Network Coding Designs Suited for the Real World: What works, What doesn't, What's Promising

Morten V. Pedersen[1], Daniel E. Lucani[1], Frank H. P. Fitzek[1], Chres W. Sørensen[1], Arash S. Badr[2]
[1]Department of Electronic Systems, Aalborg University, Denmark
[2]School of Electrical Engineering, Computer Science and Mathematics, Technische Universität Hamburg–Harburg, Germany
Email:{ mvp, del, ff, cws }@es.aau.dk, arash.shahbaz@tuhh.de

***Abstract*—Network coding (NC) has attracted tremendous attention from the research community due to its potential to significantly improve networks' throughput, delay, and energy performance as well as a means to simplify protocol design and naturally providing security support. The possibilities in code design have produced a large influx of new ideas and approaches to harness the power of NC. But, which of these designs are truly successful in practice? and which designs will not live up to their promised theoretical gains due to real–world constraints? Without attempting a comprehensive view of all practical pitfalls, this paper seeks to identify key ingredients to a successful design, critical and common limitations to most intra–session NC systems as well as promising techniques and ideas to guide future models and research problems grounded on practical concerns.**

I. Introduction

Since its introduction in 2000 [1], network coding (NC) has been shown to provide significant theoretical gains in a variety of fields, from multicasting of data in wireline and wireless networks, to more efficient designs for peer–to–peer communication and distributed storage systems, to wireless meshed networks even with highly dynamic changes. The fundamental idea of network coding is to encourage the system to mix different data packets at intermediate nodes through coding, rather than storing and forwarding copies of packets that are routed through the network. Under this premise, it is no longer required for the system to keep track of which packets have been received: receivers need only aim at accumulating enough coded packets in order to recover the information. Network coding thus provides resilient and throughput efficient ways for mobile, heterogeneous, and/or content–driven networks to gather, store, and transmit information in dynamic environments. A simple, distributed and asymptotically optimal solution is random linear network coding (RLNC), which relies on generating coded packets as linear combinations of original packets by choosing the coefficients of these combinations uniformly at random from the elements of a finite field [2].

Although some practical implementations have been reported, e.g., COPE [3], CATWOMAN [4], MORE [5], and CATWOMAN has recently made its way into the Linux kernel, a wide spread adoption of network coding in real world systems and off the shelf devices remains elusive. One reason why NC is not deployed yet on multiple platforms is the believe that NC will add too much complexity to the installed communication systems. Complexity is a result of how NC is implemented and the choice of coding parameters. Let's have a look at these two issues independently:

This work was financed in part by the Green Mobile Cloud project granted by the Danish Council for Independent Research (Grant No. 10-081621) and the Colorcast project granted by Danish Ministry of Science (Grant No. 12-126424/FTP).

- Until recently there was no available, fully tested cross platform software library that allows system integrators/researchers to deploy/research NC without implementing their own solutions. This has changed with the introduction of Kodo [6], which comes with a dual license for both research and commercial use. Throughout this paper all presented results and findings are based on the Kodo software library. Therefore, we claim that the implementation complexity of basic RLNC algorithms for a large number of commercially available platforms is mostly solved by Kodo [6]. However, as device hardware evolves the algorithms will also require adaptation to exploit the system in the best possible manner, e.g., use of Single Instruction Multiple Data (SIMD) on Intel devices can speed up large finite field computation.
- The choice of coding parameters, on the other side, depends heavily on the application and network topology. A key objective of the paper is to introduce to the reader the most important parameters to consider under various conditions. Often the choice of parameters differs dramatically between theoretical research and implementation.

Besides dealing with implementation and coding complexity, practical adoption of network coding also requires adaptation of existing communication protocols and integration with legacy systems. This often introduces the need to deal with heterogeneous receivers, unreliable feedback, dynamic topologies and time variable delays. These aspects are discussed in more detail throughout the paper.

II. Complexity and Green Considerations

Once the performance gains of network coding over other approaches are well understood from a research perspective, the question arises of how complex network coding really is and how does this impact the platform it is running on. Understanding this complexity is important to derive the applicability to existing platforms, for which the complexity that can be added is limited due to cost or available resources. Also, additional computation/complexity is directly linked to increased energy consumption to carry out those operations. Since network coding has the potential to reduce the energy consumption by reducing the number of transmissions in some application fields, there is a clear trade–off between the complexity we want to invest into the network coding and the potential energy gain we get by reducing transmissions. The complexity of network coding is dominated by three main parameters, namely, the generation size, the field size and the sparsity of the network codes. In the following, we explain those parameters in detail.

A. Generation Size

The generation size constitutes how many packets are coded together. This has a direct impact on the application and it is thus predefined in some scenarios. For example, streaming live events would need very small generation sizes to reduce delay, while file downloads would even work with very large generation sizes. In theory, from a throughput perspective, larger generation sizes are preferable. However, network coding in the real world is less efficient for large generation sizes, since its complexity increases cubically with the generation size if Gaussian elimination is used. Thus, the generation size is one of the key parameters that determines complexity. Creating packet overlaps in the generation constructions has been shown as an interesting way of combating the complexity growth [7],[8], while providing throughput performance equivalent to larger generation sizes. In addition, alternative approaches using sliding window coding or online network coding, a feature unique to network coding, has shown promising results [9].

B. Field Size

The field size refers to the representation of the coding coefficients. The larger the field size is, the lower is the risk of producing linearly dependent packets. Linearly dependent packets would be redundant for a receiver and should therefore be avoided. In practice, a general rule of thumb is that the implementation complexity increases with an increasing field size. Large field sizes often result in slower implementations, increasing the field size more than necessary is therefore unwanted. The use of larger fields also comes at the cost of additional overhead to signal the coding coefficients used to both intermediate nodes and the end receiver. This overhead reduces efficiency and pushes designs to smaller field sizes and moderate or small generation sizes.

C. Sparse network codes

Sparse network coding is an interesting research topic to tackle complexity. The main idea is to make matrix operations simpler by making a large number of coding coefficients zero, i.e., mixing only a few packets in a coded packet. Although it may be counterproductive, if done carefully, the overall complexity can be reduced while maintaining performance.

Although tailored algorithms may reap higher benefits, they may not be necessary to achieve high performance. Fig. 1 shows the processing speed of a simple Gaussian Elimination algorithm implemented in Kodo [6], [10] and ran on a Sony Xperia mobile phone. We observe that order of magnitude gains are possible with respect to dense codes. Of course, processing speed is only part of the problem. Making packets less dense also makes them more likely to be linearly dependent from each other, which translates in a higher number of coded packets received before decoding. To illustrate this, Fig. 2 shows the number of received coded packets for a generation size of 48 packets using sparse codes with $GF(2)$. Density is here the mean number of non–zero coefficients, i, divided by the generation size.

Fig. 2 shows that an $i \geq 5$ is sufficient to have a mean performance that is close to dense scenarios, e.g., RLNC, even when the worst case scenario can be larger. After $i \geq 10$ for $GF(2)$, even the worst case performance is comparable

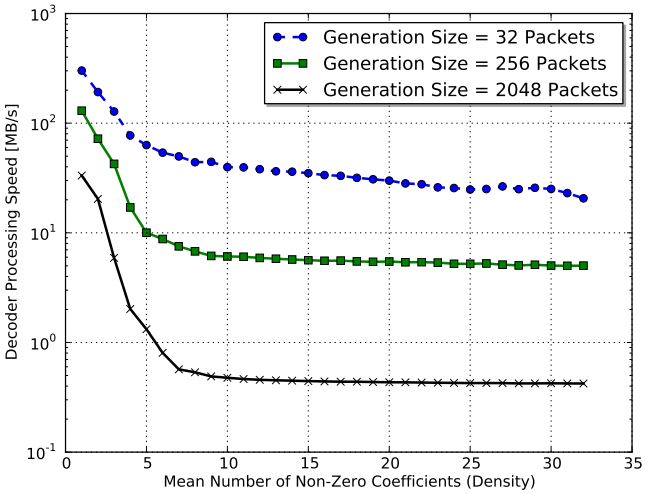

Fig. 1. Processing speed of encoding and decoding (MB/s) versus mean number of non–zero coefficients for a Sony Xperia mobile phone for different generation sizes using $GF(2)$.

to non–sparse RLNC. This performance trend is preserved for a large regime of generation sizes (from tens to several thousand packets) and shares characteristics given by i. This is encouraging as Fig. 1 already predicts high gains for $i \leq 7$, and moderate gains for $i \leq 10$.

Tunable sparse network coding [11] is promising technique to allow time–varying density distributions, which contrasts with the specific, fixed density distribution of end to end rateless codes, e.g., LT codes [12]. The key is that most coded packets (even very sparse ones) have a high probability of being linearly independent in the beginning of the transmission. Thus, exploiting a very sparse structure at the beginning will benefit from fast, inexpensive decoding. As the receivers collect more linear combinations, density is increased slowly to preserve the processing speed benefits but maintaining a low probability of generating linearly dependent packets. If 90% or more of the transmission can be carried out at speeds that are an order of magnitude faster than dense decoding speeds, this translates in order of magnitude gains in processing speed without compromising delay performance. Of course, this comes at a price: additional feedback. Investigating the amount and type of feedback is crucial to enable this technique.

Another challenge of sparse network codes is how to control recoding to preserve a reasonable density level. Conventional wisdom considers that recoding sparse packets naturally results in denser packets. This may be true if intermediate nodes combine all data packets in their buffers. However, this may be denying the possibility of simple, but effective protocols that control how much recoding occurs in each coded packet.

III. Processing in the Real World

Processing speed in the real world has a large dependence on the specific platform and software implementation. Although there is a link between theoretical measures of complexity, e.g., number of operations to perform encoding/decoding, and actual processing speed, the former is not a direct indication of the latter. The former typically allows us to predict scaling behaviour, e.g., what happens when the number of packets combined is large, but this is hardly the best indicator in more practical settings. The underlying hardware

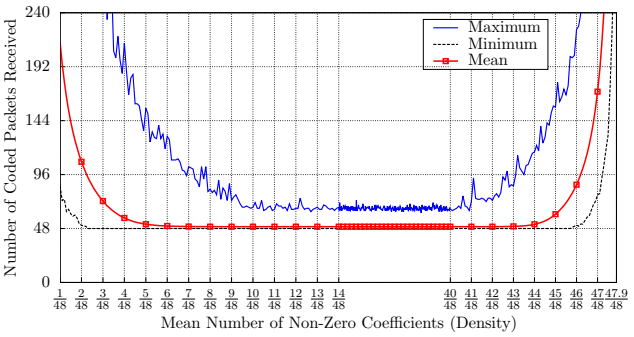

Fig. 2. Mean number of packets transmitted for a generation of size 48 packets for different densities using $GF(2)$. Best and worst case occurrences are also reported for runs with $100,000$ generations.

structure, including cache and memory management is a critical issue which is not usually incorporated as assumptions in the design of algorithms. This can translate in theoretically efficient algorithms performing poorly in practice.

One of the key aspects of designing efficient systems and algorithms, is a basic understanding of how modern day computers are designed. The traditional single CPU, storage and I/O (e.g., network cards, graphic cards) is no longer an accurate depiction of what a computer looks like. Nowadays, a computer consists of a number of cores (CPUs) connected via a number of memory caches to the main memory and I/O devices. Thus, efficiently utilizing the hardware of a modern computer requires algorithms to run on multiple cores and optimizing memory usage to efficiently take advantage of caching.

A. Cache and Memory

Typical applications using network coding have focused on transmissions over wireless systems, where the data packet restriction comes from the Maximum Transmission Units (MTU), e.g., roughly 1500 B in 802.11 and all Ethernet. However, this need not be the case for a variety of other applications. For example, cloud storage applications could benefit from larger packet sizes to be able to reap the benefits of network coding while (i) requiring less overhead to store coefficients, (ii) allowing the system to operate on smaller generation sizes (faster decoding/encoding), and (iii) allowing a cloud user to make less data requests to the cloud to gather the data. The latter has the added advantage of reducing the download time since each block of data requested (irrespective of the size) needs to invest some time to request the appropriate data of the cloud. Larger data blocks offset this request time.

Of course, these benefits may be negated by the processing time of this larger data blocks, particularly when we consider the effect of cache misses for data blocks that are of the same order (or larger) than the available cache. Fig. 3 shows the performance of a Samsung S3 mobile phone and a MAC OS Laptop for a fixed generation size of 16 packets using different packet sizes from 200 B to 1.6 MB. The MAC Laptop has 256 KB of L2 cache and 3 MB of L3 cache. We observe that best performance point occurs at 6.4 KB per packet in each generation, i.e., a total of $16 \cdot 6.4 \approx 102.4$ KB. Until packet sizes are below 12.8 KB (total of around 204 KB of data for that generation) a similar performance is maintained. This is expected as the number of L2 cache misses will be

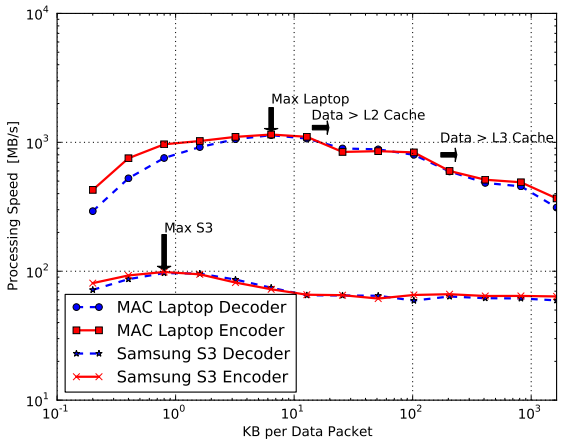

Fig. 3. Processing speed of encoding and decoding (MB/s) versus data packet size (KB) for a Samsung S3 mobile phone and a MAC OS Laptop. Generation size is 16 packets and random linear network coding in $GF(2)$.

small and the data block is of the same order of the cache. However, a first performance drop occurs after this point. More specifically, the processing speed for data packets of 25.6, 51.2, and 102.4 KB is roughly 75% of the maximum processing speed and quite stable for the different values. The total memory occupied by these values is 409.6, 819.2, and 1638 KB, respectively, which are values that can be supported by the L3 cache. A new decrease occurs for higher packet sizes (204.8 KB and above), which corresponds to total data values higher than the L3 cache and with a processing speed that is 50% or less with respect to the top processing speed for the laptop. As seen here the performance penalty of accessing data further down the memory hierarchy can be significant. As an example in the Intel Core i7 cache hierarchy accessing the L3 cache due to a cache miss is up to 10 times slower than accessing the L1 cache [13].

It is clear that (memory) size does matter in terms of processing speed. However, there are other, perhaps more relevant factors. A key issue is related to how memory is accessed and how packets or coded packets are been stored. A cache typically loads data from memory that is close to each other. Thus, it works very efficiently with algorithms that access data in memory sequentially and implementations that have allocated a continuous memory block for it. Our example of a simple Gaussian elimination algorithm uses this intuition, which makes the performance loss smaller. To illustrate this, consider that once a packet is linearly combined with others, the product and summation processes will run for every symbol of the packet. Particularly for RLNC, a larger number of the packets participate in each linear combination. Performance can drop significantly if applications allocate memory for each coded packet independently from others, which will increase the probability of having a cache miss.

However, an algorithm that uses a fully random memory access can also be equally detrimental. For example, we have seen a performance decrease in algorithms that create generations but that create a random overlapping of those generations with a uniform distribution [14]. Although conceptually these type of randomness is a good idea, from a practical perspective

it can become a show stopper, negating the processing benefits seeked by overlapping generations. More structured overlapping techniques (although still random), that promote a more sequential overlapping are more promising from a processing speed perspective. This lesson should be remembered also for the design of sparse codes, particularly if the number of packets (more precisely, the overall size of the data file) is large with respect to the cache size. Performance degradation could be worse than in non–sparse RLNC, because of the worse spatial locality of the memory access.

B. Harnessing Multiple Cores

As Herb Sutter indicated in 2005, the free lunch is over meaning that the clock frequency of a single core was at it's limit and that concurrency is needed to improve performance [15]. Interestingly, another of the key aspects that was pointed out in the article was that cache would continue to provide additional benefits for processing. However, the rate at which memory improvements occur are also slower and more expensive. This poses a key challenge for network coding as awareness of these limitations and underlying hardware structures becomes critical to provide fast and efficient coding and decoding algorithms.

Ultimately, designing parallelizable decoding algorithms is critical to speed up encoding, recoding, and decoding. However, parallel processing does not need to be complex in the beginning. Let us consider the example of cloud data storage where large packets were used to maintain small generation sizes (higher processing speed) while processing large amounts of data and still providing gains of network coding in terms of resilience and simplified coordination. The system could create smaller chunks of the coded data packets and carry out several decoding operations in parallel using the same coding coefficients. This simple technique allows us to process large packets while maintaining a reasonable amount of data in the cache and exploiting multiple cores simultaneously by multiple threads in the code, each managing a decoder that processes of a part of the packets. If done appropriately, limited or no blocking is needed between threads to reap these benefits. For example, a naive method could assign a fixed number of chunks from the start to each thread.

IV. Protocol Considerations

There is a marked difference in concept and feedback requirements between standard routing protocol design and efficient network coding design. This problem is complicated by the fact that usual assumptions in network coding research may not be directly applicable in practice. For example, feedback is no longer reliable or immediate. Although recent work has considered these effects, e.g., [16], there are additional effects in practice, e.g., time–varying delays due to medium access control (MAC) and to buffering in the protocol stack, that can have important effects on the overall performance. Finally, both protocol and code design are complicated by the fact that devices are no longer homogeneous in their capabilities, in their channels, or even in the energy available to them, e.g., remaining battery of a mobile device or sensor. Most current approaches assume that the system chooses a single code configuration, i.e., an one–size–fits–all approach, but this can be detrimental to network coding systems allowing for little adaptability.

A. Delay and Feedback

Practical systems use a layered design approach using buffers in each layer to store incoming and outgoing packets. This introduces additional delay on packet transmissions but can be a considerable challenge for NC systems. Many NC designs are rateless in nature, i.e., the source transmits until the destinations gather enough packets to decode and signal that decoding event. This poses a key challenge as the local transmission buffer could be filled with coded packets at the time the receivers have acknowledged reception, translating into a long period of transmission of redundant coded packets. The larger the buffers, the larger the performance hit. A related effect, called bufferbloat, has been reported for standard systems. In standard systems, bufferbloat occurs when the transmission link becomes a bottleneck causing the over–sized buffers to be filled [17]. In network coding, bufferbloat could happen due to a lack of proper redundancy control in rateless schemes. Credit–based schemes, e.g., [18], constitute a promising solution to avoid bufferbloat. They essentially provide a node with a budget for generation and transmission of coded packets from its buffer every time a new linearly independent packet is received, where the budget is calculated based on channel characteristics (e.g., loss probability of a packet). However, appropriate and timely feedback as well as a judicious calculation of the budget is required to guarantee low delay and high throughput in the system.

A related problem occurs when the system limits the generation size and the Round Trip Delay (RTD) is comparable or larger than the transmission of the generation, a single active generation is not sufficient to provide the best throughput benefits. A multi–generation design may be more suited, but it imposes additional signaling challenges to the system. The problem described above can also affect online network coding based systems [19], which may need a limit on the number of packets involved in the combinations before closing a window in order to maintain a reasonable complexity and per packet delay.

B. Heterogeneous Receivers

In practical networks it is common to see a wide variety to devices with widely different hardware characteristics in terms of CPU, memory, screen sizes etc. The support of heterogeneous devices is addressed already by several technology solutions such as Scalable Video Coding (SVC). In order to not destroy the newly introduced capabilities, structure is needed in choosing the coding coefficients (e.g. as seen in [20]).

C. Network Support

An issue that is typically ignored in NC research is the fact that different applications will share the same network. Currently, each new proposed solution and application is usually considered independently from others and its parameters, e.g., field size, are chosen to improve that application's performance. From a practical perspective, this can create a tremendous burden on the network devices. In the worst case, devices would need to support different configurations for each application or data flow, e.g., different field sizes for the underlying arithmetic, to achieve a target performance. Supporting disparate configurations translates into high costs in hardware, firmware, and/or software. In computationally

constrained devices, e.g., sensors, the support for encoding, recoding, or decoding in higher fields is prohibitive due to the large processing effort required. On the other end of the spectrum, computationally powerful devices may also be unable to support multiple configurations. For example, high-load, high-speed Internet routers would require deep packet inspection to determine the coding configuration, followed by a different treatment of each incoming packet. This translates into additional expensive hardware to provide high-processing speeds. Providing simple solutions that allow flows with different characteristics without significantly compromising performance is of paramount importance to make NC a reality and should be a focus of research in the near future.

V. Where to use NC

Even though some might consider network coding as the Holy Grail for all kind of setups, there are situations where network coding will not perform better than other schemes, but at least perform equally well. In the following, we give some examples illustrating where to use network coding and where not. In case of point to point or point to multi point communication, network coding will perform as good as any other advanced coding scheme, but no further improvement can be expected. Nevertheless, in some point to point scenarios, online network coding can provide benefits for applications such as coded TCP or video streaming. For multi–path communication, network coding will in general provide gains if there are losses, the feedback is delayed or the capacity of the links is varying. If none of those three occur, no gain can be expected. In the future, we will see these applications. For example, channel bundling of LTE and WiFi for mobile devices to serve small cell scenarios. Since both links will have different characteristics in terms of packet loss probabilities and feedback delay, gains are expected using network coding.

Multi hop networks will gain by network coding as long as there are losses. The recoding capabilities of the network coding ensure that the end to end delay and redundancy used on each link is minimized – but again no losses, no gain. In cooperative settings, e.g. wireless meshed networks, recoding is used with network coding providing gains as long as there are losses or there is no immediate and loss free feedback. Scheduling would be equally good if we would have perfect knowledge among all involved communication nodes (e.g. index coding or simple relaying topologies). In general, this is a very strong use case for network coding as it takes away the need for overwhelming signaling among communication nodes and losses are generally present in such scenarios. For multi–source and multi–destination meshed networks (e.g. distributed storage) network coding will provide always a gain as long as there are dynamic changes in the network (change of routes, change of route characteristics, failure or adding of source and/or destination).

VI. Conclusions

This paper presented some designing rules for network coding in the real world. In particular, we focus on what designs, protocols, and network topologies are best suited for the real world, i.e., designs that are simple enough to be deployable and yet capable of reaping the benefits of network coding. For this reason, we provide details on real hardware and system constraints to motivate more deeply these choices. Some of the conclusions are counter–intuitive: 1) small generation sizes are preferable to larger generations, 2) simple implementations of Gaussian elimination may perform order of magnitudes faster for sparse codes, 3) sparse coding may perform just as well as RLNC with potential processing speed advantages, and 4) purely rateless schemes without some rate control may be wasteful in terms of system throughput due to the presence of buffers in the protocol stack. 5.) in practice NC algorithms have to take into account the platform they are running on to achieve optimal performance.

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] T. Ho, M. Medard, R. Koetter, D. R. 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, 2006.

[3] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: practical wireless network coding," *IEEE/ACM Trans. on Netw.*, vol. 16, no. 3, pp. 497–510, 2006.

[4] 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.

[5] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *Conf. on App., tech., archi., and prot. for comp. comm. (SIGCOMM)*. New York, NY, USA: ACM, 2007, pp. 169–180.

[6] Steinwurf ApS. (2012) Kodo git repository on github. [Online]. Available: http://github.com/steinwurf/kodo

[7] D. Silva, W. Zeng, and F. R. Kschischang, "Sparse network coding with overlapping classes," *CoRR*, vol. abs/0905.2796, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr0905.html#abs-0905-2796

[8] Y. Li, E. Soljanin, and P. Spasojevic, "Effects of the generation size and overlap on throughput and complexity in randomized linear network coding," *IEEE Trans. Inf. Theor.*, vol. 57, no. 2, pp. 1111–1123, Feb. 2011. [Online]. Available: http://dx.doi.org/10.1109/TIT.2010.2095111

[9] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets tcp: Theory and implementation." *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490–512, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/pieee/pieee99.html#SundararajanSMJMB11

[10] 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.

[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] M. Luby, "Lt codes," *Proc. of the ASFCS*, Nov. 2002.

[13] R. E. Bryant and D. R. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 2nd ed. USA: Addison-Wesley Publishing Company, 2010.

[14] Y. Li, E. Soljanin, and P. Spasojevic, "Collecting coded coupons over overlapping generations," *CoRR*, vol. abs/1002.1407, 2010. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1002.html#abs-1002-1407

[15] H. Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software," *Dr. Dobb's Journal*, vol. 30, no. 3, 2005.

[16] 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.

[17] J. Gettys, "Bufferbloat: Dark buffers in the internet," *Internet Computing, IEEE*, vol. 15, no. 3, pp. 96–96, 2011.

[18] D. Koutsonikolas, W. Chih-Chun, and Y. C. Hu, "CCACK: Efficient Network Coding Based Opportunistic Routing Through Cumulative Coded Acknowledgments," in *IEEE INFOCOM*, 2010, pp. 1–9.

[19] J. K. Sundararajan, D. Shah, and M. Medard, "Arq for network coding," in *IEEE. Int. Symp. on Info. Theory (ISIT)*, 2008, pp. 1651–1655.

[20] J. Krigslund, F. Fitzek, and M. Pedersen, "On the Combination of Multi-Layer Source Coding and Network Coding for Wireless Networks," in *2013 IEEE 18th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2013.