Multiple Network Coded TCP Sessions in Disruptive Wireless Scenarios

Chien-Chia Chen¹, Clifford Chen², Joon-Sang Park³, Soon Y. Oh⁴, Mario Gerla¹, M. Y. Sanadidi¹

¹Computer Science Dept., UCLA, Los Angeles, CA 90095, USA
²Carnegie Mellon University, Silicon Valley, NASA Research Park, CA 94035, USA
³Dept. of Computer Engineering, Hongik University, Seoul, Korea
⁴Utopia Compression, LA, CA 90064, USA

chenchiennentla@ucla.edu, cliff.chen@sv.cmu.edu, jsp@hongik.edu, soonoh@utopia.compression.com, {gerla, medy}@cs.ucla.edu

Abstract—TCP over wireless networks is challenging due to random losses and DATA-ACK interference. Random linear coding schemes have been proposed to improve TCP robustness against extreme random losses, but a critical issue still remains of DATA-ACK interference. To address this problem, we use inter-flow coding between DATA and ACK at potential intersecting nodes to mitigate self-interference. In addition, a pipelined random linear coding scheme with adaptive redundancy is introduced to overcome high loss rates over unreliable links. The resulting coding scheme, ComboCoding, combines inter-flow and intra-flow coding to provide robust, fair TCP communication in multi-flow disruptive wireless networks. Simulation results show that TCP with ComboCoding delivers higher throughput than other coding options in high loss scenarios. Moreover, we study the behavior of multiple TCP flows intersecting and interfering with each other in the same ad hoc network. The results show that ComboCoding consistently provides better fairness among multiple co-existing TCP sessions when compared with TCP without coding. The main contributions of this paper are: the practical design of random linear coded packets and XOR coded packets on bi-directional streams (DATA and ACK), the adaptive redundancy control scheme, and the multiple coded TCP sessions fairness evaluation.

Keywords—Network Coding, TCP, Fairness, Random Linear Coding, XOR Coding, Wireless Multihop, Lossy Channels

I. INTRODUCTION

The Transport Control Protocol (TCP) is the most commonly used reliable transport protocol in the Internet. In addition to end-to-end reliable transmission, TCP also provides fair congestion control for better sharing of network resources. TCP-NewReno, the most widely used TCP variant, employs a loss-based congestion control algorithm that interprets all packet losses as congestion losses caused by router buffer overflow. Such an interpretation works for the wired Internet, the environment TCP-NewReno was originally designed for, where most links are point-to-point. However, this design is suboptimal in wireless ad hoc networks, where the majority of loss is due to unpredictable link quality, random interference, and in tactical scenarios, jamming. It has been shown that in wireless multihop scenarios, TCP suffers significantly from misinterpreting random loss as congestion loss. TCP DATA and ACKs also contend for the shared wireless medium, which causes self-induced collisions.

To improve loss-based congestion control in wireless networks, a number of approaches have been proposed, including Loss Discrimination Algorithms (LDA) [1] and TCP parameter tuning [2]. A recently proposed approach to help TCP in wireless networks is to exploit network coding, which has been successfully applied to a variety of network environments and traffic scenarios [3]. Random linear coding further enhances the practical code implementation in wireless mobile networks [4-5]. Network coding has been shown to help alleviate wireless interference by reducing the number of transmissions in multicast or multi-flow environments [6-7]. In addition, network coding helps overcome high loss rates in wireless networks [8-9].

The benefits of coding help improve throughput, but every solution so far does not solve every problem we need for robust TCP communication in wireless networks. In COPE [6], S. Katti et al. argue that network coding does not significantly improve TCP performance. Y. Huang et al. later rebut that it is mainly because the COPE design does not consider bi-directionality, i.e., in TCP, DATA and ACK flows are in opposite directions [10]. Thus, instead of mixing arbitrary number of flows together, they presented an approach to alleviate TCP self-interference by opportunistically XORing DATA and ACK packets within a TCP session. The coding of two flows in opposite directions is referred to as inter-flow coding. L. Scalia et al. further propose a similar idea, where the TCP throughput is improved by opportunistically XORing the TCP ACK and DATA flows [11]. Following the work presented in [11], P. David et al. proposed a MAC layer modification to further improve the coding gain [12]. One of the earliest proposals to improve TCP in lossy networks is based on an intra-flow random linear coding scheme [13]. Intra-flow implies that only TCP DATA packets (not ACKs) within a single TCP session are coded. Although performance is improved, the semantic of TCP ACKs is redefined by requiring TCP modifications or an additional adaptation layer at both sender and receiver.

All the above solutions address either the ACK interference problem or the DATA random loss problem, but not both. In this paper, we present a network coding scheme that integrates inter-flow and intra-flow coding to address both problems. The proposed coding scheme, ComboCoding, provides a robust, comprehensive solution for TCP in lossy wireless scenarios by mitigating random losses using random linear intra-flow coding, and reducing interference through encoding the ACK and DATA packets together. Moreover, it adjusts coding redundancy adaptively to further deal with unstable wireless link quality.

The contributions of ComboCoding are as follows. (1) ComboCoding provides a practical implementation that combines inter- and intra-flow coding together; (2) ComboCoding features a novel loss adaptation algorithm that effectively handles transient, unstable link conditions. (3) ComboCoding is transparent to TCP and applications and is thus forward compatible with any future upper layer protocol redesigns. (4) ComboCoding differs from [12]...
because it requires neither software nor hardware modifications to the MAC or lower layer.

Note that in order to deploy our proposed coding scheme, proper modifications on the network layer at the wireless nodes are inevitable as for any network coding implementation. However, modifying the network layer at the node itself is well accepted and is part of the ad hoc network design [14].

In addition, we also study how intra-flow coding and inter-flow coding perform under multiple TCP sessions. The interaction/interference among TCP sessions is a key concern in ad hoc networks because of fairness problems. Several studies have been made to address this problem [15]. It is critical to study the fairness problem when TCP flows are network coded.

The rest of the paper is organized as follows. The literature on combining inter- and intra-flow coding is reviewed in Section II. The design and implementation of the proposed ComboCoding scheme are presented in Section III. Section IV evaluates the performance of ComboCoding in single and multiple session settings. Section V concludes the paper.

II. RELATED WORK

Traditional TCP coding schemes address either the random loss issue or the self-interference issue. Only recently was a combined intra- and inter-flow coding scheme proposed [16], which considers general opposing streams (with TCP ACK as another case of opposing stream). It applies random linear coding to all streams and also mixes opposing flows using XOR-based coding. This scheme has two major limitations when run under TCP traffic. First, as pointed out in [16], their design has an unsolved undecodable problem when multiple flows are mixed together. In contrast, ComboCoding mixes only TCP DATA and ACK and only over a single hop, which was proved in [11] to guarantee 100% decodability. Second, their design relies on ACK-based redundancy control, which has a very high overhead in disruptive networks. This is because both coded packets and corresponding ACKs must be delivered successfully in order to move to the next generation. In contrast, ComboCoding has no control overhead since it adjusts the coding redundancy based on loss rate samples carried by overheard packets.

Following [16], H. Seferoglu et al. proposed a similar design in [17], which is based on an improved version of COPE [6] and a similar random linear coding scheme as in [13]. Based on the Network Utility Maximization (NUM) technique, they formulate their design as an optimization problem. However, similar to [16], their work mixes arbitrary number of flows together, which causes difficulties in infrastructure-less ad hoc network deployment. In addition, a known limitation of NUM is the fact that it requires many network parameters, such as number of flows, buffer length at all nodes, and buffer occupation of every flow as inputs to the optimization framework. As a consequence, re-optimization is triggered whenever a parameter changes value. In contrast, ComboCoding requires none of the above parameters as inputs. In fact it is self-sufficient in estimating losses and detecting link condition changes. Based on link condition estimates, the adaptation algorithm adjusts local redundancy.

III. CODING SCHEME DESIGN AND IMPLEMENTATION

This section describes the design and implementation of ComboCoding including the intra-flow coding scheme, the inter-flow coding scheme and the loss adaptation algorithm.

A. Intra-Flow Coding

The intra-flow coding scheme we choose is not a conventional batch-based scheme but a novel pipelined random linear coding scheme. Our early report in [18] introduced a preliminary version Pipeline Coding, and demonstrated how it reduces the end-to-end coding delay for UDP applications. Following the concept of progressive encoding and decoding, in this paper we further extend the original Pipeline Coding design and implement an improved version to support TCP over disruptive environments.

Consider a sequence of equal-sized packets $P_1, P_2, P_3, \ldots$ that are generated by an application and let $k$ denote the number of packets in a coding generation. A coded packet $c$ in the $i^{th}$ generation is defined as:

$$c = \sum_{j=1}^{m} e_j P_{j+i},$$

where $m$ is the number of data packets currently in the generation buffer, $e_j$ is a random element in a particular Galois field $\mathbb{F}_{2^e}$, and $i \times k$ is the total number of packets transmitted before the $i^{th}$ generation. Throughout the paper, lowercase boldface letters represent vectors, frames, or packets, uppercase letters represent matrices, and italics represent variables or fields in the packet header. All arithmetic operations are over $\mathbb{F}_{2^e}$, so that data packets $P_i$ and coded packets $c$ are also regarded as vectors over $\mathbb{F}_{2^e}$.

Based on Eq. (1), the source generates a coded packet upon the arrival of each data segment. Given the source coding redundancy $r$, where $r \geq 1$, the source produces $k \times r$ coded packets for each generation. If all coded packets are delivered without errors, the destinations can construct the following lower triangular matrix by simply sorting the rank of each packet:

$$
\begin{bmatrix}
  e_1 \\
  \vdots \\
  e_t \\
\end{bmatrix} = 
\begin{bmatrix}
  e_1^{(1)} & 0 & \cdots & 0 \\
  e_2^{(1)} & e_2^{(2)} & \cdots & \vdots \\
  \vdots & \vdots & \ddots & \vdots \\
  e_t^{(1)} & e_t^{(2)} & \cdots & e_t^{(k)} \\
\end{bmatrix}
\begin{bmatrix}
P_1 \\
\vdots \\
P_t \\
\end{bmatrix}
$$

(2)

The above linear equation can be solved progressively without waiting for generation completion. For example, upon receiving $e_t$, destinations can decode $P_t$, and so on. Intermediate nodes also participate in re-encoding, as explained in [19]. In addition, intermediate nodes also have a forwarding redundancy to determine how many re-encoded packets to generate upon receiving each innovative packet.

Fig. 1 shows an example of the pipelined random linear coding, with generation size $k = 4$ and source coding redundancy $r = 1.25$. Note that we have omitted packet re-encoding at intermediate nodes for simplicity. Data packets are encoded instantly upon arrival and decoded immediately at the destination. This allows our intra-flow coding module to avoid triggering the retransmission timeout of DATA packets, which
is critical to TCP. This is the fundamental reason why the pipelined random linear coding is compatible with TCP, while conventional batch network coding is not.

In addition, the pipelined random linear coding can partially recover a subset of the data packets in a generation and deliver them to the upper layer. This significantly differs from batch-based coding in that batch coding either delivers a complete generation to upper layers or discards the entire generation. For example, assuming that \( e_1 \) of Fig. 1 is lost, data packets \( #7 \) and \( #8 \) will never have a chance to be decoded, regardless of which coding scheme is used. With Batch Coding, none of the data packets in the 2nd generation can be decoded, whereas with pipelined coding, we can still decode data packets \( #5 \) and \( #6 \).

Note that the improved Pipeline Coding used in our scheme does not require any TCP modification or any additional adaptation layer since it functions at the network layer, which is the key difference between our approach and the scheme proposed in [13]. The approach in [13] encodes all packets in the congestion window and redefines the semantic of TCP acknowledgements, while our proposed scheme uses its own coding generation buffer that has no relationship with the TCP congestion window. Since ComboCoding does not guarantee reliability, a loss or a timeout event still triggers TCP retransmission.

**B. Inter-Flow Coding**

Our inter-flow coding design is similar to COPE [6], but with modifications to address special types of bi-directional flows—TCP DATA and ACKs. The inter-flow coding used in ComboCoding improves a previous study, PiggyCode [11], and functions at the network layer.

The main goal of the original PiggyCode is to mitigate TCP self-interference by opportunistically XORing TCP DATA and ACKs. The inter-flow coding used in ComboCoding improves a previous study, PiggyCode [11], and functions at the network layer.

The main advantage of such an inter-flow coding is that it requires no TCP modification. However, as the recent study in [12] points out, the major challenge here is that an inter-flow coded packet is conceptually a “dual-cast” packet in the link layer, in that there are two intended receivers that should receive the packet correctly. The authors argued that PiggyCode coding gain is limited due to the lack of dual MAC layer ACK support in 802.11. Hence in [12], a MAC layer modification is proposed to support a “dual-MAC-ACK” so that both intended receivers will send a MAC-ACK to the PiggyCode sender. With this extra dual-cast support, they reported a TCP throughput improvement of up to 100% compared to the original PiggyCode.

One of our goals is to propose a coding scheme that requires changes only to the network layer, so ComboCoding adopts neither the original PiggyCode nor the above dual-cast enhancement. Rather, we introduce in ComboCoding an inter-flow coding buffer at the network layer. All TCP DATA packets will be buffered in the coding module at each relay for a given inter-flow coding timer \( T \). If an ACK arrives before timeout, the DATA will be XOR’ed with the ACK and the inter-flow coded packet will be sent in the DATA flow direction. If no ACK arrives in \( T \), the DATA will be re-encoded based on the algorithm given in the previous section and sent to the next-hop via unicasting.

In practice, there are two major concerns for such an implementation. First, the inter-flow coded packet is protected by MAC layer reliability only in the DATA flow direction. We argue that this causes only minimal impact as ACKs in TCP are cumulative, so an infrequent loss of ACK is not as severe as a loss of DATA; we have also validated this from our simulation results. Second, in TCP specifications, TCP allows full-duplex communications (although this occurs very rarely in practice since the data flow is in just one direction; the reverse direction only sends application level control packets) and thus TCP ACKs can be piggybacked on DATA segments. In our design, we do not distinguish DATA segments from ACK segments, but rather, by TCP DATA flow and TCP ACK flow we refer to the two flows associated with the same TCP session but in opposite directions. Similarly, a “DATA packet” refers to the packet in one of the two flows and so does “ACK packets.”

**C. Loss Adaptation Algorithm**

Since the link quality in emergency and tactical ad hoc networks varies significantly over time, we propose a feedback-based redundancy control algorithm to adaptively adjust the coding and forwarding redundancy. In random linear coding schemes, the ideal link-local redundancy is \( 1/(1-p) \), where \( p \) is the link loss probability. As \( p \) is not given and can change at all times, ComboCoding adjusts the redundancy based on loss rate estimates for each link. We have found through simulations that the same redundancy should be applied to both the TCP DATA and ACK flows due to the use of symmetric links. Therefore, we collect loss rate samples only from the TCP DATA flow. Note that the algorithm is designed for single path topologies but is capable of being extended to multiple path scenarios.

Our loss estimate algorithm relies on loss rate samples, which are carried in each coded packet by a new field in the
network header. This field tracks the number of coded packets received in the current generation at node $i$, which is denoted by $N_i$. This count is carried in the TCP ACKs, which are sent by unicasting, as well as in the re-encoded TCP DATA packets, which are received by overhearing. Assuming node $i+1$ is the downstream neighbor of node $i$ in the TCP DATA flow, for every received $N_i$ we derive the “instantaneous” link loss rate from node $i$ to node $i+1$ as follows:

$$p_i^0 = \frac{M_i - N_{i+1}}{M_i},$$

(3)

where $M_i$ is the number of re-encoded packets sent from node $i$ to node $i+1$, which is recorded locally at node $i$. To avoid aliasing, we further take the average of every ten $p_i^0$ samples, which is represented by $\bar{p}_i^0$.

Since the loss rate may vary significantly over time, a smoothed loss rate is then calculated by taking the exponential moving average of the instantaneous link loss rate as follows:

$$\bar{p}_i^1 = \bar{p}_i^0 + \alpha \times (\bar{p}_i^0 - \bar{p}_i^0),$$

(4)

where $\alpha$ is the smoothing factor, which is set to 1/6 in our simulation as it works the best in all of our simulations. The redundancy for the link from node $i$ to node $i+1$ is then set as follows:

$$R_i = (K_i, -1) + \frac{1}{1 - \bar{p}_i^1},$$

(5)

where $K_i$ is the base redundancy that is needed at node $i$ in the absence of losses. $K_i$ is used to introduce extra redundancy to recover packets that have been lost and to compensate for future potential packet losses. In our simulation, $K_i$ is set to 1.6.

IV. SIMULATION RESULTS

The proposed ComboCoding scheme was tested on QualNet 4.5, where the module is implemented at the network layer. We compare ComboCoding with the original PiggyCode [11] and the unmodified Pipeline Coding [18, 20]. The simulation topology is a string as shown in Fig. 3. Nodes are 250 meters apart, and the Physical and MAC layer protocols are standard 802.11g, with RTS/CTS disabled. The channel bit-rate is 54Mbps. To exploit the MAC layer reliability, nodes communicate using pseudo-broadcast as in [6]. The transport layer protocol is TCP-NewReno and the application traffic is a Generic-FTP, which continuously generates packets of 1500 bytes and TCP backlogged. The generation size of the random linear coding ( intra-flow coding) module is set to 16 packets. The inter-flow coding timer is set to 4ms, i.e., TCP DATA packets will be buffered in the inter-flow coding module for up to 4ms. In all simulation sets, variable link loss rates of up to 50% are introduced in order to simulate a challenging environment subject to random interference and jamming. Note that we assume routing is a given input to our problem and thus all simulations are based on single-path topologies with variable hops and different number of flows in the network.

Fig. 3 Simulation Topology

A. ComboCoding Evaluation

In the first set of simulations, the loss adaptation in ComboCoding is disabled in order to evaluate the coding performance gain without being affected by other factors.

Fig. 4 presents the goodput-to-loss curves of TCP-NewReno for the following cases: no coding, PiggyCode, Pipeline Coding, and ComboCoding. We notice that for 0% loss, PiggyCode outperforms all other schemes as it reduces interference without introducing significant coding overhead. Pipeline Coding performs worst, because in order to reduce coding delay, it adopts a non-uniform inclusion of original packets into a coded packet. Because of this property, pipelined based random linear coding requires a relatively higher redundancy as discussed in our preliminary report [14]. Unlike Pipeline Coding, under 0% loss, TCP ComboCoding still achieves the same throughput as TCP with no coding, which confirms the fact that ComboCoding does not introduce a penalty in normal network conditions.

![Fig. 4 Goodput-to-Loss (3-hop String)](image)

As the packet error rate increases, the throughput of TCP without coding deteriorates rapidly, and collapses beyond 30% loss. This is because without redundant packet transmission, TCP throughput is inversely proportional to the square root of the packet loss rate as shown in [21]. By virtue of network coding redundancy, both Pipeline Coding and ComboCoding are more robust to losses. Most importantly, ComboCoding is significantly strengthened by inter-flow coding, and as a result its throughput is consistently higher than Pipeline Coding throughput.

Note that in this set of simulations, both Pipeline Coding and ComboCoding are equipped with a coding redundancy that was experimentally tuned to their optimal parameters.

B. Adaptation to Time-Varying Jamming

We next consider a 7-hop string topology with a loss rate that varies dynamically over time. This is of particular importance in tactical environments, because it represents a time varying external interference such as intermittent jamming.

Fig. 5 shows the goodput over time plot in the time-varying scenario, where the application starts sending packets at time 20 seconds, and the packet loss rate for all links is 0%, 40%, and 20% during the time intervals of 20–50 seconds, 50–80 seconds, and 80–110 seconds respectively. All other simulation parameters remain the same and we tested both ComboCoding with and without the loss adaptation feature.
Note that both non-adaptive Pipeline Coding and the non-adaptive ComboCoding operate with a fixed redundancy that performs well under 20% loss rate.

In Fig. 5, we first notice that while links are perfectly reliable, TCP with no coding and PiggyCode outperform all other coding schemes. Pipeline Coding and both adaptive and non-adaptive ComboCoding perform worse because of the extra overhead due to the redundancy of intra-flow coding.

In the interval from 50 to 80 seconds, a 40% packet loss rate is introduced on every link. During this interval, all non-adaptive coding schemes drop to almost zero throughput, while the loss adaptive ComboCoding still achieves around 0.75Mbps. This period shows the importance of loss adaptation and the effectiveness of redundancy, as the adaptive ComboCoding quickly reacts to losses and continues to perform with stable throughput.

From 80 to 110 seconds, the per link packet loss rate is lowered from 40% to 20%. We notice that TCP with PiggyCode reemerges but it is very unstable due to the lack of proper redundancy. Furthermore, TCP with Pipeline coding and with non-adaptive ComboCoding takes a long time to stabilize because random linear coding needs time to discard undecodable generations resulting from loss. The best coding scheme is adaptive ComboCoding, which instantly reacts to loss reduction and delivers high and stable throughput.

C. Multiple-Flow Scenarios

In our last set of simulations, more coexisting TCP flows are introduced to further evaluate the adaptive ComboCoding performance in complex topologies. Two topologies, as shown in Fig. 6, are used: X-topology and grid topology. In the X-topology, there are two coexisting flows, and in the grid, there are four. All other simulation parameters remain the same. Note that nodes can hear diagonally so at most five nodes can fall within the same collision domain. Fig. 7 shows the goodput for different coding options under the 2-flow X-topology.

We notice that without random linear coding, TCP No Coding and TCP PiggyCode both encounter severe capturing problems. This was expected since TCP is prone to capture as demonstrated in several previous studies [15]. Also, as expected, the aggregate throughput is high, i.e., it is the same as the single TCP throughput since only one flow is transmitting at a time. Unfortunately, this solution is unacceptable, as it may shut off one TCP session for an indeterminate amount of time. Once the random linear coding is enabled, the two flows start sharing the bandwidth fairly. The aggregate throughput is slightly lower than for the non-coded TCP, since now some of the bandwidth is wasted to manage the fair sharing among the flows. ComboCoding further outperforms Pipeline Coding thanks to the inter-flow coding.

To understand capture, note that due to extreme losses, TCP No Coding and PiggyCode suffer frequent timeout events and thus are both in the slow start phase all the time. This was observed from the congestion window vs. time plot. Due to exponential window expansion during slow start, bottleneck buffers tend to be filled up by one of the flows, resulting in capture. By inspecting the congestion window behavior of the TCP Pipeline Coding and ComboCode, we noticed that these flows stay mostly in the congestion avoidance phase. This leads to more stable flows, and results in better fairness.

We next extend the topology to a 4-flow grid as shown in Fig. 6(b). The results in figures 8 and 9 show the overall goodput of all four flows and the instantaneous Jain’s fairness index respectively [22]. Consistent with the X-topology, both
We also demonstrated that under multi-flow scenarios, in the coding schemes and quickly adapts to changes in link quality. Was tested on a longer string topology with time-varying link coding delivers only 200Kbps. The adaptive ComboCoding 30% per link packet loss rate, while TCP-NewReno with no ComboCoding successfully achieves 2 Mbps throughput with simulation results show that in a 3-hop string topology, the interference between DATA and ACKs within a TCP situation where conventional TCP collapses, ComboCoding reduces other. For ComboCoding and Pipeline Coding, the overall throughput is lower but the fairness is almost optimal since losses caused by either interference or link quality are overcome by coding, so all four flows are transmitting and sharing the bandwidth.

TCP No Coding and PiggyCode provide a higher aggregate throughput, but a much poorer fairness among flows. In both non-coding runs, flows 1 and 2 and flows 3 and 4 capture the channel in turns, as those two flows do not interfere with each other. For ComboCoding and Pipeline Coding, the overall throughput is lower but the fairness is almost optimal since losses caused by either interference or link quality are overcome by coding, so all four flows are transmitting and sharing the bandwidth.

V. Conclusion

In this paper, we presented a novel coding scheme, ComboCoding, which combines intra- and inter-flow coding and features a novel loss adaptation algorithm. By exploiting the benefits of both types of coding, ComboCoding reduces the interference between DATA and ACKs within a TCP session and also exhibits robustness to high link loss rates. The simulation results show that in a 3-hop string topology, ComboCoding successfully achieves 2 Mbps throughput with 30% per link packet loss rate, while TCP-NewReno with no coding delivers only 200Kbps. The adaptive ComboCoding was tested on a longer string topology with time-varying link loss, and simulation results show that it outperforms all other coding schemes and quickly adapts to changes in link quality. We also demonstrated that under multi-flow scenarios, in the situation where conventional TCP collapses, ComboCoding manages to stay fair and stable. More general topologies and encoding schemes will be tested and will be the subject of future studies.

Acknowledgement

J.-S. Park was supported in part by National Research Foundation of Korea (2010-0005334, 2010-0027410) and NAP of Korea Research Council of Fundamental Science & Technology.

References