Computer Communications xxx (2008) xxx–xxx

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Buffer Coding for Reliable Transmissions over Wireless Networks ☆

Zizhan Zheng *, Prasun Sinha

Department of Computer Science and Engineering, Ohio State University, Room 395, Dreese Labs, 2015 Neil Avenue, Columbus, OH 43210, USA

ARTICLE INFO

Article history:
Received 5 February 2008
Received in revised form 25 September 2008
Accepted 25 September 2008
Available online xxxx

Keywords:
Wireless communication
Algorithm/protocol design and analysis

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 WLANs. We further propose a *self-adaptive* algorithm that tunes coding degree on the fly based on the measured coding behavior and packet loss probability. Using simulations in *ns-2*, we observe that in our simulation settings, when the size of the retransmission buffer in 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.

1. 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 a challenging problem, 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 [2–5]. 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 [6,7]. 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. Notice that P_i can also be recovered from $(P_i \oplus P_j) \oplus P_j$ as well, so that storing $P_i \oplus P_j$ allows recovering both packets given that only one is lost. Decoding of XOR-ed 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 [3], 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

☆ An earlier version of this paper appeared at IEEE BROADNETS '07 [1].

* Corresponding author. Tel.: +1 6143123013.
E-mail addresses: zhengz@cse.ohio-state.edu (Z. Zheng), prasun@cse.ohio-state.edu (P. Sinha).

0140-3664/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2008.09.033

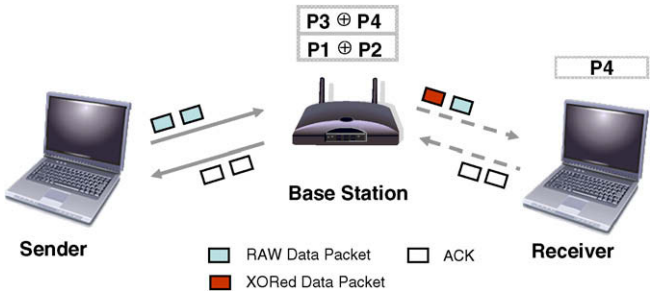

Fig. 1. A simple example of how XOR coding optimizes the retransmission buffer on the intermediate node (the base station in this case). The base station caches XOR coded packets to save buffer while ensuring high chance of local recovery. For instance, assume that packet P_1, P_2, and P_4 have been received and packet P_3 is lost at the receiver, then the receiver only needs to cache P_4. P_3 can be recovered by transmission of $P_3 \oplus P_4$ from the base-station.

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 the 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 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 explores the well-known interleaving technique used in channel codification by allowing 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 [3]. We choose Snoop as the baseline because it is well known and readily available for use for both simulations and experiments. Note that XBC can be easily implemented as a plug-in to other reliable protocols that support 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 3, we present the design of XBC and analyze its benefit for networks with random or burst losses. Section 5 describes the implementation of XBC on Snoop and Section 7 evaluates the performance of Snoop with or without XBC. Finally in Section 8, we conclude our work and give pointers for future research.

2. 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.

2.1. Transport layer recovery

To support reliable communication over wireless networks, both end-to-end approaches [8–10] and in-network enhancements [3,11,12] 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).

2.2. Link-layer recovery

A link-layer protocol supporting local recovery may simply perform retransmissions without the knowledge of the transport layer so that it can be used by different higher layer protocols, as in the case of 802.11 MAC [7]; or it can smartly use that knowledge to achieve better throughput, as in the case of Snoop [11]. Extensions of Snoop to multi-hop wireless networks have also been suggested [2,4]. 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 [4], 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 [2] 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 the retransmission buffer on the achievable benefit of in-network recovery, which is the focus of this paper.

FEC-based protocols [13,14] achieve reliability by adding redundancy to the transmitted packets. Such mechanisms could be used at the sender or the intermediate nodes. As no acknowledgements are needed, FEC-based protocols are buffer-free and incur low latency. However, optimal FEC coding highly depends on the error nature of networks and cannot be easily achieved for wireless networks due to the unpredicable channel quality [15].

2.3. In-network recovery in broadcast protocols

The idea of local recovery is not restricted to unicast, but also common in reliable broadcast and multicast procotols [5,16,17]. 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 [16] is also a reliable broadcast protocol that uses the nodes in a connected dominating set for distribution of data and recovery of lost packets.

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

ReMHoc [17] 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.

2.4. Network coding

As a simple way to put the elegant idea of network coding [18] 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 [19,20]. 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 [21] 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 [19] or CODEB [20]. 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 the number of transmissions, but to reduce the number of end-to-end retransmissions by efficient use of buffers.

3. XOR-based buffer coding (XBC): design and analysis

3.1. Overview

XBC operates over 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 such intermediate node 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, say $P_1 \oplus P_2 \oplus \ldots \oplus P_n, n \geqslant 1$. The node then replaces the packet with $P \oplus P_1 \oplus P_2 \oplus \ldots \oplus P_n$. 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 [3], 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 5.2 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 5.2). Assume the receiver receives an XOR-ed packet $P(s_j) = P_1(s_1) \oplus P_2(s_2) \oplus \ldots \oplus P_n(s_n)$, 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 $(1 \leqslant j \leqslant n)$ in this case. The receiver searches for each coded component in its local buffer. If all of them except P_j are available, the receiver retrieves P_j by computing $P \oplus P_1 \oplus P_2 \oplus \ldots P_{j-1} \oplus P_{j+1} \ldots \oplus P_n$.

Notice that in the above scenario if more than one packets in $P_1, \ldots, P_n$ are lost, none of them can be recovered by using only P and those received packets. This introduces two types of overhead. First, the buffer utilization is reduced. Second, the intermediate node may retransmit P several times before giving up, thus, wasting bandwidth. Although it is impossible to avoid this scenario completely, we apply the following two strategies to reduce its impact. First, coding parameters are carefully selected to maximize the chance of local recovery, which is the main topic of this section. Second, the intermediate node makes use of the extra information carried by the acknowledgements from the receiver to learn of undecodable packets so that these packets can be removed from its buffer and the source can be informed to retransmit them as early as possible, which is discussed in detail in Section 5.

3.2. 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 $N \geqslant 2$.

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, we have $q_{xbc} = (1-p) \times 2/N$.

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.

3.3. 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 $K \geqslant 1$. Second, to get more benefit from XOR coding, we consider a general coding degree $d \geqslant 2$. 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

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

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,

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

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

Given window size N, the benefit of XBC depends on the buffer size K. We consider the following three cases: (a) $K \geqslant N$; (b) $K \leqslant N/d$; and (c) $N/d < K < N$. In the first case, coding has no benefit since all the raw packets can be cached in the buffer. In the second case, $q_{xbc}/q_{raw} = d \times (1-p)^{d-1}$. To see the benefit of XBC for different loss rates and code degrees, we plot q_{xbc}/q_{raw} 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 the increased chance of multiple losses within a coded packet. Finally, in the third case, $q_{xbc}/q_{raw} = N/K \times (1-p)^{d-1} < d \times (1-p)^{d-1}$. Therefore, Fig. 2 actually shows the maximum possible benefit of XBC.

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. Although this simplified and restrictive implementation loses optimality in comparison to ideal cases shown in Fig. 2, it still performs better than the case where coding is not used as shown in our evaluations (Section 7).

3.4. 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 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 [22] for single link can be extended to a multi-hop path. In particular, 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 [23].

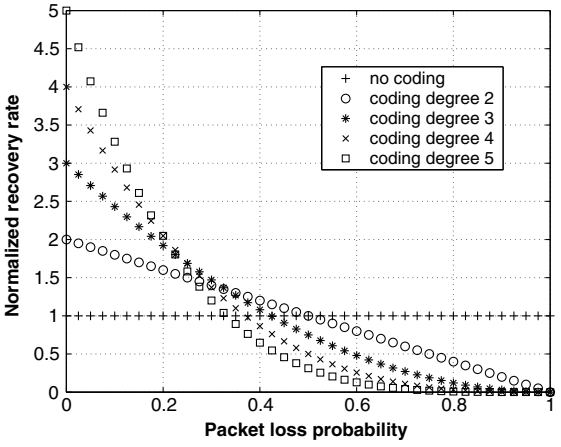

Fig. 2. Normalized packet recovery probabilities under random losses.

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 explores the concept of coding distance to allow the packets separated by a fixed interval of sequence numbers 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: $P = P_s \oplus P_{s+l} \oplus \ldots \oplus P_{s+(d-1)\times l}$, where s, $s+l$, etc., are sequence numbers of the packets being coded and $s \bmod (d \times l) \in [0 \ldots l-1]$. Although such a fixed coding format sacrifices certain coding opportunities, it simplifies both the encoding and decoding processes. Table 1 gives the snapshots of a sender buffer with a capacity of 8 packets, with specified coding degree d and coding distance l.

3.4.1. XBC for an unbounded buffer

To see the impact of coding distance in a real setting, we first assume that the size of the sender buffer is unbounded, and calculate the recovery probabilities of lost packets for the five loss models derived in Seda [23] based on real experiments (see Table 2).

In our calculation, a sequence of bits is first generated for each loss model, and then each bit is marked as either in-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 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 that 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, single retransmission of lost data packets, 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.

Table 1
Examples of XBC coding.

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

Table 2
Loss models [23].

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

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

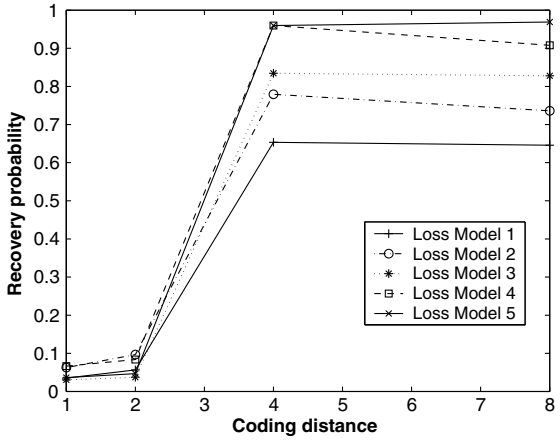

Fig. 3. Packet recovery probabilities under burst losses for unbounded buffer.

3.4.2. XBC for a bounded buffer

Given a buffer of size K, 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 the buffer can be fully utilized 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 the good state to the bad state, and p_{bg} is the probability from the bad state to the 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

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

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

where Q is the probability that all the other packets that P are encoded with are received given that P is lost. Eq. (3) holds because q_{raw} is independent of the type of losses. Suppose $d = 2$. 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 the good state after l steps given that it is in the bad state now. This l-step transition probabilities of the Markov chain [24] can be easily obtained:

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

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

Therefore, for coding degree 2,

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

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

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.

The above model can be simplified by assuming $p = 1$, that is, in the bad state, all the packets are lost. In this case, burst losses can be characterized by average burst error length (ABEL), which is N_b, and frame error rate (FER) ϵ [25]. We then have:

$$\epsilon = \frac{a}{a + b} \tag{9}$$

$$Q = \frac{b - b \times (1 - a - b)^l}{a + b} = (1 - \epsilon) \times \left(1 - \left(1 - \frac{1}{N_b \times (1 - \epsilon)}\right)^l\right) \tag{10}$$

Fig. 5 plots $q_{xbc}/q_{raw} = 2Q$ under different coding distances using this simplified model.

4. Self-adaptive XOR coding

As we can see from the analytical results in the previous section and the simulation results in Section 7, at any given time, the optimal coding scheme, that is the best coding degree and coding distance, depends on both the available buffer capacity and the networking environment, especially link quality and window size. Therefore, instead of using a fixed coding scheme all the time, a mature solution should be able to adjust coding scheme on the fly according to current environmental parameters. The two key challenges are how to measure the critical parameters accurately and how to determine the optimal coding scheme quickly. In this section, we describe a simple *self-adaptive* coding algorithm (see Algorithm 1).

Algorithm 1. Tuning coding degree online

```
1:  /* err: sampled packet loss probability; serr: average packet loss probability;
    rec: sampled local recovery probability; srec: average local recovery
    probability;
    cdr: current coding degree, pdr: previous coding degree */
2:  compute err and rec;
3:  errvar ← |err-serr|/serr;
4:  if state = 0 then
5:    if errvar ⩾ t1 then
6:      state ← 1;
7:      schedule the next invocation after s1 seconds;
8:      return;
9:    if state = 1 then
10:     if errvar ⩾ t1 then
11:       state ← 2;
12:     else
13:       state ← 0;
14: if state = 2 then
15:   /* koN = K/N in Eq. (2) */
16:   koN = rec/((1 − err)^(cdr−1) × cdr)
17:   compute the maximal local recovery probability nrec and the
    corresponding coding degree d using Eq. (2);
18:   if nrec ⩾ (1 + t2) ∗ rec then
19:     pdr ← cdr;
20:     UpdateCode(d);
21:     state ← 3;
22:     schedule the next invocation after s2 seconds;
23:     return;
24:   else
25:     state ← 0;
26: if state = 3 then
27:   if rec < (1 − t3) ∗ srec then
28:     UpdateCode(pdr);
29:   state ← 0;
30: update serr and srec;
31: schedule the next invocation after s seconds.
```

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

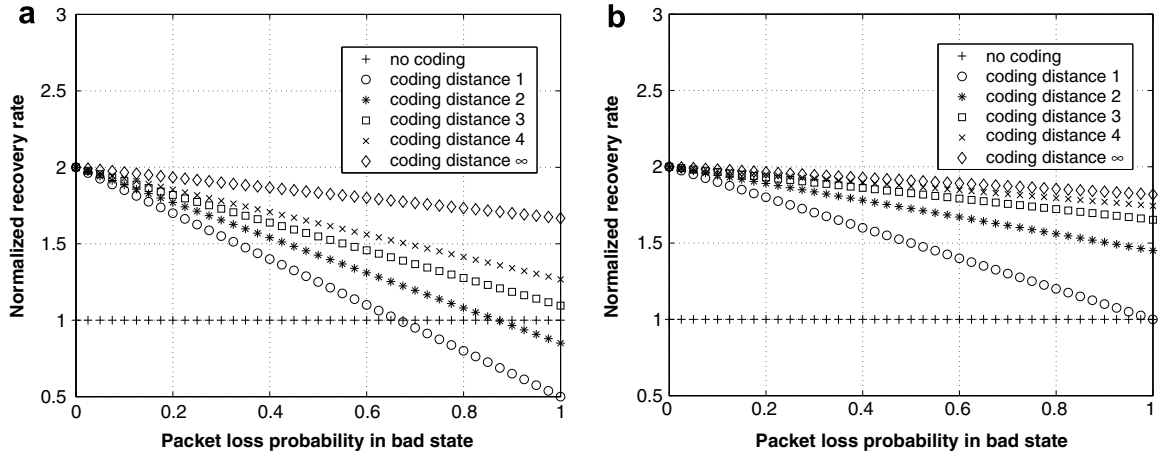

Fig. 4. Normalized packet recovery probabilities under burst losses (coding degree = 2). (a) $N_g = 20, N_b = 4$; (b) $N_g = 20$, $N_b = 2$.

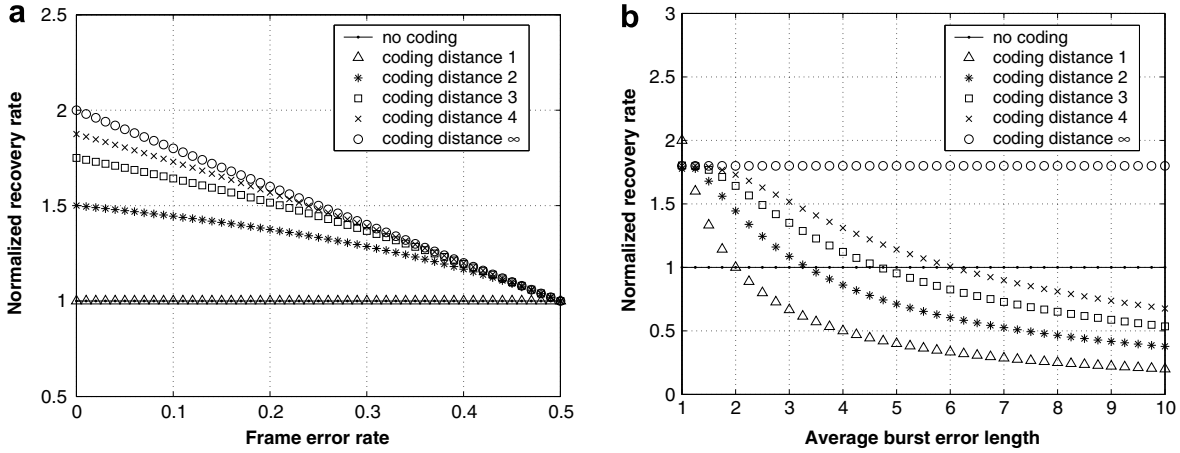

Fig. 5. Normalized packet recovery probabilities under burst losses (coding degree = 2) using the simplified 2-state model. (a) *ABEL* = 2 and (b) *FER* = 0.1.

The algorithm assumes that packet loss can be characterized by a single loss probability, which may vary over time, and the buffer capacity is fixed. The routine is invoked after a snoop agent is initialized, and invoked again every s seconds (line 31) in the stable state. Once invoked, the routine first computes the packet loss probability *err* and local recovery probability *rec* since last invocation (line 2). *err* is computed as $1 - n_2/n_1$, where n_1 is the number of packets cached during the last sampling period, and n_2 is the number of cached packets that are acknowledged by the destination before retransmission. *rec* is computed as m_2/m_1, where m_1 is the number of duplicated acknowledgements received during the last sampling period, and m_2 counts the number of times a required packet can be found in the buffer. Notice that only those packets that have not been retransmitted are counted in m_1 and m_2.

The routine then compares the loss probability (*err*) measured in the last sampling period and the average loss probability (*serr*) before that (see lines 3–13). If the difference is larger than a threshold t_1 and is stable for s_1 seconds, then a new coding degree is determined. To find the optimal coding degree, Eq. (2) is applied. The main challenge here is that the window size N can not be easily measured. Therefore, instead of measuring N directly, K/N is computed using measured local recovery probability (see line 16). An optimal coding degree is then founded using Eq. (2). If the potential performance gain is high enough, the new coding degree is applied (see lines 18–23). However, since the new coding degree is computed based on the theoretical model with several assumptions, and the measurements of the parameters may not be accurate enough, it is possible that the new degree is not better than the old one. If this does happen, the old degree is retrieved (see lines 26–29).

The update of coding degree is completed by the routine *UpdateCode*, which involves two steps. First, the buffer has to be reset. Although it is possible to combine the old packets together when the new degree is a multiple of the old one, the benefit is small since the buffer size is usually small. Therefore, the buffer is simply cleaned in the current implementation. Second, the snoop agent has to inform the TCP sink the new coding degree. To implement this, the new degree is inserted into the XOR header of the first several data packets forwarded by the Snoop agent after the new degree is applied, until a cached packet is acked, because the sink must have known the new coding degree at that time.

5. Implementation of XBC over Snoop

We have implemented XBC over the Snoop protocol [3] in the *ns – 2* simulator [26]. We first give a brief description of Snoop[1], and its implementation in ns – 2 followed by the implementation details of XBC.

[1] This subsection can be skipped without lack of continuity by readers familiar with Snoop.

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

5.1. 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 [3] or set the Explicit Loss Notification (ELN) bit [8] 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.

5.2. 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 throughout is affected by the buffer assigned to Snoop (see Fig. 6). 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 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 of type *hdr_xor* is inserted into every XOR-ed packet, which is simply a bit map specifying the sequence numbers of the encoded packets. 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_xor*s 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 extra 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 in terms of the number of packets equals to $(d-1)\times l$.

5.2.1. 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. The Snoop agent will retransmit the lost packets for up to three times (the default setting in ns-2) before giving up.

In order to reduce the overhead of transmitting un-decodable packets, Selective acknowledgments (SACKs) can be used. It was shown in [11] 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 cannot completely eliminate unnecessary retransmissions as

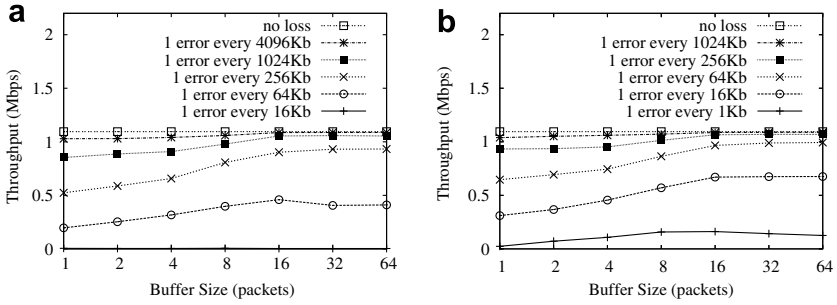

Fig. 6. Snoop performance under different buffer sizes. For burst losses, the bit error rates in the bad state are given; N_g is the mean size of good state and N_b is the mean size of bad state in terms of packets. (a) Random losses and (b) burst losses ($N_g = 20, N_b = 4$).

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

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.

5.2.2. 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 [19,20,27]. 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. 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.

6. Discussion

6.1. Loss models

The performance of XBC is closely related to the loss pattern that a real wireless network experiences. In this paper, the Gilbert model has been used to model the burst packet losses in a wireless network for both the analysis and simulation purposes. Although more accurate models with higher complexity have been proposed, we use the Gilbert model for the following reasons.

(1) The Gilbert model is one of the most commonly used models for network simulations. The model simplifies the analysis significantly, and is directly supported by ns-2, the simulator we use to evaluate the performance of XBC. On the other hand, although many models with various accuracy and complexity have been proposed, there is no common agreement on which model is the best choice even for a specific wireless network standard, such as 802.11b.
(2) It has been shown in [28] that the simple Gilbert model captures the packet loss patterns of 802.11b at 2 and 5.5 Mbps data rates quite well. In a more recent work [29] that compares the accuracy of the two classical Markov models, the Gilbert model and the Hidden Markov model (HMM), and two Markov-based trace analysis algorithms, it is shown that for the two WLAN traces evaluated, the Gilbert model captures error burst CDF well and performs much better than HMM, but none of the evaluated models can capture error-free burst CDF accurately. Furthermore, when the packet loss probability in the lossy state is not very high ($\leqslant 0.4$), the Gilbert model performs best for a large portion of the parameters evaluated.

We will analyze and evaluate the performance of XBC under other loss models in the future work.

6.2. The Applicability of XBC

Although we use XBC over Snoop as a case study in this paper, XBC can be used to improve the performance of other reliable protocols, as long as the following conditions are satisfied: (1) in-network caching is used to provide local recovery. (2) Acknowledgements are used to notify the reception of packets. (3) The window size of an intermediate node with respect to the receiver should be larger than the buffer size. The first two conditions are needed for the implementation of XBC and the third condition ensures that there is space for XBC to improve performance. To reduce the overhead of XBC, the following condition is preferred; and (4) The receiver side uses a buffer to achieve in-order packet delivery.

6.3. Beyond XOR coding

In this section, we discuss the possibility of using a general linear code instead of XOR coding to optimize buffer usage. Specifically, we consider the case of Reed-Solomon code (RScode) and use it in a similar way shown in [20]. 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 Θ in finite field $\mathscr{F}_{2^8}$ (byte level coding). Let $\mathbb{P}$ be the set 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 ($M \leqslant K$) 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 follows. Let $\mathbb{R}$ be the $N - K$ raw packets cached by the receiver, and Λ 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.

7. Evaluation

7.1. 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 exist in the network. After that, several mobile scenarios are evaluated. Finally, the adaptive coding algorithm is evaluated on a static network.

In all the simulations, FTP traffic from the wired end to the wireless end is applied and throughput measured at the wireless end is the performance metric used. SACKs are enabled in all the following simulations. We also did simulations with SACK disabled, and found that with SACKs enabled, higher throughout can

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

be achieved in all the cases simulated. In addition, non-LRU is used as the caching strategy in all the simulations. For all the base stations and wireless nodes, the transmission range is 250 m, and the size of interface queue is 50 packets.

Losses are introduced using ns-2 error model, and only applied to the downstream traffic from base stations to wireless nodes. Wired links are reliable and so is the upstream traffic containing ACKs. MAC layer retransmission is disabled for downstream traffic so that we can accurately study the performance under different loss rates. 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 16 kb to 1 error every 4096 kb, 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 1 kb to 1 error every 1024 kb. In addition, mean inter-cluster size is 20 packets and mean 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.

7.2. XBC over Snoop: static scenarios

We simulated two static 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. The data rates of wired and wireless links are 10 and 2 Mbps, respectively, and the delay of the wired link is 50 ms, similar to those used in [11]. RTS/CTS is enabled in these simulations.

All the experiments run FTP traffic for either 1000 s (3-node network) or 500 s (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 WLAN user downloads a file of several megabytes through an FTP or P2P client.

7.2.1. 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.

From Fig. 6, 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.

7.2.2. Single base station

7.2.2.1. Random losses. In this section, XBC over Snoop is evaluated under different random loss rates. Fig. 7 plots the results of two 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 3.3.

Observation 2. For small buffer sizes, 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. When buffer size is relatively large such that it is close to the window size, such as 16 in Fig. 7, XBC cannot get benefit, which is also consistent with our analysis in Section 3.3.

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.

7.2.2.2. Burst losses. Fig. 8 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. 8, 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.

7.2.3. Multiple base stations

To further evaluate XBC, a 3×3 grid network with nine wired-cum-wireless pairs is used. The grid is deployed in a 1000 m $\times$ 1000 m 2D area, with 400 m between every 2 base stations. Each wireless node is 50 m away from the corresponding base station. Since 400 m is less than the default carrier sensing range (550 m) 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.

Fig. 9 plots the average throughput for the nine TCP flows. Similar random loss rates as before are used. The results for burst losses show similar trends and are therefore omitted. From Fig. 9, 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.

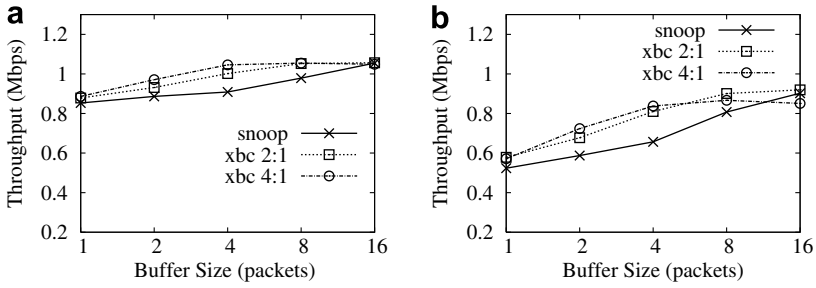

Fig. 7. Average TCP throughput under random losses with or w/o XBC: *snoop* means XOR coding is not applied; *xbc 2:1* means XOR coding is applied with coding degree 2 and coding distance 1. (a) 1 error every 1024 Kb and (b) 1 error every 256 Kb.

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

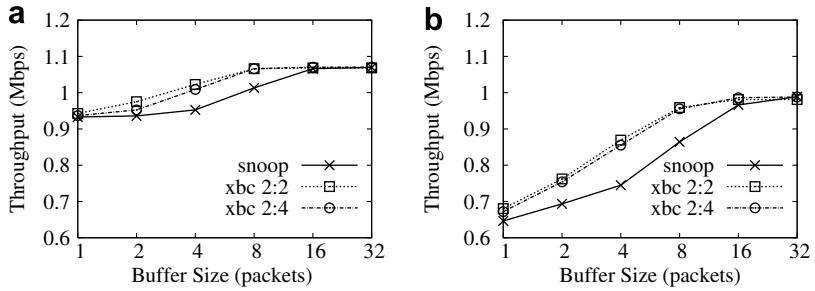

Fig. 8. Average TCP throughput under burst losses with or w/o XBC. The bit error rates in the bad state are given. $N_g = 20, N_b = 4$. (a) 1 error every 256 Kb and (b) 1 error every 64 Kb.

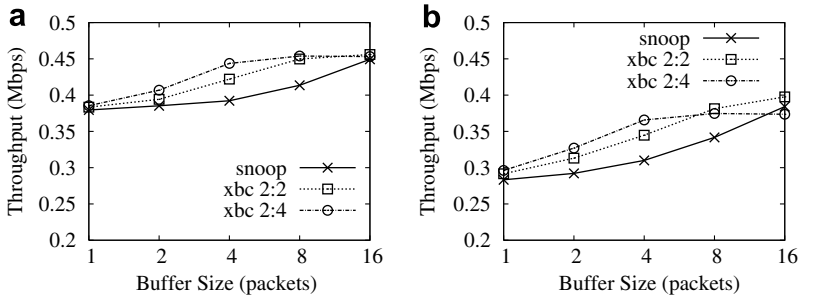

Fig. 9. Average TCP throughput in a grid network with 9 TCP flows over nine wired-cum-wireless pairs under random losses. (a) 1 error every 1024 Kb and (b) 1 error every 256 Kb.

7.3. XBC over Snoop: mobile scenarios

To further evaluate the performance of XBC over Snoop, mobility is introduced. In the following simulations, 16 base stations are deployed regularly forming a 4×4 grid, while M mobile nodes move randomly between base stations. All the base stations share a single channel. Two wired nodes W1 and W2 are connected, and W2 is further connected with every base station through wired links. For all the wired links, the data rate is $(M \times 10)$ Mbps, and the delay is 25 ms. The data rate of wireless links is 5 Mbps.

Mobile IP is used in all the experiments. Each mobile node is assigned one base station as its home agent. To reduce overhead, instead of using a full fledged ad hoc routing protocol, a simple routing agent is set up to forward packets between a mobile node and its foreign agent. One FTP flow is set up from W1 to each mobile node. All the experiments run FTP traffic for 500 s.

In mobile scenarios, multiple types of losses may happen. Besides artificially introduced random bit error rate in the downstream links, which is fixed to 1 error every 1024 Kb in the follow simulations, losses due to collision also exist since multiple mobile nodes share the same channel. In addition, handover may introduce some losses. RTS/CTS is disabled in all the mobile scenarios.

In the following subsections, we vary each of the following three parameters while keeping the other two: (1) the number of mobile nodes; (2) the moving speed of mobile nodes; (3) the density of base stations. For each case, five moving scenarios are generated randomly, and the average results are given. The moving speed is uniformly distributed within certain range and the pause interval is 10 s.

7.3.1. Varying the number of mobile nodes

In this section, 16 base stations are deployed in a 1000 m $\times$ 1000 m network, with 200 m between adjacent base stations. Mobile nodes randomly move within the network, with their speeds varying within [10 m/s, 20 m/s] uniformly. Fig. 10 gives the average throughput when varying the number of mobile nodes. From the figure, we can see that the throughput decreases as more mobile nodes introduced, due to the fact that more interference occurs and one base station may have to serve multiple mobile nodes simultaneously. However, XBC with coding degree 2 and 3 performs better than Snoop in almost all the cases.

7.3.2. Varying the moving speed of mobile nodes

In this section, similar network scenario as above is deployed, while the number of mobile nodes is fixed to 4. The moving speed of every mobile node varies within $(s, s + 10]$ (m/s) uniformly and independently, where s is set to 0, 10, 20, and 30 in turn. Fig. 11 gives the average throughput under different speeds. From the figure, we can see that the results are very similar in all cases, and XBC with coding degree 2 and 3 performs better than Snoop in almost all the cases.

7.3.3. Varying the density of base stations

In this section, 16 base stations are deployed in a $x \times x$ (m^2) network, with $x/4$(m) between adjacent base stations, where x is selected from 200, 400, 800, 1200 in turn. 4 mobile nodes randomly move within the network, with their speeds varying within [10 m/s, 20 m/s] uniformly. Fig. 12 gives the average throughput when varying the density of base stations. From the figure, we can see that the throughput increases when density decreases due to the fact that less interference occurs in a sparser network. Again, XBC with coding degree 2 and 3 performs better than Snoop in almost all the cases.

7.4. Random deployment of base stations

So far we have evaluated XBC under grid topology. In this section, we consider a 1000 m $\times$ 1000 m area with 30 base stations randomly deployed as follows. The entire area is first divided into 50 m $\times$ 50 m squares. Each base station is then deployed at the center of a randomly selected square. At most one base station can be deployed within any square. A single mobile node moves within the area with its speed uniformly distributed in [10 m/s,

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

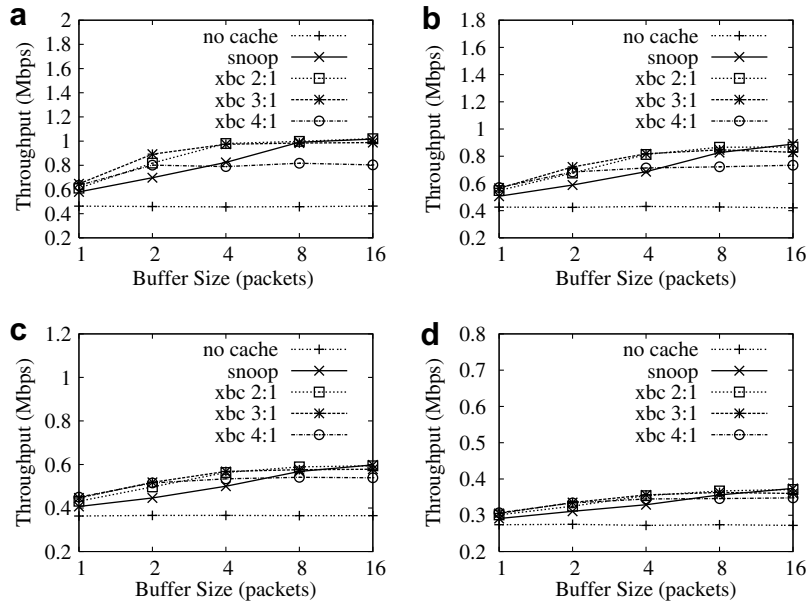

Fig. 10. Average TCP throughput under various number of mobile nodes: *no cache* means there is no in-network caching and retransmission. (a) 1 MH, (b) 2 MHs, (c) 4 MHs, and (d) 8 MHs.

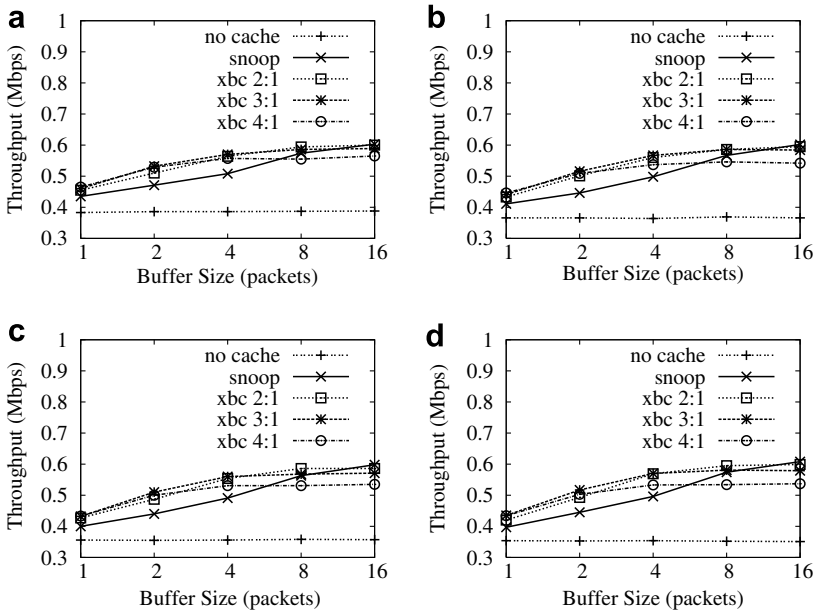

Fig. 11. Average TCP throughput under various moving speeds of mobile nodes. (a) s = 0, (b) s = 10, (c) s = 20, and (d) s = 30.

20 m/s]. The data rate of wireless links is 5.5 Mbps. Other settings are same as the previous section. Five random deployments and five moving scenarios are generated and the average results for the 25 combinations are given. Fig. 13 shows that XBC can achieve similar improvement over Snoop as in the case of grid topology, that is, the average TCP throughput is improved by up to 30% for random loss and up to 20% for burst loss.

7.5. Tuning coding degree on the fly

In this section, the adaptive coding algorithm is evaluated on the single base station scenario used before while varying the bit error rate over time. The simulation is run for 1000 s. At the beginning of every 250 s, the bit error rate is set to a new value: 1 error every y Kb, where y is 1024, 256, 64, and 32 in turn. The parameters used in Algorithm 1 are assigned following values: $s = 5$ s, $s_1 = s_2 = 20$ s, $t_1 = 0.4, t_2 = t_3 = 0.05$. The initial coding degree is set to 1, that is, no XOR coding is applied initially. The coding distance is fixed to 1. Fig. 14 shows the average throughput for different coding degrees. We can see that the adaptive coding protocol can achieve optimal or near optimal performance for different buffer sizes, while a fixed coding degree only works well for certain buffer sizes.

8. Conclusions and future work

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

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033

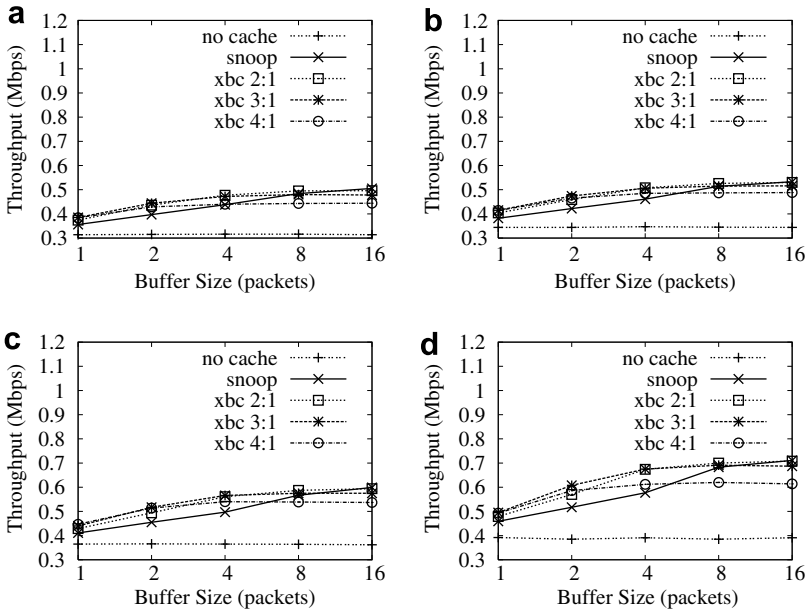

Fig. 12. Average TCP throughput under various densities of base stations. (a) 200 × 200, (b) 400 × 400, (c) 800 × 800, and (d) 1200 × 1200.

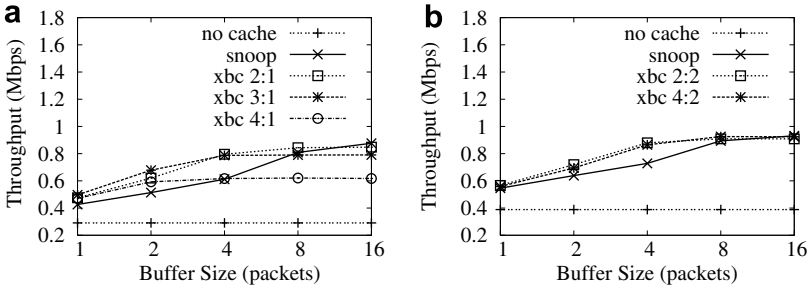

Fig. 13. Average TCP throughput under random deployment. (a) 1 error every 256 Kb. (b) One error every 64 Kb in the bad state, $N_g = 20, N_b = 4$. (a) Random losses and (b) burst losses.

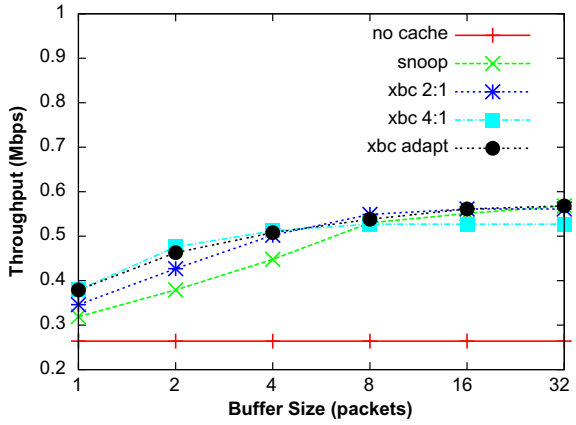

Fig. 14. Average TCP throughput in a single wired-cum-wireless network where packet loss rate changes over time.

support in-network recovery. XBC's performance is analyzed with respect to two parameters, coding degree and coding distance, for random as well as burst losses. Based on the analytical model, a simple adaptive algorithm that tunes coding degree on the fly according to packet loss probability is proposed. XBC 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, the two-state Markov model used in the paper is inadequate to characterize various type of losses in real wireless networks. Besides losses due to poor link quality, packets can be lost due to collision and mobility. Closely modeling the losses is critical for high performance of XBC. In addition, the loss probability is not fixed in reality due to the time-varying nature of wireless channels. In the evaluation section we have observed that both 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. We will extend the simple adaptive algorithm to handle more general cases, especially the joint optimization of coding degree and coding distance. Second, the overhead of XBC 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.

Acknowledgements

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 recom-

mendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Z. Zheng, P. Sinha, XBC: XOR-based buffer coding for reliable transmissions over wireless networks, in: Proc. of IEEE BROADNETS, Raleigh, NC, USA, September 2007.
[2] A. Adinegara, W. Lau, 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, April 2006.
[3] H. Balakrishnan, S. Seshan, E. Amir, R.H. Katz, Improving TCP/IP performance over wireless networks, in: Proc. 1st ACM Conf. on Mobile Computing and Networking, Berkeley, CA, USA, November 1995.
[4] T. Braun, T. Voigt, A. Dunkels, Energy-efficient TCP operation in wireless sensor networks, PIK Journal Special Issue on Sensor Networks 28 (2) (2005) 93–100.
[5] C.-Y. Wan, A.T. Campbell, L. Krishnamurthy, PSFQ: a reliable transport protocol for wireless sensor networks, in: Proceedings of the First ACM international workshop on Wireless sensor networks and applications, Atlanta, Georgia, USA, September 2002.
[6] S. Kim, R. Fonseca, 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, October 2004.
[7] p.a. IEEE 802.11 WG, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Standard Specification, IEEE, 1999.
[8] H. Balakrishnan, R. Katz, Explicit loss notification and wireless web performance, in: IEEE Globecom Internet Mini-Conference, Sydney, Australia, November 1998.
[9] G. Holland, N.H. Vaidya, Analysis of TCP performance over mobile ad hoc networks, in: Proc. of IEEE/ACM MOBICOM, August 1999, pp. 219–230.
[10] W.-P. Chen, Y.-C. Hsiao, J.C. Hou, Y. Ge, M.P. Fitz, Syndrome: a light-weight approach to improving TCP performance in mobile wireless networks, Wireless Communications and Mobile Computing Journal 2 (1) (2002) 37–57.
[11] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, R.H. Katz, A comparison of mechanisms for improving TCP performance over wireless links, IEEE/ACM Transactions on Networking 5 (6) (1997) 756–769.
[12] V. Anantharaman, S. Park, K. Sundaresan, 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 4 (2) (2004) 203–222.
[13] L.-J. Chen, T. Sun, M.Y. Sanadidi, M. Gerla, Improving wireless link throughput via interleaved FEC, in: Proc. of Ninth International Symposium on Computers and Communications (ISCC), 2004, pp. 539–544.
[14] C. Barakat, E. Altman, Bandwidth tradeoff between TCP and link-level FEC, Computer Networks: The International Journal of Computer and Telecommunications Networking 39 (5) (2002) 133–150.
[15] A.P. Jardosh, K.N. Ramachandran, K.C. Almeroth, 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, August 2005.
[16] V. Naik, A. Arora, P. Sinha, 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) 6 (7) (2007) 777–789.
[17] A. Sobeih, H. Baraka, A. Fahmy, ReMHoc: a reliable multicast protocol for wireless mobile multihop ad hoc networks, in: Consumer Communications and Networking Conference (CCNC), January 2004, pp. 146–151.
[18] R. Ahlswede, N. Cai, S.-Y.R. Li, R.W. Yeung, Network information flow, IEEE Transactions on Information Theory 46 (4) (2000) 1204–1216.
[19] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, J. Crowcroft, XORs in the air: practical wireless network coding, in: SIGCOMM, Pisa, Italy, 2006.
[20] L.E. Li, R. Ramjee, M. Buddhikot, S. Miller, Practical and efficient broadcast in mobile ad hoc networks, in: Proc. of IEEE INFOCOM, Anchorage, Alaska, USA, 2007.
[21] S.-Y.R. Li, R.W. Yeung, N. Cai, Linear network coding, IEEE Transactions on Information Theory 49 (2) (2003) 371–381.
[22] J. Ebert, A. Willig, "A Gilbert-Elliot bit error model and the efficient use in packet level simulation, in: TKN Technical Reports Series of Technical University Berlin, March 1999.
[23] R.K. Ganti, P. Jayachandran, H. Luo, T. Abdelzaher, Datalink streaming in wireless sensor networks, in: Proc. of ACM SenSys, Boulder, Colorado, USA, 2006.
[24] D.P. Bertsekas, J.N. Tsitsiklis, Introduction to Probability. Athena Scientific, June 2002.
[25] J. McDougall, S. Miller, Sensitivity of wireless network simulations to a two-state markov model channel approximation, in: Proc. of GLOBECOM, 2003.
[26] "The Network Simulator – ns-2. Availabel at: <http://www.isi.edu/nsnam/ns/>.
[27] A. Kamra, V. Misra, J. Feldman, D. Rubenstein, Growth codes: maximizing sensor network data persistence, in: Proc. of SIGCOMM, Pisa, Italy, 2006.
[28] S.A. Khayam, H. Radha, Markov-based modeling of wireless local area networks, in: Proc. of MSWiM, San Diego, California, USA, September 2003.
[29] A. Konrad, B.Y. Zhao, A.D. Joseph, Determining model accuracy of network traces, Journal of Computer and System Sciences 72 (7) (2006) 1156–1171.

Please cite this article in press as: Z. Zheng, P. Sinha, Buffer Coding for Reliable Transmissions over Wireless Networks, Comput. Commun. (2008), doi:10.1016/j.comcom.2008.09.033