Random Linear Network Coding: Use cases and Implementations

Niklas Haas
niklas.haas@uni-ulm.de

Abstract—We establish a basic introduction to random linear network coding and briefly touch on the ways it can be used to improve point-to-point communication, one-to-many broadcast, multicast, content delivery, peer-to-peer mesh networks and more. We also investigate and briefly summarize some examples and evaluations of random linear network coding being used in the wild, including libraries and actual products. Finally, we touch on some implementation difficulties, current limitations as well as avenues for further research.

1 Introduction

NETWORK coding is a class of algorithms and techniques that has been generating interest throughout most of the 21st century due to its interesting properties. It was originally researched by Ahlswede et al. as a way to bypass fundamental limitations of routing-based networks [3]. In this context, routing refers to any technique in which data is transmitted throughout a network by sending exact copies of individual packets between points. In contrast to this, network coding permits nodes to modify the packets as they're sending them — in particular, to create linear combinations thereof. In some sense, network coding is a generalization of routing, but the techniques involved are very different.

2 Network Coding

For a simple illustration of routing's limitations, the canonical example is to consider a butterfly network (see figure 1). In this network, two sources have pieces of information A and B (respectively), and two sinks both want A *and* B. When only routing, connections on the common path shared by both sources and both sinks (here labelled $A + B$) can either transmit A (benefiting the first source) or B (benefiting the second source) at a time, but never both. With network coding, they can instead transmit a linear combination of both A and B (in this example $A + B$) at the same time. The two receiving ends can then reconstruct A from $A + B$ and B (or B from $A + B$ and A) by calculating the linear difference $(A+B)-B$. As a result, it's as if these common paths were used by both sources simultaneously, each independent of the other. In general, network coding can achieve the maximum possible information flow in any multicast scenario as long as the linear combinations are picked correctly [3].

In addition to maximizing information flow for multicast networks, network coding can also be used to minimize the amount of overhead needed to compensate for packet loss in individual links. For example, when transmitting pieces of information A, B and C from one node to another, the first node could first send A, B and C, then $A + B$, $B + C$, $A + B$, then $A + B + C$ (stopping when the second node verifies successful delivery). If, for example, B gets dropped, then it can be recovered from A, C and $A + B$. This process is unique in that the sender does not need to know which packets the second node is missing in order to continue sending additional redundant information.

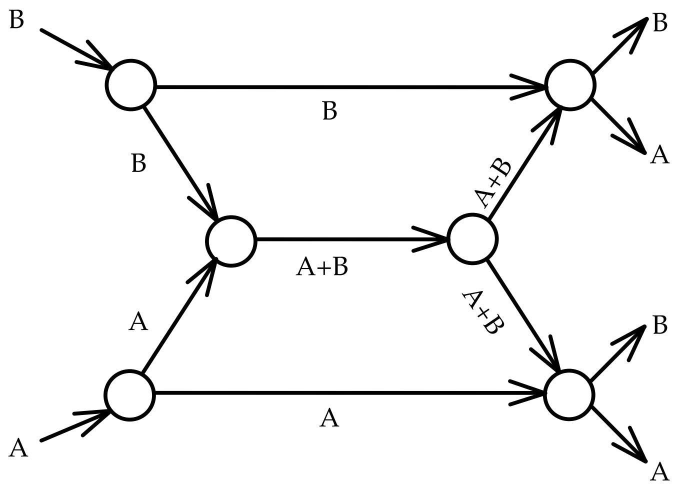

Figure 1: Butterfly network [3].

One deficiency in this example can be seen when considering the case of C being dropped: The packet $A + B$ would be fully redundant with information that the receiver already has, and is therefore wasted. This is a result of the fact that $A+B$ is a linear combination of A and B. For network coding to achieve optimal efficiency, therefore, all packets should be linearly independent from every previously received packet (or as close to this as possible). For this example, such a packet sequence could be A, B, C, $A+B+C$, but after this it's no longer possible to guarantee that property. (Which of the three remaining possible packets $A + B$, $B + C$ and $A + C$ is linearly independent of the two already received packets depends on the exact packets that were received)

The limitation arises here because these examples were effectively only considering vectors over $GF(2)$, the field of single bits. To improve the process, one naturally has to extend the field size to larger finite fields, which improves the number of linearly independent vectors one can code in sequence in exchange for reducing the processing efficiency (since restoring the original packets A, B and C requires Gauss-Jordan elimination over the chosen finite field).

Another deficiency arises from the fact that the receiving end has to know which of the vectors a received packet actually corresponds to, so at least some additional metadata needs to be transmitted to indicate the sequence number or linear coefficients used.

2.1 Random Linear Network Coding

Since agreeing on an exact set of coefficients to send each packet with is difficult if not outright impossible (when considering multicast networks with incomplete knowledge), a good way to probabilistically approximate an optimal result is to simply pick the mixing coefficients randomly. The probability of successful decoding improves exponentially as the field size increases [17], making e.g. $GF(2^8)$, the field of 8-bit vectors, more than sufficient for real-world usage. Network coding with random linear coefficients is known as *random linear network coding (RLNC)*.

2.2 Terminology

When using network coding in the real world, several parameters and definitions need to be fixed. An arbitrarily long stream of information S is split up into individual *symbols* $S_0, S_1, \ldots$ of a fixed size (known as the symbol size). For example, this symbol size could be roughly the MTU (Maximum Transfer Unit) of an individual packet in a network.[1] It is expected that each symbol is either delivered, or not delivered — but never delivered only partially.

In the simplest technique, a fixed number of these symbols $S_0, S_1, \ldots S_G$ are grouped into a so-called *generation* of size G (the generation size), which will be transmitted until completion before moving on to the next generation. Linear combinations of symbols are made only within each generation, and once G linearly independent vectors from a generation have been received, the entire generation can be restored as a unit using Gauss-Jordan elimination. In theory a generation could be large enough to cover an entire file that needs to be transmitted, but for latency and computational efficiency reasons this is often undesirable.

1. Note that additional room needs to be made for coding the linear coefficients, so the calculation of an optimal symbol size based on the MTU needs to take these into account.

Finally, the field size itself needs to be known. Typically this is $GF(2^8)$, which fits into a byte on common computer platforms and is small enough to be realizable efficiently but large enough to provide realistic chances of picking independent vectors at random.

3 Scenarios

RLNC is a versatile technology that can be used in a number of different real-world usage scenarios. This section seeks to cover some of the most important examples and how RLNC could be used to improve them.

3.1 Reliable Point-to-point Communication

A mere point-to-point link is already enough to show the benefits of (random or deterministic) linear network coding. If a single packet gets lost during transmission, its contents can be reconstructed on the receiving side as long as a sufficient number of other, linearly independent packets within the same generation have been received. What this means for transmission is that the sender merely needs to keep on sending (ideally unique) linear combinations of the generation until the receiver has received enough to reconstruct everything [16].

In this scheme there is no pre-determined rate of redundancy during transmission — rather, it emerges as a dynamic property of how many coded packets had to be sent for the receiver to recover everything. As a consequence, the level of redundancy will dynamically grow to meet the needs of the link. A connection with next to no packet loss will impose almost no overhead, whereas a connection with 10% packet loss would require something on the order of 10% additional transmissions of random linear combinations. RLNC can therefore use the hardware almost optimally without requiring advance knowledge of its reliability.[2]

In addition to this, in a point-to-point connection, a RLNC implementation can start transmitting each generation in *systematic* mode, where every packet gets sent in sequence and uncoded [15]. Since the receiving end will have received no packets to begin with, every received packet will be linearly independent. Once every original packet has been transmitted once, the sender can then start sending random linear combinations of them as usual.

3.2 One-to-many Broadcast

Multicast and broadcast problems are a good example of RLNC's advantages over routing with TCP. If one source needs to transmit an identical chunk of data to arbitrarily many clients, each of which have some small amount of packet loss, then RLNC's necessary redundant overhead

2. Unnecessary overhead can still be incurred due to the nonzero probability that a randomly chosen vector will be a linear combination of already received vectors, and therefore provide no information. As the field size and generation length increases, this probability approaches 0, at the cost of computational expense.

is determined by the maximum of the individual error rates, rather than their sum [26].

This is due to the fact that any sufficient number of linearly independent packets can be used to reconstruct a generation, regardless of which other packets may have been dropped. As such, simply sending additional linear combinations will simultaneously correct the errors for *all* clients that have experienced packet loss, without the source needing to know anything about the clients. This is especially important in environments where a single source can broadcast the same packet to multiple clients at the same time (for example, a wireless network or a grid network with multicast). In contrast, TCP requires the sender to re-broadcast a different symbol for every client, and also require the clients to use relatively expensive signalling in order to notify the sender about exactly which packets arrived and which didn't.

In practical terms, when only best-effort delivery is required (for example streaming live video at events), an error rate can be estimated at the source and plugged into the RLNC implementation. That level of redundancy will then be automatically sent in advance by the source without needing feedback from individual clients (a technique known as *forward error correction*). Depending on the chosen error correction rate, integrity of the generation can be ensured on a satisfactory fraction of the clients.

3.3 Large-scale Content Distribution

A variant of the broadcast scenario dominates the World Wide Web in the form of *content delivery networks (CDNs)*. A large CDN consists of many different servers, each of which have copies of a given file, and many different clients, each of which request said file. Classically, all clients exist independently of one another, and each receive a full copy of the file from exactly one of the sources. Horizontal bandwidth (amongst clients) as well as extra bandwidth from additional links to different servers are wasted. RLNC permits the use of these additional links to dramatically reduce the number of overall packets that need to be sent at the source [15].

Some CDNs have already started using existing peer-to-peer file-sharing protocols such as BitTorrent for this purpose [4]. While providing similar benefits to RLNC, BitTorrent suffers from multiple drawbacks which decrease its performance [12]. In particular, BitTorrent peers need to make a good decision about what chunks of a file to request from where — because different chunks have different levels of global representation. The status quo is to use a "local rarest" policy, in the hopes of this representing chunks that are also rare globally. But this can, for example, cause unnecessary strain on the source as multiple different peers suddenly try requesting the same rare chunks from the only peer that has them [12]. In addition to this, files can be left "dead" if a single chunk is missing from every single peer. Even though many different peers might store many different copies of the same few chunks, they cannot use these chunks to reconstruct the missing chunk amongst themselves.

RLNC solves the majority of these problems [12], in part due to the fact that the set of possible coded symbols is very large (which makes duplicate information rare), and in part due to the fact that clients do not need to estimate global symbol rarity in order to make decisions about what symbols to send to their peers — as long as the field size is high enough, almost any random linear combination will suffice.

Note that the claimed improvement over BitTorrent has been contested [8], and improvements to the BitTorrent algorithm cover the same deficiencies (which is part of the reason why BitTorrent still works well and is used widely for file sharing at extremely large scales in 2016).

Content delivery networks for video streaming (such as Netflix and YouTube) have been making up the vast majority of bandwidth on the Internet since at least 2010 [24], according to one study by the networking firm Sandvine. Presumably, the majority of this bandwidth is streaming the same few episodes and movies to millions of people simultaneously — in other words, the majority of the traffic on the Internet is redundant and highly centralized. By using RLNC (or other peer-to-peer networking technologies) to fully utilize the Internet's connectivity, the effective use of all this bandwidth could be dramatically improved in practice, raising consumer network speeds as well as reducing costs.

3.4 Ad-hoc Meshed Networks

Most traditional network infrastructures are highly tree-shaped. Clients are ordered into hierarchies, and endpoints typically connect to an "upstream" gateway that is responsible for delivering content to the rest of the network. Central traffic is dominated by relatively few, large exchanges and backbones connecting them. In contrast to this, a mesh network has a mostly grid-like topology: Instead of being ordered into hierarchies, peers homogeneously connect with their neighbours in every part of the network. Such topologies are especially prevalent in off-the-grid ad-hoc networks (e.g. Project Byzantium [14]) or virtual private networks that piggyback on top of existing infrastructure (e.g. I2P [2] or cjdns [5]).

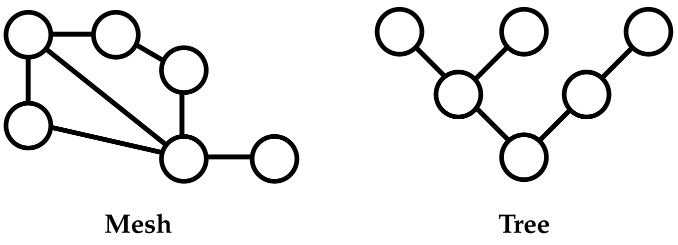

Figure 2: Mesh network topology comparison [20].

In a mesh network, any two hosts are typically connected by a very high number of different paths. Current mesh networks rely on point-to-point communication among every hop, which at best means that the overall bandwidth between clients is that of the best single path between them. When using RLNC, however, multiple distinct paths can be combined to form a single, large path that has a much higher overall bandwidth. To realize this, each hop along the way needs to (randomly) recode previously received packets from a generation and send them out to other peers (in the right direction). As many different linearly independent packets arrive at the receiver from different paths, the receiver will receive the minimum number of required packets to decode a generation much more quickly than when relying on individual paths alone [11].

A limited version of the same principle can also be applied on top of traditional network infrastructure: If a host has multiple different uplinks (for example, a phone with both LTE and Wi-Fi connections, or a phone located at the edge of two LTE cells) or two hosts combine together to "share" their uplinks (e.g. two cooperating neighbours), the connections can be used in combination to transmit the net total of their bandwidths in a reliable way, that is independent of their relative bandwidths or degrees of packet loss [22].

3.5 Distributed Storage

The techniques from network coding can also be applied to domains that initially seem to be unrelated to network communications, such as distributed storage systems. The key principle is the same: By combining enough linearly independent versions of a file, the original file can be reproduced. By storing as many extra linear combinations as required, a distributed storage system can achieve any level of redundancy without requiring much configuration or complicated book-keeping. This makes the technique suitable for use in highly decentralized storage networks, for example peer-to-peer storage, large data centers or for storage in wireless networks. This approach allows for near-optimal trade-off between redundancy and reliability [9] because the number of redundant linear combinations can be chosen freely.

[9] also discusses a variant of linear network codes known as *minimum bandwidth regenerating (MBR)* codes, which can be used to efficiently construct new, unique linear combinations of existing storage fragments without needing to inspect every single fragment (by using a storage pattern that provides asymptotic bounds on the independence of symbols). This makes it suitable for use in bandwidth-limited environments where nodes are expected to enter and leave the network at a high rate, while still providing reliability guarantees.[3] In addition to using network coding to store redundant copies, network coding can also be used for communication inside a distributed storage system, much in the same way as in section 3.4.

3. This is in contrast to randomly picking linear coefficients, which can only provide reliability guarantees asymptotically.

4 Implementations

Currently available real-world implementations of RLNC can be roughly divided into three categories: general-purpose real-world network coding libraries, experimental real-world platforms using these libraries, and experimental academic implementations. We will briefly summarize some of the most interesting protocols, implementations and academic results.

4.1 Avalanche

Microsoft's Avalanche is a protocol for peer-to-peer file distribution originally proposed by Microsoft in [12] and later developed into a public customer technology preview called *Microsoft Secure Content Distribution (MSCD)*, which was used to distribute Microsoft Visual Studio 2008 Beta-2 [23]. This marks one of the first times RLNC has been publicly demonstrated as being a viable alternative to BitTorrent and other file distribution systems, and also validates that RLNC can largely simplify the complex block propagation scheduling issues involved in BitTorrent [6].

Microsoft has recently introduced peer-to-peer distribution of updates into Windows 10. While the exact technology used in that system is unknown and presumably proprietary in nature, it's reasonable to assume Avalanche could be involved.

4.2 R^2

R^2 is a peer-to-peer live streaming approach developed by Wang et al. which fully utilizes RLNC [31] to deliver video in incremental chunks from one source to many clients. The real-time and segmented nature of live video streaming imposes additional implementation challenges [6], since the file must be made available roughly in-order to all peers, and because the file is generated and consumed in realtime. If data arrives too late, it is discarded (and therefore a huge waste of bandwidth).

4.3 UUSee

UUSee [1] is a large-scale real-world peer-to-peer video streaming platform developed by Liu et al. which is currently operational and uses RLNC to deliver video to clients [19]. The measurement studies in [19] confirm the effectiveness of RLNC in achieving shorter initial buffering delay, faster seeks, minimization of server bandwidth usage, and highly reliable video playback.

Instead of embedding the linear coefficients in every single coded symbol, UUSee instead uses a consistent *pseudo-random number generator (PRNG)* and only transmits initial seeds where necessary. This is done in part due to UUSee's choice to use a small symbol size to minimize the amount of wasted bandwidth for redundant symbols and increase the chance of successful decoding for a given number of received symbols.

4.4 Kodo

Kodo [25] is a general-purpose network coding library developed by Steinwurf ApS. It is based on their custom library for efficient, CPU-optimized finite field arithmetic, known as *Fifi*. Kodo is cross-platform, runs on both desktop and mobile environments, and comes with high-level bindings to a number of popular programming languages. Steinwurf supplies usage documentation and examples for all of these, as well as explanations about what network coding is and how to use it. While Kodo is supplied under a research- and education-friendly license, it is not free software, and must be licensed commercially for real-world applications. This may hinder its adoption in actual products.

A collaboration between researchers from MIT, Caltech and the University of Aalborg known as Code On Technologies has evaluated Kodo against Intel's Intelligent Storage Library as well as Jerasure, which are two libraries for *Reed-Solomon erasure coding* (an alternative to RLNC popular in the area of big data storage [21]) and found it to be up to 5x faster in the context of storage area networks (SANs) [22]. Code On has also claimed a 5x improvement in video streaming bandwidth when examining a lossy Wi-Fi connection with a 3% error rate, comparing Kodo's RLNC against native TCP [22]. They say this is because RLNC did not require expensive retransmissions when packets were lost.

4.4.1 wurf.it

wurf.it [28] is a video streaming platform being developed by Steinwurf as a proof-of-concept based on Kodo. While not yet available to the public, Steinwurf has released multiple public tech demos and other examples of it in action. Using RLNC, it can be used to broadcast video streams in realtime to a high number of wireless clients simultaneously, while achieving low latency and high reliability. They intend it to be easy-to-use and compatible with most mobile devices [28].

4.5 Lava

Lava was developed by Wang et al. as a real-world testbed for evaluating the computational efficiency and network benefits of RLNC. To establish a fair comparison, they simulate a real-world network and benchmark a reference implementation of a RLNC-based live streaming protocol against a reference implementation using existing peer-to-peer technologies (similar to e.g. CoolStreaming or PPLive). They pick the field $GF(2^8)$ and vary the generation size, settling on $G = 32$. They find that picking a low generation size and very high symbol size (as high as 256KB) works best for reducing the overall amount of processing, coefficient overhead and transmission time needed to fully transmit a file [30].

They come to the conclusion that RLNC offers decreased bandwidth consumption, better performance when bandwidth capacity barely exceeds the requirements, and improved buffer consistency as peers dynamically leave and join the network. Their evaluations of the computational cost introduced by network coding suggests that decode times are very low at typical media streaming rates, and negligible when progressive Gauss-Jordan elimination (as discussed in section 5.3) is used. They also produce numerous graphs evaluating the effects of tuning RLNC's various parameters [30].

4.6 CUDA

Chu et al. have developed a GPU-accelerated implementation of Gauss-Jordan elimination using *CUDA*, which can achieve decoding performance on the order of a few hundred Mbps using contemporary graphics hardware of 2009 (which is about equivalent in processing power to the latest generation of mobile phones in 2016 [29]), a result which they demonstrate is a 1-2 order of magnitude speedup over CPU implementations [7].

This result also includes GPU-accelerated implementations of homomorphic hash functions needed for ensuring data integrity and preventing poisoning attacks (as discussed in section 5.1). They pick the field $GF(2^8)$ and generation sizes $64 \leq G \leq 256$.

5 Problems and Research Trends

While the benefits of RLNC are numerous and have been demonstrated in practice, it also (inevitably) comes with implementation challenges and open topics of current research.

5.1 Vulnerability to malicious poisoning

All peer-to-peer systems suffer from *poisoning* attacks (here also known as *jamming* or *pollution* attacks) in which a malicious peer sends out invalid or altered data. Normally such attacks are mitigated by comparing the checksums of received chunks against known checksums (e.g. from a BitTorrent's file metadata or Merkle tree root). In a system based on routing, like BitTorrent, this means that a malicious peer can at best waste some fraction of a client's download bandwidth (equal in total to its own upload bandwidth). Once that client realizes the malicious peer is sending bad data, it will typically blacklist the peer and use other connections instead.

When using RLNC, however, these attacks become a more worrying problem: Not only is it difficult to use hashes to verify integrity (since the data you receive will be a linear combination of many different chunks, which each have their own individual hashes), but also because a single bogus symbol will potentially cause the entire generation to fail decoding [6]. Additionally, if a peer is mixing a single bogus block in with other genuine blocks, the resulting block is also bogus — so an attacker's outgoing bandwidth is amplified by all its peers (and their peers, and so forth).

One possible countermeasure to this style of pollution attack can be mitigated by using a *homomorphic hash function*, which is a hash function h with the property that $h(x \oplus_F y) = h(x) \oplus_h h(y)$ for all x, y in the chosen finite field $(F, \oplus_F)$. Conventional hash functions such as SHA256 etc. do not satisfy this property. However, hash functions which do can be constructed based on modular exponentiation, since $g^{x+y} \equiv g^x + g^y \mod p$. An example of such a construction is given in [6]. These homomorphic hash functions are further sources of inefficiency in practical implementations, especially since they are typically orders of magnitude slower than conventional hash functions.

One possible approach to cope with the gross inefficiency of homomorphic hash function checking is to use *cooperative security* schemes [13] in which peers dynamically join trust groups in which hash checking is distributed, and within which peers use light-weight secure checksums to verify identity and prove cooperation.

5.2 Confidentiality and privacy

RLNC is particularly susceptible to attacks involving traffic analysis [6], and also reveals to users of a peer-to-peer system who else is using the same system to download the same file (much in the way the public BitTorrent swarm does).

To combat these, it's possible to use novel techniques such as employing *homomorphic encryption* on the coefficients of each coded symbol [10]. A network coding router can then use these encrypted coefficients, alongside inexpensive addition operations on the encrypted group, to mix new random linear combinations of the incoming coded symbols without knowing any of their decryption keys. Similar to homomorphic hash functions, homomorphic encryption functions have the property that $E(k, x \oplus_F y) = E(k, x) \oplus_E E(k, y)$. (Examples arise again in modular exponentiation, e.g. unpadded RSA or ElGamal)

RLNC can also be intrinsically extended into a cipher in its own right [18], providing message confidentiality for "free" on top of the algorithms that are already being used [32]. More work in this direction is certainly not wasted.

5.3 Performance issues and latency

Both Gauss-Jordan elimination and homomorphic hash functions are very slow to compute. Even modern devices struggle to saturate common household connections with current software implementations, while also consuming more power. However, processing overhead can be decreased in exchange for increasing bandwidth overhead by picking smaller generation sizes.[4] *Dedicated application-specific integrated circuits (ASICs)* for performing the algorithms would also help with both power usage and performance tremendously, so there are still lots of promises for future growth.

4. This bandwidth overhead comes from the fact that a smaller generation size is less likely to be decoded successfully after G random linear symbols have arrived [27], so additional symbols may need to be transmitted.

While this does not help with power efficiency or throughput limits, a way to mask the decoding time needed per generation is to do partial elimination of individual symbols as they arrive [30], in essence hiding the computational expense in the time needed to transmit the rest of the generation. Once the final symbol has arrived, if everything else has already been eliminated as much as possible, the final operation is cheap. This technique is known as *progressive decoding*.

Since every symbol in a generation is made available at the same time, the minimum latency of such a signal is bound by the time needed to transmit an entire generation. The generation size can be decreased to compensate for this, but by doing so, the field size needs to be increased at the same time (which increases the computational overhead) in order to maintain the same overall bandwidth consumption.

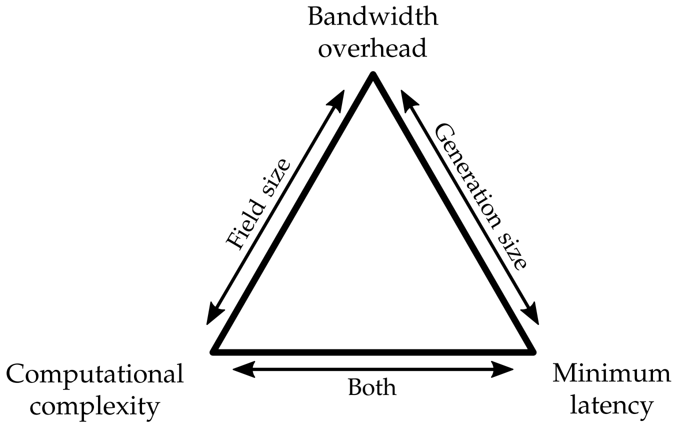

Figure 3: approximate 3-way tradeoff

In other words, there is a 3-way trade-off between computational complexity, bandwidth consumption and latency in which decreasing any parameter requires increasing one of the two others (see figure 3).

6 Conclusion

While the implementation of RLNC is not without its hurdles, we have demonstrated multiple scenarios in which RLNC could prove to be a valuable asset for improving the status quo of peer-to-peer and multicast networks, as well as a number of emerging implementations that are moving towards a realization of these goals.

Further research in this area is certainly not wasted, and experimental implementations should be encouraged and studied. File sharing networks and video streaming platforms are likely to be the first implementations of RLNC that gain popularity, but it's not unthinkable that in the future, RLNC might dominate network traffic in general.

References

[1] "UUSee," 2013. [Online]. Available: http://www.uusee.com [Accessed: 2016-06-01]

[2] "The Invisible Internet Project," 2016. [Online]. Available: https://geti2p.net/en/ [Accessed: 2016-05-06]

[3] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *Information Theory, IEEE Transactions on*, vol. 46, no. 4, pp. 1204–1216, 2000.

[4] Blizzard Entertainment, "Blizzard FAQ," 2016. [Online]. Available: http://us.blizzard.com/en-us/company/about/legal-faq.html [Accessed: 2016-05-02]

[5] Caleb James DeLisle, "cjdns," 2016. [Online]. Available: https://github.com/cjdelisle/cjdns [Accessed: 2016-05-06]

[6] X. Chu and Y. Jiang, "Random linear network coding for peer-to-peer applications," *Network, IEEE*, vol. 24, no. 4, pp. 35–39, 2010.

[7] X. Chu, K. Zhao, and M. Wang, *NETWORKING 2009: 8th International IFIP-TC 6 Networking Conference, Aachen, Germany, May 11-15, 2009. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ch. Practical Random Linear Network Coding on GPUs, pp. 573–585. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01399-7_45

[8] Cohen, Bram, "Avalanche," 2005. [Online]. Available: http://bramcohen.livejournal.com/20140.html [Accessed: 2016-06-04]

[9] A. G. Dimakis, P. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539–4551, 2010.

[10] Y. Fan, Y. Jiang, H. Zhu, and X. S. Shen, "An efficient privacy-preserving scheme against traffic analysis attacks in network coding," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 2213–2221.

[11] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: an instant primer," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 63–68, 2006.

[12] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *IEEE INFOCOM*, no. MSR-TR-2004-80, March 2005, p. 12. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=67246

[13] C. Gkantsidis, P. Rodriguez *et al.*, "Cooperative security for network coding file distribution." in *INFOCOM*, vol. 3, 2006, p. 5.

[14] HacDC, "Project Byzantium," 2016. [Online]. Available: http://project-byzantium.org/about/ [Accessed: 2016-05-06]

[15] J. Heide, M. V. Pedersen, F. H. P. Fitzek, and T. Larsen, "Network coding for mobile devices - systematic binary random rateless codes," in *2009 IEEE International Conference on Communications Workshops*, June 2009, pp. 1–6.

[16] 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 Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, Oct 2006.

[17] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," 2003.

[18] L. Lima, M. Médard, and J. Barros, "Random linear network coding: A free cipher?" in *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*. IEEE, 2007, pp. 546–550.

[19] Z. Liu, C. Wu, B. Li, and S. Zhao, "UUSee: Large-scale operational on-demand streaming," *Population*, vol. 100, no. 200, p. 3000, 2010.

[20] Maksim, "Network Topologies," 2006. [Online]. Available: https://en.wikipedia.org/wiki/File:Butterfly_network.gif [Accessed: 2016-07-02]

[21] Mulnix, David, "Intel and Qihoo 360 Internet Portal Datacenter - Big Data Storage Optimization Case Study," 2014. [Online]. Available: https://software.intel.com/en-us/articles/intel-and-qihoo-360-internet-portal-datacenter-big-data-storage-optimization-case-study [Accessed: 2016-05-27]

[22] S. M. Patterson, "How MIT and Caltech's coding breakthrough could accelerate mobile network speeds," 2014. [Online]. Available: http://www.networkworld.com/article/2342846 [Accessed: 2016-05-04]

[23] P. Rodriguez, "Avalanche is now live: Microsoft secure content distribution (mscd)," 2007.

[24] Sandvine, "Global Internet Phenomena Report." [Online]. Available: https://www.sandvine.com/trends/global-internet-phenomena/ [Accessed: 2016-06-05]

[25] Steinwurf ApS, "Kodo Network Coding Library," 2016. [Online]. Available: http://steinwurf.com/kodo/ [Accessed: 2016-05-01]

[26] ——, "Recoding Data," 2016. [Online]. Available: http://docs.steinwurf.com/kodo/kodo-cpp/tutorial/recoding_data.html [Accessed: 2016-05-01]

[27] ——, "The basic functionality of Kodo," 2016. [Online]. Available: http://docs.steinwurf.com/kodo/kodo-cpp/tutorial/the_basics.html [Accessed: 2016-06-04]

[28] ——, "wurf.it media streaming service," 2016. [Online]. Available: http://steinwurf.com/wurf-it/ [Accessed: 2016-05-01]

[29] A. Voica, "PowerVR Series7XT GPUs push graphics and compute performance to the max," 2014. [Online]. Available: http://blog.imgtec.com/powervr/powervr-series7xt-gpus-push-graphics-and-compute-performance [Accessed: 2016-06-04]

[30] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. IEEE, 2007, pp. 1082–1090.

[31] ——, "R2: Random push with random network coding in live peer-to-peer streaming," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 9, pp. 1655–1666, 2007.

[32] P. Zhang, Y. Jiang, C. Lin, Y. Fan, and X. Shen, "P-coding: secure network coding against eavesdropping attacks," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.