Pipeline Network Coding for Multicast Streams

Chien-Chia Chen, Soon Y. Oh, Phillip Tao, Mario Gerla, M. Y. Sanadidi
UCLA Computer Science Department, Los Angeles, CA 90095, USA
{chienchiachen, soonoh, ptao, gerla, medy}@cs.ucla.edu

ABSTRACT

This paper studies the performance of pipeline network coding for multicast stream distribution in high loss rate MANET scenarios. Most of the previous network coding implementations have been based on batch network coding, where all blocks in the same batch are mixed together. Batch coding requires that the entire batch is received before decoding at destination. Thus, it introduces high decoding delays that impact the stream reception quality. Instead of waiting for the entire batch (i.e., generation), pipeline network coding encodes/decodes packets progressively. Consequently, pipeline network coding yields several benefits: (1) reduced decoding delay, (2) further improved throughput, (3) transparency to higher layers (UDP, TCP, or other applications), (4) no special hardware support and (5) easier implementation. We show performance gain of pipeline coding compared to batch coding via extensive simulation experiments.

Keywords: Network Coding, Multicast

1 INTRODUCTION

Due to the nature of wireless communications, MANETs are vulnerable to channel errors, interference and jamming. There are two well known approaches for effective error recovery; Forward Error Correction [3] and ARQ. Since the targeted application here is real time multicast streaming, the ARQ scheme is not appropriate. In order to efficiently control losses, recent research in this area has exploited FEC-based coding schemes, such as erasure coding [4] and network coding [5-7].

The word "erasures" at the network layer means "missing packets in a stream" [4] and erasure coding is classified as FEC coding for binary erasure channels since a source achieves error correction by injecting redundant data. A source generates n packets from k original ones. The original data can be reconstructed from a subset of n packets. The most common approach of erasure coding proposed in [4][1] and in this paper we define a new term "batch erasure coding" that generates encoded packets only from the same batch. As for network coding, we do a similar thing. A stream of packets is split into k packets called "generation" [7] and the source produces n coded packets for each generation using random linear coding. We define $r=n/k$ as the "coding redundancy" in this paper. At destinations, any subset of k linearly independent coded packets is sufficient to reconstruct the original generation. In other words, n-k losses are allowed in a group of n coded packets. The difference between Erasure coding and network coding is that in the former the coding is done only at the source while in the latter the coding is done also at intermediate nodes. In fact, Erasure coding can be viewed as a subset of network coding.

The above erasure coding scheme is called "pre-defined rate coding" since coding redundancy is decided before transmission. Another form of erasure coding is called "rateless coding," which can generate infinite encoded packets. Namely, there is feedback from destination to source and the source will adaptively adjust the redundancy depending on reception quality at destination etc. However, end to end feedback can be slow and is not practical in multicast/broadcast because of feedback control message "explosion" Fountain Codes [17] and Raptor Codes [18] are this type of coding schemes. In this paper, we only consider pre-defined rate coding. In this respect, it should be noted that network coding allows dynamic adjustment of redundancy WITHOUT expensive end to end feedback since it can just rely on downstream neighbor feedback to adjust retransmissions.

In disruptive networking environments, end-to-end erasure coding is not sufficient to achieve reliable packet transmission [8]. Thus, researchers have applied network coding to such challenged environments, where the broadcast nature of wireless medium renders network coding particularly effective [1]. Unlike source coding, as mentioned earlier all nodes in the network participate in the encoding process. In this paper, we use "batch network coding" to refer to a network coding scheme where source and relay nodes encode data packets in the same generation using random linear coding. Batch network coding is one of the most common approaches of network coding [1, 6-7, 9-11][1].

However, there are two critical drawbacks in both batch erasure coding and batch network coding. First of all, they introduce an encoding and decoding delay that proportionally increases with generation size. Thus the quality of real-time streaming degrades due to delay and jitter. Furthermore, because of the generation based coding, a generation is decodable only once enough linearly independent (i.e., innovative) packets arrive; otherwise, the entire generation is discarded. In other words, throughput may significantly

[1] There are a lot of other approaches for both erasure coding and network coding. In this paper, we narrow down the definition to the specific coding schemes as described in our paper. Other approaches not used in this paper are given as references only.

degrade due to frequent generation discarding in a lossy network.

To solve these batch coding problems, we propose a coding scheme, "pipeline coding," where both encoding and decoding can proceed progressively. Instead of waiting for all packets in a generation to arrive, sources start encoding and sending coded packets whenever a new data packet arrives. At the destination, data packets can also be reconstructed "progressively," i.e., incrementally. Consequently, pipeline coding achieves (1) a lower coding delay, (2) a higher throughput, (3) transparency to higher layers (UDP, TCP, or other applications), and (4) no special hardware requirement.

In this paper we evaluate the impact of the pipeline concept on network coding. A future paper will evaluate pipelining for erasure coding. The rest of the paper is organized as follows. Section 2 depicts the detail of the proposed coding scheme. Simulation results and testbed experiment results are presented in Section 3. Section 4 concludes the paper.

2 PIPELINE CODING

This section first gives the mathematical definition of batch coding; next the mathematical definition of pipeline coding is presented; and the proposed encoding and decoding procedures are described lastly. Throughout this paper, the term "erasure coding" refers to source-side only network coding, and "network coding" refers to the case where relays participate in encoding along with the source. Table 1 below provides a summary of the terminology we adopt in this paper.

2.1Batch Coding

As defined previously, in this paper, batch coding refers to both batch erasure coding and batch network coding. Suppose at the source, an application generates a series of equal-sized packets $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$,…,. Let k be the number of packets in a single generation. A coded packet $\mathbf{c}$ in i^{th} generation is then defined as:

$$\mathbf{c} = \sum_{j=1}^{k} e_j \mathbf{p}_{i\times k+j} \tag{1}$$

where e_k is a particular element in a particular finite Galois field $\mathbb{F}$, and $i\times k$ is the total number of packets transmitted so far in the file, before the i^{th} generation. Throughout this article, we use lowercase boldface letters to denote vectors, frames, or packets; uppercase letters to denote matrices; and italics to denote variables or fields in the packet header. Every arithmetic operation is over $\mathbb{F}$ so that data packets $\mathbf{p_i}$ and coded packets $\mathbf{c}$ are also regarded as vectors over $\mathbb{F}$. Let r denote the coding redundancy, where $r\geq 1$. For each generation, the source will produce $k\times r$ coded packets. At destination side, after receiving k linearly independent packets, the destination can then reconstruct the original content by solving the following linear system of equations as given in eq. (2) using Gaussian Elimination.

Table 1. Definitions of Terms Used in This Paper

Term	Definition
Erasure Coding	Source-side only coding.
Network Coding	Source-side and relay coding.
Batch Coding	Every coded packet will encode all data packets within the same generation. Coding and decoding begins only when the generation rank is "full"
Pipeline Coding	Coded packets will be generated upon every new data packet arriving. Destinations decode the data packets progressively if possible.
Generation	A set of packets that are encoded or decoded together as a unit.
Coding Vector (Encoding Vector)	A vector of coefficients that reflect the linear combination of data packets.
Rank (Degree of Freedom)	Number of linearly independent combinations of data packets.
Innovative Packet	A packet that increases the rank.
Coding Redundancy	Number of coded packets sent per generation divided by generation size.
Delay	The time difference between packet reception by destination application and transmission from source.

$$\begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_k \end{bmatrix} = \begin{bmatrix} e_1^{(1)} & \cdots & e_k^{(1)} \\ \vdots & \ddots & \vdots \\ e_1^{(k)} & \cdots & e_k^{(k)} \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_k \end{bmatrix} \tag{2}$$

In batch erasure coding, only a source encodes packets and other nodes relay encoded packets while in batch network coding, relays also participate in coding. Upon receiving a coded packet, relays first check whether it is innovative. Non-innovative packets are discarded while innovative packets are stored in the generation buffer. For each newly arrived innovative packet, relays generate and send out a new coded packet, which is a new linear combination of all received innovative coded packets in the same generation. This procedure is called "re-encoding." Note that relays do not attempt to recover the original data packets. The re-encoded packets, by induction, are still linear combination of original data packets. This simple innovative checking and re-encoding on relays will make a major difference in performance, as will be shown in Section 3.

Since each coded packet must be a linear combination of ALL data packets in the same generation, at the source a delay is incurred until all data packets in a generation arrive. Further,

at the destinations, either all data packets in a generation will be decoded successfully, or none will be, under batch coding.

In order to formulate the coding delay, we first define the "delay" as the time difference from the moment when a packet is received by the application at the destination, to when it is delivered to the application at the source. Let t_i denote the time that the source generates $\mathbf{data_i}$ and t_{ci} denote the time that i^{th} coded packet is sent out. Assume the four delay components - processing, transmission, propagation, and queuing delay - are constant. Let d_n be the constant delay in the network including all delay components, and d_c be the constant decoding delay. Assume each generation has k data packets. Therefore, in lossless links, the delay to deliver $\mathbf{data_i}$ will be:

$$t_{c\lceil i/k \rceil * k} - t_i + d_n + d_c \tag{3}$$

For example, the delay for delivering $data_1$ will be t_{c4}-t_1+d_n+d_c as shown in Fig. 1, where we assume the generation size is $k=4$ and the coding redundancy is $r=1.25$. Therefore, $k*r=5$ coded packets will be sent out for every generation. Note that the figure does not show the re-encoding process because ~~the latter does not significantly affect the coding analysis while at the same time creates a graph that is too complicated to draw in limited space. A detailed description of re-encoding can be found in [9]~~.

Assuming now that packet losses are possible over the communications path, data packets can still be decoded as long as any subset of 4 linearly independent coded packets out of the 5 transmitted by the source is received. The generation #2 in Fig. 1 shows an example of loss that was recovered and resulted in a successful decoding. However, if the redundancy level is not enough to compensate for losses, none of the data packets in this generation will be decoded as depicted in Fig. 2. Here the loss of two packets renders it impossible to decode this generation.

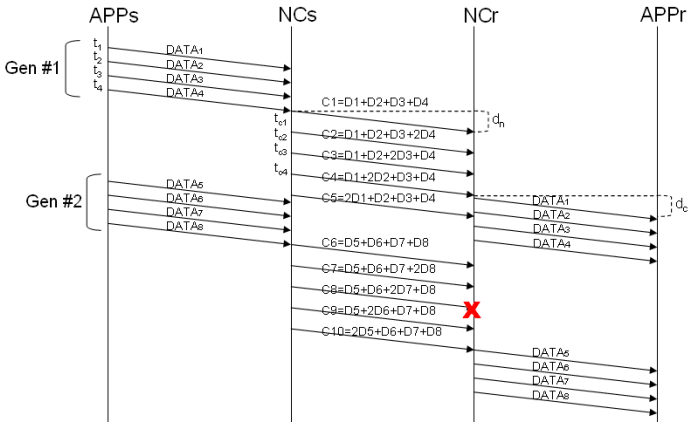

Fig. 1 Batch Coding Example

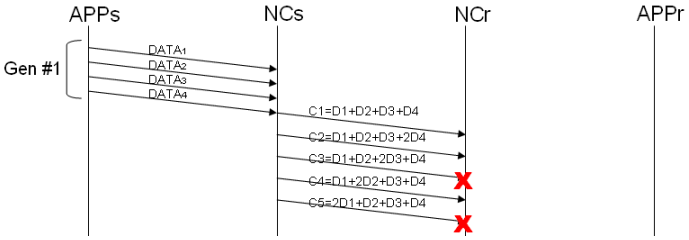

Fig. 2 Batch Coding: Undecodable Case

2.2 Pipeline Coding

2.2.1Pipeline Encoding and Decoding

Our pipeline coding scheme aims to reduce the coding delay as well as to further improve the throughput. Normally, packets are not sent from an application all at once. Therefore, in pipeline coding, we relax the limitation of waiting for all data packets of a generation to be received from the application. Adopting the same notation in section 2.1, the encoding function is then changed as following:

$$\mathbf{c} = \sum_{j=1}^{m} e_j \mathbf{p}_{i \times k + j} \tag{4}$$

Where the new variable m is the number of data packets currently present in the generation buffer. In other words, upon receiving a new data packet, the source will instantly trigger the encoding process based on currently received data packets. If all coded packets are delivered successfully, destinations can construct the following lower triangular matrix without any extra computation:

$$\begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_k \end{bmatrix} = \begin{bmatrix} e_1^{(1)} & 0 & \cdots & 0 \\ e_1^{(2)} & e_2^{(2)} & & \vdots \\ \vdots & & \ddots & 0 \\ e_1^{(k)} & e_2^{(k)} & \cdots & e_k^{(k)} \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_k \end{bmatrix} \tag{5}$$

The above linear equation can be solved progressively without waiting for generation completion. For example, upon receiving $\mathbf{c_1}$, destinations can decode $\mathbf{p_1}$, and so on. Coding redundancy is applied at the source in order to mitigate losses, in a slight different way from batch coding. Let r be the coding redundancy, where $r \geq 1$. Per each data packet, with a probability of $r - \lfloor r \rfloor$, r+1 coded packets are generated and sent; with a probability of $1 - (r - \lfloor r \rfloor)$, r coded packets are generated and sent. In other words, redundancy is uniformly distributed among every data packet. Also, if network coding is used, relays will all participate in coding, as explained in [9], in order to produce further redundancy of coded packets.

In the batch coding example (Fig. 1), the delay of data packets is shown in eq. (3), which is two constant delays plus the time difference between the data packet sent from the source application and the last innovative packet sent from the source. Following the same notation, t_i denotes the time when the source generates $\mathbf{data_i}$ and t_{ci} denotes the time when the i^{th} coded packet is sent out. If pipeline coding is adopted, in lossless links, the delay of $\mathbf{data_i}$ will become t_{ci}- t_i+ d_n+ d_c,

which is much lower than eq. (3) delay in most cases. Thus, the delay is greatly reduced.

Fig. 3 presents an example of pipeline coding, with generation size, $k = 4$ and source coding redundancy, $r = 1.25$. The re-encoding part is not shown for simplicity. As shown in the figure, data packets are encoded instantly upon arrival. Similarly, at the destination, coded packets can be decoded immediately once a new innovative packet arrives. For example, if $\mathbf{c_1}$ is delivered successfully, it is decoded immediately at the destination. This gives us a delay of t_{c1}- t_1+ d_n+ d_c.

In the case of packet losses as shown in the 2nd generation of Fig. 3, the destination will store un-decodable packets and wait for the next packet with which, hopefully, the previous loss can be recovered. As soon as a new innovative coded packet arrives and thus Gaussian Elimination can decode new unknown packets, newly decodable packets will then be delivered to upper layer. In Fig. 3, after receiving $\mathbf{c_9}$ and $\mathbf{c_{10}}$, $\mathbf{data_7}$ and $\mathbf{data_8}$ will be decoded. Let t_{c10} denote the time that c_{10} is sent. The delay in this case will be $t_{c10} - t_7 + d_n + d_c$ for $\mathbf{data_7}$ and $t_{c10} - t_8 + d_n + d_c$ for $\mathbf{data_8}$. These two data packets will have the same amount of delay as in batch coding scheme, but $\mathbf{data_5}$ and $\mathbf{data_6}$ still benefit from pipeline coding.

Pipeline coding can partially recovers a subset of the data packets of a generation and deliver them to the upper layer. This is a significant difference from batch coding, which either delivers an entire generation to the upper layer, or discards the whole generation. For example, assume that $\mathbf{c_{10}}$ of Fig. 3 is lost, data packet #7 and #8 will never have a chance to be decoded regardless of which coding scheme is used. However, without pipeline coding, none of the data packets in 2nd generation can be decoded, while with pipeline coding, we can still decode data packets #5 and #6.

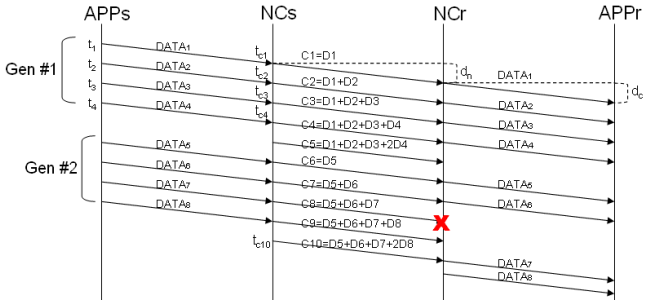

Fig. 3 Pipeline Coding Example

The ability of "partial" generation recovery is important in real time stream applications. It would not be useful of course in reliable data applications, where the entire generation must be received ("all or nothing").

2.2.2Encoding and Decoding Procedure

Fig. 4 shows the encoding procedure of our proposed pipeline coding. The coding module is implemented at the network layer and it does encoding, decoding, and broadcasting. When the source receives a data packet from the transport layer, it first stores the data in the generation buffer. Thereafter, based on the coding redundancy, a number of coded packets are generated and sent out. For each generated coded packet, the source first randomly generates the coding coefficients. Secondly, it checks the generation buffer. For a missing packet, which has not arrived yet, the corresponding coefficient is set to zero. Finally, the source encodes the data packets based on this encoding vector, and sends it to the lower layer. Note that the '**if**' statement in the while loop is to make sure redundant packets are generated uniformly instead of always in the end of the generation.

Encoding Procedure

```
Function Handle_Packets_from_Transport(data)
  Store data to generation buffer
  Coded_Packet_Gen(coding_redundancy)
Function Coded_Packet_Gen(num)
  i=0
  while i<num do
    if (num-i<1 and rand()%100 >num-i) then
      break
    Generate Coding Vector
    for j=0..generation_size-1do
     if generation_buffer[j] is NULL then
      coding_vector[j]=0
    Encode packet
    Send Coded Packet to MAC layer
    i=i+1
```

Fig. 4 Pipeline Coding—Encoding Procedure

Fig. 5 is the decoding procedure. A destination first examines whether the received coded packet is innovative or not. An innovative packet will be stored in the generation buffer. Afterward, the decoding module invokes Gaussian Elimination routines and attempts to decode data packets. After decoding, newly decoded data packets will be stored into another decoding buffer and then delivered to the upper layer. Note that in some rare cases, a few packets might be decoded out-of-order. In order to reduce the impact to TCP, we have a function to avoid out-of-order data packet delivery. However, this contributes only little to this performance study, and thus the detail is not shown in the pseudo-code in this paper.

Decoding Procedure

```
Function Handle_Packets_from_MAC(data)
  if (not innovative(data)) then
    Message_Free(data)
    return
  store data to generation buffer
  Gaussian_Elimination(generation)
  if (decodable packets cause no reordering) then
    Deliver_to_Tranport(newly decoded data packets)
```

Fig. 5 Pipeline Coding—Decoding Procedure

3 EXPERIMENTS RESULTS

We run simulation experiments and testbed experiments. For the simulation, the proposed pipeline network coding scheme was implemented on QualNet 4.5 [15], network simulator. For the testbed, the proposed pipeline network coding was implemented in Linux kernel as a coding element of the Click modular router [12]. The following two sections describe the configuration of the simulation setup and present the simulation results. Section 3.3 further shows the results of the testbed experiments.

3.1 Simulation Scenario

The simulation topologies under study are the single path string topology shown in Fig. 6 and the multipath braided topology shown in Fig. 7. Nodes on both topologies are placed on grids, with grid edge =150m. For each topology, we use a single traffic flow from a single source to a single destination. The generated traffic is CBR/UDP traffic. Note that since the proposed coding scheme exploits a pure-broadcasting channel access protocol, multicast/broadcast can also be supported. Namely, all nodes in the network can hear the coded packets and thus can decode the stream if they are multicasting clients.

For the channel access protocol, standard 802.11b (CSMA/CA) is used in "single path without coding" scenario. For all the remaining cases, a pure-broadcasting scheme is used. Pure-broadcasting means all packets are sent in standard 802.11b broadcast mode. The reason why we do not adopt the pseudo-broadcasting approach proposed in [13] is that such design is not supported by all hardware [14]. However, since RTS/CTS will not be used in broadcast mode, we exploit the idea from [10], where a small amount of random delay is added before delivering the packets to MAC layer. This short random delay effectively avoids phasing, namely the situation where all nodes receive an innovative packet, finish re-encoding, and attempt to send it out at the same instant.

The following subsections will discuss the simulation results of UDP. Further details of the simulation configuration are given in table 2.

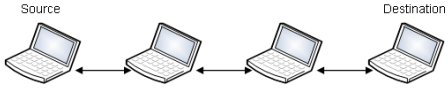

Fig. 6 String Topology

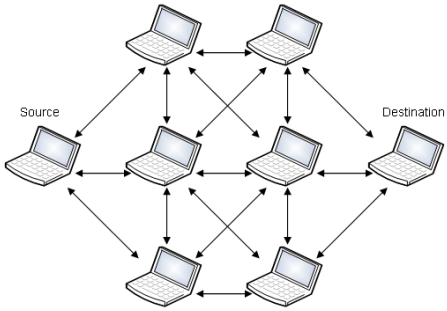

Fig. 7 Braided Topology

Table 2. Simulation Configuration

Parameter	Value
Grid Distance	150 m
Channel Bit-rate	11 Mbps
Access Control	802.11b +RTS/CTS for string topology , no coding; 802.11b broadcast mode, for all other cases.
Transport and Application Layer	CBR/UDP (820Kbps)
Per Link Loss Rate	0%~60%
Packet Size	1500Bytes

3.2 Simulation Result

A CBR application of 820Kbps sending rate is configured at the source in our simulation study. 820Kbps is below the saturation throughput of the 3-hop wireless network, thus it leaves room for redundant packets. Topologies and coding schemes tested using CBR/UDP traffic are summarized in tables 3.

Table 3 CBR/UDP Simulations over String Topology

Traffic Type	Topology	Coding Scheme	Coding Redundancy
CBR/ UDP	String	No coding (Unicast)	No Coding
	Braided	No Coding (Broadcast)	No Coding
		Batch Network Coding	2.5
		Pipeline Network Coding	2.5

Note that the coding redundancy is chosen to be 2.5 based on a previous simulation study, which shows that a redundancy level of 2.5 is about the right level to compensate losses without congesting the network.

The throughput-to-loss plot is demonstrated in Fig. 8. The single path without coding case is included as the base case. As shown in Fig. 8, generally, as the loss rate increases, all throughput curves drop. Also, all braided cases significantly outperform the string without coding. Pipeline network coding performs the best regardless of the link loss rate. The throughput of pipeline network coding shows no degradation for loss rate under 35%. Multipath without coding achieves 2nd highest throughput, which is slightly better than multipath with batch network coding case. This fact was also noted in [16]. Based on these simulation results, we notice that supporting partial decoding of a generation can significantly improve throughput compared to batch network coding.

Fig. 9 presents the delay-to-loss plot for the same set of configurations. As in Fig. 8, the single path no coding case is shown as the baseline case. From Fig. 9, we notice that, multipath without coding has almost the same delay as single path, while Fig. 8 shows that the throughput is greatly improved. Also, pipeline network coding reduces the delay significantly from batch network coding. In addition, we observe that network coding delay increases as the loss rate increases.

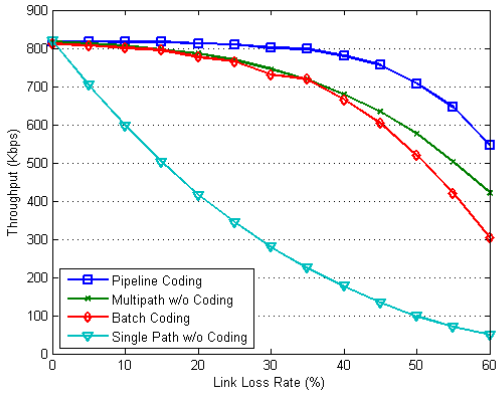

Fig. 8 UDP Throughput (braided topology)

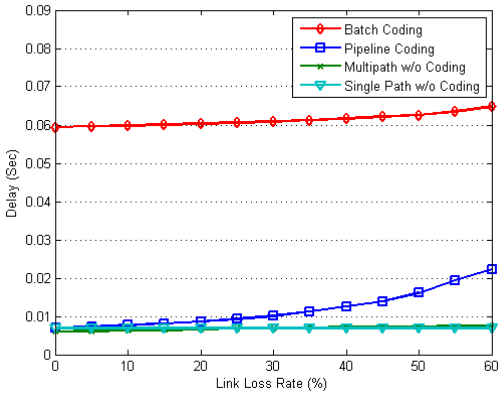

Fig. 9 UDP Delay (braided topology)

3.3 Testbed Experiment Results

The proposed coding scheme is also implemented in Click modular router [12] as a coding element inside Linux kernel. Fig. 10 shows the configuration of our experiment topology. The bit-rate of the source stream is 192 kbps. The source coding redundancy at the streaming server is 2.0 for both batch coding and pipeline coding. Similar to the simulation, we artificially introduce random drops at the receiving sides to emulate a highly lossy scenario.

Fig. 11 shows the packet delivery ratio of one of the clients for the simple multipath without coding, batch coding, and pipeline coding cases. As we expect from the simulation experiments, batch coding does not significantly outperform multipath without coding. Pipeline coding does much better than the two previous schemes and constantly delivers more than 98% of the packets. Fig. 14 shows the video quality in PSNR for these three cases, which shows a significant improvement when using pipeline coding.

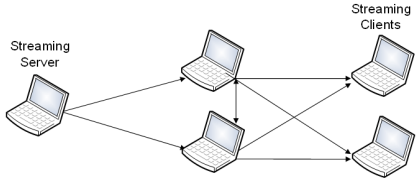

Fig. 10 Testbed Experiment Topology

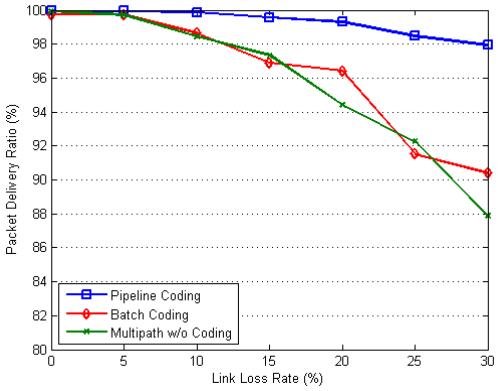

Fig. 11 Packet Delivery Ratio

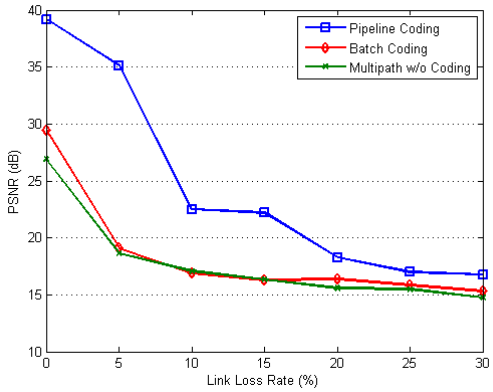

Fig. 12 PSNR

4 CONCLUSIONS

In this paper, we presented a pipeline coding scheme to be incorporated into erasure coding and network coding. Simulation results show that the proposed coding scheme significantly improves UDP streaming throughput in a high loss, multihop wireless scenario. In addition, the delays are greatly reduced with respect to batch coding. Moreover, the pipeline coding scheme is totally transparent to both higher and lower layers, and requires no special hardware support. The results in this paper represent a preliminary exploration of the performance of pipeline coding. From this study, several research issues have emerged which will stimulate further explorations: in particular, the use of pipeline coding to protect from other causes of degradation beyond packet loss (e.g., mobility, jamming etc); the use of pipelining in erasure coding and the comparison with Pipeline network coding, and; the relaxation of the fixed generation concept and its replacement with a variable size “generation window”

5 REFERENCES

[1] S. Chachulski, M. Jennings, S. Katti, D. Katabi, "Trading Structure for Randomness in Wireless Opportunistic Routing," in *Proc. of ACM SIGCOMM 2007*.

[2] V. Kawadia, P.R. Kumar, "Experimental investigations into TCP Performance Over Wireless Multihop Networks," in *Proc. of ACM SIGCOMM 2005*.

[3] J. F. Kurose and K. W. Ross, *Computer Network: A Top-Down Approach 5/e*, Addison Wesley Publishing Co., Inc., Boston, MA, 2010.
[4] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols," in *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 2, pp. 24-36, Apr. 1997.
[5] R. Ahlswede, N. Cai, S.-Y. R. Li, R. W. Yeung, "Network information flow," *IEEE Trans. on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.
[6] R. Koetter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Trans. on Networking*, vol. 11, no. 5, pp. 782–795, 2003.
[7] P. A. Chou, Y. Wu, K. Jain, "Practical Network Coding," in *Proc. of Allerton Conference on Communication, Control, and Computing*, 2003.
[8] A. Fujimura, S. Y. Oh, M. Gerla, "Network coding vs. erasure coding: Reliable multicast in ad hoc networks," in *Proc of MilCom 2008*.
[9] T. Ho, M. Medard, J. Shi, M. Effros, D. Karger, "On randomized network coding," in *Allerton*, 2003.
[10] J.-S. Park, M. Gerla, D. S. Lun, Yu. Yi, M. Medard, "CodeCast: A Network Coding based Ad hoc Multicast Protocol," *IEEE Wireless Communications*, October 2006.
[11] C.-C. Chen, C.-N. Lien, U. Lee, S. Y. Oh, "CodeCast: Network Coding Based Multicast in MANETs," in Demos of the 10th International Workshop on Mobile Computing Systems and Applications (HotMobile 2009), Feb. 2009.
[12] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems* 18(3), pp. 263-297, August 2000..
[13] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, J. Crowcroft, "XORs in the Air: Practical Wireless Network Coding," *IEEE/ACM Trans. on Networking*, vol. 16, no. 3, June 2008.
[14] Y. Huang, M. Ghaderi, D. Towsley, W. Gong, "TCP Performance in Coded Wireless Mesh Networks," in *Proc. of IEEE SECON 2008*.
[15] Scalable Networs Inc. QualNet. http://www.scalble-networks.com.
[16] S. Y. Oh, M. Gerla, "Robust MANET Routing using Adaptive Path Redundancy and Coding," in *Proc of THE FIRST International Conference on COMmunication Systems and NETworkS (COMSNETS)*, January 2009.
[17] D. MacKay, “Fountain Codes,” IEE Proceedings on Communications, vol. 152, no. 6, pp. 1062-1068, Dec. 2005.
[18] A. Shokrollahi, “Raptor Codes,” IEEE Trans. On Networking, vol. 14, pp. 2551-2567, Jun. 2006.