Network Coded Software Defined Networking: Design and Implementation

Jeppe Krigslund¹, Jonas Hansen¹, Daniel E. Lucani¹, Frank H.P. Fitzek¹, Muriel Médard²

¹Department of Electronic Systems, Aalborg University, Denmark
²RLE - Massachusetts Institute of Technology, USA

Email: krigslund@steinwurf.com, {jh, del, ff}@es.aau.dk, medard@mit.edu

Abstract—Software Defined Networking (SDN) and Network Coding (NC) are two key concepts in networking that have garnered much attention in recent years. On the one hand, SDN’s potential to virtualize services in the Internet allows a large flexibility not only for routing data, but also to manage buffering, scheduling, and processing over the network. On the other hand, NC has shown great potential for increasing robustness and performance when deployed on intermediate nodes in the network. This new paradigm changes the dynamics of network protocols, requiring new designs that exploit its potential. This paper advocates for the use of SDN to bring about future Internet and 5G network services by incorporating NC functionalities. The inherent flexibility of both SDN and NC provides a fertile ground to envision more efficient, robust, and secure networking designs, that may also incorporate content caching and storage, all of which are key challenges of the future Internet and the upcoming 5G networks. This paper proposes some of the keys behind this intersection and supports it with use cases as well as an implementation that integrated the Kodo library (NC) into OpenFlow (SDN). Our results on single-hop, multi-hop, and multi-path scenarios show that gains of 3x to 11x are attainable over standard TCP and Multi-Path TCP.

I. INTRODUCTION

Current communication networks have been designed using a layered structure with the goal to allow for new technologies, algorithms, and protocols to be supported with minimal or no change to the other layers. This has been particularly successful for higher layers, dealing with applications and services, and for lower layers, dealing with constantly evolving access technologies. However, the intermediate network layer protocols (IP layer) have seen little change for decades. This stagnation has come in part from the fact that (i) multiple technologies already depend on this IP layer to operate [1], and (ii) the challenges to configure and manage networks beyond simple routing operations. The downside to current network protocols is that they were designed with a narrow set of goals determined several decades ago. This is currently limiting the effectiveness and feasibility of more complex and resource demanding applications.

In order to address these challenges, networking theory and practice is currently considering two new concepts, namely, SDN and NC, which can deliver higher performance and more flexible networking designs. SDN enables the network to provide a flexible and on-the-fly allocation of resources, including buffer management, dynamic routing, exploitation of multiple paths, by considering a more holistic view of network operation and even incorporating knowledge from various layers in the network stack. SDN virtualizes services in the network by separating the data transmission and control of the network in order to achieve these benefits. Although early designs considered the control plane to be implemented with additional controller devices, recent work has envisioned more refined and distributed control mechanisms as well as incorporating security considerations for SDN [2]. Finally, practical implementations, e.g., OpenFlow [1], are available for designing new systems and protocols with SDN functionalities.

NC [3] constitutes a new paradigm for networking that breaks from the store-and-forward approach currently used in communications networks. NC allows intermediate nodes in the network to recode incoming data packets, thus providing a store-code-forward paradigm. This ability to code within the network contrasts with end-to-end erasure correcting codes, e.g., Reed-Solomon, LT codes, Raptor codes, and allows NC-enabled networks to generate redundant packets where they are needed instead of injecting them end-to-end as other erasure codes would do. NC gains in terms of throughput and robustness come from the fact that (i) the network itself only needs to deliver linear combinations of the original data packets, thus providing a richer set of possible actions within network; and (ii) senders and receivers no longer need to track individual packets but rather focus on gathering enough independent linear combinations to decode. Gains in practice have been shown in a variety of implementations, e.g., [4], [5], [6]. Although part of the challenge to deploy NC lies in the difficulty to change or retrofit routers in the network, enabling even a limited number of devices can have a large impact in performance if we are able to identify them and shape the routing and encoding decisions based on this knowledge. The holistic view of the network used by SDN can ease this process and enable other functionalities. However, the work on both concepts has been in isolation to this date.

Our goal is to bring together these two concepts through a fundamental analysis and understanding of their joint potential and demonstrating their combination with a real implementation using OpenFlow [1] and the dodo network coding library [7]. We leverage key examples to carry out our demonstrations. Our implementation uses NC’s recoding capabilities to mask packet losses in the network on single and multi-path transmissions. We compare our results to standard TCP and Multi-Path TCP (MPTCP) [8] and show that our simple approach provides three fold to eleven fold gains even with moderate losses.

II. MOTIVATION

The combination of SDN and NC brings forth an interesting potential for the management and operation of future Internet...
Acknowledgement (ACK) is used to ensure delivery of every Linear Network Coding (RLNC) coding scheme as Forward following benefits are possible by this combination:

- **Exploitation of multiple communication paths**: NC is particularly well suited to exploit multiple communication interfaces and routes [9], which can then exploit SDN’s ability to recognize multiple communication paths between source and destination. This is key in 5G networks to comply with reliability requirements as well as an appropriate management of heterogeneous interfaces, e.g., mmWave for increased speed and another technology for continuous connectivity or for M2M purposes.

- **Management of data storage and caching**: SDN’s capabilities to virtualize and/or identify caching and storage nodes in the network are key to exploit NC for enhancing the impact and reducing the storage cost of caching/storage by relying on linearly coded packets instead of replication of the original data per storage location/device. NC also provides a single code for both storage and data transmission, which is key to treating data as a single holistic process. This management can also include the pre-emptive caching of data of a user as it moves through the network using location information. The goal is to guarantee low latency for access to the user’s data.

- **Adaptation of redundancy based on link quality**: SDN provides simple mechanisms to identify the link quality, including packet losses, for the transmission routes used for a flow. This capability is particularly relevant for using network coding’s recoding capabilities to generate the right level of redundancy per link, instead of introducing redundancy end-to-end to compensate for packet losses which is an inherently inefficient strategy.

- **Assessment of system load and complexity allocation**: SDN is useful for identifying whether a device can commit resources to recoding and how many. This may allow us to choose the network coding parameters to meet the current network demands. This is particularly relevant with novel NC schemes that provide a fluid allocation of complexity, e.g., Fulcrum network codes [10], by performing linear combinations using different finite fields end-to-end and at different nodes in the network. The choice of the finite field has a direct impact on the computational effort required by a given node. This added flexibility is key to dealing with energy efficiency in 5G networks, not only for the infrastructure but for connected machines (e.g., sensors, actuators) and end-user devices.

### III. Coding Approach

The software implementation uses a systematic Random Linear Network Coding (RLNC) coding scheme as Forward Error Correction (FEC), i.e., neither positive nor negative Acknowledgement (ACK) is used to ensure delivery of every packet. However, other coding schemes are supported.

#### A. Random Linear Network Coding

In order to use RLNC, the software implementation uses Kodo [7] which is a C++11 network coding library. The encoder uses a systematic code [11] where all the original packets are transmitted uncoded (but with a header added by Kodo and zero-padded to the RLNC symbol size) the first time. Throughout the performance measurements, we use a generation size of 10 symbols. The Galois field in use is $\text{GF}(2^{256})$, and the symbol size is set to 1,356 bytes, Maximum Transmission Unit (MTU) + 2 bytes for zero-padding length + 4 bytes for packet ID. Some of the benefits of a systematic code is lower decoding complexity and lower decoding delay [11]. Lower decoding delay is achievable since all uncoded packets can be forwarded directly and having uncoded packets reduces the decoding complexity of Gaussian elimination, which is used to decode the RLNC generation. Normally, RLNC is considered a block code. However, Kodo supports RLNC on-the-fly where packets can be added to the encoder on-the-fly, i.e., as they arrive. Similarly, packets can be extracted from the decoder as soon as they are decoded, without the need of decoding the entire generation first. One of the benefits of RLNC on-the-fly is that coded packets can be transmitted before the entire generation is fed to the encoder, i.e., adding redundancy on-the-fly. The combination of systematic and on-the-fly coding allows the encoder to transmit uncoded and coded packets with minimal delay, which is critical for control packets in Internet communication. Of course, using on-the-fly coding trades off improved delay performance for a reduction in decoding probability. The reason lies in the fact that coded packets have more potential for correcting a loss when more original packets are involved in the linear combination. Thus, coding on-the-fly reduces the strength of the code.

Fig. 1 shows an example of a transmission from one node to another. In the example, the link has a packet loss probability of 33 %, and the encoder adds 50 % redundancy to compensate for said losses. Since the encoder uses RLNC on-the-fly the first coded packet is transmitted as the third packet. P1 can be forwarded directly, since the first packet is received and is uncoded. The second packet is lost, but P2 can be extracted from the third packet, the first coded (redundancy) packet. The fourth packet, containing an uncoded P3, is lost. P4 can be forwarded after the fifth packet is received. When the sixth packet is received, P3 can be extracted and forwarded, this causes out-of-order delivery of P3 and P4. Packets 7 – 9 are received and therefore P5 and P6 are forwarded. The tenth packet is lost. The eleventh packet is received, and P8 is forwarded. P7 is unrecoverable if no further packets are sent from the generation, since the twelfth packet is lost and the implementation is a FEC code. The example shows that most of the packets can be received after a minimal amount of decoding, thereby also minimising the decoding delay. Instead of an packet loss probability of 33 % (on the physical layer) the end-node will only experience a packet loss probability of 12.5 %. The cost of this improvement in packet loss probability is clearly in the introduction of redundant packets, e.g., packet 9 which does not add any new information by the time it is received. Fig. 2 shows the TUN stack used to implement the coding scheme. A TUN (namely network
TUNnel is meant to simulate a network layer device.

1) RLNC Recorder: Throughout the network, several intermediate nodes may be used to forward the packets. This can be used to coordinate multiple paths and/or utilise RLNC recoding. These nodes do not include a coding interface and therefore not a TUN stack. These nodes require two UDP stacks to receive and transmit packets on each link. Fig. 3 depicts a UDP stack at an intermediate node, which recodes the RLNC packets it receives. When one stack receives a packet, it will recode at least one (more if required by the redundancy amount) new packet, and forward it to the other UDP stack which transmits it to the next-hop. The recoding process is only performed in the stack that read the packet from the socket, i.e., packets are not recoded in both stacks.

2) Packet structure: All RLNC packets contain five header fields. The first four fields always provide the same type of information: symbol size, generation size, generation ID, and encoder rank. The receiver must know the generation size and symbol size before creating a decoder. The generation ID is used to keep track of the different generations at the decoder. When an encoder has transmitted an entire generation, including redundancy packets, it prepares the next generation, i.e., releasing memory allocated for the completed generation and incrementing the generation ID. When the encoder rank and decoder rank are equal, they specify the amount of fully decoded symbols. In order to cope with packet reordering, the decoder keeps track of the maximum encoder rank it has received. The last header field is either the coding vector used to create the symbol or the index of the uncoded symbol. The index has a constant size of 4 bytes, while the size of the coding vector is dependent on both the Galois field in use and the generation size.

**Fig. 1:** Example of a coded transmission between two nodes.

**Fig. 2:** TUN stack of an RLNC coder setup.

**Fig. 3:** One of the two UDP stacks in a recoder setup.

### IV. SOFTWARE IMPLEMENTATION

The end-nodes are equipped with a coding interface that resembles a normal network interface. These coding interfaces are TUN interfaces\(^1\), which allows any application to use the benefits of the coding seamlessly. The MTU of the TUN interfaces is set to 1350 bytes, in order to keep it well below the MTU of the regular network interface (\(~1500\) bytes) to avoid packets being split by the IP-layer. Fig. 4 shows a model of the system. The MAC and PHY layers are not shown since the end-nodes may be connected using several intermediate nodes. The coding layers form a tunnel between them and exchange packets using UDP. The TUN interface is given an IP-address and a subnet mask. When an application opens a socket and transmit packets to this subnet, the Operating System (OS) will forward the packets to the coding layer. The coding layer will then process the packets and, depending on the setup, transmit them through the regular network interface to the other end of the tunnel. On the opposite end, the packets are received by the node on the regular network interface and then forwarded to the coding layer. When the coding layer has processed the packet, it is written to the TUN interface after which the OS will determine where to forward it. This could be e.g. to an application through a socket, or to an internal process such as ping, where packets are not forwarded, but replied with a pong packet. The software is available at GitHub\(^2\).

When a packet leaves the lowest IP layer and transmitted towards the other end of the tunnel, it has been prepended with several headers. The two outer most headers are the headers added to allow packets to be transferred between the coding layers on the two nodes, i.e., UDP (e.g. on port 1025) over

---

\(^1\)kernel.org/doc/Documentation/networking/tuntap.txt

\(^2\)github.com/14gr1010/software
The developed software and the virtual network environment in which the software is deployed indicates that an integration of network coding as a functionality of a software defined network is indeed possible. However, this alone does not show that this combination of technologies is actually beneficial. In order to show that the proposed integration of network coding is plausible, a series of performance measurements has been carried out using the standard tool for network performance measurement, iperf. As a secondary result, this should also show that the coding approach can be applied without breaking functionality with the conventional TCP/IP network protocol stack.

**A. Performance References**

We compare most of our performance results with two references: the theoretical performance of TCP (both coded and uncoded), and the maximum capacity of the network. This comparison is done to further clarify the gains and to show how much there is left to gain, using some other approach.

1) **Theoretical Performance of TCP**: In [12] the authors derived a theoretical model for a mean field approximation for the performance of uncoded TCP, $T_{TCP}(W_{max}, RTT, e)$. The three inputs for said model are: the maximum window size ($W_{max}$), the round trip time ($RTT$), and the packet loss probability ($e$). Since in all scenarios the $RTT$ is a fixed parameter it is possible to estimate $W_{max}$ using the throughput performance measurements of uncoded TCP:

$$T_{TCP} = W_{max}/RTT \Rightarrow W_{max} = \frac{T_{TCP}}{RTT}.$$  \hfill (1)

We fitted the model for uncoded TCP to our coding approach by transforming two of the input variables, namely $W_{max} \rightarrow W_{coded}$ and $e \rightarrow e'$. Transforming the window size is done by scaling $W_{max}$ with the redundancy factor, i.e., $W_{coded} = W_{max}/R$, where $R = 1.2$ for RLNC +20%. The transformation of the packet loss probability was carried out using a Monte Carlo simulation of redundancy levels and packet loss probabilities. The result of this simulation was curve-fitted using a third degree polynomial function $f(e) = e0$ where $r^2 > 0.9999$. The theoretical performance of TCP using our coding approach is then given by

$$T_{coded}(W_{max}, RTT, e, R) = T_{TCP}(W_{max}/R, RTT, f(e)).$$  \hfill (2)

On Fig. 6b the theoretical performance is marked by the thin dashed lines.

2) **Maximum Capacity**: When assuming the packet losses on the link are individual Bernoulli trials then the maximum capacity is a linear function of the packet loss probability. The maximum is given as: $C_{max}(e) = C_{max}(0) \cdot (1 - e)$, where $C_{max}(0)$ is the maximum capacity without errors, i.e., the link speed, and $e \in [0, 1]$ is the packet loss probability. On Fig. 6b the maximum capacity is marked by the thick dashed line.

**B. Single-Hop**

The single-hop scenario consists of two virtual nodes, each connected to individual virtual switches. These switches are then connected with a virtual ethernet connection, on which delay and loss is introduced. This simple topology along with specified delay and packet loss is depicted in Fig. 6a. By isolating the coding approach to a single link between coders the consequence of the data recovery process within the developed coding approach is revealed. Despite the potential ability to recover every single erasure that may occur on the link, the performance of the transported TCP communication may not resemble error-free TCP communication. This is due to the inevitable delay of the error recovery phase, from when a lost packet should have been received to the point where it is successfully decoded. The amount of additional interference in terms of delay and jitter is reduced to a bare minimum in this single-hop scenario. The channel conditions on the
investigated link is then adjusted to illustrate the tolerance of both the TCP communication and the deployed coding. Increasing packet loss on the link reveals the robustness added by the coding. The bandwidth of an uncoded TCP connection is compared to TCP connections carried by the deployed coding approach using systematic on-the-fly RLNC. The achievable throughput for both uncoded and coded data flows is stated in Fig. 6b. This is compared with the maximum channel capacity and the theoretical maximum throughput. The throughput is measured with four different redundancy levels: 0 %, 20 %, 40 % and 100 %. Using additional redundancy levels may yield a slightly finer grained curve, however it would also require the system to be able to estimate the loss probability with an accuracy of approximately 1 – 2 %. From the performance of the coded data flow a gain of \( 3x \) is obtained already at 0.5 % packet loss probability. This performance boost increases up to \( 9x \) at 10 % packet loss probability. Furthermore, the obtained performance of the coded flow follows a trend similar to the theoretical maximum coded throughput, showing coherence between theory and practice.

C. Multi–Hop

In order to illustrate the benefits of recoding, we utilize a multi-hop setup where both link experience different loss probability and link delay. This network setup is presented in Fig. 7a. This network scenario is included to imitate mesh-like network structures such as dedicated sensor networks, mobile ad-hoc networks and vehicular communication networks. While uncommon in consumer oriented networks, such networks are expected to gain popularity in the future. The setup illustrates the necessity of intermediate coding (recoding) when all links are prone to erasures. While prior research efforts have already drawn this conclusion, the setup tests the validity of this with a simple feedback-less coding approach. Furthermore, recoding may introduce channel irregularities such as packet reordering and additional delay and jitter. By running TCP on top of a recoded data flow, these recoding issues are tested in practice. Fig. 7b reveals the strength of recoding. Apart from the up to \( 11x \) gains over uncoded TCP, the recoding also reduces congestion on the first link \( a \rightarrow b \) and introduces a higher achievable throughput, compared to the end-to-end RLNC coding approaches.

D. Multi–Path

The final investigated network scenario is a multi–path setup, where multiple data paths span between nodes. The conventional methods for communication in such scenarios is choosing the best of the available paths. This is naturally the simplest approach, and while the chosen data path provides adequate capacity it is probably also the best approach. However, some communication scenarios may benefit from utilising several of the available data paths. MPTCP has been developed for such scenarios, but suffers from similar behaviour towards packet loss and link delay as that of conventional TCP. Using a combination of network coding and SDN to accommodate packet loss on each individual path may provide similar benefits to MPTCP as with conventional TCP in the single-hop and recoding scenarios. Fig. 8a illustrates the multi–path scenario investigated. This consists of one direct single-hop path (path 2) and two indirect paths with an additional hop. The two multi-hop paths \( a \rightarrow b_1 \rightarrow c \) and \( a \rightarrow b_2 \rightarrow c \) are denoted path 1 and path 3, respectively.

In the multi–path scenario, the achievable throughput of uncoded MPTCP is compared to that of a coded approach, where RLNC is utilized to protect each MPTCP subflow individually. The resulting performance is illustrated in Fig. 8b. This also includes a theoretical upper bound for the performance of MPTCP [13] along with a coded conventional TCP flow, carried over path 2. Due to the high sensitivity towards packet loss and link delay, even the theoretical upper bound for MPTCP indicates poor performance in the multi–path network, and even the single-path coding approach outperforms MPTCP even though only a third of the total capacity is available to this approach. The multi–path coding provides a performance increase of up to \( 2.5x \) over the theoretical MPTCP upper bound and \( 4.5x \) over the obtained MPTCP performance.

VII. CONCLUSIONS

This paper brought together two relevant networking concepts, namely, network coding and software-defined networking, to show their joint potential to operate current and future networks more efficiently, with higher resiliency, providing higher throughput, and allowing control of data location to enable low latency services. Beyond discussing concepts, we showed that the essential software packages from each concept, namely, OpenFlow and Kodo, are already available and can be integrated to provide the required functionalities to current and future networks. We described the overall architecture for a joint implementation of these two concepts and demonstrated their potential in three key topologies and compared the performance of standard TCP and MPTCP with and without the use of SDN-NC strategies to stabilise network
Fig. 7: Multi-hop network coding - Recoding potential of network coding

Fig. 8: Multi-path network coding - Subflow coding

links. That is, we provided a use of SDN and NC that would be transparent to end-to-end protocols as a first step to understand the potential and as a way to show the benefits that even subsets of SDN and NC capable nodes could bring to the end-to-end performance of the system. We showed that gains of 3x to 11x are attainable with our simple approach. Our future work shall focus on real deployments with SDN–capable switches and high–end desktops to carry out more realistic measurements. For this, we have recently built a testbed with 16 programmable high-performance network nodes, including a FPGA programmable 10 Gbps PCI-Express network card, and one real SDN–enabled switch from NEC.

ACKNOWLEDGMENT
This work was partially financed by the Green Mobile Cloud project (Grant No. DFF - 0602-01372B) granted by the Danish Council for Independent Research and by the VELUX Visiting Professor Programme 2013–2014 granted by the VELUX Foundation.

REFERENCES