



Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Networked Systems and Services

INTEGRATING NETWORK CODING INTO SDN NETWORKS

NETWORK CODING INTEGRÁCIÓJA SDN HÁLÓZATOKBA

Author

Nagy Szilárd Attila

Supervisors

Szabó Dávid
Csoma Attila

Contents

Absztrakt	3
Abstract.....	4
Introduction.....	5
1 Novel technologies.....	7
1.1 Software Defined Networking	7
1.2 Network Function Virtualization	9
1.3 Network Coding.....	11
1.3.1 Random Linear Network Coding.....	12
1.3.2 Kodo library	15
1.3.3 RLNC compared with other scenarios.....	17
2 NFV implementation platform	21
2.1 General overview	21
2.1.1 NFV Platform implementations.....	22
2.1.2 A feasible NFV platform implementation	25
2.2 ClickOS.....	26
2.2.1 About ClickOS.....	26
2.3 Acquiring ClickOS	28
3 Creating a VNF	31
4 Performance Measurements	36
4.1 VNF performance in ClickOS and in User-level Click	36
Implementing RLNC as ClickOS VNF	40
Conclusion	41
List of figures.....	42
References.....	44

Absztrakt

Napjainkra a hálózatokkal szemben támasztott követelmények elérték azt a szintet, ami a jelenlegi rendszer teljesítőképességének a határait feszegeti. Ezt remekül érzékelteti a hálózatos körökben egyre gyakrabban elhangzó 1 ms késleltetési idő elvárása a 2020-ra beharangozott 5G-vel kapcsolatban. A szigorodó igények kiterjednek a rendelkezésre állásra, a biztonságra és a sokrétű nagy sebességű szolgáltatásokra is. Számos szakember szerint a megfelelő válasz a korábbi évek inkrementális fejlesztésével szemben (2G, 3G, 4G) egy paradigmaváltás, amely képes új alapokra helyezni a rendszert.

Több elképzelés is létezik a lehetséges irányokról, amelyek közül a jelenlegi kutatási eredményeket is figyelembe véve a Network Coding tűnik a legígéretesebbnek. Ez a technológia szakít a hagyományos csomagkapcsolt hálózatok kommunikációs gyakorlatával és a jelenlegi router-ekben alkalmazott “tárol és továbbít” megközelítést “feldolgoz és továbbít” megközelítésre cseréli. A Network Coding mögött mély matematikai háttér húzódik, amely végeredményben lehetővé teszi a hálózati eszközök hatékonyságának növelését, a késleltetés csökkentését és a biztonság egyfajta implicit módon való fokozását, így megfelelő választ jelenthet a fent vázolt problémákra.

Mindazonáltal az alkalmazhatósághoz vezető első lépésként szükség van Network Coding képes eszközökre a hálózatban, amely a jelenlegi szemlélettel majdnem az összes csomópont cseréjét igényli.

Erre a problémára jelenthet megoldást a Software Defined Networking (SDN) és a Network Function Virtualization (NFV) technológiák együttes alkalmazása. Dolgozatomban azt vizsgálom, hogyan lehet hatékonyan integrálni a Network Coding funkcionalitást SDN hálózatokban, ezáltal lehetővé téve a széleskörű alkalmazást egy a jelenleginél hatékonyabb hálózat működésének érdekében.

Abstract

Nowadays, with the ever-growing demands from subscribers toward network reliability and performance has grown so wide that the current system is unable to provide the required services. Besides, the planned new telecommunication network, 5G has set up requirements like zero-latency (1 ms) to support tactile internet, machine control or augmented reality, which seems impossible with the current architecture. Thus, technology shift is required in this field.

These new possible approaches are still under construction and haven't been reduced to a single idea. Numerous experts believe that the solution could be the Network Coding technology to increase network capacity and providing almost no latency between any nodes. This novel approach also achieves greater security due to encoding every packet, and enable the possibility to use multipath routing in a seamless way. Another virtue of Network Coding is the new compute-and-forward mechanism, which breaks with the common store-and-forward approach. This new access enables to forward every incoming packet instantly, and also has the possibility to generate redundancy, which can eliminate packet losses.

However, the problem with this new approach is that it is impossible to implement it in the current network architecture, due to the fact, that every single middlebox and all of the other network nodes are needed to be replaced with compatible devices.

Therefore, it is very crucial to find a solution to implement Network Coding into the current networks, thereby in this paper I'm going to take essential steps toward achieving that goal.

Introduction

In today's telecommunications network the number of middleboxes (any device that manipulates the network traffic) has grown significantly. Their importance is also increasingly grow due their essential functions such as traffic filtering, load balancing or intrusion detection (IDS). However, they also come with immense drawback, due to their high cost of design and development. Furthermore, their maintenance and operations are not user-friendly either. Another issue of middleboxes is the impossibility to improve their functionality and to develop them further, so the only way to replace or acquire a new middlebox with the required functions.

It is also rather unfortunate that the current network is still based on the same principles which were invented in the dawn of computer systems by engineers. In the last decades it was refined in several times and with technological improvements it became possible to achieve better performance. However, this approach already reached its limits. To resolve this obstacle a new approach need to be introduced.

According to current researches this paradigm shift could be the integration of Network Coding (NC) into the network. This novel technology enables to increase the speed of network communication, data encryption and also reduce network latency by introducing the compute-and-forward mechanism, instead of the store-and-forward mechanism at network devices. As a consequence, integrating NC into the telecommunication networks would require to replace most of the existing middleboxes which is a highly non-trivial problem. A solution could be to improve the hardware based network devices, but as mentioned before, it would be nearly impossible.

The combination of the recently expanding Software Defined Networking (SDN) combined with Network Function Virtualization (NFV) technology offer a feasible solution to the problem (the previously mentioned difficulties in developing and modifying the hardware based middleboxes). One of the most promising networking tendency (SDN), enables to control network traffic while the NFV approach makes it available to implement and integrate different functions into the network as a software. These technologies create an opportunity to expand the network capabilities in such a way, that it doesn't require to replace the whole architecture and network nodes, or if it does, then it enables us to do it in small steps and also in a much more cost effective way.

This paper is organized as follows: in Section I I provide deeper understanding for these new innovative technologies (SDN, NFV, NC); in Section II I present some possible NFV platform solutions, then I'm going to choose one on which I'm going to deploy a VNF; in Section III, I demonstrate step-by-step how to make a VNF on the chosen platform; in Section IV I'm going to provide measurements results; and last but not least, I make conclusion about the obtained results.

1 Novel technologies

In the following subchapters, I'm going to present SDN, NFV and NC principles to get a deeper understanding about how are these technologies work, and why they are essential in the future telecommunication network architectures.

1.1 Software Defined Networking

The demands of networked computer systems has changed so dramatically over these years that it requires to re-evaluate the system. With the increasing usage of cloud-based networks, virtualized desktops and servers, or remote data-storage devices to name a few, the need of computing power, resource distribution, and planning are inevitable to deploy these services in the network. However, these network functions are also needed to be maintained to function properly, which gets harder every day, due to the current architecture.

That is the main goal of Software Defined Networking paradigm (SDN), namely to provide a programmatic interface to network devices, so it becomes possible for network engineers to manage network services through a higher-level of abstraction. The main idea is to separate the data plane from the control plane. By doing that, it creates a possibility for a centralized network control which dramatically decreases the cost of installation, maintenance and management. Furthermore, SDN enables network administrators to handle the whole infrastructure as one entity.

This novel approach creates a dynamic network architecture which can be customized and optimized as needed through an application programming interface (API). SDN uses two APIs, one is called Northbound API which is the interface towards applications, and the interface towards the data plane is called Southbound API. We can see in Figure 1, how an SDN architecture is built.

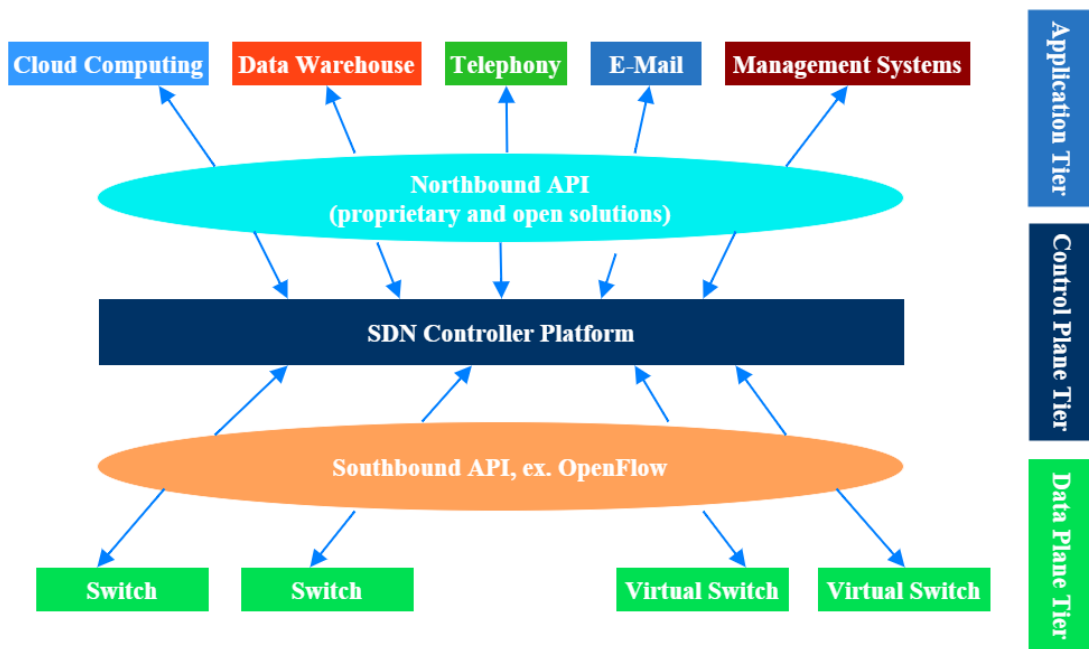


Figure 1: SDN architecture [1]

For the Southbound API, the most preferred protocol is the OpenFlow, which was created by the Open Networking Foundation (ONF) [2]. Since it was established (2011), they were determined to promote the SDN approach, so they designed the OpenFlow protocol for networking hardware, thereby providing a reliable interface to manage SDN switches.

Due to the logic is sourced in the controller, therefore it doesn't require to use specialized and smart network devices. Commercial, off-the-shelf (COTS) hardware can be placed on their steads, because OpenFlow enables the communication between them and the controller platform, and provides a way for the instructions to be executed.

The power of OpenFlow comes from its simplicity. It uses flow tables, which contains rules. If an incoming packets matches any rule, then it will be processed accordingly, without any interaction needed from the controller, thus decreasing latency. Otherwise, if the packet doesn't match any rule, or the flow table is empty, then it is forwarded to the controller, which can decide what to do with that peculiar packet, and can install new rules accordingly to the switches/routers. Because of this behaviour, after some time the flow-tables can achieve a semi-stable state on the network, so every packet can be handled only on the data path, which means increased speed in processing.

The other benefit, which comes with flow table that it opens up plenty new ways for routing, thus enabling new services for each customers separately. For example it becomes possible, to not only route based on destination address, but also on source address.

SDN is a flexible, agile, programmable and most important, open standards-based and vendor neutral platform, that is capable of handling the most demanding current, and future networking need.

1.2 Network Function Virtualization

Modern telecommunication networks contain an ever increasing number of network devices apart from switches/routers. These are the hardware appliances, a.k.a. middleboxes, that manipulates network traffic implements various essential functions like traffic filtering, load balancing, domain name service (DNS) resolving, intrusion detection (IDS), network address translation (NAT) and caching to name a few.

Although, these are stationary appliances, and there are often a need for functionalities to be deployed elsewhere. Another problem with middleboxes lie in their hardware based nature, because they are not just extremely expensive, but excessively hard to maintain and operate them. Using OpenFlow would solve these problem, but it would bring forth another one, namely it would decrease network performance and increase delay in each communication, due to filling each middleboxes' queue. Further issue with them is the impossibility to update or upgrade with new features. Therefore a new approach is desired to be able to deploy middleboxes whenever and wherever is needed efficiently.

As a result of these drawbacks, in October 2012, seven of the world's largest telecom operators presented a new proposal, named Network Function Virtualization at the SDN and OpenFlow World Congress [3]. It is important to note that NFV differs from SDN, although it is common to use them together due they complement each other. While SDN aims to provide a programmatic interface to network devices, NFV targets to turn hardware middleboxes into software components.

First white paper released in November 2012, and selected the European Telecommunications Standards Institute (ETSI) [4] to be the main home of NFV. ETSI

approved the proposal and created the NFV Industry Specification Group (NFV ISG) with the objective to develop the required standards. NFV ISG publishes NFV use cases, proof of concepts, architectural framework and terminology, which is required for researchers to effectively realize NFV tools and principles. ETSI NFV ISG vision for NFV which relies on COTS hardware and software delivered through the cloud can be seen in Figure 2 [5].

NFV offers a new way to design, deploy and manage networking services, because instead of using hardware appliances, it realizes middleboxes as a software component. Due to this new approach, it becomes possible to deliver agility and flexibility to networks, because middleboxes now can be deployed anywhere on demand, can be scaled up or down to address changing demands from network users, and most importantly doesn't require dedicated special hardware, because it can run on COTS hardware or even in a virtualized computer. Another advantage of NFV, that it reduces the time needed to deploy new services into networks and because they are a software, it can be changed as required to consolidate new needs. In addition, NFV reduces the capital expense (CapEx) by eliminating wasteful overprovisioning and also reduces the operational expense (OpEx), because it doesn't require space, nor heat-ventilation-air conditioning (HVAC) and it simplifies the management and roll out of network services.

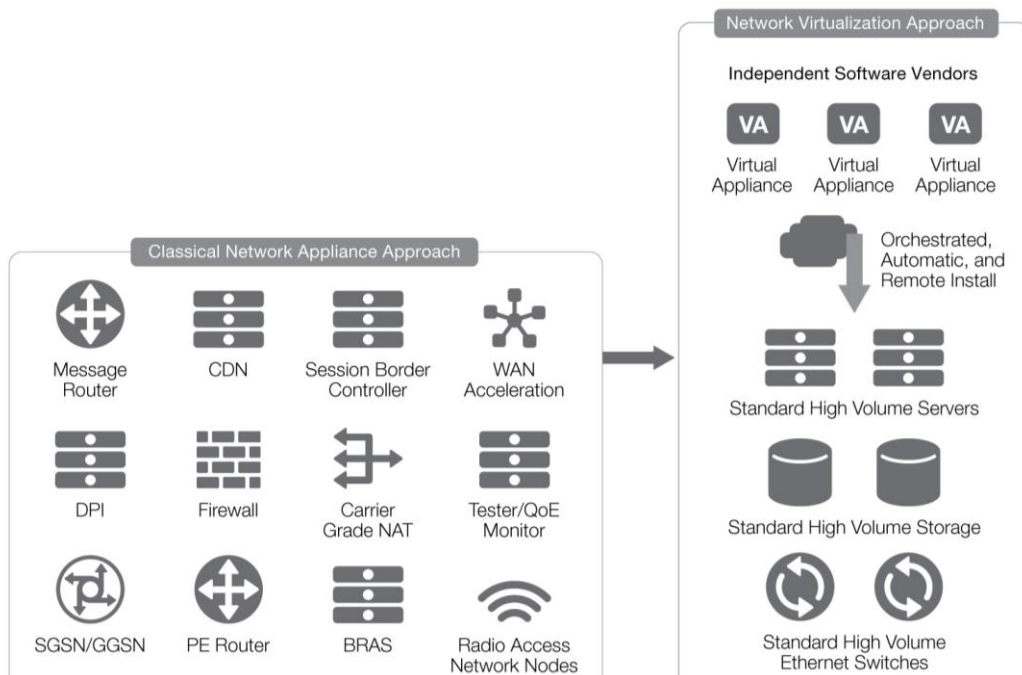


Figure 2: ETSI vision for NFV [5]

NFV decouples the network functions from proprietary hardware appliances so they can run as a software.

1.3 Network Coding

Network Coding (NC) [6] is a new discipline in which the transmitted data is encoded at source, recoded in the path, and only decoded at the destination. This new approach increases network throughput, significantly reduces latency, makes the network more robust and also gives provides greater security against eavesdropping, hacking and other forms of attack.

NC breaks with the current store-and-forward mechanism, in which packets are copied then forwarded towards its destination. Furthermore, Kirchoff's law (the sum of bits flowing into that node is equal to the sum of bits flowing out of that node) doesn't apply anymore because network coding treats each independent data flow as algebraically combinable information. Therefore the sum of bits flowing out of the node are not required to be equal of the sum of bits flowing into this node (so the output is the function of the input).

These nodes that apply some function on the data flows are called either sources or relays, according to their position in the actual route between two communicating devices. Implicitly, the node that will apply the inverse function, to retrieve the original flow is called the decoder. Therefore each path always contains one source and one sink, and can contain zero or more relays as shown in Figure 3, which is a simple topology that uses multipath routes between the source and destination nodes.

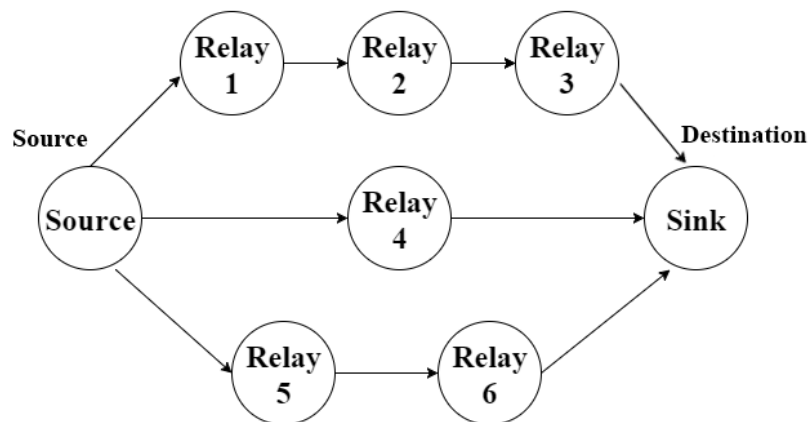


Figure 3: Simple (multipath) topology for naming conventions

In the beginning, when network coding was first introduced [7], the elementary butterfly topology, as can be seen in Figure 4, widespread over the world, and mistakenly cause a confusion about the technology, thinking network coding can only be used in such a farfetched situation. This misunderstanding even appears in literature, forestalled its evolution and also clouded its immense assets. Today it has been eliminated, therefore NC can be used in any network topology.

Nowadays, NC has greatly evolved since its formalization and there are different implementations of the mathematical concept. One of these implementation is the random linear network coding (RLNC), which breaks up with “butterfly topologies” and XOR operators, instead it is creating linear combination of the incoming flows. The way the coding vector are produced is based on random number generation, hence its name. Furthermore, RLNC defines a new function for relays by enabling them to recode incoming packets without waiting for the whole original data to arrive. This improves transmission speed and also reduces latency between communicating participants. In the implementation relays are often called as recoders. (Although they are not required to always recode every data flow, it can be set as demanded.)

1.3.1 Random Linear Network Coding

RLNC is a rateless code, which means an infinite number of representations of the original data can be created. This makes it available for this technique to recover from any number of erasures. It comes from the mathematical construct behind it, which is called Galois Field (GF) or finite field. A finite field is a variable where special rules are defined for the arithmetical operators. These rules guarantee that operating on a GF, the

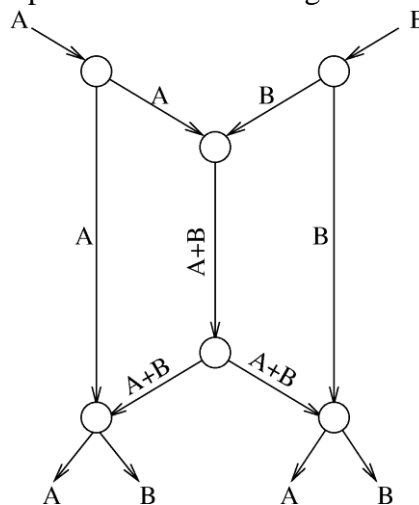


Figure 4: Elementary butterfly topology [8]

result of the operation will also be in the same finite field. A common field for instance is a GF(2), where the addition is defined by the XOR operator. An illustrative example can be seen on Figure 4, which is the mentioned elementary butterfly topology, where recoders haven't existed, and each node still needed to wait some packets to be able to encode them together. Still, it is a great example to show how we can reach a higher throughput rate on the middle link using a basic network coding over a GF(2).

Essential naming conventions that RLNC uses [9] :

A **symbol** is a vector of GF elements that represent some data. The size of it depends on the number of elements and the size of each element.

A **coded symbol** is a symbol which is a combination of the original symbols in a generation, therefore it represent all the data in it, but it has the same size as the original symbol.

A **generation** contains g coded symbols of size m , where g is called the generation size. The g original symbols in one generation are arranged in the \mathbf{M} ($\mathbf{m}_1; \mathbf{m}_2; \dots; \mathbf{m}_g$) matrix, where \mathbf{m}_i is a column vector. The original data with the size of \mathbf{B} , is divided in $\left\lceil \frac{B}{m \cdot g} \right\rceil$ pieces creating $\left\lceil \frac{B}{m \cdot g} \right\rceil$ generations.

A **coding vector** describes how the symbol was encoded. It contains the coefficient for each symbol in the generation.

A **coded packet** is a pair of a coded symbol and a coding vector. These must travel together, due the only possible way to decode the symbols is by knowing the corresponding coded vector.

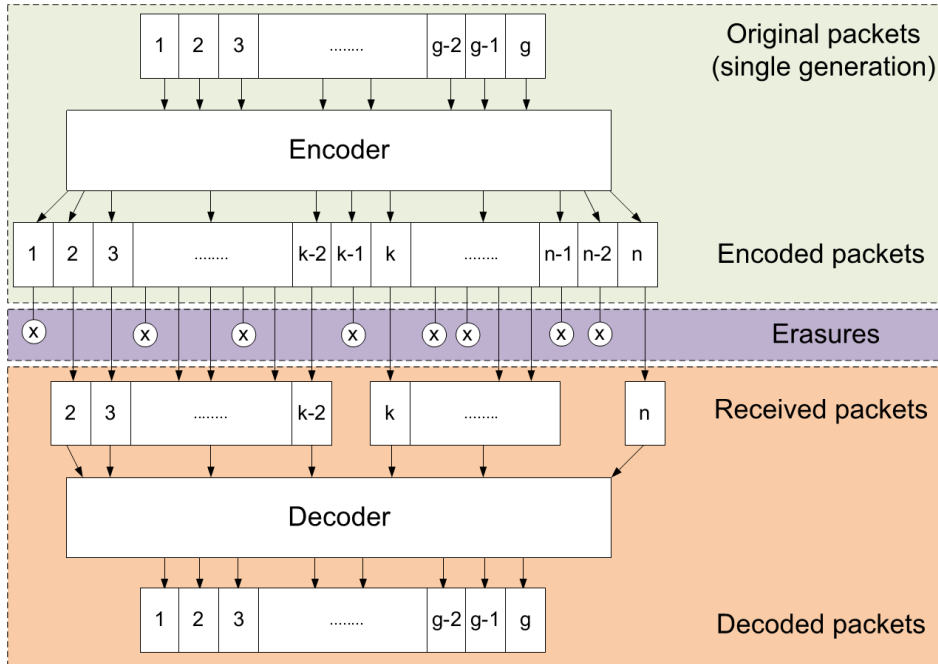


Figure 5: Overview of RLNC [10]

A basic overview of RLNC can be seen in Figure 5. If the original data is large, then it needs to be split into multiple generation, because if the whole data would be considered as one generation, then the computing complexity would be very high. In this scenario, the encoder generates linear combinations of the original symbols which will represent some part of the original data but has the same size as one symbol. Because it operates over a GF, means it can create infinite number of linear combinations, thus providing the possibility to recover any number of erasures. After the encoded packets transmitted, some will be erased in the lossy channel. This doesn't matter, if at least g linearly independent packets arrive at the decoder, because that way the decoder can still decode the original data (by choosing a feasible GF size there is a very high probability (almost 1) that a linearly independent packet will be generated). All received symbols are placed in the matrix $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1; \hat{\mathbf{x}}_2; \dots; \hat{\mathbf{x}}_g]$ and the coding vectors in the matrix $\hat{\mathbf{V}} = [\hat{\mathbf{v}}_1; \hat{\mathbf{v}}_2; \dots; \hat{\mathbf{v}}_g]$. The original \mathbf{M} data can be decoded as $\mathbf{M} = \hat{\mathbf{X}} * \hat{\mathbf{V}}^{-1}$.

Network Coding involves some overhead, due to the coding vectors, because it is needed to be added in the encoded packet. The size of it depends on the GF size, the generation size (g) and the representation of the coding vector, but in practise, it is smaller compared to the payload size. The other source of overhead can come from the GF size,

if we chose a small field size, due to it might produce a small number of linearly dependent coded packets.

In the profile of delays, the generation size is what matters. With g generation size, at least g symbols, which means $g * m$ amount of data must be received to start decoding,

RLNC is useful in many different scenarios, for example in point-to-point communication due to it can eliminate packet losses on a lossy link. It can realize reliable multicast in wireless networks by efficiently using broadcast transmissions. Furthermore, by overshooting symbols in each generation, it work as a Forward Error Correction (FEC) code as well, eliminating possible link losses, and if retransmission is needed, there is a high probability that one retransmission will be sufficient for multiple sinks. Due to this ability of RLNC, it is also a great way to utilize best-effort multicast, for example in video streaming. In a multi-hop network with the ability to recode, intermediate node helps to minimize signalling between two communicating devices, due each recoder can generate another linear combination, which means it will contain new information with a high probability. Therefore, instead of the source providing new information, a closer node can do that task.

Network Coding changes the current network's packet forwarding principles, and nowadays with Random Linear Network Coding, it can greatly decrease latency, improve throughput, reliability and security.

1.3.2 Kodo library

There are many implementations of Random Linear Network Coding including some proprietary solutions as well. I chose the Kodo implementation, because compared to others, it provides the highest coding speed and also the most functionalities that comes with the package.

The first comparison (Table 1), compares the coding speed of the existing libraries using different generation size parameters (in other words, using more and more packets coded together). All measurements used $GF(2^8)$ with 1 MB packet size.

Table 2 shows the different functionalities that each implementation supports.

$F = GF(2^8)$ $P = 1 \text{ MB}$	Kodo 17	ISA-L	Jerasure 2.0	OpenFEC
G=8	3096/280	2255/2635	1250/1365	353/292
G=9	2542/2559	1961/2252	1096/1185	305/264
G=10	2136/2227	1724/1796	997/1072	285/245
G=16	1807/1496	1075/1180	628/644	179/160
G=30	950/647	266/271	349/361	96/90
G=60	594/329	123/122	184/184	48/46
G=100	383/209	74/73	111/111	29/28
G=150	266/141	47/46	74/74	19/19

Table 1: Performance comparison of RLNC implementations in coding speed [22]

Library Capabilities	Kodo	Jerasure 1.2	Jerasure 2.0	ISA-L	Open FEC
Reed-Solomon Codes Supported	✓	✓	✓	✓	✓
Network Coding Supported	✓				
Updated with Novel Code Support	✓				(✓)
Contiuous Optimization of Algorithms	✓				
Automatic Adaptation to CPU Features	✓				
OS Support	Ubuntu, Debian, Arch Linux, Windows, Android, IOS, Kindle Fire HD, Raspberry PI, Open WRT	Ubuntu, Debian	Ubuntu, Debian	FreeBSD, Ubuntu, Debian, Windows	Ubuntu, Debian, MacOSX
Compiler Support	GCC, Clang, MS VS, Apple LLVM 6.0	?	GCC	GCC	?
Date of Lat Release	10/2015	8/2008	1/2014	11/2013	12/2014
Hardware Acceleration on Intel Chipsets	SSSE3, SSE4.2, CLMUL, AVX2		SSSE3	SSSE3, CLMUL	SSE
Hardware Acceleration on ARM Chipsets	NEON				
Multi-core support	✓				
Simulation support	Internal, NS3				

Table 2: Functionality comparison [23]

All of the compared erasure code implementations are open source projects if used for research or individual purpose.

1.3.3 RLNC compared with other scenarios

In this section some measurement results are provided that compare random linear network coding against block or fountain coding schemes (Raptor, Reed-Solomon) [24]. These codes can be used basically in two way: in an end-to-end (E2E) manner or in a hop-by-hop (HbH) manner. In E2E, encoding and decoding only performed once, in the endpoints of the communication at E and D (Figure 6). This implies that E should emit enough amount of extra packets to ensure that all of the information can be recorded at D. The intermediate nodes acts as simple store-and-forward nodes, namely they only forward successfully received packets. The HbH approach unburdens the network from unnecessary packets, but at the same time, it also infuses extra latency as every relay has to wait to receive the full message in order to be able to start encoding.

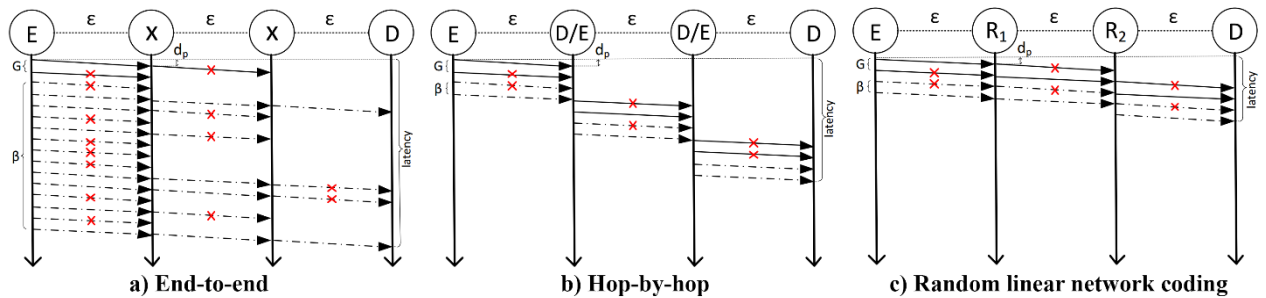


Figure 6: Illustration of E2E, HbH and RLNC coding schemes, with 50% probability loss on each link

The measurements were carried out on a fully fledged implementation of the compute and forward router with the respective networking scenarios built in Click. The parameter setting include erasure probability (ϵ), packet size (L), generation size (G), number of hops (H) and channel rate. The analysis is done assuming a single path – multi hop channel where E delivers a message of G packets through H number of relays to a decoder D. The link loss probability on each link set on $0 < \epsilon < 1$.

The first measurement examines the overall number of sent packets that D requires to successfully decode the message. In Figure 7, where packets conveyed in a three hop communication network, the theoretical (indicated by (T)) and measured results can be seen. It shows that while RLNC and HbH packet number linearly with the loss, the E2E increases exponentially.

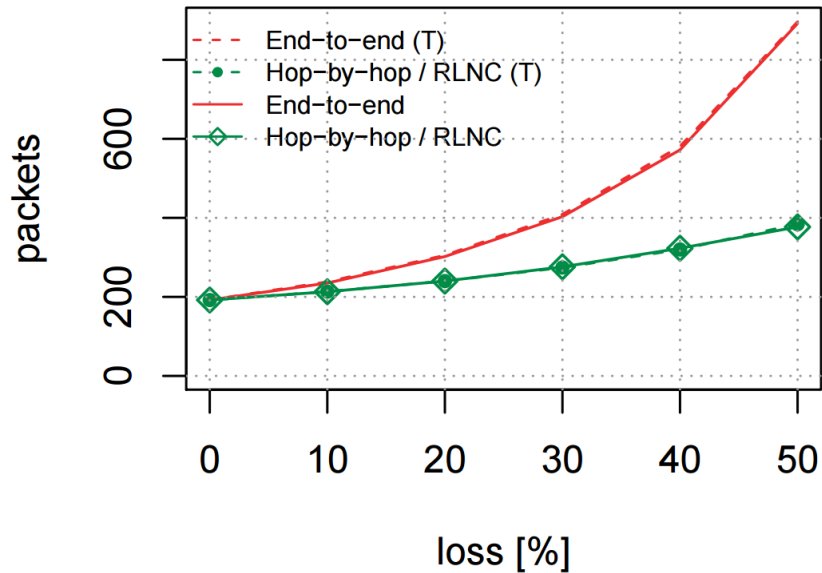


Figure 7: Number of overall packets require to successfully decode the full message

The next measurement gives a different result, where the concern was on the latency versus different channel rate. This simulation was taken on the same three hop communication without any loss. As the result shows (Figure 9), this time RLNC is compared with E2E coding. Since there are no packet losses E2E with immediate forwards performs just as well as RLNC. In the HbH case each node has to wait to fully receive the whole message in order to be able to start forwarding. The gain of RLNC over HbH remain constant for the higher values which means that the ratio of latency is independent from the bandwidth.

The latency results change a lot if the channel is error prone with the probability of 50% (Figure 8). Now the advantage of RLNC over the other two schemes becomes evident and E2E is now even worse than HbH. After having a look again at Figure 6 this is not so surprising, since E2E have to send through all redundancy on the whole channel.

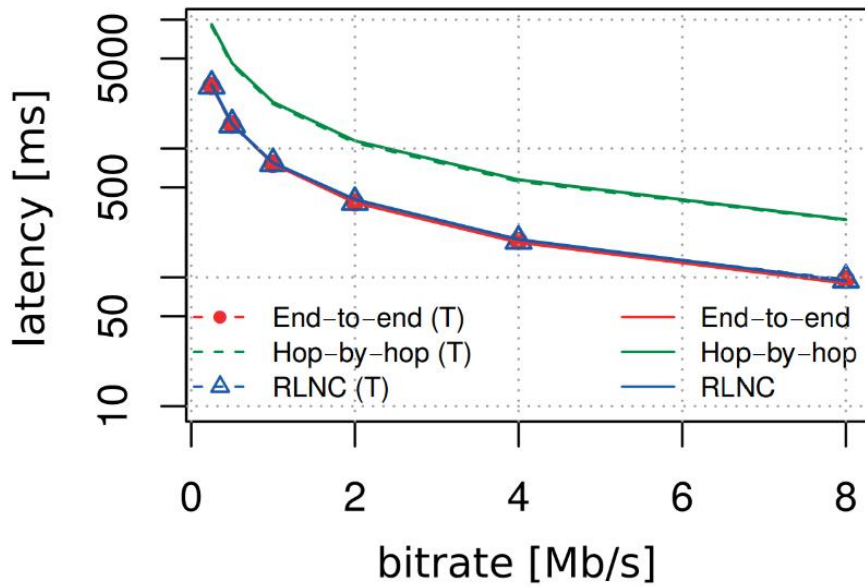


Figure 8: Latency results over different channel rates, and no losses

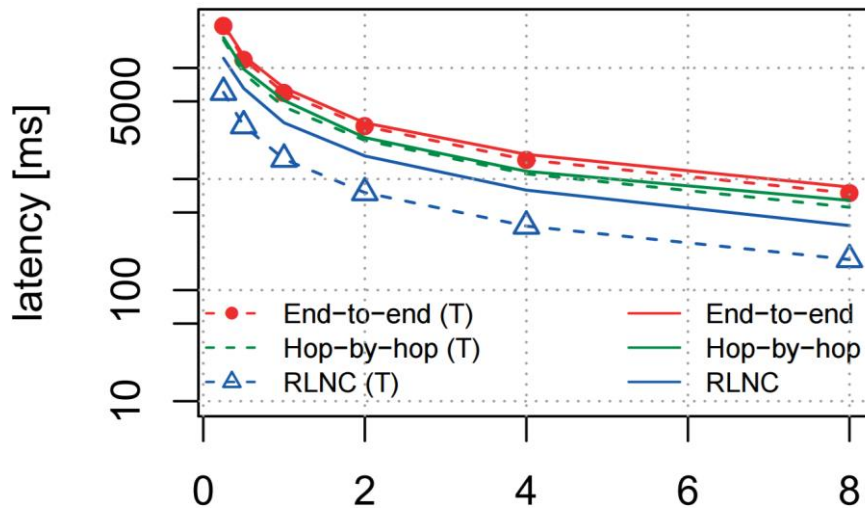


Figure 9: Latency results over different channel rates and 50% probability of loss

Finally, in Figure 10 the latency for the three transmission schemes depending on number of hops and loss probabilities is given. In the case of small number of hops with low loss E2E can keep pace with RLNC, at the expense of more sent packets. However, the latency increases significantly for large number of hops that are highly error prone. For HbH it increases linearly with the number of hops and increases with the probability of losses. RLNC has a lower latency than the other two schemes over a wide range of parameters.

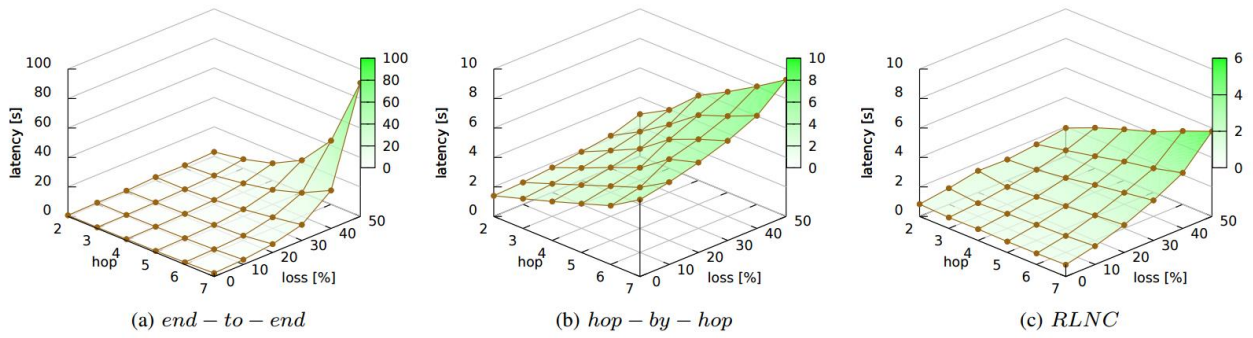


Figure 10: Latency for the three transmission schemes depending on number of hops and loss (Packets 64 – Size 205 B – Bitrate 0.25 Mb/s).

2 NFV implementation platform

2.1 General overview

The utilization of NFV in the networks, would dramatically change the current networking practices. NFV can help reducing operation and equipment cost, can also lower power consumption and reduce time-to-market for new services or functionalities. However, to successfully integrate VNFs into the current system, an NFV platform/NFV Infrastructure is required.

An NFV platform must fulfil plenty requirements [11]. The foremost requirement is to be reliable and efficient, since service providers (SP) won't accept, if a network function is unavailable due to a cloud data centre or an entire region loses service. To guarantee this, it has to provide five-nines availability and also to offer at least the same quality of service levels as telecommunication networks. Since NFV platform runs on COTS equipment, therefore to provide availability, it is needed to be measured at the system level, and has to be able to transfer whole services if an error occurred. The efficiency part comes from the software based profile of VNF, because it provides a much higher degree of automatization, and can be programmed on demand.

Since they are intended to concurrently run software, they must support multi-tenancy. Therefore collocated middleboxes should be isolated in both performance and security point of view, in CPU, memory and device access. Furthermore, NFV platforms must accommodate with different OSes, APIs and software packages, hence the requirement for being flexible.

Further demand is to be able to achieve high throughput with low delay. It is more concerned requirement, due to middleboxes would be deployed typically in operator environments, so it needs to handle large traffic rates while adding negligible delay to RTTs.

Another important necessity is to be scalable. Since SPs would run middleboxes for third-parties, they must be very efficient. The platform should ideally support large number of middleboxes belonging to different third-parties. For this reason, and to make better use of additional resources, or additional servers, the platform should be able to quickly scale out processing on demand.

By taking ETSI ISG view of what requirements should a NFV Infrastructure platform fulfil, we can see the resemblance with the criteria above, just in a more general, service provider point of view. In Alcatel-Lucent's whitepaper titled "Why service providers need an NFV platform", they provide a detailed description for a NFV platform demands [12].

A NFVI platform must support distributed architecture to provide the needed flexibility with as little delays as possible, just as in the telco networks. Moreover it should automatically find the optimal workload locations to further improve in the performance aspect. The distributed datacenters and networks should also be managed and orchestrated as a single virtual cloud, in order to be able to analyse and monitor the entire cloud platform in real time.

The next condition is about the cloud nodes that they must be highly automated, and should be pre-configured to be able to replace services on demand, or to eliminate possible mechanical failures in the COTS hardware.

A further requirement for a NFV platform is to have an automated application lifecycle management, to enable new services to be deployed in minutes, instead of weeks, like nowadays.

To maintain the network with deployed VNFs, the platform must be rapidly configurable, and should have a flexible network abstractions, thereby should have access to SDN, to be able to automatically create the required communication paths between different VNF applications.

Last but not least, it needs to be an open and shared environment, due to the platform should be able running applications from different vendors. Therefore, it is important to support industry-standard APIs.

2.1.1 NFV Platform implementations

CloudBand [12]

This NFV platform consists two major components. One is the CloudBand Node, which provides the infrastructure, that the ETSI NFV set up, and the other on is the CloudBand Management System, which provides the required management and orchestration framework.

Their north and southbound APIs are using industry-standard open APIs, such as OpenStack. For the lifecycle management, it uses Carrier PaaS, while the cloud optimization functions runs on their own algorithms. CloudBand integrates with the Nuage Networks programmable SDN solution, and uses its framework to automatically set up the network structure.

CloudBand Red Hat approach

In this implementation [13], the platform uses CloudStack instead of OpenStack, which greatly increases the performance of the platform.

The other difference is that the virtual infrastructure manager (VIM) of it relies on Red Hat Enterprise Linux OpenStack Platform without any modification.

The drawback for these implementation is the OpenStack service, due to it is still in its early ages, and under heavy development in many areas. The other issue with CloudBand that it is not ready to use out of the box. The OpenStack need a lot of configuration, which are not very straightforward.

OPNFV

OPNFV [14] is a carrier-grade, integrated open source platform that looks to realize the ETSI NFV ISG's architectural framework. This solution is focused on the NFVI and VIM portions, because they aim to provide for the industry a good basis to build on. Therefore it is still under construction, and it is done by upstreaming and project collaborations. In this way it can ensure every requirements of the industry is fulfilled.

OPNFV Arno Release

The Arno release is a developer-focused release. It is aimed at those who are exploring NFV for proof-of-concepts, or developing VNF applications or just interested in performance and use case-based testing. It provides an initial build with the required infrastructure and VIM components of ETSI NFV architecture. OPNFV integrates components from the upstream of Arno release, and the community integrated components from upstream communities like Ceph, KVM, OpenStack, Open vSwitch etc. The advantage of OPNF is that it implements the ETSI NFV architecture, fully open source, and provides a good basis to build on, but it is still under development, not ready to deploy at the moment.

There are many other implementations that uses the OPNFV solution, but changes some functionality, or some parts of the implementation, therefore they become proprietary solutions.

HP OpenNFV Architecture

This is one of those implementation, which uses OPNFV as a basic. It relies on OpenStack de-facto standard, but made it more robust for communication service provider (CSP) environments. This cloud compute operating systems is called HP Helion OpenStack Carrier Grade [15].

vCloud NFV

Another platform that utilizes OpenStack bases is their solution is VMware [16]. Their platform is the vCloud NFV. It is also made for CSP environments using some of their own technologies like vSphere, vSAN, vCloud Director or VMware to provide to required functionalities.

Dell NFV platform

Dell also made their version of NFV platform, which is also based on OPNFV architecture [17]. For the NFV platform at the moment there are two starter kits available for early adopters, but they both runs on specific dell devices.

Intel NFV platform

Intel's NFV platform are based on OpenFlow protocol as the SDN southbound API, and runs on Wind River Linux distribution and uses KVM and OpenStack solutions. The problem with these platform comes from proprietary source profiles, therefore needs a special hardware/software components which are COTS devices or not open source.

2.1.2 A feasible NFV platform implementation

In another point of view, if we consider how middleboxes should be programmed, then it is recommended to support code re-use to reduce cost, time and overhead. Therefore it isn't needed for the platform to run one commodity OS just to support middleboxes coded as applications.

Continuing on this approach, there are plenty of ways to implement such a platform, given the goal is to run middleboxes on the same COTS hardware. One solution is using a container like chroot, FreeBSD Jails, Solaris Zones, OpenVZ to name a few. The advantage of using a container is that they are very popular, lightweight and easy-to-use. Although it forces all middleboxes to run on the same operating system, which is a limitation that conflicts the flexibility requirement as mentioned above.

Instead of using a container, the other possibility is to use a hypervisor. Hypervisors provide the flexibility that is needed for multi-tenant middleboxes, but this comes with the price of low performance. For this issue, a common solution is to utilize device pass-through, where the virtual machines can directly access the network interface card (NIC). Although this also has downsides, namely it complicates the live migrations and the COTS device is monopolized by that virtual machine, which harms the scalability criteria. There is also a workaround for the latter issue, this new technology is called hardware multi-queuing, but using this solution would still limit the number of VMs that can be concurrently run.

Further solution that can come to mind, is using a minimalistic OSs, or micro kernels. The reason they are attractive is due to they aim to provide just the required functionalities. Although they typically lack driver support, especially NICs, and most do not run in virtualized environments.

The final solution is to combine some of these approaches, to achieve a system that fulfils the delineated requirements. To achieve the flexibility, isolation and multi-tenancy a hypervisor based solution is needed. As mentioned before, it comes with higher cost of performance, but this can be reduced to a negligible cost, by using para-virtualization. Para-virtualization makes only minor changes to the guest OSs, therefore greatly reducing the overhead that would exists if a full virtualization would be used.

As indicated before, there is a need of a programming abstraction. Instead of writing a user-space application on top of a commodity OS in C, which is the de-facto

programming language due to it offers high performance, we need to use a language that flexible, has a high performance while the written code can be re-used. According to researches, one of the best tool today is the Click Modular Router software [18]. Click comes with hundreds of stock elements, and can be extended with new elements, therefore we are not limited by only out of the box functionalities. Another advantage of Click that it is easy to re-use previously written elements.

Therefore a feasible solution for a lightweight, fast, reliable NFV platform should use a hypervisor based para-virtualization that runs a minimalistic OS on which Click can be integrated.

2.2 ClickOS

ClickOS meets all these requirements, moreover it is wildly in the research community. An example for that is the latest SIGCOMM conference (August 2015, in London) where they were creating VNF using Click [19], although they built its own OS named Scylla, but it is similar to ClickOS. In previous SIGCOMM conferences there were also attendants whom were using Click or implicitly the ClickOS platform [11].

From all the possible solutions ClickOS excels, due to its impressive nature. ClickOS virtual machines are small (5 MB), has a fast booting time (about 30 milliseconds) meanwhile adding the lowest delays into the networks (45 microseconds) and hundreds of them can be concurrently run on a COTS server while saturating a 10 GB pipe. Due to these reasons why I chose to use ClickOS NFV platform.

2.2.1 About ClickOS

ClickOS [21] is using a Xen Hypervisor based virtualization technology running a minimalistic OS, MiniOS, that is able to run the Click Modular Router software.

Other possibility for virtualization could be KVM technology, but performance results showed it yields lower performance than Xen. Furthermore, Xen's support for para-virtualized VMs provides the required high throughput demands with low delay. Although to achieve full potential changes needed to be made in the system. These changes concerns Xen's network I/O subsystem, the software switch (Open vSwitch), and the netback-, netfront drivers.

The original architecture of an out of the box ClickOS can be seen in Figure 11.

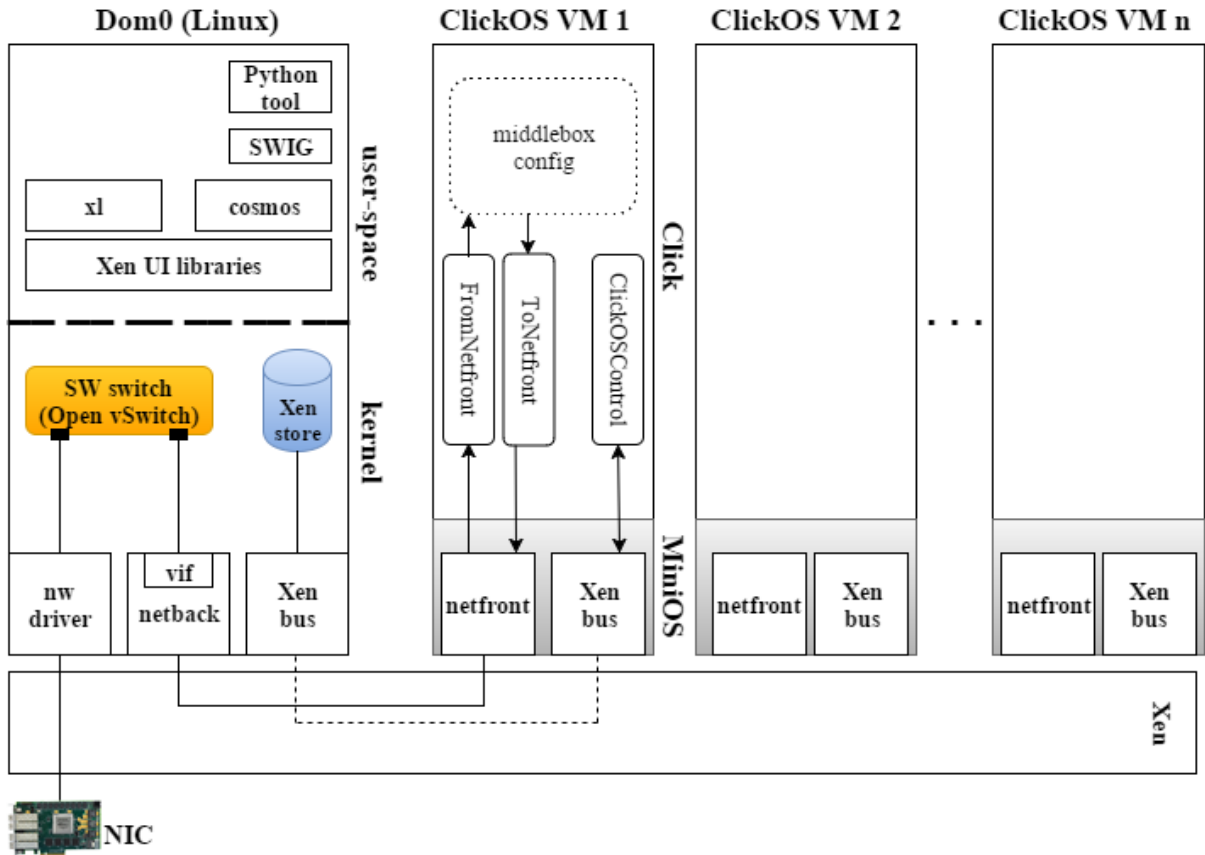


Figure 11: ClickOS architecture [21]

Xen uses a split network driver model, where the netback driver running on the host OS in kernel domain, while the netfront drivers running in each guest domain (ClickOS). They communicate to each other via shared memory (ring-based API). Meanwhile, the NIC is linked to a virtual network device, called vif, through a Linux Bridge (SW switch, or in newer Xen versions, via an Open vSwitch). When a packet is received, it is forwarded to the virtual network device named vif (it can be found in the netback driver), then it queues the packet at the netback driver. Later, one of the netback's thread picks it up and puts on the shared ring, meanwhile notifying the netfront driver.

Without any optimization, it performs poorly, therefore it needed to be changed. The first optimization was carried out at the netfront drivers, to pull for packets instead of waiting for an interrupt. Second change was that once a guest domain share information with the host domain, Xen keeps alive their shared memory, instead of reallocating it each time. Further changes has been made at the netback driver, namely it had been completely removed and a netmap-based driver has been set to operate, which allows to directly map

the network buffers of each port of the backend software switch onto the VM's local memory. This rework provides greater performance by enabling a much more direct route between the NIC and the guest domains.

There was another bottleneck in the system, namely the software switch (Open vSwitch) which significantly reduced the achievable speed. This has been also replaced with another device, namely with a VALE switch (often referenced as ClickOS Switch), which relies on a netmap driver, thus making it is easy to interact with the netfront driver. Further modification is about the port number, it has been increased from 64 to 256, to accommodate a larger number of VMs. In addition, the ring size has been altered to take 2048 slots. Moreover, the switching behaviour has been set to modular switching instead of a learning one. In Figure 12, we can see the changes illustrated.

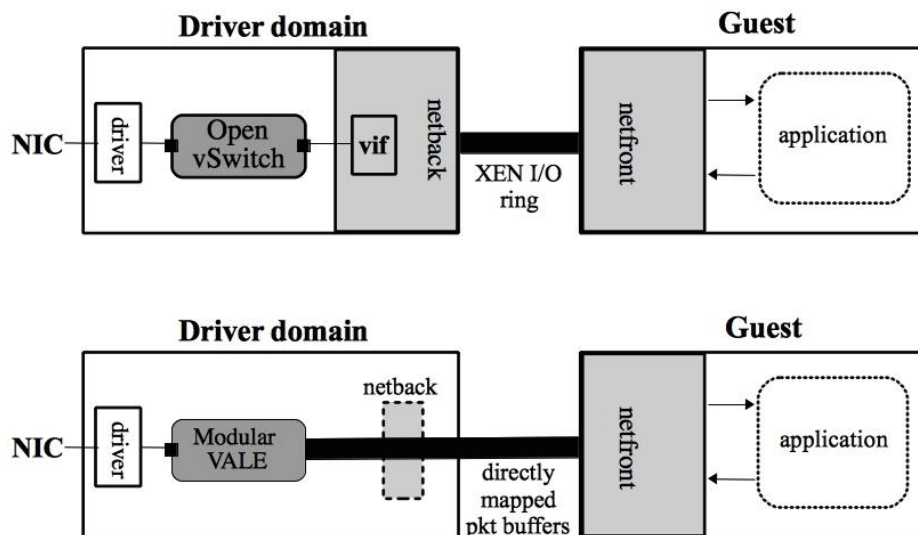


Figure 12: Standard ClickOS pipe on top,
Optimized pipe on bottom [21]

2.3 Acquiring ClickOS

Getting started with ClickOS and installing of it, is not as straightforward as it is mentioned in several publications. Even in the official webpage of the ClickOS developers [20], the only tutorial that is available for public use, has some not precise instructions. Furthermore, it isn't indicated, that those steps aim to optimize ClickOS's performance, instead of acquiring it.

The first thing on the way, is to acquire the kernel source files that matches the version of the running one. I was using the 3.16.7 Linux kernel, and this is important because if a newer kernel version would be used, then a different version of Xen would be required.

For instance, on the 3.19 Linux kernel only the 4.5 version of Xen hypervisor is accessible. But if someone would try to install Cosmos (the management system for ClickOS) with those versions, then it would throw a runtime exception saying some functions requires missing parameters, other are completely missing due to the code has been refactored between versions. For this particular problem, it has only been solved in 6. October by creating a side-branch for those who are running Xen 4.5.

The next thing to do is to get the Xen sources, preferably the 4.4 version of it. After a successful installation of the Xen hypervisor and a reboot, the running kernel's configuration and System map must be copied into the kernel source directory, which was previously downloaded.

The following step is to install SWIG to be able to connect libraries written in C or C++ with scripting languages, in this scenario with python. This is a prerequisite for ClickOS. After this, the MiniOS source files can be obtained.

Due to ClickOS redefines some function definitions that MiniOS implements, and also uses some of STL functions, there is a need to download ClickOS's toolchain, which contains their own libraries. Thereby the ClickOS sources can be configured without any error, and the ClickOS image can be created.

As for the management service, namely Cosmos, should be acquired after this step. If someone would follow those instruction on the official webpage, it would make a different outcome as it would be expected. To properly install Cosmos, it should be configured with a flag, which specifies to enable the xcl (Xen Control Light) domain library. Without using this option, it would be impossible to manage running instances of ClickOS, only by manually overwriting Xen created temporary files.

Although, as I tried to install with this option, it failed with an error message that didn't provided any information about why did that error occur. Since there was no solution to this problem, I tried to use a newer kernel version, namely the 3.19, in which case the error message I got was the following: Unknown type name.

Due to these setback, I chose to do the required management manually, because this doesn't affect the performance of the platform, only a convenience.

By all of the mentioned failures at installations, we can draw a lesson, that newer versions are incompatible with previous one. Although, even if it is possible to build the required programs, that doesn't necessarily mean that everything will be compatible with each other. I experienced it when I tried to modify the interface's name of a running ClickOS. For this to take action, an option should be enabled in the Xen configuration file. When Xen tried to parse this configuration, it throw another runtime exception, in which the message pointed toward Xen scripts, which handles virtual device creations. I managed to restrict the source of the problem to a function call. I had to modify this function to be able to execute this simple setting.

After a successful implementation of a VNF in a simulated prototype architecture, the first milestone is to deploy a VNF in a real-life environment. The first step-stone to achieve that goal, is to select a VNF Platform, which enables to do just so.

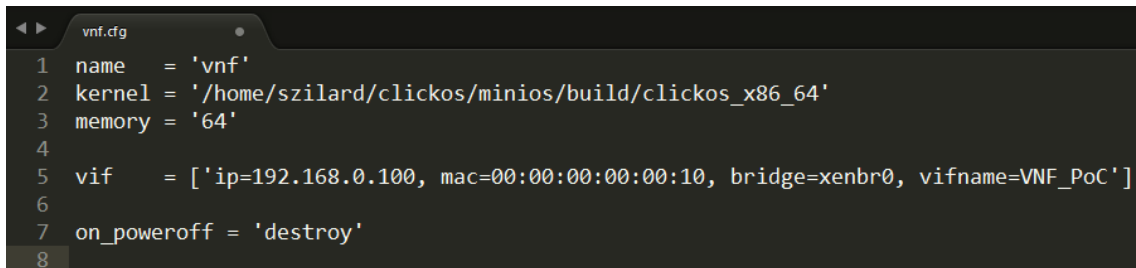
Therefore, to overcome all of these setbacks, and to be able to create a ClickOS instance is essential, because it provides a powerful tool to create any kind of VNF which can be used real time, on real, COTS hardware.

3 Creating a VNF

After the VNF platform instance has been successfully created, the next target to aim is a VNF creation on that platform, to provide a proof of concept. To create a VNF in the ClickOS platform, a click and a Xen configuration file are required.

The Xen configuration file must define some aspects of the running ClickOS like name, on which that specific instance can be accessed, full path to the ClickOS kernel image, how much memory should be allocated for the VM, and finally, the name of the virtual bridge should be specified. These are the required options, but more can be set on demand, an example can be seen on Figure 13.

With these secondary options, settings like preferred IP address, MAC address, and virtual interface name can be provided, also the click file path can be set here (which lets Cosmos to simplify the creation of the VNF). The default virtual device creation script can be also replaced with a different one to adapt to different environments.



```
vnf.cfg
1 name = 'vnf'
2 kernel = '/home/szilard/clickos/minios/build/clickos_x86_64'
3 memory = '64'
4
5 vif = ['ip=192.168.0.100, mac=00:00:00:00:00:10, bridge=xenbr0, vifname=VNF_PoC']
6
7 on_poweroff = 'destroy'
8
```

Figure 13: A minimalistic VNF Xen configuration file

Meanwhile the click file should contain the implementation of the chosen network function. This middlebox must be written in Click language. It is important to note, while in a traditional Click file, the FromDevice and ToDevice elements must be set to an interface, in this configurations it should not be initialized with anything (or only with the number zero) due to the fact, that the running instance has only one interface, and they are numbered instead of named.

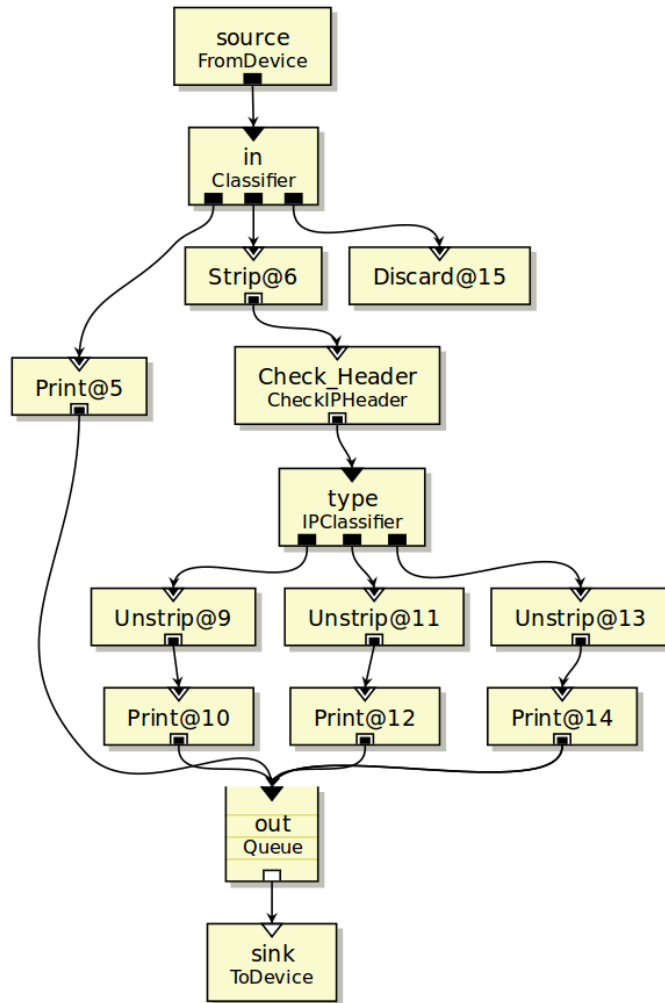


Figure 14: Click example in a graph view

An example Click configuration can be seen on Figure 14, which objective is only to demonstrate how Click works. This illustrative example reads packets from the interface, which has been assigned through the FromDevice element, and then Prints packets by their protocol.

First, the FromDevice and ToDevice elements should be observed. As mentioned before, these provides the interface to connect to NICs, therefore a configuration must always contain one of these to be able to have a dynamic behaviour. (The other possibility is to use random packet generator, which only generates dummy packets, thereby only used for debugging.)

The following element, which needs to be concerned, is the Classifier named “in”. This element has N outputs, each associated with the corresponding pattern from the

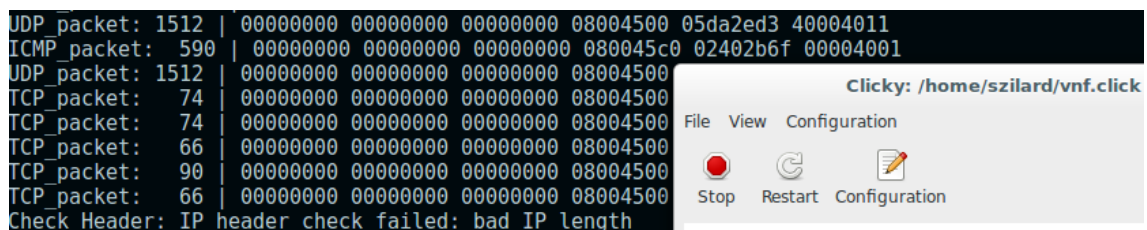
configuration string. The pattern is a set of clauses, where clause is an offset/value pair. Therefore this element should be used, if multiple type of packets can be transferred.

The IPClassifier element has a similar function, namely to filter incoming packets, but in this case, it requires each packet to start with its IP header. Due to this, the filtering can be simplified as patterns can be used like TCP, ICMP, UDP, and can also filter by source or destination address etc.

Click provides many different elements for packet verification, for which an example in this configuration is the CheckIPHeader element. This job is to only let through packets which has a valid IP header, checksum and length.

There are elements which can modify packets. For this functionality, the Strip, Unstrip elements can be shown as an example, this can strip and unstrip given bytes from the start of the packet.

The rest of the element that this peculiar configuration contains, are the Print, Discard and Queue elements. The first element write out packets contents in hexadecimal format, the second just simply throws the package, meaning it won't be processed any longer, and the last element is implementing a FIFO queue. On Figure 15, the output can be observed while the VNF runs as a user-level component on a local machine.



```
UDP_packet: 1512 | 00000000 00000000 00000000 08004500 05da2ed3 40004011
ICMP_packet: 590 | 00000000 00000000 00000000 080045c0 02402b6f 00004001
UDP_packet: 1512 | 00000000 00000000 00000000 08004500
TCP_packet: 74 | 00000000 00000000 00000000 08004500
TCP_packet: 74 | 00000000 00000000 00000000 08004500
TCP_packet: 66 | 00000000 00000000 00000000 08004500
TCP_packet: 90 | 00000000 00000000 00000000 08004500
TCP_packet: 66 | 00000000 00000000 00000000 08004500
Check Header: IP header check failed: bad IP length
```

Clicky: /home/szilard/vnf.click
File View Configuration
Stop Restart Configuration

Figure 15: The example VNF's output running on user-lever component

Now, when both the Click and Xen configuration are done, it can be deployed in a running instance of ClickOS.

In Figure 16, part of the ClickOS initializing sequence can be seen, as well as some of its property, but most importantly, the proof of the theory can be perceived,

namely that any network function can be deployed as a software. At the shutdown process (Figure 17), can be also seen that the integration of a VNF was successful.

```

Thread "xenstore": pointer: 0x2004002800, stack: 0x350000
xenbus initialised on irq 1 mfn 0x19e543
Thread "shutdown": pointer: 0x2004002fb0, stack: 0x360000
Dummy main: start info=0x262fc0
Thread "main": pointer: 0x2004003760, stack: 0x370000
sparsing 0MB at 17f000
"main"
[on_status:205] router id 0
[on_status:206] status change to Running
Thread "click": pointer: 0x2004010a00, stack: 0x390000
Failed to read /local/domain/0/backend/vif/1/0/feature-netmap.
***** NETFRONT for device/vif/0 *****

net TX ring size 256
net RX ring size 256
backend at /local/domain/0/backend/vif/1/0
mac is 00:00:20:20:20:20
*****
[router_thread:157] Starting driver...

UDP_packet: 342 | ffffffff fffffeff ffffffff 08004510 01480000 00008011
UDP_packet: 342 | ffffffff fffffeff ffffffff 08004510 01480000 00008011
UDP_packet: 342 | ffffffff fffffeff ffffffff 08004510 01480000 00008011
UDP_packet: 342 | ffffffff fffffeff ffffffff 08004510 01480000 00008011
ARP_packet: 42 | ffffffff fffffeff ffffffff 08060001 08000604 0001feff

```

Figure 16: VNF running on a ClickOS platform

```

[app_shutdown:365] Requested shutdown reason=poweroff
[router_stop:172] Stopping all routers...

[router_stop:179] Stopping instance = 0...

[router_thread:160] Stopping driver...

close network: backend at /local/domain/0/backend/vif/1/0
[router_thread:164] Master/driver stopped, closing router_thread
Thread "click" exited.
[router_stop:186] Stopped all routers...

[main:350] Shutting down...
close(0)
close(1)
close(2)
main returned 0
MiniOS will shutdown (reason = poweroff) ...

```

Figure 17: Shutdown process of ClickOS

The graph view of the VNF, which uses Network Coding, and implements a basic router functionality (can handle ARP and ICMP messages, and can forward IP packets), also encapsulates specific TCP packets into UDP packets, on which it can utilize RLNC (Figure 18).

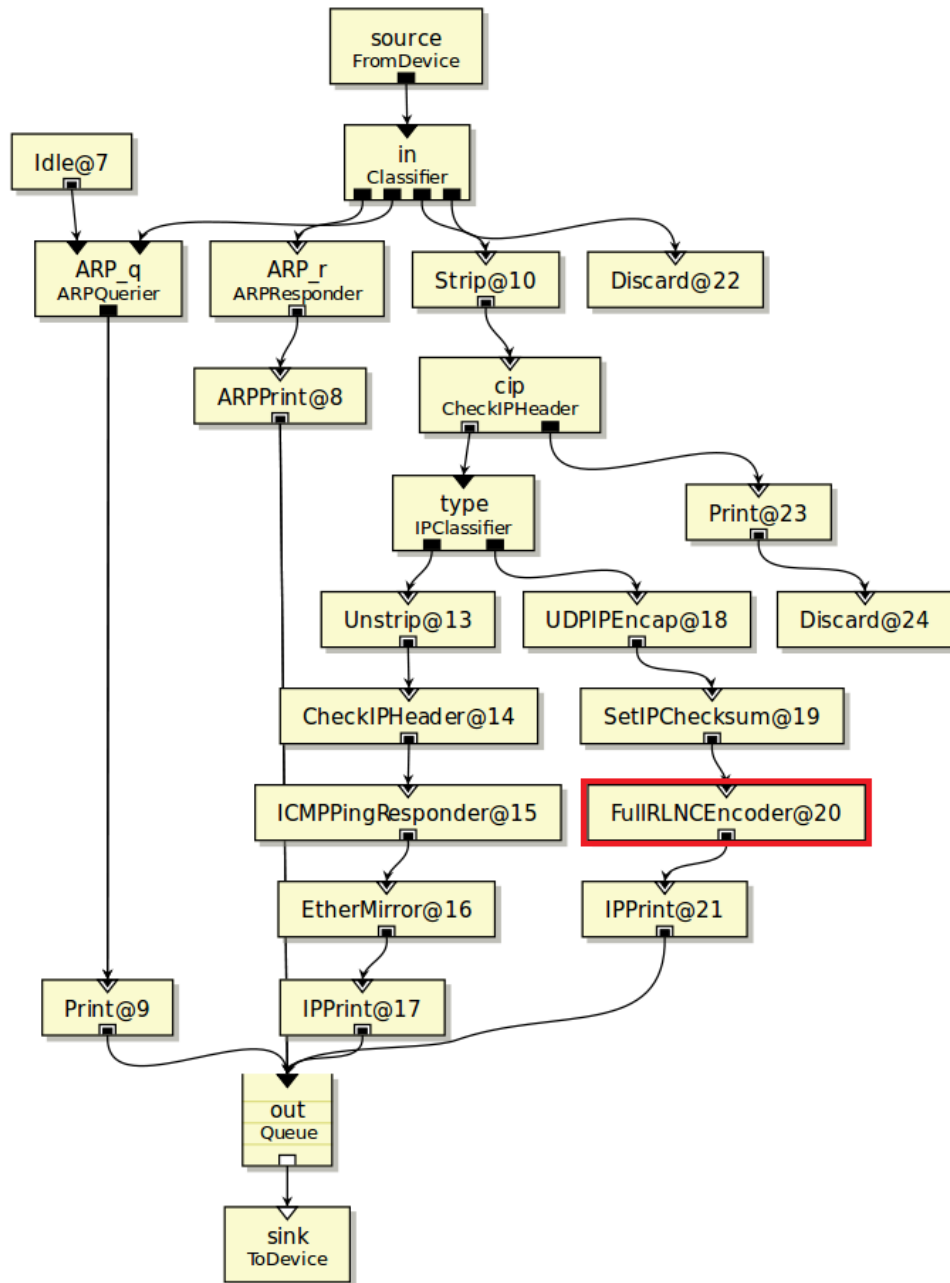


Figure 18: VNF that utilizes RLNC

4 Performance Measurements

4.1 VNF performance in ClickOS and in User-level Click

All measurement was taken on the same environment, using a Debian 8 operating system. For the user-level components, I created three network namespaces; two for the hosts, and one which was running Click. For the ClickOS analysis, I created a similar topology with two namespace for the two hosts, but instead using another namespace for Click, I used a running instance of ClickOS which contained the same middlebox implementation (Figure 19).

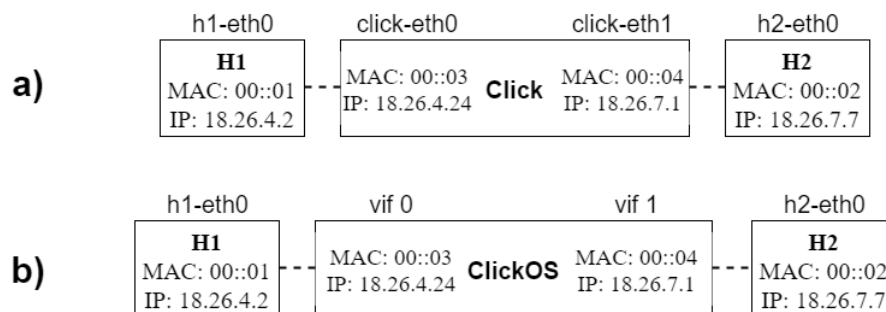


Figure 19: The user-level topology on a)
The ClickOS topology on b)

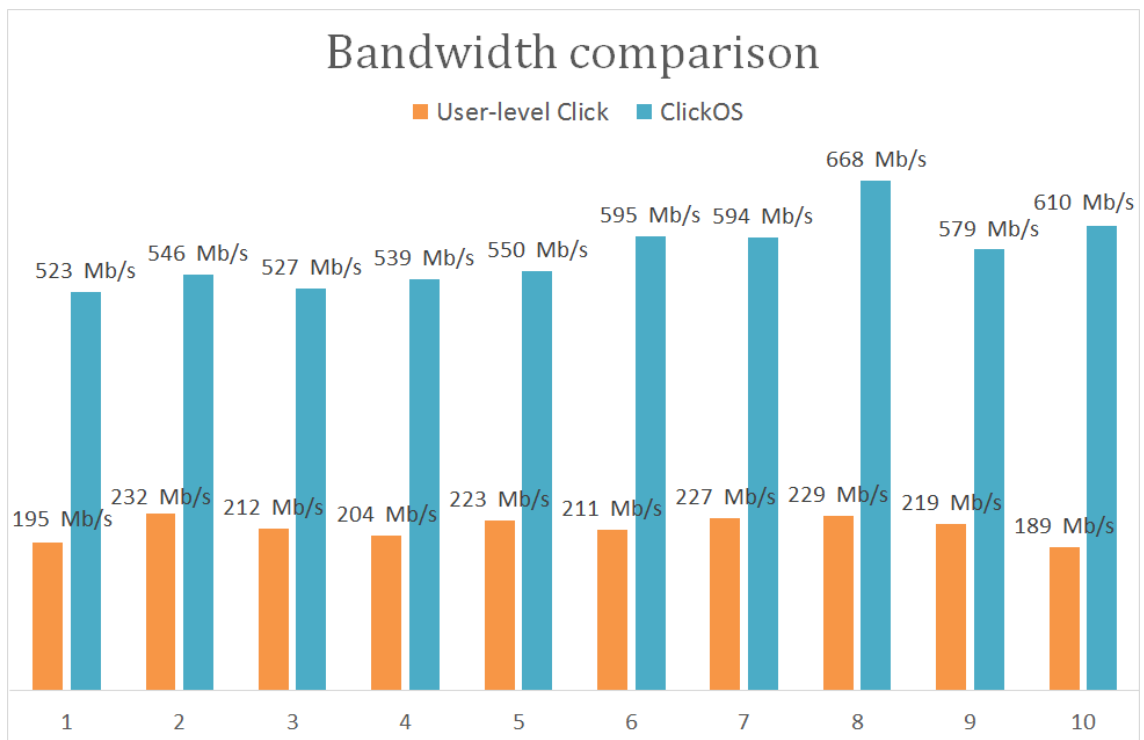


Figure 20: Bandwidth results for 10 measurements

The first aspect that I investigated was the bandwidth attribute, for which I used the *iperf* tool. In Figure 20, the results can be seen.

The difference between them significance even more, if viewed both cases's results together (Figure 21), where the maximum, minimum and average bandwidth values can be seen. This difference in bandwidth capabilities in practice can generate approximately 25 times higher packet processing.

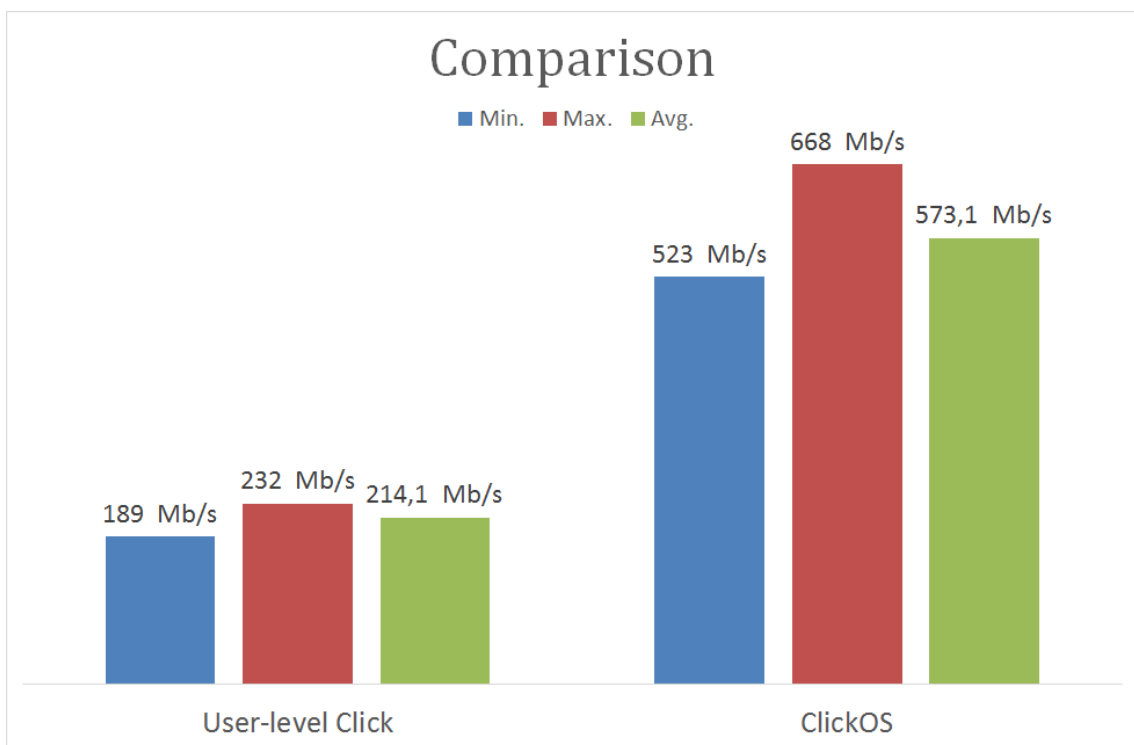


Figure 21: Comparison on minimum, maximum and average values

A further feature in which I compared this two environment is the scalability. As the results show in Figure 22, using ClickOS over a user-level Click provides a much higher performance even in a more extreme condition.

On the current VNF, I experienced with its memory usage under ClickOS, and I could narrow it down till 8 Mbyte. In comparison with a general computer running a standard operating system, which memory usage is in GB magnitudes, it fairs

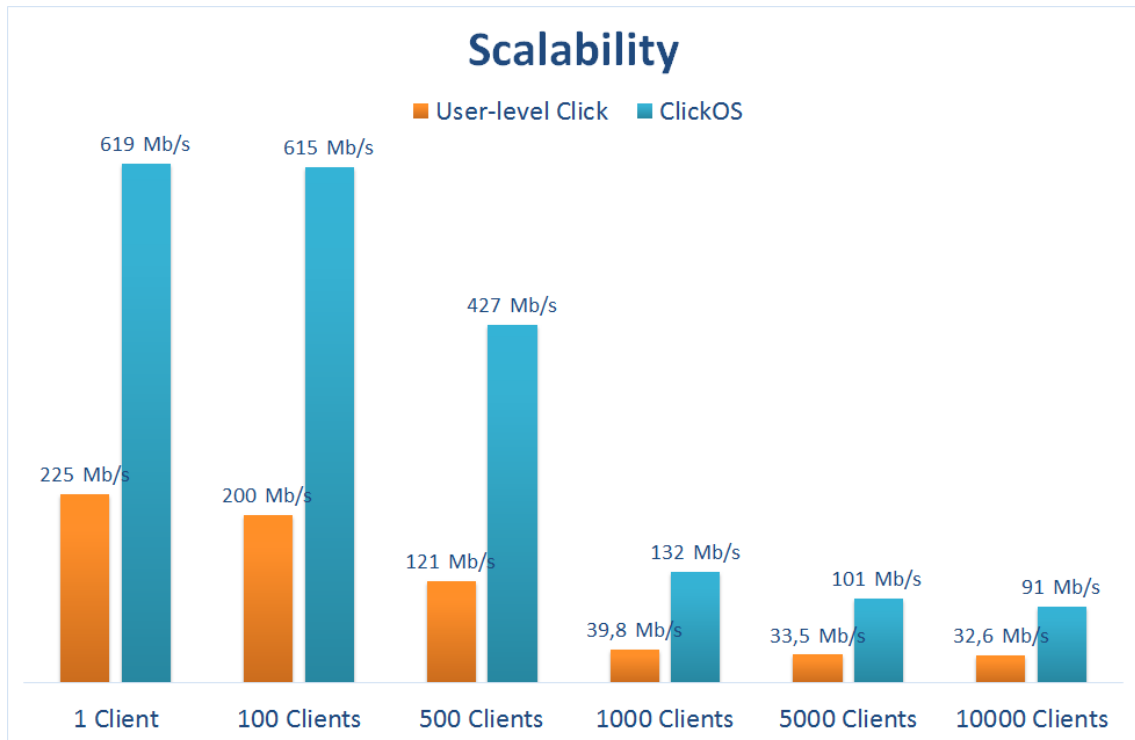


Figure 22: Scalability results

extraordinarily. Therefore, if ClickOS would be deployed on a Raspberry Pi (with 512 MB of memory for 48 USD) would mean it could run concurrently 64 different VNFs.

The next aspect that I concerned, was again a scalability quality, but this time I measured the time needed to create one thousand times a ClickOS instance, then uploading a VNF into it, and destroying it before repeating this action. To do this efficiently, I created a script which saves the time before the first instance of ClickOS is deployed, and after the last one finished. This can be seen in Figure 23, while Figure 24 shows that 1000 ClickOS has been created in the process.

```
Start time= 19:14:00
End time= 19:15:44
```

Figure 23: Time needed to create 1000 times ClickOS

```
root@debian:/home/debi/examples# xl list
Name          ID   Mem VCPUs   State   Time(s)
Domain-0      0   3796  1      r----- 116.3
clickos       2    32   1      ---sc-  0.1
root@debian:/home/debi/examples# xl list
Name          ID   Mem VCPUs   State   Time(s)
Domain-0      0   3796  1      r----- 117.3
clickos      1002  32   1      -----  4.4
root@debian:/home/debi/examples# █
```

Figure 24: Proof, that 1000 ClickOS instance has been created

The data shows, it took 94 seconds to create one thousand instance of ClickOS, therefore and average time to create one is around 94 milliseconds.

I also examined the CPU usage in both cases. First I repeated the bandwidth measures with 100 parallel clients connected to the server, but this time I was concentrating on CPU usage. The user-level Click used almost 44% of the CPU time as shown in Figure 25, while ClickOS was using approximately 42% as shown in Figure 26.

Task	CPU ▲
click -p8001 -f r_userlevel.click	43.75%

Figure 25: User-level Click CPU usage

42.34%	0.00 ns/ex	Goten
0.00%	0.00 ns/io	Blocked
0.00%	0.00 ns/ex	Waited

Figure 26: ClickOS CPU usage

Implementing RLNC as ClickOS VNF

So based on the previous results combining ClickOS VNF with Network Coding is something that really worth trying. With these technologies it is possible to create network middleboxes with small memory footprint, portability and efficient forwarding.

As a last step of my work I started to implement an RLNC VNF in ClickOS based on the other VNF I created (described in Section 3). Based on ClickOS seminal paper [21], official tutorial and my previous experience this is a quite straightforward process. The only extra step required is to copy the necessary third party libraries into a specific folder and define its name. All the rest is handled by their compile script.

However I didn't managed to do this. Even after a few weeks of debugging it seemed impossible for me. Finally I reduced the problem to the toolchain shipped with the ClickOS, created by the same authors, and it turned out that this tool is originally dedicated to mask original system headers with custom ones tailored to ClickOS. However this masking is partial and other important functions was removed including vital headers for Kodo library. I contacted with the developers of both sides (ClickOS and Kodo) and this problem was confirmed from the ClickOS team. In the current state of their project it seems the merging is require a lot of unexpected extra work that is extending ClickOS core and modifying Makefiles and source of the toolchain.

So summarize my experience with ClickOS implementing a VNF is not as straightforward as they promise if it requires to use third party libraries. Another very interesting constraint that I discovered is a 1009 byte limit for the length of the Click configuration file implementing the desired VNF (which is independent from the previously described issue).

Conclusion

Initially, I have provided an insight to SDN, NFV and Network Coding technologies, which can change the concept about how we design the network thus enable to shift current networks onto a completely new basis.

With Software Defined Networking we can make network description more abstract that leads to better optimization possibilities and also provides a more understandable view of the system. Utilizing NFV technology in this concept enables us to deploy middleboxes as software components which can be scaled, moved, upgraded and replaced on demand. Finally, Network Coding can ensure the technical solution for future requirements. It can provide throughput improvements, high degree of robustness, low delay and latency, and a more secure, reliable communication. Moreover, NC can utilize multipath routing in a seamless way.

I investigated current NFV platform implementations in order to be able to deploy any VNF, including Network Coding functionality as well and selected one of the best available tool, ClickOS. Afterwards, I went through the steps of creating a simple VNF and provided measurement results that is a proof of using ClickOS as a NFV platform for VNF implementation has way more better performance, than using the same VNF on a local machine as a user-level component. Moreover, using ClickOS provides an easier VNF management and there are no physical restrictions. In addition, it requires very small amount of memory (8 Mbyte), unlike typical full operating systems. Finally I tried to compile our existing custom Network Coding Click elements into ClickOS VNF and it turned out it is highly non trivial and also beyond the scope of this paper. As a future work it is worth to consider of using another MiniOS implementation.

List of figures

Figure 1: SDN architecture [1]	8
Figure 2: ETSI vision for NFV [5]	10
Figure 3: Simple (multipath) topology for naming conventions	11
Figure 4: Elementary butterfly topology [8].....	12
Figure 5: Overview of RLNC [10]	14
Figure 6: Illustration of E2E, HbH and RLNC coding schemes, with 50% probability loss on each link	17
Figure 7: Number of overall packets require to successfully decode the full message.....	18
Figure 8: Latency results over different channel rates and 50% probability of loss	19
Figure 9: Latency results over different channel rates, and no losses	19
Figure 10: Latency for the three transmission schemes depending on number of hops and loss (Packets 64 – Size 205 B – Bitrate 0.25 Mb/s).	20
Figure 11: ClickOS architecture [21].....	27
Figure 12: Standard ClickOS pipe on top, Optimized pipe on bottom [21]	28
Figure 13: A minimalistic VNF Xen configuration file	31
Figure 14: Click example in a graph view	32
Figure 15: The example VNF's output running on user-lever component	33
Figure 16: VNF running on a ClickOS platform	34
Figure 17: Shutdown process of ClickOS.....	34
Figure 18: VNF that utilizes RLNC.....	35
Figure 19: The user-level topology on a) The ClickOS topology on b)	36
Figure 20: Bandwidth results for 10 measurements	36
Figure 21: Comparison on minimum, maximum and average values	37

Figure 22: Scalability results	38
Figure 23: Time needed to create 1000 times ClickOS	38
Figure 24: Proof, that 1000 ClickOS instance has been created.....	38
Figure 25: User-level Click CPU usage.....	39
Figure 26: ClickOS CPU usage	39

References

- [1] K. Kirkpatrick, “Software-defined networking”, in ACM, New York, NY, USA, 2013 Available: <http://dl.acm.org/citation.cfm?id=2500473>
- [2] “ONF Overview - Open Network Foundation”, Open Network Foundation, [Online]. Available: <https://www.opennetworking.org/about/onf-overview>, [Accessed 23 October 2015].
- [3] „Network Functions Virtualisation”, in SDN and OpenFlow World Congress, Darmstadt-Germany, 2012. Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf
- [4] „ETSI - NFV”, ETSI, [Online]. Available: <http://www.etsi.org/technologies-clusters/technologies/nfv>. [Accessed: 23 October 2015].
- [5] F. Yue, “Network Functions Virtualization – Everything Old Is New Again”, 2013. Available: <https://f5.com/Portals/1/Cache/Pdfs/2421/network-functions-virtualization--everything-old-is-new-again.pdf>
- [6] C. Fragouli, J.-Y. Le Boudec and J. Widmer. “Network Coding: An Instant Primer”, in ACM Sigcomm Computer Communication Review, vol. 36, num. 1, p. 63-68, 2006. Available: <http://infoscience.epfl.ch/record/58339/files/rt.pdf>
- [7] R. Ahlswede, N. Cai, S.-Y. R. Li and R. W. Yeung, “Network Information Flow”, in IEEE Transactions on information theory, 2000. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=850663
- [8] Linear network coding [Online] Available: https://upload.wikimedia.org/wikipedia/commons/thumb/3/33/Butterfly_network.svg/2000px-Butterfly_network.svg.png [Accessed 23 October 2015].
- [9] “Frequently Asked Questions – Kodo master documentation” Steinwurf [Online] Available: <http://kodo-docs.steinwurf.com/en/latest/faq.html> [Accessed 23 October 2015].
- [10] “Introduction to Network Coding” Steinwurf [Online] Available: http://kodo-docs.steinwurf.com/en/latest/nc_intro.html [Accessed 23 October 2015].
- [11] J. Martins, M. Ahmed, C. Raiciu és F. Huici, „Enabling Fast, Dynamic Network Processing with ClickOS,” 2013. Available: <http://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p67.pdf>
- [12] Alcatel Lucent, „Why Service Providers Need an NFV Platform,” 2013. Available: https://networkbuilders.intel.com/docs/NP2013113597EN_NFV_Platform_StrawhitePaper.pdf
- [13] Alcatel Lucent, redhat, „CloudBand with OpenStack as NFV platform ,” 2013. Available:

<http://www.tmcnet.com/tmc/whitepapers/documents/whitepapers/2014/10694-cloudband-with-openstack-as-nfv-platform.pdf>

- [14] „Technical Overview | Open Platform For NFV”, OPNFV, [Online]. Available: <https://www.opnfv.org/software/technical-overview>. [Accessed: 23 October 2015].
- [15] HP, „Network functions virtualization,” 2015. Available: <http://www8.hp.com/h20195/v2/GetDocument.aspx?docname=4AA5-1114ENW>
- [16] VMware, „VMware vCloud NFV,” 2015. Available: <https://www.vmware.com/files/pdf/solutions/vmware-vcloud-nfv-datasheet.pdf>
- [17] „Dell Introduces Network Function Virtualization Platform and Starter Kits to Accelerate Carrier Trials and Applications | Dell”, Dell, [Online]. Available: <http://www.dell.com/learn/us/en/uscorp1/press-releases/2014-10-14-dell-software-open-networking-network-functions-virtualization> [Accessed: 23 October 2015].
- [18] E. Kohler, R. Morris, B. Chen, J. Jannotti and K. M. Frans, “The Click Modular Router,” ACM Transactions on Computer Systems (TOCS), 2000. Available: <http://dl.acm.org/citation.cfm?id=354874>
- [19] R. Riggio, J. Schulz-Zander and A. Bradai, “Virtual Network Function Orchestration with Scylla,” 2015. Available: <http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p375.pdf>
- [20] „Cloud Networking Performance Lab | ClickOS | Modular VALE | XEN”, CNP Lab, [Online]. Available: <http://cnp.neclab.eu/getting-started>, [Accessed: 23 October 2015].
- [21] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco and F. Huici, “ClickOS and the Art of Network Function Virtualization,” in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14), 2014. Available: <https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-martins.pdf>
- [22] F. H. P. Fitzek and G. Fettweis, “Holistic View on 5G,” in IT Gipfel: Fokusgruppe 5G, Berlin, Germany, 2015. Available: <https://www.aut.bme.hu/Upload/Pages/Events/5GCommunication/Holistic.pdf>
- [23] M. V. Pedersen, D. Lucani and F. H. P. Fitzek, “Network Coding: Theory and Implementation,” in *European Wireless 2014*, Barcelona, Spain, 2014
- [24] D. Szabó, A. Csoma, P. Megyesi, A. Gulyás, F. H. P. Fitzek “Network Coding as a Software Defined Networking Service” in *European Wireless*, Budapest, 2015