

# XBC: XOR-based Buffer Coding for Reliable Transmissions over Wireless Networks

Zizhan Zheng and Prasun Sinha  
Department of Computer Science and Engineering  
The Ohio State University  
Columbus, OH 43210  
Email: {zhengz, prasun}@cse.ohio-state.edu

*Abstract*—In-network caching is a useful technique for reducing latency and retransmission overhead of lost packets for reliable data delivery in wireless networks. However, in-network caching is challenging to implement in memory constrained devices such as RFIDs and sensors, and also in Wireless LAN (WLAN) gateways for large-scale deployments. In this paper we propose a novel technique for management of in-network caches using XOR coding for optimizing the use of limited buffer space in presence of random and burst packet losses. We identify two critical parameters, *coding degree* and *coding distance* for the coding scheme. As a case-study we implement our approach over Snoop and evaluate its performance for a WLAN. Using simulations in *ns-2*, we observe that when the size of the retransmission buffer on the gateway is less than 16 packets per TCP flow, the throughput can be enhanced by up to 30% for random losses and up to 20% for burst losses.

## I. INTRODUCTION

Wireless networks have been widely deployed in recent years due to their low cost, reduced dependence on infrastructure, and support for emerging mobile and sensing applications. However, due to the high bit-error rates of wireless links and losses induced by interference and mobility, supporting reliable transmissions over wireless networks remains to be challenging, especially for networking devices with limited resources. To deal with the high loss rate of wireless networks and improve end-to-end throughput, many existing reliable protocols support *in-network* packet recovery [1]–[4]. In these protocols packet caching at the intermediate nodes is used for faster and low-overhead recovery (compared to recovery from source) of lost packets. As a special case, hop-by-hop recovery at MAC layer is widely adopted by wireless networks [5] [6]. However, in-network caching is challenging in memory-constrained devices such as RFIDs or sensors, and also in devices that are handling a large number of flows such as WLAN gateways for large installations.

A retransmission buffer with limited size restricts the performance of reliable protocols in two ways. First, the number of concurrent flows that can be supported by either the sender or intermediate nodes is limited. Second, intermediate nodes can only cache a subset of the forwarded packets, thus reducing the benefit of in-network caching. Limitations on buffer size force the caching node to make decisions on which packets to cache and which not to. However, an intermediate node can not predict which of the packets that it is forwarding will be

lost in the network en-route to the destination, and thus has to resort to random sampling of packets to cache.

In this paper, we propose a novel XOR coding based approach named XBC to optimize the use of limited retransmission buffer space. In addition to raw packets, intermediate nodes can store the XOR of two or more packets in the buffer. Thus a larger number of packets can be stored in the limited buffer space. Upon discovery of a loss, a packet can be recovered if it is stored raw, or if the other packets with which it is encoded have been received by the receiver. For example if  $P_i \oplus P_j$  is stored in the buffer, and if  $P_j$  is lost in the network, but  $P_i$  is known to have been received correctly, then the cache can transmit  $P_i \oplus P_j$  and the receiver can compute  $(P_i \oplus P_j) \oplus P_i$  to extract  $P_j$ . Decoding of XORed packets at the receiver requires previously received packets from the receiver's buffer, which is readily available for reliable protocols supporting in-order delivery.

Fig. 1 shows a simple example of how XBC can be useful. We consider a network with three nodes, in which a base station connects a wired link and a wireless link. Suppose there is a TCP flow going through the base station. In a traditional protocol supporting local recovery, like Snoop [2], the base station caches packets received, forwards them to the receiver, and retransmits the lost ones upon receiving an ACK indicating packet loss from the receiver. Assume that the wireless link experiences random losses and the packet loss rate is low, say less than 10%. Further assume that the base station can forward 4 packets during the round trip time between it and the receiver. Then to ensure 100% local recovery, the base station needs a buffer that can cache 4 packets. On the other hand, the base station can cache the XOR of every two raw packets. Since the chance that two packets in same coding pair are lost is low, half of the buffer space can be saved while the chance of local recovery is still very high.

We identify two pivotal parameters in the design of XBC: *coding-degree* and *coding-distance*. Coding-degree is the number of packets coded together. The higher the coding-degree, the less memory the buffer requires; or equivalently, the more the number of cached packets. However, the chance that an XOR-ed packet will get successfully decoded also reduces with increasing degree. Thus, the optimum coding-degree is a function of the packet loss probability. In networks with burst losses, coding packets with contiguous sequence numbers

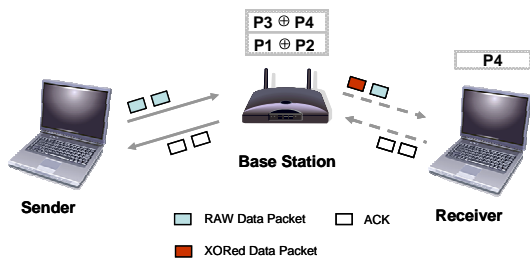


Fig. 1. A simple example of how XOR coding optimizes the retransmission buffer on the intermediate node (the base station in this case). The base station caches XOR coded packets to save buffer while ensuring high chance of local recovery. For instance, assume that packet  $P1$ ,  $P2$ , and  $P4$  have been received and packet  $P3$  is lost at the receiver, then the receiver only need to cache  $P4$ .  $P3$  can be recovered by transmission of  $P3 \oplus P4$  from the base-station.

is not a good choice as more than one of these packets can be lost simultaneously, resulting in decoding failure. To deal with burst losses, XBC allows non contiguous packets to be coded together, where the gap between the sequence number of packets that can be coded together is called the coding distance. The optimum coding distance depends on the characteristics of the burst loss, such as the average burst length and the average inter-burst separation.

As a case-study, we show how XBC can be integrated with the Snoop protocol [2]. We choose Snoop as the baseline because it is well known and ready for use for both simulations and experiments. Note that XBC can be easily implemented as a plug-in to any other reliable protocol that supports in-network recovery through retransmission buffers.

The remainder of this paper is organized as follows. In the next section, we summarize the related work on reliable transmissions on wireless networks and other applications of XOR coding. In section III, we present the design of XBC and analyze its benefit for networks with random or burst losses. Section IV describes the implementation of XBC on Snoop and Section V evaluates the performance of Snoop with or without XBC. Finally in section VI, we conclude our work and give pointers for future research.

## II. RELATED WORK

Reliable transmissions over wireless networks have been widely studied. Many of these protocols support in-network recovery, and require a recovery buffer to be set up at some or all of the intermediate nodes in an end-to-end connection, where our approach has potential to optimize the buffer usage. Equivalently, our approach can be used to make use of the limited buffer to enhance performance of in-network recovery.

**Transport Layer Recovery:** To support reliable communication over wireless networks, both end-to-end approaches [7]–[9] and in-network enhancements [2], [10], [11] have been explored. In the end-to-end approaches the receiver determines the cause of failure and notifies the TCP sender through explicit end-to-end feedback. For non-congestion related losses the sender does not invoke congestion control. On the other hand, the idea of in-network solutions is to provide a reliable

link layer to higher layers through link layer automatic repeat request (ARQ) or forward error correction (FEC).

**Link-Layer Recovery:** A link-layer protocol supporting local recovery may simply perform retransmissions without the knowledge of transport layer so that it can be used by different high layer protocols, as in the case of 802.11 MAC [6]; or it can smartly use that knowledge to achieve better throughput, as in the case of Snoop [10]. Extensions of Snoop to multi-hop wireless networks have also been suggested [1], [3]. The benefits of these protocols over pure hop-by-hop link-layer retransmissions include (1) the ability to perform recovery over multiple hops so that loss due to queue overflow or mobility may be handled, which is not possible in hop-by-hop retransmission protocols; (2) the ability to make use of additional information from routing or transport layers to further improve performance by mechanisms such as bypassing an unreachable neighbor and suppressing duplicate ACKs for packets lost due to bit errors.

In DTC [3], each intermediate node caches one TCP segment that has the highest sequence number seen and not received by the next hop (through link layer feedback), so that segments with higher sequence numbers are cached on nodes closer to the receiver. ACKs are used for both acknowledging new data and requesting local recovery. Round-trip time estimation is therefore needed to distinguish between these two cases. Different from DTC where only the receiver generates transport-layer ACKs, InPCM [1] allows intermediate nodes to cache both data packets and ACKs, and an intermediate node may detect losses and send ACKs to its previous hop. Besides reducing end-to-end retransmissions, these approaches also help balance the energy consumption among nodes, which is especially important for sensor networks. However, all the above approaches either focus on improving end-to-end performance or reducing energy consumption, and none of them study the impact of retransmission buffer on the achievable benefit of in-network recovery, which is the focus of this paper.

FEC-based protocols [12] [13] achieve reliability by adding redundancy to the packets transmitted. Such mechanisms could be used at the sender or the intermediate nodes. Because no acknowledgements are needed, FEC-based protocols are buffer-free and incur low latency. However, the optimal FEC coding highly depends on the error nature of networks and can not be easily achieved for wireless networks due to the unpredictable channel quality [14].

**In-network recovery in broadcast protocols:** The idea of local recovery is not restricted to unicast, but also common in reliable broadcast and multicast protocols [4], [15], [16]. PSFQ is a reliable broadcast protocol that uses hop-by-hop recovery in which each node caches packets received so far, forwards packet in-sequence, and requests lost ones from its neighbors at a faster rate than that for forwarding. Sprinkler [15] is also a reliable broadcast protocol that uses the nodes on a connected dominating set for distribution of data and recovery of lost

packets. ReMHoc [16] is a NACK-based reliable multicast protocol for mobile ad-hoc wireless networks, in which every multicast group member is responsible for in-network packet recovery.

**Network Coding:** As a simple way to put the elegant idea of network coding [17] into practice, XOR coding has attracted attention in the networking field, and has been used to improve the throughput of both unicast and multicast over wireless networks [18], [19]. In both cases, the idea is to perform XOR coding at the crossing point nodes of a multi-hop wireless network, assisted by overhearing, and inference or local feedback, to reduce the number of packet transmissions. Although XOR coding is usually less optimal than the general linear codes [20] in terms of reducing unnecessary transmissions, it has several benefits. It is much easier to implement, incurs lower encoding and decoding cost, has lower buffer requirement, and introduces lower delay due to coding. In this paper, XOR coding is used in a very different way and with a different optimization target from previous works. First, coding is used by the intermediate nodes that have local retransmission buffers, but are not necessarily crossing point nodes. Second, XOR coding is applied to packets received at different times instead of different flows as in the case of COPE [18] or CODEB [19]. Third, the raw packets used for decoding are received earlier but not overheard from other nodes. Finally, the purpose of XBC is not to locally reduce number of transmissions, but to reduce number of end-to-end retransmissions by efficient use of buffers.

### III. XOR-BASED CODED BUFFER: DESIGN AND ANALYSIS

#### A. Overview

XBC sits upon a reliable protocol supporting in-network recovery and performs packet encoding at the intermediate nodes with local buffers and packet decoding at the receiver of a reliable communication flow. Consider one of such intermediate nodes and the end-receiver in the following discussion.

For each packet  $P$  received, the intermediate node first searches in its local buffer for the packet that should be combined with  $P$  according to the coding strategies discussed below. Assume there is such a packet, which can be either raw or XOR-ed packet,  $P_1 \oplus P_2 \oplus \dots \oplus P_i$ . The node then replaces the packet with  $P \oplus P_1 \oplus P_2 \oplus \dots \oplus P_i$ . If such a packet is not present and the buffer has available space, the node simply caches  $P$  in its buffer. Otherwise, the node may either replace an old packet in the buffer with  $P$  or simply not cache  $P$ , depending on which policy is preferred. Then the node forwards  $P$  to the receiver.

If  $P$  is lost, the intermediate node will be notified by either intercepting and analyzing the acknowledgments from the receiver to the sender or through a timeout [2], which is provided by the in-network recovery protocol. When this happens, the node will search the buffer for either the lost packet or the XOR-ed packet that contains the lost one, and retransmit it if such a packet exists. The sequence number of the retransmitted packet is the same as the lost one and the

sequence numbers of each component of the coded packet is recorded in its packet header (see Section IV-B for the implementation details).

To support in-order delivery, the receiving side of most reliable protocols maintain a buffer for the out-of-order packets. XBC makes use of this buffer and requires a small additional memory to cache a few old packets which may have been delivered to the application layer to assist decoding (see section IV). Assume the receiver receives an XOR-ed packet  $P(s_j) = P_1(s_1) \oplus P_2(s_2) \oplus \dots \oplus P_i(s_i)$ , where  $s_i$  is the sequence number of  $P_i$ , and the sequence number of the encoded packet is the same as that of the lost component, which is  $P_j$  ( $1 \leq j \leq i$ ) in this case. The receiver searches for each coded component in its local buffer. If all of them but  $P_j$  is missing the receiver can retrieve the missing one by computing  $P \oplus P_1 \oplus P_2 \oplus \dots \oplus P_{j-1} \oplus P_{j+1} \dots \oplus P_i$ .

#### B. Basic Coding Model

Assume the buffer on an intermediate node can cache at most  $K$  packets. We define window size  $N$  to be the maximum possible number of packets forwarded by this node but unacked by the receiver, which can be calculated from the RTT between this node and the receiver and the network bandwidth. If  $N = 1$ , then there is no chance to perform XOR coding. Therefore we only consider the case when  $N \geq 2$ .

To see the benefit of XBC, consider a simple case where  $K = 1$ , and coding degree is fixed at two, that is, at most two packets will be coded together. We further assume that the packet loss rate on the path from this node to the receiver is fixed to  $p$ , while the backward path from the receiver to this node is reliable (therefore ACKs will not be lost), and this node only performs one retransmission for every lost packet. The node can either cache 1 of the  $N$  packets or the XOR coding of any randomly selected 2 of the  $N$  packets in the buffer. In other words, we assume that a window of packets are received and processed at the same time. We will calculate the local recovery probability  $q$  for the above two cases, which is defined as the probability of recovering any packet forwarded by this node that is lost enroute to the destination.

If no coding is performed, then each lost packet can only be recovered if it is in the local buffer. Therefore,  $q_{raw} = 1/N$ . On the other hand, assume packet  $P_i$  and  $P_j$  are coded together. Then for any packet  $P$ , if it is lost in the first transmission, then it can be recovered only if it is either  $P_i$  or  $P_j$  and not both of them are lost. Therefore, it is easy to see that  $q_{xbc} = (1 - p) \times 2/N$ .

Compare  $q_{raw}$  with  $q_{xbc}$ . It is easy to see that  $q_{xbc} > q_{raw}$  iff  $p < 0.5$ . Since packet loss rate in wireless networks is usually less than 0.5 when the intermediate node is not far away from the receiver, this simple coding scheme ensures that more lost packets can be recovered locally, which saves network bandwidth and reduces transmission delay.

#### C. Dealing with Random Losses

In this section, we discuss a direct extension of the simple case discussed above. First, we consider a general buffer size

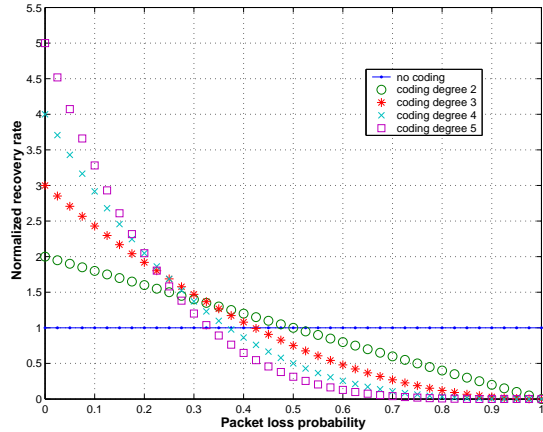


Fig. 2. Normalized packet recovery probabilities under random losses

$K \geq 1$ . Second, to get more benefit from XOR coding, we consider a general coding degree  $d \geq 2$ . We are interested in local recovery probability at an intermediate node in this general setting. In other words, for any contiguous  $N$  packets that are forwarded by this node and suffer from loss probability  $p$ , we are interested in the probabilities of recovering an arbitrary lost packet under two cases: (1)  $K$  of the  $N$  packets are randomly selected and cached in the buffer; (2)  $K \times d$  of the  $N$  packets are randomly selected, every  $d$  of them are then encoded together (the actual coding schema is not important here) and cached in the buffer. Following a similar argument as above, it is not hard to see that in this general case,

$$q_{raw} = \frac{\min(K, N)}{N} \quad (1)$$

$$q_{xbc} = \frac{\min(K \times d, N)}{N} \times (1 - p)^{d-1} \quad (2)$$

To simplify discussion, we only consider the case that  $K \times d \leq N$ . To see the benefit of XBC for different loss rates and code degrees, we plot  $q_{xbc}/q_{raw} = d \times (1 - p)^{d-1}$  in Fig. 2. We can see that the higher the coding degree, the higher the benefit of using XBC, but the region where XBC is useful is narrower. Also for very high loss rates, using XBC can degrade the performance, as the probability of decoding becomes low due to increased chance of multiple losses within a coded packet.

In the above discussion, we implicitly assumed that the packets coded together are picked up randomly. However, in practice, random coding can only be carried out when  $N$  is known and may lead to inefficient use of local buffer when  $K$  and  $N$  are both small. Therefore, XBC performs coding on sequence numbers that are pre-determined. Furthermore, in the case of random losses, XBC always codes contiguous packets together. However, this simplified and restrictive implementation loses optimality in comparison to ideal cases shown in Fig. 2.

#### D. Dealing with Burst Losses

In this section, we discuss the extension of the basic coding model for networks with burst losses. Since it is still not

TABLE I  
EXAMPLES OF XBC CODING

0 $\oplus$ 2	0 $\oplus$ 4	0 $\oplus$ 2 $\oplus$ 4
1 $\oplus$ 3	1 $\oplus$ 5	1 $\oplus$ 3 $\oplus$ 5
4 $\oplus$ 6	2 $\oplus$ 6	6 $\oplus$ 8 $\oplus$ 10
5 $\oplus$ 7	3 $\oplus$ 7	7 $\oplus$ 9 $\oplus$ 11
8 $\oplus$ 10	8 $\oplus$ 12	12 $\oplus$ 14 $\oplus$ 16
9 $\oplus$ 11	9 $\oplus$ 13	13 $\oplus$ 15 $\oplus$ 17
12 $\oplus$ 14	10 $\oplus$ 14	18 $\oplus$ 20 $\oplus$ 22
13 $\oplus$ 15	11 $\oplus$ 15	19 $\oplus$ 21 $\oplus$ 23

$d = 2, l = 2$

$d = 2, l = 4$

$d = 3, l = 2$

clear how to model the burst losses on multi-hop wireless networks, we assume that, as a first step, the two state Markov chain Gilbert-Elliot (GE) model [21] used in Seda [22] and Syndrome [9] for single link can be extended to a multi-hop path. Similar to Seda, when the path between the intermediate node and the receiver is in the good state, there is no packet loss; when the path is in the bad state, it suffers from a specific bit-error-rate (BER). The error model is therefore defined by a set of transition probabilities and the BER of the bad state. Equivalently, the interleaving good and bad states can also be characterized by the mean and standard deviation of the error cluster size and inter-cluster size, where the size is defined as the number of bits transmitted in each state [22].

Based on the above error model, it is clear that coding packets with contiguous sequence numbers together is not a good choice since more than one of these packets may be lost when the path is in the bad state, and therefore none of them can be decoded. To deal with this problem, XBC introduces the concept of coding distance (see section I) and allows the packets separated by a fixed number of packets to be coded together.

With coding degree and coding distance taken into account, the general coding schema of XBC is as follows. For coding degree  $d$  and coding distance  $l$ , each coded packet has the following format:  $P = P_s \oplus P_{s+l} \oplus \dots \oplus P_{s+(d-1) \times l}$ , where  $s \bmod (d \times l) \in [0, l-1]$ , and  $s, s+l, \dots$ , are sequence numbers of the packets being coded. In the current implementation of XBC, a packet  $P_s$  can not be arbitrarily encoded with  $P_{s-l}$  or  $P_{s+l}$ . The benefit of the fixed coding format is that both encoding and decoding can be simplified, while losing the ability to code some groups of packets that have the proper separation of sequence numbers. Table I gives the snapshots of a sender buffer with the capacity of 8 packets, with specified coding degree  $d$  and coding distance  $l$ .

1) *XBC for Unbounded Sender Buffer*: To see the impact of coding distance in real setting, we first assume that the size of sender buffer is unbounded, and calculate the recovery probabilities of lost packets for the five loss models derived in Seda based on real experiments (see table II).

In our calculation, a sequence of bits is first generated for each loss model, and then each bit is marked as either error or error free according to the loss model used. Each sequence is then divided into 25-byte blocks and 4-block frames (default values used by Seda), and a block is marked as error if there

TABLE II  
LOSS MODELS [22]

Loss Model	Mean Error Cluster Size (bits)	Mean Inter-cluster Size (bits)	$e_b$	Overall BER
1	250	1000	0.40	0.080
2	100	1000	0.40	0.036
3	386	3234	0.43	0.045
4	120	3234	0.36	0.013
5	386	9690	0.40	0.015

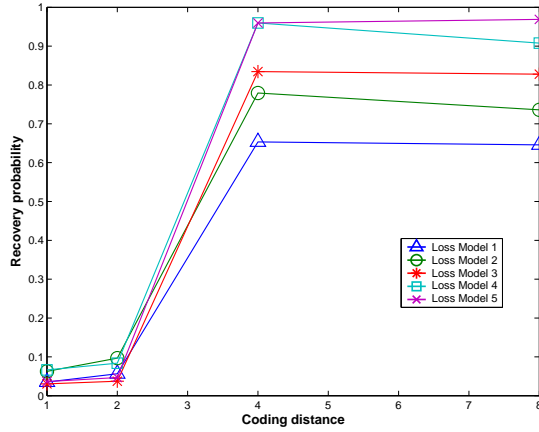


Fig. 3. Packet recovery probabilities under burst losses for unbounded buffer

is bit error(s) in the corresponding frame header or the block itself. For all cases, coding is performed on blocks, and coding degree is fixed at 2. For each block with error, if the block which it should be coded with is error free, then it can be recovered; otherwise both of them will be lost. We calculate the average chance of recovery for each block with error, which can be viewed as the ideal case of the above defined recovery probability  $q_{xbc}$  when there is no delay for ACKs and the assumptions made in the last section hold (no loss for ACKs, retransmitting once, etc.). Fig. 3 shows the results when coding distance is 1, 2, 4 and 8 respectively. We can see that for all these loss models, a coding distance of 4 can achieve much higher recovery probability than distance 2, even if the buffer is infinite. However, the results for distance 8 are a little worse than those of distance 4. The reason is that if degree is too high, it is possible that two blocks belonging to two adjacent error clusters are coded together, which has the similar effect as coding two blocks within the same error cluster.

2) *XBC for bounded sender buffer*: Given a buffer of size  $K$ , in this section we compute the recovery probability  $q_{xbc}$  for coding distance  $l$ . To simplify the discussion, we make the following assumptions. First, we assume that  $N$  is a multiple of  $d \times l$  and  $K$  is a multiple of  $l$ , so that every packet within a window will be cached if one of them does and the buffer can be fully used under the coding scheme discussed above. Second, the two state loss model is applied in the packet level instead of bit level, and in the good state there is no

packet loss. Let  $p_{gb} = a$  and  $p_{bg} = b$ , where  $p_{gb}$  is the transition probability from good state to bad state, and  $p_{bg}$  is the probability from bad state to good state. Then  $p_{gg} = 1 - a$  and  $p_{bb} = 1 - b$ . Let  $p$  be the packet loss probability in the bad state. Third, similar to the random loss case, a window of packets is received and processed at the same time.

Notice that under the above coding scheme, for a given packet, the sequence number of the packets it is encoded with are fixed. If  $P$  is lost, then  $P$  can only be recovered if the coded packet that  $P$  belongs to is buffered, and except  $P$  all other packets that  $P$  is coded with are received. Therefore, we have

$$q_{raw} = \frac{\min(K, N)}{N} \quad (3)$$

$$q_{xbc} = \frac{\min(K \times d, N)}{N} \times Q \quad (4)$$

where  $Q$  is the probability that all the other packets that  $P$  are encoded are received given that  $P$  is lost. Equation (3) holds because  $q_{raw}$  is independent of the type of losses. When  $d = 2$ ,  $Q$  is the probability that  $P_{i+l}$  is received given that  $P_i$  is lost, or equivalently the probability that  $P_{i-l}$  is received given that  $P_i$  is lost. Since the two state loss model is symmetric in time, we only consider the former case. Since  $P_i$  is lost, the current state must be the bad state since there is no loss in the good state. Therefore,  $Q = p_{bg}^{(l)} + p_{bb}^{(l)} \times (1 - p)$ , where  $p_{bg}^{(l)}$  is the probability that the system will be in good state after  $l$  steps given that it is in bad state now. This  $l$ -step transition probabilities of the Markov chain [23] can be easily obtained:

$$p_{bg}^{(l)} = \frac{b - b \times (1 - a - b)^l}{a + b} \quad (5)$$

$$p_{bb}^{(l)} = \frac{a + b \times (1 - a - b)^l}{a + b} \quad (6)$$

Therefore, for coding degree 2,

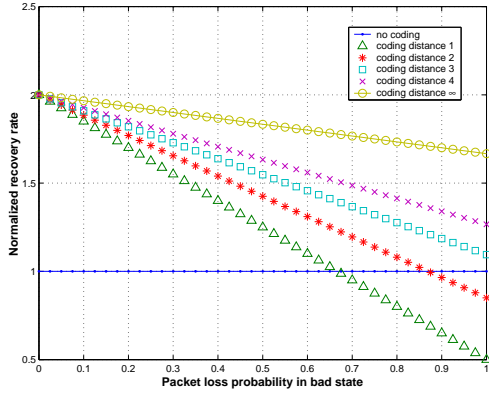
$$Q = \frac{b - b \times (1 - a - b)^l}{a + b} + \frac{a + b \times (1 - a - b)^l}{a + b} \times (1 - p) \quad (7)$$

$$q_{xbc}/q_{raw} = 2Q \quad \text{if } d = 2 \text{ and } K \times d \leq N. \quad (8)$$

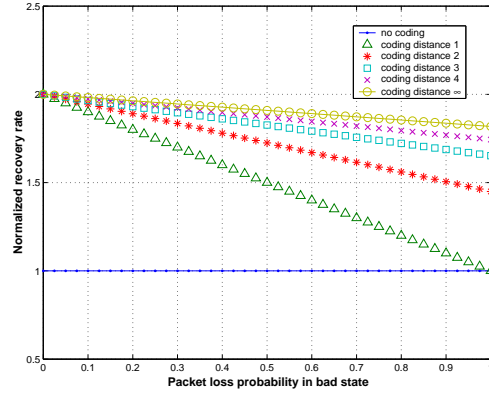
Given mean error cluster size  $N_b$  and mean inter-cluster size  $N_g$  in terms of packets, we have  $a = 1/N_g$  and  $b = 1/N_b$ . Fig. 4 plots  $q_{xbc}/q_{raw}$  under different coding distances and loss probabilities in the bad state. Compared with random losses, the benefit of coding is lower in the case of burst losses.

### E. Beyond XOR Coding

In this section, we discuss the possibility of using a general linear code instead of XOR coding to optimize buffer usage. Specially, we consider the case of Reed-Solomon code (RScode) and use it in a similar way shown in [19]. Given window size  $N$  and buffer size  $K$ , the RScode is applied to the  $N$  raw packets to get  $K$  coded packets, which are cached in the buffer. The code used is represented by a  $K \times N$  Vandermonde matrix  $\Theta$  in finite field  $\mathcal{F}_{2^8}$  (byte level coding). Let  $\mathbb{P}$  be the set



(a)  $N_g = 20, N_b = 4$



(b)  $N_g = 20, N_b = 2$

Fig. 4. Normalized packet recovery probabilities under burst losses (coding degree = 2)

of raw packets within a window, then the  $K$  packets buffered are simply  $\mathbb{Q} = \Theta \times \mathbb{P}$ .

The benefit of RScode is that it can achieve 100% reliability in the ideal case. If any  $M$  ( $M \leq K$ ) packets within a window are lost, and the receiver caches any  $N - K$  successfully received packets in the same window, then all these lost packets can be recovered once the  $K$  coded packets are received, which can be shown as following. Let  $\mathbb{R}$  be the  $N - K$  raw packets cached by the receiver, and  $\Lambda$  be such a matrix that  $\mathbb{R} = \Lambda \times \mathbb{P}$ . Then we have  $\mathbb{P} = \begin{pmatrix} \Lambda \\ \Theta \end{pmatrix}^{-1} \times \begin{pmatrix} \mathbb{R} \\ \mathbb{Q} \end{pmatrix}$ . This is feasible because  $\begin{pmatrix} \Lambda \\ \Theta \end{pmatrix}$  has full rank.

However, RScode has some limitations in practice. First, both encoding and decoding overheads are high, especially when  $N$  and  $K$  are large. Second, while encoding operation can be done incrementally by updating every buffered packet upon receiving a new packet, a packet can only be decoded after all the  $K$  coded packets are received. Therefore, delay due to coding can be high. Furthermore, partial decoding is not possible. If more than  $K$  packets are lost, all the packets in the buffer are useless.

#### IV. IMPLEMENTATION OF XBC OVER SNOOP

We have implemented XBC over the Snoop protocol [2] in *ns-2* simulator [24]. We first give a brief description of Snoop<sup>1</sup>, and its implementation in *ns-2* followed by the implementation details of XBC.

##### A. Snoop Overview

Snoop was initially designed for TCP flows over heterogeneous networks composed of wired and wireless links connecting users to the base-stations. For TCP flows initiated at the wired end, the Snoop agent on the base station intercepts the data packets from wired end and the ACKs going in the

opposite direction. It caches data packets before forwarding them and selectively forwards ACKs to wired end. The snoop agent differentiates between loss due to congestion and link error, and retransmits the packets that are lost due to link error after two duplicate ACKs are received.

To support TCP connections initiated at the wireless end, two solutions were proposed. The Snoop agent may either send NACKs to the wireless node [2] or set the Explicit Loss Notification (ELN) bit [7] on the header of ACKs from the wired end for packet losses due to link errors. The wireless node then retransmits the lost packets immediately without waiting for three duplicate ACKs or a timeout.

In *ns-2*, Snoop is implemented as a new link layer sitting between a routing agent and MAC layer on the base station. An instance of Snoop agent is created for every TCP flow going through the base station. For TCP flows initiated at the wired end, a local buffer that can cache at most 100 data packets by default is set up. When a new packet is received and the buffer is full, two simple caching strategies are supported: LRU or non-LRU. If LRU (Least Recently Used) is applied, then the oldest packet in the buffer is dropped to create space for the new one. On the other hand, if non-LRU is used and the buffer is full, then the new packet will be forwarded without caching. In addition, for TCP traffic from the wireless node, ELN-based Snoop is implemented.

##### B. XBC over Snoop

Although in the original design of Snoop, the buffer requirement at the base station is not evaluated, simulations show that the achievable throughput is affected by the buffer assigned to Snoop (see Fig. 5). When only the last hop of an end-to-end connection is a wireless link as in the scenario that the original Snoop was designed for, our measurement shows that a buffer with capacity of 10 packets is needed to ensure high throughput of a single TCP flow. However, the idea of in-network recovery can be easily extended to multi-hop wireless networks, which leads to higher buffer requirement due to

<sup>1</sup>This subsection can be skipped without lack of continuity by readers familiar with Snoop.

increased delay. As a result, to support a large number of TCP flows at the same time, the memory requirement at the base station will be huge.

We use Snoop as the baseline to demonstrate the benefit of XBC, and only consider the case where data traffic is from the wired-end to the wireless-end. We modified the Snoop code at the base station to support XOR encoding, and the TCP code at the wireless-end to support decoding. To simplify the implementation and limit the changes to the Snoop code, the data type of buffer slot is not modified. In *ns-2*, the data type is the class *Packet*, which contains pointers to header information and data. Without coding, different buffer slots refer to different packet data. When two or more packets are encoded together, each of their *Packet* structures still occupies one buffer slot. However, the data pointers of all these *Packet* structures refer to the same coded data. In other words, XOR coding is performed on packet data, not on packet headers.

To support XOR coding, a new packet header type *hdr\_xor* is introduced and appended to every XOR-ed packet, which is simply a bit map specifying which packet sequence numbers are combined in the packet. Since *hdr\_xor* is part of the header information, it can be accessed through the *Packet* structure. When two or more *Packets* in different buffer slots refer to the same XOR-ed packet, their *hdr\_xors* are also same. In addition, the receiver can examine the *hdr\_xor* of a received XOR-ed packet to assist in decoding.

At the base station, when a new data packet is received by the Snoop agent and there is no slot in the buffer for the packet, various strategies can be used to replace an old packet. If LRU is enabled, then the oldest packet in the buffer and all the packets XOR-ed with it are dropped so that one or more buffer slots are emptied and one of them can be used by the new packet. At the receiver, TCP receiver buffer caches the out-of-order raw packets received so far. However, some more buffer space is needed because to decode an XOR-ed packet, some old packets received in-order may be needed. Given coding degree  $d$  and coding distance  $l$ , to ensure decoding whenever possible, the extra buffer needed at the receiver equals to  $(d - 1) \times l$ .

1) *Cumulative ACKs vs. SACKs*: In Snoop, acknowledgments that identify the packet losses due to link error will usually be suppressed by the Snoop agent if the packet has been cached before to avoid unnecessarily invoking the congestion control mechanism at the sender. However, when XOR-coding is applied, lost packets may not be locally recoverable even if they are buffered. For instance, two or more raw packets that are coded together into a single packet can be lost on the wireless link. So the Snoop agent will recover the lost packets for up to three times before giving up.

In order to reduce the overhead of transmitting un-decodable packets, Selective acknowledgments (SACKs) can be used. It was shown in [10] that Snoop with SACKs can achieve better performance than pure Snoop when burst losses occur because SACKs provide more information than cumulative ACKs, which can be used by the Snoop agent to make smart decisions about recovery. While this kind of optimization is not

implemented in *ns-2*, XBC can still get benefit from SACKs, which is orthogonal to how Snoop uses SACKs. When a SACK is received, XBC checks whether the packets needed for decoding the lost one have also been lost. If that is the case, the SACK is forwarded to the sender and the corresponding coded packet is deleted from the buffer. However, SACKs can not completely eliminate unnecessary retransmissions as Snoop also uses timeout triggered retransmission. After a timeout at the Snoop agent, there is no information to determine whether the packet to be retransmitted can be decoded or not.

2) *Overhead analysis*: XBC incurs the overhead of coding and requirement of extra memory. Due to the simplicity of XOR coding, both the encoding and decoding operations are very efficient, which has been demonstrated by several practical works on XOR coding [18] [19] [25]. In addition, the extra packet header inserted into every coded packet is ignorable. Every coded packet is intended for recovering one lost packet whose sequence number has been included in the transport layer packet header. Therefore, the extra header only needs to specify which components have been combined together. Furthermore, since deterministic coding scheme is used, the receiver can easily figure out which packets are supposed to be coded with the lost one. As a result, there is only a two byte bitmap in the header, which is much smaller than a normal TCP data packet.

While the goal of XBC is to optimize the buffer usage on intermediate nodes so that more concurrent flows can be supported, XBC itself consumes some extra memory. First, it expands the code size on the base station. However, although XBC has not been implemented on real platforms at this stage, we predict that the code size can be very small because it is built upon an existing protocol supporting local recovery and only adds a few modifications for buffer management. Second, as discussed above, XBC requires a buffer on every receiver for decoding. We claim that such buffers are usually available on the nodes for caching out-of-order packets, and the extra buffer needed beyond this is limited as discussed above.

## V. EVALUATION

### A. Simulation Setup

In order to evaluate the benefit of XBC, we conducted simulations on *ns-2*. As stated above, XBC is implemented above the Snoop protocol with small modifications. Therefore, we first evaluated the existing Snoop implementation in *ns-2* with bounded buffer and demonstrated that buffer size does have impact on achievable TCP throughput. Then we compared the performance of Snoop with XBC and that without XBC under random losses and burst losses, respectively. Besides artificially introduced losses according to specific loss models, we also simulated the case where losses due to collision also exists in the network.

We simulated two scenarios: a simple three-node network where a wired link and a wireless link connected by a base station, and a grid network with 9 such wired-cum-wireless pairs. For each pair, FTP traffic from the wired end to the wireless end is applied and throughput measured at the

wireless end is the performance metric used. The data rates of wired and wireless links are 10Mbps and 2Mbps respectively, and the delay of the wired link is 50 ms, same as those used in [10]. All the experiments run FTP traffic for either 1000s (3-node network) or 500s (grid network) because loss models require a relatively long time to approach stability and the benefit of XBC relies on the loss model used. Such a long and data intensive TCP connection with downstream traffic is very common in practice, e.g. a wireless LAN user downloads a file of several megabytes through an FTP or P2P client.

Losses are introduced using *ns-2* error model, and only applied to the downstream traffic from base station to the wireless end. Wired link is reliable and so does the upstream traffic containing ACKs. We simulated both random losses and burst losses, and the loss rates used are commonly seen in reality. For random losses, the loss rates evaluated are from 1 error every 16Kb to 1 error every 4096Kb, which correspond to 40% of packet loss to 0.2% of packet loss, where TCP packet size is 1040 bytes. For burst losses, there is no packet loss in good state and the loss rates in bad state are from 1 error every 1Kb to 1 error every 1024Kb. In addition, mean inter-cluster size is 20 packets and error cluster size is 4 packets. The case where  $N_g = 20$  and  $N_b = 2$  was also simulated. The results are similar and omitted in this paper.

SACKs are enabled in all the following simulations. We also did simulations with SACK disabled, and found that with SACKs enabled, higher throughput can be achieved in all the cases simulated. Furthermore, MAC layer retransmission is disabled so that we can accurately study the performance under different loss rates. All nodes are static in our simulations. Losses due to mobility are unpredictable and challenging to model. Loss modeling for mobile scenarios and adaptive optimization of the coding strategy is part of future work.

### B. The Buffer Requirement of Snoop

Intuitively, the capacity of the retransmission buffer on a base station should be proportional to the congestion window size of the TCP flow going through the base station. However, the impact of constraints on available buffer has never been investigated before. In this section, we evaluate the performance of a single TCP flow going through a Snoop agent equipped with a limited retransmission buffer. The three-node topology is studied for random and burst losses.

From Fig. 5, it is easy to see that higher throughput can be achieved using a larger buffer in most cases. Only when loss rate is very high or very low, buffer size has little impact on the throughput. In addition, a buffer larger than 16 packets is not very helpful in most cases. The reason is that the average congestion window size is about 20 in this scenario, and the window size  $N$  for the link between the base station and the receiver is even smaller. An even larger buffer may actually diminish throughput. Therefore, the default buffer size in *ns-2* (100 packets) is not optimal in this scenario.

### C. Snoop with XBC: Single Base Station

1) *Random Losses*: In this section, XBC over Snoop is evaluated under different random loss rates. Due to the limited space, Fig. 6 only plots the results of three representative cases. From the figure, we make the following observations.

*Observation 1*: With decreasing loss rate, XBC can achieve benefit over a wider range of buffer sizes. Coding degree larger than 2 can only be helpful when loss rate is relatively low. This is consistent with our analysis in Section III.

*Observation 2*: When buffer capacity is less than 10 packets, the average throughput improvement by using XBC is about 20% under various loss rates, and can be as high as 30% for low loss rates. However, when buffer capacity is higher than 16 packets, XBC can not get benefit because the buffer capacity is close to the window size.

Therefore, XBC is most useful when loss rate is not very high and the available memory is limited, either due to resource constraints or due to multiple concurrent TCP flows going through the same base station. These graphs also show that the optimum coding degree depends on the loss rates.

2) *Burst Losses*: Fig. 7 plots the results of XBC over Snoop under different burst losses. Same loss rates and cluster sizes as the first experiment are used. According to Fig. 7, XBC can achieve 10%-20% higher throughput on average, when buffer capacity is less than 16 packets. We observe that the benefit of XBC increases when loss rate decreases, which is similar to the random loss case. In addition, coding distance 2 is better than distance 4 when buffer is small. The reason is that higher distance can actually decrease the chance of coding when the buffer is small, because to make use of the buffer space optimally, buffer size should be a multiple of the coding distance.

### D. Snoop with XBC: Multiple Base Stations

To further evaluate XBC, a  $3 \times 3$  grid network with 9 wired-cum-wireless pairs is used. The grid is deployed in a  $1000m \times 1000m$  2D area, with  $400m$  between every 2 base stations. Each wireless node is  $50m$  away from the corresponding base station. Since  $400m$  is less than the default carrier sensing range ( $550m$ ) and a single channel is used for all the pairs, all base stations experience collisions. The rate of collision is highest for the base station at the center of the grid. For all the base stations and wireless nodes, the transmission range is  $250m$ , and the size of interface queue is 50 packets.

Fig. 8 plots the average throughput for the 9 TCP flows. Similar random loss rates as before are used. The results for burst losses shows similar trends and are therefore omitted due to space limitation. From Fig. 8, we observe that while the throughput is lower compared with the case of single base station due to collisions, XBC can still achieve 5%-20% improvement.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an XOR coding based approach to optimize the retransmission buffers on intermediate nodes

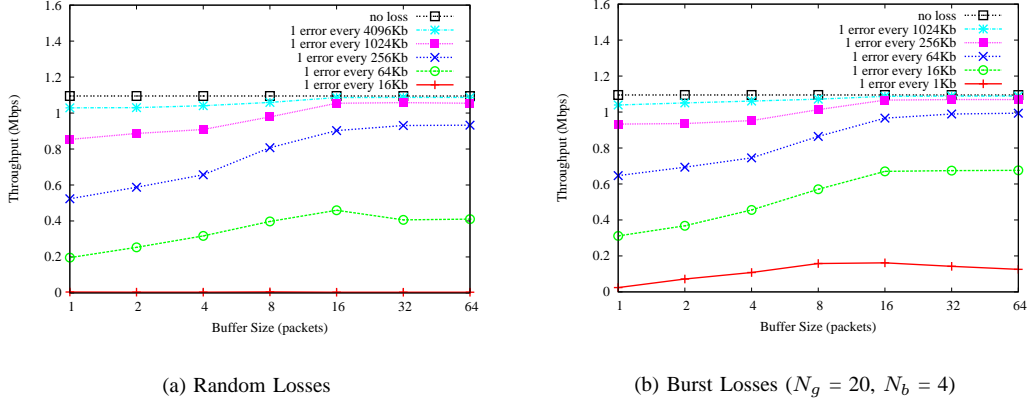


Fig. 5. Snoop performance under different buffer sizes; For burst losses, the bit error rates given are only for bad state since there is no error in good state;  $N_g$  is the mean size of good state and  $N_b$  is the mean size of bad state in terms of packets

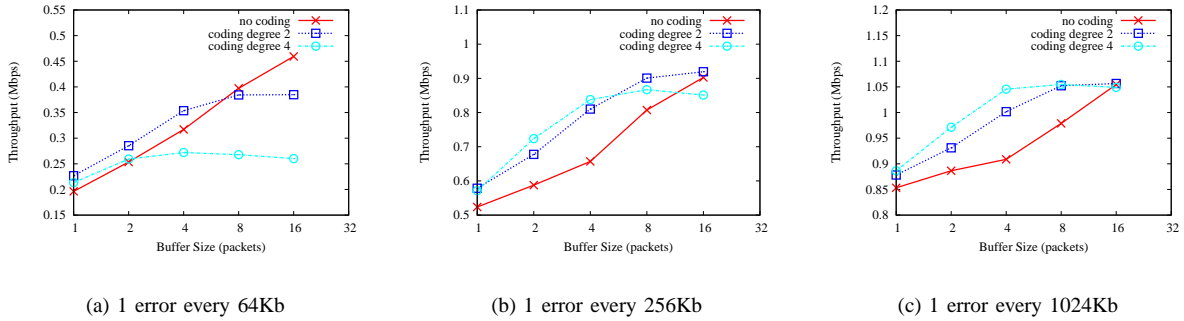


Fig. 6. Snoop performance under random losses with or w/o XBC

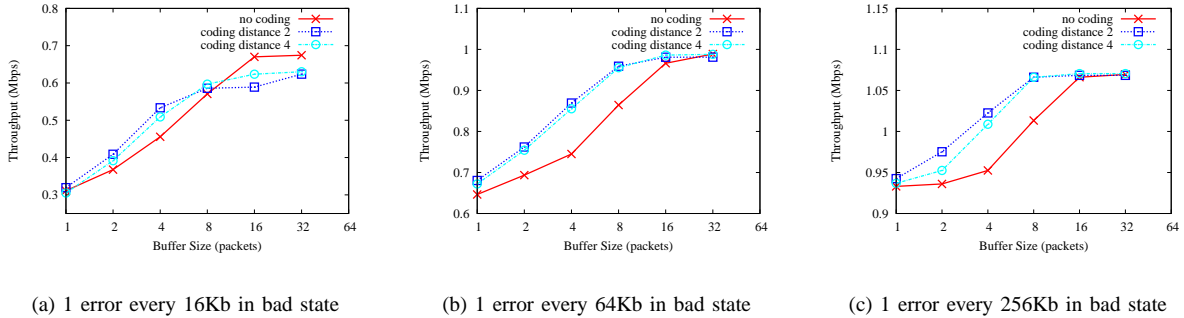


Fig. 7. Snoop performance under burst losses ( $N_g = 20, N_b = 4$ ) with or w/o XBC

that support in-network recovery. XBC's performance is analyzed with respect to two parameters, the coding degree and coding distance, for random as well as burst losses. It is then evaluated by using Snoop as the base-line. Simulations show that TCP throughput can be improved up to 20% in all the scenarios considered.

As part of future work, we plan to address a few key challenges that can further improve the applicability of XBC to real environments. First, while in this paper we assume that the packet loss in wireless networks can be well modeled and

the loss probability is fixed, in practice these may vary with time. Besides losses due to the wireless medium, packets can be lost due to collision and mobility. Closely modeling the losses is critical for high performance of XBC. In addition, for wireless losses, the loss probability is not fixed in reality due to the time-varying nature of channels. In the evaluation section we have observed that the optimum coding degree and coding distance are dependent on the loss characteristics. Thus, dynamic adaptation of these parameters is required for optimum performance of XBC. Second, the overhead of XBC

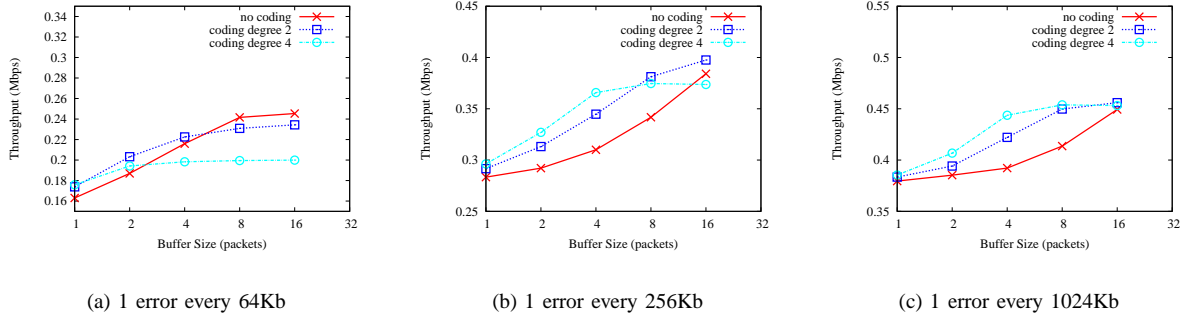


Fig. 8. Average throughput in a grid network with 9 TCP flows over 9 wired-cum-wireless pairs under random losses

in terms of coding operations and extra memory used has to be carefully evaluated in real environment. We plan to implement XBC on a resource limited platform while optimizing the code size.

## VII. ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grants CNS-0546630 (CAREER Award) and CNS-0403342. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## REFERENCES

- [1] A. Adinegara, W. Lau, and K.-W. Chin, "InPCM: A Network Cache Technique for Improving the Performance of TCP in Wireless Ad-Hoc Networks," in *IEEE Wireless Telecommunications Symposium (WTS)*, Pomona, CA, USA, Apr. 2006.
- [2] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP Performance over Wireless Networks," in *Proc. 1st ACM Conf. on Mobile Computing and Networking*, Berkeley, CA, USA, Nov. 1995.
- [3] T. Braun, T. Voigt, and A. Dunkels, "Energy-Efficient TCP Operation in Wireless Sensor Networks," *PIK Journal Special Issue on Sensor Networks*, vol. 28, no. 2, 2005.
- [4] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, Atlanta, Georgia, USA, Sept. 2002.
- [5] S. Kim, R. Fonseca, and D. Culler, "Reliable Transfer on Wireless Sensor Networks," in *The First IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON)*, Santa Clara, CA, USA, Oct. 2004.
- [6] p. a. IEEE 802.11 WG., "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *Standard Specification*, IEEE, 1999.
- [7] H. Balakrishnan and R. Katz, "Explicit Loss Notification and Wireless Web Performance," in *IEEE Globecom Internet Mini-Conference*, Sydney, Australia, Nov. 1998.
- [8] G. Holland and N. H. Vaidya, "Analysis of TCP Performance Over Mobile Ad Hoc Networks," in *Proceedings of IEEE/ACM MOBICOM '99*, Aug. 1999, pp. 219–230.
- [9] W.-P. Chen, Y.-C. Hsiao, J. C. Hou, Y. Ge, and M. P. Fitz, "Syndrome: A Light-Weight Approach to Improving TCP Performance in Mobile Wireless Networks," *Wireless Communications and Mobile Computing Journal*, vol. 2(1), pp. 37–57, Feb. 2002.
- [10] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, 1997.
- [11] V. Anantharaman, S. Park, K. Sundaresan, and R. Sivakumar, "TCP Performance over Mobile Ad-hoc Networks: A Quantitative Study," *Wireless Communications and Mobile Computing Journal, Special Issue on Performance Evaluation of Wireless Network*, vol. 4, no. 2, pp. 203–222, 2004.
- [12] L.-J. Chen, T. Sun, M. Y. Sanadidi, and M. Gerla, "Improving Wireless Link Throughput via Interleaved FEC," in *Proceedings of the Ninth International Symposium on Computers and Communications (ISCC)*, 2004, pp. 539–544.
- [13] C. Barakat and E. Altman, "Bandwidth Tradeoff between TCP and Link-Level FEC," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 39, no. 5, pp. 133–150, 2002.
- [14] A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer, "Understanding Link-Layer Behavior in Highly Congested IEEE 802.11b Wireless Networks," in *SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis (E-WIND)*, Philadelphia, PA, USA, Aug. 2005.
- [15] V. Naik, A. Arora, P. Sinha, and H. Zhang, "Sprinkler: A Reliable and Energy Efficient Data Dissemination Service for Extreme Scale Wireless Networks of Embedded Devices," *IEEE Transactions on Mobile Computing (TMC)*, vol. 6, no. 7, pp. 777–789, July 2007.
- [16] A. Sobeih, H. Baraka, and A. Fahmy, "ReMHoc: A Reliable Multicast Protocol for Wireless Mobile Multihop Ad Hoc Networks," in *Consumer Communications and Networking Conference (CCNC)*, Jan. 2004, pp. 146–151.
- [17] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.
- [18] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in The Air: Practical Wireless Network Coding," in *SIGCOMM '06*, Pisa, Italy, 2006.
- [19] L. E. Li, R. Ramjee, M. Buddhikot, and S. Miller, "Practical and Efficient Broadcast in Mobile Ad hoc Networks," in *IEEE INFOCOM*, Anchorage, Alaska, USA, 2007.
- [20] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, 2003.
- [21] J. Ebert and A. Willig, "A Gilbert-Elliott Bit Error Model and the Efficient Use in Packet Level Simulation," in *TKN Technical Reports Series of Technical University Berlin*, Mar. 1999.
- [22] R. K. Ganti, P. Jayachandran, H. Luo, and T. Abdelzaher, "Datalink Streaming in Wireless Sensor Networks," in *Proceedings of ACM SenSys*, Boulder, Colorado, USA, 2006.
- [23] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to Probability*. Athena Scientific, June 2002.
- [24] "The network simulator - ns-2," <http://www.isi.edu/nsnam/ns/>.
- [25] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth Codes: Maximizing Sensor Network Data Persistence," in *SIGCOMM '06*, Pisa, Italy, 2006.