CodeMP: Network Coded Multipath to Support TCP in Disruptive MANETs

Chien-Chia Chen¹, Guruprasad Tahasildar¹, Yu-Ting Yu¹, Joon-Sang Park², Mario Gerla¹, M. Y. Sanadidi¹

¹Computer Science Dept., UCLA, Los Angeles, CA 90095, USA
chenchienchia@ucla.edu, {get, yutingyu}@cs.ucla.edu, jsp@hongik.edu, {gerla, medy}@cs.ucla.edu

²Dept. of Computer Engineering, Hongik University, Seoul, Korea

Abstract—TCP over Mobile Ad-hoc Networks (MANETs) is challenging due to frequent route breaks, high random errors, and DATA-ACK interference. Network coded multipath approaches have been shown in several previous studies to be an effective transmission paradigms in disruptive networks. However, most of the previous studies either have no adaptive redundancy control or rely on theoretical models that require knowledge of all relays of the entire network state. In this paper, we propose a network coded multipath scheme for conventional TCP—CodeMP—that adapts to frequent link changes in MANET and requires no explicit control messages. The scheme exploits multipath-redundancy and maintains total transparency to transport layer protocols. The proposed coding scheme is based on three components: (1) random linear coding scheme with adjustable redundancy, (2) multipath routing, and (3) ACK Piggy coding. Simulation results show that in a 3-hop static scenario, the proposed multipath scheme improves TCP flows goodput by 70% compared to our previous single-path scheme. In an extreme MANET scenario where two TCP sessions co-exists and nodes are moving as fast as 25 m/s with up to 40% packet error rate (an environment in which regular TCP collapses completely), CodeMP achieves at least 700Kbps aggregate TCP goodput, with a Jain’s fairness index of 0.99.

Keywords—Adaptive Network Coding, TCP, MANET, Fairness, Random Linear Coding, XOR Coding, Wireless Multipath, Lossy Channels

I. INTRODUCTION

The Transport Control Protocol (TCP) is the most commonly used reliable transport protocol in the Internet. The objective of TCP is to provide not only end-to-end reliable transmission but also fair congestion control for better sharing of network resources. TCP-NewReno, one of the widely used TCP variants, employs a loss-based congestion control algorithm that assumes all packet losses are due to congestion and thus are the result of router buffer overflow. This assumption works in a low random error rate, mobility free, and point-to-point network, the context in which TCP-NewReno was originally designed. However, in a disruptive mobile ad-hoc network (MANET) environment, TCP encounters new challenges such as frequent route breaks due to mobility, time-varying random error due to unpredictable link quality and random interference, and in tactical scenarios, jamming. In addition, because of the broadcast nature of wireless media, TCP DATA and its ACKs contend for the shared wireless channel resulting in self-interference.

A variety of TCP modifications over loss-based congestion control in wireless networks have been proposed, e.g., Loss Discrimination Algorithms (LDA) [1] and TCP parameter tuning [2]. Recent research has been devoted largely to network coding based solutions, which have been successfully applied to a number of network environments [3]. In [4] and [5], R. Koetter and P. A. Chou et al. propose random linear coding (RLC) as a practical code implementation. Numerous following studies have shown that in multicast or multi-flow scenarios, network coding helps mitigate wireless interference by reducing the number of transmission [6-7]. In lossy environments, network coding also improves robustness [8-10].

Although the benefits of coding include throughput improvement, every solution so far only partially addresses the challenges of robust TCP communication in MANETs. PiggyCode [11] and its variants [12-13] utilize inter-flow coding, as in COPE [6], to mitigate TCP self-interference through the opportunistic XOR of DATA and ACK packets within a TCP session. However, these schemes do not claim to be suitable for lossy scenarios.

To address problems in lossy networks, one of the earliest proposals is based on an intra-flow RLC scheme [14], where 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 TCP sender and TCP receiver. Following [14], several approaches have been proposed to address both self-interference and random errors issues, such as I2NC [15] and ComboCoding [16]. Nevertheless, all the above schemes assume a pre-determined single-path route given to the coding module, which is impractical in disruptive MANET scenarios.

In order to overcome the problem of frequent route breaks and highly unstable links in disruptive MANETs, multipath routing is inevitable. In [7], S. Chachulski et al. propose a multipath coded routing scheme, called MORE. Subsequent to MORE, many extensions have been proposed [10, 17-18] to further improve its performance. Although these studies have shown the effectiveness of multipath coded routing, most of them rely on explicit control messages and/or explicit ACKs at the network layer. The dependence on explicit control messages can lead to excessive overhead in MANET environments where most of the network parameters, such as number of nodes, network topology, number of flows, buffer length at all nodes, and buffer occupancy by every flow, are changing at all times. Furthermore, all these approaches do not claim to be compatible with and transparent to the TCP layer.

This research is supported through participation in the Global Environment for Network Innovations (GENI) Project sponsored by the National Science Foundation under Grant 1797-ARRA, and by Singapore DSO National Labs under Grant DSCCO10350. J.-S. Park was supported by National Research Foundation of Korea Grant funded by the Korean Government (2010-0005334, 2010-0027410).
In this paper, we address the above problems with conventional TCP running on an adaptive network coded multipath scheme. While totally transparent to TCP, the combination of conventional TCP over multipath, which we refer to as CodeMP-TCP, provides a robust solution for TCP in disruptive MANET environments by mitigating random losses via intra-flow coding, and reducing interference through encoding the ACK and DATA packets together. Moreover, without explicit control messages, CodeMP adjusts coding redundancy and multiple-path redundancy adaptively to further deal with route breaks and unstable wireless link quality in MANET scenarios.

The contributions of the proposed adaptive multipath coding scheme are as follows. (1) It provides a practical and efficient approach to exploit multiple-path redundancy in disruptive MANETs. (2) It adapts the redundancy to network changes using a heuristic-based link estimate algorithm. (3) The adaptation algorithm requires no explicit control messages. (4) The design and implementation are transparent to transport layer protocols such as TCP. (5) The proposed coding scheme achieves better fairness in the presence of multiple coexisting TCP sessions in the MANET scenarios.

Note that in order to deploy CodeMP, proper network layer modifications 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 MANET design [19].

The rest of the paper is organized as follows. The literature on multipath network coding is reviewed in Section II. The design and implementation of CodeMP are presented in Section III. Section IV evaluates the performance of our multipath coding scheme in single and multiple session settings. Section V concludes the paper.

II. RELATED WORK

Recent related work on network coding strategies for lossy wireless networks mainly falls in two categories: multipath coded routing and single-path coded routing (generally with TCP). Numerous research studies have shown that Network Coding has helped improve the throughput or robustness or both. We review the most popular schemes below. Anticipating the conclusions, we report that no previous study has addressed the problem of TCP performance in lossy, mobile MANETs using multiple paths (for redundancy) and no previous study has evaluated fairness among competing flows.

Multipath Coded Routing

J.-S. Park et al. proposed and implemented CodeCast, one of the first Network Coding schemes designed for a disruptive, ODMRP-mesh based, multicast environment [8]. CodeCast was successfully demonstrated in lossy environments particularly for multicast streaming applications. S. Oh et al. later extended CodeCast to unicast UDP traffic [20], where relays can dynamically switch between random linear coding or plain multipath routing based on network loss conditions. CodeCast and its successor work [20], however, assume fixed redundancy and are thus not very suitable for disruptive MANET environments.

MORE, proposed by S. Chachulski et al., like CodeCast, uses multipath coded routing. It adapts coding redundancy to changing link loss rates [7]. In MORE, relays opportunistically form multiple paths on which packets are re-encoded and forwarded. Based on ETX, relays adaptively adjust their redundancy. Relays also rely on explicit network layer ACKs to determine when a generation is completed so that a new generation can be started. As a result of this “stop-and-wait” design, MORE induces potentially high delay and delay jitter that is not suitable for some TCP applications. Also, MORE uses explicit control messages that tend to cause excessive overhead in disruptive MANETs. In [17], Y. Lin et al. point out that the design of MORE does not fully utilize the available capacity. They introduce an enhanced, window-based flow control algorithm, CodeOR. CodeOR improves the performance of MORE, but it is still not designed for MANETs and still suffers from a significant amount of explicit control message overhead.

Along a similar path, but with a more theoretical emphasis, X. Zhang et al. propose an extension of MORE called Dice [18]. Dice considers the sharing of network resources among two or more UDP sessions, and using a game theory framework, optimizes resource allocation for better efficiency-fairness tradeoffs. The environment is static and loss free and thus the scheme is hardly applicable to MANET scenarios. Another MORE variant is later proposed by the same group in [10]. Different mathematical optimization techniques are used to address the resource sharing problem, this time with random errors. Both work in [10] and [18] provide a good theoretical study of MORE properties using optimization-based approaches. The schemes, however, offer little insight into routing and flow control in a constantly changing MANET. In addition, as in MORE, the dependence on explicit control messages remains a major problem.

In the family of network coding solutions, one must also mention the inter-flow coding solutions, like COPE [6], which have the goal of opportunistically saving spectrum in multi-flow scenarios, generally trading bandwidth for robustness. These schemes do not claim to be suitable for disruptive MANET operations.

Single-Path Coded TCP

While the above multipath proposals address mainly robust unicast and multicast applications without TCP, there have been several single path proposals that are directly targeting TCP. The reason for preferring single paths for TCP is quite obvious—the avoidance of out-of-order packets due to multiple paths. Yet, the single path configuration renders TCP performance precarious in presence of mobility and high errors. In [14], J. K. Sundararajan et al. propose one of the pioneer schemes for single-path coded TCP over lossy links. They show that random linear coding improves TCP performance in lossy scenarios after clever modifications to TCP acknowledgement semantics are made. The work in [14] assumes a fixed Network Coding redundancy. H. Seferoglu et al. extend the above work in I2NC, where, using Network Utility Maximization (NUM) techniques, the network coded TCP protocol design is formulated as an optimization problem [15]. I2NC also employs inter-flow coding, in addition to intra-flow coding, to mitigate interference. Nevertheless, since I2NC is based on NUM framework, it has the inherent limitation of requiring the fine tuning of several parameters, such as number of nodes, network topology, number of flows, buffer size at all nodes and buffer occupancy by every flow. As a consequence, re-optimization
must be triggered whenever a parameter changes value. Similar to the approach presented in [15], C.-C. Chen et al. propose a practical network coding design, ComboCoding, which uses both intra- and inter-flow coding and adapts the redundancy to the estimated loss rates [16]. ComboCoding further eliminates all explicit control messages and relies completely on overhearing. As we shall see, this combination strengthens the protocol and improves its fairness in multi-flow disruptive scenarios.

Most of the above single-path coded TCP studies, however, assume static scenarios with a pre-determined single-path route from the TCP source to the TCP destination. This is totally impractical in MANET environments. Besides, although each of the above schemes supports multiple concurrent sessions and works in lossy wireless networks, none has been tested in a disruptive MANET where mobility and time-varying jamming and random errors are the norm. Given these premises, it appears that no study has addressed the situation where TCP must run on a multiple paths (for redundancy); moreover no study has reported on the ability for two or more TCP flows to share resources fairly in a multipath, network coded scenario. In this paper, we approach this problem leveraging our prior experience with TCP on single path. Our aim is to leverage single-path coded TCP and the multipath coded routing results to design a practical scheme for TCP traffic in disruptive MANETs.

### III. CODEMP DESIGN AND IMPLEMENTATION

CODEMP consists of three main elements: random linear coding with adaptive redundancy, adaptive multipath routing, and ACK Piggy coding. The first two components interact closely with each other to provide efficient and reliable communications. The ACK Piggy coding is used to mitigate TCP DATA-ACK interference.

#### A. Intra-Flow Random Linear Coding

The intra-flow coding scheme is based on our previously published Pipeline Coding [16, 21] and can be described as follows. Consider a sequence of equal-sized packets $p_1$, $p_2$, $p_3$, ..., that are generated by application and let $k$ denote the number of packets in a coding generation. A coded packet $c_i$ in the $i^{th}$ generation is constructed as follows:

$$c_i = \sum_{j=1}^{m} e_{ij} p_{i+k+j}$$

where $m$ is the number of data packets currently in the generation buffer, $e_{ij}$ is a random weight for a Galois field $\mathbb{F}_q^k$, and $i \times k$ is the total number of packets transmitted before the $i^{th}$ generation. $\{e_{ij}\}$ is the encoding vector carried in the packet header. 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 $\mathbb{F}_q$, so that data packets $p_i$ and coded packets $c_i$ are also regarded as vectors over $\mathbb{F}_q$.

Upon arrival of a data segment, the flow source generates a coded packet based on Eq. (1). 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 row by row by extracting the encoding vector from each packet:

$$
\begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\begin{bmatrix}
e^{(1)}_1 \\
e^{(2)}_1 \\
e^{(3)}_1 \\
d_1 \\
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\begin{bmatrix}
p_1 \\
p_2 \\
p_3 \\
p_4 \\
\end{bmatrix}
$$

The above linear equation system can be solved incrementally without waiting for a full-rank generation. For example, upon receiving $c_1$, destinations can decode $p_1$, and so on. Intermediate nodes also participate in re-encoding, as explained in [22]. Besides, relay nodes also have a forwarding redundancy to determine the number of re-encoded packets to generate when an innovative packet arrives. Readers are referred to our previous publications for Pipeline Coding details [16, 21, 23].

#### B. Adaptive Redundancy Control and Multipath Routing

The redundancy and multipath adaptation algorithm consists of four parts: slice assignment, forwarding control, cross-slice loss estimates, and intra-slice redundancy distribution. The main idea of our adaptive control algorithm is to first group together the nodes within the same distance from the flow source. In our design, such a group is called a “virtual slice” or a “slice.” We use the number of hops from the flow source as our distance metric. Once nodes assign themselves a slice number, they then start learning and estimating the network conditions and adjust the redundancy accordingly. If a node notices that it delivers only a few packets to the adjacent downstream slice or it is not within a reasonable distance to the destination, the node suppresses all of its packet forwarding until the network conditions change. Note that all the control and adaptation are done per “flow,” so each flow will form their own multipath forwarding set with different redundancy at each node. In our design, TCP DATA flow and ACK flow within the same session are treated as two separate flows.

### Slice Assignment and Forwarding Control

To determine the number of hops a node is away from the flow source, an extra 1-byte field called “hop-count” is added to our coded packet header. It is initialized to zero by the flow source and incremented at each relay. Note that all these procedures are applied only to “innovative packets,” to avoid loops. Fig. 1 below presents an example of a snapshot of a sliced network topology.

![Slice Example](image)

Since topology changes dynamically, each relay maintains a “hop-count” window of 10 samples for each flow. Every time the sampling window is filled, a relay resets its hop-count to the minimum number of hop-count number in the current sampling window. A similar approach is also used so that relays could learn the current shortest path length from the flow source to flow destination. Based on the hop-count and the shortest path length, a relay could determine itself a non-helpful node and suppress all of its forwarding until it is within a reasonable range to the flow destination.
Cross-Slice Loss and Redundancy Estimate

Based on the above slice assignment algorithm, relays then actively estimate the loss rate between their own slice (slice\(^i\)) and the adjacent downstream slice (slice\(^{i+1}\)), where the superscript \(i\) denotes the hop-count calculated as previously described. Each relay \(j\) annotates in the packet header the total number of packets it has sent in the current generation, which is denoted by \(S_j\). The relays in the downstream slice then stamp in the header \(N_{\text{sent}}^{i+1}\), which is the sum of all \(S_j\). Also, another field, \(N_{\text{recv}}^{i+1}\), is added to the header to denote the total number of packets a particular node in slice\(^{i+1}\) has received. Both \(N_{\text{sent}}^i\) and \(N_{\text{recv}}^i\) will be overheard by adjacent upstream relays. These additional counters are all of two bytes length and thus to avoid overflow, they are reset when every new generation arrives. Fig. 2 below shows an example of the above procedure where node 1, 2, and 3 are the upstream nodes in slice\(^i\) and we assume only one downstream node in slice\(^{i+1}\) for simplification.

![Fig. 2 Piggybacked Counters](Image)

**Fig. 2 Piggybacked Counters**

Based on the overhead counters \(N_{\text{sent}}^i\) and \(N_{\text{recv}}^{i+1}\) from downstream nodes, a relay in the slice\(^i\) then computes an instantaneous loss rate from slice\(^i\) to slice\(^{i+1}\) as follows:

\[
P_{\text{st}} = 1 - \frac{N_{\text{sent}}^{i+1}}{N_{\text{recv}}^i}.
\]

The relay then takes an average for every ten \(P_{\text{st}}\) samples, which is denoted by \(\overline{P}_{\text{st}}\). 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:

\[
\hat{P} = \hat{P} + \alpha \times \left( \overline{P}_{\text{st}} - \hat{P} \right),
\]

where \(\alpha\) is a proper smoothing factor (set to 1/6 in our simulation experiments). The cross-slice redundancy from slice\(^i\) to slice\(^{i+1}\) is then set as follows:

\[
R = K + \frac{1}{1 - \hat{P}},
\]

where \(K\) is the base redundancy (set to 0.8 in our experiments). The value 0.8 is chosen based on our simulation studies in which it introduces the minimum additional redundancy to avoid under-estimate.

### Intra-Slice Redundancy Distribution

The redundancy distribution algorithm is based on the heuristic that if a relay learns that its neighboring nodes are making good progress, it should reduce its share to avoid overstressing the network.

The procedure works as follows. Upon receiving an innovative packet from upstream node \(X\), a relay will initiate a re-encoding event. Namely, it creates a number of new re-encoded packets that include the ID = \(X\) in the header. When relays in the upstream slice overhear this packet, they increment the count of the trigger node. Upon a new generation arrival, based on the track of triggering nodes in the previous generation, a relay \(j\) then calculates the percentage of packets that are triggered by the relay itself or an unknown node, which is denoted by \(\text{share}_j\). Note that we use an aggressive approach here to consider all re-encoding events triggered by unknown nodes as if triggered by the relay itself. This is because overhearing has a high loss rate in MANETs and in case a relay misses piggybacked feedback, it has to assume no other nodes will compensate for such a loss.

Based on the estimated \(\text{share}_j\), a local redundancy for node \(j\) in slice \(i\) is then updated as:

\[
R_j^i = R \times \text{share}_j.
\]

Again, the local redundancy is per flow, which means relay \(j\) now uses redundancy \(R_j^i\) only for the corresponding flow. With this estimate, if a node learns that all of the triggers are by some other neighbors, it will effectively suppress itself in the next generation.

### Impact of Inaccurate Estimates

As the adaptation is designed for MANET scenarios, inaccurate estimates are inevitable. Our coding scheme tries to avoid under-estimation of loss rates and relies on TCP to recover from over-estimates. In the current design, the base redundancy \(K\) and the aggressive estimate of redundancy share introduce extra redundancy to protect against under-estimate. Over-estimates lead to congestion and more losses, not recoverable by RLC. This triggers the TCP congestion control mechanism, which in turn reduces the sending window. As the TCP sending rate slows down, the contention is alleviated and thus the redundancy falls back to the affordable range. In addition, our adaptation treats congestion/collision losses and random errors identically. Thus, flows might all encounter more losses when congestion happens, which eventually exceeds the working range of redundancy and triggers TCP congestion window reduction, similar to over-estimates. It is possible to further shorten this feedback cycle by introducing loss discrimination algorithms to CodeMP.

### C. Ack Piggy Coding

To alleviate TCP DATA-ACK self-interference, we adopt a network layer XOR-based coding that is extended from our previous work [16]. The idea of such an XOR-based inter-flow coding is similar to COPE [6], but with modifications to address special types of bi-directional flows—TCP DATA and ACKs.

Similar to our previous work [16], the Ack Piggy coding module employs 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 Piggy 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 broadcasted. 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 broadcast. If DATA and ACK packets are not of the same length, padding zeros are appended to the shorter packet.

In addition to our previous design, there are still changes needed to accommodate multipath scenarios. In the single path scenario, it has been shown by L. Scalia et al. that XORing
In the proposed scheme above, we have simplified our system design for this early study of multipath coding in MANETs. We are aware of a number of design enhancements, which we discuss briefly here, and which we intend to investigate in future work. First, our current design XOR an incoming ACK with only the oldest DATA packet in the buffer. It is possible to further improve the multipath redundancy for ACK flows by allowing ACKs to be XOR’ed with all or with a number of the buffered DATA packets. For example, in Fig. 3, if by the time node 4 receives A1, it buffers not only C1 from node 2 but also C1’ from node 1 and C1゛ from node 3, node 4 can instead generate 3 different XOR’ed packets C1 ⊕ A1, C1’ ⊕ A1, and C1゛ ⊕ A1. Such a modification would allow A1 to be potentially decoded and propagated by nodes 1, 2, and 3.

In addition, our design buffers TCP DATA and waits for TCP ACK, which so far is an engineering decision since we found this works better in our simulations. It is also possible that one could buffer TCP DATA and wait for TCP DATA, although it might need additional tuning. Further, we use a fixed timer of 5 ms in all our simulations. We have found from our simulations that 5ms is not always the best timer for all cases. More analysis and experiments are required to fine tune the Piggy ACK strategy.

D. Impact of Packet Reordering

One of the major challenges for TCP over multiple concurrent paths is to packet reordering. Simulation shows that in a naïve multipath broadcast scheme without network coding, data packets frequently arrive out-of-order. Out-of-order deliveries lead to dup-acks, which are treated as congestion indications and thus cause the sender to unnecessarily reduce its congestion window. In CodeMP, the out of sequencing due to multiple paths is prevented by Network Coding. In fact, the NC middleware at the destination reassembles packets within each generation and delivers them in sequence to the destination. Duplicate ACKs can be created by multiple paths on their way back to sender. However, ACKs are stamped with the generation number. If they are recognized by the source to belong to the same generation, they are dropped. In CodeMP, however, there is a possibility that a new generation arrives before the proceeding generations are not fully decoded, in which case a duplicate ACK is generated and it is properly interpreted as a congestion indicator (as it belongs to a new generation). Recall that the window increases linearly in CodeMP, spanning multiple generations. In case of heavy congestion, generations fail to be timely assembled. Then, a timeout causes a window reduction, and so does a triple dup-ACK. Our simulation traces show that dup-ACKs are extremely rare that one can say that practically network coding resolves the TCP sequencing problem over multiple paths. The larger the generation size, the rarer the out-of-ordering but the higher the coding overhead and the end to end delay. From simulation, we have found experimentally that a generation size of 8 is the best compromise in most situations.

IV. SIMULATION RESULTS

CodeMP was tested on QualNet 4.5. We evaluate our proposed coding scheme in both static and MANET scenarios. For static topologies, we compare the multipath coding scheme with non-coded single-path TCP and a single-path TCP with ComboCoding [16]. For MANET scenarios, we compare the multipath coding scheme with TCP running on a popular MANET routing protocol, OLSR [24]. In both static and MANET scenarios, we also compare our scheme with a multipath broadcasting approach, in which all relays forward a received packet once; a “unit generation size” random linear coding scheme is used to eliminate packets that have been forwarded by the same relay.

In all simulations, unless otherwise specified, nodes are 150 meters apart, and the Physical and MAC layer protocols are standard 802.11g, with RTS/CTS disabled, even in the MAC unicast mode. The channel bit-rate is 54Mbps. In all multipath runs, nodes communicate using pure-broadcast as we rely on multipath opportunistic forwarding. For baseline single-path simulations, nodes communicate using pseudo-broadcast as in
[6]. The transport layer protocol is TCP-NewReno and the application traffic is a Generic-FTP, which backlogs the TCP port with continuously generated 1500-byte packets. The generation size of the random linear coding (intra-flow coding) module is set to 8 packets in all simulations.

The ACK Piggy coding timer is set to 5ms, i.e., TCP DATA packets will be buffered in the Ack Piggy coding module for up to 5ms. In all simulation sets, variable link loss rates of up to 40% are introduced in order to simulate a challenged environment subject to random interference and jamming. Also, unless otherwise noted, all simulations last for 110 seconds and the FTP application starts at 20 second.

A. Static Scenarios with Time-Varying Jamming

In the first set of simulations, we perform a series of static topology runs to validate the effectiveness of the proposed coding scheme and to collect baseline measurements. In this set of simulations, all single path topology assumes known routes and thus uses pseudo-broadcast; all multipath (grid) topology assumes no topology knowledge and uses pure-broadcast.

Single-Session Scenarios

We first perform several single TCP session simulations over a 3-hop topology as shown in Fig. 4. Two configurations are used for the single path topology: no coding and ComboCoding. Three configurations are used for the multipath (grid) topology: multipath broadcasting, multipath coding, and multipath coding with ACK Piggy coding. The multipath broadcasting is done by setting the generation size to 1 with a fixed redundancy of 1.00 at every node.

Fig. 5 above plots the instantaneous TCP goodput over time. The application starts sending packets at time 20 seconds, and the packet error rate (PER) for all links is 0%, 40%, and 20% during the time intervals of 20–50 seconds, 50–80 seconds, and 80–110 seconds respectively. The first observation is that with zero errors, single path without coding performs the best. In a perfect scenario with given route and no random errors it is well known that coding is ineffective. This is due to the fact that CodeMP is designed for mobility and random errors, it cannot dynamically adapt to perfect link conditions. A future extension of this work will address the ability to monitor link loss and disable coding, i.e., setting generation size = 1 and redundancy = 0, when the loss estimate drops below a threshold, say 5%.

We next find that as random losses occur, TCP without coding fails to work. Also, single path TCP + ComboCoding works better than multipath TCP + NC when links have no random errors, but with 40% random errors, multipath NC outperforms single path ComboCoding since it exploits multipath redundancy. For multipath runs, we found that multipath NC with ACK Piggy coding achieves 10% to 30% higher goodput than without Ack Piggy coding. Besides, TCP + multipath broadcast performs the worst, which is mainly due to the out-of-order data packets.

Fig. 6 shows a typical snapshot of the received TCP data sequence number and the corresponding timestamp for three selected schemes: 1-path ComboCoding, CodeMP (without ACK Piggy coding), and multipath broadcast. Note that since each scheme has different progress, to show three lines in a single snapshot, we use different offsets \( Y_0 \) for each of them as noted in the legend; in other words the real sequence number should be \( Y + Y_0 \). Fig. 6, each data point represents a TCP sequence number received by the TCP destination along with its corresponding receiving timestamp. If all data packets arrive in order, it should show a monotonically increasing line. Each drop in Fig. 6 implies an out-of-order data deliver, which will then trigger a dup-ACK. We find that for both 1-Path ComboCoding and CodeMP, sequence numbers grow stably over time. In contrast, in multipath broadcast, they often arrive out of order. This is because the single-path ComboCoding has virtually no chance to create out-of-order packets, and CodeMP, mitigates reordering by virtue of network coding generations. Although we still see from our packet trace that CodeMP has out-of-order data packets, it happens rarely and does not cause noticeable impact. Also note that Fig. 6 omits single-path no coding and CodeMP with Ack Piggy coding for a cleaner presentation. However, the single-path no coding demonstrates a similar pattern as single-path ComboCoding. Moreover, both CodeMP with and without Ack Piggy coding show a similar trend. Hence, we conclude that it is packet reordering that causes the dramatic goodput degradation of multipath broadcast.
Fig. 7 below shows the normalized transmission overhead for each scheme. The normalized transmission overhead is defined as:

\[ \sum_{i} \frac{S_{txi}}{D \times N} \]  

(7)

where \( N \) is the total number of nodes, \( D \) is the total number of DATA packets received by the TCP destination, and \( S_{txi} \) is the number of MAC frames physically transmitted by node \( i \). The number of MAC frames transmitted is obtained from the simulation statistics reported by QualNet. Eq. (7) can be interpreted as the average number of MAC frames needed per node per successful TCP DATA packet delivery.

We find that the overhead in single path experiments is higher than in multipath ones. This is because for single path, unicast MAC is used, which produces relatively inefficient link-layer retransmission. In the multipath cases, broadcast MAC is used. Redundant RLC packets compensate for losses, causing less overhead than MAC retransmission. We also found that in this static single session simulation, ACK Piggy coding does not significantly reduce the overhead since the TCP DATA-ACK self-interference does not present as a major problem in this simple multipath configuration.

2-Session Scenarios

The next set of experiments reconfigures the topologies to accommodate two TCP sessions as shown in Fig. 8.

Table 1 summarizes the TCP goodput per session for each setting in each loss rate period. Note that we also consider the simple single-path adaptive NC without Ack Piggy coding in this run, which is literally ComboCoding with zero Ack Piggy buffer timer. The results show similar trend except that the single path ComboCoding works the best. We also notice that Ack Piggy coding helps more in single path than in multipath.

We focus on two important performance measures of the 2-session experiments: aggregate throughput and fairness. In terms of aggregate throughput, we note that (for the Piggy ACK versions—both single- and multi-path) the aggregate throughput in the 2-session case is only marginally lower (by at most 10%) than the single session case. Moreover, the two sessions share the resources fairly. In contrast, the no coding versions exhibit extreme unfairness, especially for 20% loss rate, where complete capture by one flow is observed. This brings up another important benefit of Network Coding especially when combined with Piggy ACKs, namely the ability to maintain fair sharing among TCP sessions, even in extreme loss situations.

Table 1 2-Session Average TCP Goodput (Kbps)

<table>
<thead>
<tr>
<th>Config</th>
<th>0% PER</th>
<th>40% PER</th>
<th>20% PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Path No Coding</td>
<td>1087.90</td>
<td>65.12</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>2113.12</td>
<td>63.08</td>
<td>159.74</td>
</tr>
<tr>
<td>1-Path NC</td>
<td>749.57</td>
<td>226.10</td>
<td>473.91</td>
</tr>
<tr>
<td></td>
<td>756.53</td>
<td>324.40</td>
<td>504.22</td>
</tr>
<tr>
<td>1-Path ComboCoding</td>
<td>1196.03</td>
<td>283.85</td>
<td>736.05</td>
</tr>
<tr>
<td></td>
<td>1375.03</td>
<td>410.42</td>
<td>661.50</td>
</tr>
<tr>
<td>Multipath NC</td>
<td>1163.67</td>
<td>553.37</td>
<td>864.26</td>
</tr>
<tr>
<td></td>
<td>647.58</td>
<td>537.40</td>
<td>660.28</td>
</tr>
<tr>
<td>Multipath NC + Piggy Coding</td>
<td>932.25</td>
<td>371.92</td>
<td>780.70</td>
</tr>
<tr>
<td></td>
<td>883.92</td>
<td>577.13</td>
<td>684.44</td>
</tr>
<tr>
<td>Multipath Broadcast over Grid</td>
<td>959.69</td>
<td>131.07</td>
<td>618.50</td>
</tr>
<tr>
<td></td>
<td>700.83</td>
<td>74.96</td>
<td>2.05</td>
</tr>
</tbody>
</table>

Fig. 9 shows the normalized transmission overhead for each setting. Similar to the single session runs, single path schemes have higher overhead than multipath schemes. Under 2 sessions, ACK Piggy coding also helps reduce overhead in the multipath since it reduces the number of transmissions and hence mitigates interference.

B. MANET Scenarios—Single-Session Corridor Mobility

We next evaluate the performance of CodeMP under mobility scenarios. We first consider the corridor model given in Fig. 10. In the corridor model, we configure three groups that are equally located between a static source and a static destination. We place four nodes in each group. Each node moves within its corridor using random ‘way-point’ model with a pause time of 10 seconds. We vary the maximum moving speed for separate runs and set the minimum speed to 10 m/s slower than the maximum speed. Since this is a dynamic topology where routes cannot be pre-configured, a dynamic MANET routing algorithm is required. For the single path runs, we select the state-of-the-art MANET routing protocol, OLSR [24]. We use all the default values specified in RFC 3626 for OLSR control timers. For multipath, we use the multipath broadcast scheme as a naive multipath without redundancy.
Fig. 11 shows the average TCP goodput vs. maximum moving speed graph. As expected, the goodput degrades for all cases as the moving speed increases. OLSR under high mobility degrades very rapidly, as expected. From the packet trace, OLSR actually shows a bimodal behavior. When OLSR can maintain a valid route, it delivers an instantaneous goodput as high as 7Mbps. As soon as the route breaks, OLSR drops to zero goodput and takes several seconds to recover from the route change. As nodes move faster, OLSR needs longer recovery times resulting in lower goodput. Multipath broadcast, likewise, is not as efficient as multipath NC. However, it is still more robust to mobility than OLSR. We next notice that in this structured topology, DATA-ACK self-interference is a major issue and thus ACK Piggy coding improves the performance significantly. With ACK Piggy coding, CodeMP improves goodput by 30% in low mobility and by 100% in high mobility.

Fig. 13 below shows the average TCP goodput vs. maximum moving speed chart. We first notice that in this more mobile scenario, OLSR fails to work when the maximum speed reaches 35 m/s. Even with around 25 m/s maximum speed, OLSR has dropped to below 100 Kbps. The packet trace reveals that OLSR still works in bimodal manner, where instantaneous goodput jumps from 0Mbps to 7Mbps frequently and rapidly, with varying recovery time. We next observe that in this scenario, as nodes move with more freedom, TCP DATA-ACK self-interference is no longer the performance bottleneck and thus ACK Piggy coding does not improve much. From the packet trace, we notice that the main challenge in this case are the frequent route breaks and thus the inability of ACK flows to exploit multipath redundancy limits the improvements introduced by ACK Piggy coding.

Fig. 12 MANET Global Mobility Model

C. MANET Scenarios—Single-Session Global Mobility

We next move to a more global mobility model, as shown in Fig. 12. In this model, 20 nodes move in a random ‘way-point’ model with 10-second pause time. TCP source and destination nodes are fixed as in the corridor mobility. All other nodes move with maximum speed varying between 10 and 35 m/s. Minimum speed is 10 m/s slower than maximum speed as in the corridor setting. We again compare CodeMP with the two non-coded schemes, OLSR and multipath broadcast.

Fig. 13 Single-Session Goodput vs. Max Moving Speed

D. MANET Scenario—2-Session; Global Mobility; Jamming

In the last set of experiments, we run an extreme scenario, in which nodes are moving as in the global mobility model (Fig. 12) with a maximum speed of 25 m/s. We introduce a second TCP session side by side with the first one. As before, a time-varying random error rate is induced by jammers as follows: 0%, 40%, and 20% during the time intervals of 20~50 seconds, 50~80 seconds, and 80~110 seconds respectively. The same set of coding schemes as in the 1-session global mobility experiments are used, namely, OLSR, multipath broadcast, and CodeMP, with and without Piggy ACKs. From these experiments, we plan to determine whether the proposed scheme maintains efficient and robust communications even in this extreme case, while guaranteeing fair share among co-existing sessions with acceptable transmission overhead.

Table 2 below summarizes the average TCP goodput per session for each setting in each loss rate period. We first find that when there is no random error, OLSR successfully delivered 3Mbps for around 25% to 50% of the time resulting a goodput as high as 1.4Mbps. However, OLSR fails to provide reasonable fairness and thus one session continuously occupies more bandwidth. Further, OLSR again fails when loss rates increase and is unable to recover. We next notice that multipath broadcast is neither efficient nor fair. Multipath broadcast also fails under high random error rates. For multipath NC cases, both with and without Ack Piggy coding perform almost the same, while with Ack Piggy coding, it is slightly fairer. Table 3 below summarizes the Jain’s fairness index of each setting in each period [25]. Overall, adaptive multipath coding with Ack Piggy coding is the fairest scheme.
coding helps reduce intra-session interference and greatly different number of co-existing sessions. It consistently provides environment changes (mobility, time-varying jamming) and to without explicit control messages or cross layer optimization. transparent to TCP or other transport layer protocols, and CodeMP scheme is implemented at the network layer, totally situation where nodes are moving as fast as 25 m/s, with varying goodput with a fairness index of 0.99, with merely 38% more transmission overhead. The RLC and multipath routing schemes adapt to (1) random linear coding, (2) multipath routing, and (3) ACK proposed scheme, CodeMP, consists of three main components: TCP sessions in disruptive MANET environments. The Network Coding over Multiple Paths , and test it with multiple multipath cases. Table 4 Normalized Transmission Overhead

<table>
<thead>
<tr>
<th>Config</th>
<th>0% PER</th>
<th>40% PER</th>
<th>20% PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLSR (Single Path)</td>
<td>0.783</td>
<td>0.000</td>
<td>0.500</td>
</tr>
<tr>
<td>Multipath Broadcast</td>
<td>0.998</td>
<td>0.999</td>
<td>0.998</td>
</tr>
<tr>
<td>Multipath NC + Piggy Coding</td>
<td>0.994</td>
<td>0.999</td>
<td>0.999</td>
</tr>
</tbody>
</table>

Table 4 below shows the normalized transmission overhead (as defined by Eq. (7)) for each setting. As shown in Table 4, OLSR has the least overhead among all schemes since it utilizes only one path at a time. As opposed to OLSR, multipath broadcast uses all paths at all times resulting in the highest overhead. In this extreme test with global mobility, random errors, and multiple TCP sessions, CodeMP with ACK Piggy coding achieves much better overall goodput than OLSR, with much better fairness and only 38% more transmission overhead, which is the least overhead of all multipath cases.

Table 4 Normalized Transmission Overhead

<table>
<thead>
<tr>
<th>Config</th>
<th>OLSR (Single Path)</th>
<th>Multipath Broadcast</th>
<th>Multipath NC</th>
<th>CodeMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.53</td>
<td>1.38</td>
<td>1.12</td>
<td>0.73</td>
<td></td>
</tr>
</tbody>
</table>

V. CONCLUSION

In this paper, we present a simple and robust coding scheme, Network Coding over Multiple Paths , and test it with multiple TCP sessions in disruptive MANET environments. The proposed scheme, CodeMP, consists of three main components: (1) random linear coding, (2) multipath routing, and (3) ACK Piggy coding. The RLC and multipath routing schemes adapt to dynamic scenarios and work closely with each other to provide robust and efficient multipath redundancy. The ACK Piggy coding helps reduce intra-session interference and greatly enhances fairness among concurrent sessions. The proposed CodeMP scheme is implemented at the network layer, totally transparent to TCP or other transport layer protocols, and without explicit control messages or cross layer optimization. Simulation results show that the proposed scheme adapts well to environment changes (mobility, time-varying jamming) and to different number of co-existing sessions. It consistently provides efficient, robust, and fair communications. In an extreme situation where nodes are moving as fast as 25 m/s, with varying packet error rate culminating at 40% and two co-existing TCP sessions, CodeMP can still deliver at least 700Kbps aggregate goodput with a fairness index of 0.99, with merely 38% more overhead than OLSR (which delivers zero goodput in presence of errors). This early study of a practical design of multipath network coding has achieved an important goal, mainly to demonstrate that conventional TCP Reno, with the help of Network Coding, can be made to work with acceptable performance even in extreme conditions where most TCP variants have failed.

REFERENCES