Network coding meets TCP

Jay Kumar Sundararajan*, Devavrat Shah*, Muriel Médard*, Michael Mitzenmacher†, João Barros‡

*Dept. of EECS
Massachusetts Institute of Technology,
Cambridge, MA 02139, USA
{jaykumar, devavrat,
medard}@mit.edu

†School of Eng. and Appl. Sciences
Harvard University,
Cambridge, MA 02138, USA
michaelm@eecs.harvard.edu

‡Instituto de Telecomunicações
Dept. de Engenharia Electrotécnica e de Computadores
Faculdade de Engenharia da
Universidade do Porto, Portugal
jbarros@fe.up.pt

***Abstract*—We propose a mechanism that incorporates network coding into TCP with only minor changes to the protocol stack, thereby allowing incremental deployment. In our scheme, the source transmits random linear combinations of packets currently in the congestion window. At the heart of our scheme is a new interpretation of ACKs – the sink acknowledges every degree of freedom (*i.e.*, a linear combination that reveals one unit of new information) even if it does not reveal an original packet immediately. Such ACKs enable a TCP-compatible sliding-window approach to network coding. Our scheme has the nice property that packet losses are essentially masked from the congestion control algorithm. Our algorithm therefore reacts to packet drops in a smooth manner, resulting in a novel and effective approach for congestion control over networks involving lossy links such as wireless links. Our scheme also allows intermediate nodes to perform re-encoding of the data packets. Our simulations show that our algorithm, with or without re-encoding inside the network, achieves much higher throughput compared to TCP over lossy wireless links. We also establish the soundness and fairness properties of our algorithm. Finally, we present queuing analysis for the case of intermediate node re-encoding.**

I. Introduction

Network coding has emerged as an important potential approach to the operation of communication networks, especially wireless networks. The major benefit of network coding stems from its ability to *mix* data, across time and across flows. This makes data transmission over lossy wireless networks robust and effective. Despite this potential of network coding, we still seem far from seeing widespread implementation of network coding across networks. We believe a major reason for this is that it is not clear how to naturally add network coding to current network systems (the incremental deployment problem) and how network coding will behave in the wild.

In order to bring the ideas of network coding into practice, we need a protocol that brings out the benefits of network coding while requiring very little change in the protocol stack. Flow control and congestion control in today's internet are predominantly based on the Transmission Control Protocol (TCP), which works using the idea of a sliding transmission window of packets, whose size is controlled based on feedback. The TCP paradigm has clearly proven successful. We therefore see a need to find a sliding-window approach as similar as possible to TCP for network coding that makes use of acknowledgments for flow and congestion control. (This problem was initially proposed in [1].) Such an approach would necessarily differ from the generation-based approach more commonly considered for network coding [2], [3]. In this paper, we show how to incorporate network coding into TCP, allowing its use with minimal changes to the protocol stack, and in such a way that incremental deployment is possible.

The main idea behind TCP is to use acknowledgments of newly received packets as they arrive *in correct sequence order* in order to guarantee reliable transport and also as a feedback signal for the congestion control loop. This mechanism requires some modification for systems using network coding. The key difference to be dealt with is that under network coding the receiver does not obtain original packets of the message, but linear combinations of the packets that are then decoded to obtain the original message once enough such combinations have arrived. Hence, the notion of an ordered sequence of packets as used by TCP is missing, and further, a linear combination may bring in new information to a receiver even though it may not reveal an original packet immediately. The current ACK mechanism does not allow the receiver to acknowledge a packet before it has been decoded. For network coding, we need a modification of the standard TCP mechanism that acknowledges every unit of information received. A new unit of information corresponds mathematically to a *degree of freedom*; essentially, once n degrees of freedom have been obtained, a message that would have required n unencoded packets can be decoded. We present a mechanism that performs the functions of TCP, namely reliable transport and congestion control, based on acknowledging every degree of freedom received, whether or not it reveals a new packet.

Our solution introduces a new network coding layer between the transport layer and the network layer of the protocol stack. Thus, we recycle the congestion control principle of TCP, namely that the number of packets involved in transmissions cannot exceed the number of acknowledgments received by more than the congestion window size. However, we introduce two main changes. First, whenever the source is allowed to transmit, it sends a random linear combination of all packets in the congestion window. Second, the receiver acknowledges degrees of freedom and not original packets. (This idea was

This work is supported by subcontract #18870740-37362-C issued by Stanford University and supported by DARPA, NSF Grant No. CNS-0627021 and subcontract # 060786 issued by BAE Systems and supported by DARPA and SPAWARSYSCEN under Contract No. N66001-06-C-2020.

previously introduced in [4] in the context of a single hop erasure broadcast link.) An appropriate interpretation of the degree of freedom allows us to order the receiver degrees of freedom in a manner consistent with the packet order of the source. This lets us utilize the standard TCP protocol with the minimal change. We use the TCP-Vegas protocol, as it is more compatible with our modifications.

In considering the potential benefits of our TCP-compatible network coding solution, we focus on the area of wireless links. It is well known that TCP is not well suited for lossy links, which are generally more prevalent in wireless systems. TCP performs poorly on lossy links primarily because it is designed to interpret each loss as a congestion signal. Adapting TCP for wireless scenarios is a very well-studied problem (see [5] and references therein for a survey). The general approach has been to mask losses from TCP using link layer retransmission [6]. However, it has been noted in the literature ([7], [8]) that the interaction between link layer retransmission and TCP's retransmission can be complicated and that performance may suffer due to independent retransmission protocols at different layers. More importantly, if we want the benefits of approaches such as multipath opportunistic routing which exploit the broadcast nature of the wireless medium, link layer retransmission is not the best approach.

Our scheme does not rely on the link layer for recovering losses. Instead, we use an erasure correction scheme based on random linear codes across packets. Coding across packets is a natural way to handle losses. A coding based approach is better suited for broadcast-mode opportunistic routing scenarios, as randomly chosen linear combinations of packets are more likely to convey new information, compared to retransmissions. The MORE scheme of [3] explains the benefits of network coding in the context of opportunistic routing. However, the problem with MORE, as explained above, is the batch processing that makes it less compatible to a sliding window protocol such as TCP. By providing an interface between TCP and a network coded system, we present a new approach to implementing TCP over wireless networks, and it is here where the benefits of our solution are most dramatic.

It is important to note that our scheme respects the end-to-end philosophy of TCP – it would work even if coding operations are performed only at the end hosts. Having said that, if some nodes inside the network also perform network coding, our solution naturally generalizes to such scenarios as well. Indeed, the queuing analysis in Section VI-B considers such a situation.

A. Previous work

Starting with the initial works of [9] and [10], there has been a rapid growth in the theory and potential applications of network coding. These developments have been summarized in several survey papers and books such as [11]. However, to a large extent, this theory has not yet been implemented in practical systems.

There have been several important advances in bridging the gap between theory and practice. The distributed random linear coding idea, introduced by Ho *et al.* [12], is a significant step towards a robust implementation. The work by Chou *et al.* [2] introduced the idea of embedding the coefficients used in the linear combination in the packet header, and also the notion of generations (coding blocks). The work by Katti *et al.* [13] used the idea of local opportunistic coding to present a practical implementation of a network coded system for unicast. The use of network coding in combination with opportunistic routing was presented in [3]. Reference [14] proposed an on-the-fly coding scheme, but the packets are acknowledged only upon decoding. The resulting decoding delay will interfere with the TCP congestion control loop. Thus, none of these works allows an ACK-based sliding-window network coding approach that is compatible with TCP. This is the problem we address in our current work.

The rest of the paper explains the details of our new protocol along with its theoretical basis, and analyzes its performance using simulations as well as an idealized theoretical analysis.

II. Preliminaries

We introduce definitions that will be useful throughout the paper (see [4] for more details). Packets are treated as vectors over a finite field $\mathbb{F}_q$ of size q. All the discussion here is with respect to a single source that generates a stream of packets. The k^{th} packet that the source generates is said to have an *index* k and is denoted as $\mathbf{p_k}$.

Definition 1 (Seeing a packet): A node is said to have *seen* a packet $\mathbf{p_k}$ if it has enough information to compute a linear combination of the form $(\mathbf{p_k} + \mathbf{q})$, where $\mathbf{q} = \sum_{\ell>k} \alpha_\ell \mathbf{p}_\ell$, with $\alpha_\ell \in \mathbb{F}_q$ for all $\ell > k$. Thus, $\mathbf{q}$ is a linear combination involving packets with indices larger than k.

The notion of "seeing" a packet is a natural extension of the notion of "decoding" a packet, or more specifically, receiving a packet in the context of classical TCP. For example, if a packet $\mathbf{p_k}$ is decoded then it is indeed also seen, with $\mathbf{q} = \mathbf{0}$. A node can compute any linear combination whose coefficient vector is in the span of the coefficient vectors of previously received linear combinations. This leads to the following definition.

Definition 2 (Knowledge of a node): The *knowledge of a node* is the set of all linear combinations of original packets that it can compute, based on the information it has received so far. The coefficient vectors of these linear combinations form a vector space called the *knowledge space* of the node.

We state a useful proposition without proof (see Corollary 1, [4] for details).

Proposition 1: If a node has seen packet $\mathbf{p_k}$*, then it knows exactly one linear combination of the form* $\mathbf{p_k} + \mathbf{q}$ *such that* $\mathbf{q}$ *is itself a linear combination involving only* **unseen** *packets.*

The above proposition inspires the following definition.

Definition 3 (Witness): We call the unique linear combination guaranteed by Proposition 1, the *witness for seeing* $\mathbf{p_k}$.

A compact representation of the knowledge space is the basis matrix. This is a matrix in row-reduced echelon form (RREF) such that its rows form a basis of the knowledge

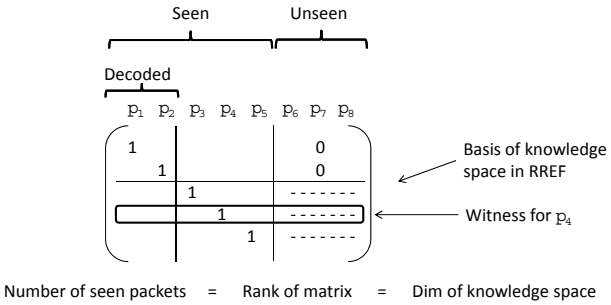


Fig. 1. Seen packets and witnesses in terms of the basis matrix

space. Figure 1 explains the notion of a seen packet in terms of the basis matrix. Essentially, the seen packets are the ones that correspond to the pivot columns of the basis matrix. Given a seen packet, the corresponding pivot row gives the coefficient vector for the witness linear combination. An important observation is that *the number of seen packets is always equal to the dimension of the knowledge space*, or the number of degrees of freedom that have been received so far. A newly received linear combination that increases the dimension is said to be *innovative*. We assume throughout the paper that the field size is very large. As a consequence, each reception will be innovative with high probability, and will cause the next unseen packet to be seen (see Lemma 1).

Example: Suppose a node knows the following linear combinations: $\mathbf{x} = (\mathbf{p_1} + \mathbf{p_2})$ and $\mathbf{y} = (\mathbf{p_1} + \mathbf{p_3})$. Since these are linearly independent, the knowledge space has a dimension of 2. Hence, the number of seen packets must be 2. It is clear that packet $\mathbf{p_1}$ has been seen, since $\mathbf{x}$ satisfies the requirement of Definition 1. Now, the node can compute $\mathbf{z} \triangleq \mathbf{x} - \mathbf{y} = (\mathbf{p_2} - \mathbf{p_3})$. Thus, it has also seen $\mathbf{p_2}$. That means $\mathbf{p_3}$ is unseen. Hence, $\mathbf{y}$ is the witness for $\mathbf{p_1}$, and $\mathbf{z}$ is the witness for $\mathbf{p_2}$.

III. The new protocol

In this section, we present the logical description of our new protocol, followed by a way to implement these ideas with as little disturbance as possible to the existing protocol stack.

A. Logical description

The main aim of our algorithm is to mask losses from TCP using random linear coding. We make some important modifications in order to incorporate coding. First, instead of the original packets, we transmit random linear combinations of packets in the congestion window. While such coding helps with erasure correction, it also leads to a problem in acknowledging data. TCP operates with units of packets, which have a well-defined ordering. Thus, the packet sequence number can be used for acknowledging the received data. The unit in our protocol is a degree of freedom. However, when packets are coded together, there is no clear ordering of the degrees of freedom that can be used for ACKs. Our main contribution is the solution to this problem. The notion of seen packets defines an ordering of the degrees of freedom that is consistent with the packet sequence numbers, and can therefore be used to acknowledge degrees of freedom.

Upon receiving a linear combination, the sink finds out which packet, if any, has been newly seen because of the new arrival and acknowledges that packet. The sink thus pretends to have received the packet even if it cannot be decoded yet. We will show in Section IV that at the end this is not a problem because if all the packets in a file have been seen, then they can all be decoded as well.

The idea of transmitting random linear combinations and acknowledging seen packets achieves our goal of masking losses from TCP as follows. As mentioned in Section II, with a large field size, every random linear combination is very likely to cause the next unseen packet to be seen. So, even if a transmitted linear combination is lost, the next successful reception will cause the next unseen packet to be seen. From TCP's perspective, this appears as though the degree of freedom waits in a fictitious queue until the channel stops erasing packets and allows it through. Thus, there will never be any duplicate ACKs. Every ACK will cause the congestion window to advance. In short, *the lossiness of the link is presented to TCP as an additional queuing delay that leads to a larger effective round-trip time*. The term round-trip time thus has a new interpretation. It is the effective time the network takes to *reliably* deliver a degree of freedom (including the delay for the coded redundancy, if necessary), followed by the return of the ACK. This is larger than the true network delay it takes for a lossless transmission and the return of the ACK. The more lossy the link is, the larger will be the effective RTT. Presenting TCP with a larger value for RTT may seem counterintuitive as TCP's rate is inversely related to RTT. However, if done correctly, it improves the rate by preventing loss-induced window closing, as it gives the network more time to deliver the data in spite of losses, before TCP times out. Therefore, losses are effectively masked.

The natural question that arises is – how does this affect congestion control? Since we mask losses from the congestion control algorithm, the TCP-Reno style approach to congestion control using packet loss as a congestion indicator is not well suited to this situation. However, it is useful to note that the congestion related losses are made to appear as a longer RTT. Therefore, we need an approach that infers congestion from an increase in RTT. The natural choice is TCP-Vegas.

TCP-Vegas uses a proactive approach to congestion control by inferring the size of the network buffers even before they start dropping packets. The crux of the algorithm is to estimate the round-trip time (RTT) and use this information to find the discrepancy between the expected and actual transmission rate. As congestion arises, buffers start to fill up and the RTT starts to rise, and this is used as the congestion signal. This signal is used to adjust the congestion window and hence the rate. For further details, the reader is referred to [15].

In order to use TCP-Vegas correctly in this setting, we need to ensure that it uses the effective RTT of a degree of freedom, including the fictitious queuing delay. In other words, the RTT should be measured from the point when a packet is first sent out from TCP, to the point when the ACK returns saying that this packet has been seen. This is indeed the case

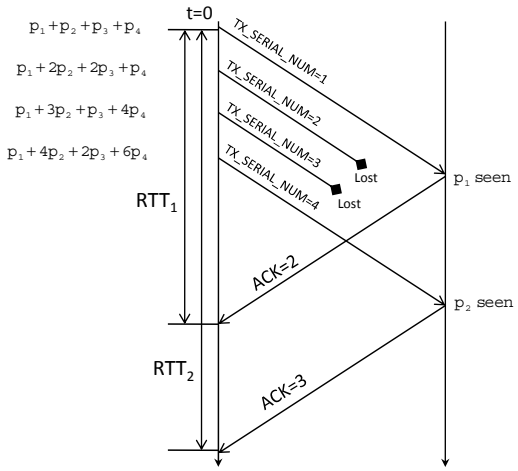


Fig. 2. Example of coding and ACKs

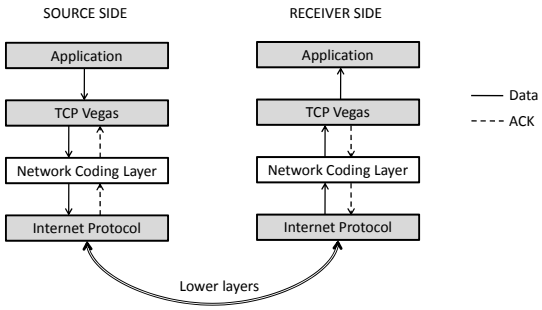


Fig. 3. New network coding layer in the protocol stack

if we simply use the default RTT measurement mechanism of TCP Vegas. The TCP sender notes down the transmission time of every packet. When an ACK arrives, it is matched to the corresponding transmit timestamp in order to compute the RTT. Thus, no modification is required.

Consider the example shown in Figure 2. Suppose the congestion window's length is 4. Assume TCP sends 4 packets to the network coding layer at $t = 0$. All 4 transmissions are linear combinations of these 4 packets. The 1^{st} transmission causes the 1^{st} packet to be seen. The 2^{nd} and 3^{rd} transmissions are lost, and the 4^{th} transmission causes the 2^{nd} packet to be seen (the discrepancy is because of losses). As far as the RTT estimation is concerned, transmissions 2, 3 and 4 are treated as attempts to convey the 2^{nd} degree of freedom. The RTT for the 2^{nd} packet must include the final attempt that successfully delivers the 2^{nd} degree of freedom, namely the 4^{th} transmission. In other words, the RTT is the time from $t = 0$ until the time of reception of ACK=3.

B. Implementation

The implementation of all these ideas in the existing protocol stack needs to be done in as non-intrusive a manner as possible. We present a solution which embeds the network coding operations in a separate layer below TCP and above IP on the source and receiver side, as shown in Figure 3. The exact operation of these modules is described next.

The sender module accepts packets from the TCP source and buffers them into an encoding buffer which represents the coding window[1], until they are ACKed by the receiver. The sender then generates and sends random linear combinations of the packets in the coding window. The coefficients used in the linear combination are also conveyed in the header.

For every packet that arrives from TCP, R linear combinations are sent to the IP layer on average, where R is the redundancy parameter. The average rate at which linear combinations are sent into the network is thus a constant factor more than the rate at which TCP's congestion window progresses. This is necessary in order to compensate for the loss rate of the channel and to match TCP's sending rate to the rate at which data is actually sent to the receiver. If there is too little redundancy, then the data rate reaching the receiver will not match the sending rate because of the losses. This leads to a situation where the losses are not effectively masked from the TCP layer. Hence, there are frequent timeouts leading to a low throughput. On the other extreme, too much redundancy is also bad, since then the transmission rate becomes limited by the rate of the code itself. Besides, sending too many linear combinations can congest the network. The ideal level of redundancy is to keep R equal to the reciprocal of the probability of successful reception. Thus, in practice the value of R should be dynamically adjusted by estimating the loss rate, possibly using the RTT estimates.

Upon receiving a linear combination, the receiver module first retrieves the coding coefficients from the header and appends it to the basis matrix of its knowledge space. Then, it performs a Gaussian elimination to find out which packet is newly seen so that this packet can be ACKed. The receiver module also maintains a buffer of linear combinations of packets that have not been decoded yet. Upon decoding the packets, the receiver module delivers them to the TCP sink.

The algorithm is specified below using pseudo-code. This specification assumes a one-way TCP flow.

1) Source side: The source side algorithm has to respond to two types of events – the arrival of a packet from the source TCP, and the arrival of an ACK from the receiver via IP.

1. Set NUM to 0.
2. *Wait state:* If any of the following events occurs, respond as follows; else, wait.
3. *Packet arrives from TCP sender:*
   a) If the packet is a control packet used for connection management, deliver it to the IP layer and return to wait state.
   b) If packet is not already in the coding window, add it to the coding window.
   c) Set NUM = NUM+R. (R=redundancy factor)
   d) Repeat the following $\lfloor NUM \rfloor$ times:

[1]Whenever a new packet enters the TCP congestion window, TCP transmits it to the network coding module, which then adds it to the coding window. Thus, the coding window is related to the TCP layer's congestion window but generally not identical to it. For example, the coding window will still hold packets that were transmitted earlier by TCP, but are no longer in the congestion window because of a reduction of the window size by TCP. However, this is not a problem because involving more packets in the linear combination will only increase its chances of being innovative.

i) Generate a random linear combination of the packets in the coding window.
ii) Add the network coding header specifying the set of packets in the coding window and the coefficients used for the random linear combination.
iii) Deliver the packet to the IP layer.

e) Set $NUM :=$ fractional part of NUM.
f) Return to the wait state.

4. *ACK arrives from receiver:* Remove the ACKed packet from the coding buffer and hand over the ACK to the TCP sender.

2) Receiver side: On the receiver side, the algorithm again has to respond to two types of events: the arrival of a packet from the source, and the arrival of ACKs from the TCP sink.

1. *Wait state:* If any of the following events occurs, respond as follows; else, wait.
2. *ACK arrives from TCP sink:* If the ACK is a control packet for connection management, deliver it to the IP layer and return to the wait state; else, ignore the ACK.
3. *Packet arrives from source side:*
   a) Remove the network coding header and retrieve the coding vector.
   b) Add the coding vector as a new row to the existing coding coefficient matrix, and perform Gaussian elimination to update the set of seen packets.
   c) Add the payload to the decoding buffer. Perform the operations corresponding to the Gaussian elimination, on the buffer contents. If any packet gets decoded in the process, deliver it to the TCP sink and remove it from the buffer.
   d) Generate a new TCP ACK with sequence number equal to that of the oldest unseen packet.

IV. Soundness of the protocol

We argue that our protocol guarantees reliable transfer of information. In other words, every packet in the packet stream generated by the application at the source will be delivered eventually to the application at the sink. We observe that the acknowledgment mechanism ensures that the coding module at the sender does not remove a packet from the coding window unless it has been ACKed, *i.e.*, unless it has been seen by the sink. Thus, we only need to argue that if all packets in a file have been seen, then the file can be decoded at the sink.

Theorem 1: From a file of n packets, if every packet has been seen, then every packet can also be decoded.

Proof: If the sender knows a file of n packets, then the sender's knowledge space is of dimension n. Every seen packet corresponds to a new dimension. Hence, if all n packets have been seen, then the receiver's knowledge space is also of dimension n, in which case it must be the same as the sender's and all packets can be decoded. ■

In other words, seeing n different packets corresponds to having n linearly independent equations in n unknowns. Hence, the unknowns can be found by solving the system of equations. At this point, the file can be delivered to the TCP sink. In practice, one does not have to necessarily wait until the end of the file to decode all packets. Some of the unknowns can be found even along the way. In particular, whenever the number of equations received catches up with the number of unknowns involved, the unknowns can be found. Now, for every new equation received, the receiver sends an ACK. The congestion control algorithm uses the ACKs to control the injection of new unknowns into the coding window. Thus, the discrepancy between the number of equations and number of unknowns does not tend to grow with time, and therefore will hit zero often based on the channel conditions. As a consequence, the decoding buffer will tend to be stable.

An interesting observation is that the arguments used to show the soundness of our approach are quite general and can be extended to more general scenarios such as random linear coding based multicast over arbitrary topologies.

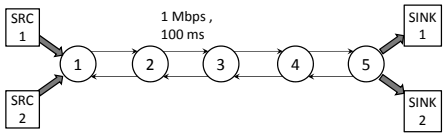


Fig. 4. Simulation topology

V. Fairness of the protocol

Here, we study the fairness property of our algorithm through simulations.

A. Simulation setup

The protocol described above is simulated using the Network Simulator (ns-2) [16]. The topology for the simulations is a tandem network consisting of 4 hops (hence 5 nodes), shown in Figure 4. The source and sink nodes are at opposite ends of the chain. Two FTP applications want to communicate from the source to the sink. There is no limit on the file size. They emit packets continuously till the end of the simulation. They either use TCP without coding or TCP with network coding (denoted TCP/NC). In this simulation, intermediate nodes do not re-encode packets. All the links have a bandwidth of 1 Mbps, and a propagation delay of 100 *ms*. The buffer size on the links is set at 200. The TCP receive window size is set at 100 packets, and the packet size is 1000 bytes. The Vegas parameters are chosen to be $\alpha = 28, \beta = 30, \gamma = 2$ (see [15] for details of Vegas).

B. Fairness and compatibility – simulation results

By fairness, we mean that if two similar flows compete for the same link, they must receive an approximately equal share of the link bandwidth. In addition, this must not depend on the order in which the flows join the network. The fairness of TCP-Vegas is a well-studied problem. It is known that depending on the values chosen for the α and β parameters, TCP-Vegas could be unfair to an existing connection when a new connection enters the bottleneck link ([17], [18]). Several solutions have been presented to this problem in the literature (for example, see [19] and references therein). In our simulations, we first pick values of α and β that allow fair sharing of bandwidth when two TCP flows without our

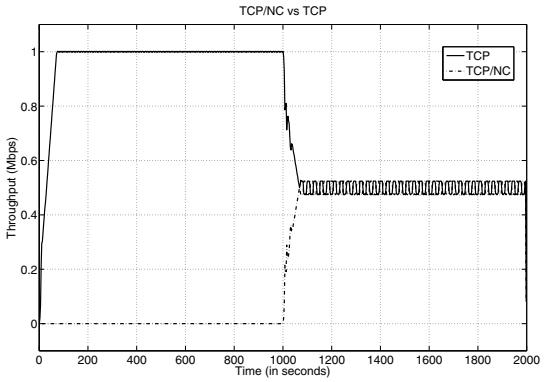


Fig. 5. Fairness and compatibility - one TCP/NC and one TCP flow

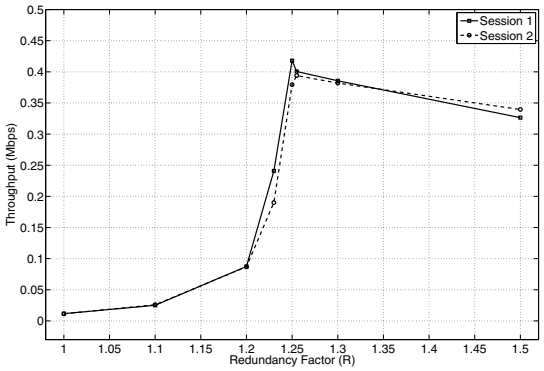


Fig. 6. Throughput vs redundancy for TCP/NC

modification compete with each other, in order to evaluate the effect of our modification on fairness. With the same α and β, we consider two cases:

Case 1: The situation where a network coded TCP flow competes with another flow running TCP without coding.

Case 2: The situation where two coded TCP flows compete with each other.

In both cases, the loss rate is set to 0% and the redundancy parameter is set to 1 for a fair comparison. In the first case, the TCP flow starts first at $t = 0.5s$ and the TCP/NC flow starts at $1000s$. The system is simulated for 2000 s. The current throughput is calculated at intervals of $2.5s$. The evolution of the throughput over time is shown in Figure 5. The figure shows that the effect of introducing the coding layer does not affect fairness. We see that after the second flow starts, the bandwidth gets redistributed fairly.

For case 2, the experiment is repeated with the same starting times, but this time both flows are TCP/NC flows. The plot for this case is essentially identical to Figure 5 (and hence is not shown here) because in the absence of losses, TCP/NC behaves identically to TCP if we ignore the effects of field size. Thus, coding can coexist with TCP in the absence of losses, without affecting fairness.

VI. Effectiveness of the protocol

We now show that the new protocol indeed achieves a high throughput, especially in the presence of losses. We first describe simulation results comparing the protocol's performance with that of TCP in Section VI-A. Next, in Section VI-B, we study the effectiveness of the random linear coding ideas in a theoretical model with idealized assumptions such as infinite buffer space, and known channel capacity. We show that in such a scenario, our scheme stabilizes the queues for all rates below capacity.

A. Throughput of the new protocol – simulation results

The simulation setup is identical to that used in the fairness simulations (see Section V-A).

We first study the effect of the redundancy parameter on the throughput of TCP/NC for a fixed loss rate of 5%. By loss rate, we mean the probability of a packet getting lost on each link. Both packets in the forward direction as well as ACKs in the reverse direction are subject to these losses. No re-encoding is allowed at the intermediate nodes. Hence, the overall probability of packet loss across 4 hops is given by $1-(1-0.05)^4$ which is roughly 19%. Hence the capacity is roughly 0.81 Mbps, which when split fairly gives 0.405 Mbps per flow. The simulation time is $10000s$.

We allow two TCP/NC flows to compete on this network, both starting at $0.5s$. Their redundancy parameter is varied between 1 and 1.5. The theoretically optimum value is approximately $1/(1-0.19) \simeq 1.23$. Figure 6 shows the plot of the throughput for the two flows, as a function of the redundancy parameter R. It is clear from the plot that R plays an important role in TCP/NC. We can see that the throughput peaks around $R = 1.25$. The peak throughput achieved is 0.397 Mbps, which is indeed close to the capacity that we calculated above. In the same situation, when two TCP flows compete for the network, the two flows see a throughput of 0.0062 and 0.0072 Mbps respectively. Thus, with the correct choice of R, the throughput for the flows in the TCP/NC case is very high compared to the TCP case. In fact, even with $R = 1$, TCP/NC achieves about 0.011 Mbps for each flow improving on TCP by almost a factor of 2.

Next, we study the variation of throughput with loss rate for both TCP and TCP/NC. The simulation parameters are all the same as above. The loss rate of all links is kept at the same value, and this is varied from 0 to 20%. We compare two scenarios – two TCP flows competing with each other, and two TCP/NC flows competing with each other. For the TCP/NC case, we set the redundancy parameter at the optimum value corresponding to each loss rate. Figure 7 shows that TCP's throughput falls rapidly as losses increase. However, TCP/NC is very robust to losses and reaches a throughput that is close to capacity. (If p is the loss rate on each link, then capacity is $(1-p)^4$, which must then be split equally.)

Figure 8 shows the instantaneous throughput in a 642 second long simulation of a tandem network with 3 hops (*i.e.*, 4 nodes), where erasure probabilities vary with time in some specified manner. The third hop is on average, the most erasure-prone link. The plots are shown for traditional TCP, TCP/NC with coding only at the source, and TCP/NC with re-encoding at node 3 (just before the worst link). The operation of the re-encoding node is very similar to that of the source – it collects incoming linear combinations in a buffer, and transmits, on average, R_{int} random linear combinations

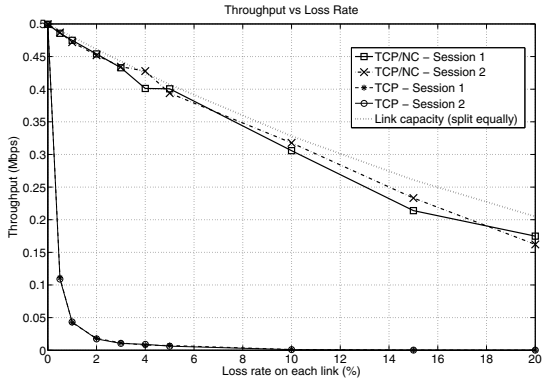
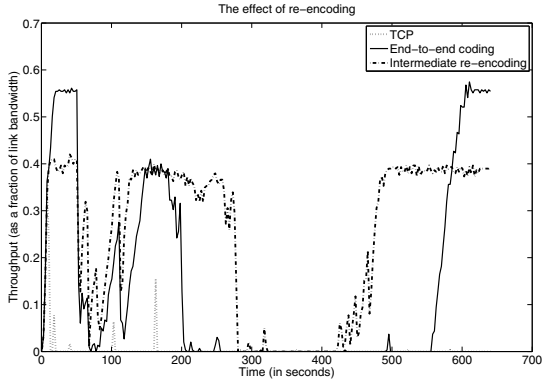


Fig. 7. Throughput vs loss rate for TCP and TCP/NC



TCP	End-to-end coding	Re-encoding at node 3 only
0.0042 Mbps	0.1420 Mbps	0.2448 Mbps

Fig. 8. Throughput with and without intermediate node re-encoding

of the buffer contents for every incoming packet. The R of the sender is set at 1.8, and the R_{int} of node 3 is set at 1.5 for the case when it re-encodes. The average throughput is shown in the table. A considerable improvement is seen due to the coding, that is further enhanced by allowing intermediate node re-encoding. This plot thus shows that our scheme is also suited to systems with coding inside the network.

Remark 1: These simulations are meant to be a preliminary study of our algorithm's performance. Specifically, the following points must be noted:

– Link layer retransmission is not considered for either TCP or TCP/NC. If allowed, this could improve the performance of TCP. However, as mentioned earlier, the retransmission approach does not extend to more general multipath routing solutions, whereas coding is better suited to such scenarios.

– The throughput values do not account for the overhead associated with the network coding headers. The main overhead is in conveying the coding coefficients and the contents of the coding window. If the source and sink share a pseudorandom number generator, then the coding coefficients can be conveyed succinctly by sending the current state of the generator. Similarly, the coding window contents can be conveyed in an incremental manner to reduce the overhead.

– The loss in throughput due to the finiteness of the field has not been modeled in the simulations. A small field might cause received linear combinations to be non-innovative, or might cause packets to be seen out of order, resulting in duplicate ACKs. However, the probability that such problems persist for a long time falls rapidly with the field size. We believe that for practical choices of field size, these issues will only cause transient effects that will not have a significant impact on performance. These effects remain to be quantified exactly.

– Finally, the decoding delay associated with the network coding operation has not been studied. We intend to focus on this aspect in experiments in the future. A thorough experimental evaluation of all these aspects of the algorithm, on a more general topology, is part of ongoing work.

B. The ideal case

In this section, we focus on an idealized scenario in order to provide a first order analysis of our new protocol. We aim to explain the key ideas of our protocol with emphasis on the interaction between the coding operation and the feedback. The model used in this section will also serve as a platform which we can build on to incorporate more practical situations.

We abstract out the congestion control aspect of the problem by assuming that the capacity of the system is fixed in time and known at the source, and hence the arrival rate is always maintained below the capacity. We also assume that nodes have infinite capacity buffers to store packets. We focus on a topology that consists of a chain of erasure-prone links in tandem, with perfect end-to-end feedback from the sink directly to the source. In such a system, we investigate the behavior of the queue sizes at various nodes.

1) System model: The network we study in this section is a daisy chain of N nodes, each node being connected to the next one by a packet erasure channel. We assume a slotted time system. The source generates packets according to a Bernoulli process of rate λ packets per slot. The point of transmission is at the very beginning of a slot. Just after this point, every node transmits one random linear combination of the packets in its queue. We ignore propagation delay. Thus, the transmission, if not erased by the channel, reaches the next node in the chain almost immediately. However, the node may use the newly received packet only in the next slot's transmission. We assume perfect, delay-free feedback from the sink to the source. In every slot, the sink generates the feedback signal after the instant of reception of the previous node's transmission. The erasure event happens with a probability $(1 - \mu_i)$ on the channel connecting node i and $(i + 1)$, and is assumed to be independent across different channels and over time. Thus, the system has a capacity $\min_i \mu_i$ packets per slot. We assume that $\lambda < \min_i \mu_i$, and define the load factor $\rho_i = \lambda/\mu_i$. The relation between the transmitted linear combination and the original packet stream is conveyed in the packet header. We ignore this overhead for the analysis in this section.

Remark 2: This model and the following analysis also works for the case when not all intermediate nodes are involved in the network coding. If some node simply forwards the incoming packets, then we can incorporate this in the following way. An erasure event on either the link entering this node or the link leaving this node will cause a packet

erasure. Hence, these two links can be replaced by a single link whose probability of being ON is simply the product of the ON probabilities of the two links being replaced. Thus, all non-coding nodes can be removed from the model, which brings us back to the same situation as in the above model.

2) Queue update mechanism: Each node transmits a random linear combination of the current contents of its queue and hence, the question of how to update the queue contents becomes important. In every slot, the sink sends an ACK directly to the source, containing the index of the oldest packet not yet seen by the sink. Upon receiving the ACK, the source drops all packets from its queue with an index lower than the sink's request. The intermediate nodes do not have direct feedback from the sink. Hence, the source has to inform them about the sink's ACK. This information is sent on the same erasure channel used for the regular transmission. This feed-forward of the sink's status is modeled as follows. Whenever the channel entering an intermediate node is in the ON state (*i.e.*, no erasure), the node's version of the sink's status is updated to that of the previous node. In practice, the source need not transmit the sink's status explicitly. The intermediate nodes can infer it from the set of packets that have been involved in the linear combination – if a packet is no longer involved, that means the source must have dropped it, implying that the sink must have ACKed it already. Whenever an intermediate node receives an innovative packet, this causes the node to see a previously unseen packet. The node performs a Gaussian elimination to compute the witness of the newly seen packet, and adds this to the queue. Thus, intermediate nodes store the witnesses of the packets that they have seen. The queue update rule is similar to that of the source. An intermediate node drops the witness of all packets up to but excluding the one requested by the sink. This is based on the most updated version of the sink's status known at the intermediate node.

3) Queuing analysis: The following theorem shows that if we allow coding at intermediate nodes, then it is possible to achieve the capacity of the network, namely $\min_i \mu_i$. Note that this theorem also implies that if we only allow forwarding at some of the intermediate nodes, then we can still achieve the capacity of a new network derived by collapsing the links across the non-coding nodes, as described in Remark 2.

Theorem 2: As long as $\lambda < \mu_k$ for all $0 \leq k < N$, the queues at all the nodes will be stable. The expected queue size in steady state at node k ($0 \leq k < N$) is given by:

$$\mathbb{E}[Q_k] = \sum_{i=k}^{N-1} \frac{\rho_i(1-\mu_i)}{(1-\rho_i)} + \sum_{i=1}^{k-1} \rho_i$$

An implication: Consider a case where all the ρ_i's are equal to some ρ. Then, the above relation implies that in the limit of heavy traffic, *i.e.*, $\rho \to 1$, the queues are expected to be longer at nodes near the source than near the sink.

A useful lemma: The following lemma shows that the random linear coding scheme has the property that every time there is a successful reception at a node, the node sees the next unseen packet with high probability, provided the field is large enough. This fact will prove useful while analyzing the evolution of the queues.

Lemma 1: Let S_A and S_B be the set of packets seen by two nodes A and B respectively. Assume $S_A \backslash S_B$ is non-empty. Suppose A sends a random linear combination of its witnesses of packets in S_A and B receives it successfully. The probability that this transmission causes B to see the oldest packet in $S_A \backslash S_B$ is $\left(1 - \frac{1}{q}\right)$, where q is the field size.

Proof: Let M_A be the RREF basis matrix for A. Then, the coefficient vector of the linear combination sent by A is $\mathbf{t} = \mathbf{u}M_A$, where $\mathbf{u}$ is a vector of length $|S_A|$ whose entries are independent and uniformly distributed over the finite field $\mathbb{F}_q$. Let d^* denote the index of the oldest packet in $S_A \backslash S_B$.

Let M_B be the RREF basis matrix for B before the new reception. Suppose $\mathbf{t}$ is successfully received by B. Then, B will append $\mathbf{t}$ as a new row to M_B and perform Gaussian elimination. The first step involves subtracting from $\mathbf{t}$, suitably scaled versions of the pivot rows such that all entries of $\mathbf{t}$ corresponding to pivot columns of M_B become 0. We need to find the probability that after this step, the leading non-zero entry occurs in column d^*, which corresponds to the event that B sees packet d^*. Subsequent steps in the Gaussian elimination will not affect this event. Hence, we focus on the first step.

Let P_B denote the set of indices of pivot columns of M_B. In the first step, the entry in column d^* of $\mathbf{t}$ becomes

$$t'(d^*) = t(d^*) - \sum_{i \in P_B, i < d^*} t(i) \cdot M_B(r_B(i), d^*)$$

where $r_B(i)$ is the index of the pivot row corresponding to pivot column i in M_B. Now, due to the way RREF is defined, $t(d^*) = u(r_A(d^*))$, where $r_A(i)$ denotes the index of the pivot row corresponding to pivot column i in M_A. Thus, $t(d^*)$ is uniformly distributed. Also, for $i < d^*$, $t(i)$ is a function of only those $u(j)$'s such that $j < r_A(d^*)$. Hence, $t(d^*)$ is independent of $t(i)$ for $i < d^*$. From these observations and the above expression for $t'(d^*)$, it follows that for any given M_A and M_B, $t'(d^*)$ has a uniform distribution over $\mathbb{F}_q$, and the probability that it is not zero is therefore $\left(1 - \frac{1}{q}\right)$. ■

For the queuing analysis, we assume that a successful reception always causes the receiver to see its next unseen packet, provided the transmitter has already seen it. A consequence of this assumption is that the set of packets seen by a node is always a contiguous set, with no gaps in between. In particular, there is no repeated ACK due to packets being seen out of order. The above lemma argues that these assumptions become more and more valid as the field size increases. In reality, some packets may be seen out of order resulting in larger queue sizes. However, we believe that this effect is minor and can be neglected for a first order analysis.

The expected queue size: We define arrival and departure as follows. A packet is said to arrive at a node when the node sees the packet for the first time. A packet is said to depart from the node when the node drops the witness of that packet from its queue. For each intermediate node, we now study the

expected time between the arrival and departure of an arbitrary packet at that node as this is related to the expected queue size at the node, by Little's law.

Proof of Theorem 2: Consider the k^{th} node, for $1 \leq k < N$. The time a packet spends in this node's queue has two parts:

1) *Time until the packet is seen by the sink:*

The difference between the number of packets seen by a node and the number of packets seen by the next node downstream essentially behaves like a $Geom/Geom/1$ queue. The Markov chain governing this evolution is identical to that of the virtual queues studied in [4]. Given that a node has seen the packet, the time it takes for the next node to see that packet corresponds to the waiting time in a virtual queue. For a load factor of ρ and a channel ON probability of μ, the expected waiting time was derived in [4] to be $\frac{(1-\mu)}{\mu(1-\rho)}$, using results from [20]. Now, the expected time until the sink sees the packet is the sum of $(N-k)$ such terms, which gives $\sum_{i=k}^{N-1} \frac{(1-\mu_i)}{\mu_i(1-\rho_i)}$.

2) *Time until sink's ACK reaches intermediate node:*

The ACK informs the source that the sink has seen the packet. This information needs to reach node k by the feed-forward mechanism. The expected time for this information to move from node i to node $i+1$ is the expected time until the next slot when the channel is ON, which is just $\frac{1}{\mu_i}$ (since the i^{th} channel is ON with probability μ_i). Thus, the time it takes for the sink's ACK to reach node k is given by $\sum_{i=1}^{k-1} \frac{1}{\mu_i}$.

The total expected time T_k a packet spends in the queue at the k^{th} node ($1 \leq k < N$) can thus be computed by adding the above two terms. Now, assuming the system is stable (*i.e.*, $\lambda < \min_i \mu_i$), we can use Little's law to derive the expected queue size at the k^{th} node, by multiplying T_k by λ:

$$\mathbb{E}[Q_k] = \sum_{i=k}^{N-1} \frac{\rho_i(1-\mu_i)}{(1-\rho_i)} + \sum_{i=1}^{k-1} \rho_i \qquad \blacksquare$$

VII. Conclusions and future work

In this work, we propose a new approach to congestion control on lossy links based on the idea of random linear network coding. We introduce a new acknowledgment mechanism that plays a key role in incorporating coding into the control algorithm. From an implementation perspective, we introduce a new network coding layer between the transport and network layers on both the source and receiver sides. Thus, our changes can be easily deployed in an existing system.

A salient feature of our proposal is that it is simultaneously compatible with the case where only end hosts perform coding (thereby preserving the end-to-end philosophy of TCP), as well as the case where intermediate nodes perform network coding. Theory suggests that a lot can be gained by allowing intermediate nodes to code as well. Our scheme naturally generalizes to such situations. Our simulations show that the proposed changes lead to large throughput gains over TCP in lossy links, even with coding only at the source. For instance, in a 4-hop tandem network with a 5% loss rate on each link, the throughput goes up from about 0.007 Mbps to about 0.39 Mbps for the correct redundancy factor. Intermediate node coding further increases the gains.

In the future, we plan to understand the impact of field size on throughput. While our current simulations assume a large field size, we believe that in practice, a large part of the gains can be realized without too much overhead. We also wish to understand the overhead associated with the coding and decoding operations in a practical setting.

This paper presents a new framework for combining coding with feedback-based rate-control mechanisms in a practical way. It is of interest to extend this approach to more general settings such as network coding based multicast over a general network. Even in the point-to-point case, these ideas can be used to implement a multipath-TCP based on network coding.

References

[1] C. Fragouli, D. S. Lun, M. Médard, and P. Pakzad, "On feedback for network coding," in *Proc. of 2007 Conference on Information Sciences and Systems (CISS 2007)*.

[2] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. of Allerton Conference on Communication, Control, and Computing*, 2003.

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

[4] J. K. Sundararajan, D. Shah, and M. Médard, "ARQ for network coding," in *Proc. of IEEE International Symposium on Info. Theory (ISIT)*, 2008.

[5] S. Rangwala, A. Jindal, K.-Y. Jang, K. Psounis, and R. Govindan, "Understanding congestion control in multi-hop wireless mesh networks," in *Proc. of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 2008.

[6] S. Paul, E. Ayanoglu, T. F. L. Porta, K.-W. H. Chen, K. E. Sabnani, and R. D. Gitlin, "An asymmetric protocol for digital cellular communications," in *Proceedings of INFOCOM '95*, 1995.

[7] A. DeSimone, M. C. Chuah, and O.-C. Yue, "Throughput performance of transport-layer protocols over wireless LANs," *IEEE Global Telecommunications Conference (GLOBECOM '93)*, pp. 542–549 Vol. 1, 1993.

[8] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *ACM Wireless Networks*, vol. 1, no. 4, pp. 469–481, December 1995.

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

[10] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, 2003.

[11] D. S. Lun and T. Ho, *Network Coding: An Introduction*. Cambridge University Press, 2008.

[12] T. Ho, "Networking from a network coding perspective," PhD Thesis, Massachusetts Institute of Technology, Dept. of EECS, May 2004.

[13] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in the Air: Practical Wireless Network Coding," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 497–510, June 2008.

[14] J. Lacan and E. Lochin, "On-the-fly coding to enable full reliability without retransmission," ISAE, LAAS-CNRS, France, Tech. Rep. [Online]. Available: http://arxiv.org/pdf/0809.4576

[15] L. S. Bramko, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of the SIGCOMM '94 Symposium*, August 1994.

[16] "Network Simulator (*ns-2*)," in *http://www.isi.edu/nsnam/ns/*.

[17] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas revisited," in *Proceedings of INFOCOM '00*, 2000.

[18] C. Boutremans and J.-Y. Le Boudec, "A note on fairness of TCP Vegas," in *Proceedings of Broadband Communications*, 2000.

[19] J. Mo, R. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in *Proceedings of INFOCOM '99*, 1999.

[20] J. J. Hunter, *Mathematical Techniques of Applied Probability, Vol. 2, Discrete Time Models: Techniques and Applications*. NY: Academic Press, 1983.