Technoscience Journal for Community Development in Africa 2:1 (2021) 69–79
ISSN 2659-0573

Proof!!! Proof!!! Proof!!!

Research Article

A scheme for resilient routing in residue number system based software defined networks

Oke Afeez Adeshina[*,1], Akinbowale Nathaniel Babatunde[2], Oloyede Abdulkarim Ayopo[3], Kazeem Alagbe Gbolagade[2]

[1] Department of Computer Science, College of Natural and Applied Sciences, Summit University, Offa, Nigeria
[2] Department of Computer Science, Faculty of Communication and Information Technology, Kwara State University, Malete, Nigeria
[3] Department of Telecommunication, Faculty of Communication and Information Sciences, University of Ilorin, Ilorin, Nigeria

Abstract: In recent times, the advances in internet technology and the rise of network traffic, owing to relentless creation of rigorous and mission-critical applications, has been giving serious attention to resilient routing in Software Defined Network (SDN). The evolution has made it possible for extensive use of SDN to respond to the current requirements of modern networking. Residue Number System (RNS) been used to minimize latency in routing tables for Openflow protocol. However, majority of network elements remain vulnerable to failures, especially transmission devices and links. It has been shown that proactive approaches to link recovery in RNS based SDN can select alternate path during link failures, however, the challenges of fast failure recovery still exist with large data path labels for both primary and alternate routes. Furthermore, the problem of early backup path failure before primary path still remains a challenge. The paper presents an approach that is intended to improve SDN routing and reduces the computational in RNS based SDN. The proposed scheme utilizes the shortest path re-route algorithm and a reactive mechanism to respond to link failure when it occurs. The proposed scheme which will be implemented as a prototype using the mininet emulator and floodlight controller is intended to effectively route packets especially when there are link failures for RNS based SDN.

Keywords: Residue Number System (RNS); Software Defined Network (SDN); resilience; OpenFlow; traffic engineering; controller

1. Introduction

In the last few decades, technologies for the planning, development and operation of communication networks have largely remained unchanged. With the exponential growth of the modern Internet, network specifications and network traffic have changed significantly, as rigorous technologies are continually evolving for cloud services, IoT (Internet of Things) technologies and the new generation of mobile communication system of 5G, server virtualization and big data (Hakiri et al., 2014; Li et al., 2020; Singh & Jha, 2017). The growing complexity of traditional networking architectures does not cater for the exponential increase of data traffic (Rehman et al., 2019). In addition, technology progresses, connectivity needs and the rise of numerous resources, such as VOIP or streaming video in High Definition where unimaginable when conventional network architecture was planned. These contributed to the issue of improving the network infrastructure control and adapting it to the recurring resource needs of modern applications. Cognitive networks and software defined networks are evolving quickly to satisfy the current networking demands of isolation, virtualization, traffic engineering,

* Corresponding author:
Email: okeafeez@summituniversity.edu.ng

KWASU PRESS AND PUBLISHING

access control and above all programming (Martinello et al., 2014). These types of networks are required since existing networks are outdated due to new and innovative technologies and the limit to network growth is being made available (Malik & Fréin, 2020).

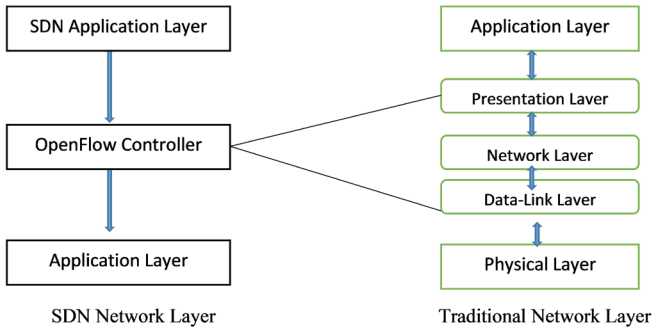


Figure 1: Traditional network layer versus SDN

The data plane and control plane are merged in traditional network architectures, as shown in figure 1, the interconnection of the Traditional layer makes it closed as it layer depends on each other. Additionally, the control plane does not hold any abstraction levels unlike the data plane. (Kreutz et al., 2015). As seen in figure 1, the SDN network layer work independent of each other. The Control Plane consists of configuration protocols such as Multi-Protocol Label Switching (MPLS) and a variety of routing protocols. In Traditional Networks, example of protocols include, Routing Information Protocol (RIP), Enhanced Interior Gateway Routing Protocol (EIGRP), Shortest Path Bridging (SPB) etc. However, this protocol addresses particular problems without basic abstraction. This causes network difficulty in connecting a computer. In addition due to packet flooding, increased time to track errors, estimating alternative paths and upgrading the router table, there is a significant recovery delay in conventional IP networks (Alzahrani & Fotiou, 2020). This makes existing networks very stagnant and cannot be regulated dynamically.

The next generation infrastructure for the internet has implemented the SDN concept to tackle traditional network architecture issues in order to create an optimal architecture that can be managed effectively, adaptably and at low costs (Braun & Menth, 2014; Hakiri et al., 2014; Zhao et al., 2019). The SDN architecture has established a network abstraction layer. As shown in figure 2, the data and control planes are characterized by network services abstraction (Saraswat et al., 2019). A centralized authorization known as the controller in figure 2, provides the communication to instruct network switches to route and monitor the traffic through the network (Braun & Menth, 2014). As the Controller has a global networking view, it would provide ideal network routes either before or on request. (Alzahrani & Fotiou, 2020). With the global view, it thus determines optimally for the disrupted flows when looking for an effective alternative route. Additionally, a well-defined programming interface between the switches and the SDN controller facilitates the distinction between control and data plane as depicted in figure 2. (Braun & Menth, 2014; Kreutz et al., 2015).

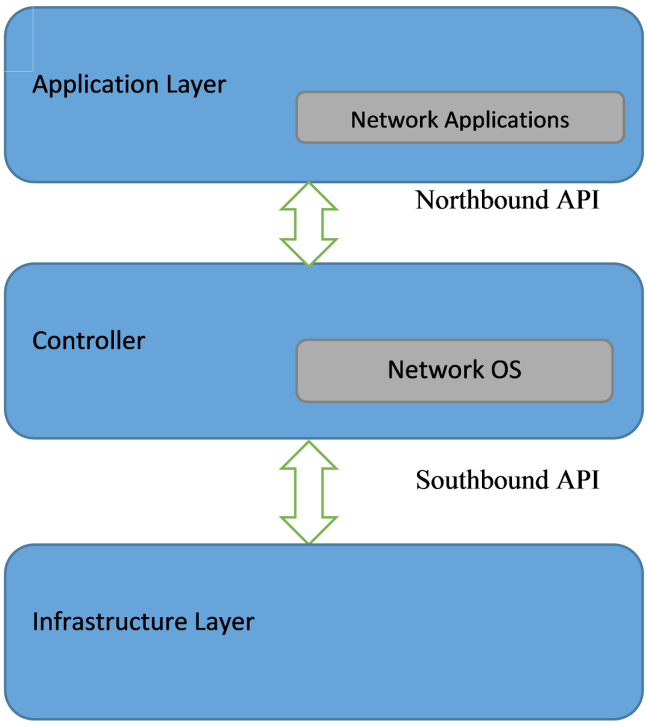


Figure 2: SDN architecture

SDN Controller configures transition flows and tracks how nodes are communicated by the switch (Braun & Menth, 2014). Therefore, it is possible to detach network devices on the switch and control traffic on the OSI layer. This is simpler than the traditional network separation.

OpenFlow is the most common and open standard communication protocol between the SDN controller and OpenFlow switches. OpenFlow was designed in 2008 and currently managed by Open Network Foundation at Stanford University (ONF). The fundamental principle of OpenFlow is that there are forwarding tables and an open API that the OpenFlow controller uses. The transmitting rules to the switch are being used as the controller knows the Network Vision. The opened specification requires separate devices from multiple vendors to be connected to one OpenFlow controller. (Braun & Menth, 2014).

While OpenFlow is a significant step towards opening the control plane, it does not simplify hardware entirely or allow flow state changes. (Martinello et al., 2014). OpenFlow architectures often have scalability problems in core networks in particular since they involve a broad number of active flow-related states (Braun & Menth, 2014).

OpenFlow's limitations led researchers to construct smoother core network components for packet forwarding using the Residue Number Method. (Liberato et al., 2018; Martinello et al., 2014, 2017; Spalla et al., 2015). By means of a combination of small numbers produced by the remaining part of a large number, RNS makes it possible to represent a large number entirely. The large number is a route ID, the small numbers are the output ports of the core switches used for the specific path and the related co-primes are the core network interfaces. Current networks are very dynamic, and therefore, the configuration changes and link failures in a single time are very high (Rehman et al., 2019; Yu et al., 2019). Consequently, many resilient routing approaches in SDN networks based on RNS routing were proposed in this regard. (Gomes et al., 2016; Liberato et al., 2018; Spalla et al., 2015), however, the configuration of backup paths to achieve high network throughput and latency in failure recovery still remains a challenge.

With substitution of table lookups by modulo operations, RNS based SDN schemes were able to mitigate latency problems occurring in OpenFlow Lookup table in SDN (Cercós et al., 2014; Martinello et al., 2014), however, the challenges of fast failure recovery still exists (Muthumanikandan, 2017). Additionally, the Residue Arithmetic based schemes where fault detection mechanism were included are all proactive based approaches (Liberato et al., 2018; Valentim et al., 2019). Here, the use of larger data path labels for both primary and emergency routes and the problem of early backup path failure before the primary path are challenges (Valentim et al., 2019).

Network elements, including forwarding devices and connections, are likely to fail (Malik & Fréin, 2020; Rehman et al., 2019). As a result, network facilities such as routing are damaged. Recovery from failure can be done either proactively, or reactively also referred to as recovery. The alternate path is prepared and reserved for security before failure happens. However, the solution is not pre-planned during restoration and can be automatically determined (on request) if a malfunction happens. Link failure in software defined networks are a continuing source of service interruption. Consequently, many re-routing methods have been suggested after a connection breakdown. This study focuses on the problem of multiple contingency paths for the treatment of single and dual link failure. In the current study, we focus on a reactive approach for fast failure recovery in other to reduce the large data labels and also build multiple alternate emergency path in case of multiple link failures for a RNS based SDN. The shortest path fast re-routing technique is employed in this scheme in order to reduce the loss of data and minimize the recovery time.

The next section of the paper provides an overview of RNS together with the common representations and definitions. Various related work on RNS based Software Defined Network are illustrated in Section III. Section IV discusses resilient routing in Software defined networks with the proposed scheme and implementation in section V and VI. Finally, the paper is concluded in Section VII with a discussion on anticipated results and the direction for further research.

2. Overview of residue number system

2.1. Overview of Residue Number System (RNS)

The RNS is a system of unusual numbers identified with the relatively prime moduli set $\{m_1, m_2, ..., m_n\}$ which is $\gcd(m_i, m_j) = 1$ where $i \neq j$ (Navi et al., 2011). Considering a number X with a modulo m, then X could be expressed by its residue r as (1).

$$r = |X|m \quad (1)$$

Equation (1) can be represented by $X \equiv r \pmod m$.

For instance, the number X=64 with modulo m = 13 can be shown as $r_X = |64|_{|13|} = 12$. Similarly, the number Z=51 with the same modulo could be represented as $r_Z = |51|_{|13|} = 12$. The modulus 13 shows both X and Z as 12. In this case, X and Z are congruent modulo m. Instead of utilizing a single modulus, a set of multiple moduli, $\{m_1, m_2, ..., m_n\}$, is proposed to represent a number. A number can then be represented as one set of residues, $\{r_1, r_2, \ldots, r_n\}_{\{m_1, m_2, \ldots, m_n\}}$ accordingly. For example, assuming there is one set of moduli {13, 16, 19}, number X and Z would be represented in RNS as $\{12, 0, 7\}_{[13,16,19]}$ and $\{\{12, 3, 13\}_{[13,16,19]}$ respectively.

The general RNS calculation model as shown in Figure 3 below includes the conversion steps to the RNS and the positional notation from the RNS is depicted.

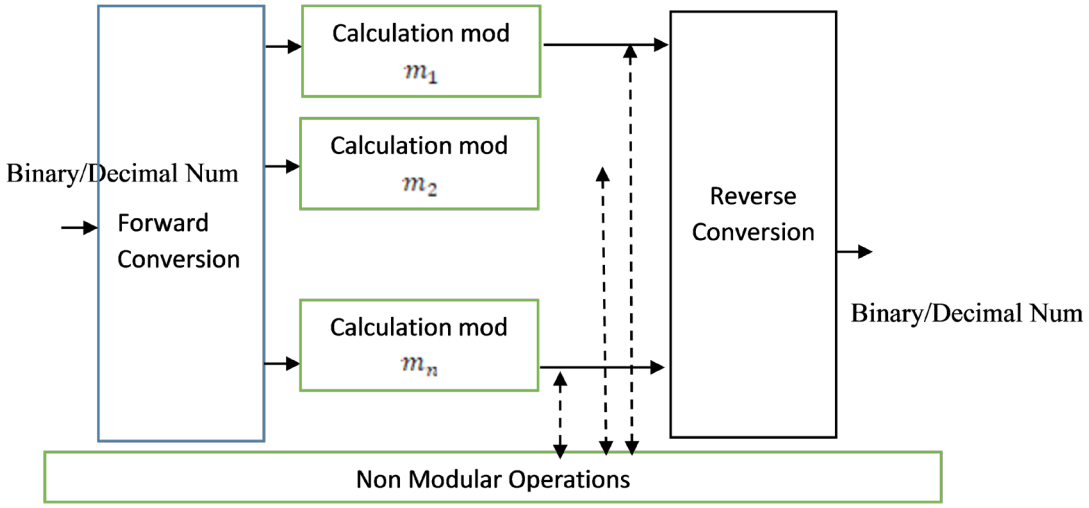


Figure 3: Model of computation in RNS

Additionally, a separate computational structure is used to represent different kinds of non-modular operations (Deryabin et al., 2018). RNS data transfer requires that the Chinese Remaining Theorem (CRT) be used by a forward or opposite converter to transform weighted operands to residual representations (Navi et al., 2011) or the Mixed Radix Conversion (MRC) (Gbolagade & Voicu, 2011; Navi et al., 2011). The forward conversion includes the transition to RNS equivalent of binary or decimal numbers, while the reverse conversion transforms RNS numbers to binary or decimal numbers, and arithmetical operations like addition, subtracting and multiplying can be carried out in parallel by using RNS without carries between residue numbers.

Chinese Remainder Theorem (CRT)

Let the residue representation of x for moduli $\{m_1, m_2, \dots, m_n\}$ be given as $(x_1, x_2, \dots, x_n)$. The Chinese Remainder Theorem makes it possible to determine $|x|_M$, provided the greatest common divisor of any pair of moduli is 1. Using the equation:

$$X = \left| \sum_{i=1}^{n} xi.M_i \left| M_i^{-1} \right|_{mi} \right|_M \quad (2)$$

Where $M_i = \frac{M}{m_i}$ and M_i^{-1} is the multiplicative inverse of M_i with respect to m_i such that $\left| M_i^{-1} * ? \right|_{mi} = 1$.

Such that

$$\left| M_i^{-1} * ? \right|_{mi} = 1$$

$$X = | m_1 | M_1^{-1} |_{m_1} x_1 + m_2 | M_2^{-1} |_{m_2} x_2 + m_3 | M_3^{-1} |_{m_3} x_3 \quad (3)$$

RNS for routing in Software Defined Networks

Let Network domain represent a set of n Switches in a desired path so that $S = S_i | i = 1,2, \dots, n$ where $S_1, S_2, \dots S_n$ need to be pairwise relatively primes. That is the switches represent the moduli set.

Let P be a set of outgoing ports $P = \{p_1, p_2 \dots p_k\}$ which is considered as a residue.

Thus, in order to establish communication between two pair of hosts, the RNS based Controller needs to calculate what is the number (route- ID) for which the modulo operations results will lead the packets to their destination.

Then, there exists a unique integer X such that $0 \leq X < \prod_{i=1}^{n} s_i$ that solves the congruence.

$$\langle X \rangle_{S_1} \equiv p_1$$
$$\langle X \rangle_{S_2} \equiv p_2$$
$$\langle X \rangle_{S_n} \equiv p_k$$

Let $M = \prod_{i=1}^{n} s_i$

Using CRT it is possible to calculate X through its residues:

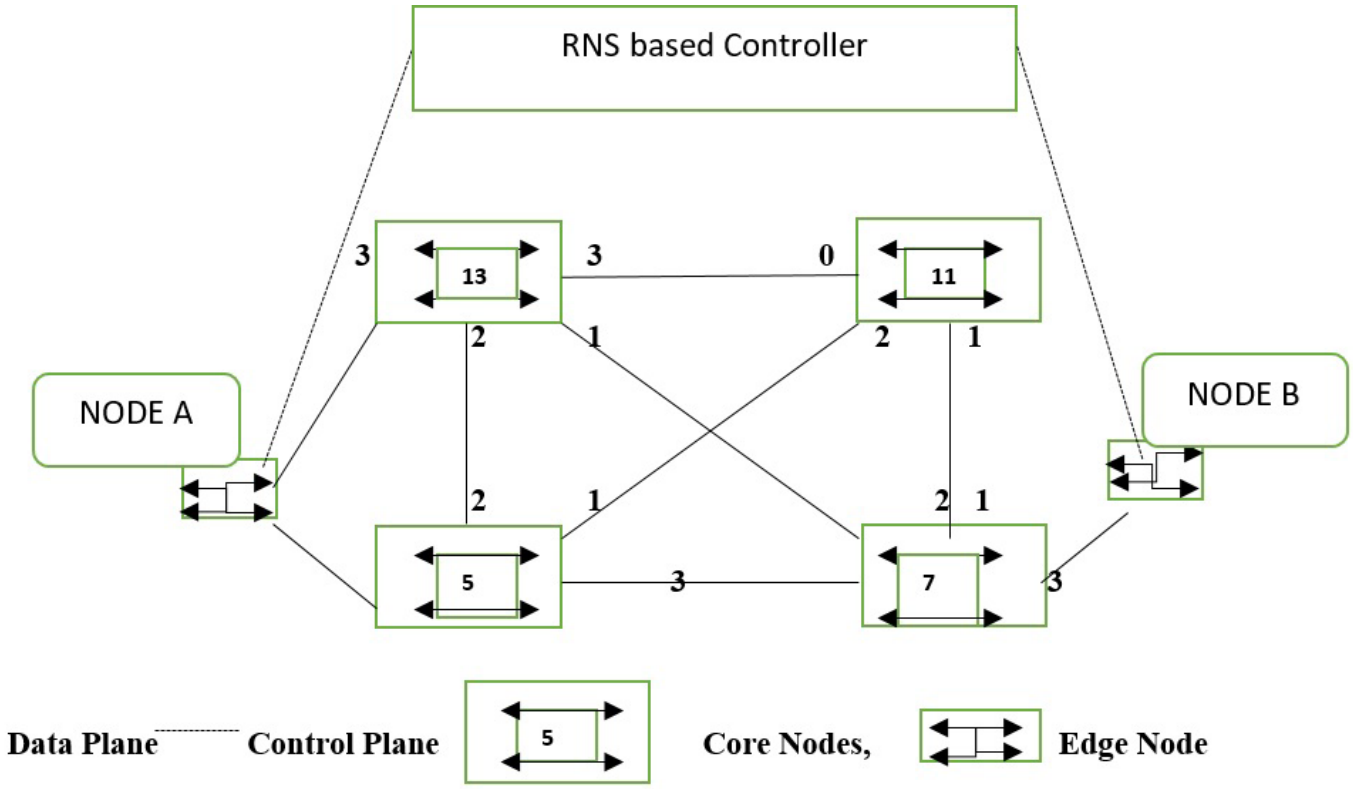


Figure 4: Sample Packet forwarding using Residue Number system based SDN's.

$$X = \left|\sum_{i=1}^{n} pi.M_i\left|M_i^{-1}\right|_{si}\right|_M \quad (4)$$

Where $M_i = \frac{m}{s_i}$ and M_i^{-1} is the multiplicative inverse of M_i with respect to m_i

$$X = \left|s_1\left|M_i^{-1}\right|_{s_1} p_1 + s_2\left|M_i^{-1}\right|_{s_2} p_2 + s_3\left|M_i^{-1}\right|_{s_3} p_3\right| \quad (5)$$

For example, from the diagram in Figure 4, core switches are assigned to co-prime numbers to establish the communication between the pair of hosts, in this example {5, 7, 11, 13}. For instance, it chooses the route to be set through the switches S = $\{s_1, s_2, s_3\}$ = {13, 5, 7} and switches' output ports are P = $\{p_1, p_2, p_3\}$ = {2, 0, 0}. A label (route id) is computed using CRT, for example 210 in the example diagram M =13 · 5 · 7 = 455 and M_1 =35, M_2 =91, M_3 =65.

$L_1 = < M_1^{-1} >_{s_1} = < 35^{-1} >_{13} = 3$

$L_2 = < 91^{-1} >_5 = 1$

$L_3 = < 65^{-1} >_7 = 4$

Using CRT, R = $<L_1 \cdot M_1 \cdot p_1 + L_2 \cdot M_2 \cdot p_2 + L_3 \cdot M_3 \cdot p_3 >$ mod M

R = < 210 +0+0 >mod 455 = 210.

On computing the route-ID, the controller sends the route-ID to the edge switches in example above 210 which then install the respective flow-table rule. Additionally, the ingress edge switch is responsible for embedding route-ID into each packet coming from src host to dst host. Thus, as shown in the Figure above, when a packet enters a core, the modulo of the route ID and the switch is used to determine the output port to forward the packet to. For example when S13 receives a packet with route-ID (R = 210), it forwards the packet to port $< 201 >_{13} = 2$; then, S5 forwards it to port $< 201 >_5 = 0$; after, S7 forwards it to port $< 201 >_7 = 0$, reaching the egress edge switch that removes the route-ID from the packet and delivers it to dst host.

3. Related works

3.1. Residue number system based software defined networks

RNS was used by Wessing et al., (2002) for packet forwarding and to avoid both optical header rewriting of photonic packet switching networks and the need for label distribution protocols. In packet forwarding in

software-defined networks, RNS is often used to replace the OpenFlow table lookup with modulus of the RNS operations that allow explicit recognition route coding at the edges and stateless forwarding techniques in the center of the network (Liberato et al., 2018; Martinello et al., 2014). The use of RNS for packet forwarding has the potential to unravel the efficiency of SDN usage in core networks. The section reviews literatures that have addressed or use RNS to improve SDN's.

Wessing, et al. (2002) applied RNS to avoid both optical header rewriting of photonic packet switching networks and the need for label distribution protocols. The authors implemented a packet transmission using the Chinese Remainder Theorem (CRT) by creating a label based on two arrays. Both of these arrays have all the node-specific keys and all the output information needed for a particular path through the core network. Where the network path is wanted, an array consists of all node-specific keys and the other array consists of the output ports for the specified path. The authors configured the network with each core node and assigned a key, however, the information of the topology of the network and the assigned keys are provided to the edge nodes. The authors computed the label using CRT when the packet enters the network and finally the label is added to the packet that is transmitted to the core network. Their system is able to calculate the output information within each of the core nodes using the modulo operation of RNS. The authors assessed the system's scalability, allowing the software to scale up to 50 core nodes for multi-protocol label switching (MPLS) networks, since only 7 bytes are needed for the label in the header. However, the minor restriction in route length and the requirement for larger number of nodes in core networks still remains a challenge.

Martinello et al. (2014) proposed a scheme to further decouple the control and data planes in the design of SDN. Their approach totally eliminated the use of lookup tables in the forwarding engine using the operations of RNS. The route path ID is computed based on RNS without further interaction with the nodes. Experimental analysis was performed using Mininet emulation environment and OpenFlow 1.0 with a Round Trip Time (RTT) above 50 percent and 30 percent reduction in keeping active flow state in the network. However, the scheme is not fault tolerant as there are no protection paths that can be pre-computed at the controller.

Cercós et al. (2014) performed a detailed and comprehensive data plane power consumption analysis of the OpenFlow switch by breaking it down its design modules The KeyFlow was proposed as an alternative since it eliminates a flow table lookup. Experimental analysis reveals that the overall power consumption is reduced by 53.7%.

Cercos et al. (2015) proposed a new south-bound protocols within SDN with the use of Residue Number System which the SDN architecture in terms of cost and energy efficient forwarding engine. This authors used the Keyflow in (Martinello et al., 2014) approach to simultaneously reduce latency, jitter, and power consumption in core network nodes of SDN. Experimental result reveals that the round-trip time (RTT) can be reduced above 50% compared to the OpenFlow protocol, especially for densely populated flow tables. The author's demonstrated the reduction of jitter by implementing the prototype on a NetFPGA based platform, which reveals that there is a 57.3% reduction in power consumption.

Spalla et al. (2015) proposed a scheme to explore the OpenFlow roles for the design of resilient SDN architectures relying on multiple controllers. The authors implemented their scheme using the Ryu controller exploring the OpenReplica service to ensure consistent state among the distributed controllers and a prototype was tested with the RouterBoards/MikroTik switches and evaluated for latency in failure recovery and switch migration for different workloads.

Gomes et al. (2016) proposed a scheme incorporating deflection guiding mechanism and RNS in 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. The authors used the RNS approach in (Martinello et al., 2014) but improved the Network 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. Experimental analysis using Mininet network emulation environment shows that the scheme was able to avoid packet loss, and using the deflection controls the enabled the packet disordering on TCP throughput to about 25%. However, source routing from the controller takes long time to recover from failures.

Liberato et al. (2018) proposed a new concept of programmable Residues Defined Networks (RDN) based on SDN concept. The authors used a protection mechanism which permits tableless switches in a faster alternative to controller-based restoration of the route to re-route packs directly onto the data plane. For the sake of fault tolerance, an alternate path is pre-computed into the packets that lead the packets to their destination if a key switch detects a fault in their networks. The authors implemented the RDN prototype in Mininet emulated environment while the core switches were implemented in NetFPGA devices to

increase the accuracy on latency measures. Experimental result shows the scheme offers an ultra-fast failure recovery, and no jitter within the RDN core while also achieving low latency in around 2.33μs for 3 hops and 2.93μs for 4 hops with RDN switching time per hop (≈ 0.6μs). However, there is overhead due to the insertion of both primary and emergency route in the packet header, the use of emergency route can be optimized for multiple link failures.

Valentim et al., (2019) proposed a Residue Defined Network Architecture for load balancing by exploiting the elephant flow isolation and strict source routing in core nodes. The authors used the flow classification operations performed on the edge using features of the OpenFlow protocol. The scheme used core switches that forward packets based on the remainder of division between the route identifier and the switch identifier. A prototype was implemented using OpenStack as the cloud manager framework. Edge and core switches were virtualized using a customized version of OpenvSwitch (OvS). Experimental result show that the scheme is able to migrate routes with low data loss rate, without compromising the communication between servers. However, the scheme is not able to detect congestion detection and queue overflow detection.

Lacan & Lochin, (2020) introduced a XOR-based source routing enable fast forwarding and low- latency communications. The scheme utilizes linear encoding to construct the route labels of unicast data and multicast data transfers. In contrast to standard table lookup, they allow quick, computationally efficient routing decisions without any packet alteration along the path. The authors compared their scheme with the scheme proposed by (Liberato, et al., 2018) that uses RNS modular property to compute a label-ID number to identify (following a reverse operation) the output switch port considering a unique router ID. The comparative analysis indicates that the scheme computes the smallest label possible and presents strong scalable properties compared to approaches based on modular arithmetic. However, their scheme does not possess a fault tolerant capability and it involves a lot of complexities.

3.2. Resilient routing in software defined networks

Resilient routing and link failure recovery in software defined networks is becoming more important nowadays because of increasing development of new internet and devices, there has been an extraordinary change in network requirements coupled with an increase in network traffic due to the constant development of rigorous applications and increasing number of users. Hence, there is need for efficient and fast failure recovering during packet routing (Muthumanikandan, 2017; Petale, et al., 2020; Rehman et al., 2019; Yu et al., 2019). Link failure recovery techniques take advantage of central control SDN unique features and flexibility of programmable data planes for real-time applications like video conferencing and voice over IP (VOIP), which can tolerate a delay of 50 ms in case of recovery. The different methods of link failures recovery in SDN can be divided widely into two categories: proactive and reactive (Petale et al., 2020).

3.2.1. Proactive approaches for link recovery

In proactive recovery, alternate backup routes have been preconfigured. The identification of failure is local and the flows from the failed link are passed automatically without contact with the control unit to the alternate direction. If a fails, the flow rules for the backup path are set on the switch already. Therefore the packets from the failed connection are diverted to the alternate route identified by the switch. Proactive Recovery proponents contend that proactive recovery is time efficient, because paths are preset and no alternate paths must be tested by the controller. Therefore, it is held to a minimum every time you consult the controller and select an alternate course. An alternate path, however, must be built for each flow of the failed link, which is impractical, violating the limitations placed on the flow table input by the data plane switches.

3.2.2. Reactive Approaches for Link Failure Recovery in SDN

Reactive failure recovery is primarily reliant on the SDN controller (Lin, et al., 2016; Petale et al., 2020). If the controller senses faults, then by sending regular heartbeat messages the controller searches for a possible route for the failing link. The controller eliminates the old flow inputs and attaches new flow inputs to the SDN switches for the revised route.

While the reactive methods the alternate path can be sought dynamically, control intervention causes a considerable duration of recovery. Because of the overall synchronization between the switches and the control, there is additional time to locate the alternate path and to initiate new flow entries.

4. The proposed scheme and implementation

The proposed system uses a reactive mechanism to respond to link failure. The Controller uses RNS arithmetic for routing instead of the OpenFlow table routing, when a link failure occurs, the switch nearest

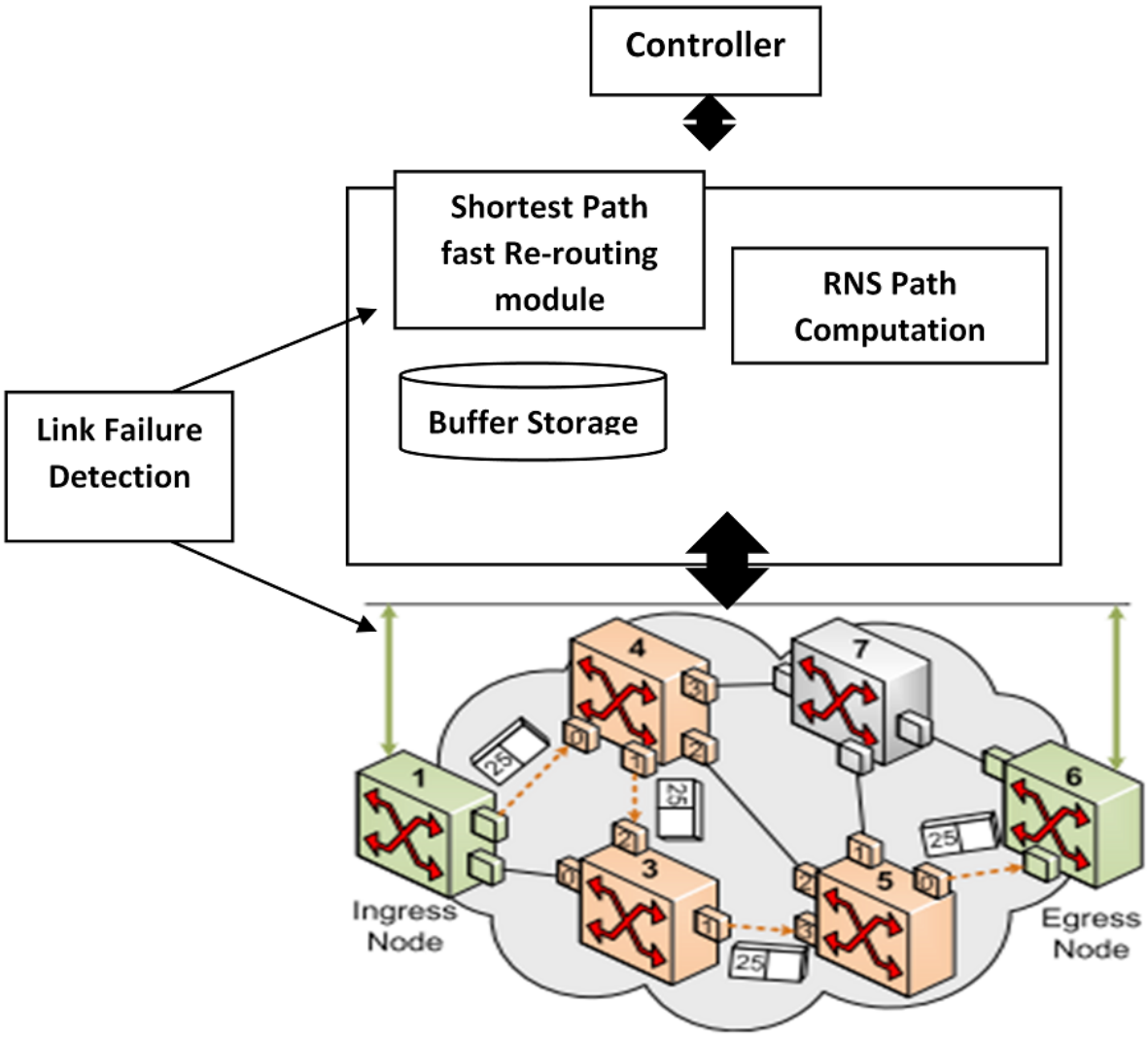


Figure 5: Proposed qrchitecture of RNS based SDN with shortest path fast rerouting

Proposed Resilient re-routing procedure

Start:
For communication between source host to destination host
Step 1: Start
Step 2: Extract Ethernet header
Step 3: if (Ethernet == type of Source Routing)
Extract SR header & get Port
Compute route id using CRT
Set Output Port and emit packet
If (Output is not available)
Calculate Alternate Path using Shortest re-route
Repeat Step 3
Endif
End if
Step 4: Deliver to Output port
Step 5: Stop

to the failure sends a signal to the Controller which then computes the shortest path from the switch to the required destination. The architecture of the proposed scheme, showing the details of the RNS path computations and the shortest path fast re-route is shown in figure 5.

Shortest path fast rerouting

The set of shortest paths as shown in figure 6, are computed reactively on demand thereby memory overhead can be reduced. Multiple backup shortest paths are computed to help in handling single and dual link failures.

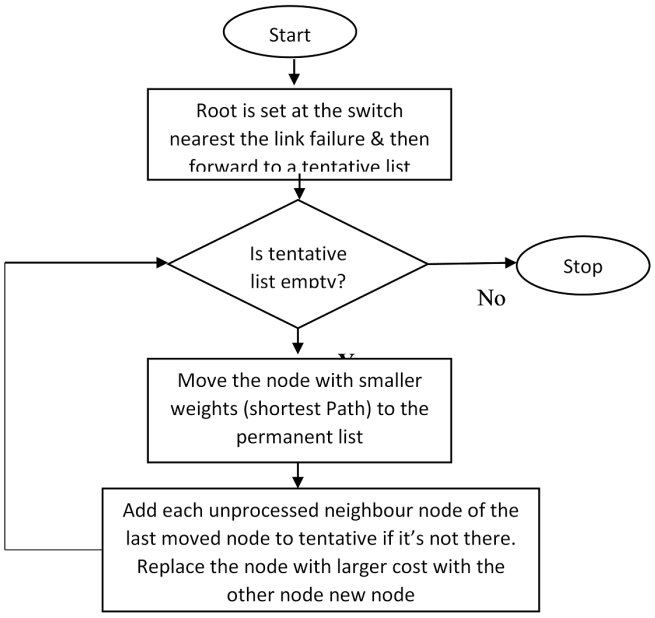


Figure 6: The shortest path algorithm

The embedding of resilient and fast link failure recovery modules in SDN is of crucial importance. Most proposals for resiliency and fault tolerance in RNS based SDNs are mostly proactive with large overheads for emergency route embedding in packet header (Liberato et al., 2018; Martinello et al., 2017). Others have used route deflection which is probabilistic and has transient loops (Gomes et al., 2016).

The step taken for reactive response to link failure using shortest path re-route is explained below:

- The controller calculates the backup path reactively upon notification of a link failure and installs it in the switch.
- The switch generates renewal packets to keep the existing backup paths alive irrespective of the idle time of flow entries
- The new route-ID generated is used to perform modulo operations on the switches to determine the next switch and the destination
- Once the failed link is up, switch notifies the controller to recalculate the best path by sending recovery packets.

5. Results and discussion

In cases of link failure, most approaches have used route restoration which tries to notify the controller to recalculate the route by excluding the faulty links from the available path. Other approaches include route deflection (Gomes et al., 2016) which is probabilistic and has transient loop and the selection of an Emergency route Id (ERI) (Liberato et al., 2018). Our approach uses a reactive approach to link failure. For the implementation of RNS based routing algorithm and shortest path fast re-route, the RNS tables routing was improved on by introducing the shortest path fast re-route during link failure. When a packet arrives at the ingress edge switch, the packet is sent to the Controller which selects the main routes among all pre-calculated paths between the source and destination. Additionally, the Controller installs OpenFlow rules at the ingress and egress switches, when a link failure occurs, the nearest switch to the failure point communicates to the controller which then reactively compute the set of alternate routes using the shortest path fast re-route algorithm.

When link failure occurs, a message is sent to the controller which chooses from the shortest path computed for re-routing. The controller computes the protection path based on the model shown in figure 5, ensuring there are no emergency route necessary thereby reducing packet header information and increasing routing resilience especially for multiple link failures. In situations where there are no link failures, there is no overhead in computing the alternate path in the packet header. The controller only responds when there is a link failure. When the switch receives a packet, it checks if the port is available, then primary route ID is used. However, if the switch port is not available, the controller selects a new primary route ID using the shortest path fast re-route algorithm, henceforth, the new route-ID is used to bypass the failed link. For the subsequent switches along the route, they just keep doing the modulo operation until the packets reach the edge.

6. Conclusion and further work

This research develops a proposal for RNS based SDN with the shortest path re-route for resilient routing. The algorithm would be introduced to SDN with the possibility of reducing the early backup path failure and the overhead caused by larger data path for both

primary and emergency route in Residue arithmetic based SDN. The next intent is to create a prototype of this methodology, test and evaluate with the RNS based proactive approach to resilient routing in SDN. The expectation here is that the proposed approach will successfully and efficiently create a fast reaction to failure, reducing the large data path for the primary and emergency route as well as the early failure of emergency route in the proactive approach.

References

Ali, J., Lee, G., Roh, B., Ryu, D. K., & Park, G. (2020). *Software-Defined Networking Approaches for Link Failure Recovery : A Survey*.

Alzahrani, B., & Fotiou, N. (2020). Enhancing Internet of Things Security using Software-Defined Networking. *Journal of Systems Architecture*, *110*(January), 101779. https://doi.org/10.1016/j.sysarc.2020.101779

Aremu, I. A., & Gbolagade, K. A. (2017). *An overview of Residue Number System*. *6*(10), 1618–1623.

Braun, W., & Menth, M. (2014). Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. *Future Internet*, *6*(2), 302–336. https://doi.org/10.3390/fi6020302

Cercos, S., Ramon, M., Ewald, A. C., Moisés, R. N., Manolova, A., Monroy, T., & Salda, S. (2015). *networking Design of a stateless low-latency router architecture for green software-defined networking*. https://doi.org/10.1117/12.2077560

Cercós, S. S., Oliveira, R. E., Vitoi, R., Martinello, M., Ribeiro, M. R. N., Fagertun, A. M., & Monroy, I. T. (2014). Tackling openflow power hog in core networks with keyflow. *Electronics Letters*, *50*(24), 1847–1849. https://doi.org/10.1049/el.2014.2346

Deryabin, M., Chervyakov, N., & Tchernykh, A. (2018). *High Performance Parallel Computing in Residue Number System High Performance Parallel Computing in Residue Number System*. *9*(February), 62–67.

Gbolagade, K. A., & Cotofana, S. D. (2009). An O(n) residue number system to mixed radix conversion technique. *Proceedings - IEEE International Symposium on Circuits and Systems*, *1*(1), 521–524. https://doi.org/10.1109/ISCAS.2009.5117800

Gbolagade, K. A., & Voicu, G. R. (2011). An Efficient FPGA Design of Residue-to-Binary Converter for the Moduli Set {2n+1,2n,2n-1}. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *19*(8), 1500–1503. https://doi.org/doi: 10.1109/TVLSI.2010.2050608

Gomes, R. R., Liberato, A. B., Dominicini, C. K., Ribeiro, M. R. N., & Martinello, M. (2016). KAR: Key-for-Any-Route, a Resilient Routing System. *Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-W 2016*, 120–127. https://doi.org/10.1109/DSN-W.2016.11

Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D. C., & Gayraud, T. (2014). Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks*, *75*(PartA), 453–471. https://doi.org/10.1016/j.comnet.2014.10.015

Kreutz, D., Ramos, F. M. V., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14–76. https://doi.org/10.1109/JPROC.2014.2371999

Lacan, J., & Lochin, E. (2020). XOR-based Source Routing. *IEEE International Conference on High Performance Switching and Routing, HPSR*, *2020-May*(February). https://doi.org/10.1109/HPSR48589.2020.9098991

Li, Y., Su, X., Ding, A. Y., Lindgren, A., Liu, X., Prehofer, C., Riekki, J., Rahmani, R., Tarkoma, S., & Hui, P. (2020). Enhancing the internet of things with knowledge-driven software-defined networking technology: Future perspectives. *Sensors (Switzerland)*, *20*(12), 1–20. https://doi.org/10.3390/s20123459

Liberato, A., Martinello, M., Gomes, R. L., Beldachi, A. F., Salas, E., Villaca, R., Ribeiro, M. R. N., Kanellos, G., Nejabati, R., Gorodnik, A., & Simeonidou, D. (2018). RDNA: Residue-Defined Networking Architecture Enabling Ultra-Reliable Low-Latency Datacenters. *IEEE Transactions on Network and Service Management*, *15*(4), 1473–1487. https://doi.org/10.1109/TNSM.2018.2876845

Liberato, A., Martinello, M., Gomes, R. L., Beldachi, A. F., Salas, E., Villaca, R., Ribeiro, M. R. N., Kondepu, K., Kanellos, G., Nejabati, R., Member, S., Gorodnik, A., & Simeonidou, D. (2018). *RDNA : Residue-Defined Networking Architecture*. *15*(4), 1473–1487.

Lin, Y., Teng, H., Hsu, C., Liao, C., & Lai, Y. (2016). *Fast Failover and Switchover for Link Failures and Congestion in Software Defined Networks*.

Malik, A., & Fréin, R. De. (2020). *applied sciences Rapid Restoration Techniques for Software-Defined Networks*. https://doi.org/10.3390/app10103411

Martinello, M., Liberato, A. B., Beldachi, A. F., Kondepu, K., Gomes, R. L., Villaca, R., Ribeiro, M. R. N., Yan, Y., Hugues-Salas, E., & Simeonidou, Di. (2017). Programmable residues defined networks for edge data centres. *2017 13th International Conference on Network and Service Management, CNSM 2017*, *2018-Janua*, 1–9. https://doi.org/10.23919/CNSM.2017.8255987

Martinello, M., Ribeiro, M. R. N., Oliveira, R. E. Z. De, & Vitoi, D. A. (2014). *KeyFlow: A Prototype for Evolving SDN Toward Core Network Fabrics*. *April*, 12–19.

Molahosseini, A. S. (2012). *Research Challenges in Next-Generation Residue Number System Architectures*. *Iccse*, 1658–1661.

Muthumanikandan, V. (2017). Link Failure Recovery Using Shortest Path Fast Rerouting Technique in SDN. *Wireless Personal Communications*. https://doi.org/10.1007/s11277-017-4618-0

Navi, K., Molahosseini, A. S., & Esmaeildoust, M. (2011). How to Teach Residue Number System to Computer Scientists and Engineers. *In IEEE Transactions on Education, 54*(1), 156–163. https://doi.org/doi: 10.1109/TE.2010.2048329.

Omondi, A., & Premkumar, B. (2007). *Residue Number Systems: Theory and Implementation*. Imperial College Press.

Petale, S., Member, G. S., & Thangaraj, J. (2020). *Link Failure Recovery Mechanism in Software Defined Networks*. *8716*(c), 1–8. https://doi.org/10.1109/JSAC.2020.2986668

Rehman, A. U., Aguiar, R. L., & Barraca, J. P. (2019). Fault-tolerance in the scope of Software-Defined Networking (SDN). *IEEE Access*, *7*, 124474–124490. https://doi.org/10.1109/ACCESS.2019.2939115

Saraswat, S., Mishra, R., & Gupta, A. (2019). *Challenges and Solutions in Software Defined Networking : A Survey Challenges and Solutions in Software Defined Networking : A Survey*. *May*. https://doi.org/10.1016/j.jnca.2019.04.020

Singh, N. (2016). *An overview of Residue Number System*. *August*.

Singh, S., & Jha, R. K. (2017). A Survey on Software Defined Networking: Architecture for Next Generation Network. *Journal of Network and Systems Management*, *25*(2), 321–374. https://doi.org/10.1007/s10922-016-9393-9

Spalla, E. S., Mafioletti, D. R., Liberato, A. B., Rothenberg, C., Camargos, L., & Martinello, M. (2015). *Resilient Strategies to SDN : An Approach Focused on Actively Replicated Controllers*, 246–259. https://doi.org/10.1109/SBRC.2015.37

Valentim, R. V., Villaca, R. S., Ribeiro, M. R. N., Martinello, M., Dominicini, C. K., & Mafioletti, D. R. (2019). *RDNA Balance: Load Balancing by Isolation of Elephant Flows using Strict Source Routing*. 1–3.

Wessing, H., Christiansen, H., Fjelde, T., & Dittmann, L. (2002). Novel scheme for packet forwarding without header modifications in optical networks. *Journal of Lightwave Technology*, *20*(8), 1277–1283. https://doi.org/10.1109/JLT.2002.800268

Yu, Y., Li, X., Leng, X., Song, L., Bu, K., Chen, Y., Yang, J., Zhang, L., Cheng, K., & Xiao, X. (2019). Fault management in software-defined networking: A survey. *IEEE Communications Surveys and Tutorials*, *21*(1), 349–392. https://doi.org/10.1109/COMST.2018.2868922

Zhao, Y., Li, Y., Zhang, X., Geng, G., Zhang, W., & Sun, Y. (2019). A Survey of Networking Applications Applying the Software Defined Networking Concept Based on Machine Learning. *IEEE Access*, *7*, 95397–95417. https://doi.org/10.1109/ACCESS.2019.2928564