TCP Performance in Coded Wireless Mesh Networks

Yong Huang*, Majid Ghaderi †, Don Towsley‡ and Weibo Gong*
*Department of Electrical and Computer Engineering, University of Massachusetts Amherst
†Department of Computer Science, University of Calgary
‡Department of Computer Science, University of Massachusetts Amherst

***Abstract*—This paper investigates the benefit of network coding for TCP traffic in a wireless mesh network. We implement network coding in a real 802.11a wireless mesh network and measure TCP throughput in such a network. Unlike previous implementations of network coding in mesh networks, we use *off-the-shelf* hardware and software and do not modify TCP or the underlying MAC protocol. Therefore, our implementation can be easily exported to any operational wireless mesh network with minimal modifications. Furthermore, the TCP throughput improvement reported in this paper is due *solely* to network coding and is orthogonal to other improvements that can be achieved by optimizing other system components such as the MAC protocol. We conduct extensive measurements to understand the relation between TCP throughput and network coding in different mesh topologies. We show that network coding not only reduces the number of transmissions by sending multiple packets via a single transmission but also results in a smaller loss probability due to reduced contention on the wireless medium. Unfortunately, due to asynchronous packet transmissions, there is often little opportunity to code resulting in small throughput gains. Coding opportunity can be increased by inducing small delays at intermediate nodes. However, this extra delay at intermediate nodes results in longer round-trip-times that adversely affect TCP throughput. Through experimentation, we find a delay in the range of 1 ms to 2 ms to maximize TCP throughput. For the topologies considered in this paper, network coding improves TCP throughput by 10% to 85%.**

I. Introduction

In traditional networks, data packets are carried by *store-and-forward* mechanisms in which the intermediate nodes (relays or routers) only repeat data packets that they have received. The concept of *network coding* was introduced for satellite communications in [1] and then fully developed in [2] for general networks. With network coding, a network node is allowed to *combine* several packets that it has generated or received into one or several outgoing packets.

The original paper of Ahlswede *et al.* [2] showed the utility of network coding for multicast in wireline networks. Recently, network coding has been applied to wireless networks and received significant popularity as a means of improving network capacity and coping with unreliable wireless links [3]–[7]. In fact, the unreliability and broadcast nature of wireless links make wireless networks a natural setting for network coding. Moreover, network protocols in wireless networks, *e.g.*, wireless mesh networks and mobile ad hoc networks, are not fully developed yet and hence there is more freedom to apply network coding in such environments compared to wireline networks such as the Internet [5].

In spite of many research papers on the application of network coding in wireless networks, surprisingly, there are not many real implementations. Because of the need for tractability, theoretical results on network coding do not account for the detailed behavior of a wireless network, *e.g.*, asynchronous transmissions due to random delays. Therefore, it is not well understood to what extent network coding improves the throughput capacity of a wireless network in a real implementation. This paper aims at characterizing and quantifying such throughput improvements in the context of wireless mesh networks (WMNs) employing network coding. We focus on TCP traffic, and hence TCP throughput improvement due to network coding, because TCP is expected to be the dominant transport protocol in WMNs. Surprisingly, TCP performance in coded wireless networks is a largely unknown area (in fact, network coding with multiple unicast flows has been just recently studied in [8]).

A few papers that have implemented network coding in a wireless setting do not specifically consider TCP (see [3] for example). The only exception is the so-called COPE [6] which implements an opportunistic network coding scheme. In [6], the authors performed TCP experiments to understand how network coding impacts TCP throughput in a mesh setting. However, they did not fully explore the interactions between TCP and network coding. In this paper, we perform extensive experiments in our COTS[1] implementation to understand how network coding interacts with TCP and exploit these interactions to maximize TCP throughput.

TCP traffic is naturally bidirectional, *i.e.*, data packets in one direction and ACK packets in the opposite direction, and hence network coding can be applied at intermediate nodes along the path even for a single TCP flow. Unfortunately, due to random delays in networks, coding opportunities at intermediate nodes may be too small to benefit TCP. We show that inducing a small *delay* at each intermediate node can increase coding opportunity for TCP traffic, especially when there are only a few TCP flows in the network. However, there is a trade-off between increased coding opportunity and increased TCP round-trip-time by increasing delay at intermediate nodes. Using real measurements from our test-bed, we characterize the proper amount of delay in different mesh topologies and

[1]Commercial Off-The-Shelf hardware and software.

show how the induced delay affects coding opportunity, packet loss probability and round-trip-time.

We also develop a simple model for TCP throughput with network coding in a line topology, and use the model to study how network coding impacts packet loss probability and TCP throughput. We show that network coding improves TCP throughput in two ways: 1) by increasing the wireless channel capacity due to coding packets together, and 2) by reducing packet loss probability due to reduced contention on the wireless channel.

The contributions of this paper can be summarized as follows:

1) It presents an implementation of network coding in a wireless mesh network using commercial off-the-shelf hardware and software.
2) It develops a simple model for TCP throughput and studies the impact of network coding on end-to-end loss probability and TCP throughput.
3) It proposes a delay-based approach to increase coding opportunities at intermediate nodes in a network and studies the impact of induced delay on coding opportunity, packet loss and TCP round-trip-time. Finally, it empirically characterizes the proper amount of delay that maximizes TCP throughput.

The rest of the paper is organized as follows. In Section II, we briefly describe the fundamental ideas behind network coding and review related work in the context of wireless mesh networks. Section III studies the impact of network coding on TCP throughput by developing a simple model for TCP throughput in a line topology. In Section IV, we provide detailed description of our implementation of network coding in a WMN. We summarize our experiments in Section V. Our conclusions as well as future work are discussed in Section VI.

II. Network Coding

Consider a network composed of a set of source-destinations and a set of intermediate nodes acting as relays, *e.g.*, routers in Internet. Traditionally, relays simply repeat the packets they have received. With network coding, relays are allowed to combine packets that they have received into one or more packets before sending the packets over outgoing links.

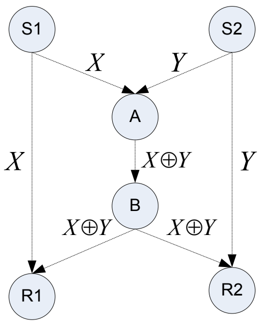


Fig. 1. Butterfly example.

The concept of network coding is easiest explained using the famous butterfly example depicted in Fig. 1. All links have unit capacity, *e.g.*, one packet per time unit, and senders S1 and S2 want to send two packets X and Y to both receivers R1 and R2. Clearly, link A-B is the bottleneck link, and hence, *four* units of time are required to transmit X and Y to R1 and R2 using a store-and-forward mechanism. However, using the transmissions outlined in the figure, the multicast problem can be solved using only *three* units of time. In this example, node A combines X and Y using XOR operation (denoted by $\oplus$) and transmits X $\oplus$ Y in a single transmission.

In general, nodes can use different coding techniques to combine packets, however, Li *et al.* [9] showed that linear coding suffices to achieve the max-flow, *i.e.*, the optimum, in single source multicast networks. Then, the problem of network coding is how to select the linear combinations that each node of the network performs. In practice, most network coding approaches are based on the concept of *random linear coding* proposed by Ho *et al.* [10]. With random linear coding, each node in the network selects the linear coding coefficients uniformly at random over a finite field in a completely independent and decentralized manner.

In wireless environments, network coding has been applied to various problems including broadcasting in ad hoc networks [7], data collection in sensor networks [7], file sharing in mesh networks [11] and reliability in lossy networks [4]. In the context of mesh networks, in particular, Wu *et al.* [3] investigated the use of network coding for the mutual exchange of independent information between two nodes in a wireless network. They showed that network coding can be used to increase the capacity of a wireless network with bidirectional traffic. Consider the network depicted in Fig. 2. Node A wants to send packet X to node C and node C wants to send packet Y to node A. With traditional store-and-forward routing, X and Y belong to two different unicast flows, one from A to C and the other from C to A. Hence, two routes are created to exchange packets between A and C. In this case, to exchange X and Y, four time slots are required.

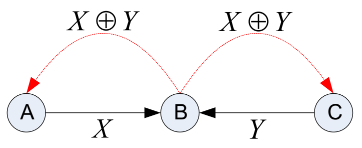


Fig. 2. Network coding for bidirectional traffic.

However, by using network coding and broadcasting, as shown in the figure, the exchange can be performed in only three time slots. In the first two time slots, X and Y are transmitted to node B, and then in the third time slot, node B broadcasts X $\oplus$ Y to nodes A and C. Upon receiving X $\oplus$ Y from B, node A (C) extracts Y(X) using its existing copy of X(Y). Therefore, one transmission is saved, which effectively increases the capacity by 25% compared to traditional store-and-forward scheme.

In this paper, we extend and implement the scenario depicted in Fig. 2 in a general wireless mesh network. In our network, whenever an intermediate node has an opportunity to transmit, it first checks to see if it is possible to code packets together before transmission. When there is only one TCP flow, the packets that are coded together include one TCP data packet in forward direction and a TCP ACK packet in the other direction. Nodes are not synchronized in our network, and hence it may not be always possible to code packets together. A challenging problem then is to increase the coding opportunity at intermediate nodes without penalizing TCP throughput.

III. Impact of Network Coding on TCP

In this section, using a simple model, we study the impact of network coding as described in the previous section on TCP throughput. Despite the simplicity of the model, it provides interesting insight about the interactions between TCP and network coding that will be helpful when discussing our experiment results in Section V.

We argue that TCP dynamics, specifically the AIMD congestion control mechanism, have a significant impact on the benefits of network coding. TCP congestion control mechanism, continuously adapts TCP sending rate to network conditions and available capacity. In particular, the AIMD mechanism is extremely sensitive to packet losses and interprets them as signs of congestion. Upon detecting a loss, TCP halves its sending rate by reducing its congestion window size to half. In a WMN based on a contention-based MAC protocol such as IEEE 802.11, there are significant number of packet losses due to wireless channel errors and contention on the wireless medium. TCP reacts to all such packet losses by reducing its transmission rate which results in poor throughput and low wireless channel utilization. Consequently, TCP throughput is primarily limited by the end-to-end loss probability rather than the available end-to-end capacity. Hence, TCP may not significantly benefit from the increased capacity due to coding compared to a non-congestion-controlled traffic such as UDP traffic (as considered in [3]).

In our test topology, due to close proximity of wireless nodes, contention on the wireless medium is relatively high. High contention results in high end-to-end packet loss probability which *prevents* TCP from fully utilizing the channel. In fact, TCP is not even able to utilize the channel capacity available to it without coding, and hence increasing channel capacity with coding does not significantly benefit TCP. Instead, TCP benefits from coding indirectly. Coding reduces the number of transmissions which results in *lower contention* on the wireless medium. This helps TCP to increase its sending rate because it faces packet loss less frequently compared to no-coding case. However, the increased throughput, in turn, leads to increased contention. In steady-state, there will be a balance between increased TCP throughput and increased contention. In the following, we develop a simple model to compute TCP throughput and loss probability that are achieved in steady-state.

A. Packet Loss Probability

To investigate how network coding impacts TCP throughput, we consider a simple line topology with n hops. We assume that there is a TCP flow from the first node (source) to the last node (destination) in the line topology. We consider a homogenous scenario and assume that packets (data packets or ACK packets) are lost over each hop with probability p_l. We further assume that the receiver sends an ACK for every successfully received data packet. If either the data packet or the ACK packet is lost, a TCP loss occurs. Therefore, to successfully send one packet from the source to the destination, $2n$ transmissions are required (n transmissions for the data packet and another n transmissions for the ACK). Let p denote the end-to-end packet loss probability seen by TCP. It is obtained that

$$p = 1 - (1 - p_l)^{2n}, \quad (1)$$
$$\approx 2np_l, \qquad \text{for small } p_l \text{ and large } n\,. \quad (2)$$

A packet is lost either due to channel errors or *collision* with other transmissions. Channel errors depend on the inherent characteristics of the wireless medium and are independent of traffic load. However, collision induced losses depend on traffic load, and hence can be different with and without network coding. Let p_e and p_c denote packet loss probability due to channel errors and contention respectively. We have

$$p_l = 1 - (1 - p_e)(1 - p_c), \quad (3)$$
$$\approx p_c, \qquad \text{for small } p_e/p_c\,. \quad (4)$$

The above approximation is valid when collision is the dominant cause of packet loss. In our experiments, all transmissions are broadcasts which are more resilient to channel errors due to stronger physical-layer coding applied in broadcast mode of IEEE 802.11 [2], and hence this is a reasonable approximation. Therefore, the end-to-end loss probability is given by $p = 2np_c$.

B. Collision Probability

To estimate the collision probability, we consider a time-slotted system where every transmission takes one time slot. In our line topology, at most three nodes interfere with each other. Consider three such interfering nodes and assume that packets arrive at a node according to a Bernoulli process with mean X packets/slot (for a normalized channel capacity of $C = 1$ packets/slot). Let λ_1 and λ_2 denote the transmission probability in a time slot with and without coding respectively. With network coding, for every other packet arrival there is one transmission. Hence the transmission probability is given by $\lambda_1 = X/2$. Without coding, for each arrival there is one transmission. Hence the transmission probability is equal to the arrival rate $\lambda_2 = X$.

Let p_{c1} and p_{c2} denote the collision probability with and without network coding. A collision occurs if more than one

[2]Which also results in a lower channel rate.

node transmit at the same time. Thus, p_{ci} $(i = 1, 2)$ is:

$$p_{ci} = \binom{3}{2}(1-\lambda_i)\lambda_i^2 + \binom{3}{3}\lambda_i^3, \quad (5)$$
$$\approx 3\lambda_i^2, \qquad \text{for small } \lambda_i \quad . \quad (6)$$

It is straightforward to extend this analysis to more than three contending nodes. In general, if m nodes contend with each other for the wireless channel, we obtain that

$$p_{ci} \leq \binom{m}{2}\lambda_i^2, \quad (7)$$
$$= \frac{m(m-1)}{2}\lambda_i^2, \qquad \text{for small } \lambda_i \, . \quad (8)$$

In our experiments, due to high packet loss probability, TCP cannot efficiently utilize the wireless channel, and hence the above approximation is reasonable.

C. TCP Throughput

Let p_1 and p_2 denote the end-to-end loss probability with and without coding. We assume that losses due to channel errors are negligible so that all losses are due to contention. Let X_1 and X_2 denote the mean TCP throughput with and without coding. Using (6), it is obtained that

$$p_{c1} = 3(\frac{X_1}{2})^2, \quad (9)$$
$$p_{c2} = 3X_2^2 \, . \quad (10)$$

By substituting in (2), we obtain that $p_1 = \frac{6}{4}nX_1^2$ and $p_2 = 6nX_2^2$.

Let L and R denote the TCP packet size and round-trip-time respectively. Then, using the well-known square root formula [12], TCP throughput can be approximated by

$$X_i \approx \frac{L}{R\sqrt{p_i/2}} \, . \quad (11)$$

In steady-state, TCP sending rate X_i and packet loss probability p_i balance each other. Therefore, it is obtained that

$$X_1 = \frac{2L}{RX_1\sqrt{3n}}, \quad (12)$$
$$X_2 = \frac{L}{RX_2\sqrt{3n}}, \quad (13)$$

which yield,

$$X_1 = \Big(\frac{2L}{\sqrt{3n}R}\Big)^{1/2}, \quad (14)$$
$$X_2 = \Big(\frac{L}{\sqrt{3n}R}\Big)^{1/2}. \quad (15)$$

Several observations can be made regarding the above expressions:

1) *Impact of path length:* As n increases, TCP throughput decreases to zero because end-to-end loss probability approaches 1.
2) *Impact of coding*: It is easy to see that $X_1 = \sqrt{2}X_2$, indicating a factor of $\sqrt{2}$ improvement in TCP throughput when coding is implemented. Contrast this with a factor of 2 improvement in throughput if UDP was used instead of TCP.

IV. Network Coding Implementation

In our test-bed, the network coding module works between the link layer (layer 2) and the network layer (layer 3) based on Mesh Connectivity Layer (MCL) toolkit [13] and is transparent to other layers. MCL is a loadable Microsoft Windows driver that provides wireless mesh connectivity. It uses a modified Dynamic Source Routing (DSR) called Link Quality Source Routing (LQSR) to support link quality metrics. When it receives packets from layer 3, MCL looks up its routing tables and encapsulates the source routing information into a LQSR header. Then the packet is passed to the link layer and finally transmitted through the physical layer device.

A. Unicast and Broadcast in IEEE 802.11

We use off-the-shelf IEEE 802.11a network cards to build our test-bed. As defined by 802.11 MAC, packets are transmitted either in unicast mode or broadcast mode. In our implementation, we had to make a decision on whether to use unicast or broadcast mode. The unicast mode has some special properties compared to the broadcast mode:

- *Link Layer Acknowledgement:* In unicast mode, the receiver node immediately sends an acknowledgement for each radio block[3] that is received correctly at the link layer. Link layer acknowledgements are fast because they are not subject to contention as IP layer packets are.
- *Link Layer Retransmission:* Until the link layer acknowledgement is obtained, the sender assumes the radio block is still in the air. The sender will retransmit the radio block if a retransmission timer expires. Retransmissions may be repeated until a maximum number of retransmissions has been reached.
- *Back-Off Window Adjustment:* Before every retransmission, the sender doubles its back-off window. A node randomly sets a back-off counter between 0 and the current back-off window size before any transmission. This counter is decremented by one time-slot every time there is no medium activity during the back-off period. The node sends the packet once the back-off counter reaches 0. Clearly, the back-off mechanism reduces collision probability significantly.
- *Dedicated Receiver:* In unicast mode, the destination is unique and its physical address is placed in the Ethernet header. If a node is not the receiver, it drops the packet immediately at the physical layer unless it works in promiscuous mode.

In comparison, the broadcast mode provides only a basic communication channel. Specifically, the broadcast mode does not support link layer acknowledgements and retransmissions. As the result, there is no link layer retransmission and no back-off window adjustment. This effectively results in higher packet loss probability which adversely impact TCP throughput. On the other hand, it uses a broadcast address which

[3]At link layer, each TCP segment may be broken up into a number of radio blocks for transmission. In our implementation, a TCP segment fits into a single radio block.

allows all nodes in the transmission range to receive the packet, and the physical layer at receiver side passes the payload to upper layers for further processing. This is a necessary feature for implementing network coding in our test-bed. Otherwise, node B in Fig. 2 has no option but to send X and Y separately to A and B, effectively eliminating the benefit of network coding.

B. Pseudo-Broadcast or Pseudo-Unicast

As described earlier, coded packets must be transmitted to multiple receivers to gain throughput improvement. One way to do it is to put nodes into promiscuous mode; another way is to use broadcast mode to transmit coded packets. Katti *et al.* [6] use the first approach in their Linux-based implementation of COPE. COPE uses the so-called *pseudo-broadcast* approach in order to send coded packets using unicast functionality of IEEE 802.11. The receivers work in promiscuous mode and receive every packet in the air. Using pseudo-broadcast, a coded packet can be delivered to multiple receivers simultaneously.

However, only a few commercial wireless cards support promiscuous mode. Even for those that support promiscuous mode, it is impossible to switch them to promiscuous mode in Windows platform without having access to the card's special driver API. Also, the pseudo-broadcast requires an intelligent and complicated broadcast operation that requires modifications in the MAC layer. To avoid such modifications that are not in general supported by commercial hardware, our implementation uses the second approach, *i.e.*, the physical broadcasting method, to implement network coding.

In our implementation, coded packets are always broadcasted. To have a fair comparison of TCP throughput with and without coding, and to eliminate the impact of different MAC-layer functionalities, we use broadcast for non-coded packets as well. Although this results in inferior TCP performance, it does provide a fair comparison between coding and no-coding performance, and any improvement achieved is *solely* due to network coding. We refer to this approach, implementing unicast with broadcast functionality of IEEE 802.11, as *pseudo-unicast*. With pseudo-unicast, TCP operates on the same MAC and link-layer with and without network coding.

C. Implementation

Network coding module generates either pseudo-unicast packets or coded packets based on LQSR packets created by the MCL module. We first show the packet structures generated by different modules in Fig. 3, and then illustrate the operation of our implementation in Fig. 4.

As shown in Fig. 3, the MCL module generates LQSR packets by inserting an additional LQSR header into Ethernet packets received from the network layer. Compared to LQSR packets, a pseudo-unicast packet has an additional network coding (NC) Header and its destination in the Ethernet header is set to the broadcast address (0xffffff). The NC header holds the packet identification information used in coding and decoding operations. The identification information includes the original destination address, the packet ID and the payload length. Fig. 3 also shows the structure of a coded packet $A \oplus B$. The destination of the coded packet is also set to the broadcast address. The NC header includes the triples (real destination address, ID, payload length) of both packet A and packet B. The rest of the coded packet consists of the LQSR headers of packets A and B and the XOR-ed payload. As shown in the figure, coded packets have a longer header than non-coded packets. A better design of the header, *e.g.*, XOR-ing the LQSR headers of A and B, can reduce this overhead.

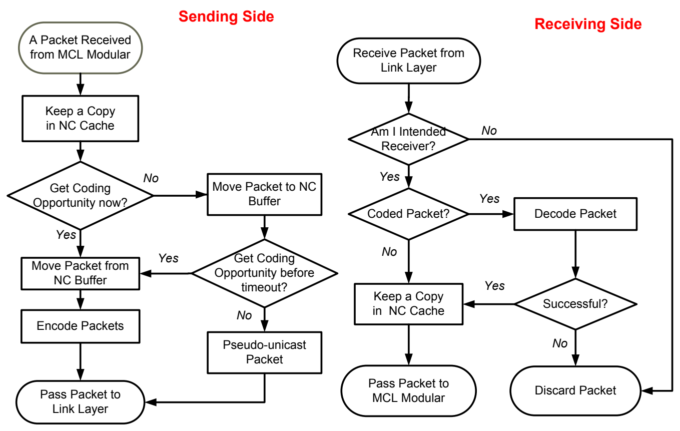


Fig. 4. Network coding implementation.

Fig. 4 depicts the operations of our network coding implementation. The module consists of sending and receiving procedures. Two First-In-First-Out (FIFO) queues are used to store the packets. The first, called NC Cache, is used to keep copies of all outgoing and incoming packets for decoding purpose. Another FIFO queue is the NC Buffer. It holds the outgoing packets that have not found a coding opportunity yet.

The sending procedure seeks coding opportunities for all outgoing packets. First, a local copy of an outgoing packet is stored in the NC Cache for decoding purpose. Then, if there is another packet in the NC Buffer going in the opposite direction, the two packets are encoded immediately. Finally, the coded packet is passed to the link layer to be transmitted.

If there is no coding opportunity when the packet arrives, it waits in the NC Buffer for a future coding opportunity. Every packet in the NC Buffer has an associated timer with an initial value of `NCBufTimeout`. If this timer expires, the packet will be pseudo-unicasted. `NCBufTimeout` is an important parameter. Note that `NCBufTimeout` may increase TCP throughput by creating more coding opportunity, but may decrease TCP throughput by increasing the round-trip-time. Hence, there is a trade-off between the increased coding opportunity and increased round-trip-time. We will dwell on this trade-off later in Section V.

When receiving a packet, as shown in Fig. 4, a receiver first checks whether its address is included in the NC Header of the received packet. If not, the packet is discarded immediately. Otherwise, if the packet is a coded packet, the receiver tries to decode it. To decode a packet, *e.g.*, packet A from a coded

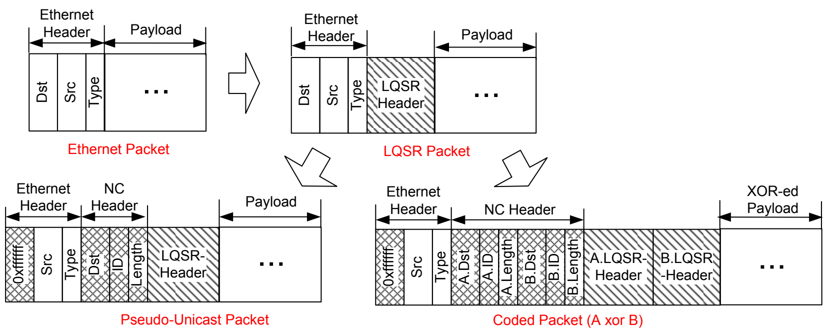
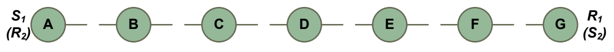


Fig. 3. Packet generated by different modules: Ethernet packets are from IP layer, LQSR packets are generated by MCL, and network coding module generates either pseudo-unicast packets or coded packets that are passed to MAC layer for transmission.

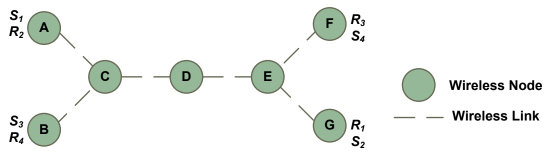
packet A⊕B, the receiver will look for packet B in its NC Cache. If packet B is found then packet A can be successfully decoded, and a local copy of packet A will be stored in the NC Cache. Otherwise, packet A⊕B is simply discarded. On the other hand, if the received packet is pseudo-unicasted (*i.e.*, it is a none-coded packet), the receiver only needs to store a local copy of the received packet into its NC Cache. Finally, the processed packet is passed to the upper layer via the MCL module.

V. Performance Evaluation

In this section, we present measurement results for TCP performance in our test-bed with different topologies. The first set of experiments are conducted over two controlled topologies consisting of 7 nodes. These topologies, depicted in Figs. 5(a) and Fig. 5(b), represent standard *line* and *dumbbell* topologies commonly used for TCP performance evaluation. These simple topologies are easy to analyze and clearly show the interactions between network coding and TCP. The second set of our experiments are conducted over an unplanned mesh topology deployed in the second floor of the Electrical Engineering Department at UMass. This topology, depicted in Fig. 7, consists of 20 nodes and is used to highlight TCP performance in more realistic scenarios.



(a) Line topology.



(b) Dumbbell topology.

Fig. 5. Controlled network topologies: S_i and R_i are the sender and receiver of TCP flow i, respectively.

The test-bed consists of 12 Dell laptops and 8 Dell desktop computers running Windows XP SP2. They are located in the second floor of Knowles Engineering Building on UMass campus. Three types of 802.11a/b/g dual-band wireless cards are used, including Dell 1450 USB, Dell 1470 MiniPCI and Linksys WPC55AG PCMCIA cards. Note that diversity of nodes and wireless cards provides a more realistic test environment. To avoid interference with existing 802.11b/g networks, the test-bed uses 802.11a. In all experiments, TCP sources generate `ftp`-like traffic with fixed-length packet size (1020 bytes) for 90 seconds periods. To control routing and hop distances, static routing is used as well. For each experiment we report the average of 10 experiments with the same test-bed configuration.

A. Performance Metrics

We are interested in the throughput gain achieved by network coding. Throughput gain is defined as:

$$\text{Gain} = \frac{\text{Throughput}_{\text{Coding}} - \text{Throughput}_{\text{NoCoding}}}{\text{Throughput}_{\text{NoCoding}}} \times 100\% \quad (16)$$

where $\text{Throughput}_{\text{Coding}}$ and $\text{Throughput}_{\text{NoCoding}}$ denote average TCP throughput when network coding is turned on and off respectively.

The following observations are made with respect to TCP throughput in our experiments:

- *Network coding may decrease loss probability, hence increase TCP throughput:* TCP can achieve a higher throughput with a lower TCP loss rate. Network coding uses fewer transmissions to carry packets between a source-destination pair, and hence reduces the contention on wireless medium. Consequently, the overall packet loss probability is reduced and TCP throughput increases. TCP usually cannot fully utilize channel capacity. Also, the coding opportunity can be very low because the data and ACK packets may not arrive at a node at the same time. Using timers at intermediate nodes can create more coding opportunity. To show the impact of coding on end-to-end loss probability, we measure the percentage of loss reduction in our experiments. Define *loss reduction* as follows:

$$\text{Loss Reduction} = \frac{\text{Loss}_{\text{NoCoding}} - \text{Loss}_{\text{Coding}}}{\text{Loss}_{\text{NoCoding}}} \times 100\% \,. \quad (17)$$

- *Network coding may increase round-trip-time, hence reduce TCP throughput:* TCP has a lower throughput with a larger round-trip-time. The extra waiting at intermediate nodes caused by the use of timers increases the average round-trip-time. In particular, large value of `NCBufTimeout` results in a longer round-trip-time, and consequently decreases TCP throughput. To show the impact of coding on round-trip-time, we measure the percentage of RTT increase in our experiments. Define *RTT increase* as follows:

$$\text{RTT increase} = \frac{\text{RTT}_{\text{Coding}} - \text{RTT}_{\text{NoCoding}}}{\text{RTT}_{\text{NoCoding}}} \times 100\% \,. \tag{18}$$

Clearly, there is a trade-off between reduced loss probability and increased round-trip-time. In general, TCP is more sensitive to loss probability than to round-trip-time (see the square-root formula). Thus, we expect to see throughput gain for small values of `NCBufTimeout`. To investigate the impact of `NCBufTimeout` on coding opportunity in the network, we measure coding ratio in our experiments. Define the average *coding ratio* (CR) as the number of coded packet transmissions over the total number of packet transmissions. In our implementation, CR depends on the parameter `NCBufTimeout`. On one hand, larger `NCBufTimeout` increases CR, and hence reduces the loss probability. On the other hand, larger `NCBufTimeout` results in longer round-trip-time.

In our experiments, we measure the following metrics:

1) TCP throughput gain,
2) Average coding ratio (CR),
3) Average TCP loss probability (p),
4) Average round-trip-time (RTT).

To highlight the impact of coding on loss probability and round-trip-time, we present the loss reduction and RTT increase in the next subsection. We also show the throughput gain achieved by network coding, and experimentally characterize the proper values of `NCBufTimeout` that maximizes TCP throughput in different topologies.

B. Experiments on Planned Topologies

Three experiments are conducted on planned topologies depicted in Fig. 5. The first experiment has one TCP flow in the line topology shown in Fig. 5(a) from node A to G. Nodes B, C, D, E, and F act as relays. The second experiment has two TCP flows in the same topology, one flow from A to G and another flow from G to A with symmetric traffic. The last experiment has four TCP flows in the dumbbell topology depicted in Fig. 5(b), from A to G, G to A, B to F, and F to B, respectively. Nodes C, D and E are relays.

Fig. 6(a) presents the TCP throughput [4] gain achieved by network coding in different topologies with different values of `NCBufTimeout`. The figure shows that 1.5 milliseconds is the best `NCBufTimeout` in all experiments. In the line topology with one TCP flow, the largest throughput gain is 70%. We observe that as the number of TCP flows increases, the throughput gain decreases. The reason is that multiple TCP flows increase channel utilization which results in high volume of packet losses due to contention on the medium. TCP is very sensitive to high loss probability and achieves very low throughput. Although, network coding reduces the loss probability, it is still high enough to degrade TCP throughput significantly. From our model in Section III, we know that as p_l approaches 1, the end-to-end loss probability approaches 1 exponentially causing poor performance with and without coding. Although our model considers a single TCP flow, the impact of packet loss due to contention with other flows in the network can be captured with p_e in the model. In this case, p_e dominates so that the improvement in p_c due to coding becomes less significant on TCP throughput.

To have a better understanding of how `NCBufTimeout` affects TCP performance, Figs. 6(b) and 6(c) present loss probability increase and RTT decrease for the three different experiments respectively. As `NCBufTimeout` increases, CR increases and the number of transmissions decreases as well. Thus the average loss probability monotonically decreases as `NCBufTimeout` increases. Although larger `NCBufTimeout` results in smaller loss probability, Fig. 6(c) show that (as expected) RTT increases linearly as `NCBufTimeout` increases. The average RTT depends on the hop distance of TCP path. The first two experiments have similar average RTT but the third experiment has a smaller average RTT with a smaller hop distance. In summary, we observed throughput gains from 20% to 70% in our experiments.

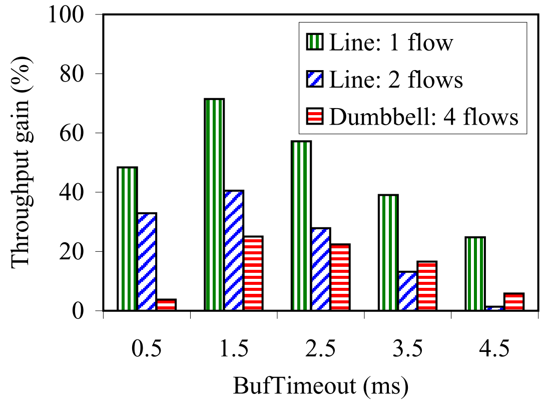
It is worth mentioning that, in our experiments, we noticed that coding ratio (CR) increases at every relay as `NCBufTimeout` increases. Also different relays experience different amount of coding ratio, and the behavior is different for line and dumbbell topologies. In line topology, those nodes that are closer to the source and destination, *i.e.*, nodes F and B, have higher coding ratios than the other nodes. The node in the middle, *i.e.*, node D, has the lowest CR (please refer to [14] for details). However, for the dumbbell topology, node D (the middle node) has higher CR than nodes C and E as `NCBufTimeout` increases. The reason is that nodes C and E can only encode the packets belonging to two TCP flows but node D can encode packets of all four TCP flows. Therefore, CR at a relay depends not only on the location of the relay but also on the number of flows that go through that relay.

C. Experiments on Unplanned Topology

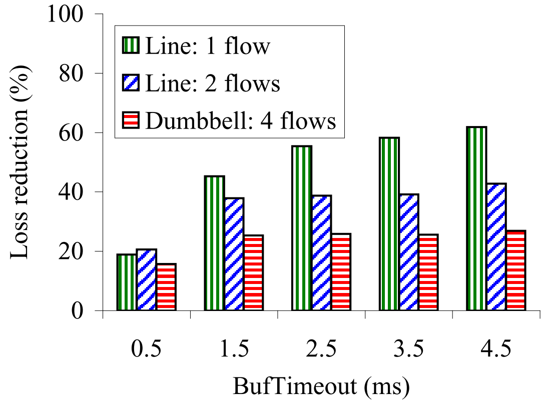
We implemented our network coding module in an unplanned mesh network with 20 nodes as depicted in Fig. 7. We are interested in throughput gain with different hop distances and different number of flows in this realistic topology.

1) Performance with Different Hop Distances: We arbitrarily select source and destination pairs from the network nodes with different hop distances $i = 2, \ldots, 5$. In order to highlight the impact of path length (hop distance) on coding ratio and TCP throughput, there is only one TCP flow in the network. Fig. 8 shows average throughput gain, loss reduction and RTT

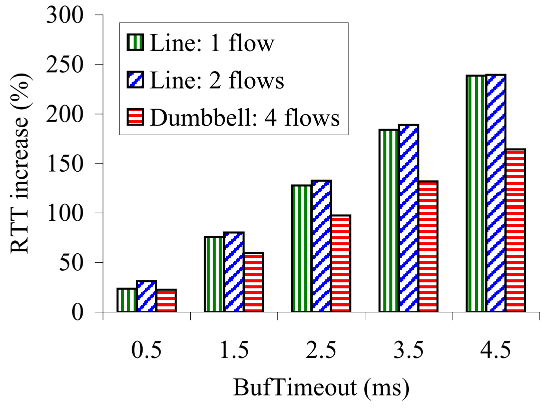
[4]The throughput computed in our experiments is based on the successfully received packets, and hence can be interpreted as the goodput.



(a) TCP throughput gain.



(b) End-to-end loss reduction.



(c) Round-trip-time increase.

Fig. 6. Experiments with multiple flows on planned topologies.

increase for different `NCBufTimeout` values when the hop distance changes from 2 to 5.

It is observed from Fig. 8(a) that there exist a `NCBufTimeout` value that maximizes TCP throughput gain. However, the proper value of `NCBufTimeout` may vary for different hop distances. The maximum throughput gain ranges from 40% to 85% with respect to path length. Figs. 8(b) and 8(c) illustrate the measurement results for loss reduction and RTT increase. Specifically, Fig. 8(b) shows that the average loss probability decreases as `NCBufTimeout` increases. The reason is that by increasing `NCBufTimeout` coding ratio increases as well which results in lower loss probability. Fig. 8(c) shows RTT increase with different hop distances. As expected, the average round-trip-time increases by increasing the path length and `NCBufTimeout`.

2) Performance with Multiple Flows: To investigate the impact of multiple flows, we choose 5 TCP flows with different hop distances as shown in Fig. 7. Table I shows the flows and their corresponding routes.

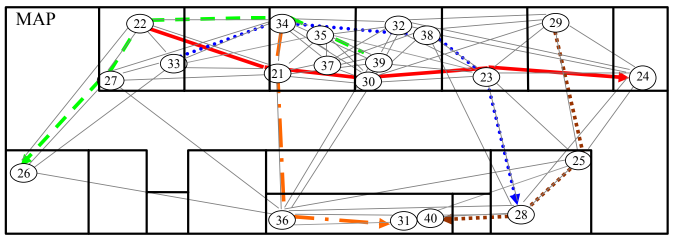


Fig. 7. Topology of the unplanned mesh network with 5 TCP flows.

TABLE I
ROUTING TABLE FOR FIG. 7.

Flow	Path
1	34–21–36–31
2	39–35–34–22–27–26
3	22–21–30–23–24
4	33–34–32–38–23–28
5	29–25–28–40

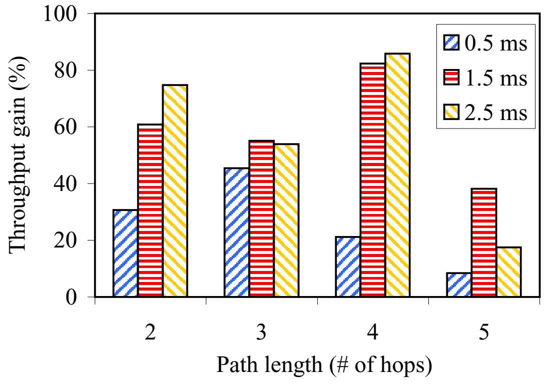
Table II presents average throughput gain, average packet loss probability and average round-trip-time with 5 concurrent flows. The maximum throughput gain is 22% when `NCBufimeout` is 0.5 ms. When `NCBufTimeout` is larger than 2.5 ms, some TCP connections are broken frequently. Interestingly, throughput gain is almost insensitive to the particular choice of `NCBufTimeout`. We conjecture that the insensitivity is due to the heterogeneity of flows (in terms of routes).

TABLE II
MULTIPLE FLOWS WITH DIFFERENT HOP DISTANCES.

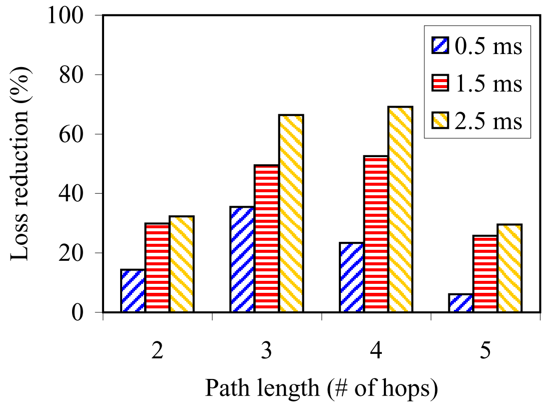
NCBufTimeout	Tput gain	Loss reduction	RTT increase
0.5 ms	22.34%	22.80%	3.64%
1.5 ms	21.26%	23.35%	7.68%
2.5 ms	21.23%	26.71%	26.47%

VI. CONCLUSIONS

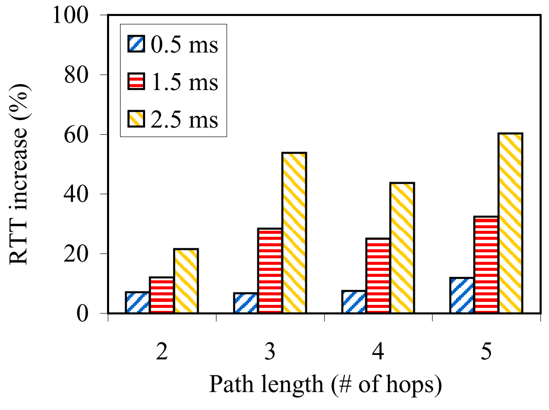
In this paper, we presented an implementation of network coding in a wireless mesh network and studied TCP throughput using measurements from our implementation. The distinguishing feature of our implementation is that it uses off-the-shelf hardware and software components and does not need any modifications to TCP or the MAC layer protocol while still being able to achieve significant throughput improvements compared to non-coding scenarios. We developed a simple model for TCP throughput in a coded wireless network, and



(a) TCP throughput gain.



(b) End-to-end loss reduction.



(c) Round-trip-time increase.

Fig. 8. Unplanned mesh topology: Experiments with different path length.

argued that the main factor improving TCP throughput is reduced packet loss thanks to network coding. To further improve TCP throughput and combat time synchronization issues, we implemented a timer at intermediate nodes to postpone the immediate transmission of packets. We then presented measurement results characterizing the relation between the timeout value and throughput improvement.

In the future, we hope to extend our analytical model to capture the impact of network coding on TCP throughput taking into consideration finer interactions between TCP and network coding. Such a model can be used to characterize the throughput benefit of network coding and study the effect of various control parameters such as the timeout value at intermediate nodes. We also plan to explore the use of multipath TCP in WMNs employing network coding.

VII. Acknowledgment

This work was supported in part by the National Science Foundation under award numbers CNS-0519922, CNS-0524323 and DMI-0330171 and in part by the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defense under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defense, or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

References

[1] R. W. Yeung and Z. Zhang, "Distributed source coding for satellite communications," *IEEE Trans. Inform. Theory*, vol. 45, no. 4, pp. 1111–1120, May 1999.

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

[3] Y. Wu, P. A. Chou, and S.-Y. Kung, "Information exchange in wireless networks with network coding and physical-layer broadcast," Microsoft Research, Tech. Rep. MSR-TR-2004-78, Aug. 2004.

[4] D. S. Lun, M. Medard, and M. Effros, "On coding for reliable communication over packet networks," in *Proc. Allerton*, Urbana Champaign, USA, Sept. 2004.

[5] S. Deb, M. Effros, T. Ho, D. R. Karger, R. Koetter, D. S. Lun, M. Medard, and N. Ratnakar, "Network coding for wireless applications: A brief tutorial," in *Proc. International Workshop on Wireless Ad-hoc Networks*, London, UK, May 2005.

[6] S. Katti, H. Rahul, W. Hu, D. Katabi, M. M. Médard, and J. Crowcroft, "XORs in the air: Practical wireless network coding," in *Proc. ACM SIGCOMM*, Pisa, Italy, Sept. 2006.

[7] C. Fragouli, J. Widmer, and J.-Y. L. Boudec, "On the benefits of network coding for wireless applications," in *Proc. WiOpt*, Boston, USA, Apr. 2006, pp. 1–6.

[8] J. Liu, D. Goeckel, and D. Towsley, "Bounds on the gain of network coding and broadcasting in wireless networks," in *Proc. IEEE INFOCOM*, Anchorage, USA, May 2007.

[9] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inform. Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[10] T. Ho, M. Medard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inform. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[11] A. A. Hamra, C. Barakat, and T. Turletti, "Network coding for wireless mesh networks: A case study," in *Proc. IEEE WoWMoM*, Buffalo, USA, June 2006, pp. 103–114.

[12] T. J. Ott, J. Kemperman, and M. Mathis, "The stationary distribution of ideal TCP congestion avoidance," in *Proc. DIMACS Workshop on Performance of Realtime Applications on the Internet*, South Plainfield, USA, Nov. 1996.

[13] Microsoft Research, Mesh Networking. [Online]. Available: http://research.microsoft.com/mesh/

[14] Y. Huang, M. Ghaderi, D. Towsley, and W. Gong, "TCP performance in coded wireless mesh networks," Electrical and Computer Engineering Department, University of Massachusetts Amherst, Tech. Rep. TR-07-CSE-02, Apr. 2007. [Online]. Available: http://tennis.ecs.umass.edu/~yhuang/papers/tcp-netcoded-techreport.pdf