KAR: Key-for-Any-Route, a Resilient Routing System

Rodolfo R. Gomes[†] Alextian B. Liberato[*] Cristina K. Dominicini[*] Moises R. N. Ribeiro[*] Magnos Martinello[*]
[*]*Federal University of Espírito Santo (UFES)*
[†]*Federal Institute of Education, Science and Technology of Espírito Santo (IFES)*
{*rodolfo,alextian,cristina.dominicini*}@*ifes.edu.br*
moises@ele.ufes.br, magnos@inf.ufes.br

Abstract—**This paper proposes KAR (Key-for-Any-Route), a new intra-domain resilient routing system in which edge-nodes set a *route ID* to select any existing route as an alternative to safely forward packets to their destination. In KAR routing system, a route is defined as the remainder of the division between a route ID and a set of switch IDs along the path(s) between a pair of nodes. KAR-enabled switches explore the existing routes by using special properties of Residue Number System as our encoding technique. Packets are deviated from the faulty link (liveness condition) with routing deflections. Deflected packets are guided to their original destination due to resilient forwarding paths added to the *route ID*. Three deflection methods are discussed along emulation experiments. Results show that KAR efficiently allows deflected packets to automatically reach their destination, imposing a bound on packets disordering measured in TCP throughput.**

Index Terms—**Source routing, routing deflections, reliability.**

1. Introduction

Networking is evolving toward extremely complex systems with ever increasing state maintaining demands required by elements sitting in the network core. This problem is particularly critical with new paradigms for future network architectures based on software-defined networking (SDN) [1]. Its fundamental point is to decouple networking data plane from control plane; the latter holds the network intelligent decisions while the former merely hosts executive tasks based on tables to process incoming flows. Individual flows are defined by packet match fields, flow priority, counters, packet processing instructions, flow timeouts and a cookie. Controllers also decide on new forwarding rules for new incoming flows and faults detected by data plane elements.

Entries in flow tables can be set proactively for services requiring strict bandwidth and delay control policies, but current limited flow table space will not meet the needs of core networks for a huge number of per-flow states to be simultaneously handled and stored in every single node. On the other hand, reactive schemes, suited to best-effort services such as online load balancing, will face both control plane communication and data plane convergence latency limitations. Control plane overloading due to the large number of events to be processed, such as first packet forwarding to controller, and per-hop flow installation procedure, will not address the needs of core networks.

Per-flow state limitation for core networks can be tackled by encoding a particular physical path to be used, a service chaining scheme, or any other scheme right on the packet header. This paradigm is known as source routing (SR) or explicit routing. Basically, the edge nodes would stick such labels (encoding a desired service in the core) to incoming packets, core elements would simply implement the required policy and egress edge would remove such service labels. In this way, a tunnel, which is a path in a forwarding fabric, would be implemented to transport packets across large cores by controlling them only at the edges [2].

Unlike traditional routing protocols that are network-controlled and all routes are computed within the network, source-routed network architectures enable sources to select the path taken by their packets. SR technique is the basis of many proposals to improve the reliability and performance of networks, essentially because it provides path diversity that reduces the dependence on a single network path with undesirable characteristics [3], [4], [5]. Therefore, SR has been revisited as a promising approach to improve flexibility of the network layer in future Internet architectures.

Despite the advantages, a classical problem in the source routing (SR) approach is how fast it reacts to failures of a link or node that belongs to a path. There are basically two high-level approaches to address the *fast failure reaction* problem. The first approach consists of sending a failure notification to the source node. While it improves failure reaction time, the source still must wait to receive the notification message. Until that failure notification is received, packets that had already left the source node are dropped. In the second approach, the failure reaction happens within the network by using alternative paths, that is a protection route, to each destination. Thus, a node can locally switch to the alternative path as soon as the node detects a failure on one of its directly connected links affecting that path. However, this approach requires every node/switch to be able to compute and store the backup paths, so that there is a dependency between each switch forwarding table and the topology of the entire network [6].

In this paper, we propose **KAR (Key-for-Any-Route)** that is a new intra-domain routing system with a novel fast failure reaction mechanism which combines the benefits of

978-1-5090-3688-2/16 $31.00 © 2016 IEEE
DOI 10.1109/DSN-W.2016.11

source-controlled routing with driven deflections as additional forwarding paths to provide network routing resiliency through path diversity. A route in the KAR routing system relies on the remainder of the division between a route ID and a set of local switch IDs on the path(s) between a pair of nodes. As for the fast failure reaction mechanism, as soon as a KAR switch realizes a link failure, it randomly deflects packets that would go through that link instead of dropping them. Those packets, then, pass through a diverse set of switches carrying driven deflections forwarding paths embedded in a route ID at packet header (loop-free for safety condition). The KAR coding technique is designed by exploiting special properties from Residue Number System (RNS) [7], [8], [9]. Thus, the KAR approach addresses link failures keeping the network connectivity allowing in-flight packets along the failed path to reach their destination (liveness condition).

Three packet deflection techniques are proposed, being packet random walk deflection used as the lower bound. Also, we analyze the network protection provided by the driven deflection forwarding paths considering realistic network topologies. Finally, we evaluate the TCP throughput in a emulation tool for single link failures discussing the trade-offs between protection versus throughput.

2. KAR Resilient Routing System Design

In this section, we present the concept of Key-for-Any-Route (KAR) discussing the design challenges. KAR achieves the main benefits of source routing architectures (flexibility in route selection and scalability) with a fast failure reaction by using driven-deflection forwarding paths further explained.

Figure 1 illustrates an overview of the concept of *KAR* design with 6-node network. KAR introduces an explicit distinction between the network edge and core. Edge nodes inspect the incoming packet (which express where to deliver the packet) and then attach a route identification (Route ID) into the packet header which is used for forwarding within the core. The core switches are built just to deliver packets to the destination. KAR route IDs have meaning only within the core and are completely decoupled from the host protocol (e.g. IPv4 or IPv6) used by the host to express its requirement to the network. There is one additional component: the network controller, that is in charge of Switch ID handling and routing decisions.

The edge nodes embed a route identification (Route ID) in packet header when a packet enters the KAR network. They remove it at egress point.

Every core node has an unique Switch ID bound to it and the set of Switch IDs in the network must be coprimes integers. These nodes do not have a forwarding table. Instead, their IDs together with packet Route ID are used for packet forwarding as it will be discussed later. The ID assignment can be done by local setup or by a network controller entity.

Finally, the router component of network controller is in control of routing decisions. It knows the entire network topology, including the Switch IDs of core nodes and any information that is important to traffic engineering. When a route is selected, it computes a Route ID, i.e. an integer number that represents a path along the core network. This number is generated by using an encoding based on the Switch IDs that belong to the desired route and on those switch output port indexes. The routing algorithm is out of the scope of this work.

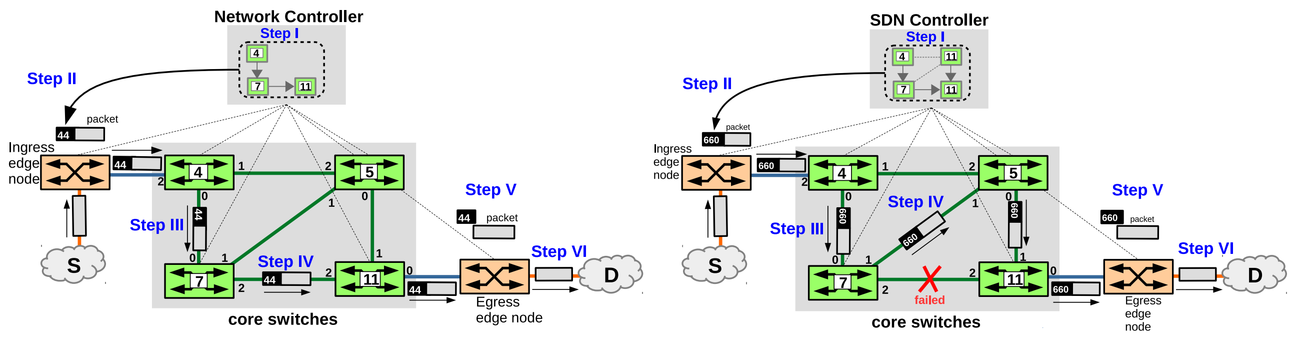
Suppose the conventional edge node S in Figure 1(a) wishes to communicate with another conventional node D through a *KAR* enabled network. The switch IDs are $\{4, 5, 7, 11\}$. Even though 4 is not a prime number, it can be used as identifier since the KAR requirement is that switches IDs must be coprimes, i.e. they do not share a common factor.

By using a selected routing algorithm (e.g. shortest path), the controller selects an end-to-end path across the *KAR* network. For instance, it chooses the set of switches $S = \{4, 7, 11\}$ composing the primary path and using, respectively, their output interfaces $P = \{0, 2, 0\}$. Then, KAR computes an unique *Route ID*, e.g. $R = 44$ (**Step I**) that should be assigned to the header of the incoming flow packets by the ingress edge node (**Step II**). KAR *route ID* computation exploits RNS properties, and will be latter explained (section 2.2).

The next hop in the KAR routing system is computed by the remainder of the division (denoted as $< a >_b \equiv a \ modulo \ b$) between the *route ID* ($R = 44$) and the *Switch ID* at every switch along that route ($S = \{4, 7, 11\}$). Thus, when switch_ID 4 (SW4) receives a packet with route ID ($R = 44$), it forwards packet to port $< 44 >_4 = 0$ (**Step III**); then, SW7 forwards it to the port $< 44 >_7 = 2$ (**Step IV**); after, SW11 forwards it to port $< 44 >_{11} = 0$ (**Step V**), reaching the egress edge node that removes the KAR *route ID* from the packet header (**Step VI**) and delivers it to node D.

In the case of link failure, one traditional approach is to notify the controller, which, in turn, recalculates the route ID excluding the faulty link from the possible paths. The problem with this approach is that all packets sent by the source before the route ID modification will be lost. To avoid packet loss (*Hitless* property), a typical mechanism for fast failure reaction is packet deflection. Deflection routing techniques are conceptually simple and allow every switch to independently decide which packets to forward to any available link [3]. When a switch detects a failure in one of its links (i.e. output port is under failure) it chooses one of its healthy ports to forward the packets. The choice of available ports is random (e.g. uniform distribution), and it leads the packets to unexpected routes (i.e. non-shortest path).

Furthemore, deflection routing may form transient loops [3]. To tackle this issue, the KAR mechanism provides a guarantee of loop-free routing even in the event of a link failure based on its *Driven Deflections* property. To this end, it is necessary to compose protection paths that are responsible for driving deflected packets to the destination by adding new nodes in the computation of the route ID. To illustrate this concept using the scenario shown in Fig. 1(b), let us assume that, in (Step I), SW5 was proactively included in the route ID as a protection path that delivers the deflected



(a) (b)

Figure 1. KAR design: (a) routing system based on shortest path, (b) fast failure reaction with driven deflection forwarding paths.

packets to SW11 when a failure happens at link SW7-SW11 (resulting in $R = 660$). Consider, also, that the selected deflection technique chooses randomly between the available ports when a link fails. Thus, when link SW7-SW11 fails, SW7 chooses between port 0 (SW4) or port 1 (SW5) to forward packets. The steps II and III happen similarly in Figures 1(a) and 1(b). However, all the packets that reach SW5 $< 660 >_5 = 0$ by deflection in SW7 (Step IV) will be forwarded to SW11. It is important to observe that a logical tree with its root at destination SW11 has been built to taken advantage of additional path to SW11 passing through SW5.

2.1. Deflection Techniques

We propose the following three deflection routing techniques in order to make deflected packets reach the destination by the *Driven Deflections* property or even by chance. Our assumption is a uniform distribution for randomness.

Hot-Potato (HP): once a packet is deflected, it follows a complete random path in network.

Any Valid Port (AVP): the switches always do the modulo operation to define the output port of a packet. When the result does not represent a valid port ID (it does not exist or it is not available), they chooses at random an active port and send packet to it.

Not the Input Port (NIP): the method described above (AVP) is improved by excluding the input port from the set of next-hop candidates even when the computation of output port tells to send a packet back to it. Besides generating less random paths, it avoids routing loops between two nodes.

This paper uses Hot-Potato deflection just as a reference method in order to validate the contribution of other ones.

NIP algorithm is shown Algorithm 1. As for NIP, the switch extracts the value of *route ID* from each packet and its input port. After that, it computes the output port and checks if this port is available and is not the input port. If so, this packet will be forwarded. Otherwise, the NIP-enabled switch chooses randomly one of available ports to send packets, not including the input port. The only difference between NIP and AVP is that the latter allows to use its incoming port as an outgoing port in any case.

Algorithm 1 NIP Deflection Technique

```
1: $route\_id \leftarrow \text{EXTRACT}(packet)$
2: $in\_port \leftarrow \text{GETINCOMINGPORT}(packet)$
3: $output \leftarrow (route\_id \mod switch\_id)$
4: while not $\text{AVAILABLE}(output)$ or $output = in\_port$ do
5:     $output \leftarrow \text{RANDOM}(0, total\_number\_port)$
6: end while
7: return output
```

There is a good reason to propose AVP or NIP: after deflection, a packet may arrive at a node included in the route ID. From there, it will follow the computed path once again, ceasing its random walk. Note that in Fig. 1(a), without any *Driven Deflection* Forwarding Paths, a packet arriving at SW5 has 50% probability to go to SW11. In contrast, the same scenario with the addition of SW5 in the route ID and the use of NIP deflection technique cause all the packets to be driven through this forwarding path (SW5$\rightarrow$SW11).

A final remark in terms of design is that an edge node can receive a packet not addressed to it. In this case, the edge node can choose two approaches: it directly returns the packet to the network without any change or it sends the packet to the controller in order to change the packet route ID appropriately before returning it to the network. In all our tests, we considered this second approach, where the controller recalculates the route ID based on the best path from the edge node to the destination.

2.2. Encoding the Forwarding Paths

The KAR switches just need to know its own switch ID and to read the route ID in the packet header in order to determine where they should send the packet to: the output port is the result of the modulo operation between the route ID and its switch ID. This is possible due to properties of the Residue Number System (RNS).

Let S be a set $S = \{s_1, s_2, \ldots, s_N\}$ of the N switch IDs on the desired path, in which all elements are pairwise coprimes numbers. Let P be a set of outgoing ports $P =$

$\{p_1, p_2, \ldots, p_N\}$, where p_i is the outgoing port index for the packet at the switch s_i.

Let M be

$$M = \prod_{i \in S} s_i \tag{1}$$

A number $R \in \mathbb{N} | 0 \leq R < M$ can be represented by a residue set given a basis modulo set S:

$$R \overset{RNS}{\longrightarrow} \{p_1, p_2, \ldots, p_N\}_S \tag{2}$$

, where

$$p_i \ = \ R \ modulo \ s_i \tag{3}$$

The network controller must find out the value of R (the explicit route ID), given a modulo set S (the switch IDs), and its RNS representation P (the switch output ports).

The Chinese Remainder Theorem [10] states that it is possible to reconstruct R through its residues in a RNS as follows:

$$R =< \sum_{i \in S} p_i \cdot M_i \cdot L_i >_M \tag{4}$$

where

$$< a >_b \ \equiv \ a \ modulo \ b \tag{5}$$

$$M_i \ = \ \frac{M}{s_i} \tag{6}$$

$$L_i \ = \ < M_i^{-1} >_{s_i} \tag{7}$$

Eq. (7) means that L_i is the modular multiplicative inverse of M_i. In other words, L_i is an integer number such that

$$< L_i \cdot M_i >_{s_i} = 1 \tag{8}$$

Returning to the example of this section, the computation of route ID from S to D is obtained as follows:

$switches = \{s_1, s_2, s_3\} = \{4, 7, 11\}$
$ports = \{p_1, p_2, p_3\} = \{0, 2, 0\}$
$M = 4 \cdot 7 \cdot 11 = 308$
$M_1 = 77, M_2 = 44, M_3 = 28$
$L_1 =< M_1^{-1} >_{s_1} =< 77^{-1} >_4 = 1$
$L_2 =< 44^{-1} >_7 = 4$
$L_3 =< 28^{-1} >_{11} = 2$
$R =< L_1 \cdot M_1 \cdot p_1 + L_2 \cdot M_2 \cdot p_2 + L_3 \cdot M_3 \cdot p_3 >_M$
$R =< 0 + 352 + 0 >_{308} = 44$

With the Driven Deflection Forwarding Paths, the route ID is computed as follows:

$switches = \{4, 7, 11, 5\}$
$ports = \{0, 2, 0, 0\}$
$M = 4 \cdot 7 \cdot 11 \cdot 5 = 1540$
$M_1 = 385, M_2 = 220, M_3 = 140, M_4 = 308$
$L_1 =< 385^{-1} >_4 = 1$
$L_2 =< 220^{-1} >_7 = 5$
$L_3 =< 140^{-1} >_{11} = 7$
$L_4 =< 308^{-1} >_5 = 2$
$R =< 0 + 2200 + 0 + 0 >_{1540} = 660$

It can be noticed in Eq. (4) that the route ID does not store or keep the information about the switch sequence the packet would travel along. Each switch data (switch ID and port ID) belongs to its own addend of the summation and is does not influence the other summation addends. As the finite summation is commutative, the switch order is irrelevant to derive the route ID. The independence of switch sequence is also valid in the output port computation (Eq. 3). This property allows us to embed, in the route ID, extra switches that are disjoint of the desired route. This is the fundamental concept of the Driven Deflection Forwarding Paths, and it is useful to protect a desired route when a packet is deflected due a faulty link since it is possible to provide guide deflected paths to the destination via path segments.

2.3. Encoding Size

The theorem of Euclidean division states that a remainder r of a division computation is an integer such that $0 \leq r < b$, where b is the divisor. Thus, Eq. (4) allows inferring that the route identifier value R lies in the integer set $\{0, 1, 2, \ldots, M-1\}$, as it is an integer division remainder.

As the route ID R is embedded in the packet header, its bit-length affects the packet overhead. The maximum number of bits required by a route ID can be computed by as follows:

$$bit\_length(R) = \lceil log_2(M-1) \rceil \tag{9}$$

Eq. (9) states that the higher the value of M, the larger the maximum required bit length. Remember that M is the multiplication of the switch IDs along the desired route. Therefore, as more switches are used to build the path, more bits are required to represent the route ID. This restriction should be considered for implementation purposes.

If the route and all the designed Driven Deflection Forwarding Paths do not fit the Route ID field length, the source routed path cannot be fully protected. However, partial protection can be used in a similar way to the loose source routing proposed by IP [11]. Instead of setting the alternative paths entirely, one can set part of them. These path segments would guide a deflected packet that occasionally reaches a switch that belongs to them.

Numerical examples of bit length requirement are presented later in Section 3.

3. Evaluation

Our investigation of KAR resilient routing considers two different topologies: one general with a 15-node network (Fig. 2) to represent an experimental scenario to illustrate the main insights of KAR routing deflection approach; and other with a 28-node network (Fig. 6) to represent an existing research and education network scenario.

In order to evaluate the impact of the packet disordering and jitter due to a link failure and the deflection routing, we implemented a KAR prototype as proof-of-concept for the proposed deflection techniques in the Mininet network emulation environment. It was developed based on the following components:

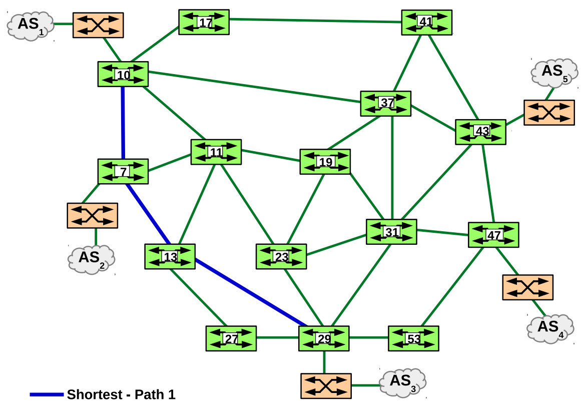


Figure 2. KAR routing system for a 15-node network.

- POX v0.3.0 as the network controller;
- OpenFlow 1.3 Software Switch which is a user-space software switch implementation;
- Mininet v2.1.0 for network emulation.

The base code of software switch was modified in order to support the three deflection techniques (HP, AVP and NIP) and the modulo computation for forwarding purposes.

Once the network is up and running, tools like *iperf* and *tcpdump* are used to analyze how deflection affects the TCP throughput. For better evaluation of deflection routing, the controller ignores all failure notifications and, then, keeps the same route with or without link failures in planned paths.

3.1. Emulated Network Topology with 15 nodes

Consider the Autonomous System AS1 of the network scenario shown in Fig 2 needs to communicate to AS2. The controller by any reason selects as primary route SW10-SW7-SW13-SW29. It may provide link failure protection by using driven deflection forwarding paths as shown in Fig 3.

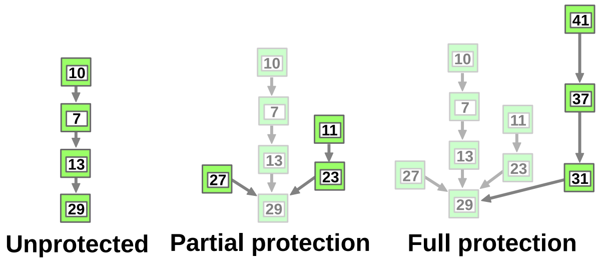


Figure 3. KAR driven deflections as protection mechanism for resilient routing.

The required bit length for each case was computed by using the equation of upper bound value Eq. (9) and it is shown in Table 1.

Time to Failure Recovery: we start to collect the TCP throughput (from AS_1 to AS_3) 30 seconds before the failure, that lasts for 30 seconds. We stop the data measuring another 30 seconds after link repair.

TABLE 1. MAXIMUM BIT LENGTH REQUIRED BY EACH PROTECTION MECHANISM FOR THE 15-NODE NETWORK

Protection mechanism	**Bit length**	**Number of switches in route ID**
Unprotected	15	4
Partial protection	28	7
Full protection	43	10

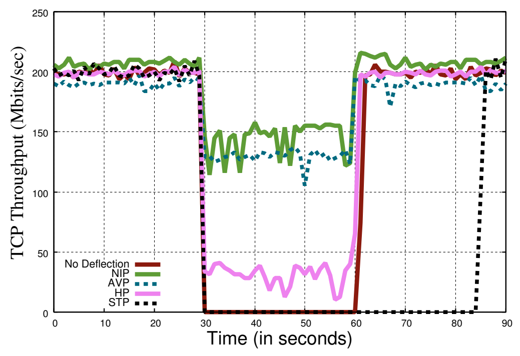


Figure 4. Analysis of TCP throughput for failed link SW7-SW13.

Fig. 4 presents results for TCP throughput considering a failure in SW7-SW23 link for different techniques of packets deflection. As it can be seen, due to deflection routing, the traffic does not stop due to the link failure. Moreover, the NIP deflection has kept the highest throughput compared to all others (HP, AVP and no deflection). The effect of packets disordering had an impact of around 25% (150Mbps of 200Mbps) on TCP throughput, when using NIP deflection technique.

A second bulk of results is presented in Fig. 5. Upon every simulated failure, we run the performance test iperf for 30 times (duration of 5 seconds each) to obtain a confidence interval of 95%. This test evaluates the tradeoffs among route protection (unprotected, partially and fully protected) and deflection techniques (AVP and NIP) and how they impact on TCP throughput.

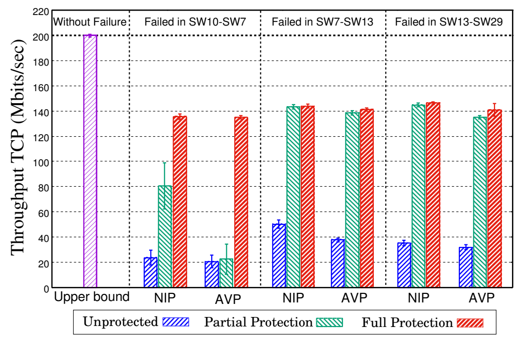


Figure 5. Analysis of TCP throughput varying the failure location: SW10-SW7, SW7-SW13 and SW13-SW29.

A first observation is that full protection based on driven deflected forwarding paths achieves the highest throughput independently of deflection technique and where the failure occurs. Actually for the full protection, the effect of packets disordering had basically an impact of around 30% (140Mbps of 200Mbps) on TCP throughput.

A second observation is that partial protection path had similar resilient routing than full protection path for failures at links (SW7-SW13) and (SW13-SW29). This is explained by the fact that partial protection was enough to enclose the alternative paths to the destination. However, for failure at link (SW10-SW7), there is still $2/3$ of packets that will be sent to switches SW17 or SW37. This explains the difference of around 60 Mbps (basically $1/3$ of packets) for NIP with partial protection (80Mbps) against full protection (140Mbps).

3.2. Emulated National Network Topology with 28-nodes

The goal of this experiment is to evaluate KAR routing resiliency under more heterogeneous and realistic conditions. To this end, we selected the Brazilian National Research and Educational Network (RNP) topology, which is composed by 28 points of presence and 40 links. Fig. 6 shows the KAR representation of this topology with core switches labelled with their local IDs. Different from the previous network topology, we considered that the links rates are proportional to RNP real link rates in this experiment. For evaluation purposes, we selected a route that connects the country from the north *Boa Vista with ID 7* to the international hub *São Paulo with ID 73* and included the links SW17-SW71, SW61-SW67, SW67-SW71 and SW71-SW73 as the driven deflection forwarding paths into the route ID as partial protection, as it can be seen in Fig. 6. The chosen deflection technique was NIP, because it presented the best results in the previous tests.

Fig. 7 presents the TCP throughput analysis with no failure and with variations in failure location: SW7-SW13, SW13-SW41 and SW41-SW73. As this figure shows, the reduction on the TCP throughput is less pronounced for the failure at link SW7-SW13 (decrease of less than 5%) than at link SW13-SW41 (decrease of approximately 40%) and at link SW41-SW73 (decrease of approximately 30%). Indeed, this behaviour can be easily explained: when the link SW7-SW13 fails and the packet is deflected, the only alternative path is to SW11 and, then, to SW17. When the packet reaches SW17, it is already covered by a protection path that leads to SW71 and finally to SW73. Thus, no other deflection is necessary and the failure causes the addition of one more hop without any packet disordering. On the other hand, when the link SW13-SW41 fails, the node SW13 is highly connected and the packet can be deflected to SW29, SW17, SW47, SW37 or SW71 with equal probability ($1/5$). When a node that is part of the selection path is chosen as output port (SW17 or SW71), the packet will be directly driven to the destination with less impact on the TCP throughput. However, when the other three nodes are selected (SW29, SW47 or SW37), the packet will be deflected until it finds a node that is part of the main route or one of the protection paths. The existence of different possible paths after the failure also justifies why failure at link SW13-SW41 causes the highest variance value (about 4%). Moreover, when the link SW41-SW73 fails, the packet can be deflected to SW17 or SW61 with equal probability ($1/2$), both part of the protection paths. If SW17 is chosen, the packet will be directed to SW71 and, then, to SW73. Otherwise, SW61 will send the packet to SW67 that also reaches SW71 and, then, the destination SW73.

Consider now the path presented in Fig. 8 is selected. This new configuration is the worst case scenario to KAR because it is affected by an intrinsic constraint of the KAR routing system: since each switch can only receive one ID, there is only one output port for a pair route ID with respective switch ID, even if there is an alternative path between the two nodes. Consequently, in the event of a failure at this output port, origin node depends on including neighbours into the deflection forwarding paths in order to drive the packets to their destination. For instance, if link SW73-SW107 fails, although there is a second path through SW109 that directly connects SW73 to the destination SW113 (see Fig. 8), this path cannot be set as default path due to the explained constraint of the KAR routing system. Therefore, SW71-SW17 and SW17-SW41 have to be included in the deflection forwarding path to force the packet to travel to the destination. In this case, if a failure happens at link SW73-SW107, there are two possible next hops (SW109 or SW71) with equal probability ($1/2$). If SW109 is chosen, the packet will arrive at the destination, otherwise SW71 (as part of the protection path) return the packet to SW73 through the path SW17-SW41-SW73, which will again decide between SW109 or SW71. Again, SW109 will lead to packet delivery while SW71 will return the packet to SW73. This protection loop will continue until SW109 is probabilistic chosen and the packet is delivered or the failure repaired. So, as a side effect of this constraint and the necessary existence of this protection path, the number of hops increases leading to a reduction on TCP throughput. In the tests, it decreases to 54.8% of the nominal bandwidth.

It is important to outline that, although the proposed schema impacts the TCP throughput, this effect was expected in case of failure and, aside the cited constraint, the scheme shows performance gains and packet loss avoidance that justify the use of KAR to enable network resiliency.

4. Related Work

Our goals are related to two areas of related work: the *source-controlled routing* and *failure reaction within network-controlled routing*.

There has been much work on failure reaction within the network. The most closely related works include MPLS Fast Reroute [12], SafeGuard [13], and OpenFlow 1.3 Fast Failover [14]. The common part among these proposals is the precomputation of alternative paths to each destination for intra-domain routing, so a router can locally switch to the alternative path without waiting for a control-plane

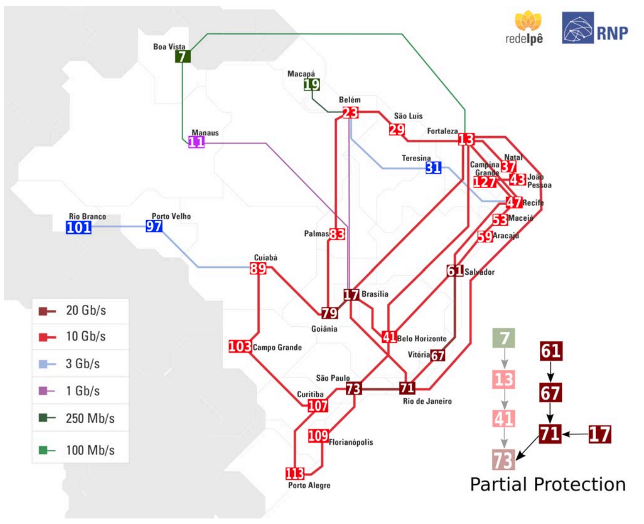


Figure 6. The RNP backbone interconnects Federal Institutions of Education in Brazil.

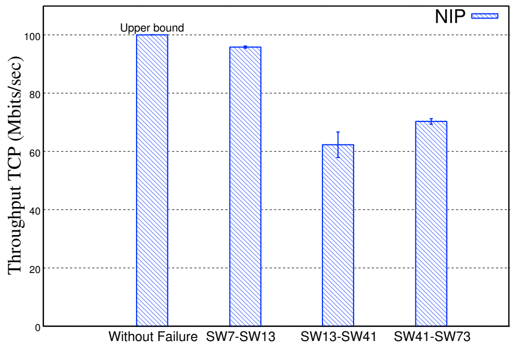


Figure 7. Analysis of TCP throughput with no failure and varying the failure location: SW7-SW13, SW13-SW41 and SW41-SW73.

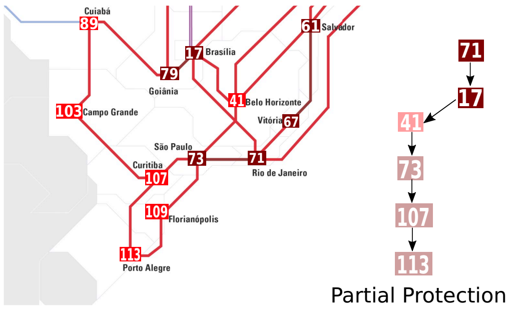


Figure 8. Analysis of a redundant path scenario.

convergence process. However, these approaches require network states stored at the switches tables (statefull) and lack the flexibility of source-controlled routing. In the case of MPLS Fast Reroute [12], it still requires the support of a signaling protocol such as Label Distribution Protocol (LDP) for MPLS enabled switches.

Although source-controlled routing is not in mainstream use of the Internet today, perhaps because source routes do not fit the Internet model in which ISPs set routing policy based on destination addresses, this approach has inspired many innovative proposals for future Internet architectures [2], [3], [4], [5], [6]. Among the main reasons to revisit this approach are [15]: i) the data plane becomes simpler. Core nodes (e.g., switches and routers) perform very simple forwarding operations. ii) Traffic engineering is more flexible, allowing application-optimized path selection at the source. iii) Routing stability is improved (e.g., no transient loops) since the path computation is centralized at the source.

Table 2 shows the main features of the aforementioned works.

TABLE 2. SUMMARY OF THE MAJOR DIFFERENCES BETWEEN OUR PROPOSAL FOR SOURCE ROUTING AND LINK FAILURES FOR INTERNET OF FUTURE.

Work	**Support multiple link failures**	**Source routing**	**State core network**
MPLS Fast Reroute [12]	Yes	Yes	Stateless
SafeGuard [13]	Yes	No	Statefull
OpenFlow Fast Failover [14]	Yes	No	Statefull
Routing Deflections [3]	Yes	Yes	Statefull
Path Splicing [4]	Yes	No	Statefull
Slick Packets [6]	No	Yes	Stateless
KeyFlow [2] and SlickFlow [5]	No	Yes	Stateless
KAR	Yes	Yes	Stateless

In particular, two of the previous approaches, Slick Packets [6] and SlickFlow [5] were proposed to achieve fast data plane failure reaction by embedding alternative routes within the packet headers at the source. The idea is to represent the paths as a sequence of segments that will be used by each switch (or router) to perform the forwarding operation. Also, both [3] and [4] use path label bits set by the source to pseudo-randomly select a next hop at each router or AS. In [3], pseudo-random forwarding can lead to forwarding loops. In [4] routers follow certain rules that ensure loop-free, but reduce path diversity. In contrast to previous works, KAR network core is stateless ([5], [6] and [4] need local states at the routers) and does not depend on the network topology. The second important difference is related to the driven deflection forwarding paths as the resilient routing mechanism for network protection. Rather than define the complete protection path, only small parts of a path can be included or even a unique node can be added to the route ID. This gives more flexibility, keeps the network core fast and simple and does not increase the route ID size.

Finally, although KeyFlow [2] has used RNS approach, KAR routing system has focused on resilient routing that is not taken into account in KeyFlow proposal [2]. In contrast to [2], KAR advances the state of art by dealing with failed links based on driven routing deflections that enables to keep the communication alive even without the controller reaction to a failure.

5. Conclusion and Future Work

This paper proposed KAR (Key-for-Any-Route) that is a novel fast failure reaction scheme to avoid packet loss and improve resiliency and performance for intra-domain routing systems. Our proposal combines a source routing technique based on the *Residue Number System* (RNS) with *Driven Deflections* property in order to enable efficient loop-free routing even in the event of a link failure.

Two different network topologies were emulated for three deflection techniques (HP, AVP, NIP) and for three protection mechanisms (Unprotected, Partial Protection, Full Protection). Results show that KAR efficiently allows deflected packets to automatically reach their destination and that NIP and AVP techniques presented substantial performance improvements when compared with a lower bound classical HP technique. Besides, the emulation analysis has verified that, in addition to avoiding packet loss, KAR using NIP deflection controls the impact of packet disordering on TCP throughput (around 25% in executed tests).

In summary, KAR poses as a fast failure reaction scheme with the following benefits: (i) stateless, which enable high forwarding performance with the use of simple, low-cost switches; and (ii) resilience, as the protection paths enable the packet delivery after a failure through loop-free alternative paths without any reconfiguration on the network nodes.

As future work, we plan to explore the use of multiple paths and improve performance indicators in the case of redundant links and to investigate the application of KAR in the service chaining of virtualized network functions.

Acknowledgment

This work has been partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grant 456143/2014-9 and by the European Union's Horizon 2020 under grant agreement no. 688941 (FUTEBOL project). This research also has received funding from CAPES, FAPES, and the Digital Information and Communication Research and Development Science Center (CTIC, Brazil).

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "Openflow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[2] M. Martinello, M. Ribeiro, R. de Oliveira, and R. de Angelis Vitoi, "Keyflow: a prototype for evolving sdn toward core network fabrics," *Network, IEEE*, vol. 28, no. 2, pp. 12–19, March 2014.

[3] X. Yang and D. Wetherall, "Source selectable path diversity via routing deflections," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 159–170, Aug. 2006. [Online]. Available: http://doi.acm.org/10.1145/1151659.1159933

[4] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path splicing," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 27–38, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402963

[5] R. M. Ramos, M. Martinello, and C. E. Rothenberg, "Slickflow: Resilient source routing in data center networks unlocked by openflow," *IEEE Conference on Local Computer Networks*, pp. 01–08, 2013.

[6] G. T. Nguyen, R. Agarwal, J. Liu, M. Caesar, P. B. Godfrey, and S. Shenker, "Slick packets," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 1, pp. 205–216, Jun. 2011. [Online]. Available: http://doi.acm.org/10.1145/2007116.2007141

[7] H. L. Garner, "The residue number system," *Transactions on Electronic Computers*, pp. 140 – 147, june 1959.

[8] R. Chokshi, K. S. Berezowski, A. Shrivastava, and S. J. Piestrak, "Exploiting residue number system for power-efficient digital signal processing in embedded processors," in *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES '09. New York, NY, USA: ACM, 2009, pp. 19–28. [Online]. Available: http://doi.acm.org/10.1145/1629395.1629401

[9] C. Chang, A. Molahosseini, A. Zarandi, and T. Tay, "Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications," *Circuits and Systems Magazine, IEEE*, vol. 15, no. 4, pp. 26–44, Fourthquarter 2015.

[10] C. Ding, D. Pei, and A. Salomaa, *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1996.

[11] J. Postel, "Internet protocol," RFC 791, http://www.ietf.org/rfc/rfc791.txt, September 1981.

[12] G. Swallow, P. Pan, and A. Atlas, "RSVP-TE fast reroute," RFC 4090, http://www.ietf.org/rfc/rfc4090.txt, May 2005.

[13] A. Li, X. Yang, and D. Wetherall, "Safeguard: safe forwarding during route changes," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 301–312. [Online]. Available: http://doi.acm.org/10.1145/1658939.1658974

[14] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "A Demonstration of Fast Failure Recovery in Software Defined Networking," *TRIDENTCOM 2012*, pp. 411–414, 2012.

[15] T. Lee, C. Pappas, C. Basescu, J. Han, T. Hoefler, and A. Perrig, "Source-based path selection: The data plane perspective," in *The 10th International Conference on Future Internet*, ser. CFI '15. New York, NY, USA: ACM, 2015, pp. 41–45. [Online]. Available: http://doi.acm.org/10.1145/2775088.2775090