Resilient Network Coding in the Presence of Byzantine Adversaries

S. Jaggi jaggi@mit.edu
M. Langberg mikel@caltech.edu
S. Katti skatti@mit.edu
T. Ho tho@caltech.edu
D. Katabi dk@mit.edu
M. Médard medard@mit.edu

***Abstract*—**

Network coding substantially increases network throughput. But since it involves mixing of information inside the network, a single corrupted packet generated by a malicious node can end up contaminating all the information reaching a destination, preventing decoding.

This paper introduces *the first distributed polynomial-time rate-optimal* **network codes that work in the presence of Byzantine nodes. We present algorithms that target adversaries with different attacking capabilities. When the adversary can eavesdrop on all links and jam z_O links , our first algorithm achieves a rate of $C - 2z_O$, where C is the network capacity. In contrast, when the adversary has limited snooping capabilities, we provide algorithms that achieve the higher rate of $C - z_O$.**

Our algorithms attain the optimal rate given the strength of the adversary. They are information-theoretically secure. They operate in a distributed manner, assume no knowledge of the topology, and can be designed and implemented in polynomial-time. Furthermore, only the source and destination need to be modified; non-malicious nodes inside the network are oblivious to the presence of adversaries and implement a classical distributed network code. Finally, our algorithms work over wired and wireless networks.

I. Introduction

Network coding allows the routers to mix the information content in packets before forwarding them. This mixing has been theoretically proven to maximize network throughput [1], [17], [13]. It can be done in a distributed manner with low complexity, and is robust to packet losses and network failures [8], [21]. Furthermore, recent implementations of network coding for wired and wireless environments demonstrate its practical benefits [16], [6].

But what if the network contains malicious nodes? A malicious node may pretend to forward packets from source to destination, while in reality it injects corrupted packets into the information flow. Since network coding makes the routers mix packets' content, a single corrupted packet can end up corrupting *all* the information reaching a destination. Unless this problem is solved, network coding may perform much worse than pure forwarding in the presence of adversaries.

The interplay of network coding and Byzantine adversaries has been examined by a few recent papers. Some detect the presence of an adversary [10], others correct the errors he injects into the codes under specific conditions [7], [12], [19], and a few bound the maximum achievable rate in such adverse environments [3], [28]. But attaining optimal rates using distributed and low-complexity codes is still an open problem.

This paper designs distributed polynomial-time rate-optimal network codes that combat Byzantine adversaries. We present three algorithms that target adversaries with different strengths. The adversary can always inject z_O packets, but his listening power varies. When the adversary is omniscient, i.e., he observes transmissions on the entire network, our codes achieve the rate of $C-2z_O$, with high probability. When the adversary's knowledge is limited, either because he eavesdrops only on a subset of the links or the source and destination have a low-rate secret-channel, our algorithms deliver the higher rate of $C-z_O$.

The intuition underlying all of our algorithms is that the aggregate packets from the adversarial nodes can be thought of as a second source. The information received at the destination is a linear transform of the source's and the adversary's information. Given enough linear combinations (enough coded packets), the destination can decode both sources. The question however is how does the destination distill out the source's information from the received mixture. To do so, the source's information has to satisfy certain constraints that the attacker's data cannot satisfy. This can be done by judiciously adding redundancy at the source. For example, the source may add redundancy to ensure that certain functions evaluate to zero on the original source's data, and thus can be used to distill the source's data from the adversary's. The challenge addressed herein is to design the redundancy that achieves the optimal rates.

This paper makes several contributions. The algorithms presented herein are *the first distributed algorithms with polynomial-time complexity in design and implementation, yet are rate-optimal.* In fact, since pure forwarding is a special case of network coding, being rate-optimal, our algorithms also achieve a higher rate than any approach that does not use network coding. They assume no knowledge of the topology and work in both wired and wireless networks. Furthermore, implementing our algorithms involves only a slight modification of the source and destination while the internal nodes can continue to use standard network coding.

II. Illustrating Example

We illustrate the intuition underlying our approach using the toy example in Fig. 1. Calvin wants to prevent the flow of information from Alice to Bob, or at least minimize it. All links have a capacity of one packet per unit time. Further, Calvin connects to the three routers over a wireless link, shown in Fig. 1 as a dashed hyperedge. The network capacity, C, is by definition the min-cut from Alice to Bob. It is equal to 3 packets. The min-cut from Calvin to the destination is $z_O = 1$ packet per unit time. Hence, the maximum rate from Alice to Bob in

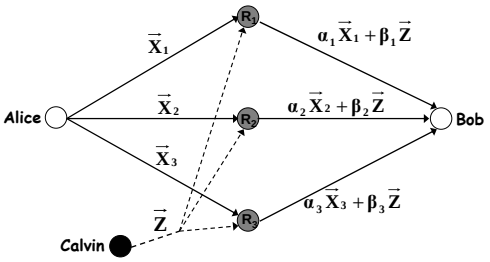

Fig. 1—A simple example. Alice transmits to Bob. Calvin injects corrupted packets into their communication. The grey nodes in the middle perform network coding.

this scenario is bounded by $C - z_O = 2$ packets per unit time as proven in [12].

We express each packet as a vector of n bytes, where n is a sufficiently large number. The routers create random linear combinations of the packets they receive. Hence, every unit of time Bob receives the packets:

$$\begin{aligned} \tilde{\mathbf{y}}_\mathbf{1} &= \alpha_1\tilde{\mathbf{x}}_\mathbf{1} + \beta_1\tilde{\mathbf{z}} \\ \tilde{\mathbf{y}}_\mathbf{2} &= \alpha_2\tilde{\mathbf{x}}_\mathbf{2} + \beta_2\tilde{\mathbf{z}} \\ \tilde{\mathbf{y}}_\mathbf{3} &= \alpha_3\tilde{\mathbf{x}}_\mathbf{3} + \beta_3\tilde{\mathbf{z}}, \end{aligned} \tag{1}$$

where $\tilde{\mathbf{x}}_\mathbf{i}$'s are the three packets Alice sent, $\tilde{\mathbf{z}}$ is the packet Calvin sent, α_i and β_i are random coefficients.

In our example, the routers operate over bytes; the i^{th} byte in an outgoing packet is a linear combination of i^{th} bytes in the incoming packets. Thus, (1) also describes the relation between the individual bytes in $\tilde{\mathbf{y}}_\mathbf{i}$'s and the corresponding bytes in $\tilde{\mathbf{x}}_\mathbf{i}$'s and $\tilde{\mathbf{z}}$.

Since the routers mix the content of the packets, Alice cannot just sign her packets and have Bob discard all packets with incorrect signatures. To decode, Bob has to somehow distill the $\tilde{\mathbf{x}}_\mathbf{i}$'s from the $\tilde{\mathbf{y}}_\mathbf{i}$'s he receives.

As a first attempt at solving the problem, let us assume that Bob knows the topology, i.e., he knows that the packets he receives are produced using (1). Further, let us assume that he knows the random coefficients used by the routers to code the packets, i.e., he knows the values of α_i's and β_i's. To decode, Bob has to solve (1). These are three equations with four unknowns $\tilde{\mathbf{z}}$, $\tilde{\mathbf{x}}_\mathbf{1}$, $\tilde{\mathbf{x}}_\mathbf{2}$ and $\tilde{\mathbf{x}}_\mathbf{3}$. Hence, Bob cannot decode.

To address the above situation, Alice needs to add redundancy to her transmitted packets. After all, as noted above, for the particular example in Fig. 1, Alice's rate is bounded by 2 packets per unit time. Thus, Alice should send no more than 2 packets worth of information. She can use the third packet for added redundancy. Suppose Alice sets

$$\tilde{\mathbf{x}}_\mathbf{3} = \tilde{\mathbf{x}}_\mathbf{1} + \tilde{\mathbf{x}}_\mathbf{2}. \tag{2}$$

This coding strategy is public to both Bob and Calvin. Combining (2) with (1), Bob obtains a system of 4 equations with 4 unknowns, which he can solve to decode.

But in the general case, Bob knows nothing about the coefficients used by the routers, the topology, or the overall network transform. To keep the example tractable, we assume that the β_i's are unknown to Bob, whereas the other coefficients are known.

Given (1) and (2), Bob is faced with 4 equations and 7 unknowns, and thus cannot decode. But note that Bob does not need to find both β_i's and $\tilde{\mathbf{z}}$; finding their product is sufficient to find $\tilde{\mathbf{x}}$. (This is because the β_is and $\vec{z}$ always appear as the product term $\beta_i\vec{z}$ in (1).) Hence he is left with 4 equations and 6 unknowns.

The first idea we use is that while $\tilde{\mathbf{z}}$ is a whole unknown packet of n bytes, each of the coefficients β_i is a single byte. Thus, instead of devoting a whole vector of n bytes for added redundancy (as in (2)), Alice just needs three extra bytes of redundancy to compensate for the β_i's being unknown.

Alice imposes constraints on her data to help Bob to decode. For instance, a simple constraint could be that the first byte in each packet equals zero. This constrain provides Bob with three additional equations. So, rewriting (1) for the first byte of each packet, Bob would get a scaled version of the β_i's i.e., they are all multiplied by z_1.

$$\begin{aligned} y_{1,1} &= \alpha_1 x_{1,1} + \beta_1 z_1 = \beta_1 z_1 \\ y_{2,1} &= \alpha_2 x_{2,1} + \beta_2 z_1 = \beta_2 z_1 \\ y_{3,1} &= \alpha_3 x_{3,1} + \beta_3 z_1 = \beta_3 z_1 \end{aligned}, \tag{3}$$

Our second observation is that the scaled version of the β_i's suffices for Bob to decode $\tilde{\mathbf{x}}$. This can be seen by a simple algebraic manipulation of (1). Bob can rewrite the equations in (1) by multiplying and dividing the second term with z_1 and appending (2) to obtain

$$\begin{aligned} \tilde{\mathbf{y}}_\mathbf{1} &= \alpha_1\tilde{\mathbf{x}}_\mathbf{1} + (\beta_1 z_1)(\tilde{\mathbf{z}}/z_1) \\ \tilde{\mathbf{y}}_\mathbf{2} &= \alpha_2\tilde{\mathbf{x}}_\mathbf{2} + (\beta_2 z_1)(\tilde{\mathbf{z}}/z_1) \\ \tilde{\mathbf{y}}_\mathbf{3} &= \alpha_3\tilde{\mathbf{x}}_\mathbf{3} + (\beta_3 z_1)(\tilde{\mathbf{z}}/z_1) \\ \tilde{\mathbf{x}}_\mathbf{3} &= \tilde{\mathbf{x}}_\mathbf{1} + \tilde{\mathbf{x}}_\mathbf{2}. \end{aligned} \tag{4}$$

Notice that Bob already knows all three $\beta_i z_1$ terms from (3). The term $(\tilde{\mathbf{z}}/z_1)$ can be considered a single unknown because Bob does not care about estimating the exact value of $\tilde{\mathbf{z}}$. Now Bob has 4 equations with 4 unknowns and they can be solved to decode as before.

One complication still remains. If Calvin knows the constraints on Alice's data, he knows that the first byte of each packet is zero. So to ensure that Bob does not obtain any information about the β_i's from (4), Calvin can just set the first byte in his packet z_1 to zero.

There are two ways out of this situation. Suppose Alice could communicate to Bob a small message that is secret from Calvin. In this case, she could compute a small number of hashes of her data, and transmit them to Bob. These hashes correspond to constraints on her data, which enables Bob to decode. If Alice cannot communicate secretly with Bob, she leverages the fact that Calvin can inject only one fake packet. Since Calvin's packet is n bytes long, he can cancel out at most n hashes. If Alice injects $n+1$ hashes, there must be at least one hash Calvin cannot cancel. This hash enables Bob to find the β_i's and decode. Notice, however, that the $n+1$ additional constraints imposed on the bytes in $\vec{x}_1$ and $\vec{x}_2$ means that Alice can only transmit at most $n-1$ bytes of data to Bob. For a large number of bytes n in a packet, this rate is asymptotically optimal against an all-knowing adversary [3].

The rest of this paper considers the general problem of network coding over completely unknown topology, in the presence of an adversary who has partial or full knowledge of the network and transmissions in it.

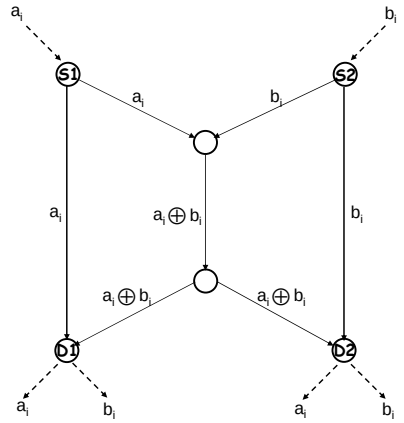

Fig. 2—A simple scenario showing how network coding improves throughput. All links have a capacity of one packet per unit of time. By sending the XOR of a_i and b_i on the middle link, we can deliver two packets per unit of time to both receivers.

III. Related Work

We start with a brief summary of network coding, followed by a survey of prior work on Byzantine adversaries in networks.

A. Network Coding Background

The idea underlying network coding is often illustrated using the famous butterfly example by Ahlswede et.al [1]. Consider the network in Fig. 2, where source S_1 wants to deliver the stream of packets a_i to both D_1 and D_2, and source S_2 wants to send the stream of packets b_i to the same two receivers. Assume all links have a capacity of one packet per unit of time. If routers only forward the packets they receive, the middle link becomes a bottleneck, which for every unit of time, can either deliver a_i to D_1 or b_i to D_2. In contrast, if the router feeding the middle link XORs the two packets and sends $a_i \oplus b_i$, as shown in the figure, both receivers obtain two distinct packets in every unit of time.

Work on network coding started with a pioneering paper by Ahlswede et al. [1], which establishes the value of coding in the routers and provides theoretical bounds on the capacity of such networks. The combination of [20], [17], [13] shows that, for multicast traffic, linear codes achieves the maximum capacity bounds, and coding and decoding can be done in polynomial time. Additionally, Ho et al. show that the above is true even when the routers pick random coefficients [8]. Researchers have extended the above results to a variety of areas including wireless networks [21], [15], [16], energy [27], secrecy [2], content distribution [6], and distributed storage [14].

B. Byzantine Adversaries in Networks

A Byzantine attacker is a malicious adversary hidden in a network, capable of eavesdropping and jamming communications. Prior research has examined these attacks in the presence of network coding and without it. In the *absence* of network coding, Dolev et al. [5] consider the problem of communicating over a known graph containing Byzantine adversaries. They show that for k adversarial nodes, reliable communication is possible only if the graph has more than $2k + 1$ vertex connectivity. Subramaniam extends this result to unknown graphs [25]. Pelc et al. address the same problem in wireless networks by modeling malicious nodes as locally bounded Byzantine faults, i.e., nodes can overhear and jam packets only in their neighborhood [23].

Scheme	Charles et.al. [4]	Jaggi et.al. [12]	Ours
Info. Theoretic Security	No	Yes	Yes
Distributed	Yes	No	Yes
Internal Node Complexity	High	Low	Low
Decoding Complexity	High	Exponential	Low
General Graphs	No	Yes	Yes
Universal	No	No	Yes

TABLE I—Comparison between the results in this paper and some prior papers.

The interplay of network coding and Byzantine adversaries was first examined in [10], which detects the existence of an adversary but does not provide an error-correction scheme. This has been followed by the work of Cai and Yeung [28], [3], who generalize standard bounds on error-correcting codes to networks, without providing any explicit algorithms for achieving these bounds. Our work presents a constructive design to achieve those bounds.

The problem of correcting errors in the presence of both network coding and Byzantine adversaries has been considered by a few prior proposals. Earlier work [19], [7] assumes a centralized trusted authority that provides hashes of the original packets to each node in the network. More recent work by Charles et al. [4] obviates the need for a trusted entity under the assumption that the majority of packets received by each node is uncorrupted. In contrast to the above two schemes which are cryptographically secure, in a previous work [12], we consider an information-theoretically rate-optimal solution to Byzantine attacks for *wired* networks, which however requires a centralized design. This paper builds on the above prior schemes to combine their desirable traits; it provides a distributed solution that is information-theoretically rate optimal and can be designed and implemented in polynomial time. Furthermore, our algorithms have new features; they assume no knowledge of the topology, do not require any new functionality at internal nodes, and work for both wired and wireless networks. Table I highlights similarities and differences from prior work.

IV. Model & Definitions

We use a general model that encompasses both wired and wireless networks. To simplify notation, we consider only the problem of communicating from a single source to a single destination. But similar to most network coding algorithms, our techniques generalize to multicast traffic.

A. Threat Model

There is a source, Alice, and a destination, Bob, who communicate over a wired or wireless network. There is also an attacker Calvin, hidden somewhere in the network. Calvin aims to prevent the transfer of information from Alice to Bob, or at least to minimize it. He can observe some of the transmissions, and can inject his own. When he injects his own packets, he pretends they are part of the information flow from Alice to Bob.

Calvin is quite strong. He is computationally unbounded. He knows the encoding and decoding schemes of Alice and Bob, and the network code implemented by the interior nodes. He also knows the exact network realization.

B. Network and Code Model

This section describes the network model, the packet format, and how the network transforms the packets.

Network Model: The network is modeled as a hypergraph [22]. Each packet transmission corresponds to a hyperedge directed from the transmitting node to the set of observer nodes. For wired networks, the hyperedge is a simple point-to-point link. For wireless, each such hyperedge is determined by instantaneous channel realizations (packets may be lost due to fading or collisions) and connects the transmitter to all nodes that hear the transmission. The hypergraph is unknown to Alice and Bob prior to transmission.

Source: Alice generates incompressible data that she wishes to deliver to Bob over the network. To do so, Alice encodes her data as dictated by the encoding algorithm (described in subsequent sections). She divides the encoded data into batches of b packets. For clarity, *we focus on the encoding and decoding of one batch.*

A packet contains a sequence of n symbols from the finite field $\mathbb{F}_q$. All arithmetic operations henceforth are done over symbols from $\mathbb{F}_q$. (see the treatment in [18]). Out of the n symbols in Alice's packet, δn symbols are redundancy added by the source.

Alice organizes the data in each batch into a matrix X as shown in Fig. 3. We denote the $(i,j)^{th}$ element in the matrix by $x(i,j)$. The i^{th} row in the matrix X is just the i^{th} packet in the batch. Fig. 3 shows that similarly to standard network codes [9], some of the redundancy in the batch is devoted to sending the identity matrix, I. Also, as in [9], Alice takes random linear combinations of the rows of X to generate her transmitted packets. As the packets traverse the network, the internal nodes apply a linear transform to the batch. The identity matrix receives the same linear transform. The destination discovers the linear relation between the packets it receives and those transmitted by inspecting how I was transformed.

Adversary: Let the matrix Z be the information Calvin injects into each batch. The size of this matrix is $z_O \times n$, where z_O is the size of the min-cut from Calvin to the destination.

Destination: Analogously to how Alice generates X, the destination Bob organizes the received packets into a matrix Y. The i^{th} received packet corresponds to the i^{th} row of Y. Note that the number of received packets, and therefore the number of rows of Y, is a variable dependent on the network topology. The column rank of eY, however, is $b + z_O$. Bob attempts to reconstruct Alice's information, X, using the matrix of received packets Y.

C. Definitions

We define the following concepts.

- The *network capacity*, denoted by C, is the time-average of the maximum number of packets that can be delivered from Alice to Bob, assuming no adversarial interference, i.e., the max flow. It can be also expressed as *the min-cut from source to destination*. (For the corresponding multicast case, C is defined as the minimum of the min-cuts over all destinations.)
- The *error probability* is the probability that Bob's reconstruction of Alice's information is inaccurate.
- The rate, R, is the number of information bits in a batch amortized by the length of a packet in bits.
- The rate R is said to be *achievable* if for any $\epsilon > 0$, any $\delta > 0$, and sufficiently large n, there exists a block-length-n network code with a redundancy δ and a probability of error less than ϵ.
- A code is said to be *universal* if the code design is independent of z_O.

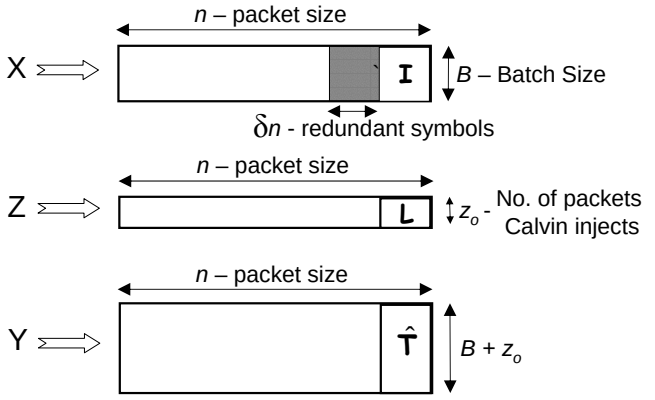

Fig. 3—Alice, Bob and Calvin's information matrices.

V. Network Transform

This section explains how Alice's packets get transformed as they travel through the network. It examines the effect the adversary has on the received packets, and Bob's decoding problem.

The network performs a classical distributed network code [9]. Specifically, each packet transmitted by an internal node is a random linear combination of its incoming packets. Thus, the effect of the network at the destination can be summarized as follows.

$$Y = TX + T_{Z\to Y}Z, \tag{5}$$

where X is the batch of packets sent by Alice, Z refers to the packets Calvin injects into Alice's batch, and Y is the received batch. The variable T refers to the linear transform from Alice to Bob, while $T_{Z\to Y}$ refers to the linear transform from Calvin to Bob.

As explained in §IV, a classical random network code's X includes the identity matrix as part of each batch. The identity matrix sent by Alice incurs the same transform as the rest of the batch. Thus,

$$\hat{T} = TI + T_{Z\to Y}L, \tag{6}$$

where $\hat{T}$ and L are the columns corresponding to I's location in Y and Z respectively, as shown in Fig. 3.

In standard network coding, there is no adversary, i.e., $Z = 0$ and $L = 0$, and thus $\hat{T} = T$. The destination receives a description of the network transform in $\hat{T}$ and can decode X as

Variable	Definition
b	Number of packets in a batch.
z_O	Number of packets Calvin can inject.
z_I	Number of packets Calvin can hear.
n	Length of each packet.
δ	Fractional redundancy introduced by Alice.
$\hat{T}$	Proxy of the transfer matrix T representing the network transform.

TABLE II—Terms used in the paper.

$\hat{T}^{-1}Y$. In the presence of the adversary, however, the destination needs to solve (5) and (6) to extract the value of X.

By substituting T from (6), (5) can be simplified to get

$$Y = \hat{T}X + T_{Z\to Y}(Z - LX) \quad (7)$$
$$= \hat{T}X + E, \quad (8)$$

where E is a $b \times n$ matrix that characterizes Calvin's interference. Note that the matrix $\hat{T}$, which Bob knows, acts as a *proxy transfer matrix* for T, which he doesn't know.

Note that in (5), all terms other than Y are unknown. Further, it is non-linear due to the cross-product terms, TX and $T_{Z\to Y}Z$. In contrast, (8) is linear in the unknowns X and E. The rest of this work focuses on solving (8) under different assumptions on Calvin's strength.

VI. Summary of Results

We have three main results. Each result corresponds to a distributed, rate-optimal, polynomial-time algorithm that defeats an adversary of a particular type. The optimality of these rates has been proven by prior work [3], [28], [12]. Our work, however, provides a construction of distributed codes/algorithms that achieve optimal rates.

(1) Shared Secret Model: This model assumes that Alice and Bob have a very low rate secret channel, the transmissions on which are unknown to Calvin. It considers the transmission of information via network coding in a network where Calvin can observe all transmissions, and can inject some corrupt packets.

Theorem 1: The Shared Secret algorithm achieves a rate of $C - z_O$ with code-complexity $\mathcal{O}(nC^2)$. This is the maximum achievable rate.

In §VII, we prove the above theorem by constructing an algorithm that achieves the bounds. Note that [7] proves a similar result for a more constrained model where Alice shares a very low rate secret channel with all nodes in the network, and the operations performed by internal nodes are computationally expensive. Further, their result guarantees cryptographic security, while we provide information-theoretic security.

(2) Omniscient Adversary Model: This model assumes an omniscient adversary, i.e., one from whom nothing is hidden. In particular, Alice and Bob have no shared secrets hidden from Calvin. It also assumes that the min-cut from the adversary to the destination, z_O, is less than $C/2$. Prior work proves that without this condition, it is impossible for the source and the destination to reliably communicate without a secret channel [12]. In §VIII, we prove the following.

Theorem 2: The Omniscient Adversary algorithm achieves a rate of $C - 2z_O$ with code-complexity $\mathcal{O}((nC)^3)$. This is the maximum achievable rate.

(3) Limited Adversary Model: In this model, Calvin is limited in his eavesdropping power; he can observe at most z_I transmitted packets. Exploiting this weakness of the adversary results in an algorithm that, like the Omniscient Adversary algorithm operates without a shared secret, but still achieves the higher rate possible via the Shared Secret algorithm. In particular, in §IX we prove the following.

Theorem 3: If $z_I < C - 2z_O$, the Limited Adversary algorithm achieves a rate of $C - z_O$ with code-complexity $\mathcal{O}(nC^2 + (n\delta)^3C^4)$. This is the maximum achievable rate.

VII. Shared Secret Model

In the Shared Secret model, Alice and Bob have use of a strong resource, namely a secret channel over which Alice can transmit a small amount of information to Bob that is secret from Calvin. Note that since the internal nodes mix corrupted and uncorrupted packets, Alice cannot just sign her packets and have Bob check the signature and throw away corrupted packets. Alice uses the secret channel to send a hash of her information X to Bob, which Bob can use to distill the corrupted packets he receives, as explained below.

Shared Secret: Alice generates her secret message in two steps. She first chooses C *parity symbols* uniformly at random from the field $\mathbb{F}_q$. The parity symbols are labeled r_d, for $d \in \{1, \ldots, C\}$. Corresponding to the parity symbols, Alice's *parity-check matrix* P is defined as the $n \times C$ matrix whose $(i,j)^{th}$ entry equals $(r_j)^i$, i.e., r_j to the i^{th} power. The second part of Alice's secret message is the $b \times C$ *hash matrix* H, computed as the matrix product XP. We assume Alice communicates both the set of parity symbols and the hash matrix H to Bob over the secret channel. The combination of these two creates the shared secret, denoted $\mathbb{S}$, between Alice and Bob. The size of $\mathbb{S}$ is $C(b+1)$ symbols, which is small in comparison to Alice's information X. (The size of X is $b \times n$; it can be made arbitrarily large compared to the size of $\mathbb{S}$ by increasing the packet size n.)

Alice's Encoder: Alice implements the classical random network encoder described in §IV-B.

Bob's Decoder: Not only is P used by Alice to generate H, but is also used by Bob in his decoding process. To be more precise, Bob computes $YP - \hat{T}H$ using the messages he gets from the network and the secret channel. We call the outcome the *syndrome matrix* S.

By substituting the value of H and using (8), we obtain

$$S = YP - \hat{T}H = (Y - \hat{T}X)P = EP. \quad (9)$$

Thus, if no adversary was present, the packets would not be corrupted (i.e., $E = 0$) and S would be an all-zero matrix. As shown in §IV, X then equals $\hat{T}^{-1}Y$. If Calvin injects corrupt packets, S will be a non-zero matrix.

Claim 1: The columns of S span the same vector-space as the columns of E.

Claim 1, proved in the Appendix, means that Calvin's interference, E, can be written as linear combinations of the columns of S, i.e., $E = AS$, where A is a $C \times n$ matrix. This enables

Bob to rewrite (8) as the matrix product

$$Y = [\hat{T}\ S] \begin{bmatrix} X \\ \hat{A} \end{bmatrix}, \tag{10}$$

Bob does not care about A, but to obtain X, he must solve (10).

Claim 2: The matrix $[\hat{T}\ S]$ has full column-rank.

Claim 2, proved in the Appendix, means that Bob can decode by simply inverting the matrix $[\hat{T}\ S]$ and multiplying the result by Y. Since Alice encodes at a rate $R = C - z_O$, the shared secret algorithm achieves the optimal rate shown by prior work [12]. Of code design, encoding and decoding, both encoding and decoding require $\mathcal{O}(nC^2)$ steps. The costliest step for Alice is the computation of the hash matrix H, and for Bob is the computation of the syndrome matrix S.

The scheme presented above is universal, i.e., the parameters of the code do not depend on any knowledge about z_O, which in some sense functions as the "noise parameter" of the network. Alice therefore has flexibility in tailoring her batch size to the size of the data which she wishes to transmit and the packet size allowed by the network. □

VIII. Omniscient Adversary Model

What if we face an *omniscient adversary*, i.e., Calvin can observe everything, and there are no shared secrets between Alice and Bob? We design a network error-correcting code to defeat such a powerful adversary. Our algorithm achieves a rate of $R = C - 2z_O$, which is lower than in the Shared Secret model. This is a direct consequence of Calvin's increased strength. Recent bounds [3] on network error-correcting codes show that in fact $C - 2z_O$ is the maximum achievable rate for networks with an omniscient adversary.

Alice's Encoder: Alice encodes in two steps. To counter the adversary's interference, she first generates X by adding redundancy to her information. She then encodes X using the encoder defined in §IV-B.

Alice adds redundancy as follows. Her original information is a length-$(bn - \delta n - b^2)$ column vector $\tilde{\mathbf{U}}$. (Here the fractional redundancy δ, is dependent on z_O, the number of packets Calvin may inject into the network.) Alice converts $\tilde{\mathbf{U}}$ into $\tilde{\mathbf{X}}$, a length-bn vector $\begin{pmatrix} \tilde{\mathbf{U}} \\ \tilde{\mathbf{R}} \\ \tilde{\mathbf{I}} \end{pmatrix}$, where $\tilde{\mathbf{I}}$ is just the column version of the $b \times b$ identity matrix. It is generated by stacking columns of the identity matrix one after the other. The second term, $\tilde{\mathbf{R}}$ represents the redundancy Alice adds. The *redundancy vector* $\tilde{\mathbf{R}}$ is a length-δn column vector generated by solving the matrix equation for $\tilde{\mathbf{R}}$.

$$D \begin{pmatrix} \tilde{\mathbf{U}} \\ \tilde{\mathbf{R}} \\ \tilde{\mathbf{I}} \end{pmatrix} = \mathbf{0}.$$

where D is a $\delta n \times bn$ matrix defined as the *redundancy matrix*. D is obtained by choosing each element as an independent and uniformly random symbol from the finite field $\mathbb{F}_q$. Due to the dependence of D on δ and thus on z_O, the Omniscient Adversary algorithm is *not* universal. The redundancy matrix D is known to *all* parties – Alice, Bob, and Calvin – and hence does not constitute a shared secret.

Alice then proceeds to the standard network encoding. She rearranges $\tilde{\mathbf{X}}$, a length-bn vector, into the $b \times n$ matrix X. The j^{th} column of X consists of symbols from the $((j-1)b+1)^{th}$ through $(jb)^{th}$ symbols of $\tilde{\mathbf{X}}$. From this point on, Alice's encoder implements the classical random network encoder described in §IV-B, to generate her transmitted packets.

Bob's Decoder: As shown in (8), Bob's received data is related to Alice and Calvin's transmitted data as $Y = \hat{T}X + E$. Bob's objective, as in §VII, is to distill out the effect of the error matrix E and recover the vector X. He can then retrieve Alice's data by extracting the first $(bn - b^2 - \delta n)$ symbols to obtain $\tilde{\mathbf{U}}$.

To decode, Bob performs the following steps, each of which corresponds to an elementary matrix operation.

- *Determining Calvin's strength:* Bob first determines the strength of the adversary z_O, which is the column rank of $T_{Z\to Y}$. Bob does not know $T_{Z\to Y}$, but since T and $T_{Z\to Y}$ span disjoint vector spaces, the column rank of Y is equal to the sum of the column ranks of T and $T_{Z\to Y}$. Since the column rank of T is simply the batch size b, Bob determines z_O by subtracting b from the column rank of the matrix Y.
- *Discarding irrelevant information:* Since the classical random network code is run without any central coordinating authority, the packets of information that Bob receives may be highly redundant. Of the packets Bob receives, he selectively discards some so that the resulting matrix Y has $b + z_O$ rows, and has full row rank. For him to consider more packets is useless, since at most $b + z_O$ packets of information have been injected into the network, b from Alice and z_O from Calvin. This operation has the additional benefit of reducing the complexity of linear operations that Bob needs to perform henceforth. This reduces the dimensions of the matrix $\hat{T}$, since Bob can discard the rows corresponding to the discarded packets.
- *Estimating a "basis" for E:* If Bob could directly estimate a basis for the column space of E, then he could simply decode as in the Shared Secret algorithm. However, there is no shared secret that enables him to discover a basis for the column space of E. So, he instead chooses a *proxy error matrix* T'' whose columns (which are, in general, linear combinations of columns of both X and E) act as a *proxy error basis* for columns of E. This is analogous to the step (8), where the matrix $\hat{T}$ acts as a proxy transfer matrix for the unknown matrix T.
  The matrix T'' is obtained as follows. Bob selects z_O columns from Y such that these columns, together with the b columns of $\hat{T}$, form a basis for the columns of Y. Without loss of generality, these columns correspond to the first z_O columns of Y (if not, Bob simply permutes the columns of Y to make it so). The $(b + z_O) \times z_O$ matrix corresponding to these first z_O columns is denoted T''.
- *Changing to proxy basis:* Bob rewrites Y in the basis corresponding to the columns of the $(b + z_O) \times (b + z_O)$ matrix $[T''\ \hat{T}]$. Therefore Y can now be written as

$$Y = [T''\ \hat{T}] \begin{bmatrix} I_{z_O} & F^Z & 0 \\ 0 & F^X & I_b \end{bmatrix}. \tag{11}$$

Here $\begin{bmatrix} F^Z \\ F^X \end{bmatrix}$ is defined as the $(b+z_O)\times(n-(b+z_O))$ matrix representation of the columns of Y (other than those in $[T''\ \hat{T}]$) in the new basis, with F^Z and F^X defined as the sub-matrices of appropriate dimensions.

Bob splits X as $X=[X_1\ X_2\ X_3]$, where X_1 corresponds to the first z_O columns of X, X_3 to the last b columns of X, and X_2 to the remaining columns of X. We perform linear algebraic manipulations on (11), to reduce it to a form in which the variables in X are related by a linear transform solely to quantities that are computable by Bob. Claim 3 summarizes the effect of these linear algebraic manipulations (proof in Appendix).

Claim 3: The matrix equation (11) is exactly equivalent to the matrix equation $\hat{T}X_2=\hat{T}\left(F^X+X_1F^Z\right)$.

To complete the proof of correctness of our algorithm, we need only the following claim, proved in the Appendix.

Claim 4: For $\delta > n(z_O+\varepsilon)$, with probability greater than $q^{-n\varepsilon}$, the system of linear equations

$$\hat{T}X_2 = \hat{T}\left(F^X+X_1F^Z\right) \quad (12)$$

$$D\tilde{\mathbf{X}} = 0 \quad (13)$$

is solvable for X.

The final claim enables Bob to recover X, which contains Alice's information at rate $R=C-2z_O$. Of code design, encoding and decoding, the most computationally expensive is decoding. The costliest step involves inverting the linear transform corresponding to (12)-(13), which is of dimension $\mathcal{O}(nC)$. □

IX. Limited Adversary Model

We combine the strengths of the Shared Secret algorithm and the Omniscient Adversary algorithm, to achieve the higher rate of $C=C-z_O$, without needing a secret channel. The caveat is that Calvin's strength is more limited; the number of packets he can transmit, z_O, and the number he can eavesdrop on, z_I, satisfy the technical constraint

$$2z_O+z_I<C. \quad (14)$$

We call such an adversary a *Limited Adversary*.

The main idea underlying our Limited Adversary algorithm is simple. Alice uses the Omniscient Adversary algorithm to transmit a "short" message to Bob at rate $C-2z_O$. By (14), $z_I<C-2z_O$, the rate z_I at which Calvin eavesdrops is strictly less than Alice's rate of transmission $C-2z_O$. Hence Calvin cannot decode Alice's message, but Bob can. This means Alice's message to Bob is secret from Calvin. Alice then builds upon this secret, using the Shared Secret algorithm to transmit the bulk of her message to Bob at the higher rate $C-z_O$.

Though the following algorithm requires Alice to know z_O and z_I, we describe in §IX-A how to change the algorithm to make it independent of these parameters. The price we pay is a slight decrease in rate.

Alice's Encoder: Alice's encoder follows essentially the schema described above, except for a technicality – the information she transmits to Bob via the Omniscient Adversary algorithm is padded with some random symbols. This is for two reasons. Firstly, since the Omniscient Adversary algorithm has a probability of error that decays exponentially with the size of the input, it isn't guaranteed to perform well to transmit just a small message. Secondly, the randomness in the padded symbols also ensures strong information-theoretic secrecy of the small secret message, i.e., we can then show (in Claim 5) that Calvin's best estimate of *any function* of the secret information is no better than if he made random guesses.

Alice's information X decomposes into two parts $[X_1\ X_2]$. She uses the information she wishes to transmit to Bob, at rate $R=C-z_O-\Delta$, as input to the encoder of the Shared Secret algorithm, thereby generating the $b\times n(1-\Delta)$ sub-matrix X_1. Here Δ is a parameter that enables Alice to trade off between the the probability of error and rate-loss.

The second sub-matrix, X_2, which we call the *secrecy matrix* is analogous to the secret $\mathbb{S}$ used in the Secret Sharing algorithm described in §VII. The size of X_2 is $b\times\Delta n$. In fact, X_2 is an encoding of the secret $\mathbb{S}$ Alice generates in the Shared Secret algorithm. The $b(C+1)$ symbols corresponding to the parity symbols $\{r_d\}$ and the hash matrix H are written in the form of a length-$b(C+1)$ column vector. This vector is appended with symbols chosen uniformly at random from $\mathbb{F}_q$ to result in the length-$(C-z_O-\delta n)\Delta n$ vector $\tilde{\mathbf{U}}'$. This vector $\tilde{\mathbf{U}}'$ could function as the input $\tilde{\mathbf{U}}$ to the Omniscient Adversary algorithm operated over a packet-size Δn, with a probability of decoding error that is exponentially small in Δn; however, we actually use a hash of $\tilde{\mathbf{U}}'$ to generate the input $\tilde{\mathbf{U}}$ to the Omniscient Adversary algorithm. To be more precise, $\tilde{\mathbf{U}}=V\tilde{\mathbf{U}}'$, where V is any square *MDS code generator matrix* [1] of dimension $(C-z_O-\delta n)\Delta n$, known to all parties Alice, Bob, and Calvin. As we see later, hashing $\tilde{\mathbf{U}}'$ with V strengthen the secrecy of $\mathbb{S}$ (and enables the proof of Claim 5 below). Alice then uses the encoder for the Omniscient Adversary algorithm to generate X_2 from $\tilde{\mathbf{U}}$.

The two components of X, i.e., X_1 and X_2, respectively correspond to the information Alice wishes to transmit to Bob, and an implementation of the low rate secret channel. The fraction of the packet-size corresponding to X_2 is "small", i.e., Δ. Finally, Alice implements the classical random encoder described in §IV-B.

Bob's Encoder: Bob arranges his received packets into the matrix $Y=[Y_1\ Y_2]$. The sub-matrices Y_1 and Y_2 are respectively the network transforms of X_1 and X_2.

Bob decodes in two steps. Bob first decodes Y_2 to obtain $\mathbb{S}$. He begins by using the Omniscient Adversary decoder to obtain the vector $\tilde{\mathbf{U}}$. He obtains $\tilde{\mathbf{U}}'$ from $\tilde{\mathbf{U}}$, by multiplying by V^{-1}. He then extracts from $\tilde{\mathbf{U}}'$ the $b(C+1)$ symbols corresponding to $\mathbb{S}$. The following claim, proved in the Appendix, ensures that $\mathbb{S}$ is indeed secret from Calvin.

Claim 5: The probability that Calvin guesses $\mathbb{S}$ correctly is at most $q^{-b(C+1)}$, i.e., $\mathbb{S}$ is information-theoretically secret from Calvin.

Thus Alice has now shared $\mathbb{S}$ with Bob. Bob uses $\mathbb{S}$ as the side information used by the decoder of the Shared Secret algorithm

[1] Secret Sharing protocols [24] demonstrate that using MDS code generator matrices guarantees that to infer even a single symbol of $\tilde{\mathbf{U}}'$ from $\tilde{\mathbf{U}}$ requires the entire vector $\tilde{\mathbf{U}}$.

	Adversarial Strength	Rate	Complexity
Shared Secret	$z_O < C$, z_I = network	$C - z_O$	$\mathcal{O}(nC^2)$
Omniscient	$z_O < C/2$, z_I = network	$C - 2z_O$	$\mathcal{O}((nC)^3)$
Limited	$z_I + 2z_O < C$	$C - z_O$	$\mathcal{O}(nC^2 + (\delta nC)^3)$

TABLE III—Comparison of our three algorithms

to decode Y_1. This enables him to recover X_1, which contains Alice's information at rate $R = C - z_O$. Since the Limited Adversary algorithm is essentially a concatenation of the Shared Secret algorithm with the Omniscient Adversary algorithm, the computational cost is the sum of the computational costs of the two (with Δn replacing n as the block-length for the Shared Secret algorithm). This quantity therefore equals $\mathcal{O}(nC^2 + (\Delta nC)^3)$. □

A. Limited Adversary: Universal Codes

We now discuss how to convert the above algorithm to be independent of the network parameters z_O and z_I. Alice's challenge is to design for all possible z_O and z_I pairs that satisfy the constraint (14). For any specific z_I, Alice needs to worry only about the largest z_O that satisfies (14) because what works against an attacker with a particular traffic injection strength works against all weaker attackers. Note that C, z_O, and z_I are all integers, and thus there are only $C - 1$ such attackers. For each of these attackers, Alice designs a different secrecy matrix X_2 as described above. She appends these $C-1$ matrices to her information X_1 and sends the result as described in the above section.

To decode Bob needs to estimate which secrecy matrix to use, i.e., which one of them is secret from the attacker. For this he needs a good upper bound on z_O. But, just as in the omniscient adversary algorithm, he can obtain this by computing the column rank of Y, and subtracting b from it. He then decodes using the secrecy matrix corresponding to $(z_O, C - 1 - 2z_O)$. This secrecy matrix suffices since z_I can at most be $C-1-2z_O$, which corresponds to Calvin's highest eavesdropping strength for this z_O. □

X. Conclusion

Random network codes are vulnerable to Byzantine adversaries. This work makes them secure. We provide algorithms which are information-theoretically secure and rate-optimal for different adversarial strengths as shown in Table I. When the adversary is omniscient, we show how to achieve a rate of $C - 2z_O$, where z_O is the number of packets the adversary injects and C is the network capacity. If the adversary cannot observe everything, our algorithms achieve a higher rate, $C-z_O$. Both rates are optimal. Further our algorithms are practical; they are distributed, have polynomial-time complexity and require no changes at the internal nodes.

References

[1] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung. Network Information Flow. In *IEEE Transactions on Information Theory*, 2000.

[2] N. Cai and R. W. Yeung. Secure Network Coding. In *ISIT*, 2002.

[3] N. Cai and R. W. Yeung. Network error correction, part 2: Lower bounds. submitted to Communications in Information and Systems, 2006.

[4] D. Charles, K. Jain, and K. Lauter. Signatures for network coding. In *Proceedings of the fortieth annual Conference on Information Sciences and Systems*, Princeton, NJ, USA, 2006.

[5] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *Journal of the Association for Computing Machinery*, 40(1):17–47, January 1993.

[6] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. In *IEEE INFOCOM*, 2005.

[7] C. Gkantsidis and P. Rodriguez. Cooperative Security for Network Coding File Distribution. In *IEEE INFOCOM*, 2006.

[8] T. Ho, R. Koetter, M. Médard, D. Karger, and M. Effros. The Benefits of Coding over Routing in a Randomized Setting. In *ISIT, 2003*.

[9] T. Ho., R. Koetter, M. Médard, D. Karger, and M. Effros. The benefits of coding over routing in a randomized setting. In *IEEE International Symposium on Information Theory (ISIT)*, page 442, Yokohama, July 2003.

[10] T. C. Ho, B. Leong, R. Koetter, M. Médard, M. Effros, and D. R. Karger. Byzantine modification detection in multicast networks using randomized network coding. In *International Symposium on Information Theory*, 2004.

[11] S. Jaggi. *Design and Analysis of Network Codes*. Dissertation, California Institute of Technology, 2005.

[12] S. Jaggi, M. Langberg, T. Ho, and M. Effros. Correction of adversarial errors in networks. In *Proceedings of International Symposium in Information Theory and its Applications*, Adelaide, Australia, 2005.

[13] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*, 2003.

[14] A. Jiang. Network Coding for Joing Storage and Transmission with Minimum Cost. In *ISIT*, 2006.

[15] S. Katti, D. Katabi, W. Hu, H. S. Rahul, and M. Médard. The importance of being opportunistic: Practical network coding for wireless environments. In *43rd Annual Allerton Conference on Communication, Control, and Computing*, 2005.

[16] S. Katti, H. Rahul, D. Katabi, W. H. M. Médard, and J. Crowcroft. Xors in the air: Practical wireless network coding. In *ACM SIGCOMM*, 2006.

[17] R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking, 2003*.

[18] R. Koetter and M. Médard. Beyond routing: An algebraic approach to network coding. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM)*, volume 1, pages 122–130, 2002.

[19] M. N. Krohn, M. J. Freedman, and D. Mazires. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the IEEE Symposium on Security and Privacy, 2004*.

[20] S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Transactions on Information Theory*, 2003.

[21] D. S. Lun, M. Médard, and R. Koetter. Efficient operation of wireless packet networks using network coding. In *International Workshop on Convergent Technologies (IWCT)*, 2005.

[22] D. S. Lun, N. Ratnakar, R. Koetter, M. Médard, E. Ahmed, and H. Lee. Achieving Minimum-Cost Multicast: A Decentralized Approach Based on Network Coding. In *IEEE INFOCOM, 2005*.

[23] A. Pelc and D. Peleg. Broadcasting with locally bounded byzantine faults. *Information Processing Letters*, 93(3):109–115, February 2005.

[24] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, Seattle, Washington, United States, 1989.

[25] L. Subramanian. *Decentralized Security Mechanisms for Routing Protocols*. PhD thesis, University of California at Berkeley, Computer Science Division, Berkeley, CA 94720, 2005.

[26] H. J. Wertz. On the numerical inversion of a recurrent problem: The Vandermonde matrix. AC-10:492, Oct. 1965.

[27] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *IEEE Infocom*, volume 2, pages 585–594. IEEE, 2000.

[28] R. W. Yeung and N. Cai. Network error correction, part 1: Basic concepts and upper bounds. submitted to Communications in Information and Systems, 2006.

Appendix

A. Proof of Claim 1

The parity-check matrix P is, by construction, a *Vandermonde matrix* [26], and therefore has full row rank. Further, since P is hidden

from Calvin, with probability at least $1 - Cnq^{-1}$ he cannot choose interference such that the matrix product EP has a lower column rank than does E (the proof of this statement follows from [12]). □

B. Proof of Claim 2

The proof of Claim 2 follows directly from [11]. Essentially, it is a consequence of the fact that with high probability over network code design, $\hat{T}$ and S both individually have full column rank, and the vector spaces their columns intersect only in the zero vector. Hence (10) the transform corresponding to $[\hat{T}\ S]$ has full column rank. □

C. Proof of Claim 3

Rewriting the right-hand side of (11) and substituting for Y from (7) results in

$$\hat{T}X + T_{Z\to Y}(Z - LX) = \hat{T}[0\ F^X\ I_b] + T''[I_{z_O}\ F^Z\ 0]. \quad (15)$$

Since the columns of T'' are spanned by the columns of $[\hat{T}\ T_{Z\to Y}]$, therefore we may write T'' as $\hat{T}M_1 + T_{Z\to Y}M_2$, where the matrices M_1 and M_2 represent the appropriate basis transformation. Thus (15) becomes

$$\hat{T}X + T_{Z\to Y}(Z - LX) = \hat{T}\left([0\ F^X\ I_b]\right) + \left(\hat{T}M_1 + T_{Z\to Y}M_2\right)[I_{z_O}\ F^Z\ 0]. \quad (16)$$

Since the vector spaces spanned by the columns of $\hat{T}$ and $T_{Z\to Y}$ are disjoint (except in the zero vector), therefore we may compare the term multiplying the matrix $\hat{T}$ on both sides of 16 (we may also compare the term corresponding to $T_{Z\to Y}$, but this gives us nothing useful). This comparison gives us the equation

$$\hat{T}X = [0\ F^X\ I_b] + \hat{T}M_1[I_{z_O}\ F^Z\ 0]. \quad (17)$$

We split the matrix equation (15) into three parts, corresponding to the sub-matrices X_1, X_2 and $X_3]$ of X. Thus (17) now splits into the three equations

$$\hat{T}X_1 = \hat{T}M_1 I_{z_O}, \quad (18)$$
$$\hat{T}X_2 = \hat{T}F^X + \hat{T}M_1F^Z, \text{ and} \quad (19)$$
$$\hat{T}X_3 = \hat{T}. \quad (20)$$

The equation (20) is trivial, since it only reiterates that X_3 equals columns of an identity matrix. The equation (18) allows us to estimate that M_1 equals X_1. We are finally left with (19), which by substituting for M_1 from (18) reduces to

$$\hat{T}X_2 = \hat{T}\left(F^X + X_1F^Z\right). \quad (21)$$

□

D. Proof of Claim 4

We rewrite the term X_1F^Z in (21) as $(F^{Z^T}X_2^T)^T$. We denote by $\tilde{\mathbf{X}}_1'$ the vector obtained by stacking the columns of X_2^T one after the other. Let $D = [D_1\ D_2]$, where D_2 corresponds to the last b^2 columns of D and D_1 corresponds to the remaining columns of D. Define $\alpha = n - (b + z_O)$. Denote by $\vec{F^X}$ the vector formed by stacking columns of the matrix F^X one after the other, and by $f_{i,j}$ the $(i,j)^{th}$ entry of the matrix F^{Z^T}. The system of linear equations (12)-(13) can be written in matrix form as

$$A\begin{pmatrix} \tilde{\mathbf{X}}_1' \\ \tilde{\mathbf{X}}_2 \end{pmatrix} = \begin{pmatrix} \hat{T}\vec{F^X} \\ -D_2\tilde{\mathbf{I}} \end{pmatrix}$$

where A is given by

$$\left[\begin{array}{cccc|ccccc} -f_{1,1}\hat{T} & -f_{2,1}\hat{T} & \dots & -f_{z_O,1}\hat{T} & \hat{T} & 0 & \dots & \dots & 0 \\ -f_{1,2}\hat{T} & -f_{2,2}\hat{T} & \dots & -f_{z_O,2}\hat{T} & 0 & \hat{T} & 0 & \dots & 0 \\ -f_{1,3}\hat{T} & -f_{2,3}\hat{T} & \dots & -f_{z_O,3}\hat{T} & \vdots & 0 & \hat{T} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 & \ddots & 0 \\ -f_{1,\alpha}\hat{T} & -f_{2,\alpha}\hat{T} & \dots & -f_{z_O,\alpha}\hat{T} & 0 & 0 & 0 & 0 & \hat{T} \\ \hline \multicolumn{4}{c|}{D_1} & \multicolumn{5}{c} \end{array}\right]$$

This matrix A is described by smaller dimensional matrices as entries. The matrix $\hat{T}$ has dimensions $(b + z_O) \times b$. The j^{th} row of matrices in the top portion of matrix A describes an equation corresponding to the j^{th} column of the matrix equation in Equation 12. The bottom portion of A corresponds to Equation 13.

Bob can recover the variables $X(i,j)$ if and only if the above matrix A has full column rank. We now analyze A to show that this is indeed the case (with high probability) for sufficiently large δn.

Let $\varepsilon > 0$ be a small constant. Since, with high probability, $\hat{T}$ has full column-rank, the last αb columns of the matrix (represented by the right side of A) have full column rank with probability at least $1 - \varepsilon$.

We now address the left columns of A. Consider performing column operations from right to left, to zero out the $\hat{T}$s in the left side of the top rows of A (that is, to zero out the upper left sub-matrix of A). A has full column rank iff after this process the lower left sub-matrix of A has full column rank. We show that this is the case with high probability over the random elements of D (when δn is chosen to be sufficiently large).

Let f_{ij}'s be the values appearing in the upper left sub-matrix of A. We show that for any (adversarial) choice of f_{ij}'s, with high probability, the act of zeroing out the $\hat{T}$'s yields a lower left sub-matrix of A with full column rank. Then using the union bound on all possible values of f_{ij} we obtain our assertion.

For any fixed values of f_{ij}, let $C(j)$, for $j = 1$ to bz_O, denote the columns of the lower left sub-matrix of A after zeroing out the $\hat{T}$'s. For each j, the vector $C(j)$ is a linear combination of the (lower part of the) j^{th} column of A with columns from the lower right sub-matrix of A. As the entries of D_1 are independent random variables uniformly distributed in $\mathbb{F}_q$, the columns $C(j)$ for $j = 1, \dots, bz_O$ consist of independent entries that are also uniformly distributed in $\mathbb{F}_q$. Standard analysis shows that the probability that the columns $C(j)$ are not independent is $q^{bz_O - \delta n}$. For the union bound we would like this probability to be at most $q^{-\alpha z_O - n\varepsilon} = q^{-(n-(b+z_O))z_O - n\varepsilon}$. Thus, it suffices to take $\delta n = n(z_O + \varepsilon)$ for an error probability of at most $q^{-n\varepsilon}$. Recall that $b = C - z_O$. We conclude that the total rate of X transmitted in our scheme is $((n-b)b - \delta n)/n = ((n-C)(C-2z_O) + \log_q \frac{1}{\varepsilon})/n$. As n grows large, the rate approaches $C - 2z_O$ as desired. □

E. Proof of Claim 5

The vector $\tilde{\mathbf{U}}$ was generated from $\tilde{\mathbf{U}}'$ via an MDS code generator matrix (see Footnote 1), and a folklore result about network codes is that with high probability over random network code design the linear transform between Alice and Calvin also has the MDS property. Thus, for Calvin to infer even a single symbol of the length-$(C - z_O - n\delta)n\Delta$ vector $\tilde{\mathbf{U}}'$, he needs to have received at least $(C - z_O - n\delta)n\Delta$ linear combinations of the variables in the secrecy matrix X_2. Since Calvin can overhear z_I packets, he has access to $z_I n\Delta$ equations that are linear in the unknown variables. The difference between the number of variables unknown to Calvin, and the number of equations Calvin has, is linear in $n\Delta$ – for large enough $n\Delta$, this difference is larger than $b(C+1)$, the length of the vector $\mathbb{S}$. By a direct extension of [24], Calvin's probability of guessing any function of $\mathbb{S}$ correctly is $q^{-b(C+1)}$. □