Sharing the Pi

Paramanathan, Achuthan; Pahlevani, Peyman; Thorsteinsson, Simon; Hundebøll, Martin; Roetter, Daniel Enrique Lucani; Fitzek, Frank Hanns Paul

Published in:
Vehicular Technology Conference (VTC Spring), 2014 IEEE 79th

DOI (link to publication from Publisher):
10.1109/VTCSpring.2014.7023090

Publication date:
2014

Document Version
Peer reviewed version

Sharing the Pi: Testbed Description and Performance Evaluation of Network Coding on the Raspberry Pi

Achuthan Paramanathan, Peyman Pahlevani, Simon Thorsteinsson, Martin Hundebøll
Daniel E. Lucani, Frank H.P. Fitzek
Department of Electronic Systems, Aalborg University
Email: {ap|pep|sthors10|mhu|del|ff}@es.aau.dk

***Abstract*—This paper presents the design and performance evaluation of an inexpensive testbed for network coding protocols composed of Raspberry Pis. First, we show the performance of random linear network coding primitives on the Raspberry Pi in terms of processing speed and energy consumption under a variety of configuration setups. Our measurements show that processing rates of up to 230 Mbps are possible with the Raspberry Pi. Also, the energy consumption per bit can be as small as 3 nJ/bit, which is several orders of magnitude smaller than the transmission/reception energy use. Surprisingly, overclocking the Raspberry Pi from 700 MHz to 1000 MHz not only produces an increase in processing speed of up to 68 % for large generation sizes, but also provides a reduction of 64 % in the processing energy per bit for most tested scenarios. Then, we show Raspberry Pi as an inexpensive, viable, and flexible platform to deploy large research networking testbeds for the evaluation of network coding protocols. We propose key parameters and representations to evaluate protocol performance in network nodes as well as validating the testbed's statistics using the case of a one–hop broadcast with random linear network coding, which is well understood in theory.**

I. Introduction

In recent years, network coding [1] has proven to be a ground breaking technology for communication and storage systems. Network coding provides an effective mechanism to increase throughput, provide reliability, and reduce protocol complexity by changing the goal of each receiver from getting individual packets to getting enough independent linear combinations. The fundamental change introduced by network coding is to treat data packets in the network as algebraic entities that can be operated upon instead of immutable objects. This radically changes the network operation's paradigm from a store and forward one to a stored, compute, and forward one. This ability to code in the network contrasts with standard end–to–end erasure correcting codes, e.g., Reed–Solomon (RS), LT codes, Raptor codes. This ability allows network coding to adapt to network losses in the best possible way without needing to introduce redundancy end–to–end, thus avoiding an inherent inefficiency in network resource use of end–to–end codes.

Substantial theoretical research work on network coding has been carried since its introduction in 2000 by Ahlswede [1] showing significant gains in a variety of scenarios. Demonstrators on commercial platforms have been introduced, e.g., [2], [3], [4], [5], but these demonstrators are rare in contrast with the prolific theoretical research in the area. The introduction of the network coding software library KODO [6] was intended to change this trend by making network coding implementation available to more research groups, particularly those developing protocols. KODO is implemented in C++ providing fast implementations of finite field operations and network coding basic primitives, i.e., encoding, decoding, and recoding, compatible with a variety of coding strategies. This software library runs on several platforms (e.g., Android, iOS, Windows, Linux, MacOS) and can be compiled by different compilers (e.g., gcc, clang). Thus, KODO enables researchers and engineers to use network coding out of the box and focus on the implementation of the communication protocols.

Fig. 1: Raspberry Pi Testbed at Aalborg University.

This paper leverages KODO to take a significant step forward in this process breaking ground to make network coding available for all interested researchers at nearly no cost. Our goal is to ease the process of building a network coding capable testbed for testing more complex protocols. Using the Raspberry Pi as an inexpensive device to allow for large scale deployments, this paper first shows a performance characterization of the Raspberry Pi for a wide range of critical network coding parameters, e.g., different field size arithmetics, generation sizes, as well as characterizing performance when overclocking the CPU and providing a thorough study of the energy per bit consumption of these devices. The paper also provides key installation and compilation steps for KODO in these devices. This paper also provides a description of the testbed's features and relevant statistics for a variety of network coding protocols. Finally, we provide measurement

results for standard communication topologies using network coding to validate our system.

II. Network Coding Principles

This section describes basic network coding principles. In particular, we discuss principles of intra–session network coding, which focuses on creating linear combinations of packets of the same session. The most standard approach is to organize the original packets into groups of g packets, e.g., $p_1, p_2, ..., p_g$. This group is called a generation of size g. Then, the packets are linearly combined to create a coded packet, e.g., $\sum_{i=1}^{g} c_i p_i$. To do this in practice, we consider that each original packet can seen as a concatenation of symbols over a finite field $GF(q)$, where each symbol has a length of $h = \lceil \log_2(q) \rceil$ bits. Thus, a linear combination of the g packets occurs on a symbol by symbol basis. The choice of the coding coefficients, c_i, is critical for determining performance and a variety of schemes are possible. However, we focus on random linear network coding (RLNC) [7], where each coding coefficient is picked uniformly at random from the elements in the finite field. The reason is that (i) a variety of proposed schemes rely on RLNC, e.g., [4], [8], (ii) key performance metrics studied for RLNC are also relevant in other coding approaches, and (iii) RLNC's performance is well understood and modeled, e.g., [9], [10], which will allow us to validate our practical results.

In order to understand the performance of network coding protocols, there are a series of key metrics and statistics that can shed light on a protocol's performance. We consider three statistics that are applicable not just to RLNC protocols but also useful for future protocols with more complex coding structures, e.g., [11], [12]. In the following, we describe these statistics as well as their expected RLNC performance under ideal conditions.

- **Mean number of linearly dependent coded packets at a node when it has a given rank:** This metric is key to understanding how effective the protocol is in delivering independent linear combinations with each transmission. Using the model in [9], [13], we can calculate the mean number of linearly dependent coded packets, $D(g-i)$, for a receiver when it has rank $g-i$ as

$$D(g-i) = \frac{1}{1-q^{-i}} - 1 = \frac{1}{q^i - 1}. \qquad (1)$$

  Although this is valid for an ideal RLNC system, specific implementations can differ. This is particularly true if we consider the effect of recoding at intermediate nodes, which increases the probability of transmitting linearly dependent combinations because the transmitting intermediate node may not have full rank. On the other end of the spectrum, this statistic is critical to understanding the performance of sparse network codes, for which performance is not characterized is such a clean fashion as for RLNC.
- **Number of received coded packets before decoding:** Relying on the absorbing Markov chain formulation in [9], the transition probability is given by

$$P_q = \begin{bmatrix} q^{-g} & 1-q^{-g} & 0 & \cdots & 0 & 0 \\ 0 & q^{-g+1} & 1-q^{-g+1} & \cdots & 0 & 0 \\ : & : & : & : & : & \\ 0 & 0 & 0 & \cdots & q^{-1} & 1-q^{-1} \\ 0 & 0 & 0 & \cdot\cdot & 0 & 1 \end{bmatrix}.$$

  The probability of decoding on k transmissions is

$$\mathcal{P}(k) = P_q^k(1, g+1) - P_q^{k-1}(1, g+1), \qquad (2)$$

  where $P_q^k(i,j)$ represents the k-th power of P_q reading the i–th row and j–th column.
- **Total transmissions per device:** This is key to characterizing the overall performance of the system, which is particularly important in the presence of multiple receivers. This metric captures the effect of channel losses. Previous research, e.g., [14] has studied the case of broadcast to multiple receivers under the assumption that the field size is large. [10] provided a thorough model of field size effects for the case of two receivers, but the extension to more receivers proved cumbersome. Analyzing these effects for a larger number of receivers through the testbed is of value for practical systems.

III. Raspberry Pi How To for RLNC

This section, briefly describes the setup for our experimental testbed to be used for conducting our measurement results.

A. Installing an Operating System

Raspberry Pi Model B [15] has been selected as the testing platform. We installed a Linux distribution known as the Raspbian OS [16]. An easy to follow guide on how to install the OS can be found in [17].

B. RLNC on Raspberry Pi

The RLNC related methods and tools used in this paper are provided by the KODO Library [6]. KODO is a C++ library that provides several methods, tools, and functions that can be used by other applications for RLNC related operations. We have used the coding and decoding of data packets in this paper, but recoding is also readily available. KODO also allows to seamlessly change key coding parameters such as the finite field size, namely, $GF(2)$, $GF(2^8)$, $GF(2^{16})$, and $GF(2^{32}-5)$, and the generation size. For a detailed description on how to install KODO, we refer the interested reader to [18]. However, KODO works directly out of the shelf on Raspberry Pi. We advise the interested reader to follow the instructions provided by the installation script in [18], this script automatically download the missing dependencies and guide the user through with compiling, e.g., code examples and benchmark applications.

C. Overclocking the Raspberry Pi

By default, the Raspberry Pi Model–B is configured for a processing speed of 700 MHz. However, it provides a series of overclocking options. We can use the

sudo raspi-config

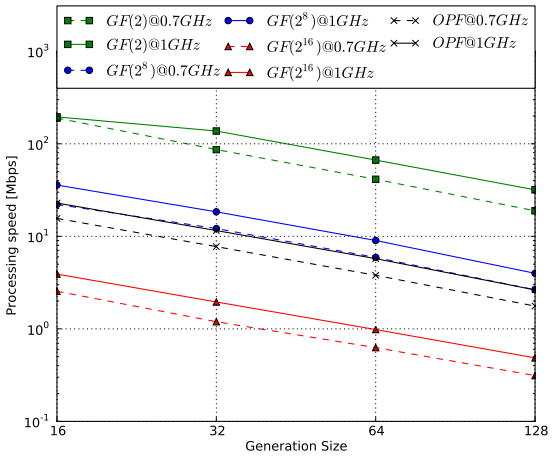

Fig. 2: Processing speed of encoding [Mbps] for the Raspberry Pi for various field sizes, generations sizes, and core operating frequency

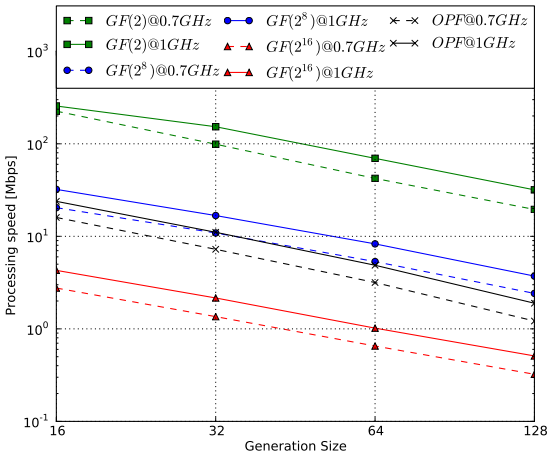

Fig. 3: Processing speed of decoding [Mbps] for the Raspberry Pi for various field sizes, generations sizes, and core operating frequency

command in a terminal on the Raspberry Pi to access the configuration to overclock the device . This command will open a GUI, here select option 7 (Overclock) and select the desired frequency. We have characterized the two extreme cases in this paper, namely 700 MHz and 1000 MHz.

D. Configuration of the Mesh

The coded packets are wirelessly exchanged between several Raspberry Pis devices over a meshed network using a Wi–Fi dongle of type TP–Link (TL–WN722N) [19]. There are several ways to achieve this. For example, one can simply hard code the mesh or use available mesh routing protocols. However, for our setup, we are using B.A.T.M.A.N. [20] as our mesh routing protocol, because it is part of the Linux kernel and it is straightforward to use. We refer the interested reader to [20] for a detailed explanation of how to setup and use B.A.T.M.A.N. [20] in Linux.

IV. RLNC Performance in Raspberry Pi

In this section, we shall characterize the processing speed and energy per bit consumption characteristics of the Raspberry Pi Model B [15] for various RLNC parameters. These measurement characteristics and strategies are similar to those that used in [21].

A. Processing Speed

The performance measurement of network coding is handled by running the KODO benchmark application. By running this application, we get the processing speed of RLNC for different coding parameters. We adjust two key parameters: the finite field and the generation size. Each packet tested in this benchmark is of size 1000 bytes.

Figure 2 and Figure 3 show the data processing speeds for encoding and decoding for two different CPU frequencies (0.7 GHz and 1 GHz) using different field size and generation size. These figures show that the Raspberry Pi is able to encode and decode at speeds of up to 230 Mbps while using $GF(2)$ and with a generation size of 16. The figures show that differences between the CPU frequencies have a significant impact on processing speed especially for higher field sizes and larger generation sizes. In fact, processing at 1 GHz outperforms 0.7 GHz up to a factor of 1.68, i.e., 68 % faster. This tendency can also be seen for higher field sizes and generation sizes.

TABLE I: Average Current Measurements

Idle Device @ 700MHz	0.4475 A
Idle Device @1000MHz	0.5211 A
Processing (Encoding/Decoding) @700MHz	0.1156 A
Processing (Encoding/Decoding) @1000MHz	0.1389 A
WiFi Reception	0.5670 A
WiFi Transmission	0.6500 A

B. Energy Consumption

The energy performance measurement focuses on one Raspberry Pi device connected to our measurement equipment (Agilent 66319 [22]). The Agilent supplies power to the device and logs both the current flow and the voltage usage of the device during the measurement. Table I shows the average current during our measurements. Combining the measurements from Table I with the processing speed, we compute the energy per bit used for processing in the device.

Figure 4 and Figure 5 shows the energy per bit for encoding and decoding respectively for frequencies 0.7 GHz and 1 GHz. Again, we see a significant difference in terms of energy cost per processed data bit. For most field sizes and generation sizes, increasing the processing speed from 0.7 to 1 GHz reduces the energy per bit by a factor of 1.64. This is explained by the processing speed increase by a factor of 1.68 and the minor increase in current used by the device (See Table I).

V. Experimental Setup: One to All Broadcast

Our measurement setup consists of ten Raspberry Pi devices, placed inside the campus building. Each device is configured using the procedure in Section III. One device has been configured as a source, which has the purpose of broadcasting data packets to the other devices in the network. The remaining nine devices have been configured as destination devices with the purpose of listening for coded packets from the source, store them, and decode them when enough linear combinations

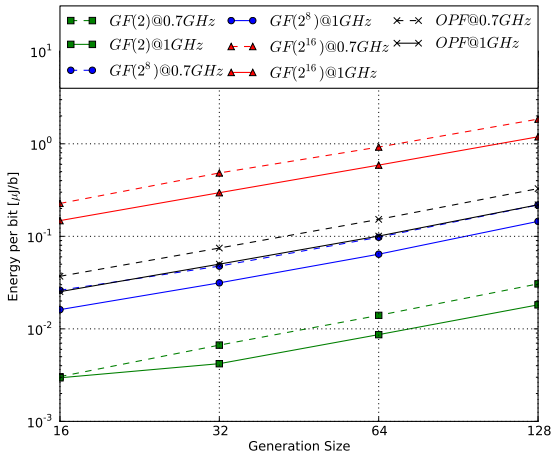

Fig. 4: Energy per bit for encoding for the Raspberry Pi for various field sizes, generations sizes, and core operating frequency

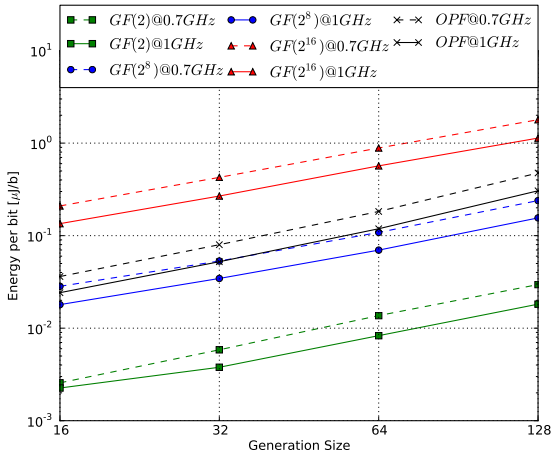

Fig. 5: Energy per bit for decoding for the Raspberry Pi for various field sizes, generations sizes, and core operating frequency

have been received. These testbed devices are controlled and configured by a central server. The server also controls the execution of tests and collection of results from each test from each of the devices to be able to calculate the desired statistics.

A. Test applications

In order to perform measurement on these devices, we wrote two applications, one for the test server (PC) and one for the clients (Raspberry Pis). The server side application can be executed on a PC with the necessary testing parameters and KODO configurations, e.g., finite field size, generation size, packet size, duration of each test.

The client application encapsulates KODO and manages the connectivity of our wireless mesh network. Depending on the role assigned within the network, it will use a different set of functionalities of the test application. If the device is targeted as the source by the server then the client application will use KODO to encode data packets according to the configuration parameters fed by the server for that test. Each coded packet is

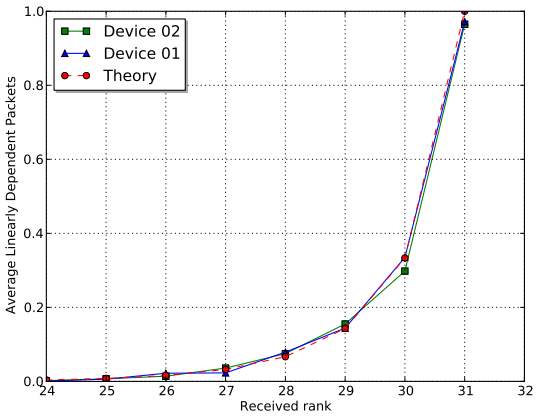

Fig. 6: Average linearly dependent packets for different received rank using $GF(2)$ and a generation size of 32

then transmitted using the wireless mesh network. If a device is chosen as a destination, then the client application will listen to any incoming coded packet from the source over the mesh network and try to decode the original packets. For both the source and destination the application generates a statistics of the performance metrics (described earlier) and log them as a test file. At the end of our experiment, this log file is then collected by the server for further analysis.

B. Configurations and Experimental Procedure

The communication link between the source and the destinations are established over UDP–socket connections on top of the B.A.T.M.A.N. routing protocol [20]. The source is configured to broadcast data packets over the established connection to the destination devices. The size of the transmitted packet is set to 1000 bytes and for the current tests we have used a $GF(2)$. We tested with the following generation sizes: 16, 32, 64, and 128 packets, where a series of tests are performed separately for each generation size. We illustrate two out of the three key statistics of Section II due to space constraints.

C. Experimental Results and Discussion

Figure 6 shows the linearly dependent packets for different received rank using $GF(2)$ and a generation size of 32 for two of the nine devices from our testbed and the theoretical RLNC model of Section II. This figure shows that the performance of each individual device matches that of the theoretical model for $GF(2)$. This behavior is expected and validates the RLNC nature of the coded packets transmitted from the source to the destinations. All other devices showed similar performance. This metric will be more relevant in practical settings where there is recoding at intermediate nodes because an aggressive relay may send too many coded packets before having collected sufficient dimensions, thus increasing the number of linearly dependent packets received. Identifying variations with the current pattern will allow us to identify inefficiencies in an RLNC protocol.

Figure 7 illustrates the maximum number of transmission needed by the source in order for the destination to decode the received packets. The packet loss ratio was below 10 %.

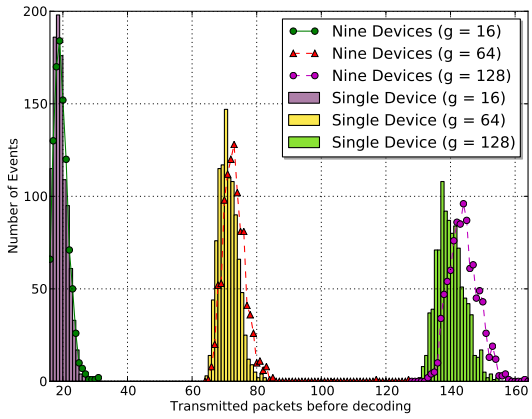

Fig. 7: Maximum transmission needed by the source before all the nodes are able to decode

This metric inherently brings together the effect of multiple receivers, field size effects, and effects of channel losses. Figure 7 shows the distribution of the number of transmissions needed by the source before all destinations are able to decode (lines) for generation sizes of 16, 64, and 128 packets. The bar plots illustrate the distribution of the number of received packet before decoding for a single device. This result shows that there is a compounded effect of the performance of individual devices on the overall transmission distribution of the source. This effect is in part due to the effect of losses and the linear dependencies that occur for RLNC in $GF(2)$, as shown in Figure 6. Figure 7 also shows that an increase in generation size can radically increase this effect even for low packet losses. This is explained by the fact that there is a larger probability to experience a losses in each individual device, which creates differences in the linear combinations accumulated by the devices during the transmission process translating in a larger deviation in the overall system performance with respect to that of individual devices.

VI. Conclusion

This paper showed the key mechanisms to setup, configure, and operate inexpensive large–scale network coding enabled deployments using the Raspberry Pi as key testing device. We showed that the Raspberry Pi device is capable of high processing speeds, especially when overclocking it, as well as providing a thorough characterization of the energy per bit consumption of it. We also proposed some key statistics of interest that can be computed and extracted at each device to better understand a network coding protocol. All these findings prove the fact that even a low cost device as the Raspberry Pi can run network coding at outstanding speeds.

Although we demonstrated the testbed's capabilities using the simple case of a one–to–all broadcast case, a variety of topologies and protocols can be tested using our approach, e.g. wireless meshed networks with different topologies. Our goal was to validate our results with a well understood topology that can be easily repeated as a first step after setting up the Raspberry Pi. The main benefit of this paper is the fact that it empowers other researchers to carry out their own network coding based research on commercial WiFi enabled devices at low cost. The basic steps explained in this paper show the way how to bring network coding on a Raspberry Pi.

Acknowledgment

This work was partially financed by the Green Mobile Cloud project granted by the Danish Council for Independent Research (Grant No. DFF - 0602-01372B).

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, , and R. W. Yeung, "Network information flow," *IEEE Trans. on Info. Theory*, vol. 46, no. 4, 2000.

[2] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: practical wireless network coding," vol. 36, no. 4, August 2006.

[3] M. Hundebøll, J. Ledet-Pedersen, S. Rein, and F. H. Fitzek, "Comparison of analytical and measured performance results on network coding in ieee 802.11 ad-hoc networks," in *Proc. of 76th IEEE Vehicular Technology Conference*, 2012.

[4] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *In Proc. of ACM SIGCOMM 2007*, 2007.

[5] J. Hansen, J. Krigslund, D. E. Lucani, and F. Fitzek, "Bridging Inter-flow and Intra-flow Network Coding for Video Applications: Testbed Description and Performance Evaluation," in *IEEE Int. Workshop on Comp. Aided Mod. and Des. of Comm. Links and Netw.(CAMAD)*, Berlin, Germany, June 2013.

[6] M. V. Pedersen, J. Heide, and F. Fitzek, "Kodo: An open and research oriented network coding library," *Lecture Notes in Computer Science*, vol. 6827, pp. 145–152, 2011.

[7] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Trans. on Info. Theory*, vol. 52, no. 10, pp. 4413 –4430, Oct. 2006.

[8] J. Krigslund, J. Hansen, M. Hundebøll, D. Lucani, and F. Fitzek, "CORE: COPE with MORE in Wireless Meshed Networks," in *IEEE Veh. Tech. Conf. (VTC)*, Dresden, Germany, June 2013.

[9] D. E. Lucani, M. Medard, and M. Stojanovic, "Random linear network coding for time-division duplexing: Field size considerations," in *IEEE Global Telecomm. Conf. (GLOBECOM)*, 2009, pp. 1–6.

[10] M. Nistor, D. E. Lucani, T. T. V. Vinhoza, R. A. Costa, and J. Barros, "On the delay distribution of random linear network coding," *IEEE Journal on Sel. Areas in Comm.*, vol. 29, no. 5, pp. 1084–1093, 2011.

[11] S. Feizi, D. E. Lucani, and M. Médard, "Tunable sparse network coding," in *Proc. of the Int. Zurich Seminar on Comm.*, March 2012, pp. 107–110.

[12] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth codes: maximizing sensor network data persistence," in *Proc. Conf. on App., tech., arch., and prot. for comp. comm.*, ser. SIGCOMM '06. New York, NY, USA: ACM, 2006, pp. 255–266.

[13] D. E. Lucani, M. Medard, and M. Stojanovic, "On coding for delay - network coding for time-division duplexing," *IEEE Trans. on Info. Theory*, vol. 58, no. 4, pp. 2330–2348, 2012.

[14] A. Eryilmaz, A. Ozdaglar, and M. Medard, "On delay performance gains from network coding," in *Conf. on Info. Sci. and Sys. (CISS)*, 2006, pp. 864–870.

[15] Raspberry pi. [Online]. Available: http://www.raspberrypi.org/

[16] Raspbian operating system. [Online]. Available: http://www.raspbian.org/FrontPage

[17] Raspberry pi guide. [Online]. Available: http://www.raspberrypi.org/quick-start-guide

[18] Kodo getting started. [Online]. Available: http://kodo.readthedocs.org

[19] Tp-link technologies, wireless usb adapter. [Online]. Available: http://www.tp-link.com/en/products/details/?model=tl-wn722n

[20] Freifunk. B.a.t.m.a.n. advanced documentation overview. [Online]. Available: http://www.open-mesh.org/

[21] A. Paramanathan, M. V. Pedersen, D. E. Lucani, F. H. P. Fitzek, and M. Katz, "Energy and power measurements for network coding in the context of green mobile clouds," in *Proc. of the The Tenth International Symposium on Wireless Communication Systems*, 2013.

[22] D. A. Technologies. 66319d dual mobile comm dc source w/ battery emulation, dvm. [Online]. Available: http://www.home.agilent.com