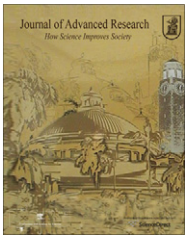
Journal of Advanced Research (2011) 2, 241–252

Cairo University

Journal of Advanced Research



ComboCoding: Combined intra-/inter-flow network coding for TCP over disruptive MANETs

Chien-Chia Chen [a,*], Clifford Chen [b], Soon Y. Oh [a], Joon-Sang Park [c], Mario Gerla [a], M.Y. Sanadidi [a]

[a] *Computer Science Department, UCLA, Los Angeles, CA 90095, USA*
[b] *Carnegie Mellon University, Silicon Valley, NASA Research Park, CA 94035, USA*
[c] *Department of Computer Engineering, Hongik University, Seoul, Republic of Korea*

Received 16 November 2010; revised 1 May 2011; accepted 3 May 2011

KEYWORDS

Network coding;
Random linear coding;
XOR coding;
TCP;
Wireless multihop;
Lossy channels

Abstract TCP over wireless networks is challenging due to random losses and ACK interference. Although network coding schemes have been proposed to improve TCP robustness against extreme random losses, a critical problem still remains of DATA–ACK interference. To address this issue, we use inter-flow coding between DATA and ACK to reduce the number of transmissions among nodes. In addition, we also utilize a "pipeline" random linear coding scheme with adaptive redundancy to overcome high packet loss over unreliable links. The resulting coding scheme, ComboCoding, combines intra-flow and inter-flow coding to provide robust TCP transmission in disruptive wireless networks. The main contributions of our scheme are twofold; the efficient combination of random linear coding and XOR coding on bi-directional streams (DATA and ACK), and the novel redundancy control scheme that adapts to time-varying and space-varying link loss. The adaptive ComboCoding was tested on a variable hop string topology with unstable links and on a multipath MANET with dynamic topology. Simulation results show that TCP with ComboCoding delivers higher throughput than with other coding options in high loss and mobile scenarios, while introducing minimal overhead in normal operation.

* Corresponding author. Tel.: +1 310 825 4367.
E-mail address: chenchienchia@ucla.edu (C.-C. Chen).

2090-1232

Peer review under responsibility of Cairo University.
doi:10.1016/j.jare.2011.05.002

Production and hosting by Elsevier

Introduction and related work

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. Among all TCP variants, the most well-known and most widely-used is TCP-NewReno, which adopts a loss-based congestion control algorithm. TCP-NewReno assumes packet losses are due to router buffer overflow. This assumption

was true in the environment it was originally designed for, the wired Internet, where most links are point-to-point.

However, the assumption that buffer overflow is the only reason behind packet loss no longer holds in wireless multihop networks. In this scenario, a significant amount of loss is due to interference and the unpredictable quality of wireless links. It has been shown that in wireless multihop scenarios, TCP suffers significantly from misinterpreting random errors as congestion. TCP ACK and DATA also contend for the shared wireless medium, which causes self-induced collisions. However, there are several approaches to address these two issues separately, including other forms of congestion control, tuning TCP parameters or protocol optimization.

One method to improve loss-based congestion control in wireless networks is to deploy Loss Discrimination Algorithms (LDA) [1–4], while another is to optimize TCP parameters. Xu et al. [5] pointed out the TCP congestion window to be key in improving TCP performance in wireless networks. Li et al. [6] went further to show that controlling the *maximum* congestion window size is even more effective. Yet another optimization is to reduce the interference between ACKs and DATA packets by reducing the ACK frequency [7]. Intelligently controlling the so-called "Delayed Acks" can reduce ACK packets in the network, thus reducing interference at the inter-flow level.

A recently proposed approach to help TCP in wireless networks is to exploit network coding. Network coding has been proposed in the past for a variety of network environments and traffic scenarios [8]. Koetter and Medard [9] and Chou et al. [10] further enhanced the practical design of coding packets by introducing random linear coding. It has been shown that network coding helps in alleviating wireless interference by reducing the number of transmissions in multicasting or multi-flow environments [11,12]. In addition, network coding also helps in effectively overcoming high loss rate in wireless networks for unicast traffic [13–15].

Nevertheless, coding overhead so far has been a significant drawback as pointed out in publications on the subject. Katti et al. [11] and Huang et al. argue that network coding does not significantly improve TCP. Huang et al. [16] claim that this is mainly because Katti et al. [11] does not take into account bi-directionality, i.e., the fact that TCP DATA flow and TCP ACK flow are naturally in opposite directions. Therefore, they propose to XOR TCP DATA and ACK opportunistically to reduce transmissions.[1] Coding of two flows in opposite directions is referred to as inter-flow coding. A similar idea is proposed by Scalia et al. [17], where the throughput is improved by opportunistically XORing the TCP ACK and DATA flow. Following the work presented in Scalia et al. [17], David et al. [18] propose a MAC layer "dual-cast" support, which further improved TCP throughput by 100%.

One of the earliest proposals to improve TCP in lossy networks is proposed by Sundararajan et al. [19], in which the authors studied an intra-flow random linear coding scheme that was based on *online network coding* [20]. Intra-flow implies that only packets within one flow are coded, which in this case is the data flow. As a result, TCP is significantly improved. However, the scheme presented by Sundararajan et al. [19] re-defines the semantic of TCP ACKs, and thus it requires TCP modifications at both the senders and the receivers.

All the above solutions address either the ACK interference problem or the high data loss problem, but not both. In this paper, we present a hybrid network coding scheme that is (1) transparent to TCP without additional adaptation layers, and (2) addresses both interference and random loss. The proposed coding scheme, ComboCoding, provides a robust, comprehensive solution for TCP in lossy wireless scenarios. ComboCoding combines the idea of inter-flow coding, and a revised version of intra-flow random linear coding [21,22]. The proposed scheme opportunistically encodes the ACK flow and DATA flow together to reduce interference. Moreover, it adjusts packet redundancy dynamically to further improve TCP performance.

The idea of combining intra- and inter-flow coding was first proposed by Qin et al. [23]. The proposed coding scheme is designed for general opposing streams and applies random linear coding to all streams. However, since to find an optimal decodable packet set among unknown number of random linear coded flows is NP-hard [24], the authors eventually implemented an XOR-based inter-flow coding scheme. This scheme has two major limitations when run under TCP traffic. First, as pointed out by Qin et al. [23], their design has an unsolved undecodable problem due to mixing all flows together. In contrast, ComboCoding only mixes together TCP DATA and ACK flows within one hop. One hop decoding was proved by Scalia et al. [17] 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 estimates.

The contribution of ComboCoding is fourfold. (1) ComboCoding combines inter- and intra-flows coding to address both high loss rate and self-induced interference. (2) ComboCoding features a novel loss adaptation algorithm that effectively handles transient, unstable link conditions. (3) ComboCoding is implemented in the network layer and is transparent to TCP and other reliable protocols at the upper layers without extra adaptation layers. This makes ComboCoding forward compatible with any future improvement of upper layer protocols. (4) ComboCoding does not rely on any new or modified MAC layer protocols. This is different from David and Kumar [18], where the authors propose a MAC layer modification to further improve coding gain. We provide a solution that is transparent and requires no hardware or software modifications to the MAC layer.

Note that the network layer in the wireless nodes will require modifications to support ComboCoding. However, modifying the network layer at the node itself is well accepted in extreme situations like emergency recovery and battlefield. This is similar to many special-purpose routing protocols proposed for challenging environments [25]. On the other hand, a transport layer modification has a much more serious impact as it creates problems of selecting a different TCP and upper layer application depending on the underlying network.

The rest of the paper is organized as follows. Section 'Coding scheme design' reviews the fundamentals of the underlying coding schemes in ComboCoding. The design and implementation of our protocol are discussed in Section 'Protocol design'.

[1] Our design does not require distinguishing DATA from ACKs. Rather, we refer "DATA" flow and "ACK" flow to the two flows from the same TCP session but in opposite directions.

Section 'Simulation results' evaluates the performance of ComboCoding and the paper is concluded in Section 'Conclusion'.

Coding scheme design

This section describes the design of both inter-flow and intra-flow coding schemes in ComboCoding. The first subsection elaborates a pipelined random linear coding that is used for the intra-flow coding, and the inter-flow coding scheme we use is presented in the second subsection. Table 1 below provides a summary of terminology we adopt in this paper.

Intra-flow coding

The intra-flow coding scheme we choose is not a conventional batch-based coding but a novel pipelined random linear coding. Our early report [21] has presented a preliminary idea, Pipeline Coding, and demonstrated how it reduces the end-to-end coding delay for UDP applications. Following the idea 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. The coding scheme detail and its interaction with TCP are presented as follows.

Given a sequence of equal-sized packets $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \ldots$ that are generated by an application, let k denote the number of packets in a coding generation. A coded packet $\mathbf{c}$ in the ith generation is defined as:

$$c = \sum_{j=1}^{m} e_j \mathbf{p}_{i\times k+j}, \tag{1}$$

where m is the number of data packets currently in the generation buffer, e_j is randomly selected from a particular Galois field $\mathbb{F}_{2^8}$, and $i \times k$ is the total number of packets transmitted before the ith generation. In this paper, we use lowercase boldface letters to denote vectors, frames, or packets, uppercase letters to denote matrices, and italics to denote variables or fields in the packet header. Every arithmetic operation is over $\mathbb{F}_{2^8}$ so that data packets $\mathbf{p_i}$ and coded packets $\mathbf{c}$ are also regarded as vectors over $\mathbb{F}_{2^8}$. Conceptually, Eq. (1) says that upon receiving a new data packet, the source will instantly trigger the encoding process based on the currently received data packets. Let r denote the source coding redundancy, where $r \geqslant 1$ so that for each generation the source produces $k \times r$ coded packets. If all coded packets are delivered successfully, destinations can construct the following lower triangular matrix without any extra computation:

$$\begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_k \end{bmatrix} = \begin{bmatrix} e_1^{(1)} & 0 & \cdots & 0 \\ e_1^{(2)} & e_2^{(2)} & & \vdots \\ \vdots & & \ddots & 0 \\ e_1^{(k)} & e_2^{(k)} & \cdots & e_k^{(k)} \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_k \end{bmatrix} \tag{2}$$

The above linear Eq. (2) can be solved progressively without waiting for generation completion. For example, upon receiving $\mathbf{c_1}$, destinations can decode $\mathbf{p_1}$, and so on. Relays also participate in reencoding, as explained by Ho et al. [26], in order to minimize the information loss per hop. In addition, relays also have a forwarding redundancy to determine how many reencoded packets to transmit upon receiving each innovative packet. A feedback-based redundancy control algorithm for the intra-flow coding is introduced and will be discussed in Section 'Protocol design'.

Fig. 1 shows an example of the pipelined random linear coding, with generation size $k = 4$ and source coding redundancy $r = 1.25$. Packet re-encoding part is not shown 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 is a significant difference from batch-based coding, which either delivers an entire generation to the upper layer or discards the whole generation. For example, assuming that $\mathbf{c_{10}}$ 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 by Sundararajan et al. [19]. Their approach encodes all packets in the congestion window and redefines the semantic of TCP acknowledgments, while our proposed scheme uses its own coding generation buffer that has no relationship with the TCP congestion window. However, as our intra-flow coding module does not provide reliable transmission, TCP

Table 1 Definitions of terms used in this paper.

Term	Definition
Network Coding	Source-side and relay coding
Batch Coding	Generation based coding scheme. Coding and decoding begins only when the generation rank is "full"
Pipeline Coding	Coding scheme that incrementally encodes and decodes once a new packet arrives
Generation	A set of packets that are encoded or decoded together
Coding vector (encoding vector)	A vector of coefficients that reflect the linear combination of data packets
Rank (degree of freedom)	Number of linearly independent packets in the generation
Innovative packet	A packet that increases the rank of a generation
Coding redundancy	Number of coded packets sent per generation divided by generation size
Delay	The time difference between packet reception by destination application and generation at source application

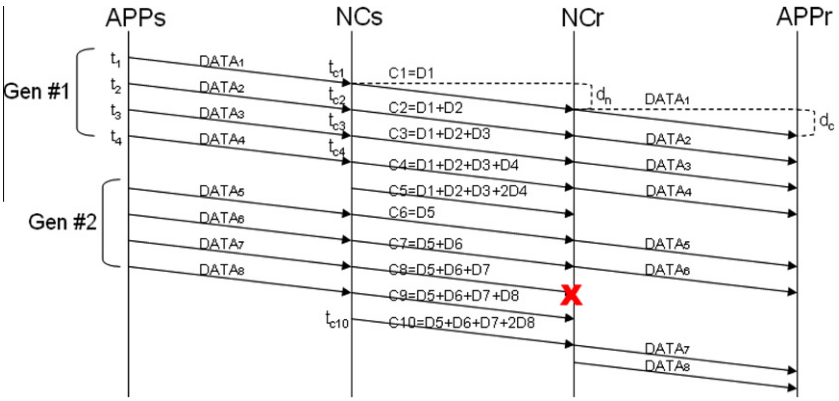


Fig. 1 Pipelined random linear coding example.

still needs to retransmit lost packets in the presence of a loss or timeout event.

Inter-flow coding

Our inter-flow coding design is similar to COPE [11], but it is re-designed specifically here for a special type of bi-directional traffic-TCP. An early study by Scalia et al. [17] proposed a similar idea, PiggyCode, while the inter-flow coding used in ComboCoding is greatly improved to function at the network layer. The concept of the original PiggyCode will be given in this section and the network layer implementation will be elaborated in the Section 'Protocol design'.

The design of PiggyCode is based on the fact that in wireless multihop scenarios, the TCP DATA flow and ACK flow may create interference with each other, which decreases throughput. The main goal of the original PiggyCode is to improve TCP performance by opportunistically XORing TCP DATA and ACK packets at intermediate nodes as shown in Fig. 2. Upon receiving a TCP ACK, the intermediate node checks its MAC layer buffer for a TCP DATA packet. If such a packet exists, the MAC layer performs an XOR of both packets with additional appropriate identification and transmits this newly created "inter-flow coded" packet; otherwise, the ACK is sent out without encoding. To accommodate this process, all packets are buffered before being sent out. Upon receiving an inter-flow coded packet, both receivers perform another XOR operation with the buffered packets to decode the original packets.

The main advantage of such an inter-flow coding is that it requires no TCP modification. However, as the recent study by David and Kumar [18] 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. Due to the lack of dual-ACK support in 802.11, the authors noticed that PiggyCode throughput gain is limited. In order to improve performance, the authors propose a MAC layer modification to introduce dual-ACK support so that both intended receivers will send a MAC-ACK to the PiggyCode sender. It has been shown in their research that with this special MAC-layer support, PiggyCode can improve TCP throughput by as much as 100%.

Protocol design

As mentioned previously, ComboCoding consists of two different types of network coding; inter-flow coding and intra-flow coding. An inter-flow-coded packet is the result of a bitwise XOR of TCP DATA and TCP ACK, where the TCP ACK is padded to make lengths equal. Note that 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 require distinguishing 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 as to the "ACK packets."

As explained earlier, the main goal of performing an XOR is to deliver both packets in one transmission. The major difference between our improved inter-flow coding implementation and the original PiggyCode design is that we do not rely on a MAC layer buffer. For transparency to the MAC layer and to avoid modifying MAC standards, we introduce a buffer at the network layer that queues DATA packets for a given time T, which we refer as "inter-flow coding timer." If an ACK happens to arrive when there exists at least one DATA packet in this buffer, the network layer module XORs both packets and sends an inter-flow-coded packet to the lower layer. If no ACK arrives within this time T, DATA packets are forwarded normally, and the buffer is freed.

Fig. 2 PiggyCode example.

For intra-flow coding we choose the improved pipelined coding, because to the best of our knowledge it is the best random linear coding scheme that is compatible and transparent to TCP without additional layering. However, as a tradeoff, pipelined random linear coding requires a little higher redundancy to mitigate losses. A more detailed discussion is given in the 'Simulation results' section.

We implemented the proposed ComboCoding in QualNet 4.5 [27] at the "network layer," i.e., at the same level as a "routing protocol". Conceptually, intra-flow coding is a network layer function as it deals with end to end flows. Inter-flow coding is a MAC layer function as it mixes flows in the same MAC collision domain regardless of ultimate origins/destinations. For QualNet implementation convenience we have implemented both intra- and inter-flow operations at the routing layer. Note that, although we frequently refer to TCP DATA and ACK, in the implementation we check only the packet "directions". For example, if we were dealing with multiple TCP flows at a crosspoint, we could mix TCP DATA packets from the two flows and ACKs from the two flows separately with no change in the code. Therefore, such a "network layer" implementation does not violate the layering principal and could function transparently to upper and lower layers without additional adaptation. The following sections describe the protocol processing at the source, destination, and relays. A loss adaptation algorithm is also given, followed by a brief discussion of the chosen (standard) channel access scheme.

Coding flow charts

Fig. 3 below shows the coding flow chart at the source. A packet from the upper layer is directly forwarded to the intra-flow coding module. This module generates the desired number of mixed redundant packets and delivers them to the lower layer. Intra-flow coded packets are stored in a local buffer so that they can be later used to decode matched inter-flow coded packets. When the source module receives an inter-flow coded packet, it forwards to the destination handler function.

Fig. 4 gives the decoding flow chart at the destination. If the packet is inter-flow coded, it must be decoded using a data packet previously sent and stored in the buffer. If the packet is not found in the buffer, it is dropped. After decoding the inter-flow coded packet, ComboCoding first examines whether the received packet is innovative, which is determined by examining the packet's coding coefficient vector. If the packet has a linearly independent coding coefficient vector, then it is linearly independent of all received coded packets in the same generation. ComboCoding will then store and decode the packets and deliver them to the upper layer.

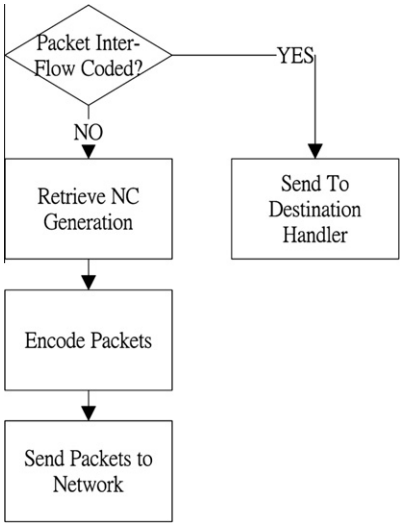


Fig. 3 Coding flow chart at source.

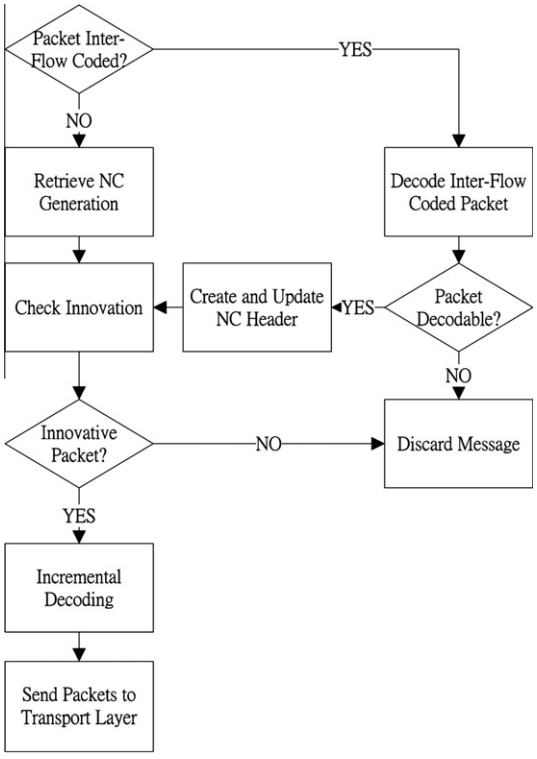


Fig. 4 Coding flow chart at destination.

Fig. 5 shows the coding flow chart for relay nodes, which has the same inter-flow decoding and intra-flow innovative checking as the destination. An inter-flow coded packet will first be decoded and delivered to the intra-flow coding module. If the received data packet is innovative, it then examines whether the packet is marked as "queued" (up to a timeout T). In our simulations, TCP source always marks DATA as queued and TCP destination always marks ACK as NOT queued. ACKs are not queued at intermediate nodes since TCP needs feedback to be delivered as soon as possible. This implies that if the ACK cannot find a DATA packet to XOR, it will be transmitted as an explicit ACK (no piggybacking). However, we have found in the simulation that to queue up DATA or ACK does not make a significant difference and thus the marking is completely for the purpose of identifying the packet direction.

If the packet is a DATA packet, it will be inserted into the inter-flow coding queue, and associate with a timer of T. If the ComboCoding module does not have any ACK to perform an XOR with the queued DATA before timeout T, the DATA will proceed to the "Reencode Packet" decision block.

If the packet is an ACK, it proceeds to look up the inter-flow coding queue. If any DATA that has not yet been mixed is in the queue, the ACK cancels the timer for that DATA

packet and performs an XOR to generate an inter-flow coded packet, which is then sent out.

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 dynamically control the coding and forwarding redundancy. In wireless multihop communications, the redundancy should ideally be set to $1/(1-p)$, where p is the link loss probability. Therefore, it is crucial to estimate the loss rate for each link in order to adaptively adjust the redundancy. In our experiment, we found that the TCP ACK flow must use the same redundancy as the TCP DATA flow due to the use of symmetric links. Therefore, the following algorithm estimates only the loss rate on the TCP DATA flow. Note that the algorithm is specifically for a single path (string) topology.

To estimate per link loss rate, we first add a field in the header of each coded packet to track the number of coded packets received in the current generation at node i, which is denoted by N_i. The count will be carried in the TCP ACKs as well as in the reencoded TCP DATA packets. Nodes receive reencoded TCP DATA packets by overhearing neighbor nodes, and receive TCP ACKs through unicasting. Assuming node $i+1$ is the downstream neighbor of node i in the TCP DATA flow, the instantaneous link loss rate from node i to node $i+1$ is estimated as follows:

$$P_i^0 = \frac{M_i - N_{i+1}}{M_i}, \tag{3}$$

where M_i is the number of reencoded packets sent from node i to node $i+1$, which is recorded locally at node i.

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

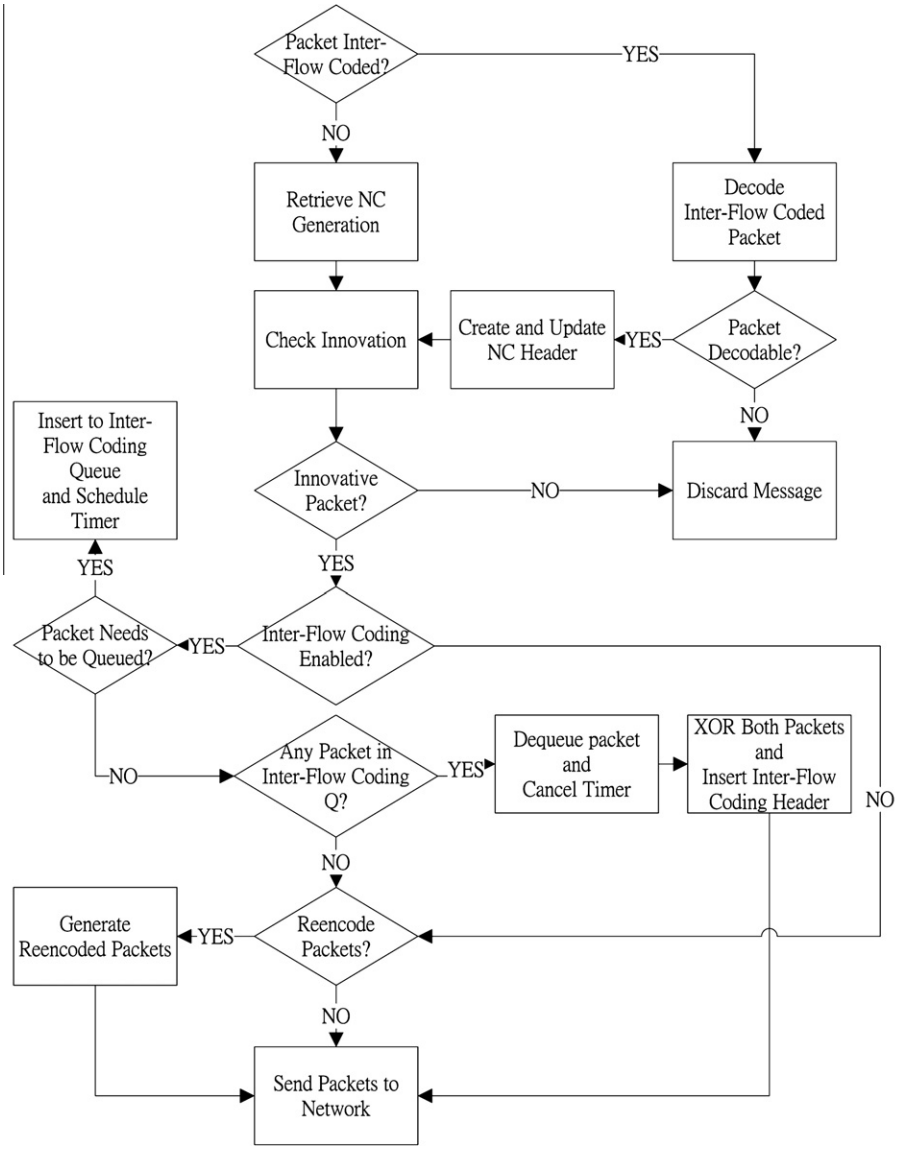


Fig. 5 Coding flow chart at relay.

$$\overline{P_i}' = \overline{P_i} + \alpha \times (P_i^0 - \overline{P_i}), \quad (4)$$

where α 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 thus estimated as follows,

$$R_i = (K_i - 1) + \frac{1}{1 - \overline{P_i}'}, \quad (5)$$

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 future potential packet losses. In our simulation, K_i is set to 1.4.

Note that to the best of our knowledge, our proposed adaptive coding scheme is the first study that addresses the time-varying and space-varying loss as well as the TCP self-interference problem. Previous schemes mostly assume the loss rate is either static or is a given input to the coding module. In addition, in our simulation, we have found that the adaptive redundancy generated by intra-flow coding works more efficiently than the MAC layer retransmission under high loss scenarios. This is because by encoding several packets together, random linear coding does not require the exact loss information since all coded packets are equally important and an innovative coded packet guarantees some new information is received.

Channel access scheme

The choice of channel access scheme is critical in changing wireless network behavior. Since ComboCoding is designed specifically for delivering TCP traffic, it is important to enable MAC-layer retransmissions to mitigate losses. Consequently, Pseudo-Broadcast was chosen in our implementation. Nodes unicast a coded packet to the intended receiver, and all nodes are set in promiscuous mode. Link layer frames will be retransmitted in the case that the intended receiver does not send back a MAC-layer ACK within a default MAC-ACK timeout. As mentioned previously, the original PiggyCode is limited by the lack of dual-ACK support in the MAC layer [18]. Therefore, it is important to choose the intended receiver such that the lack of dual-ACK has minimal impact on TCP. Since TCP ACKs are cumulative and DATA is not, all inter-flow coded packets in ComboCoding are *destined* for the next hop of the DATA flow in the direction of the TCP destination. In a practical (although rare) situation when ACKs are piggybacked on DATA segments, an alternative could be sending the inter-flow coded packets to the next hop that needs a larger packet.

RTS/CTS is also an important issue in configuring a desired channel access scheme. A previous study [28] suggested that RTS/CTS is not effective in ad hoc networks, as it introduces overhead while not entirely helpful in preventing hidden terminals. Similar observations are found by most of the network coding work [11,12,17]. As a result, ComboCoding disables RTS/CTS, and our simulations confirm that this choice provides better performance.

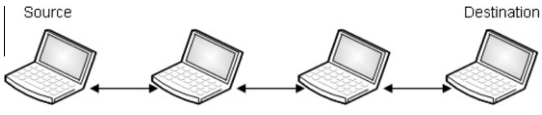


Fig. 6 Simulation topology.

Simulation results

The proposed ComboCoding scheme was tested on QualNet 4.5 [27], where the ComboCoding module is implemented at the network layer. We compare ComboCoding with the original PiggyCode [17] and the unmodified Pipeline Coding [21,22]. The simulation topology is a string as shown in Fig. 6. Nodes are 250 m apart, and the Physical and MAC layer protocols are standard 802.11 g, with RTS/CTS disabled. As discussed in Section 'Protocol design', nodes communicate using pseudo-broadcasting. The transport layer protocol is TCP-NewReno and the application traffic is a Generic-FTP, which will always keep TCP occupied. Table 2 summarizes the configuration and parameters. As mentioned earlier, the high loss rates are meant to simulate a challenged environment subject to random interference and jamming.

ComboCoding evaluation

In this set of simulations the inter-flow timer is set to 4 ms and the generation size of the random linear coding (intra-flow coding) is set to 16. The dynamic redundancy control algorithm is turned off in this set of simulations in order to demonstrate the performance gain of coding without being affected by other factors.

Fig. 7 presents the goodput-to-loss curve 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. For example, the first packet has the highest chance to be included and transmitted in coded packets, but the last packet has only one chance. Due to this property, Pipeline Coding requires a relatively higher redundancy as discussed in our preliminary report [21]. Unlike Pipeline Coding, under 0% loss, TCP ComboCoding still achieves the same goodput as TCP with no coding, which confirms the fact that ComboCoding does not introduce a penalty in normal network conditions.

As the packet error rate increases, the performance of TCP with no coding deteriorates rapidly, and collapses beyond 30% loss. This is because without redundant packet transmission, TCP goodput is inversely proportional to the square root of the packet loss rate as shown by Padhey et al. [29]. Thanks to network coding redundancy, both Pipeline Coding and ComboCoding are more robust to losses. Most importantly,

Table 2 Simulation configuration.

Parameter	Value
Node distance	250 m
Channel bit-rate	54 Mbps
Channel access control	802.11 g (CSMA/CA) RTS/CTS disabled
Transport and application layer	Generic FTP/TCP-NewReno
Per link packet loss rate	0–50%
Packet size	1500 Bytes
Generation size	16 Packets

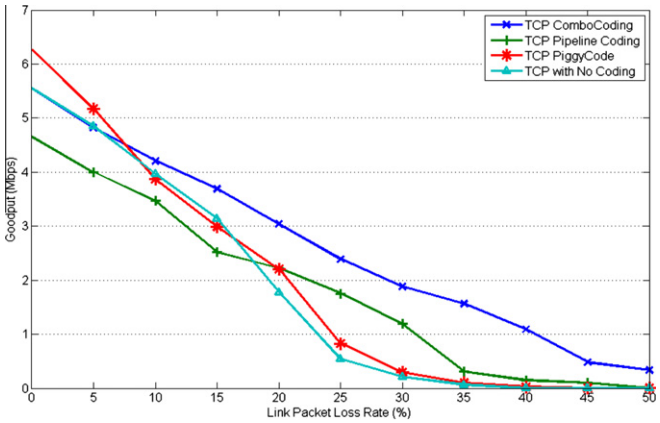


Fig. 7 Goodput-to-loss.

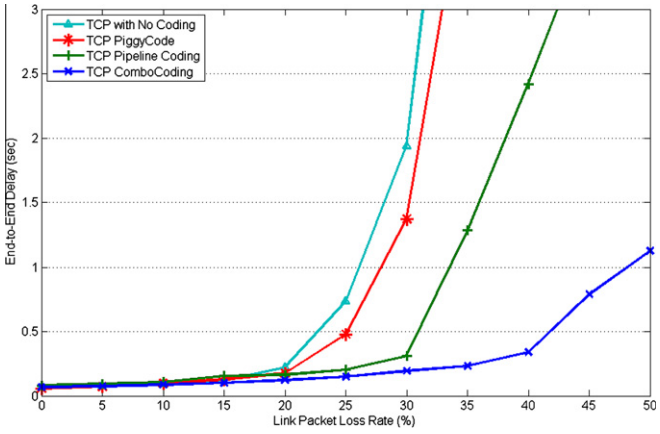


Fig. 8 Delay-to-loss.

ComboCoding is significantly strengthened by intra-flow coding, and as a result its goodput is consistently higher than Pipeline Coding goodput.

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.

The benefits of ComboCoding are also shown in our delay analysis. From the results shown in Fig. 8, we observe that a higher loss rate results in a higher delay. This is because as more packets are lost, it takes more time to deliver a packet to the destination. By correlating delay results with goodput results in Fig. 7, we notice that for all cases, once the throughput drops to almost zero, the delay increases dramatically. Since ComboCoding still achieves about 400 Kbps under 50% packet loss rate, the delay of ComboCoding never increases beyond 2 s and is consistently lower than the other cases.

As network coding relies on redundant packet transmission to compensate for packet losses, it is important to evaluate the transmission overhead. For ComboCoding, the transmission overhead is expected to be reduced by means of inter-flow packet mixing. We define the term "transmission overhead" as:

$$\frac{\sum_{i=1}^{N} S_{tx_i}}{D \times N}, \tag{6}$$

where N is the total number of nodes, D is the total number of DATA packets received by the TCP destination, and S_{tx_i} is the number of MAC frames physically transmitted by node i. This metric is based on the number of frames transmitted in the MAC layer rather than the number of packets sent in the network layer. This is because the reduction of transmitted packets also implies a lower probability of interference with other nodes. Consequently, packet collision probability is reduced and thus fewer MAC frames need to be retransmitted. The number of MAC frames transmitted is obtained from the simulation statistics reported by QualNet. Eq. (6) can be interpreted as the average number of MAC frames needed per node per successful TCP DATA packet delivery.

As mentioned previously, Pipeline Coding requires a higher coding redundancy, and consequently results in the highest transmission overhead as shown in Fig. 9. In the case of perfect links, Pipeline Coding still needs 4.5 transmissions per node in order to deliver 1 TCP DATA packet, which explains why it has the worst goodput when no loss is present. In contrast, PiggyCode has low overhead since it does not introduce any redundant packets, and further reduces transmissions by mixing DATA and ACK opportunistically.

Without random loss, TCP with no coding still needs 4 transmissions per node for a single DATA packet delivery due to collisions. Potential collisions significantly increase the number of transmissions required per successful packet deliv-

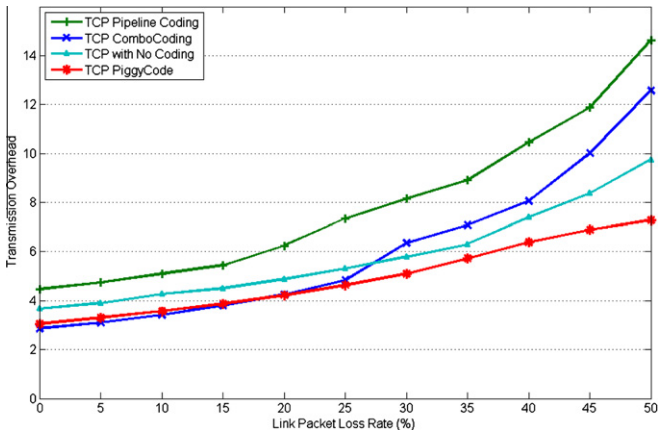


Fig. 9 Overhead-to-loss.

ery. The efficient nature of inter-flow coding is shown in the low loss rate cases, where both PiggyCode and ComboCoding reduce transmission overhead by 20%. The overhead of ComboCoding eventually increases, because the optimal coding redundancy requires more coded packets to be sent to recover more losses. Figs. 7 and 9 show that ComboCoding is robust to losses, while reducing redundant packet transmission is overhead by up to 30% when compared to Pipeline Coding.

Loss adaptation evaluation

We next consider a network with loss rate that varies dynamically over time. This is representative of time varying external interference (e.g., jamming). We wish to evaluate the performance of various coding schemes under this time variable jamming. The application starts sending packets at time 20 s, and during time 20–50 s the packet loss rate for all links is 0%. As shown in Fig. 10, TCP with no coding and PiggyCode outperform all other coding schemes when links are perfectly reliable. Pipeline Coding and ComboCoding perform worse because of the extra overhead due to the redundancy of intra-flow coding. In addition, ComboCoding with loss adaptation performs slightly worse than without, because the adaptive algorithm reacts to short-term losses and thus wastes extra time to lower redundancy.

In the interval from 50 to 80 s, a 40% packet loss rate is introduced on every link. During this interval, all TCP variants without adaptation drop to almost zero throughput, while the loss adaptive ComboCoding still achieves around 1 Mbps. This period shows the importance of loss adaptation and the effectiveness of redundancy, as the adaptive ComboCoding quickly reacts to the loss and continues to perform with acceptable throughput.

From 80 to 110 s, the per link packet loss rate is lowered from 40% to 20%. We notice that TCP with no coding and PiggyCode reemerge and have the highest instantaneous goodput of all, but they are both very unstable due to the lack of proper redundancy. Furthermore, ComboCoding without the adaptive algorithm takes a long time to stabilize because random linear coding needs time to discard undecodable generations resulting from loss. Pipeline Coding takes even longer to recover and does not deliver as much as ComboCoding

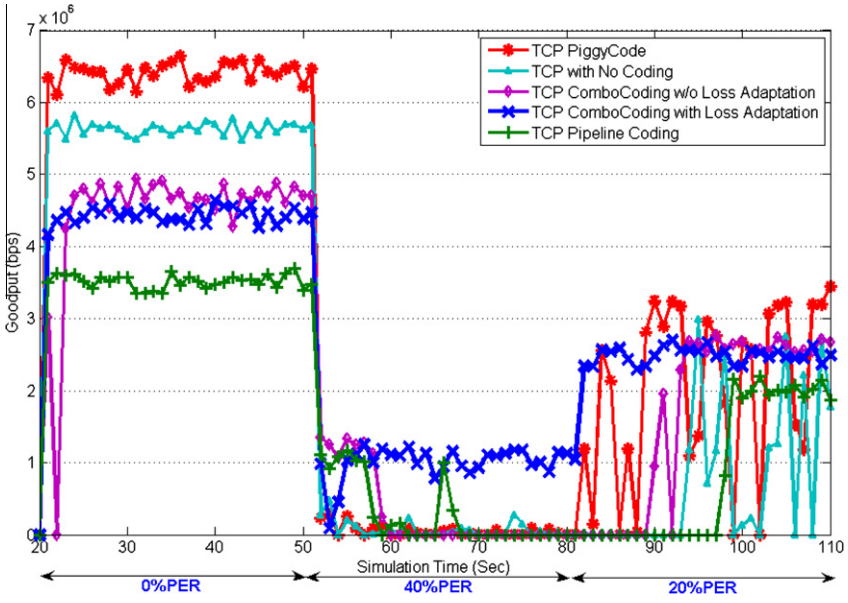


Fig. 10 Goodput over time for loss adaptation.

without adaptation. The best coding scheme is adaptive ComboCoding, which instantly reacts to loss reduction and delivers high and stable throughput.

Long string evaluation

In the last set of simulations, the string topology is further extended step by step from 3 hops to 9 hops. All other configuration parameters remain the same as described in Table 2. The same variable link loss pattern is used as in the previous section, with 0% loss during the 20–50 s period, 40% loss during the 50–80 s period, and finally 20% loss during the 80–110 s period. The same loss adaptation parameters and PiggyCode timer are used for ComboCoding for all simulations in this experiment. An experimentally optimized coding redundancy is set for Pipeline Coding in order to compare other schemes to the best case Pipeline Coding performance.

Table 3 summarizes the simulation results. The pattern already observed in Fig. 10 is repeated for all cases in Table 3. During the first 30 s when the links are perfect, PiggyCode achieves the highest goodput, TCP with no coding has the second highest, ComboCoding is the third, and Pipeline Coding is the worst. When the link loss rate increases to 40% during 50–80 s, only ComboCoding persists, as in Fig. 10. Pipeline Coding still delivers little goodput since the coding redundancy has been tuned for 40% link loss rate over the whole simulation. Note that Pipeline Coding would not survive if such an optimization were skipped. During the last 30 s when the link loss rate is down to 20%, ComboCoding is still the highest among all other coding schemes. Pipeline Coding is the second best. TCP with no coding and PiggyCode tend to be more unstable as the number of hops increases. For example, TCP with no coding is completely inadequate to operate over such high loss rates.

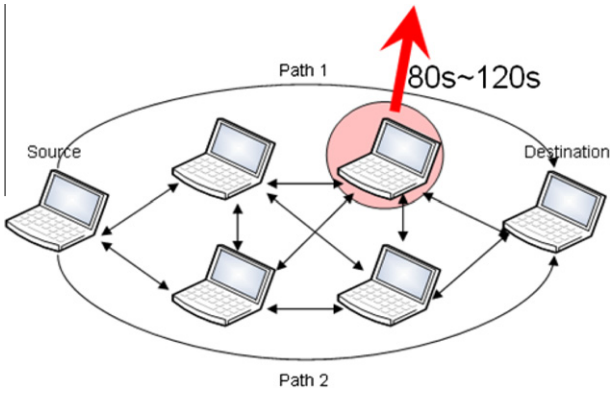


Fig. 12 Multipath MANET topology.

Table 3 Average goodput (Mbps) per loss period for different number of hops.

Number of hops	ComboCoding			TCP-NewReno			Pipeline Coding			PiggyCode		
	20–50 s	50–80 s	80–110 s	20–50 s	50–80 s	80–110 s	20–50 s	50–80 s	80–110 s	20–50 s	50–80 s	80–110 s
3	4.43	1.07	2.50	5.42	0.04	1.78	3.22	0.93	1.93	6.23	0.05	1.64
4	3.67	0.73	1.89	4.17	0.02	0.95	2.70	0.32	1.56	4.92	0.04	1.26
5	3.27	0.81	1.77	3.85	0.02	0.44	2.41	0.27	1.23	4.69	0.02	0.97
6	2.96	0.65	1.53	3.67	0.02	0.37	2.25	0.23	1.16	4.30	0.01	0.69
7	2.66	0.67	1.51	3.61	0.02	0.00	2.19	0.12	1.21	3.96	0.01	0.11
8	2.71	0.44	1.57	3.51	0.00	0.00	1.84	0.08	1.16	3.75	0.01	0.08
9	2.46	0.43	1.46	3.01	0.01	0.00	2.00	0.05	0.70	3.58	0.00	0.10

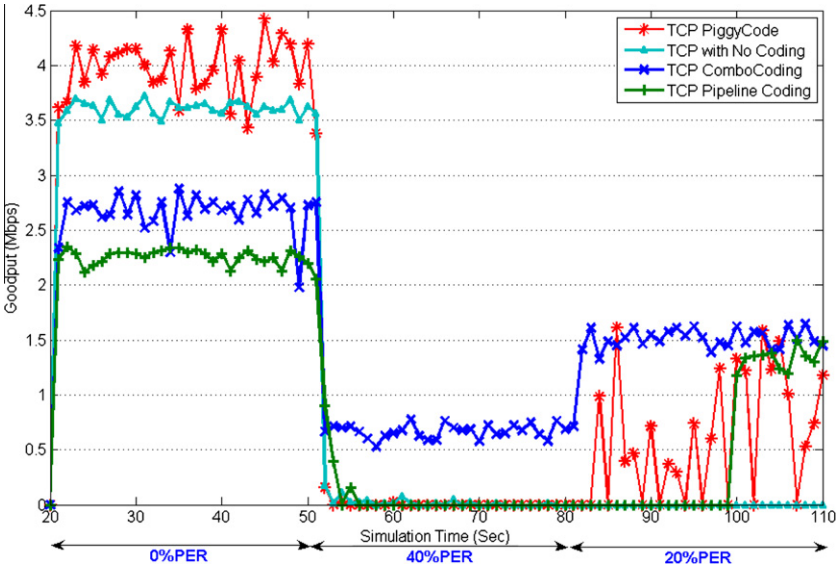


Fig. 11 7-Hop variable loss.

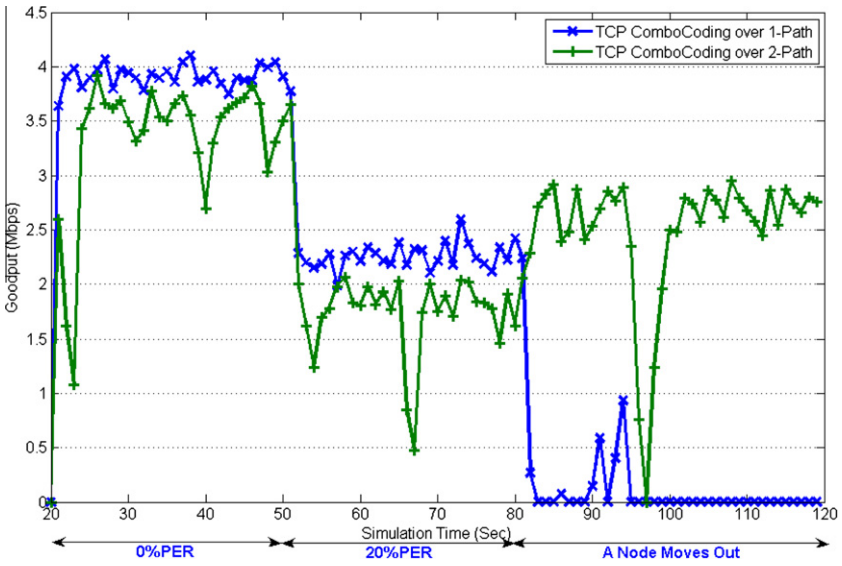


Fig. 13 Goodput over simulation time on multipath MANET.

Fig. 11 selects a representative example from Table 3, which is the 7-hop string simulation. During the first 30 s with perfect links, Fig. 11 shows exactly the same order of performance as in Table 3. When the link loss rate increases from 0% to 40%, similar to Fig. 10, ComboCoding is the only coding scheme that persists. However, since the string is much longer than the topology in Fig. 10, TCP with no coding is unable to recover when the link loss rate reduces from 40% to 20% during 80–110 s. During the 20% loss period, PiggyCode is similarly unstable and Pipeline Coding still takes a long time to recover. Most importantly, ComboCoding recovers quickly and reacts effectively to unstable transient losses.

Preliminary results for multipath MANETs

We next evaluate the proposed loss-adaptive ComboCoding over a multipath mobile ad-hoc network. The simulation configurations are the same as given in Table 2 except an additional path is added as presented in Fig. 12. In this set of simulations, all links had no loss in 20–50 s, and then the per link loss rate increased to 20% during 50–80 s. After 80 s, the link loss rate is back to 0% but one of the relays in path 1 proceeds to gradually move out of the communications range. We compare: (a) ComboCoding over a single path, and (b) ComboCoding over 2 paths as shown in Fig. 13.

Fig. 13 shows that during 20–50 s, ComboCoding over 2 paths delivered the least because more nodes are contending for the channel. During 50–80 s when all links had 20% loss, we note that ComboCoding over 2 paths provides a lower goodput than ComboCoding over a single path. The reason is a higher overhead due to extra contentions as already noticed in the 20–50 s interval.

However, when one relay starts moving away from the area after 80 s, ComboCoding over a single path becomes unstable, and the TCP session is eventually terminated. Path 1 breaks out completely at about 97 s. At this time ComboCoding over 2 paths first drops to zero and then quickly recovers thanks to loss adaptation. This result demonstrates the effectiveness of the proposed loss-adaptive feature in ComboCoding. These results are preliminary and invite future work. In addition to adaptive redundancy control, in the future we plan to evaluate the strategy of adapting the coding mode, from no coding to ComboCoding and if feasible adapting to changes in the width of the breadth configuration, i.e., to detect and adapt to changes in the number of parallel paths.

Conclusion

In this paper, we present 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 codes, ComboCoding reduces the interference between data and ACKs within a TCP session and also exhibits robustness to high link losses. The simulation results show that in a 3-hop string topology, ComboCoding successfully achieves 2 Mbps goodput with 30% per link packet loss rate, while TCP-NewReno with no coding delivers only 200 Kbps. Compared to the original Pipeline Coding, ComboCoding also reduces transmission overhead by 30% under perfect link conditions and by 10% overhead in most other cases. The adaptive ComboCoding was tested on a variable hop string topology with unstable links and on a multipath MANET with dynamic topology. Simulation results show that adaptive ComboCoding outperforms all other coding schemes and quickly adapts to link quality changes. Generalized MANET topologies, better optimized coding schemes, analytic models, and measurement experiments remain to be explored and will be the subject of future studies.

Acknowledgment

J.-S. Park was supported in part by National Research Foundation of Korea Grant funded by the Korean Government (2010-0005334, 2010-0027410).

References

[1] Handley M, Floyd S, Padhye J, Widmer J. TCP friendly rate control (TFRC). RFC 3448; 2003.

[2] Ahn G-S, Campbell AT, Veres A, Sun L-H. SWAN: service differentiation in stateless wireless ad hoc networks. In: Proceedings of the IEEE INFOCOM; 2002.

[3] Vicente E, Mujica V, Sisalem Dorgham, Popescu-Zeletin Radu, Wolisz Adam. TCP-friendly congestion control over wireless networks. In: Proceedings of the European wireless; 2004.

[4] Haider Naqvi Ijaz, Perennou Tanguy. A DCCP congestion control mechanism for wired-cum-wireless environments. In: Proceedings of the IEEE wireless communications and networking conference; 2007.

[5] Xu K, Bae S, Lee S, Gerla M. TCP behavior across multihop wireless networks and the wired internet. In: Proceedings of the ACM WoWMoM 2002; 2002.

[6] Li J, Blake C, Couto D, Lee H, Morris R. Capacity of ad hoc wireless networks. In: Proceedings of the ACM SIGMETRICS 2000; 2000.

[7] Chen J, Gerla M, Lee YZ, Sanadidi MY. TCP with delayed ACK for wireless networks. In: Ad hoc networks, vol. 6; 2008. p. 1098–116.

[8] Ahlswede R, Cai N, Li S-YR, Yeung RW. Network information flow. IEEE Trans Inf Theory 2000;46(4):1204–16.

[9] Koetter R, Medard M. An algebraic approach to network coding. IEEE/ACM Trans Networking 2003;11(5):782–95.

[10] Chou PA, Wu Y, Jain K. Practical network coding. In: Proceedings of the Allerton conference on communication, control, and computing; 2003.

[11] Katti S, Rahul H, Hu W, Katabi D, Medard M, Crowcroft J. XORs in the air: practical wireless network coding. IEEE/ACM Trans Networking 2008;16(3).

[12] Chachulski S, Jennings M, Katti S, Katabi D. Trading structure for randomness in wireless opportunistic routing. In: Proceedings of the ACM SIGCOMM; 2007.

[13] Park J-S, Gerla M, Lun DS, Yi Yu, Medard M. CodeCast: a network coding based ad hoc multicast protocol. IEEE Wireless Commun 2006.

[14] Oh SY, Gerla M. Robust MANET routing using adaptive path redundancy and coding. In: Proceedings of the first international conference on communication systems and networks (COMSNETS); 2009.

[15] Chen C-C, Lien C-N, Lee U, Oh SY. CodeCast: network coding based multicast in MANETs. In: Demos of the 10th international workshop on mobile computing systems and applications (HotMobile 2009); 2009.

[16] Huang Y, Ghaderi M, Towsley D, Gong W. TCP performance in coded wireless mesh networks. In: Proceedings of the IEEE SECON; 2008.

[17] Scalia L, Soldo F, Gerla M. PiggyCode: a MAC layer network coding scheme to improve TCP performance over wireless networks. In: Proceedings of the IEEE GLOBECOM; 2007.

[18] David PS, Kumar A. Network coding for TCP throughput enhancement over a multi-hop wireless network. In: Proceedings of the IEEE COMSWARE; 2008.

[19] Sundararajan JK, Shah D, Medard M, Mitzenmacher M, Barros J. Network coding meets TCP. In: Proceedings of the IEEE INFOCOM; 2009.

[20] Sundararajan JK, Shah D, Medard M. Online network coding for optimal throughput and delay – the three-receiver case. In: Proceedings of the international symposium on information theory and its applications (ISITA 2008); 2008.

[21] Chen C-C, Oh SY, Tao P, Gerla M, Sanadidi MY. Pipeline network coding for multicast streams (invited paper). In: Proceedings of the 5th international conference on mobile computing and ubiquitous networking (ICMU 2010); 2010.

[22] Tao P, Chen C-C, Oh SY, Gerla M, Sanadidi MY. Demo abstract: pipeline network coding for multicast streams. In: Demos of IEEE INFOCOM 2010; 2010.

[23] Qin Chuan, Xian Yi, Gray Chase, Santhapuri Naveen, Nelakuditi Srihari. I2MIX: integration of intraflow and interflow wireless network coding. In: Proceedings of the IEEE SECON workshops; 2008.

[24] Alimi R, Li L, Ramjee R, Viswanathan H, Yang Y. ipack: in-network packet mixing for high throughput wireless mesh networks. In: Proceedings of the IEEE infocom; 2008.

[25] Tiwari A, Ganguli A, Sampath A, Anderson DS, Shen B-H, Krishnamurthi N, et al. Mobility aware routing for the airborne network backbone. In: Proceedings of the IEEE MILCOM; 2008.

[26] Ho T, Medard M, Shi J, Effros M, Karger D. On randomized network coding. In: Allerton; 2003.

[27] Scalable Networs Inc. QualNet. http://www.scalble-networks.com.

[28] Xu K, Gerla M, Bae S. Effectiveness of RTS/CTS handshake in IEEE 802.11 based ad-hoc networks. In: Ad hoc networks, vol. 1; 2003. p. 107–23.

[29] Padhey J, Firoiu V, Towsley D, Kurose J. Modeling TCP throughput: a simple model and its empirical validation. In: Proceedings of the ACM SIGCOMM; 1998.