A Recovery Prototype based on Network Coding and Software Defined Networks

Víctor Pilco, Eng.[1], Iván Bernal, Ph.D.[1], and David Mejía, M.Sc.[1]
[1] Escuela Politécnica Nacional, Ecuador, {victor.pilco, ivan.bernal, david.mejia}@epn.edu.ec

***Abstract*– This document presents a prototype that uses Network Coding as a tool for the recovery of lost packets in a butterfly network which is implemented using Software Defined Networking (SDN). The system is based on Ryu as the SDN controller; modules were developed for setting up the rules of the switches that structure the butterfly network, one without network coding and another for using it. Two complementary applications were developed, one for performing the tasks associated to network coding that requires generating new packets and the other for performing the necessary tasks to recover lost packets. Finally, the results of several tests performed using the prototype are presented and discussed and conclusions are drawn.**

***Keywords*-- Network Coding; Computer Networks; IP Networks; Software Defined Networking; Software Systems.**

I. Introduction

At all times, thousands of people require access to the Internet, whether for business, education, or entertainment purposes. However, as the network grows, the need to optimize its operation becomes a priority.

The standard protocols used in current networks allow the interoperability of devices. However, within a specific organization, many of these protocols are not required, and only a few of them are needed for certain tasks. Therefore, there exists the need to provide network administrators with a way to customize their networks in order to improve their performance and optimize the handling of data [1].

Within this context, Software Defined Networks (SDN) [2] present an option to create customized solutions for better control and management of the data circulating through the network due to its ease of implementation, capacity of improvement and adaptability to the present circumstances within the needs of the network. Examples of applications developed for SDN are described in [3-6].

SDN is a networking architecture that decouples the control and switching planes [7]. The switches or white boxes are required to be extremely efficient at their task of switching and must reduce their intelligence to a minimum.

The intelligence of the control plane is derived to a controller (Fig. 1) that executes software modules that define the functionality of the switches and generate rules that must be installed on them. Several controllers are available as shown in [8]. These modules are written in high level programming languages such as C, Java, Python, etc. The controller communicates with the switches by means of a protocol (e.g. Openflow); in the switch side, this protocol allows manipulating the flow table of the switch.

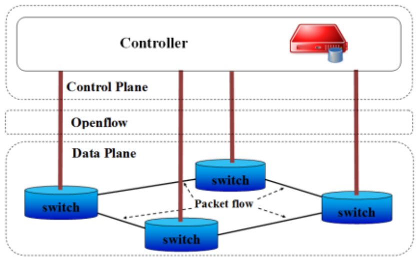

Fig. 1 SDN architecture

Network Coding [9] is defined as the technique for performing "encoding on a node of a network", where the term "encoding" is left open for any type of input and output data handling on the node. That is, intermediate nodes in the network not only forward the data, but also process it for specific tasks [10].

Network Coding has been used with Software Defined Networks. For example in [11] authors propose an optimized routing algorithm to improve network performance by optimizing the network topology using network coding. In [12] a cache management framework based on SDN where a controller is responsible for determining the optimal caching strategy and content routing via linear network coding is described. In [13] it is show how network coding improves latency and reduces packet re-transmission. In addition [14] presents a SDN architecture exploiting Network Coding as a Service.

One of the problems that frequently occur in IP networks is the loss of packets due to errors in the links that interconnect a set of nodes. In this case, the traditional approach to solve this problem is based on the use of protocols with resending mechanisms from the terminals that generated the information [15].

The use of Network Coding as a tool to design and implement a data recovery mechanism within a butterfly network implemented using Software Defined Networks is proposed as an alternative to the traditional solution of lost packets.

The remaining of the paper is organized as follows: Section II explains the use of network coding as a recovery mechanism, Section III presents the implemented system consisting of the butterfly network implemented with software defined network and the network coding application developed to recover packets, Section IV describes the results obtained, and finally, conclusions and future work are presented in Section V.

Digital Object Identifier (DOI): http://dx.doi.org/10.18687/LACCEI2018.1.1.242
ISBN: 978-0-9993443-1-6
ISSN: 2414-6390

II. Recovery of Lost Packets using Network Coding

Network Coding is thus intended to give a level of data processing to network nodes in order to improve the operation of the net. For understanding the operation of Network Coding for recovery of packages the scenario described below is proposed, which is shown in Fig. 2: a) a butterfly network that has two sources and two destinations: Server1 communicates with Client1 and Server2 communicates with Client2. Six switches named as s1, s2, s3, s4, s5 and s6 are used to send data; b) the links between s1 and s3, and, s2 and s4 have losses, which means that they may lose packets with a rate greater than zero (error rate > 0); c) the rest of the links have no losses, i.e. their error rate is equal to zero (error rate = 0); and, d) the link between s5 and s6 is slower than the other links, i.e. it has greater delays in packet transmission than the other links (delay (1) <delay (2)).

The encoding process should be performed in s5, which takes the incoming packets from nodes s1 and s2, and generates a new packet p3 (coded packet) through the binary sum (XOR, exclusive or) of p1 and p2. The packet p3 is forwarded to node s6. Switches s3 and s4 receive packets from nodes s1 and s2, respectively, and forward them to Client1 and Client2 respectively, as well as to s6. Switch s6 receives a copy of the packets that arrive to s3 and s4 originated in s1 or s2, as well as the packet p3 sent by s5. This process of data exchange in the network is shown in Fig. 3.

Due to the speed difference of the links, p3 will arrive at s6 after p1 and p2 have arrived (2 * delay (1) < delay (2)). Therefore, switch s6 can take one of the following actions:

a) If packet p3 has been received, and both p1 and p2 have been received, then p3 is discarded, since packet recovery is not necessary, this is shown in Fig. 4.

b) If packet p3 is received, but only one of the packets is received, either p1 or p2, then the missing packet is reconstructed using p3 and forwarded to the corresponding node to be delivered to its destination, the recovery of p1 is shown in Fig. 5.

c) If packet p3 is received, and no other packet is received, p3 is discarded and no other action is taken. In this case, packet retrieval must be done using recovery mechanisms of superior layers at the client.

This scenario has the following advantages: a) a certain level of capacity is provided to the network to recover lost packets without the need for terminals or sources to be aware of this loss; b) automatic packet recovery at this level allows to decide if it is more convenient that hosts use protocols that do not employ resending, such as UDP; c) in the case of TCP, the number of resends decreases significantly, due to a reduction in the probability of loss.

However, there are also some possible disadvantages: a) the processing level at nodes s5 and s6 is increased, which could increase the total network delay; b) This packet recovery mechanism does not help when both packets are lost, and the coded packet generated does not have any purpose.

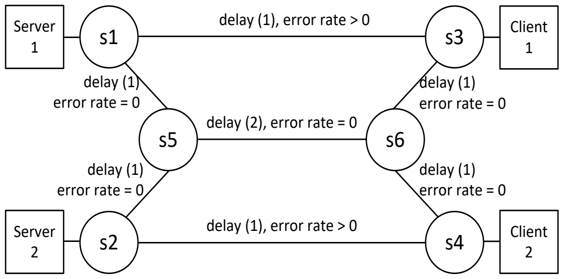

Fig. 2. Basic scenario for analysis

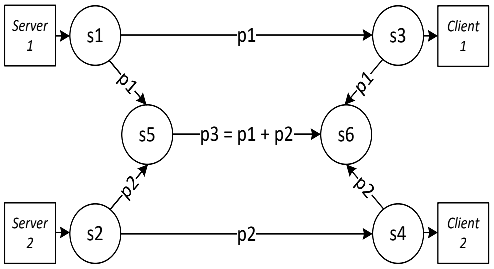

Fig. 3. Packet Flow

III. Implemented Prototype

The system was implemented using seven virtual machines that were configured inside a physical machine, six VM are hosts and one is used for instantiating six virtual switches for the overall structuring of a virtual network environment using Mininet [16]. The virtual machines were configured using VMWare Workstation 12 and Ubuntu Linux as the operating system.

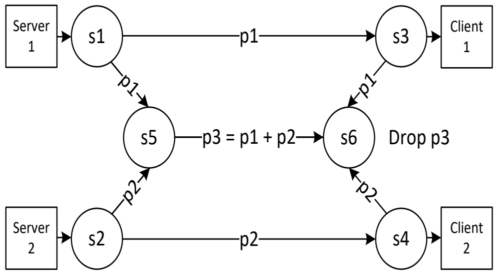

Fig. 4. Packets p1 and p2 arrive to s6 without any losses

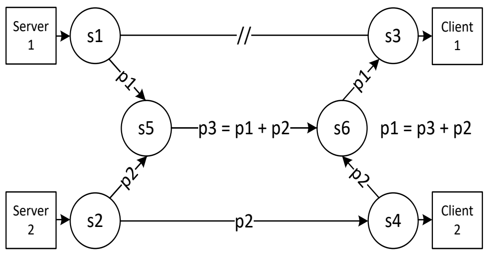

Fig. 5. Packet recovery of p1 at s6

VMs were assigned names according to each of the functions performed within the implemented scenario, as follows: NC Master, NC Host1, NC Host2, NC Host3, NC Host4, NC Switch5 and NC Switch6, whose functions are detailed below (see Fig. 6 and Fig. 7). NC Master is the central virtual machine of the network environment; it executes both the virtual network in Mininet and the Ryu [17] controller.

To generate the virtual network, a script made in Python was used. This script allows stablishing loss rate and delays on each link via a configuration file; also, it defines six virtual switches called s1, s2, s3, s4, s5 and s6 and interconnects them according to a butterfly topology. On s1, in port one is connected NC Host1, port 2 is used to interconnect the port 1 of s2, and port 3 is used to interconnect the port1 of s5. On s2, in port one is connected NC Host2, port 2 is used to interconnect the port 2 of s4, and port 3 is used to interconnect the port 2 of s5. Same configuration is used on s3 and s4. While port 3 of s5 is connected with port 3 of s6, and port 4 of s5 is used to connect NC Swith5, and port 4 of s6 is used to connect NC Swtich6.

Each of the virtual switches on the network is connected to a Ryu controller, housed within the NC master, via the local network interface. Two modules described next will generate rules and send them to Ryu controller, and the Ryu Controller will send those rules to the switches using the protocol OpenFlow [18], the rules specify how the traffic will flow on the network.

Two scenarios also were stablished: the first one without using network coding, while the second one uses network coding. In order to stablish each scenario, two modules for Ryu were developed. These modules generate rules that control network devices.

The first module generates rules that allows traffic from NC Host1 to be sent to NC Host3 using the link between s1 and s3, and from NC Host2 to NC Host4 using the link between s2 and s4. In this case, all the links associated to s5 and s6 are not used; neither NC Switch5 nor NC Switch6 are used. Therefore, on this first scenario, network coding will not be used (Fig. 6). The information of the rules are outlined on Table I, where mac_NC_H1, mac_NC_H2, mac_NC_H3 and mac_NC_H4 corresponds to MAC address of NC Host1, NC Host2, NC Host3 and NC Host4 respectively.

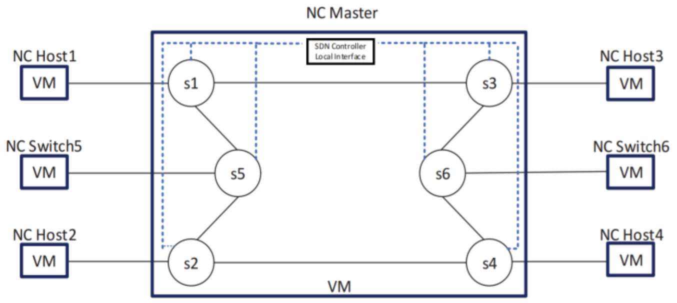

Fig. 6. Network environment implemented without network coding

TABLE I.
RULES FOR FIRST SCENARIO

switch	Input port	src. MAC	dst. MAC	Output port
s1	1	mac_NC_H1	mac_NC_H3	2
	2	mac_NC_H3	mac_NC_H1	1
s2	1	mac_NC_H2	mac_NC_H4	2
	2	mac_NC_H4	mac_NC_H2	1
s3	1	mac_NC_H3	mac_NC_H1	2
	2	mac_NC_H1	mac_NC_H3	1
s4	1	mac_NC_H4	mac_NC_H2	2
	2	mac_NC_H2	mac_NC_H4	1

The second module allows generating rules that allow traffic to flow from NC Host1 to NC Host3 using two routes: s1 - s3 and s1 - s5; and from NC Host2 to NC Host4 using two routes: s2 - s4 and s2 - s5. In this case, all the links are used, and the module generates rules to send the generated coded packet from s5 to s6, structuring a SDN that uses network coding (Fig. 7).

The rules required on the second scenario are show in Table II. It can be seen that if a packet is received from NC Host1, this packet has to be forward to s3 and to s5, using the correct ports in the switch; the same applies to packets received from NC Host2, but they are forwarded to s2 and s4 instead. Also, this module, for those packets that arrive to s5 or s6 and that need to be send to NC Switch5 or NC Switch6, the rule establish to change the destination MAC address (represented as mac_NC_S5 or mac_NCS6) in order to avoid been rejected by NC Switch5 or NC Switch6.

The remaining virtual machines (NC Host1, NC Host2, NC Host3, NC Host4, NC Switch5 and NC Switch6) are connected to the NC Master. The information about IP Address and MAC Address of each virtual machine is presented in Fig. 8.

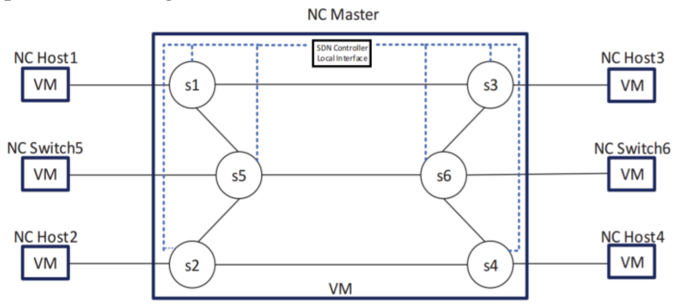

Fig. 7. Network environment implemented with network coding

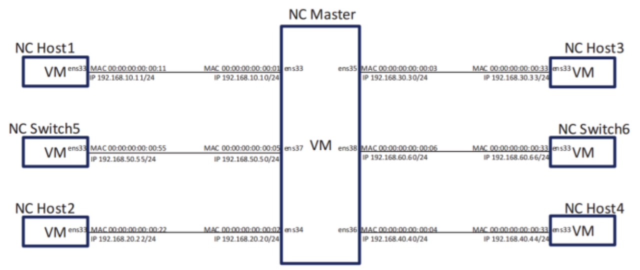

Fig. 8. Network configuration

TABLE II.
RULES FOR SECOND SCENARIO

Switch	Input port	src MAC	dst MAC	new dst MAC	Output port
s1	1	mac_NC_H1	mac_NC_H3	-	3
	2	mac_NC_H3	mac_NC_H1	-	3
	3	mac_NC_H1	mac_NC_H3	-	2
	3	mac_NC_H3	mac_NC_H1	-	1
s2	1	mac_NC_H2	mac_NC_H4	-	3
	2	mac_NC_H4	mac_NC_H2	-	3
	3	mac_NC_H2	mac_NC_H4	-	2
	3	mac_NC_H4	mac_NC_H2	-	1
s3	1	mac_NC_H3	mac_NC_H1	-	3
	2	mac_NC_H1	mac_NC_H3	-	3
	3	mac_NC_H3	mac_NC_H1	-	2
	3	mac_NC_H1	mac_NC_H3	-	1
s4	1	mac_NC_H4	mac_NC_H2	-	3
	2	mac_NC_H2	mac_NC_H4	-	3
	3	mac_NC_H4	mac_NC_H2	-	2
	3	mac_NC_H2	mac_NC_H4	-	1
s5	1	mac_NC_H1	mac_NC_H3	mac_NC_S5	4
	2	mac_NC_H2	mac_NC_H4	mac_NC_S5	4
	1	mac_NC_H3	mac_NC_H1	mac_NC_S5	4
	2	mac_NC_H4	mac_NC_H2	mac_NC_S5	4
	4	mac_NC_H1	mac_NC_H3	N/A	1
	4	mac_NC_H2	mac_NC_H4	N/A	2
	4	mac_NC_H3	mac_NC_H1	N/A	1
	4	mac_NC_H4	mac_NC_H2	N/A	2
s6	1	mac_NC_H1	mac_NC_H3	mac_NC_S6	4
	2	mac_NC_H2	mac_NC_H4	mac_NC_S6	4
	1	mac_NC_H3	mac_NC_H1	mac_NC_S6	4
	2	mac_NC_H4	mac_NC_H2	mac_NC_S6	4
	4	mac_NC_H1	mac_NC_H3	N/A	1
	4	mac_NC_H2	mac_NC_H4	N/A	2
	4	mac_NC_H3	mac_NC_H1	N/A	1
	4	mac_NC_H4	mac_NC_H2	N/A	2

Additionally, two applications for performing network coding tasks were developed and were written using Python. These applications are required given that neither OpenFlow nor the switches can perform the network coding tasks for themselves.

NC Host1, NC Host2, NC Host3 and NC Host4 just function as terminal equipment, as either a client or a server, and only send and receive data frames. NC Switch5 and NC Switch6 are hosts that execute one of the applications that perform the network coding process through a binary sum of the frames sent by the terminals. Besides, these applications are responsible for receiving frames from each terminal (NC Host1, NC Host2, NC Host3 and NC Host4), and perform the binary addition for the network coding tasks, both generate new frames or coded packets as a result of the network coding process, and recover lost frames if needed.

IV. RESULTS

The tests consist of sending ICMP packets on the network. In order to synchronize some tasks and send data at the same time from both terminals (NC Host1 and NC Host2) the Linux *at* command was used. Using Wireshark the frames sent and received at each terminal are captured, and with this information, the number of frames that are lost considering several percentages of losses in the links between s1 and s3, and between s2 and s4 are calculated. The percentage of losses are calculated for both scenarios, the one without using network coding, and the one using network coding. Based on the difference between the percentages of losses obtained for both cases, the recovery percentage is also estimated.

It is not the intend at this stage of the paper to analyse the time difference or the delays introduced by the use of network coding, nor to analyse the level of redundancy, nor to calculate the number of additional frames generated as result of the binary sum of frames sent by each terminal.

By using network coding, a clear decrease in the percentage of losses is observed from the results. Table III presents a summary of the average results obtained without and with network coding under different percentages of pre-established losses.

In Fig. 9, it is noticed that when applying network coding to the system, the percentage of losses decreases considerably, especially in values below 80%. This is because, without network coding, each link has an independent probability of losing frames, which is pre-set in the configuration file

However, when applying network coding the probability of loss is reduced, since there is the possibility of retrieving the information through the new coded frames that travel through the network. There is only loss of information if both links lose their frames at the same time, i.e. the probability that a frame does not reach its destination is no longer the probability of loss of each link, and it becomes the probability that both links lose packets concurrently. This probability is calculated as the probability of simultaneous occurrence of two independent events, since the link between s1 and s3 and the link between s2 and s4 do not depend on each other for their operation they are considered independent.

Therefore, the probability of loss in the butterfly network with network coding is calculated as the product of the probability of loss between the s1 - s3 link and the s2 - s4 link.

The results obtained in the tests prove that the percentages of obtained losses are close to the percentages of expected theoretical losses for a network with network coding where the probability of loss of the network is equal to the probability of simultaneous losses in the two links of the butterfly network.

It is verified that the percentage of losses using network coding is kept less than 50% provided that the percentages of losses in each of the links does not exceed 70%, once this value is exceeded the losses of the network approach to those values obtained without network coding.

It can be seen how the values obtained in the performed tests follow the trend of theoretical curves as shown in Fig. 9 for tests without network coding and for tests with network coding.

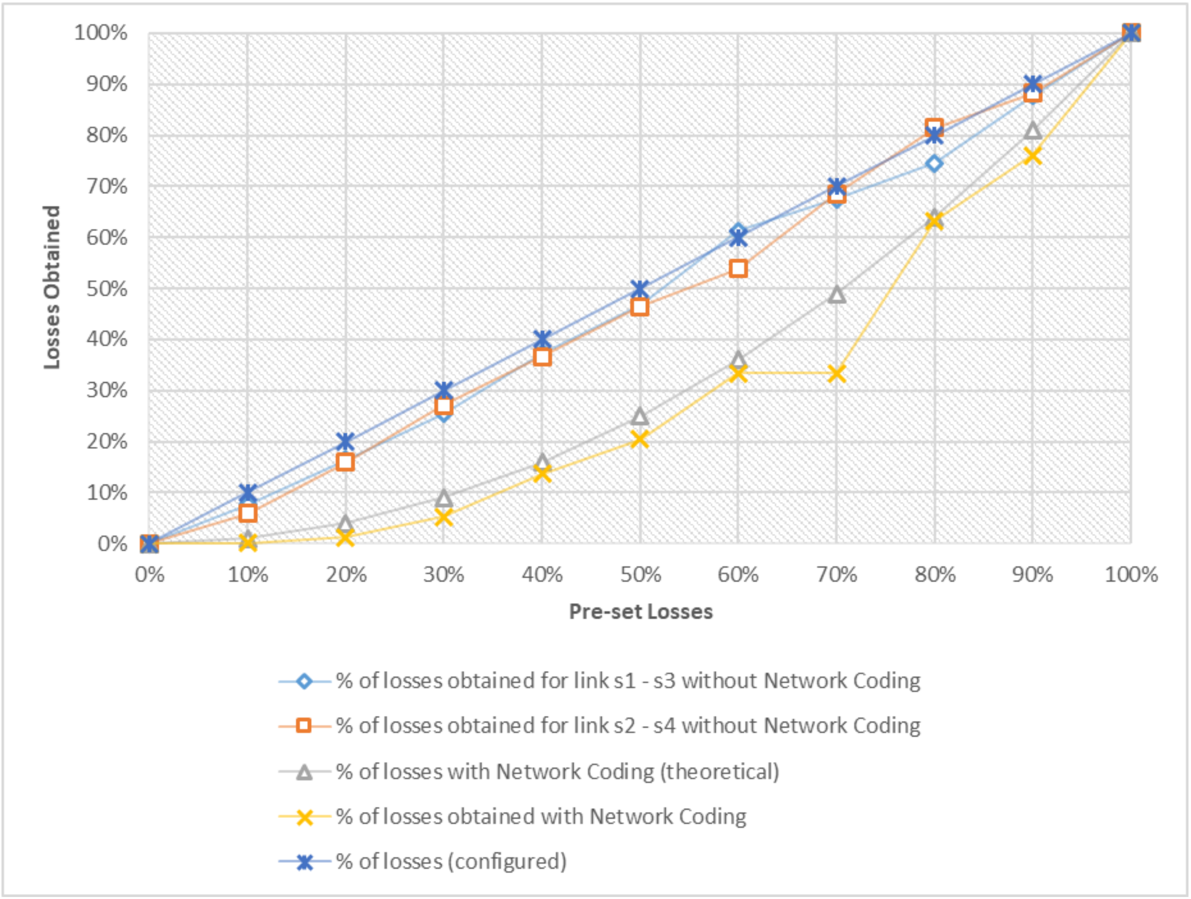

Fig. 9 Losses with and without Network Coding

V. CONCLUSIONS

The applications developed to implement the mechanisms of network coding and the modules for Ryu are indispensable for this prototype.

The modules generate rules that the controller has to install on the switches in order to command the forwarding of every frame to the applications that generate coded packets and recover lost packets. In addition, due to the limitations of OpenFlow and the incapability of switches that are not able to edit the content of a data frame beyond the headers of layer 2, 3 or 4 protocols.

For the tests, ICMP was used because it is ease to monitor it and to interpret the obtained results. ICMP does not perform resend in case of no response, and this facilitates the determination of the number of lost packets and the number of recovered packets thanks to network coding. The individual request and response mechanism of ICMP allows a step-by-step tracking of the path of each packets through the network with a butterfly topology, which facilitated obtaining the results.

As the percentage of losses in each link increases, network coding becomes less efficient in recovering lost packets, because of the greater probability of two simultaneous losses in the network.

TABLE III.
TESTS RESULTS

% of losses (configured)	% of losses obtained for link s1 - s3 without Network Coding	% of losses obtained for link s2 - s4 without Network Coding	% of losses with Network Coding (theoretical)	% of losses obtained with Network Coding
0%	0.0%	0.0%	0%	0.0%
10%	7.6%	5.9%	1%	0.1%
20%	16.4%	16.0%	4%	1.3%
30%	25.5%	27.1%	9%	5.3%
40%	37.3%	36.7%	16%	13.7%
50%	46.7%	46.5%	25%	20.4%
60%	61.3%	53.9%	36%	33.4%
70%	67.5%	68.5%	49%	33.4%
80%	74.5%	81.5%	64%	63.3%
90%	87.8%	88.3%	81%	76.1%
100%	100.0%	100.0%	100%	100.0%

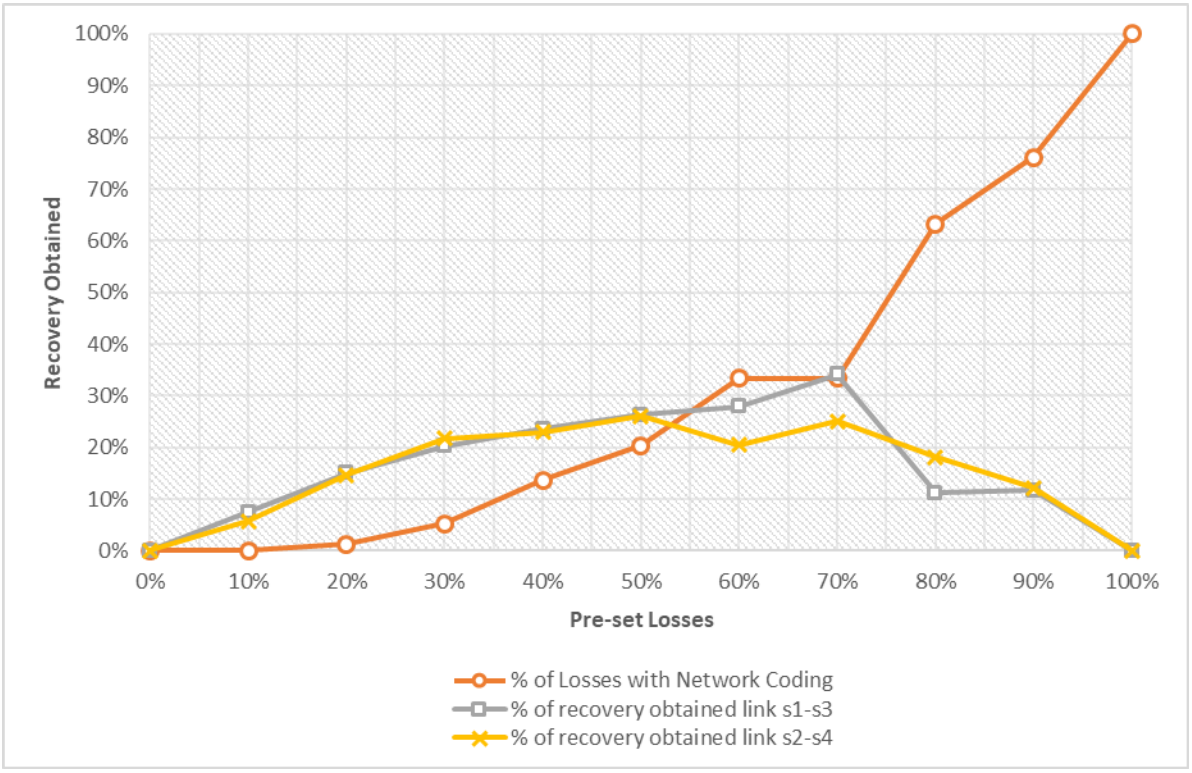

Fig. 10. Recovery percentage obtained

The network coding mechanism for lost packets recovery is considered a very promising tool for link loss values of less than 50%, as can be seen in Fig. 10. Once this value is exceeded, the percentage of losses expected in the network increases, approaching the value of losses without network coding. Table IV resumes the recovery percentage obtained considering certain percentage of loss packets.

Based on the obtained results, it is verified that the prototype allows the use of network coding as an error recovery tool, where lost packets in one of the links of the butterfly network can be recovered base on a new coded packet sent through the network.

TABLE IV.
RECOVERY PERCENTAGE OBTAINED

% of Losses with Network Coding	**% of recovery obtained link s1-s3**	**% of recovery obtained link s2-s4**
0.0%	0.0%	0.0%
0.1%	7.5%	5.8%
1.3%	15.1%	14.7%
5.3%	20.2%	21.8%
13.7%	23.6%	23.0%
20.4%	26.3%	26.1%
33.4%	27.9%	20.5%
33.4%	34.1%	25.1%
63.3%	11.2%	18.2%
76.1%	11.7%	12.2%
100.0%	0.0%	0.0%

The implemented prototype using SDN generates delays due to the need to control the order in which the frames arrive to different nodes, as well as the processing done in the data encoding and decoding equipment. Simulation of simultaneous transmission of packets with *at* command allowed to send packets with the least possible time difference.

These time differences between packets are due to the internal clock of the processors of each terminal and cannot be fully synchronized by an NTP server due to intrinsic delays, as well as the rest of the applications that are running in each of the terminals, which occupy time spaces in the processor that cannot be controlled.

ACKNOWLEDGMENT

The authors wish to express their gratitude to Escuela Politécnica Nacional, for the financial and logistical support, particularly during the development of the activities of the Internal Research Project PII-15-13.

REFERENCES

[1] W. Braun and M. Menth, “Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices,” *Future Internet*, vol. 6, no. 2, pp. 302–336, May 2014, doi: 10.3390/fi6020302.

[2] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined

Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[3] A. Gupta *et al.*, "SDX: a software defined internet exchange," in Proc. ACM SIGCOMM, 2014, pp. 551-562, doi: 10.1145/2619239.2626300.

[4] A. Hirata, D. Miyamoto, M. Nakayama, and H. Esaki, "INTERCEPT+: SDN Support for Live Migration-Based Honeypots," in Proc. BADGERS, 2015, pp. 16-24, doi: 10.1109/BADGERS.2015.013.

[5] M. E. Olaya, I. Bernal, and D. Mejia, "Application for load balancing in SDN," in Proc. EATIS, 2016, pp. 53-60, doi: 10.1109/EATIS.2016.7520102.

[6] N. Zope, S. Pawar, and Z. Saquib, "Firewall and load balancing as an application of SDN," in Proc. CASP, 2016, pp. 354-359, doi: 10.1109/CASP.2016.7746195.

[7] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem", *SIGCOMM Comput. Commun. Rev.*, Vol 42, No 4, pp.473-478, September 2012.

[8] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi, "An architectural evaluation of SDN controllers," in Proc. ICC, 2013, pp. 3504–3508, doi: 10.1109/ICC.2013.6655093.

[9] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010, doi: 10.1109/TIT.2010.2054295.

[10] R. Bassoli, H. Marques, J. Rodriguez, K. W. Shum, and R. Tafazolli, "Network Coding Theory: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1950–1978, 2013, doi: 10.1109/SURV.2013.013013.00104.

[11] J. Di and J. Dong, "A Network Coding architecture base on OpenFlow network," in Proc. MMME, 2016, doi: 10.2991/mmme-16.2016.50.

[12] J. Wang, J. Ren, K. Lu, J. Wang, S. Liu, and C. Westphal, "An optimal Cache management framework for information-centric networks with network coding," in Proc. IFIP Networking, pp. 1–9. 2014, doi: 10.1109/IFIPNetworking.2014.6857127.

[13] D., Szabó, A. Gulyás, F. Fitzek, and D. Lucani, "Towards the Tactile Internet: Decreasing Communication Latency with Network Coding and Software Defined Networking," in Proc. European Wireless Conference, 2015.

[14] D. Szabó, F. Németh, B. Sonkoly, A. Gulyás, and F. H. P. Fitzek, "Towards the 5G Revolution: A Software Defined Network Architecture Exploiting Network Coding as a Service," in Proc. ACM Conference on Special Interest Group on Data Communication, 2015, pp. 105-106, doi: 10.1145/2785956.2790025.

[15] C. Barakat, E. Altman, and W. Dabbous, "On TCP performance in a heterogeneous network: a survey," *IEEE Communications Magazine*, vol. 38, no. 1, pp. 40–46, Jan. 2000, doi: 10.1109/35.815451.

[16] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and Ligia Rodrigues Prete, "Using Mininet for emulation and prototyping Software-Defined Networks," in Proc. COLCOM, 2014, doi: 10.1109/ColComCon.2014.6860404.

[17] S. Wang, H, Chiu, C. Chou, "Comparisons of SDN OpenFlow Controllers over EstiNet: Ryu vs. NOX", in Proc. ICN the Fourteenth International Conference on Network, 2015, pp 249-249.

[1] N. McKeown *et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, Mar. 2008, doi: 10.1145/1355734.1355746.