CodeCast: A Network Coding Based Ad Hoc Multicast Protocol

Joon-Sang Park[†], Desmond S. Lun[‡], Yunjung Yi[§], Mario Gerla[†], and Muriel Médard[¶]
[†] Computer Science Department, University of California, Los Angeles
[‡] Coordinated Science Laboratory, University of Illinois at Urbana-Champaign
[§] Honeywell Laboratory
[¶] Laboratory for Information and Decision Systems, Massachusetts Institute of Technology

***Abstract*— In this article, we present CodeCast, a network coding based ad hoc multicast protocol. CodeCast well-suited especially for multimedia applications with low loss, low latency constraints such as audio/video streaming. The key ingredient of CodeCast is random network coding which transparently implements both localized loss recovery and path diversity with very low overhead. Simulation results show that in a typical setting, CodeCast yields near 100% delivery ratio as compared to 94% delivery ratio by traditional multicast. More importantly, the overhead is reduced by as much as 50%.**

I. Introduction

Major applications in mobile ad hoc networks (MANETs) include teleconferencing, disaster relief coordination, and battlefield operations which are group-oriented and mission-critical, requiring both accurate data delivery and timeliness. Undoubtedly, low loss, low latency multicast is a basic building block supporting such applications, especially in the face of frequent route outages and random packet drops due to mobility, fading, external interference, etc.

In this article we consider a multicast source that delivers multimedia data at rather constant rate, with a delay constraint. To illustrate the concept, a good example would be a security system transmitting images collected from various video cameras distributed in a large industrial plant. The source multicasts these images in a carousel fashion to dozens of security agents that are patrolling the area either on foot or in vehicles. The agents are immersed in a much larger MANET represented by employees, contract workers, visitors etc. It takes T seconds to cycle over all the video cameras, thus the useful delay bound to deliver an image to a security guard is T. Beyond T, data is stale and should be dropped.

Multicast packets may be corrupted and lost because of many reasons. First, fading, environment interference, and mobility can produce random like losses. In the case of military or civil defense surveillance, adversary/terrorist jamming may cause additional data loss. Another cause of loss is packet collision. In particular, collisions among hidden terminals are quite frequent in multicast where usual protection against hidden terminals à la Request-To-Send/Clear-To-Send (RTS/CTS) in 802.11 Media Access Control (MAC) is unavailable. If source rate exceeds network capacity, congestion builds up. Packet collisions and buffer overflow intensify, eventually causing network collapse unless proper end to end flow and congestion control strategy is in place. In this study, however, we simply assume that the source rate has been set to a value that does not cause congestion in normal operating mode.

Random losses cause quality degradation in the received images. In fact, beyond a certain threshold, the quality may be so poor to undermine the surveillance application. Thus, it behooves us to control loss by using packet loss recovery. However, 100% recovery is not the answer since full recovery may violate the delay constraints (thus rendering the recovered data useless). Moreover, a very aggressive recovery scheme may introduce excessive overhead, causing congestion to set in and possibly forcing the system to total collapse unless input rate is also controlled.

The goal of our multicast design is to use loss recovery judiciously, i.e., controlling loss while at the same time keeping latency in check. We call this problem the *controlled loss, bounded delay* multicast problem. Besides video surveillance, several other applications fit this model, such as tactical situation awareness dissemination, periodic sensor measurement distribution, entertainment audio/video steaming, etc.

Reliable multicast is closely related to the problem at hand. In fact, reliable multicast has been an active research area in wired IP networks over the years and various techniques have been proposed (e.g., [3], [12]). Most techniques and protocols developed for wired IP networks, however, hardly work as intended

in MANETs. As exemplified in [13], if one of the *wired* protocols runs in MANETs without any modification, it will incur excessive control overhead for maintaining underlying routing structure and also unreasonably long latency due to frequent route outages and heavily contended broadcast medium. Clear, for MANET protocols, the design choices should be made considering unique characteristics of MANETs and to leverage unique opportunities that MANETs provide.

Packet losses in MANETs tend to be random and locally contained, i.e., uncorrelated across nodes and packets. That is, it is likely that each node in a neighborhood undergoes dissimilar packet reception characteristic. Therefore, upon packet losses, a localized (or neighbor) loss recovery strategy, i.e., neighbors' helping each other, can be very efficient and effective at the same time. To fight correlated losses (e.g., losses experience by multiple receivers due to common upstream broken link) path diversity, i.e., the use of multiple, disjoint paths, paves an effective way. Path diversity is abundant typically in MANETs when nodes are densely packed. The main question thus is how to utilize path diversity efficiently. Our protocol named **CodeCast** achieves the controlled loss, bounded latency multicast with the help of localized loss recovery and path diversity. The key ingredient of CodeCast is random network coding [4] which transparently implements both localized loss recovery and path diversity with very low overhead.

By network coding, we refer to the notion of performing coding operations on the contents of packets throughout a network. This notion is generally attributed to Ahlswede et al. [1], who showed the utility of the network coding for multicast. The work of Ahlswede et al. was followed by other work by Koetter and Médard [7] that showed that codes with a simple, linear structure were sufficient to achieve the capacity of multicast connections in lossless, wireline networks. This result was augmented by Ho et al. [4], who showed that, in fact, a random construction of the linear codes was sufficient.

The utility of such random linear codes for reliable communication over lossy packet networks—such as MANETs—was soon realized [10]. In [9], a prescription for the efficient operation of MANETs is given, which proposes using the random linear coding scheme of [10] coupled with optimization methods for selecting the times and locations for injecting coded packets into the network. This problem of selecting the times and locations for injecting packets is called *subgraph selection*. The prescription given in [9] allows potentially for the optimal way of setting up a single connection to be found, but finding an optimal solution may be complex, especially under the complex constraints imposed by MANETs.

The protocol we propose in this article is essentially a heuristic implementation of the ideas in [9]. We propose heuristic methods for subgraph selection that are suitable for MANETs under 802.11 MAC, and we introduce some heuristic modifications to the random linear coding scheme of [10]. Heuristic methods for subgraph selection in wireless networks are also given in [14], but they differ from ours because they are chosen for energy efficiency of broadcasting applications in networks under the ideal collision-free MAC. Our work forms part of an emerging body of work on protocol development and assessment for wireless networks with network coding. Other works in this category are [5], [6], both of which focus on protocols solely for unicast.

The rest of this article is organized as follows. Section II illustrates the operation of CodeCast and an evaluation of CodeCast through simulation is presented in Section III. Finally, section IV concludes the article.

II. CodeCast Protocol

We assume for simplicity that an application generates a stream of equal size frames $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, ... where subscripts denote unique and consecutive sequence numbers. If an application generates a stream of varying size data frames, the frames are fragmentized and/or padded to render them into equal size frames. We also assume that each stream (or an application generating the stream) can be uniquely distinguished by a source address and port number pair (as in usual multicast protocols) or a globally unique identification number assigned to each stream. The stream of frames is logically reorganized into a stream of blocks, each of which is a set of frames with adjacent sequence numbers. We use a tuple ($blockid$, $blocksize$) to denote a block to which frames with sequence numbers equal to or larger than $blockid$ and smaller than ($blockid$ + $blocksize$) belong. ($blockid, blocksize > 0$.) An application frame $\mathbf{p}_k$ is said to be *in* $(blockid,\ blocksize)$ iff $k \geq blockid$, $k < (blockid + blocksize)$. A coded packet $\mathbf{c}_{(blockid,blocksize)}$ is a linear combination of frames in ($blockid$, $blocksize$). That is, $\mathbf{c}_{(blockid,blocksize)} = \sum_{k=1}^{blocksize} e_k \mathbf{p}_{(k-1+blockid)}$ where e_k is a certain element in a certain finite field $\mathbb{F}$. Every arithmetic operation is over $\mathbb{F}$. Application frame $\mathbf{p}$'s and coded packet $\mathbf{c}$'s are also regarded as vectors over $\mathbb{F}$. In the header of a coded packet, the *encoding vector* $\mathbf{e} = [e_1 \ldots e_{blocksize-1}]$ is stored along with $blockid$ and $blocksize$ for the purpose of later *decoding* at the receivers. Transporting the encoding vectors along with coded packets was first suggested in [2]. When generating a coded packet $\mathbf{c}$, each e_k is drawn randomly from $\mathbb{F}$, hence the name of random linear coding. Similarly

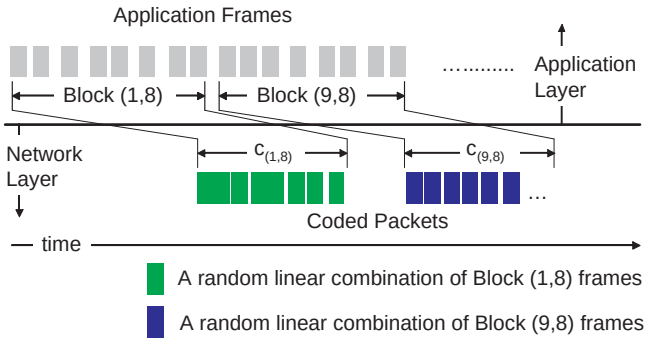


Fig. 1. Application frames, blocks, and coded packets ($blocksize$ = 8)

to the application frames, a coded packet $\mathbf{c}$ is said to be *in* $(blockid,\ blocksize)$ iff the coded packet $\mathbf{c}$ is tagged with $(blockid,\ blocksize)$. Fig. 1 illustrates the relationship between application frames, blocks, and coded packets. We assume for simplicity that blocks are non-overlapping, i.e, at the source applications frames are organized and coded packets are generated such that not an application frame nor a coded packets appears in two different blocks. Throughout this article, we abuse lowercase boldface letters to denote vectors, frames, or packets, uppercase letters to denote matrices, italics to denote variables or fields in the packet header.

We refer to the entity (in network layer) that performs encoding and decoding for the protocol as CodeCast Agent. Since a block of application frames is required to generate a coded packet, the agent has to collect and buffer the application frames. We simply assume that data storage on each node is large enough to store all the data for a limited amount of time. Once a whole frame block ($blockid$, $blocksize$) is amassed, the agent generates $blocksize$ coded packet $\mathbf{c}_{(blockid,blocksize)}$'s, and broadcast them to the neighborhood. (See Fig. 1) $blocksize$ need not be a fixed value. If the agent receives known application frames, $blocksize$ can be determined on-the-fly according to the delay constraints of the frames. The objective is to make the size of each block big enough to gain efficiency while minimizing the possibility of delivering packets with delay constraint violations. (In general, the bigger the block size the greater the efficiency gain is, and also the delay.) Otherwise, the agent uses a predefined number to limit maximum wait time in the buffer. For example, if an application generates frames at a rate of 10 frames/sec and they all expire 1 second after creation, the agent sets the block size to 8 frames—giving 0.3 seconds of leeway for delivery.

On reception of a coded packet $\mathbf{c}_{(blockid,blocksize)}$, every node stores the packet in its local memory for later decoding and forwarding. To decode and recover $blocksize$ application frames belonging to ($blockid$, $blocksize$), a node must collect more than $blocksize$ coded packets tagged with $(blockid, blocksize)$ and encoding vectors that are linearly independent of each other. Once collected, CodeCast Agent decodes and recovers the $blocksize$ original application frames and deliver them to the application. Let $\mathbf{c}_k$ be a coded packet labeled $(blockid, blocksize)$ in a node's local memory, $\mathbf{e}_k$ be the encoding vector prefixed to $\mathbf{c}_k$, and $\mathbf{p}_{blockid+k-1}$ be an application frame to be decoded and recovered where k = 1, ..., $blocksize$. Further, let $\mathbf{E}^\mathsf{T}$ = $[\mathbf{e}_1^\mathsf{T}$ ... $\mathbf{e}_{blocksize}^\mathsf{T}]$, $\mathbf{C}^\mathsf{T}$ = $[\mathbf{c}_1^\mathsf{T}$ ... $\mathbf{c}_{blocksize}^\mathsf{T}]$, and $\mathbf{P}^\mathsf{T}$ = $[\mathbf{p}_{blockid}^\mathsf{T} \cdots \mathbf{p}_{blockid+blocksize-1}^\mathsf{T}]$ where superscripts T denotes the transpose operation, then conceptually $\mathbf{P}$ = $\mathbf{E}^{-1}\mathbf{C}$, which obtains the original application frames. Note that all $\mathbf{e}_k$'s must be linearly independent to be able to invert $\mathbf{E}$.

If a node receives a coded packet with a new tuple ($blockid$, $blocksize$), it sets up a timer for the tuple ($blockid$, $blocksize$) expiring in $blocktimout$ seconds. When the timer expires, it broadcasts to the neighborhood r coded packet $\acute{\mathbf{c}}_{(blockid,blocksize)}$'s after local re-encoding. r is set to 1 in the simplest setting. The local re-encoding is through the same process that the data source has undergone to generate a coded packet, i.e., a random linear combination of packets tagged with the same ($blockid$, $blocksize$) available in local memory. Note that though the packets in memory are coded ones thus the re-encoded packet $\acute{\mathbf{c}}_{(blockid,blocksize)}$ = $\sum_{k=1}^{blocksize} \acute{e}_k \mathbf{c}_k$ is tagged with the encoding vector $\acute{\mathbf{e}}$ = $\sum_{k=1}^{blocksize} \acute{e}_k \mathbf{e}_k$ where each $\acute{e}_k$ is drawn uniformly from $\mathbb{F}$. $\mathbf{c}_k$ and $\mathbf{e}_k$ are a coded packet labeled $(blockid, blocksize)$ in local memory and the encoding vector prefixed to $\mathbf{c}_k$ respectively. The timer for ($blockid$, $blocksize$) is reset on expiration unless a decodable set of coded packets in ($blockid$, $blocksize$) is collected, i.e, all the applications frames in ($blockid$, $blocksize$)

are decoded and recovered. The timer for ($blockid$, $blocksize$) can be cleared if the deadline has passed for all the frames in ($blockid$, $blocksize$).

On the expiration of the timer for ($blockid$, $blocksize$), even though there are less number than $blocksize$ of code packet $\mathbf{c}_{(blockid,blocksize)}$'s in local memory, a node has to generate and transmit a coded packet using packets available in the memory. The number of packets that are actually combined to yield a coded packet is recorded as $rank$ in the header of the coded packet. A coded packet **c** with $rank$ smaller than $blocksize$ is augmented with a nullspace vector **n** which is a vector in the nullspace of all encoding vectors of packets that are combined to yield the coded packet **c**. Since a coded packet $\mathbf{c}_{(blockid,blocksize)}$ with $rank$ smaller than $blocksize$ indicates that the sender of the $\mathbf{c}_{(blockid,blocksize)}$ is in need of more coded packets labeled ($blockid$, $blocksize$) to be able to decode and recover application frames in ($blockid$, $blocksize$), any node receiving such a packet transmits another coded packet to help the sender of $\mathbf{c}_{(blockid,blocksize)}$ collecting more coded packets if there is in local memory a packet with the encoding vector not orthogonal to the nullspace vector of $\mathbf{c}_{(blockid,blocksize)}$.

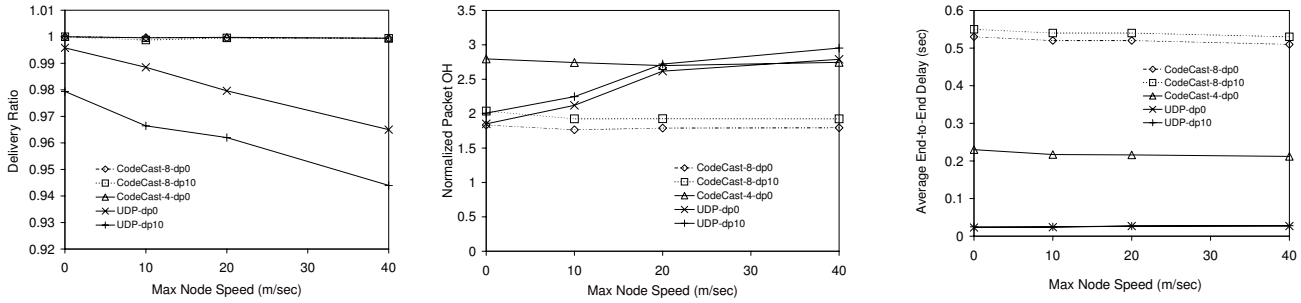
Besides $blocksize$, $blocktimeout$ is another key parameter having an impact on performance—especially end-to-end delay. For simplicity we use a fixed value for $blocktimeout$. A reasonable value is a multiple of the average broadcast jitter. Since every transmission in CodeCast is MAC/link layer broadcasting, a small random amount of wait time before each transmission called broadcast jitter is applied to reduce collisions. Without broadcast jitter, MAC/link layer broadcasting suffers severely from the hidden terminal problem.

Clearly, it is sub-optimal, producing unnecessary data transmissions, if the entire network participates in the forwarding of multicast data to deliver it from the source to a set of designated receivers. As indicated earlier, the problem of finding the optimal set of forwarding nodes and the frequency of injecting packets is called *subgraph selection*. We use a simple heuristic for the problem in CodeCast. Every coded packet carries in the header three more fields, $vldd$, $dist$, and $nust$. The one-bit field $vldd$ is set if either the sender is a multicast receiver or has received a previous-block coded packet with $vldd$ bit set from one of the sender's downstream nodes. A node considers a neighboring node to be a downstream node if the neighboring node transmits a packet with a larger $dist$ value than the $dist$ value the node maintains. Each node maintains as a local variable $dist$ indicating the hop-distance from the multicast data source and copies its value to every code packet the node transmits. Every time a node transmits a coded packet **c**, $dist$ is recalculated as one plus the biggest $dist$ value found in the headers of the packets which are combined to yield the coded packet **c**. Conversely, a node considers a neighboring node to be a upstream node if the neighboring node transmits a coded packet in a new block ($blockid$, $blocksize$) or a smaller $dist$ value than the $dist$ value the node maintains. Each node also maintains $nust$ indicating the number of upstream nodes as a local variable and records its value in the header of every packet the node transmits. As mentioned earlier, a node broadcasts to the neighborhood r coded packet $\mathbf{c}_{(blockid,blocksize)}$'s when the timer for ($blockid$, $blocksize$) expires. r can be a constant value (e.g., r = 1) and the same for every node. Or nodes can have time-varying r: r is set to $\lceil \frac{blocksize}{mn} \rceil$ where mn is the minimum value of $nust$s received from any downstream node. This way, each node determines individually the frequency of injecting packets. If a node does not receive any packet with $vldd$ bit set transmitted by any downstream node while processing $prunecount$ consecutive blocks, it stops forwarding for $sleeptimout$ seconds. This way, unnecessary nodes are pruned off the forwarding subgraph.

In CodeCast, there is no clear notion of paths or routes that packets follow through as in conventional multicast. Rather *innovative* packets are propagated through the forwarding subgraph which provides rich path diversity. By an innovative packet, we mean a packet that contributes to decoding and recovering of application frames on a node, i.e., a packet carrying the encoding vector linearly independent of those in the node's local storage. In fact, the forwarding subgraph in CodeCast is very robust to node mobility and random errors. Displacement or failure of a few nodes in the forwarding subgraph (or structure) does not affect the delivery of packets much in CodeCast whereas in conventional multicast such events might result in a series of packet drops.

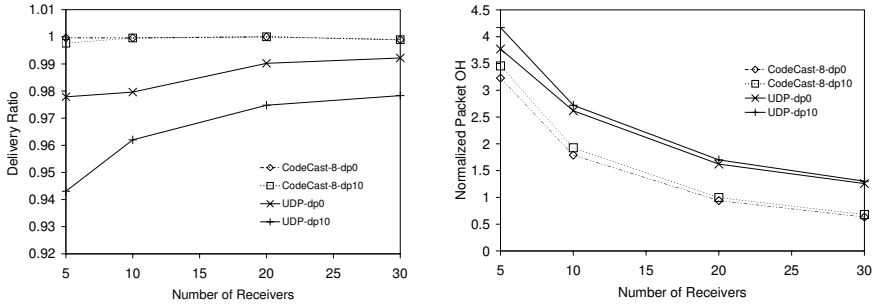
III. Performance Evaluation

To evaluate the performance of CodeCast, we implemented CodeCast in QualNet [11], a packet-level network simulator, and conducted a set of simulations using the following settings: 802.11 MAC; two-ray ground path-loss propagation model; 376 m of transmission range and 2 $Mbits/sec$ of channel bandwidth; 100 nodes randomly placed on 1500 $\times$ 1500 m^2 field; 120 seconds of simulation time; single multicast group with single source and 10 receivers unless otherwise specified; constant bit-rate application traffic tranmitting 512 byte packets at 5 $Kbytes/sec$ rate; Random Waypoint Mobility model with 0 pause time, 0 minimum speed, and varying maximum speed unless otherwise specified. Results are averaged over 10 runs with various random



(a) Packet delivery ratio (b) Normalized packet overhead (c) End-to-end delay

Fig. 2. Comparison of CodeCast and ODMRP: Varying node speed



(a) Packet delivery ratio (b) Normalized packet overhead

Fig. 3. Comparison of CodeCast and ODMRP: Varying group size

seeds. For CodeCast, we set $blocktimout$ to 40 ms, $prunecount$ to 5, $sleeptimout$ to 15 seconds, and the deadline of each packet to 1 second after creation. Note that we set r to 1 on every node always. Using $\mathbb{F}_{2^8}$ field, each data packet carries an additional 16 byte header to hold the encoding vector, $blockid$, $blocksize$, $rank$, etc. In case of packets with $rank < blocksize$, additional 8 bytes are needed to hold the nullspace vector.

We contrast CodeCast to the plain User Datagram Protocol (UDP) running on top of a conventional multicast protocol in mobile settings. In this comparison, UDP assumes On Demand Multicast Routing Protocol (ODMRP) [8] as the underlying multicast protocol. We restrict our attention to ODMRP since as shown in [8] it is one of the best performing protocols especially in mobile and lossy channel settings in which we are specially interested. The challenge of MANET is, in fact, maintaining network operations in face of nodes' mobility and lossy wireless channel. To simulate the lossy channel, nodes are forced to drop successfully received packets randomly with some probability. For CodeCast, two different block sizes are used to evaluate the impact of the block size on the performance. In figures, CodeCast-α-dpβ denotes CodeCast using α-packet blocks and operating in the artificial lossy channel with packet drop probability β%. CodeCast-8-dp0 indicates the case where 8-packet block and packet drop probability 0 are used, CodeCast-8-dp10 does the case with 8-packet and packet drop probability 10%, and CodeCast-4-dp0 is for the 4-packet block in combination with packet drop probability 0 case. Similarly, UDP-dpβ denotes UDP for packet drop probability β% case.

In Fig. 2(a), CodeCast demonstrates near 100% data delivery regardless of node speed, block size, packet drop probability. On the other hand, the packet delivery ratio of the conventional multicast represented by ODMRP degrades to 94% as mobility and packet drop probability increase. The packet delivery ratio is defined as the ratio of data packets received by all receivers over total data packets sent. More importantly, as shown in 2(b), CodeCast incurs less overhead than the conventional multicast (when the block size is 8 packets). When the maximum node speed is 40 m/sec the reduction in overhead is as much as 40%. To measure protocol overhead, we use a commonly used metric, the normalized packet overhead defined as the total number of packets transmitted to the wireless channel by any node in the network divided by the total number of data packets delivered to any receiver. The overhead of CodeCast with 4-packet blocks is also comparable to the conventional multicast when mobility is high.

Random errors (forced errors in our simulation) and route breakage due to node mobility are two main causes of packet losses in the conventional multicast. In conven-

tional multicast, a tree (or mesh) structure is constructed and maintained for the packet delivery but the tree (or mesh) structure is prone to be broken and difficult to maintain when nodes are moving fast. Thus packet delivery ratio deceases as node speed increases. To cope with the problem, ODMRP superimposes multiple meshes using more nodes as forwarding nodes such that it can survive single mesh failure. ODMRP tends to use more and more nodes as forwarding nodes as mobility increases, which is equivalent to trading overhead off for high packet delivery ratio and explains ODMRP's overhead increase (sub)linear in the node speed. Despite the effort, unfortunately, ODMRP permits packet losses in highly dynamic MANETs. CodeCast, whereas, always shows near perfect packet delivery ratio since it builds a forwarding structure (or subgraph) that is very robust to node mobility.

Fig. 2(c) shows the drawback of CodeCast, end-to-end delay. The end-to-end delay is the difference between packet generation time at the source and packet delivery to the application at the receiver. In CodeCast, a certain level of increase in end-to-end delay is inevitable since at the source it takes time to collect a block of packets such that coding over the block is possible. In our simulations, the application generates packets at a rate of 10 packets/sec so if the block size is 8 packets each packet spends on average 0.35 seconds waiting in the buffer; it spends 0.15 seconds if the block size is 4 packets. This in part explains CodeCast-4's low average end-to-end delay compared to those of CodeCast-8s.

Fig.3(a) and Fig.3(b) illustrate CodeCast's performance with varying number of receivers in the multicast group when the maximum node velocity is fixed to 20 m/sec. We observe again that CodeCast achieves near 100% data delivery while retaining very low overhead regardless of environmental changes. Notably, CodeCast's overhead is only the half compared to the conventional multicast when the number of receivers is over 20.

IV. Conclusions

In the article, we presented CodeCast, a network coding based controlled loss, bounded delay multicast protocol. The main ingredient of CodeCast is random network coding which is used to implement both localized loss recovery and path diversity transparently. Through simulation, we demonstrated that CodeCast achieved near perfect packet delivery ratio while maintaining lower overhead than conventional multicast.

Considering the fact that MANETs are extremely sensitive to overload, an immediate future work of CodeCast is to incorporate a congestion control mechanism to avoid congestion collapse. In this article, we assumed that the source rate was fixed a priori, possibly based on careful engineering of resource allocations. In practice, the a priori resource allocation is difficult at best in a mobile environment. Moreover, an increasing number of applications allow the adjustment of the source rate to match network conditions (e.g., adaptive coded video sources, adaptive resolution data dissemination, etc). It is thus appropriate to develop an extension of CodeCast that includes source rate adaptation.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. Inform. Theory*, 46(4):1204–1216, July 2000.

[2] P. Chou, Y. Wu, and K. Jain. Practical network coding. In *Proc. 51st Allerton Conf. Communication, Control and Computing*, Oct.

[3] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *IEEE/ACM Trans. Networking*, 5(6):784–803, Dec. 1997.

[4] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. Submitted to *IEEE Trans. Inform. Theory*.

[5] S. Katti, D. Katabi, W. Hu, H. Rahul, and M. Médard. The importance of being opportunistic: Practical network coding for wireless environments. In *Proc. 43rd Annual Allerton Conference on Communication, Control, and Computing*, Sept. 2005.

[6] R. Khalili and K. Salamatian. A new relaying scheme for cheap wireless relay nodes. In *Proc. Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks* , 2005.

[7] R. Koetter and M. Médard. An algebraic approach to network coding. *IEEE/ACM Trans. Networking*, 11(5):782–795, Oct. 2003.

[8] S.-J. Lee, W. Su, and M. Gerla. On-demand multicast routing protocol in multihop wireless mobile networks. *ACM/Kluwer Mobile Networks and Applications, special issue on Multipoint Communications in Wireless Mobile Networks*, 2002.

[9] D. S. Lun, M. Médard, and R. Koetter. Efficient operation of wireless packet networks using network coding. In *Proc. International Workshop on Convergent Technologies (IWCT) 2005*, June 2005. Invited paper.

[10] D. S. Lun, M. Médard, R. Koetter, and M. Effros. On coding for reliable communication over packet networks. Submitted to *IEEE Trans. Inform. Theory*.

[11] Scalable Network Technologies, http://www.scalable-networks.com.

[12] S. Paul, K. K. Sabnani, and S. B. J. C. Lin. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, 1997.

[13] K. Tang, K. Obraczka, S.-J. Lee, and M. Gerla. Reliable Adaptive Lightweight Multicast Protocol. In *Proc. IEEE Internationl Conference on Communications*, 2003.

[14] J. Widmer, C. Fragouli, and J.-Y. Le Boudec. Low-complexity energy-efficient broadcasting in wireless ad-hoc networks using network coding. In *Proc. WINMEE, RAWNET and NETCOD 2005 Workshops*, Apr. 2005.