MIT-CSAIL-TR-2006-009 February 16, 2006

Network Coding Made Practical

Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, and Jon Crowcroft

Network Coding Made Practical

Abstract

We propose a new architecture for wireless mesh networks. In addition to forwarding packets, routers mix (i.e., code) packets from different sources to increase the information content of each transmission. We show that intelligently mixing packets increases network throughput. Our design is rooted in the theory of network coding. In contrast to prior work on network coding, which is mainly theoretical and focuses on multicast traffic, ours is practical and solves the common case of unicast traffic. We present the first implementation of network coding in a wireless network. Our system introduces a coding layer between the IP and MAC layers. It works with UDP and TCP traffic, and hence seamlessly integrates with existing applications. We evaluate our design on a 34-node wireless testbed and show that it delivers a 3-4*x* increase in the throughput of wireless mesh networks.

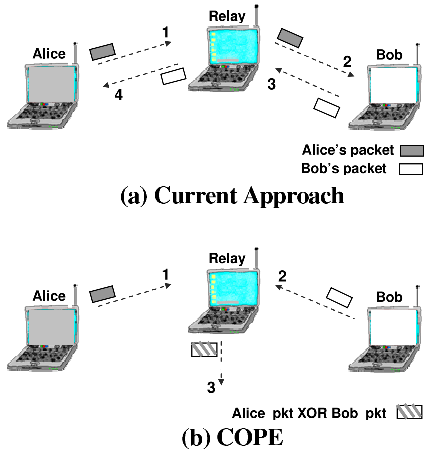


Figure 1—A simple example of how COPE reduces bandwidth consumption. It allows Alice and Bob to exchange a pair of packets using 3 transmissions instead of 4 (numbers on arrow refer to the order of transmission).

1 Introduction

It is clear that wireless will be the dominant medium of communication in the future. Current wireless implementations, however, struggle with an intrinsic limitation: bandwidth. For a dense large-scale wireless network to work properly, we need more efficient bandwidth usage.

This paper uses *network coding* [2] to address the bandwidth limitation of wireless networks. In contrast to the more traditional source coding, network coding allows routers to mix (i.e., code) information in packets from different sources. Coding at the router compresses the information whenever possible, and reduces the number of transmissions required to drain packets from a router's queue. A fewer number of transmissions translates directly to reduced bandwidth consumption and higher throughput. Indeed, it has been proven that, for multicast traffic, network coding maximizes the achievable throughput [2].

Our work shows how to transform network coding from an elegant theory to a practical system that provides several-fold improvement in the throughput of wireless mesh networks. Prior work on network coding is mostly theoretical and focuses on analytical tractability; it usually assumes multicast traffic, senders and receivers are known *a priori*, and traffic rates are known and smooth [21, 17, 19, 14, 15, 23, 10, 22]. Given these assumptions, it runs an optimization to determine how the routers should code the packets so as to minimize bandwidth consumption [24, 34]. In contrast, our approach works for the common case of multiple unicast flows. It is distributed, and makes no assumptions about senders, receivers, or traffic characteristics.

We present COPE, a new forwarding architecture. COPE inserts a coding shim between the IP and MAC layers, which identifies coding opportunities and benefits from them by forwarding multiple packets in a single transmission.

To give the reader a feel for how COPE works, we start with a fairly simple example. Consider the scenario in Fig. 1, where Alice and Bob want to exchange a pair of packets via a router. In current approaches, Alice sends her packet to the router, which forwards it to Bob, and Bob sends his packet to the router, which forwards it to Alice. This process requires 4 transmissions. Now consider a network coding approach. Alice and Bob send their respective packets to the router, which XORs the two packets and broadcasts the XOR-ed version. Alice and Bob can obtain each other's packet by XOR-ing again with their own packet. This process takes only 3 transmissions, reducing the required bandwidth, and producing a throughput gain of $\frac{4}{3} = 1.33$.

In fact, COPE leads to much larger bandwidth savings than are apparent from this example. COPE exploits the shared nature of the wireless medium which, for free, broadcasts each packet in a small neighborhood around its path. Each node stores the packets it overhears for a limited period. It also tells its neighbors which packets it has heard by annotating the packets it sends. This creates an environment conducive to coding because nodes in each area have a large and partially overlapping reservoir of packets they can use to decode. When a node transmits a packet, it uses its knowledge of what its neighbors have heard to perform *opportunistic coding*; the node XORs multiple packets and sends them in a single transmission if each intended nexthop has enough information to decode the encoded packet. This extends COPE beyond two flows that traverse the same nodes in reverse order (as in the illustrative example), and allows COPE to XOR more than a pair of packets.

In this paper, we make the following key contributions:

(1) We build the first system architecture for network coding, and plug it into the current network stack. Prior work on network coding has a strong theoretical flavor [2, 21, 19]. Some recent papers present simulation results [32, 10, 7]. The only prior implementation we know of is a simple prototype by our team, where a single node was made to XOR pairs of packets and broadcast them. In contrast, this paper presents a full-fledged implementation that is integrated into the Linux network stack, works with both TCP and UDP flows, and runs real applications.

(2) We report the results of the first field experiments of network coding in a wireless network. We run our implementation on a 34-node wireless mesh network. Our experiments reveal the following findings:

- COPE increases the throughput by 3-4x for an actual wireless network with a state-of-the-art routing protocol and realistic traffic patterns.
- For UDP traffic, the benefits of COPE stem from two factors. First, coding allows the routers to deliver more packets per transmission, which reduces bandwidth consumption and improves throughput. Second, coding reduces the router queue size, preventing a downstream router from dropping packets that have already consumed network resources. This second factor contributes as much to the performance of COPE as the reduction in the number of transmissions (§7.1).
- The interaction of TCP with coding is different from UDP's. TCP congestion control prevents a sender from sending too many packets that will be dropped at some downstream bottleneck. Thus, the throughput improvement observed with TCP is purely due to the reduction in the number of transmissions (§7.2).
- Overhearing (§3-a) and optimistically coding packets based on guessing (§4.1) almost doubles the throughput gain of COPE. Furthermore, COPE is effective in recovering from guessing errors.

For network coding, we believe that this paper is where the rubber meets the road. It demonstrates through actual implementation and field experiments that network coding provides important practical benefits.

2 Related Work

A rich body of systems research has tackled the problem of improving the throughput of wireless networks. The proposed solutions range from designing better routing metrics [9, 5, 11] to tweaking the TCP protocol [29], and include improved routing and MAC protocols [6, 18, 12]. Similar to this line of research, we focus on practical system design.

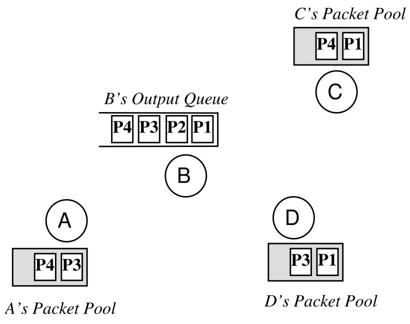
Term	Definition
Native Packet	A non-encoded packet
Encoded or XOR-ed Packet	A packet that is the XOR of multiple native packets
Nexthops of an Encoded Packet	The set of nexthops for the native packets XOR-ed together to generate the encoded packet
Packet Id	A 32-bit hash of the packet's IP source address and IP sequence number
Output Queue	A FIFO queue at each node, where it keeps the packets it needs to forward
Packet Pool	A buffer where a node stores all packets heard in the past T seconds
Throughput Gain	The ratio of the network throughput with COPE to the throughput without COPE

Table 1—Definitions of terms used in the paper.

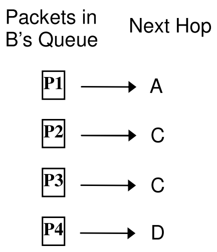
However, our work addresses the throughput issue from a fundamentally different perspective. By adopting network coding and integrating it into the current network stack, we propose an architectural extension to the current designs of wireless mesh networks. In particular, we demonstrate an approach that reshapes the theories of network coding to deliver a practical and high-throughput implementation of wireless mesh networks. Furthermore, our approach is complementary to *all* of the protocol enhancements above.

Recent years have seen a substantial advancement in the theory of network coding. Ahlswede et al. started the field with their pioneering paper [2], which shows that having intermediate nodes in the network mix information from different flows increases the throughput and allows the communication to achieve broadcast capacity. This was soon followed by the work of Li et al., who showed that, for the multicast case, linear codes are sufficient to achieve the maximum flow bounds [21]. Koetter and Médard [19] presented polynomial time algorithms for encoding and decoding, and Ho et al. extended these results to random codes [14]. Some recent work has studied network coding in the wireless environment [10, 27]. In particular, Lun et al. have studied network coding in the presence of omni-directional antennae and shown that the problem of minimizing the communication cost can be formulated as a linear program and solved in a distributed manner [24]. All of this work, however, is mainly theoretical, and assumes multicast traffic, known sender and receivers, as well as smooth traffic. In contrast, this paper focuses on a practical implementation and measured throughput improvements.

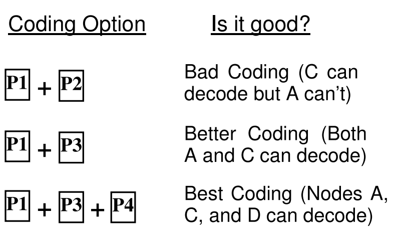
Additionally, our approach addresses network coding of multiple unicast flows. There is little prior work on network coding for the important problem of unicast communication, and most theoretical results are negative [35]. It is known that for a single unicast session, coding does not provide a gain over pure forwarding. The case for multiple unicast sessions is largely unknown territory. A few papers focus on specific scenarios showing that, for the stud-



(a) B can code packets it wants to send



(b) Nexthops of packets in B's queue


Coding Option
Is it good?
P1 + P2
Bad Coding (C can decode but A can't)
P1 + P3
Better Coding (Both A and C can decode)
P1 + P3 + P4
Best Coding (Nodes A, C, and D can decode)

(c) Possible coding options

Figure 2—**Example of Opportunistic Coding; Node B has 4 packets in its queue, whose nexthops are listed in (b). Each neighbor of B has stored some packets as depicted in (a). Node B can make a number of coding decisions (as shown in (c)), but should select the last one because it maximizes the number of packets delivered in a single transmission.**

ied scenario, network coding results in better throughput than pure forwarding [36, 13, 33]. In contrast, our work provides a working protocol that supports multiple unicast sessions. We show through an actual implementation that our approach increases the throughput of a wireless network and outperforms current designs. These practical gains generate optimism about finding theoretical bounds.

3 COPE OVERVIEW

COPE is a new forwarding architecture. It inserts a coding shim between the IP and MAC layers, which detects coding opportunities and exploits them to forward multiple packets in a single transmission. Before delving into the details, we refer the reader to Table 1, which defines the terms used in the rest of the paper.

COPE's current design targets stationary multi-hop wireless networks, such as Roofnet and community wireless networks [1, 28, 4]. Nodes in such networks are normally not resource-constrained. Therefore, in this paper we assume that the wireless nodes are not limited by memory or processing power. Nevertheless, this assumption is not stringent, and our requirements are easily satisfied by today's mobile devices.

COPE uses three main techniques to achieve its performance improvements:

(a) Overhearing: Wireless is a broadcast medium, creating many opportunities for nodes to overhear packets, when equipped with omni-directional antennae. COPE sets the nodes in promiscuous mode, makes them snoop on all communications over the wireless medium, and stores the overheard packets for a limited interval, T. The value of T should be larger than the maximum packet latency (the default is $T = 0.5s$). In addition, each node broadcasts *reception reports* to tell its neighbors which packets it has stored. Reception reports are sent by annotating the data packets the node transmits. A node that has no data packets to transmit periodically sends the reception reports in special control packets.

(b) Opportunistic Coding: The key question, however, is what packets to code together to maximize throughput. A node may have multiple coding options, but its final decision should strive to *maximize the number of native packets delivered in a single transmission, while ensuring that each of the intended nexthops has enough information to decode its native packet.*

The above is best illustrated with an example. In Fig. 2-a, node B has 4 packets in its output queue p_1, p_2, p_3, and p_4. Its neighbors have overheard some of these packets. The table in Fig 2-b shows the nexthop of each packet in B's queue. When the MAC permits B to transmit, B takes packet p_1 from the head of the queue. Assuming that B knows which packets each neighbor has, it has a few coding options as shown in Fig. 2-c. It could send $p_1 \oplus p_2$. Since node C has p_1 in store, it could XOR p_1 with $p_1 \oplus p_2$ to obtain the native packet sent to it, i.e., p_2. However, node A does not have p_2, and so cannot decode the XOR-ed packet. Thus, sending $p_1 \oplus p_2$ would be a bad coding decision for B, because only one neighbor can benefit from this transmission. The second option in Fig. 2-c shows a better coding decision for B. Sending $p_1 \oplus p_3$ would allow both neighbors C and A to decode and obtain their intended packets from a single transmission. Yet the best coding decision for B would be to send $p_1 \oplus p_3 \oplus p_4$, which would allow all three neighbors to receive their respective packets all at once.

The above example demonstrates a simple rule for making coding decisions.

> To transmit n packets, $p_1, ..., p_n$, to n nexthops, $r_1, ..., r_n$, a node can XOR the n packets together only if each nexthop r_i has all $n-1$ packets p_j for $j \neq i$.

This rule ensures that each nexthop can decode the XOR-ed version to extract its native packet. Whenever a node has a chance to transmit a packet, it chooses the largest n that satisfies the above rule to maximize the benefit of coding.

(c) Learning Neighbor's State: But how does a node know what packets its neighbors have? As explained earlier, each node announces to its neighbors the packets it stores in reception reports. However, this may be insufficient. At times of severe congestion, reception reports may get lost in collisions, while at times of light traffic, they may arrive too late, after the node has already made a suboptimal coding decision. Therefore, a node cannot rely solely on reception reports and may need to guess whether a neighbor has a particular packet.

To guess intelligently, we leverage the routing computation. State-of-the-art wireless routing protocols compute the delivery probability between any pair of nodes and use it to identify good paths. For example, the ETX metric [9] periodically computes the fraction of packets delivered between any two nodes in the network. It assigns each link a weight equal to *1/(delivery probability)*. These weights are broadcast to all nodes in the network and used by a link-state routing protocol to compute shortest paths. We leverage these probabilities for guessing. In the absence of deterministic information, COPE estimates the probability that a particular neighbor has a packet as the delivery probability of the link between the packet's previous hop and the neighbor.

Allowing these informed guesses enables a node to make smart encoding decisions to maximize the benefit of coding. The occasional incorrect guess, however, would cause the coded packet to be undecodable at some nexthop. In this case, the relevant native packet will have to be retransmitted, potentially encoded with a new set of native packets.

4 MAKING IT WORK

In order to integrate COPE effectively within the current network stack, we need to address some important system issues.

4.1 Packet Coding Algorithm

To build the coding scheme, we have to make a few design decisions. First, we design our coding scheme around the principle of *not adding extra delay to packets*. When the wireless channel is available, the node takes the packet at the head of its output queue, checks which other packets in the queue may be encoded with this packet, XORs those packets together, and broadcasts the XOR-ed version. If the MAC is ready and there are no encoding opportunities, our node does not wait for the arrival of a matching codable packet. Compared with the current approach, therefore, COPE lets the node opportunistically overload each transmission with additional information when possible, but does not wait for additional codable packets to arrive.

Second, COPE *gives preference to XOR-ing packets of similar lengths*, because XOR-ing small packets with larger ones reduces bandwidth savings. Empirical studies of the packet size distribution in the Internet show that the distribution is bimodal with peaks at 40 bytes and 1500 bytes [25]. We can therefore limit the overhead of searching for packets with the right sizes by distinguishing between small and large packets. We might still have to XOR packets of different sizes. In this case, the shorter packets are padded with zeroes. The receiving node can easily remove the padding by checking the packet size field in the IP header of each native packet.

Third, notice that COPE will *never code together packets headed to the same nexthop*, since the nexthop will not be able to decode these packets. Hence, when searching for codable packets, we only need to consider packets headed to different nexthops. COPE therefore maintains two virtual queues per neighbor; one for packets smaller than 100 bytes (usually TCP ACKs), and another for packets larger than 100 bytes. When a new packet is added to the output queue, an entry is added to the appropriate virtual queue based on the packet's nexthop and size.

Searching for appropriate packets to code is efficient due to the maintenance of virtual queues. When making coding decisions, COPE first dequeues the packet at the head of the FIFO output queue, and determines if it is a small or a large packet. Depending on the size, it looks at the appropriate set of virtual queues. For example, if there is a small packet at the head of the queue, COPE first looks at the virtual queues for small packets. COPE looks only at the heads of the virtual queues corresponding to the different neighbors to limit packet reordering. After exhausting the virtual queues of a particular size, the algorithm then looks at the heads of virtual queues for packets of the other size. Thus for finding appropriate packets to code COPE has to look at $2M$ packets in the worst case, where M is the number of neighbors of a node.

Another concern is *packet reordering*. We would like to limit reordering packets from the same flow because TCP mistakes excessive reordering as a congestion signal. Thus, we always consider packets according to their order in the output queue. Still, reordering may occur because we prefer to code packets of the same size. Note, however, that reordering is quite limited because most TCP flows send data in one direction and acks in the opposite, with data packets usually being larger than 100 bytes. Thus, our two virtual queues for each neighbor end up mainly separating ack streams from data streams. It is also possible to restrict reordering even further. Since the number of flows in a wireless network is relatively small, we could hash a TCP flow to a particular queue and pin this map for the duration of the flow. We will see in §4.4, however, that reordering might arise from reasons other than queuing, particularly the need to retransmit a packet that has been lost due to a mistake in guessing what a neighbor can decode. Thus, we choose to deal with any reordering that might happen inside the network at the receiver. COPE has a module that puts TCP packets in order before delivering them to the transport layer as explained in §4.5.

Finally, we want to ensure that each neighbor to whom a packet is headed has a high probability of being able to decode its native packet. Thus, for each packet in its output queue, our relay node estimates the probability that each of its neighbors has already heard the packet. Sometimes the node can be certain about the answer, for example, when the neighbor is the previous hop of the packet, or when the reception reports from the neighbor state so. When neither of the above is true, the node leverages the delivery probabilities computed by the routing protocol; it estimates the probability the neighbor has the packet as the delivery probability between the packet's previous hop and that neighbor. The node uses this estimate to ensure that encoded packets are decodable by all of their nexthops with high probability.

In particular, suppose the node encodes n packets together. Let the probability that a nexthop has heard packet i be P_i. Then, the probability, P_D, that it can decode its native packet is equal to the probability that it has heard all of the $n-1$ native packets XOR-ed with its own, i.e.,

$$P_D = P_1 \times P_2 \times \ldots \times P_{n-1}.$$

Consider an intermediate step while searching for coding candidates. We have already decided to XOR $n-1$ packets together, and are considering XOR-ing the n^{th} packet with them. The coding algorithm now checks that, for each of the n nexthops, the decoding probability P_D, after XOR-ing the n^{th} packet with the rest stays greater than a threshold G (the default value $G = 0.8$). Checking that the above conditions are met for every nexthop ensures that each nexthop can decode its packet with at least probability G. Finally, we note that for fairness we iterate over the set of neighbors according to a random permutation.

Formally, each node in the network maintains the following data structures.

- Each node has a FIFO queue of packets to be forwarded, which we call *the output queue*.
- For each neighbor, the node maintains two *per-neighbor virtual queues*, one for packets smaller than 100 bytes, and the other for larger packets. The virtual queues for a neighbor A contains pointers to the packets in the output queue whose nexthop is neighbor A.
- Additionally, the node keeps a hash table, *packet info*, that is keyed on packet-id. For each packet in the output queue, the table indicates the probability of each neighbor having that packet.

Whenever the MAC signals a sending opportunity, the node executes the procedure illustrated in Alg. 1.

1 Coding Procedure

```
Pick packet $p$ at the head of the output queue.
Natives = $\{p\}$
NextHops = $\{nexthop(p)\}$
if $size(p) > 100$ bytes then
    which_queue = 1
else
    which_queue = 0
end if
for Neighbor $i = 1$ to $M$ do
    Pick packet $p_i$, the head of virtual queue $Q(i, which\_queue)$
    if $\forall n \in$ Nexthop $\cup\{i\}$, Pr[can decode $p \oplus p_i$] $\geq G$ then
        $p = p \oplus p_i$
        Natives = Natives $\cup\{p_i\}$
        NextHops = NextHops $\cup\{i\}$
    end if
end for
which_queue = !which_queue
for Neighbor $i = 1$ to $M$ do
    Pick packet $p_i$, the head of virtual queue $Q(i, which\_queue)$
    if $\forall n \in$ Nexthop $\cup\{i\}$, Pr[can decode $p \oplus p_i$] $\geq G$ then
        $p = p \oplus p_i$
        Natives = Natives $\cup\{p_i\}$
        NextHops = NextHops $\cup\{i\}$
    end if
end for
return $p$
```

4.2 Packet Decoding Algorithm

Packet decoding is simple. Each node maintains a *Packet Pool*, in which it keeps a copy of each native packet it has received or sent out. The packets are stored in a hash table keyed on packet id, and the table is garbage collected every few seconds. When a node receives an encoded packet consisting of n native packets, the node goes through the ids of the native packets one by one, and retrieves the corresponding packet from its packet pool if possible. Ultimately, it XORs the $n-1$ packets with the received encoded packet to retrieve the native packet meant for it.

4.3 Pseudo-broadcast

Since COPE broadcasts encoded packets to their next hops, the natural approach to build COPE would be to use 802.11 broadcast. Unfortunately, this does not work. In particular, the 802.11 MAC has two modes: unicast and broadcast. 802.11 unicast packets are immediately ack-ed by their intended nexthops. The MAC interprets the lack of an ack as a collision signal, to which it reacts by backing off exponentially, thereby allowing multiple nodes to share the medium. In contrast, an 802.11 broadcast packet has many intended receivers, and it is unclear who should ack. Thus, 802.11 broadcast packets are not ack-ed. This means that a broadcast source cannot detect collisions, and thus does not back off. If multiple backlogged nodes share the broadcast channel, and each of them continues sending at the highest rate, ignoring the others, the resulting throughput would be very poor, due to high collision rates. Ideally one would design a backoff scheme suitable for broadcast channels (e.g., Idle Sense [12]), but we are interested in an implementation of COPE that can be deployed in the near future using off-the-shelf 802.11 products.

Moreover, 802.11 broadcast is not reliable. 802.11 unicast ensures reliability by retransmitting the packet at the MAC layer for a fixed number of times until a synchronous ack is received. In the absence of these synchronous acks, the broadcast mode offers no retransmissions and consequently very low reliability.

Our solution is *pseudo-broadcast*, which piggybacks on

802.11 unicast and benefits from its reliability and backoff mechanism. Pseudo-broadcast unicasts packets that are meant for broadcast. The link layer destination field is set to the MAC address of one of the intended recipients. An XOR-header is added after the link-layer header, listing all nexthops of the packet, Since all nodes are set in the promiscuous mode, they can overhear packets not addressed to them. When a node receives a packet with a MAC address different from its own, it checks the XOR-header to learn whether it is a nexthop. If so, it processes the packet further, else it stores the packet in a buffer as an opportunistically received packet. As all packets are sent using 802.11 unicast, the MAC can detect collisions and backoff properly.

Pseudo-broadcast is also more reliable than simple broadcast. The packet is retransmitted multiple times until its designated MAC receiver receives the packet and acks it synchronously, or the number of retries is exceeded. A desirable side effect of these retransmissions is that nodes which are promiscuously listening to this packet have more opportunities to hear it.

4.4 Hop-by-hop ACKs and Retransmissions

(a) Why hop-by-hop acks? Encoded packets require all nexthops to acknowledge the receipt of the associated native packet for two reasons. First, COPE may optimistically guess that a nexthop has enough information to decode an XOR-ed packet, when it actually does not. To recover from such errors, the sender expects the nexthops of an XOR-ed packet to decode the XOR-ed packet, obtain their native packet, and ack it. Second, encoded packets are headed to multiple nexthops, but the sender gets synchronous MAC layer acks only from the nexthop that is set as the link layer destination of the packet. There is a higher probability of loss to the other nexthops from whom it does not get synchronous acks. The standard solution to this problem is to mask these error-induced drops by recovering lost packets locally through acknowledgments and retransmissions [3, 16].

If any of the native packets is not ack-ed within a certain interval, the packet is retransmitted, potentially encoded with another set of native packets.

(b) Asynchronous Acks and Retransmissions: How should we implement these hop-by-hop acks? For non-coded packets, we do not need to do anything; we leverage the 802.11 synchronous acks.

Unfortunately, extending this synchronous ack approach to coded packets is highly inefficient, as the overhead incurred from sending each ack in its own packet with the necessary IP and WiFi headers would be excessive. Thus, in COPE encoded packets are ack-ed asynchronously.

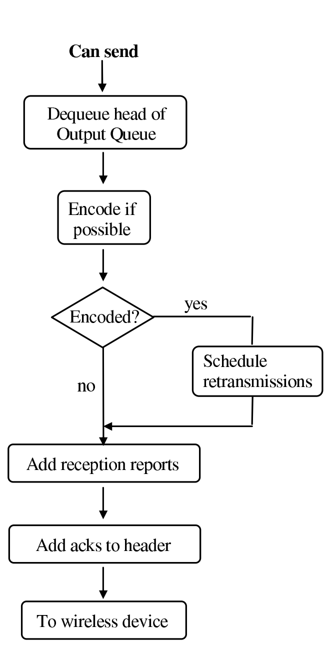
Every node maintains the following data structures:

- Ack Queue: A queue of pending acknowledgments.
- Control Packet Timer: when the node has no data packets to send, this timer fires periodically to trigger the transmission of a small control packet that carries pending reception reports and acks.
- Retransmission Event List: an ordered list of pending retransmissions. Each entry contains a pointer to a packet in the Packet Pool as well as the due time of the next retransmission and the number of attempted retransmissions.
- Retransmission Timer: a timer scheduled to fire at the next retransmission event. When it fires, the native packet due for retransmission is inserted at the head of the output. The number of attempted retransmissions is incremented. If it exceeds a threshold (default is 2 times), the retransmission event is deleted from the Retransmission Event List. Otherwise the retransmission event is inserted back in the Event List according to its new due time. Finally the timer is rescheduled to fire at the next retransmission event.

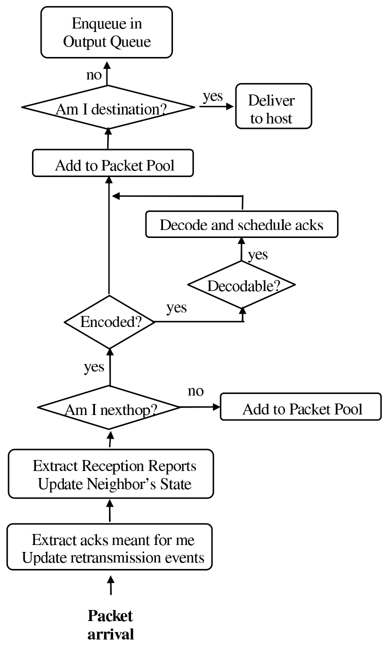
When a node sends an encoded packet, it schedules a retransmission event for each of the native packets in the encoded packet. If any of these packets is not ack-ed within T_a seconds, the packet is inserted at the head of the output queue and retransmitted at the next sending opportunity. (T_a is initialized to be slightly larger than the round trip time of a single link, and is adapted using an exponentially decaying average, similarly to TCP's RTT estimate.) Similarly to fresh data, retransmitted packets may get encoded with other packets according to the scheme in §4.1. A nexthop that receives an encoded packet decodes it to obtain its native packet, and immediately pushes an ack event into the Ack Queue. COPE sends acks as annotations on data packets. Thus, before sending a packet, the node checks its Ack Queue and incorporates the pending acks in the ack information in the COPE header. If the node has no data packets to transmit, it sends the acks in periodic control packets—the same control packets used to send reception reports.

4.5 Preventing TCP Packet Reordering

The reader might have noticed that asynchronous acks can cause packet reordering, which may be confused by TCP as a sign of congestion. To deal with this issue, COPE has an *ordering agent*, which ensures that TCP packets are delivered to the transport protocol in order. The agent ignores all packets whose final IP destinations differ from the current node, as well as non-TCP packets. These packets are immediately passed to the next processing stage. For each TCP flow ending at the host, the agent maintains a packet buffer and records the last TCP sequence number passed on to the network stack. Incoming packets that do not produce a hole in the TCP sequence stream are immediately dispatched to the transport layer, after updating the sequence number state. Otherwise, they are withheld in the buffer till the gap in the sequence numbers is filled, or until a timer expires.



(a) Sender side



(b) Receiver side

Figure 4—Flow chart for our COPE Implementation.

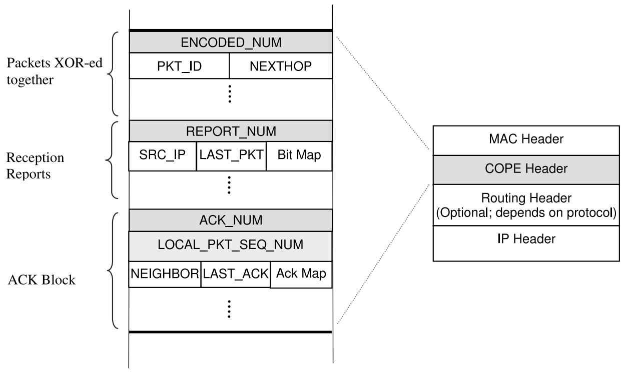


Figure 3—COPE Header. The first block identifies the native packets XOR-ed and their nexthops. The second block contains reception reports. Each report identifies a source, the last IP sequence number received from that source, and a bit-map of most recent packets seen from that source. The third block contains asynchronous acks. Each entry identifies a neighbor, an end point for the ACK map, and a bit-map of ack-ed packets.

5 Implementation Details

COPE adds special packet headers and alters the control flow of the router to encode and decode packets. We describe both these parts in the following sections.

5.1 Packet Format

COPE inserts a variable-length coding header in each packet, as shown in Fig. 3. If the routing protocol has its own header (e.g., Srcr [5]), COPE's header sits between the routing and the MAC headers. Otherwise, it sits between the MAC and IP headers. Only the shaded fields in Fig. 3 are required in every COPE header. The coding header contains 3 blocks. The first block records meta-data to enable packet decoding. It starts with `ENCODED_NUM`, the number of native packets XOR-ed together. For each native packet, the header lists its `ID`, which is a 32-bit hash of the packet's source IP address and IP sequence number. This is followed by the MAC address of the native packet's `NextHop`. When a node hears an XOR-ed packet, it checks the list of `NextHops` to determine whether it is an intended recipient for any of the native packets XOR-ed together, in which case it decodes the packet, and processes it further according to the IP header.

(a) Reception reports: Each node broadcasts *reception reports* to tell its neighbors which packets it has stored. Reception reports constitute the second block in the XOR header, as shown in Fig. 3. The block starts with the number of the reports in the packet, `REPORT_NUM`. Each report specifies the IP source of the reported packets `SRC IP`. This is followed by the IP sequence number of the last packet heard from that source `Last PKT`, and a bit-map of recently heard packets. For example, a report of the form {`128.0.1.9, 50, 10000001`} means that the last packet this node has heard from source `128.0.1.9` is packet 50, and the node has also heard packets 42 and 49 from that source but none of the packets in between. The above representation for reception reports has two advantages: compactness and effectiveness. In particular, the bit-map allows the nodes to report each packet multiple times with minimal overhead. This guards against reception reports being dropped at high congestion.

(b) Expressing asynchronous acks compactly and robustly: One option is to list the ids of the ack-ed packets in a special section in the XOR Header attached to each packet, but this will require a 32-bit field for every ack-ed packet. More importantly, since congestion increases the probability of an ack being lost along with its carrier data packet, acks have to be repeated on multiple data packets. To ensure robust ack delivery with minimum overhead, we use cumulative acks. Each node maintains a per-neighbor 16-bit counter, called `Neighbor_Seqno_Counter`. Whenever the node sends a packet to that neighbor, the counter is incremented and its value is assigned to the packet as a local sequence number, `Local_PKT_SEQ_NUM`. The two neighbors use this sequence number to identify the packet. Now, a node can use cumulative acks on a per-neighbor basis. Each coded packet contains an ack block in its header as shown in Fig. 3. The ack block starts with the number of ack entries, followed by the packet local sequence number. Each ack entry starts with a neighbor MAC address. This is followed by a pointer to tell the neighbor where the cumulative acks stop, and a bit-map indicating previously received and missing packets. For example, an entry of {`A, 50, 01111111`} acks packet 50, as well as the sequence 43-49, from neighbor *A*. It also shows that packet 42 is still missing. Note that though we use cumulative acks, we do not ensure reliability at link layer. In particular, each node retransmits a lost packet a few times (default is 2), and then gives up.

5.2 Control Flow

Fig. 4 abstracts the architecture of COPE. On the sending side (Fig. 4-a), whenever the MAC signals an opportunity to send, the node takes the packet at the head of its output queue and hands it to the coding module (§4.1). As described earlier, choosing the set of packets to code and the actual XOR-ing of packets is very efficient. If the node can encode multiple native packets in a single XOR-ed version, it has to schedule asynchronous retransmissions. Either way, before the packet can leave the node, pending reception reports and acks are added.

On the receiving side (Fig. 4-b), when a packet arrives, the node extracts any acks sent by this neighbor to the node. It also extracts all reception reports and updates its view of what packets its neighbor stores. Further processing depends on whether the packet is intended for the node or it is just an overheard packet. If the node is not a next hop for the packet, the packet is stored in the Packet Pool. If the node is a next hop, it then checks if the packet is encoded. If it is, the node tries to decode by XOR-ing the encoded packet with the native packets it stores in its Packet Pool. After decoding it acks this reception to the previous hop and stores the decoded packet in the Packet Pool. The node now checks if it is the ultimate destination of the packet, if so it hands the packet off to the higher layers of the network stack. If the node is an intermediate hop, it pushes the packet to the output queue. If the received packet is not encoded, the packet is simply stored in the Packet Pool and processed in the same fashion as a decoded packet.

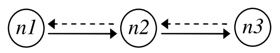
6 Testbed

(a) Characteristics: We have a 34-node wireless testbed that spans two floors in our building connected via an open lounge. Paths between nodes are between 1 and 6 hops in length, and the loss rates of links on these paths ranges between 0 and 30%. The experiments described in this paper run on 802.11a with a bit-rate of 6Mb/s, because it provides more controllable environments than 802.11b. Running the whole testbed on 802.11b suffers from a high level of interference from the local wireless networks. We have, however, run the experiments in §7.1, §7.2, and §7.3 over 802.11b with various static and dynamic bit-rates. The results are qualitatively similar.

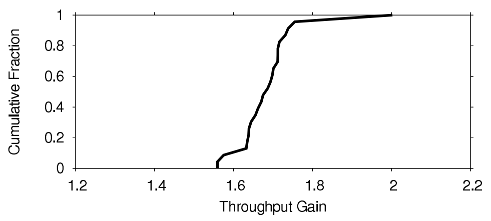
(b) Software: Nodes in the testbed run Linux, a routing protocol and COPE implemented using the Click toolkit [20]. Our implementation runs as a user space daemon on Linux, which sends and receives raw 802.11 frames from the wireless device using a libpcap-like interface. The implementation exports a network interface to the user that can be treated like any other network device (e.g., `eth0`). Applications interact with the daemon as they would with a standard network device provided by the Linux kernel. No modifications to the applications are therefore necessary. The implementation is agnostic to upper and lower layer protocols, and can be used by various protocols including UDP and TCP.

(c) Routing: Our testbed nodes run the Srcr implementation [5], a state-of-the-art routing protocol for wireless mesh networks. The protocol uses Djikstra's shortest path algorithm on a database of link weights. The weights are assigned based on the ETT metric [5], which is an estimate of the average time taken to successfully transmit a 1500-byte packet on that link, including the expected number of MAC level retransmissions. The protocol also source-routes the packets to avoid routing loops when link metrics change.
(d) Hardware: Each node in the testbed is a PC equipped with an 802.11 wireless card attached to an omni-directional antenna. The cards are based on the NETGEAR 2.4 & 5 GHz 802.11a/g chipset. They transmit at a power level of 15 dBm, and operate in the 802.11 ad hoc mode, with RTS/CTS disabled.

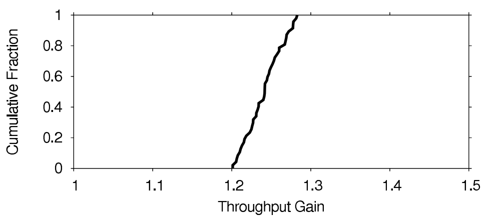
(e) Traffic Model: For the UDP experiments, we generate traffic in accordance with empirical studies of Internet traffic [26, 8]. In particular, our flows have Poisson arrivals, and a Pareto file size with the shape parameter set to 1.17. They are generated using a utility program called `UDPgen` [31]. As for TCP, we use `ttcp` [30] to generate long lasting flows, to study the interaction of TCP's congestion control with COPE.



(a) Alice-and-Bob Topology

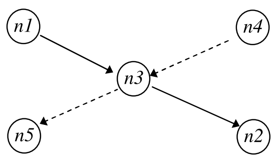


(b) CDF of throughput gain with UDP

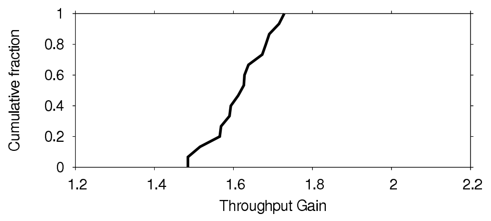


(c) CDF of throughput gain with TCP

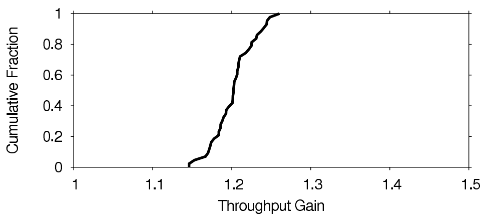
Figure 5—Throughput gains obtained with COPE, for the Alice-and-Bob topology. The arrows show the direction of the two flows. The figure shows TCP achieves the predicted coding gain, while UDP achieves higher gains because COPE compensates for the lack of congestion control.



(a) "X" Topology



(b) CDF of throughput gain with UDP



(c) CDF of throughput gain with TCP

Figure 6—Throughput gains obtained with COPE, for the "X" topology. The arrows show the direction of the two flows. The figure shows that because of overhearing and guessing, COPE can produce significant throughput gains even when the flows do not traverse reverse paths.

7 Experimental Results

In this section, we compare the performance of COPE with that of current approaches. We define three metrics used throughout the evaluation.

- *Network Throughput:* the total end-to-end throughput, i.e., the sum of the throughput of all flows in the network as seen by their corresponding applications.
- *Throughput gain:* the ratio of the network throughput with COPE to the network throughput without it. We compute the throughput gain from two consecutive experiments, with coding turned on, then off.
- *Coding gain:* the number of transmissions required by the current approach to deliver a packet from each flow to its destination divided by the number of transmissions used by COPE to deliver the same set of packets. For example, in the Alice-and-Bob scenario presented in §1, coding reduces the number of transmissions from 4 to 3, thus producing a coding gain of $\frac{4}{3} = 1.33$.

7.1 When and Why COPE Wins

We would like to identify the parameters that affect COPE's performance. Therefore, we start by looking at a few toy topologies. We revisit the Alice-and-Bob scenario presented in §1, and reproduced in Fig. 5-a. Two nodes n_1 and n_3 communicate via a router. In current approaches, exchanging two packets between the two nodes requires four transmissions. In COPE, however, the same two packets can be delivered using three transmissions: n_1 and n_3 transmit to n_2, which XORs the two packets and broadcasts the XOR-ed version. Thus, the coding gain of this scenario is $\frac{4}{3} = 1.33$.

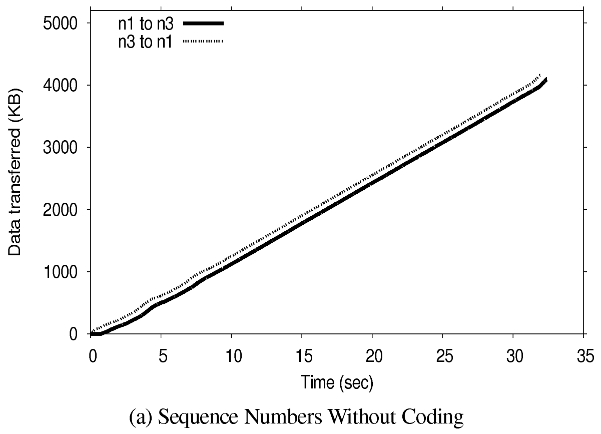
In practice, how close is the throughput gain to the coding gain? For UDP, surprisingly, the answer is that the throughput gain is much higher than the coding gain. Fig. 5-b shows the CDF of the throughput gain for 40 experiments taken over different 2-hop paths in our testbed. The figure shows that the throughput gain is around 1.7, which is significantly higher than the coding gain.

This high UDP gain is due to the interaction between COPE and the bandwidth allocation policy of the 802.11 MAC. The MAC tries to divide the bandwidth equally between the nodes. Without coding, however, the relay, n_2, needs to forward twice as many packets as the edge nodes. COPE allows the bottleneck, n_2, to XOR pairs of packets and drain them twice as fast, doubling the throughput of the network. In practice, it is unlikely that the relay will have coding opportunities for every packet. Further, the WiFi, the Srcr, and the IP headers are not coded. As a result, the observed throughput gain is about 1.7, as shown in Fig. 5-b.

To summarize, for UDP traffic, the benefits of COPE stem from two factors. First, coding allows the routers to deliver more packets per transmission, which reduces bandwidth consumption and improves throughput. Second, COPE alleviates the mismatch between the congestion level at a node and the rate at which the MAC allows it to send. In fact, the larger the number of flows traversing the bottleneck, the more opportunities it has to combine packets together in a single transmission. This second factor contributes as much to the performance of COPE as the reduction in the number of transmissions.

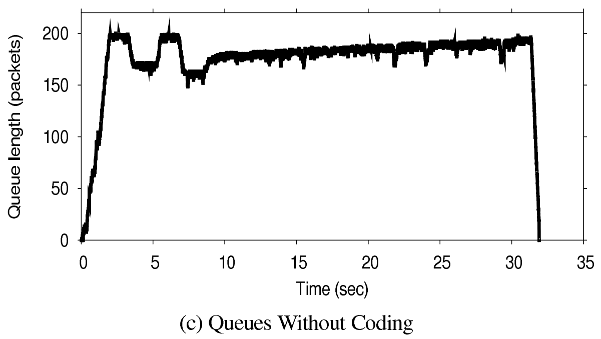
7.2 What About TCP?

Now, let us repeat the same experiments with TCP. One main difference between TCP and UDP is congestion control; we are interested in how this congestion control protocol interacts with coding. When we run UDP with COPE,



(a) Sequence Numbers Without Coding

(b) Sequence Numbers With COPE and 2 Retransmissions



(c) Queues Without Coding

(d) Queues With COPE and 2 Retransmissions

Figure 7—TCP sequence numbers and queue sizes at the bottleneck router for a typical run of the Alice-and-Bob topology.

the coding helps in matching the bottleneck's drain rate to its input rate. This prevents the bottleneck from dropping packets that have already consumed network resources, and consequently increases the throughput. But with TCP, when multiple flows get bottlenecked at the same router, as in Figs. 5, the senders back off and prevent excessive drops due to queue overflow. With or without coding, the congestion control protocol naturally matches the input rate at the bottleneck to its capacity, and prevents the excessive drops that we saw with UDP. Thus, with TCP, the throughput gain is primarily from the reduction in the number of transmissions in the network, i.e., the coding gain.

Fig. 5-c shows the CDF of the throughput gain for the Alice-and-Bob topology in Fig. 5, but with TCP. Clearly, this value is lower than the corresponding gain with UDP, but close to the theoretical coding gain of 1.33 derived earlier. We examine this case in greater detail by looking at the time plots of a typical TCP run, illustrated in Fig. 7. The figure shows the sequence number plots of the two TCP flows, as well as the queue sizes at the router, n_2. The throughput gains provided by coding are shared fairly across both flows. Further, both flows ramp up steadily, demonstrating the effectiveness of our asynchronous retransmission scheme in masking any residual wireless losses. Additionally, coding allows the router, n_2, to drain packets at the same rate as they arrive from the edge nodes, thereby limiting queue buildup at the middle node. While the queue size without coding is almost as high as the buffer size, the queue under COPE

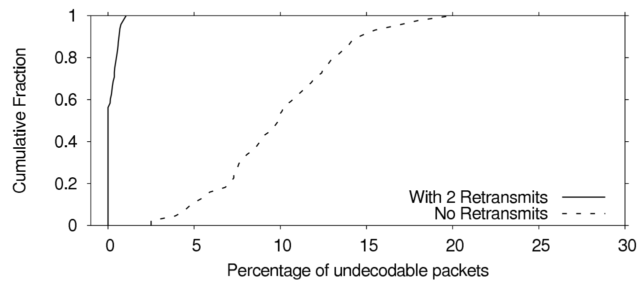
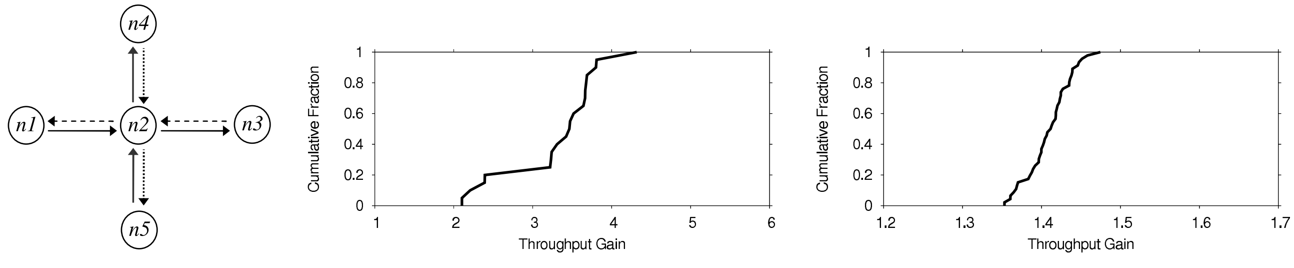


Figure 8—CDF of the percentage of undecodable packets, with and without retransmissions. The fi gure shows that the combination of acks and retransmissions allows COPE to recover effectively from bad coding guesses.

remains fairly small.

7.3 Overhearing & Guessing

COPE's ability to exploit the broadcast nature of the wireless medium through overhearing and guessing is one of its signature features. This sets it apart from all prior work on network coding. To quantify the benefits of this approach, we experiment with the "x"-topology in Fig. 6-a. This is the analogy of the Alice-and-Bob topology but the two flows travel along link-disjoint paths. Fig. 6-b shows the CDF of the UDP throughput gain, taken over 40 measurements at different locations in our testbed. The gain is only slightly lower than the throughput gain when the two flows are on the reverse path of each other (Fig. 5-b). The correspond-



(a) Cross Topology (b) CDF of throughput gain with UDP (c) CDF of throughput gain with TCP

Figure 9—Throughput gains obtained with COPE, for the cross topology. The arrows show the direction of the four flows. The fi gure shows that COPE's performance gains increase with the number of flows intersecting at the bottleneck.

ing CDF for TCP is shown in Fig. 6-c. We see again that the throughput gain is comparable to that for the Alice-and-Bob experiment with TCP. This result is important, because in a real wireless network, there might be very few flows traversing the reverse path of each other, but one would expect many flows to intersect at a relay, and thus can be coded together using overhearing and guessing.

What if COPE guesses wrongly? Fortunately, COPE recovers from such mistakes. The combination of acks and reception reports allows a node to quickly detect what packets its neighbors failed to decode, and hence need to be retransmitted. Fig. 8 shows the CDF of the percentage of undecodable packets, with and without local retransmissions. Each CDF is taken over 40 measurements over the topology in Fig. 6. The experiments with retransmission turned on use a retransmission timeout of 40ms and a maximum of 2 retransmits. The figure shows that due to imperfect overhearing, guessing alone might result in undecodable packets. But guessing with acks and retransmissions produces high throughput while reducing the number of undecodable packets to negligible values.

7.4 Higher Throughput Gains

But what is the maximum achievable throughput gain, i.e, what is the theoretical capacity of a wireless network that employs COPE? While, the capacity of network coding for unicast traffic is still an open question [35, 13], we know that certain constructs increase the throughput gain. For example, consider the cross topology in Fig. 9-a. There are four UDP flows: $n_1 \rightarrow n_3$, $n_3 \rightarrow n_1$, $n_4 \rightarrow n_5$, and $n_5 \rightarrow n_4$. Without coding, the router, n_2, is a bottleneck; for every four packets n_2 receives, the MAC gives it, on average, one chance to send, resulting in many drops and wasted bandwidth. With COPE, and assuming perfect overhearing, the bottleneck node, n_2, can code all four packets that it receives from its neighbors in one transmission, achieving 4-fold throughput increase. Fig. 9-b shows the CDF of the throughput gain for 40 experiments taken over different cross topologies in our testbed. Interestingly, this distribution has a small knee when COPE codes only 2 packets (emulating either the Alice-and-Bob or "x" topologies), while the most prevalent mode of operation codes packets from all 4 flows. In practice, the gain is slightly lower than $4x$ because of header overhead and imperfect overhearing.

TCP also exhibits higher throughput gains in the cross topology of Fig. 9-a. TCP's gains are primarily due to coding gains, as explained above. Assuming perfect overhearing, the middle node can XOR 4 packets in each transmission, thus reducing the number of transmissions from 8 to 5. The resulting coding gain is $\frac{8}{5} = 1.6$. The throughput measurements shown in Fig. 9-c are slightly lower than the corresponding coding gains again because of the header overhead and imperfect overhearing.

Conceptually, it is possible to keep increasing the number of flows intersecting at a single bottleneck to increase the throughput gain. In practice, however, it is unlikely that the routing protocol will take such routes. In §7.6, we show that on our testbed with a random choice of flows, COPE XORs up to 5 packets together.

7.5 What Limits COPE's Performance?

It is important to know when COPE cannot help. COPE's throughput increase relies on the availability of coding opportunities, which depend on the diversity of the packets in the queue of the bottleneck node. In the Alice-and-Bob topology, if only 10% of the packets in the bottleneck queue are from Alice and 90% from Bob, then coding can at best sneak Alice's packets out on Bob's packets. Unless 50% of the packets are from Alice and 50% from Bob, we do not obtain the theoretical coding gain of 1.33. Indeed, coding favors fairness!

Fairness depends on the comparative quality of the channels from the sources to the bottleneck. For example, if the channel between Alice and the router is worse than that between Bob and the router, Alice might be unable to push the same amount of traffic as Bob. This unfairness is usually exacerbated by the *capture effect*. Although the 802.11 MAC should make all backlogged senders back off equally, the sender with the better channel (here Bob) usually captures the medium for long intervals, preventing other nodes (i.e., Alice) from obtaining a fair share of the bandwidth.

In practice, capture always happens to some degree, and the routing protocol tries to discount the capture effect by always selecting the stronger links. Yet, to study the effect of

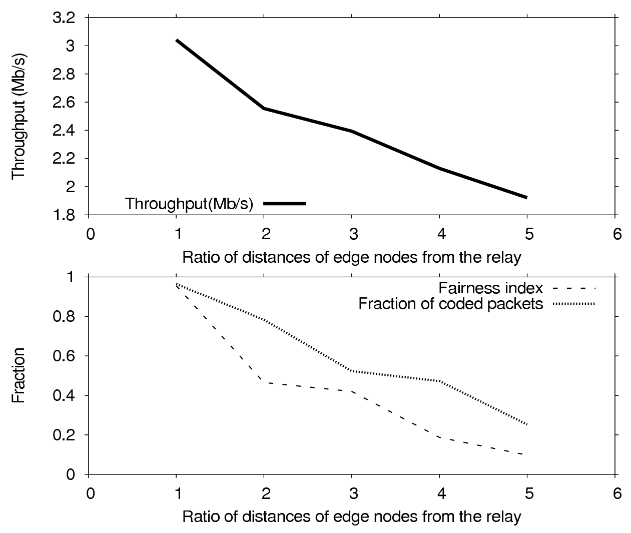


Figure 10—Effect of unequal channel qualities on coding opportunities and throughput gain in the Alice-and-Bob topology. COPE aligns the fairness and effi ciency objectives. Increased fairness increases coding opportunities and hence improves the aggregate throughput.

capture on fairness and thus coding opportunities, we intentionally stress the links in our topology. We take the Alice-and-Bob topology, set it such that both Alice and Bob are equidistant from the router, and compute the total network throughput (i.e., the sum of Alice's and Bob's throughput). We then gradually move Alice's node away from the router, and repeat the experiment and the measurements.

Fig. 10 shows the network throughput as a function of the ratio of Alice's and Bob's distance to the router. It also shows the percentage of coded packets and the *fairness index computed as the ratio of Alice's throughput to Bob's*. As Alice moves further away, Bob increasingly captures the channel, reducing fairness, coding opportunities, and the aggregate network throughput. Interestingly, without coding, fairness and efficiency are conflicting goals; throughput increases if the node with the better channel captures the medium and sends at full blast. Coding, however, aligns the two objectives; increasing fairness increases the overall throughput of the network, as apparent from Fig. 10.

7.6 In a Larger Network

We have advocated a simple approach to network coding in wireless environments, where each node relies on its local information to detect coding opportunities, and when possible XORs the appropriate packets. However, it is unclear how often such opportunities arise in realistic settings.

We run large-scale experiments on our 34 node testbed to gauge the increase in throughput provided by COPE. The flows arrive according to a Poisson process, pick sender and receiver randomly, and transfer files whose sizes follow the distribution measured on the Internet [8]. We focus on UDP flows, as TCP's behavior in a congested multi-hop wireless network is not yet well-characterized. We vary the arrival rates of the Poisson process to control the offered load. For each arrival rate, we run 10 trials, with coding on and then off (for a total of 500 experiments), and compute the network throughput in each case.

Fig. 11 shows that COPE dramatically improves the throughput of congested wireless networks, by a factor of 3-4*x* on average. The figure plots the aggregate end-to-end throughput as a function of the demands, both with COPE and without. At low demands (below 2Mb/s), coding opportunities are scarce, and COPE performs similarly to no coding. As the demands increase, both the extent of network congestion and the number of coding opportunities increase. In such dense networks, the performance without coding deteriorates because of the high level of contention and consequent packet loss due to collisions. In contrast, coding reduces the number of transmissions for the same amount of data, alleviates congestion, and consequently yields higher throughput.

It is interesting to examine how much of the coding is due to guessing, as opposed to reception reports. Fig. 13 plots the percentage of packets that have been coded because of guessing for the experiments in Fig.11. The percentage is calculated as follows: If n packets are coded together, and at most k packets could be coded using reception reports alone, then $n - k$ packets are considered to be coded due to guessing. The figure shows that the benefit of guessing varies with demands. At low demands, the bottleneck nodes have small queues, leading to a short packet wait time. This increases dependence on guessing because reception reports could arrive too late, after the packets have been forwarded. As the demands increase, the queues at the bottlenecks increase, resulting in longer wait times, and consequently allowing more time for reception reports to arrive. Hence, the importance of guessing decreases. However, as the demands surge even higher, the network becomes significantly congested, leading to high loss rates for reception reports. Therefore, a higher percentage of the coding decisions is again made based on guessing.

Let us now examine in greater detail the peak point in Fig. 11, which occurs when demands reach 5.6 Mb/s. Fig. 14 shows the PDF of the number of native packets XOR-ed at the bottleneck nodes (i.e., the nodes that drop packets). The figure shows that, on average, nearly 3 packets are getting coded together. Due to the high coding gain, packets are drained much faster from the queues of the bottleneck nodes, avoiding drops. The result is an average throughput gain of 3-4*x*. In fact, as illustrated in Fig. 12, COPE provides consistent throughput gains, even as congestion increases.

8 Conclusion

To date, network coding has remained an elegant theoretical idea. It has focused mainly on analytical tractability, usually assuming multicast traffic, known senders and receivers, and smooth traffic rates. Our work adapts this theory to the design of the first practical system that plugs network coding into the current network stack, in a manner

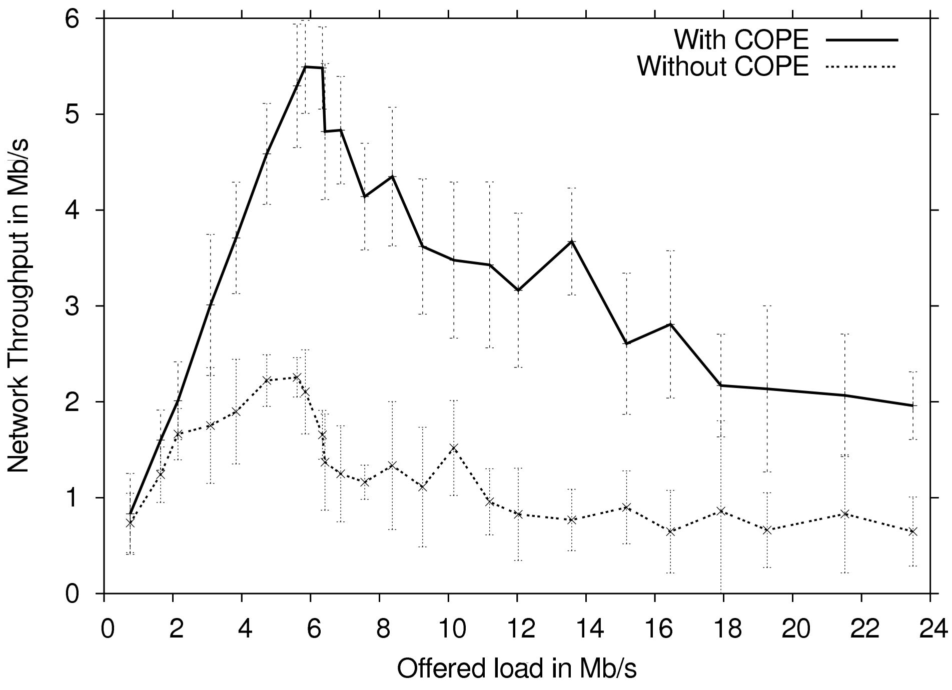


Figure 11—COPE provides a several-fold increase in the throughput of wireless mesh networks. Results are for flows with randomly picked source-destination pairs, Poisson arrivals, and heavy-tail size distribution.

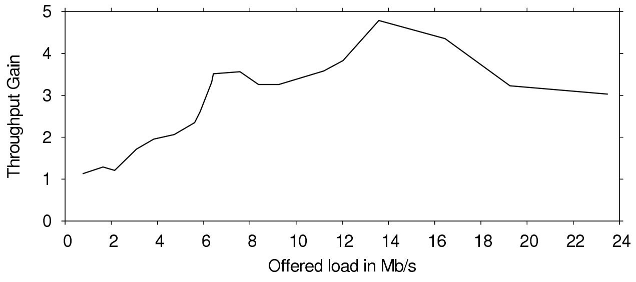


Figure 12—COPE's throughput gain as a function of offered load, for the set of experiments in Fig. 11.

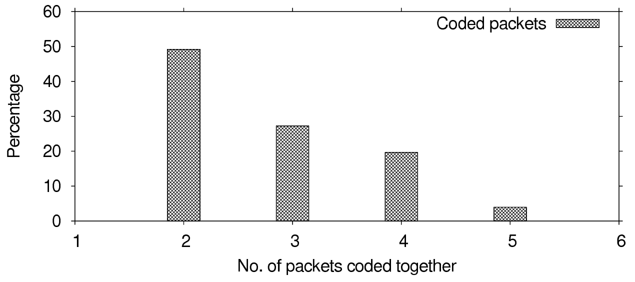


Figure 14—Distribution of number of packets coded together in the test bed at the peak point of Fig. 11.

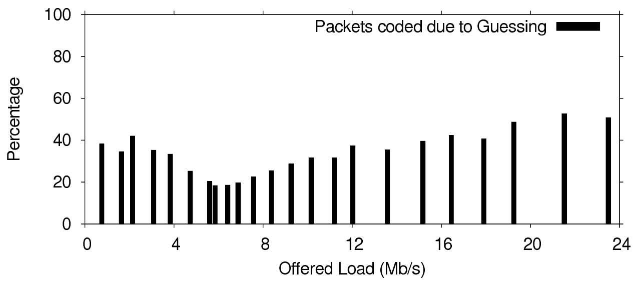


Figure 13—Percentage of packets coded in the testbed due to guessing, as a function of offered load, for the set of experiments in Fig. 11.

that can be seamlessly leveraged by existing applications. Our results show that, in practice, network coding provides several-fold improvements in the throughput of wireless networks.

By incorporating network coding into the router architecture and introducing it to the systems community, our work opens up many venues for future research. In addition to bandwidth, wireless networks struggle with power consumption in constrained applications such as sensor networks and mobility. Network coding provides a unified framework to address these limitations. Coding reduces the number of transmissions needed to deliver the information, which translates directly to a reduction in communication power and bandwidth consumption. Thus, a network coding layer may be the first step towards a new generation of low-power wireless networks that could achieve significantly better throughput than current approaches.

References

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *ACM SIGCOMM, 2004.*

[2] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung. Network Information Flow. In *IEEE Transactions on Information*

Theory, 2000.

[3] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links.

[4] Bay area wireless user group. http://www.bawug.org.

[5] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *ACM MobiCom, 2005*.

[6] S. Biswas and R. Morris. Opportunistic routing in multi-hop wireless networks. *ACM SIGCOMM, 2005*.

[7] Christos Gkantsidis and Pablo Rodriguez. Network Coding for Large Scale Content Distribution. In *IEEE INFOCOM, 2005*.

[8] M. E. Crovella, M. S. Taqqu, and A. Bestavros. Heavy-tailed probability distributions in the World Wide Web. In R. J. Adler, R. E. Feldman, and M. S. Taqqu, editors, *A Practical Guide To Heavy Tails*, chapter 1, pages 3–26. Chapman and Hall, New York, 1998.

[9] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *ACM MobiCom '03*, San Diego, California, September 2003.

[10] S. Deb, M. Effros, T. Ho, D. R. Karger, R. Koetter, D. S. Lun, M. Médard, and N. Ratnakar. Network coding for wireless applications: A brief tutorial. In *IWWAN*, 2005.

[11] R. Draves, J. Padhye, and B. Zill. Comparison of Routing Metrics for Multi-Hop Wireless Networks. In *Proceedings of ACM SIGCOMM*, 2004.

[12] M. Heusse, F. Rousseau, R. Guillier, and A. Duda. Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless lans. In *ACM SIGCOMM, 2005*.

[13] T. Ho and R. Koetter. Online incremental network coding for multiple unicasts. In *DIMACS Working Group on Network Coding*, 2005.

[14] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros. The Benefi ts of Coding over Routing in a Randomized Setting. In *ISIT, 2003*.

[15] T. Ho, B. Leong, Medard, R. Koetter, Y. Chang, and M. Effros. The Utility of Network Coding in Dynamic Environments. In *IWWAN*, 2004.

[16] p. a. IEEE 802.11 WG. Wireless lan medium access control (mac) and physical layer (phy) specifi cations. *Standard Specification,IEEE, 1999*.

[17] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*, 2003.

[18] B. Karp. *Geographic Routing for Wireless Networks*. PhD thesis, Harvard University, 2000.

[19] R. Koetter and M. Medard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking, 2003*.

[20] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems, August 2000*.

[21] S. R. Li, R. W. Yeung, and N. Cai. Linear network coding. In *IEEE Transactions on Information Theory*, 2003.

[22] D. S. Lun, M. Médard, and R. Koetter. Effi cient operation of wireless packet networks using network coding. In *International Workshop on Convergent Technologies (IWCT)*, 2005.

[23] D. S. Lun, M. Médard, R. Koetter, and M. Effros. Further results on coding for reliable communication over packet networks. In *IEEE International Symposium on Information Theory (ISIT 05)*, 2005.

[24] D. S. Lun, N. Ratnakar, R. Koetter, M. Mdard, E. Ahmed, and H. Lee. Achieving Minimum-Cost Multicast: A Decentralized Approach Based on Network Coding. In *IEEE INFOCOM, 2005*.

[25] Internet packet size distributions: Some observations. http://netweb.usc.edu/ rsinha/pkt-sizes/.

[26] V. Paxson and S. Floyd. Wide-area traffi c: the failure of poisson modeling. In *IEEE/ACM Transactions on Networking, 3(3):226–244*, 1995.

[27] A. Ramamoorthy, J. Shi, and R. Wesel. On the capacity of network coding for wireless networks. In *41st Annual Allerton Conference on Communication Control and Computing*, Oct. 2003.

[28] Seattle wireless. http://www.seattlewireless.net.

[29] P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTCP: A reliable transport protocol for wireless wide-area networks. *Wireless Networks*, 8(2-3):301–316, 2002.

[30] ttcp. http://ftp.arl.mil/ftp/pub/ttcp/.

[31] udpgen. http://pdos.csail.mit.edu/click/ex/udpgen.html.

[32] J. Widmer and J.-Y. L. Boudec. Network Coding for Effi cient Communication in Extreme Networks. In *Proceedings of ACM SIGCOMM WDTN*, 2005.

[33] Y. Wu, P. A. Chou, and S. Y. Kung. Information Exchange in Wireless Networks with Network Coding and Physical-layer Broadcast. *MSR-TR-2004-78*.

[34] Y. Wu and S.-Y. Kung. Distributed utility maximization for network coding based multicasting: a shorted path approach. submitted to IEEE INFOCOM 2006.

[35] Z. Li and B. Li. Network Coding in Undirected Networks. In *CISS 04*, 2004.

[36] Z. Li and B. Li. Network coding: The case for multiple unicast sessions. In *Allerton Conference on Communications*, 2004.