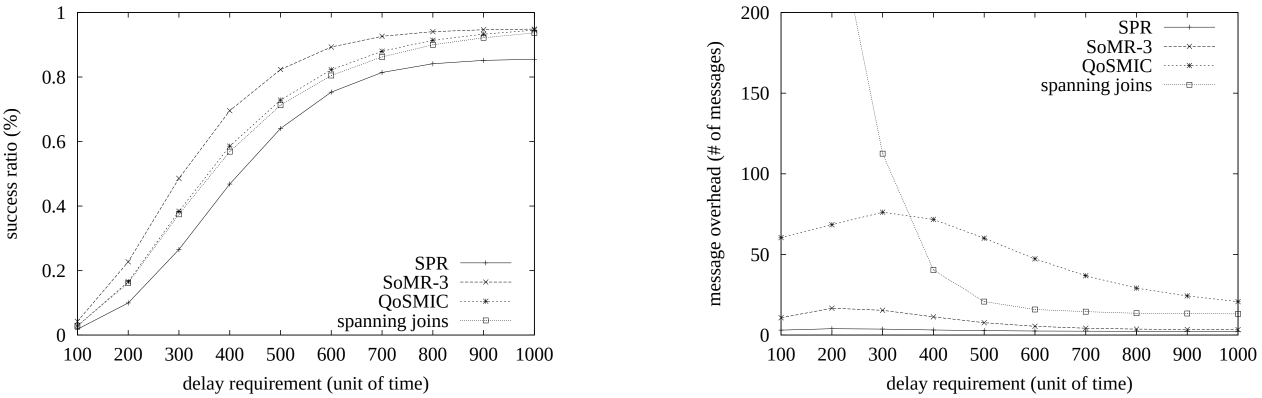

Fig. 3. Power-Law topology, 600 nodes, 5% links saturated

which is per-group-per-join information. In this section, we do not compare SoMR with QMRP because SoMR is mainly designed for additive QoS requirements such as delay whereas QMRP was designed for non-additive QoS requirements such as bandwidth.

Our simulations were conducted on Power-Law network topologies [9] with 600 nodes. In the simulation, five percents of all links in the topology are randomly selected as *saturated links*, which refuse to accept more QoS traffic due to the lack of resources. The delays of these links are thus considered to be infinite.[4] When the unicast path has a link that is saturated for QoS traffic, SPR will fail but the other protocols may still succeed because they explore more paths than the unicast one.

In each simulation run, the delays of unsaturated links are randomly generated in the range of [0, 200] units of time. The root of the tree is randomly selected, and a delay requirement for the multicast tree is set. Then, the nodes in the network start to join the tree in a random order; each node attempts once. Upon completion, the next simulation run starts. Two hundred simulation runs are conducted on each of six randomly generated topologies. The average result (success ratio/message overhead) of all simulation runs yields one data point in the figure.

Fig. 3 compare the success ratio and the message overhead of the four routing protocols. The horizontal axis represents different delay requirements of the multicast trees. The figure shows that the success ratio of SoMR-3 is better than those of QoSMIC and spanning joins. Remarkably, SoMR-3 achieves better success ratio at much lower message overhead, as shown in the right plot. When the delay requirement is small (i.e., 100), the spanning joins protocol has very large overhead (more than 600 messages per join request). That is because the multicast tree is always small and most join requests result in large scale flooding. Although the overhead of SoMR-3 is higher than that of SPR, it worth mentioning that for join requests SPR is able to find feasible paths, SoMR-3 behaves just like SPR and thus has the same overhead. Only for join requests SPR is unable to find feasible paths, SoMR-3 sends more control messages.

VI. Conclusion

We presented SoMR, a new QoS multicast routing protocol that has a favorable tradeoff between the communication overhead and the success probability. It was shown that the protocol overhead is lower then previously suggested protocols, spanning join and QoSMIC, while its success probability is higher in most cases than other protocols (In some cases QoSMIC has comparable success probability but with higher overhead). The protocol maintains no state in the network and works with both additive and non-additive QoS requirements.

References

[1] B. Wang and C.-J. Hou, “A survey on multicast routing and its QoS extension: problems, algorithms, and protocols,” *IEEE Network*, January/February 2000.

[2] S. Chen, K. Nahrstedt, and Y. Shavitt, “A QoS-Aware Multicast Routing Protocol,” *IEEE JSAC*, vol. 18, no. 12, pp. 2580–2592, Dec. 2000.

[3] K. Carlberg and J. Crowcroft, “Building Shared Trees Using a One-to-Many Joining Mechanism,” *Computer Communication Review*, pp. 5–11, January 1997.

[4] I. Cidon, R. Rom, and Y. Shavitt, “Multi-Path Routing Combined with Resource Reservation,” *IEEE INFOCOM’97*, pp. 92 – 100, April 1997.

[5] M. Faloutsos, A. Banerjea, and R. Pankaj, “QoSMIC: Quality of Service sensitive Multicast Internet protoCol,” *SIGCOMM’98*, September 1998.

[6] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, “RSVP: A New Resource ReSerVation Protocol,” *IEEE Network*, September 1993.

[7] H. Zhang, “Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks,” *Proceedings of the IEEE*, vol. 83, no. 10, October 1995.

[8] Handley and V. Jacobson, “SDP: Session Directory Protocol (draft 2.1),” *Internet Draft — Work in Progress*, February 1996.

[9] M. Faloutsos, P. Faloutsos, and C. Faloutsos, “On Power-Law Relationships of the Internet Topology,” *ACM SIGCOMM 1999*, Aug. 1999.

[4] Each link typically has a ”quota” on the maximum amount of resources allowed to be reserved for QoS traffic in order not to starve the best-effort traffic. Once this quota is reached, the link refuses to accept more QoS traffic. It is then a *saturated link*. Note that the delay of a saturated link is infinite for new QoS traffic but is not infinite for the best-effort traffic. While the underlying unicast routing algorithm works on the best-effort traffic, it may select saturated links on its routing paths.