Energy-Aware Hardware Implementation of Network Coding

Georgios Angelopoulos, Muriel Médard, and Anantha P. Chandrakasan

Massachusetts Institute of Technology,
Mass. Av. 77, 02139 Cambridge, US
{georgios,medard,anantha}@mit.edu

Abstract. In the last few years, Network Coding (NC) has been shown to provide several advantages, both in theory and in practice. However, its applicability to battery-operated systems under strict power constraints has not been proven yet, since most implementations are based on high-end CPUs and GPUs. This work represents the first effort to bridge NC theory with real-world, low-power applications. In this paper, we provide a detailed analysis on the energy consumption of NC, based on VLSI design measurements, and an approach for specifying optimal algorithmic parameters, such as field size, minimizing the required energy for both transmission and coding of data. Our custom, energy-aware NC accelerator proves the feasibility of incorporating NC into modern, low-power systems; the proposed architecture achieves a coding throughput of 80MB/s (60MB/s), while consuming 22uW (12.5mW) for the encoding (decoding) process.

Keywords: Network Coding VLSI Implementation, Energy Optimization of Network Coding, Network Coding for Mobile Applications

1 Introduction

Network Coding (NC), initially introduced in 2000 by Ahlswede et al. [2], has received extensive research attention in the Information Theory and Networking community. The revolutionary idea of NC is to allow intermediate nodes within a network to mix or code previously received or locally generated packets together and let the final destinations decode the mixtures. Some of the reported advantages of NC are throughput gains, increase in data robustness, security and better utilization of network resources [7, 9, 4]. However, coding complexity and energy cost should be carefully examined in order to make these advantages practical, especially when low-power applications are considered, such as Body Area Networks (BANs).

In the literature, a few papers deal with implementation issues of NC, and almost all of them use high-performance CPUs or GPUs, focusing on the maximum achievable throughput, without analyzing the energy trade-offs of incorporating NC into a system architecture. For instance, in [14] a 3.6 GHz Xeon Dual-Core processor is used to perform NC, achieving a coding throughput of approximately 5MB/s, while, for similar settings, in [3] NC is implemented using a 800MHz Celeron CPU, achieving a throughput of 44MB/s. In addition, in [5] a special type of systematic NC over GF(2) is implemented, both on a cell phone and a laptop, achieving maximum reported throughput of 40MB/s and 1.5GB/s, respectively. However, the authors do not consider the energy analysis neither of the coding process, nor of the implications that the specific algorithmic parameters may have in the total system's energy; for instance, a possible increase in packet retransmissions due to linear dependent packets. Authors in [8, 12, 10, 13] make use of multi-core CPUs and GPUs to speed up both encoding and decoding of NC. While remarkable effort is required to achieve this coding performance, the power budget of these approaches is in the order of 100 to 500W, number which is generally prohibitive for low-power systems. Finally, an iPhone is used as the implementation device in reference [11] , where a maximum throughput of 420 KB/s is reported, while NC is responsible for approximately 33% of the reduction in the total battery life-time.

Although high-end CPUs and GPUs have been successfully used as platforms for implementing NC in previous works, as more and more mobile devices, sensors and other battery-operated systems are used, the need of a highly efficient and optimized implementation of NC is required. To the best of our knowledge, none of prior works deal with the energy analysis of a custom VLSI implementation of NC. In this paper, we provide a detailed analysis for incorporating NC into a low-power system architecture, giving insight on the trade-offs between algorithmic complexity, energy consumption and coding performance. We also propose an architecture of a custom accelerator, capable of performing NC while consuming tiny amounts of power, dictated by the strict energy constraints of modern low-power systems. Some possible target applications of our analysis and design may be low-power wireless sensor and mobile networks. Our energy-scalable, ultra low-power NC accelerator consumes 22uW (12.5mW), achieving a coding throughput of 80MB/s (60MB/s), for the encoding (decoding) process, showing that NC can be incorporated successfully into modern low-power systems.

This work is organized as follows. In Section 2, we briefly describe the encoding and decoding procedure of NC, while in Section 3, we cover in more detail some of the Galois field fundamentals. In Section 4, we present an approach for modeling the required energy of NC based on hardware (VLSI) simulation results and we analyze how different values of algorithmic parameters affect the total system's energy. Finally, in Section 5, we present our implementation's results and in Section 6, conclusions are summarized.

2 Network Coding Overview

Assume that a node has to transmit n packets, $\mathbf{P} = [P_1, \ldots, P_n]$, each of L bits length. If the node uses Random Linear Network Coding (RLNC) [6], it will first create n linear combinations of them, and then will transmit the result of the coding process. More specifically, q consecutive bits in each packet are considered to form a symbol over the field GF(2^q), resulting in L/q symbols per packet. The node randomly generates n sets of coefficients, $\mathbf{C} = [C_1, \ldots, C_n]$, each of them associated with a specific coded packet X_i. The coded packets $\mathbf{X}$ are generated with a matrix multiplication: $\mathbf{X} = \mathbf{C} * \mathbf{P}$. As soon as a node has received n linear independent coded packets, it can start the decoding process, which is actually a problem of solving n equations with n unknowns, recovering the original packets.

Operations like addition, multiplication and division over GF are involved in the en-/decoding process. As a result, for an energy efficient implementation of NC, a detailed examination of these operations should be done in advance.

3 Galois Field Fundamentals

In this Section, we provide a brief description of the most important concepts of Abstract Algebra related to NC. Our description serves only the purpose of explaining the decisions made during the design steps; for a more mathematically rigorous approach readers are referred to Algebra books.

A *field* $\mathbb{F}$ is a set of at least two elements, with the operations $\otimes$ and $*$ (often called addition and multiplication), satisfying certain properties. One of these properties is that $\mathbb{F}$ is closed under the two operations, meaning that when an operation is applied to some elements of $\mathbb{F}$, the result will also be an element of this field. The number of elements in $\mathbb{F}$ is called *order* and, when this number is finite, the field is called *finite field*, denoted also as *Galois field* (GF). For any prime number p, it is always defined a GF with order p, represented as GF(p), having exactly p elements: GF(p)={0,1, …, p-1}. We can also create a GF(p^q), for any *q>0*, called *extension field* of GF(p). The definition of *field size* is often used to characterize the size of a field, denoted as q, where $q = log_p p^q$. Elements from GF(p^q) are usually considered and treated as vectors $[a_{q-1}, \ldots, a_0]$ or polynomials of degree at most (q-1) with coefficients from GF(p). Finally, all GFs contain a zero, an identity and a primitive element, and have at least one primitive polynomial of degree q associated with them.

The representation basis of the elements in a field is a crucial aspect, determining the efficiency and complexity of the implementation of different arithmetic operations. There are several representation bases; the more popular among them are the standard (or polynomial) and the normal basis. The standard basis is the set of elements $\Omega = \{1, \omega, \omega^2, \ldots, \omega^{q-1}\}$, where ω is a primitive element of the field GF(p^q), while the normal basis is the set $\Psi = \{\psi, \psi^p, \psi^{p^2}, \ldots, \psi^{p^{q-1}}\}$, where ψ is a generator of the basis. Although the normal basis is more suitable for multiplying two numbers, we choose for our implementation to work entirely on the standard basis in order to avoid conversions when data are exchanged between the accelerator and other hardware modules, because standard basis represents numbers in the same way as fixed-point representation does.

In general, GFs play an important role in many communication systems, such as FEC and cryptographic schemes. Since digital computing machines use Boolean logic, the binary field GF(2)=$\{0, 1\}$, and its extension fields GF(2^q), are widely used, due to the direct map between their elements and the Boolean values. In the rest of this work we consider only binary fields.

3.1 Addition over Galois Fields

As mentioned previously, each element from GF(2^q) can be represented as a q-bit vector or polynomial of degree (q-1). Adding two elements is equivalent of adding the coordinates of each vector, or adding the two polynomials, using GF(2) arithmetic. This means that the implementation of addition over GF(2^q) corresponds simply to a bit-wise XOR operation. Table 1 summarizes the area, power and delay requirements of a standard (a q-bit carry ripple adder) and a GF adder. In this table, area is calculated as the required number of gates, power is approximated to be analogous to area and delay is considered the maximum time for each operation, assuming that AND, OR and XOR gates have the same area, power and delay.

Table 1: Comparison between standard and Galois Field arithmetic

q-bit arithmetic		Area	Power	Delay
Addition	Standard	$5q$	$5q$	$2q+1$
	Galois Field	q	q	1
Multiplication	Standard	$6q^2-8q$	$6q^2-8q$	$3q-2$
	Galois Field	$2q^2+2q$	$2q^2+2q$	$4q$

3.2 Multiplication over Galois Fields

Using standard basis representation, multiplication over GF(2^q) of two elements, b and c, can be computed as: D(x) = (B(x)C(x)) mod$p(x)$, where *p(x)* is primitive polynomial of the field. As we see, GF multiplication is equivalent of polynomial multiplication followed by polynomial modulo reduction. These two operations can be performed separately, leading to fully parallel, modular and standard multiplication architectures.

In general, there are several ways to implement a GF multiplication and the resulting performance is highly dependent on the underlying hardware platform. In previous implementations of NC, logarithmic look-up tables and iterative approaches have been used because of the CPU-based approaches. In this work, trying to minimize the energy consumption of NC, we use a custom, low-power GF multiplier. A widely used algorithm in cryptographic and error correction applications for GF multiplication is Rijndael's algorithm. Our architecture implements a modified version of this algorithm, computing the product of two elements in one clock cycle and having no pipeline stages. The reason for such a choice is that our design aims an ultra low-power operation; direct implementation of the iterative Rijndael's algorithm would result in approximately q times less critical path delay, but also in larger overall power consumption. Table 1 summarizes area, power and delay requirements of a standard q-bit array and our GF multiplier.

4 Energy Modeling and Optimization of NC

In the following paragraphs, we try to model the energy consumed during the encoding process and specify optimum algorithmic parameters, such as field size, taking into consideration the total system's energy (the results presented in the following paragraphs have been obtained after modeling every circuit component using Verilog, synthesizing and performing post-layout simulations, with a 65nm TSMC process, using standard VLSI CAD tools, such as SPICE). Our main focus is on the encoding, since NC does not follow the end-to-end coding paradigm; intermediate nodes are allowed to re-encode packets without decoding them first, and only final destination nodes have to decode the mixtures.

The encoding process is equivalent of generating a new packet as a linear combination of the existing blocks, weighted according to some random coefficients. Assume that a source wants to transmit n packets, each of length L bits, using RLNC over GF(2^q). The required energy per packet for the encoding process is:

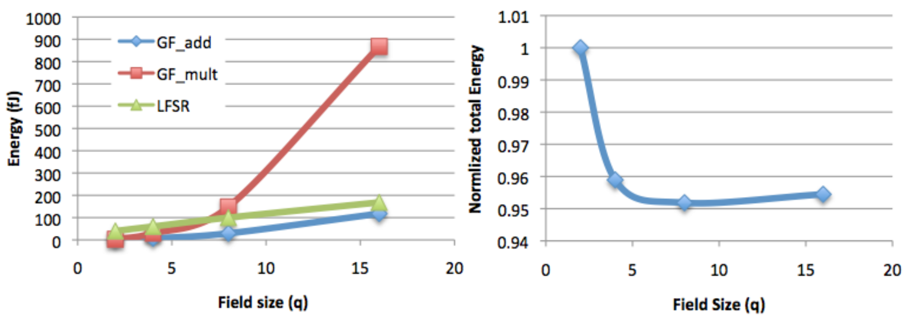
$$E_{COD} = nE_{LFSR} + \frac{L}{q}(nE_{MULT} + (n-1)E_{ADD}) \ , \tag{1}$$

where E_{MULT} and E_{ADD} is the energy consumed per multiplication and addition, respectively, and E_{LFSR} is the energy consumed for generating a q-bit coefficient using a LFSR (*Linear Feedback Shift Register*). In Fig. 1a is shown how the choice of field size affects the energy for each operation. In this comparison, we keep the processing data rate same, since, doubling the field size q results in doubling the critical path delay of the GF multiplier, which is used as the reference time period for our circuits; in other words, the ratio of the number of processed bit per operation over the required time for each operation, remains constant.

Furthermore, apart from the processing energy, field size also affects the probability of two coded packets being linearly dependent. It can be shown that the expected number of transmitted packets until receiving n linear independent combinations, using RLNC over GF(2^q), is given by the following formula:

$$\bar{n} = \sum_{i=1}^{n} \frac{1}{1-(\frac{1}{2^q})^i} \ . \tag{2}$$

Now it becomes clear that a small field size lowers the required processing energy but results in extra retransmitted packets, increasing the total system's energy.



(a) (b)

Fig. 1: a) Energy per operation for different values of q. b) Normalized total system's energy of a node transmitting 8 coded packets, using RLNC over GF(2^q).

This trade-off can be further explained by examining the expected total system energy (E_{TOT}), given by:

$$E_{TOT} = \bar{n}(E_{COD} + E_{TX}) \ , \tag{3}$$

where E_{TX} is the transmission energy per packet. In Fig. 1b, the total normalized system energy, including both coding for NC and transmission energy, is plotted, assuming that 8 packets are coded together, each of 1KB length, and a transmission energy per bit of $200pJ/bit$ [1]. Examining the plot we confirm that, when a small field size is used, the total system's energy in dominated by the extra RF energy due to packet retransmissions. However, as field size becomes large, increased energy is required for performing the coding process, without significantly affecting the expected number of transmitted packets, resulting in higher system's energy consumption.

5 Energy-Aware NC Accelerator

In the following paragraphs we present the results of our energy-aware VLSI implementation of NC, giving its architecture but not focusing in the low-level, circuit-related details. Given our target low-power applications, the number of packets encoded together is expected to be relatively small; in our analysis, we assume that up to 8 packets can be coded together, each of length 1KB. Based on the analysis of the previous Section, we use GF(2^8) arithmetic, since field size of 8 appears to be the optimum operation point. Our NC encoder architecture is shown in Fig. 2. It is a parallel implementation of the encoding process, making use of standard low-power VLSI techniques, such as clock gating, parallelism and voltage scaling, to reduce power consumption and achieve energy scalability.

We also design a custom accelerator performing the decoding process. A standard Gauss-Jordan elimination algorithm is implemented, capable of transforming the coefficients' matrix into row echelon form after receiving every packet.

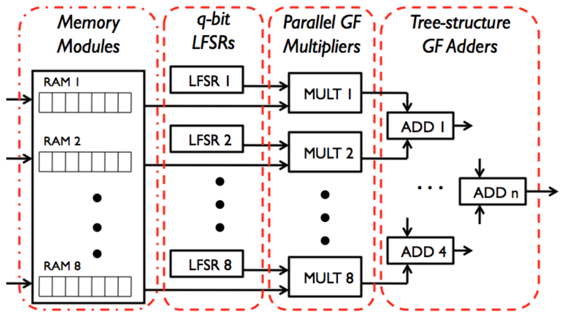


Fig. 2: Block diagram of the NC encoder.

The same GF adder and multiplier modules, discussed previously, are used also in the decoder. For calculating the inverse value of an element, look-up tables are used. The reasons for using look-up tables in the inversion and not in the multiplication process are, first, that the multiplication requires look-up tables of q^2 elements and, second, that the GF inversion algorithm is more complex and energy consuming than the multiplication one. Our hardware implementation results, using a TSMC 65nm process, are shown in Table 2 (reported numbers for the NC encoder are post-layout measurements, while for the NC decoder are post-synthesis).

Table 2: Implementation results of our energy-aware, custom NC accelerator.

	NC Encoder		**NC Decoder**
Supply Voltage	$0.4V$	$1.0V$	$1.0V$
Frequency	10 MHz	250 MHz	50 MHz
Throughput	80 MB/s	2 GB/s	60 MB/s
Power	22.15 uW	10.98 mW	12.5 mW

6 Conclusions

It has been shown that Network Coding can provide several advantages to a network, but it is associated with an extra energy cost. In this paper, we provide an energy analysis of NC, especially useful to power constrained networks. We answer the question of how much energy is required for incorporating NC into a system architecture, using a custom and optimized implementation. With detailed energy modeling of the required NC operations, based on custom VLSI measurement, we optimize the trade-offs between coding performance, computational complexity and energy consumption. By designing, to the best of our knowledge, the first energy-aware VLSI NC accelerator and specifying optimum algorithmic parameters, we believe that a further step is made to bridge NC with real-world, low-power applications, such as sensor and mobile networks.

7 Acknowledgment

The authors would like to thank X. Shi, P. Nadeau and A. Paidimarri for useful discussions on Network Coding and RF design. This material is based upon work supported by the Georgia Institute of Technology Under Award Number: 017894-010.

References

1. A. P. Chandrakasan, N. Verma, D. C. Daly: Ultralow-power electronics for biomedical applications. Annu. Rev. Biomed Eng 10, 247–274 (2008)
2. R. Ahlswede, N. Cai, S. R. Li, R. Yeung: Network Information Flow. Information Theory, IEEE Transactions on 46(4), 1204 –1216 (Jul 2000)
3. S. Chachulski, M. Jennings, S. Katti, D. Katabi: Trading structure for randomness in wireless opportunistic routing. SIGCOMM Comput. Commun. Rev. 37, 169–180 (August 2007)
4. C. Fragouli, E. Soljanin : Network Coding Fundamentals. Now Publisher, pp. 1–133 (January 2007)
5. J. Heide, M. Pedersen, F. Fitzek, T. Larsen: Network coding for mobile devices - systematic binary random rateless codes. In Workshop on Cooperative Mobile Networks, 2009. ICC09 IEEE, (June 2009)
6. T. Ho, R. Koetter, M. Medard, D. Karger, M. Effros: The benefits of coding over routing in a randomized setting. In Proc. of IEEE ISIT'03. (July 2003)
7. S. Katti, et al.: XORS in the air: practical wireless Network cCoding. IEEE/ACM Trans. Netw. 16, 497–510 (June 2008)
8. H. Li, Q. Huan-Yan: Parallelized Network Coding with SIMD instruction sets. In Proc. of International Symposium on Computer Science and Computational Technology, 2008. ISCSCT'08. vol. 1, pp. 364 –369 (Dec 2008)
9. D. S. Lun, M. Médard, R. Koetter, M. Effros: On coding for reliable communication over packet networks. CoRR abs/cs/0510070 (2005)
10. H. Shojania, B. Li, X. Wang: Nuclei: GPU-accelerated Many-core Network Coding. In: INFOCOM 2009, IEEE. pp. 459 –467 (April 2009)
11. H. Shojania, B. Li: Random Network Coding on the iPhone: Fact or Fiction?, In Proc. of the 18th Int. Workshop on Network and Oper. Systems Support for Digital Audio and Video. pp. 37–42. NOSSDAV '09, ACM, New York, NY, USA (2009)
12. P. Vingelmann, P. Zanaty, F. Fitzek, H. Charaf: Implementation of Random Linear Network Coding on openGL-enabled graphics cards. In European Wireless Conference, 2009. EW 2009. pp. 118 –123 (May 2009)
13. P. Vingelmann, Frank H. P. Fitzek : Implementation of Random Linear Network Coding using NVIDIA's CUDA toolkit. In: Networks for Grid Applications, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 25, pp. 131–138. Springer Berlin Heidelberg (2010)
14. M. Wang, B. Li: How practical is network coding? In Proc. of 14th IEEE Int. Workshop on QoS, IWQoS 2006, pp. 274 –278 (June 2006)