SLRouting: Server Link Router state Routing Protocol Design and Implementation

Janaka L. Wijekoon
Hiroaki Nishi Laboratory
Dept. of Information and Computer Science
Keio Univ., Yokohama, Japan
janaka@west.sd.keio.ac.jp

Hiroaki Nishi
Dept. of System Design Engineering
Keio Univ., Yokohama, Japan
west@sd.keio.ac.jp

ABSTRACT

Packet propagation delay reduction is becoming the primary concern of the Internet. Internet service providers (ISPs) attempt to optimize packet routing to offer the best route to their subscribers by achieving desired network performance. Consequently, a route computation metric that uses packet propagation delay instead of link state will be a strong incentive for ISP routing optimization, in which case ISPs could ensure minimal delay route paths for their subscribers. To this end, we present SLRouting, a novel Interior Gateway Routing Protocol. SLRouting calculates a composite route metric using packet waiting delays of servers and routers as well as the packet propagation delay of network links. SLRouting computes the route matrix by selecting the minimal delay path for destination networks. This paper presents the first version of the SLRouting including its theory, design, and implementation notes. A prototype of the proposed protocol is implemented using the ns-3 simulator, and the results were used to evaluate the proposed protocol.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing protocols; C.2.6 [**Internetworking**]: Routers

General Terms

Design, Experimentation, Performance, Verification

Keywords

Loop-free Hybrid Routing Protocol, Routing Utilization, Traffic Engineering

AINTEC'15, November 18–20, 2015, Bangkok, Thailand.

DOI: http://dx.doi.org/10.1145/2837030.2837031

1. INTRODUCTION

Packet propagation delay, i.e., data download latency, has become the primary concern regarding Internet performance [11]. To ensure low-latency data delivery, internet service providers (ISPs) solve the traffic engineering (TE) problem (i.e., packet routing) [19]. Moreover, to offer fast data delivery, content providers (CPs) deploy their networks on ISP networks, whereas the end users can download data from the nearest server [1]. In general, ISPs are attempting to optimize packet routing at a given server [15]. However, the distributed infrastructure of CPs' server placement violates basic assumptions of TE; the traffic matrix is point-to-point and constant [8, 19]. Nevertheless, such networks introduce unpredictability in network packet routing [15].

Routing optimization, which finds the best route to a given destination by achieving desired network performance, is a major component of TE [19]. Reference [2] discusses the urge of contemplating both ISPs' and CPs' information for optimal path selection to ensure maximal routing optimization. Further, among many, references [17] and [9] discussed of using dynamically changing factors for route matrix computation. However, conventional protocols are a lack of considering the CPs' impact for packet routing [8]. In fact, they do not consider the effects of network components (i.e., packets' waiting time at a router) for route matrix computation. To achieve routing optimization, we form a hypothesis that determines minimal delay route paths by interpreting the states of network devices at a common scale, time.

This paper proposes a new server link router state routing protocol, SLRouting. Figure 1 illustrates the basic concept of SLRouting. SLRouting employs packet waiting delays on servers ($\mathring{\mathrm{d}}_j$) and routers ($\hat{\mathrm{d}}_k$) as well as packet propagation delay of links (d_{kj}) to calculate a composite metric (M_{ij}). The composite metric comprises the cumulative packet traveling delays between two nodes (i.e., i and j). The Bellman-Ford algorithm [3] is used to select the minimal delay path (henceforth called the effective path ep) between two nodes. To ensure loop-free route management, SLRouting incor-

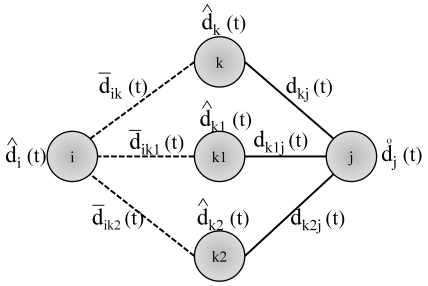


Figure 1: Basic concept of SLRouting (solid line denotes a direct link; dashed line denotes distance) i source, k intermediate nodes and j destination; note that j can be either a route or a server. Router Metric ($\hat{d}_k$): average time a given packet has to wait at the router, Link Metric (d_{kj}): average time a given packet travel through the link, Server Metric ($\dot{d}_j$): average time a given packet has to wait at the server.

porates the diffusing computation concept [5] and the coordinated update approach [7, 13]. Thus, SLRouting can be categorized as a loop-free hybrid IGRP.

The rest of the paper is organized as follows. Section 2 briefs the mathematical background of employing states of the server, router, and a link for composite metric calculation. Section 3 elaborates the protocol specifications, including its header structures, route and neighbor management. Section 4 covers the evaluation notes, including the limitations of the current version of the protocol. Finally, Section 5 concludes the paper.

2. SLROUTING STATE OF THE ART

Given the fact that SLRouting employs servers, links, and routers to calculate the *ep* between two nodes, the real challenge is to interpret those parameters at a common scale wherever they are addable. References [18] and [10] explain that a router and a server can be represented using the M/M/1 queue model such that the packet waiting time at both the server and the router is measurable. Further, links have their transmission and propagation delays. In contrast, at a given time t (i.e., update arrival time), SLRouting calculates the cumulative packet travel time between two nodes. Using the Bellman-Ford algorithm, it then selects the path with the minimum travelling time as the effective path (*ep*) (see Fig.1).

A server can be denoted using the M/M/1 queue, where the Poisson process determines the arrival rate (λ_S), and job service (μ_S) has an exponential distribution. Using the detailed explanation given in [13, 18] and the Little's formula [12], at the steady state, the packet waiting time ($\mathring{d}_j(t)$) of a given server j can be calculated using (1), where j_n denotes the server index and n = 1, 2, 3, ...

$$\dot{d}_{j_n}(t) = 1/(\mu_{j_n}(t) - \lambda_{j_n}(t)) \quad (1)$$

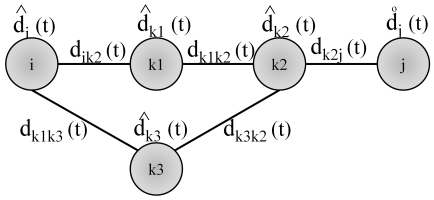


Figure 2: Five node network example

Reference [10] explains that routers are also developed on the basis of the queue model. Assuming a Poisson packet arrival rate (λ_R) and exponentially distributed service rate (μ_R), similarly, routers can also be observers by using the M/M/1 queue model. The packet waiting time ($\hat{d}_k(t)$) of a given router k can be calculated using (2), where k_n denotes the router index and n = 1, 2, 3, ...

$$\hat{d}_{k_n}(t) = 1/(\mu_{k_n}(t) - \lambda_{k_n}(t)) \quad (2)$$

The packet traveling delay of a link depends on two major points: 1.) packet transmission delay (T_{trans}) and 2.) packet propagation delay (T_{prop}). T_{trans} of a link kj (see Fig.1) depends on the available bandwidth $L_b(t) = L_c - L_l(t)$ of a link, where L_l denotes the used bandwidth and L_c represents the total bandwidth [4]. In this case, for a packet of size Z propagates through the link, and T_{trans} can be calculated using (3).

$$T_{trans}(t) = Z/L_B(t) \quad (3)$$

Further, T_{prop} of a link depends on the length and the medium of the link [4]. Consequently, total traveling delay ($d_{kj}(t)$) of a given link kj can be written using (4).

$$d_{kj}(t) = T_{prop_{kj}}(t) + T_{trans_{kj}}(t) \quad (4)$$

2.1 Using the Bellman-Ford algorithm to determine the *ep*

For the sake of exposition, let us assume a five-node network as displayed in Fig.2. Note that all nodes are connected via direct links, and j can be either a server or a router. Assuming the route is through k_1, the total traveling delay (i.e., composite metric (M_{ij})) can be calculated by (5).

$$M_{ij}^{k_1}(t) = K1[\hat{d}_i + \hat{d}_{k_1} + \hat{d}_{k_2}] + K2[d_{ik_1} + d_{k_1k_2} + d_{k_2j}] + K3[\dot{d}_j] \quad (5)$$

Similarly, the total traveling delay through node k_3 can also be written as as (6).

$$M_{ij}^{k_3}(t) = K1[\hat{d}_i + \hat{d}_{k_3} + \hat{d}_{k_2}] + K2[d_{ik_3} + d_{k_3k_2} + d_{k_2j}] + K3[\dot{d}_j] \quad (6)$$

$K1$, $K2$, and $K3$ are controllable coefficient values (henceforth called $K - Values$), and Section 2.2 explains them in detail. SLRouting selects the minimal traveling delay path between (5) and (6) as the ep_{ij}. The other

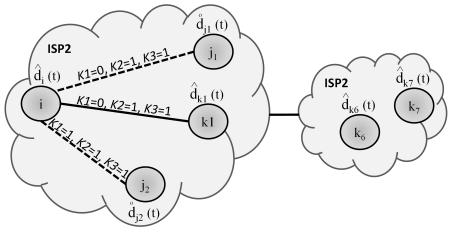


Figure 3: A simple, conceptual, graph view of two ISPs connected by default routes

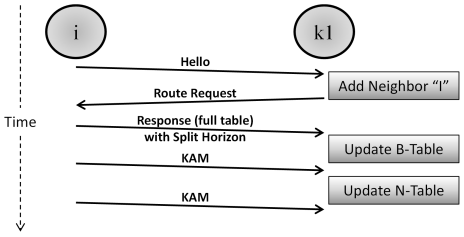


Figure 4: Protocol message exchange when an interface is activated

path will be maintained as the backup path for the given destination.

Formally, by using Bellman-Ford algorithm and employing (1), (2), (4), and (5), for a given time t, the total traveling delay of ep_{ij}; $D^{k_1}_{ep_{ij}}(t)$, can be simplified as (7). Note that (7) assumes the ep_{ij} is through k_1.

$$M^{k_1}_{ep_{ij}}(t) = d_{ik_1}(t) + \hat{d}_{k_1}(t) + M^{k_2}_{ep_{k_1 j}}(t) \qquad (7)$$

2.2 Configurable coefficients ($K - Values$)

Packet propagation delay is considered to be the most important factor of Internet performance [11]. Therefore, as described in Section 2.1, for more accurate or better route matrix calculation, SLRouting computes a composite metric using the average packet propagation delay between two nodes along with three nonnegative coefficient values, *K-Values*. The *K-Values* are introduced to enable controllability for composite metric calculation by considering $K1$, $K2$, and $K3$ for servers, links, and routers, respectively. Note that ISPs will have to determine the K-Values according to the requirement of their subscribers.

Given the fact that SLRouting assumes all servers in an ISP participate for route matrix computation, the default *K-Values* are set as; $K1$=1, $K2$=1, and $K3$=1. However, in a situation where servers do not participate in the route matrix calculation (j_1 node in Fig.3), M_{ij_1} is calculated by setting *K-Values* as $K1$=0, $K2$=1, and $K3$=1. In such cases, SLRouting works as a hybrid routing protocol by controlling only $K2$ and $K3$.

Nevertheless, as depicted in Fig.3, ISP1 computes *ep*s for the routes learned from ISP2 by implicitly setting $K1$=0, $K2$=1, and $K3$=1. This is because ISP1 assumes that ISP2 used relevant *K-Values* to calculate the *ep*s. Note that, this implementation (the first of SLRouting), is incapable of route conversion, i.e., convert routes from OSPF to SLRouting and vice versa [14]. However, protocol headers are implemented with reserved bits to introduce such features in future versions (see Fig.6).

3. SLROUTING SPECIFICATIONS

The SLRouting protocol is built by assuming three important factors: 1.) within a finite time period a node should be able to determine the existence of a new node or the connectivity loss of a neighbor, 2.) the protocol should ensure the reliability of the route update messages, and 3.) any route message event should be processed individually.

The protocol can be divided into two parts: route management and neighbor management. Neighbor management ensures the aforementioned first factor, whereas route management handles the other two factors by using the defusing computation concept [5] and coordinated update approach [7, 13]. SLRouting is intended to allow routers to exchange information for computing routes using IPv4-based networks (note that this version supports only IPv4). The protocol is developed as a UDP-based protocol and uses UDP port 275 for its route advertisements.

3.1 Neighbor management

Each node maintains a list of neighboring/adjacent nodes to ensure a loop-free route computation. The neighbor management consists of two major methods: 1.) neighbor discovery, and 2.) maintenance of the neighbor table including adding, updating, validating, and deleting neighbor records.

The building of the neighbor table will initiate when nodes (interfaces) are first activated. When an interface is activated, as shown in Fig.4, the node sends a *hello* message using the protocol structure displayed in Fig.5. When k_1 receives the *hello* message, it adds i to its neighbor table (henceforth called the *N-Table*) with relevant information. k_1 then sends a *route request message (RRQ)* to i. After receiving the *RRQ* from k_1, i sends the *route response message (RRS)* to k_1, which then updates its routing table according to the received *RRS*. Finally, in each *KAMTimer* interval, both i and k_1 exchange *Keep-alive Messages (KAMs)* using the same protocol structure displayed in Fig.5.

All records in the *N-Table* are associated with two events using two different timers: *NbrTimeout*, and *Garbage-collection* (default values are listed in Table.1). All new neighbor records are set as *VALID*, and events are scheduled to expire after *NbrTimeout* seconds. For every received *KAM* from the neighbors, the timeout event of that particular neighbor record will be re-set. However, if the neighbor record does not receive *KAM*s

Table 1: SLRouting timers and default values

Timer	Description	Default value
Periodic_Update	Amount of time between route updates	20s
TrigMAX	Maximum time between triggered updates	5s
TrigMIN	Minimum time between triggered updates	1s
Timeout	Amount of time after which a route is considered unreachable	100s
Hold_down	Amount of time after which route is moved to *M-table* from the *B-table*	60s
Garbage-collection	Amount of time after which a route is removed from the routing table	10s
KAMTimer	Amount of time between the two *KAM*s	30s
NbrTimeout	Amount of time after which a neighbor record considered unreachable	90s
Svr-RtrTimer	Amount of time a server advertises to its adjacent gateway	240s

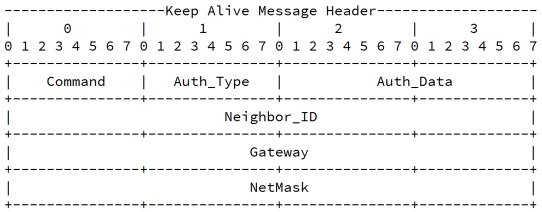
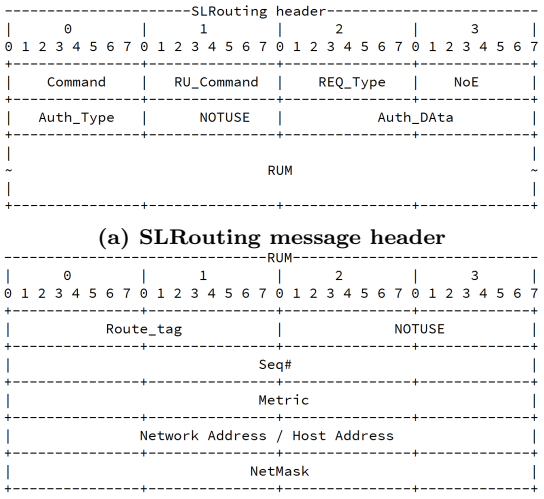


Figure 5: Hello/KAM packet structure



(a) SLRouting message header

(b) Route update message (RUM)

Figure 6: Protocol header structures

until the timeout event expires, the neighbor record will then be marked as *INVALID* and be scheduled for removal from the *N-Table* after *Garbage-collection* seconds. Further, all route records which refer to that neighbor as their gateway will also undergo a route invalidation process (a detailed explanation can be found in Section 3.2.5).

3.2 Route management

Route management is the combination of routing tables, route update structures, route update message processing and route invalidation. Given the fact that SLRouting uses defusing computation with coordinated updates, route management uses various protocol messages and several route states.

3.2.1 Routing tables

Every node configured with SLRouting is supposed to maintain two routing tables: main table (*M-Table*) and backup table (*B-Table*). Each table has at least one entry for every destination that can possibly be reached from that particular node. Each entry has the following information: the *destination network* prefix and *network mask*, *gateway IP* address, *interface ID*, *sequence number* of the last update, *metric* (i.e., propagation time), *flag* that indicates whether the route is changed, *state*, and *type* of route. In addition, for stable route update management, various timers are associated with each route (see Table.1). On the basis of the connectivity and responsiveness of neighbors, interfaces, and destination networks, each route record is assigned to one of the following states: *VALID*, *INVALID*, *BROKEN*, *DISCONNECTED*, and *LOCALHOST*.

M-Table stores 1.) route records for all locally connected networks by assigning the gateway as "*0.0.0.0*", and 2.) *ep* learned from nodes' neighbors (henceforth called the "*m-route*"). Initially, the type and the state of all routes in the *M-Table* are marked as *PRIMARY* and *VALID* unless otherwise specified.

B-Table has two route records for each destination prefix. One is the reference route to the *m-route* and is marked a *PRIMARY* (henceforth called the "*p-route*"). The second is the route with the next best propagation delay (i.e., greater than that of the *m-route*). Such records are marked as *SECONDARY* (henceforth called the "*s-route*").

3.2.2 Protocol messages

SLRouting uses packet structures given in Fig.6 for its protocol messages. For faster and reliable route update management, SLRouting uses techniques such as sequence numbering, split horizon, and poison reverse [6]. Nevertheless, *push* and *pull* message techniques are also used for the route messaging [18]. *push* messages can

be categorized as *hello*, *KAM*, *RRS*, whereas pull messages can be categorized as *RRQ*. Note that the *B-Table* is not used to generate both *RRS* and *RRQ* messages; the *B-Table* is used only to store temporary and backup route records.

Route Request (RRQ) messages are used to request either entire or partial routing table from a node. Note that this implementation supports requests only for the entire routing table. A *RRQ*, which requests entire routing table sets the SLRouting header, given in Fig.6.a, as follows: *command* → RouteUpdate, *RU_Command* → REQUEST, *REQ_Type* → EntireTable, *NOE* → 255, and both *Auth_Type* and *Auth_data* → as negotiated in neighbor discovery.

Route Response (RRS) messages are generated either as a response to a request or as route advertisement messages. SLRouting uses two types of route advertisement messages: periodic and triggered. For example, as displayed in Fig.4, when i responds to k_1, i sets the parameters of Fig.6.a as follows: *command* → RouteUpdate, *RU_Command* → RESPONSE, *REQ_Type* → 0xff, *NOE* → as per the MTU, and both *Auth_Type* and *Auth_data* → as negotiated in neighbor discovery. After setting the SLRouting header parameters, i adds RUMs for each *VALID* route record present in the *M-Table*. *NoE* is determined according to the *MTU* value of a particular interface. Depending on the *MTU* value and the number of route records in the *M-Table*, i may send more than one *RRS* message to k_1. Note that split horizon is considered while generating the *RRS* messages.

Route_Tag of RUM header (See Fig.6.b) is considered to be the most vital field. In this implementation, *Route_Tag* is segregated as follows. The most significant byte is reserved for future implementations, and the least significant byte is divided as given below.

- ▌K3 | K2 | K1 | 0 | TrigUpdate | ValidRoute | PoisonedRoute | 0 ▐

By setting $K1$, $K2$, and $K3$, SLRouting indicates the parameters used for composite metric calculation. Triggered updates are marked using the "*TrigUpdate*" flag. In addition, SLRouting uses "*ValidRoute*" and "*PoisonedRoute*" flags to mark *VALID* routes and poisoned routes (i.e., unreachable destinations) respectively.

Periodic update messages: Nodes generate periodic update messages by using routes in the *M-Table*, and employing both split horizon with poison reverse techniques at each *Periodic-Update* interval. Nodes use interfaces' *MTU* values to calculate the number of *RUM*s (i.e., set *NOE* in Fig.6.a) that can be added to an advertisement. Further, nodes forward periodic advertisement messages to all neighbors present in the *N-Table*.

Triggered update messages: Triggered updates are introduced to provide fast route propagation. Using triggered update messages, the route convergence time of the Bellman-Ford algorithm is reduced to $O(|k|*|E|)*5s$, where $|k|$ is number of nodes, $|E|$ is number of links, and 5s is the maximum time between two triggered updates. However, given the explanations in [4] and [6], triggered updates may cause excessive loads on the network links, particularly when a large number of routers are connected using slow links.

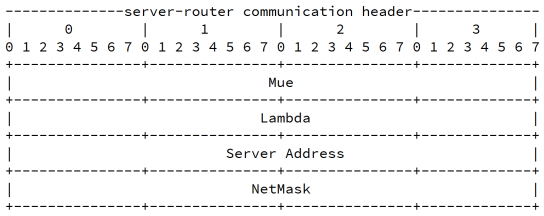


Figure 7: Server-router communication header

To avoid excessive loads, SLRouting uses two provisions: 1.) only changed routes are considered for triggered updates; 2.) to limit the frequency of triggered updates, SLRouting uses a waiting timer, *TriggerHoldDown*, between two triggered update messages. The *TriggerHoldDown* timer is an event that expires within a random interval between *TrigMAX* and *TrigMIN*.

For complex networks, even with the triggered updates, the route convergence time is quite high [20]. Therefore, to handle critical scenarios such as advertisements about unresponsive neighbors or broken routes, SLRouting uses fast triggered update messages. In such scenarios, before *TriggerHoldDown* expires, nodes send fast triggered update messages by setting *TrigUpdate* and *BrokenRoute* flags in the *Route_Tag* of SLRouting header. Such messages will propagate through the network with a maximum link propagation delay.

3.2.3 Server-Router Communication Protocol

If the $K3$ controllable coefficient is considered for route computation, SLRouting requires the servers to advertise their usage information to the gateway router (assuming that the gateway routers are configured with SLRouting). The gateway router is the device that connects end hosts to the ISP network. For example, the gateway router can be a router provided by an ISP. All servers that participate in SLRouting must have the IP address of the gateway router configured.

Servers advertise their usage information at each *SvrRtrTimer* period by filling the server-router communication header (SRC) illustrated in Fig. 8. Servers periodically advertise their details including the IP address, network mask, and μ and λ.

When a gateway router receives an *SRC* message, using (8), the router calculates the average packet processing delay at server j (see Fig.2). The router then updates the associated route record of the *M-Table*. As a matter of fact, the route record is updated regardless

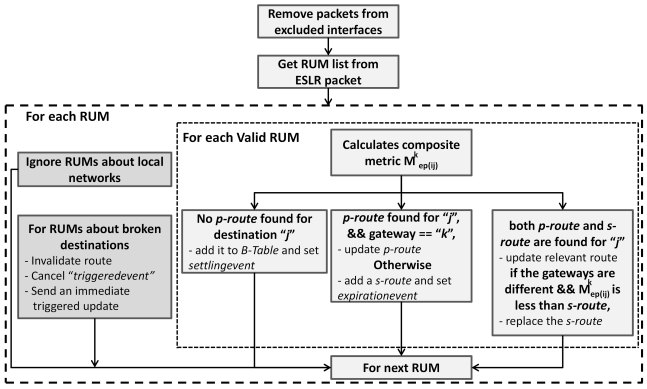


Figure 8: Processing steps of RUMs

the existing route metric value (i.e., greater or lesser).

$$\dot{d}_j = 1/(j_\mu - j_\lambda) \tag{8}$$

3.2.4 Process route responses (RRS)

Nodes can receive response message for one of the following reasons: as a response to an *RRQ*, as a periodic update message, as a triggered update message, or as an *SRC*. Regardless of the message type, initial processing is same for *RRS*. At the initial processing, nodes validate message packets for their authenticity. Packets are then analyzed for IP and transport layer information (i.e., socket tags). Finally, according to the command of the SLRouting header, nodes forward messages for process modules. Note that the processing of *KAM*, *RRQ*, and *SRC* messages were explained in Section 3.1, Section 3.2.2, and Section 3.2.3, respectively.

Owing the fact that the processing of *RRS*s may edit the routing table, the messages should be validated carefully. First, all messages received via excluded interfaces are ignored. If the message is from a legitimate interface, then the message is authenticated to determine whether the message is from a valid neighbor. Because SLRouting uses link local broadcast/multicast addresses to send route update messages, it is also worth checking to see whether the update message is from one of its interfaces. Such messages are also ignored.

After validating the update message, the node processes *RUM*s individually. Figure 8 illustrates the processing steps. If the *RUM* is about a poisoned route, i.e., unresponsive network, the node then invalidates the corresponding routes on both the *M-Table* and the *B-Table*. After that, the node sends the triggered updates about the unresponsive routes (see Fig.8). Otherwise, as explained in Section 2.1, for the destination network prefix (j), the node calculates the composite metric (M_{ij}^k). The node uses the following parameters to calculate M_{ij}^k: average processing delay of processing node (i), average link delay between i and the adjacent node k, and the metric value listed in the RUM ($\bar{\mathrm{d}}_{kj}$).

The node then checks whether an *m-route* and a *p-route* are available for j. If no route is found, the node adds the j to the *B-Table* as a *p-route* and sets a *settling-event*. In the case where a node finds that a *p-route* is available in the *B-Table* for j, the node processes the *RUM* as follows.

If the gateway of the *p-route* is k, the node updates the *p-route* and resumes its existing event. Otherwise, the node creates an *s-route* to the destination j using the values of *RUM*, and sets an expiration event. In case both the *p-route* and the *s-route* are found for j, the node selects the relevant route record and updates it by comparing the sender address (k) with a gateway address. However, if none of the routes have their gateway as k, and the calculated M_{ij}^k is smaller than the metric of the *s-route*, the node replaces the existing *s-route* by the values of new *RUM*.

*p-route*s in the *B-Table* will be moved to the *M-Table* after expiring the settling event. This event might either add a new route to the *M-Table* or update an existing *m-route*. In either case, SLRouting sets the *changed* flag, sets status as *VALID* and, marks the route as *PRIMARY*. In addition, an *expiration-event* will be added to the updated route. Note that, while updating *p-route*s in *M-Table*, SLRouting make sure that *B-Table* does not posses a stable *s-route* which has less metric than that of the *p-route*.

SLRouting uses highly varying parameters, i.e., average packet waiting time at a router, to calculate the composite metric. Therefore, an argument can be made that highly varying parameters may cause route oscillations or frequent route update messages. Nevertheless, it is possible that one or more events can occur before completion of a particular event (e.g., a particular interface goes down and comes back up frequently). SLRouring uses several techniques to overcome such problems.

The first technique is that SLRouting uses hold-down timers. When a hold-down timer starts for a particular event, another event for the same route is not permitted to be processed in the node until the first hold-down timer expires (e.g., no route in the *M-Table* will update until the route is settled in the *B-Table*). The second technique is that SLRouting uses stable routes (i.e., routes in the *M-Table*) to generate update messages and send them periodically rather than flooding as link-state protocols do [13]. Thus, periodic updates may not cause route oscillations. However, given the fact that SLRouting uses triggered update messages, nodes may create *chatty* mode updates [13].

3.2.5 Route invalidation

SLRouting uses route invalidation to handle route records that move from *VALID* state to *INVALID*, *BROKEN*, or *DISCONNECTED* states.

- The associated expiration event has expired (move to *INVALID*).

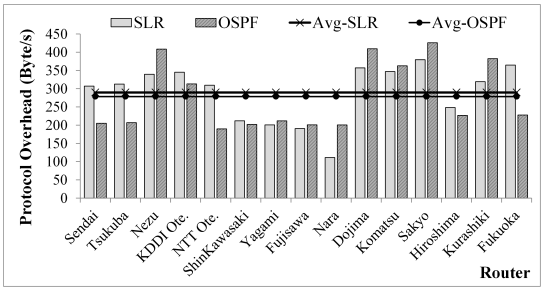


Figure 9: Protocol overhead for nodes

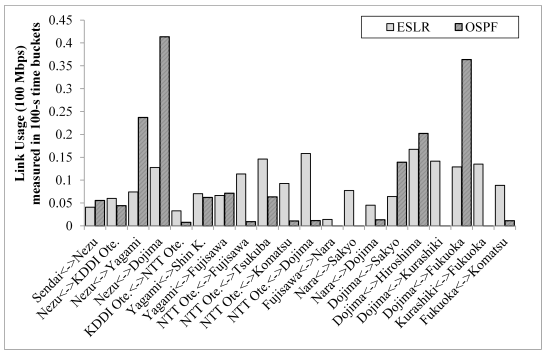


Figure 10: Link utilization

- The attached interface is set as down, or a neighbor record is invalidated and the neighbor module invalidates all route records referring to the invalidated neighbor as their gateway (moved to *DISCONNECTED*).

- A broken destination prefix is received from a neighbor (moved to *BROKEN*).

Depending on the state and the routing table, nodes process the changed routes as follows:

- An *m-route* is invalided by changing its state to *INVALID*, and the SLRouting checks for the associated *p-route* and *s-route* that matches the destination of the *m-route*. If no *s-route* is found, then the *m-route* is updated using the latest values of the *p-route*. However, if an *s-route* is found and the metric of the *s-route* is smaller than that of the *p-route*, both the *m-route* and the *p-route* are then replaced using the values of the *s-route*.

- An *m-route* is invalidated by changing its state to *DISCONNECTED*, and an *s-route* is found. Both the *m-route* and the *p-route* are then replaced using the values of the *s-route*. Otherwise, events will be scheduled (using *Garbage-collection* time) to remove both the *m-route* and the *p-route*.

- An *m-route* is invalidated by changing its state to *BROKEN*. Without further ado, events are scheduled to remove both the *m-route* and the *p-route* from the routing tables.

- If an *s-route* is invalidated, regardless of the route state, an event is scheduled to remove the route.

Note that, owing to the protocol specification, *p-routes* are not designed to be invalidated independently.

4. EVALUATION

The first prototype of the SLRouting protocol is implemented using the ns-3 simulator. The ns-3 simulator is used because it supports most of the general packet structures, and its IP stack is similar to the Linux IP stack. The first version of SLRouting is publically available and can be found in reference [20].

For the basic functional evaluation, we used WIDE network's topology [16]. By assuming the WIDE network as an ISP, for more realistic evaluation, we placed both content servers and clients at random locations. We made sure that packet arrivals for each source and destination pair, including routers, followed a Poisson process.

We executed five simulations; in each simulation, we randomly disconnected links and routers to check the message flow and the route recovery features of the protocol. We noticed that SLRouting was able to recover a broken link within of 2.5 to 10 ms. Moreover, the protocol displayed 99% packet delivery.

The SLRouting first evaluated for routing overhead compared to OSPF. As plotted in Fig.9, on average, both OSPF and SLRouting create similar overhead for the routers. However, when considering routers such as KDDI-Otemachi and NTT-Otemachi, the proposed protocol results in reasonable protocol overhead. The reasons are that SLRouting still does not support route summarization [18], and this in turn increases the size of the routing tables of those routers. Such circumstances may cause chattiness in the protocol [18]. Subsequently, those routers must process more route update messages.

We then compared the primary goal of TE: resource utilization. Observing the results plotted in Fig.10, we can notice that OSPF was unable to utilize the available network links properly. SLRouting, however, utilized almost every link in the topology. Thus, none of the links display any excessive load compared to OSPF. This is because SLRouting performs packet routing based on the propagation delay, and the propagation delay is thus calculated mainly according to the states of the network resources. Strategically, if a node identifies that a network path (i.e., a route) is busy, and the propagation time is higher, the node then quickly switches to the backup path. It is then proved that the proposed hypothesis: determine minimal delay route paths according to the states of network devices, can be used to

achieve routing optimization.

5. CONCLUSIONS

This paper introduced a novel routing protocol, SLRoutuing, which computes the route matrix by calculating the cumulative packet propagation delay. The SLRouting is introduced to contemplate both ISP and CP information to facilitate TE according to the requirements of both the ISP and the CP. SLRouting ensures loop-free route management using diffusing computation concept and the coordinated update approach. The first version of SLRouting was implemented using the ns-3 simulator [20]. Results suggest that routing optimization can be achieved by contemplating both ISP and CP information for packet routing.

This study is, however, an ongoing research. The next version, Extended SLRouting (ESLR), will be compatible with route summarization (i.e., VLSM), IPv6, and interoperability. Nevertheless, for rigorous test and evolution under real ISP conditions, the Rocket Fuel networks will be used with OSPF-TE configure on it.

6. ACKNOWLEDGMENTS

We express our sincere gratitude to Prof. Fumio Teraoka for his invaluable discussions for this project. This work was also partially supported by funds of SECOM Science and Technology Foundation, MEXT/JSPS KAKENHI Grant (B) Number 24360230 and 25280033, MEXT Scholarship for Research Students, Keio University Doctorate Student Grant-in-Aid Program, and the Keio University KLL Ph.D. Research Grant Program.

7. REFERENCES

[1] Akamai. Facts and figures. https://www.akamai.com/us/en/about/facts-figures.jsp, Accessed:2015/Aug.

[2] Antic, M., Maksic, N., Knezevic, P., and Smiljanic, A. Two phase load balanced routing using ospf. *Selected Areas in Communications, IEEE Journal on 28*, 1 (January 2010), 51–59.

[3] Cheng, C., Riley, R., Kumar, S. P. R., and Garcia-Luna-Aceves, J. J. A loop-free extended bellman-ford routing protocol without bouncing effect. *SIGCOMM Comput. Commun. Rev. 19*, 4 (Aug. 1989), 224–236.

[4] Dally, W., and Towles, B. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[5] Garcia-Lunes-Aceves, J. Loop-free routing using diffusing computations. *Networking, IEEE/ACM Transactions on 1*, 1 (1993), 130–141.

[6] Heap, G. T., and Maynes, L. *CCNA Practical Studies*. Cisco Press, 2002.

[7] Jaffe, J., and Moss, F. A responsive distributed routing algorithm for computer networks. *Communications, IEEE Transactions on 30*, 7 (July 1982), 1758–1762.

[8] Jiang, W., Zhang-Shen, R., Rexford, J., and Chiang, M. Cooperative content distribution and traffic engineering in an isp network. *SIGMETRICS Perform. Eval. Rev. 37*, 1 (jun 2009), 239–250.

[9] Khanna, A., and Zinky, J. The revised arpanet routing metric. *SIGCOMM Comput. Commun. Rev. 19*, 4 (Aug. 1989), 45–56.

[10] Kloth, A. K. *Advanced Router Architectures*. CRC Press, Inc., 2005.

[11] Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J., and Jahanian, F. Internet inter-domain traffic. *SIGCOMM Comput. Commun. Rev. 41*, 4 (Aug. 2010).

[12] Little, J. D. C. Or forum—little's law as viewed on its 50th anniversary. *Oper. Res. 59*, 3 (May 2011), 536–549.

[13] Medhi, D., and Ramasamy, K. 7 - {IP} traffic engineering. In *Network Routing*, D. Medhi and K. Ramasamy, Eds. Morgan Kaufmann, 2007, pp. 194 – 236.

[14] Medhi, D., and Ramasamy, K. 8 - {BGP}. In *Network Routing*, D. Medhi and K. Ramasamy, Eds. Morgan Kaufmann, San Francisco, 2007, pp. 238 – 279.

[15] Poese, I., Frank, B., Smaragdakis, G., Uhlig, S., Feldmann, A., and Maggs, B. Enabling content-aware traffic engineering. *SIGCOMM Comput. Commun. Rev. 42*, 5 (Sept. 2012), 21–28.

[16] Project, W. Wide internet. http://two.wide.ad.jp/, Accessed:2015/Aug.

[17] Sobrinho, J. Algebra and algorithms for qos path computation and hop-by-hop routing in the internet. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2001), vol. 2, pp. 727–735.

[18] Twain, M. 3 - routing protocol framework and principles. In *Network Routing*, D. Medhi and K. Ramasamy, Eds. Morgan Kaufmann, 2007, pp. 194 – 236.

[19] Wang, N., Ho, K., Pavlou, G., and Howarth, M. An overview of routing optimization for internet traffic engineering. *Communications Surveys Tutorials, IEEE 10*, 1 (First 2008), 36–56.

[20] Wijekoon, J. Server link router state routing protocol for ns3. https://github.com/janakawest/ESLR, Accessed:2015/Aug.