

# A Distributed Codec Placement Algorithm for Network-Embedded FEC

Mingquan Wu and Hayder Radha

Department of Electrical and Computer Engineering

Michigan State University

East Lansing, MI 48823

{wumingqu, radha}@egr.msu.edu

## *Abstract-*

**Network-Embedded FEC (NEF) for overlay and peer-to-peer (p2p) multicast networks was introduced in [8], and the impact of packet loss correlation on the performance of NEF was further analyzed and evaluated in [9]. In these previous works, a centralized codec placement algorithm and a greedy optimization algorithm to place a specified number of NEF codecs in a multicast distribution network were developed and analyzed. In the centralized codec placement algorithm, it has been assumed that the network topology and the loss rate on each branch of the network were known. In reality, it is very difficult to know the loss rate of each branch on the multicast tree beforehand. Further, the topology of a multicast session changes over time as users join and leave the session randomly. In this paper, we design and implement a distributed algorithm to place NEF codecs in the intermediate nodes of a multicast distribution network without knowledge of the network topology; the algorithm can also cope with the dynamics of the network very well. Our simulation results, based on the popular network simulator 2 (ns2), show that the proposed distributed NEF placement algorithm provides very competitive performance when compared to the centralized algorithm and it always outperforms traditional end-to-end FEC.**

## I. INTRODUCTION

Peer-to-peer systems are becoming increasingly popular in applications such as object lookup, content distribution and event notification [1]-[5]. Most of the research efforts in this area are focused on distribution of content to a group of users by employing a store and forward reliable transmission on the top of the underlying unicast transport protocol such as TCP. For realtime applications, however, TCP is not a viable option. As TCP prefers reliable rather than on-time delivery, a detected packet loss may intrigue a significant decrease in data rate, which may cause the quality of the realtime service drop to an unacceptable level.

Realtime applications often can tolerate limited packet losses but have strict time delay constraints, using automatic repeat request (ARQ) for error recovery may add large round trip time (RTT) delay to recover a lost packet; also using ARQ in multicast applications causes the well-known feedback implosion problem; this makes

FEC more appealing for many multicast applications (realtime and non-realtime). For realtime multicast applications that support a large number of receivers, end-to-end FEC may require a significant number of redundant (parity) packets to provide reasonable level of protection and reliability. This leads to a very low channel-coding rate, which in turns lead to very low source (e.g., video) rate.

In [8], we advocate the deployment of Network-Embedded FEC (NEF) for content distribution over peer-to-peer networks. In NEF, FEC encoders and decoders (codecs) are placed in the intermediate nodes of a multicast distribution network. These FEC codecs can detect and recover lost packets in FEC blocks at earlier stages before these blocks reach deeper nodes in the multicast tree or final leaf nodes, thus significantly decrease the undecodable probability and improve the message goodput. As packet losses in the Internet are often correlated and occur in bursts, we further analyzed the impact of packet loss correlation on the performance of NEF [9] and conclude that NEF outperforms end-to-end FEC under packet loss correlation.

In our previous works, we have adopted a centralized codec placement algorithm to analyze and evaluate the performance of NEF, where we implemented a greedy algorithm to place a specified number of codecs in a multicast distribution network. In the centralized codec placement algorithm, we assume that the network topology and the loss rate on each branch of the network are known or can be accurately estimated. In reality, it is very difficult to know the loss rate of each branch on the multicast tree beforehand. In particular, the loss rates could change rather frequently according to network conditions and cross traffics. Furthermore, the topology of a multicast session usually vary over time as users join and leave the session randomly. Moreover, it does not scale to run a centralized algorithm for a large network.

In this paper, we design and implement a distributed algorithm to place NEF codecs in the intermediate nodes of a multicast distribution network without knowledge of the full topology. We show that the algorithm can also cope with the dynamics of the network very well.

The remainder of the paper is organized as follows. Section II describes the design and implementation of the distributed codec placement algorithm. Section III evaluates the performance of the distributed algorithm. A brief summary is presented in Section IV.

## II. ALGORITHM DESIGN AND IMPLEMENTATION

In a large network, we want to limit the number of codecs in a multicast session even if every node is willing to act as codec; too many codecs may cause longer delay penalties and may transmit too many unnecessary packets into the network. In order for a node  $i$  to act as a codec, we require that: first, on average, the node  $i$  must be able to decode FEC blocks successfully; second, the number of children (of node  $i$ ) that require extra parity packets must be above a certain threshold.

In the distributed algorithm, each node  $i$  keeps a link list of *child\_info* data structure; each *child\_info* data structure keeps the information associated with each of its immediate children. The *child\_info* has the following data members:

- *req\_parity\_imm* indicates the parity packets required by the child from its immediate codec upstream;
- *req\_children* indicates the number of nodes in the sub-tree rooted at the node that, on average, can not decode FEC blocks if it does not receive parity packets produced by its immediate codec upstream.

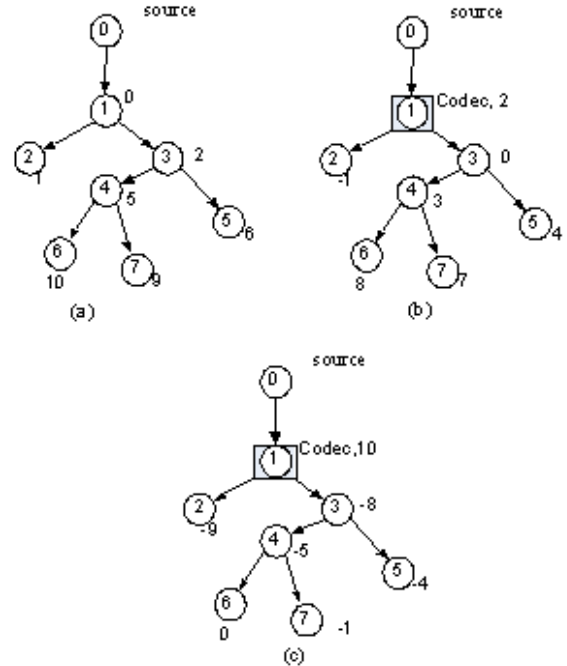
Immediate codec upstream means the closest codec to the node on the path that leads the node to the source. In our distributed algorithm, a node keeps a record for each codec upstream from this node. When a node receives a parity packet produced by a codec, it records its IP address and the hop-count from the codec to itself, so the node can tell which codec is its immediate codec. Each node also keeps the following records about itself:

- *avg\_recv*, an estimate for the average received packets per FEC block;
- *avg\_recv\_imm*, an estimate of the average received packets per FEC block without counting the parity packets it receives from its immediate codec upstream.

The packets that a node receives (*avg\_recv*) consist of message and parity packets, which are sent by the source, and parity packets that are sent by one or more codecs upstream. The estimate of the average received packets can be used by a node to decide if it is a qualified codec candidate. This estimation, however, can not be used to calculate the number of extra parity packets it requires from a codec in order to decode a FEC block.

Let assume that we are using  $RS(n, k)$  Reed Solomon FEC code and  $avg\_recv_i$  is the estimate of the average received packets per FEC blocks of node  $i$ . When there are no codecs upstream,  $k - avg\_recv_i$  is the average

number of extra parity packets needed for node  $i$  to decode an FEC block. If there are already one or more codecs upstream, then  $k - avg\_recv_i$  represents the incremental requirement based on the number of parity packets those codecs have already sent and the changed network conditions.



**Figure 1 the number of parity packets of a node requires changes as codec sends different number of packets**

Figure 1 shows an example. In Figure 1(a), there is no codec in the multicast tree, each number outside the circle is the number of parity packets that node requires; in this case, node 3 requires 2 packets and node 6 requires 10 packets, etc. In Figure 1(b), node 1 receives feedbacks from node 2 and 3 and becomes a codec. Node 1 then produces and transmits 2 extra parity packets downstream. If these parity packets do not suffer from being lost and the network condition does not change, then the average received packets for nodes downstream from node 1 will increase by 2, and  $k - avg\_recv_i$  will decrease by 2. In Figure 1(b), the number outside each circle is updated to reflect this change. In this case, node 3 requires 0 parity packets and node 6 requires 8 parity packets. Figure 1(c) reflects the updated number when node 1 receives the first feedback message from node 6 and produces and sends 10 extra parity packets downstream.

Comparing Figure 1(a) and (b), using node 6 as an example. When  $k - avg\_recv$  is changed from 10 to 8, node 6 does not know whether this change is caused by a codec upstream sending extra parity packets or caused by the improved network condition. Comparing Figure 1(b)

and (c),  $k - avg\_recv$  of node 6 is changed from 8 to 0; again, node 6 does not know if the change is caused by a codec sending a different number of parity packets or by the dynamic network condition--in a distributed environment, a node in general does not know how many parity packet a codec upstream produces and transmits. In either case, it is difficult for node 6's parent, node 4, to update as exactly how many parity packets node 6 is requiring.

In order to circumvent this problem, we require each node to evaluate another estimate: the average received packets per FEC block without counting the parity packets it receives from its immediate codec upstream. If we use  $avg\_recv\_imm$  to represent this estimate, then  $k - avg\_recv\_imm$  represents the number of parity packet a node requires from its immediate codec upstream. In the above example, if the network conditions do not change, then  $k - avg\_recv\_imm$  of node 6 will remain unchanged (as 10) no matter how many parity packets the codec (node 1) produces and transmits. However, if the network conditions are changed,  $k - avg\_recv\_imm$  will be changed to reflect the differences.

In Figure 1(b), the immediate codec upstream of node 6 is node 1. Here, we can see that once node 1 becomes a codec, node 3 can now receive enough packets to become a codec candidate. Node 3 has 4 children that can not decode FEC blocks, if this number passes the threshold we have set, then node 3 becomes a codec. Now the immediate codec upstream of node 6 becomes node 3. After node 3 becomes a codec, it will still send feedback to its parent, node 1; the number of parity packets it requires is not the biggest number required among its children (in this case, 10), but the number of parity packets itself requires (in this case, 2); the number of children that can not decode FEC blocks, however, is the same as that when node 3 is not a codec (in this case, it is 5, including node 3 itself). Node 3, as a codec, will forward all packets transmitted from node 1 to its children, including parity packets produced by node 1. Node 6, now have two codecs upstream, will only estimate the number of parity packets it requires from its nearest codec, node 3. If the network condition remains the same as that when only node 1 is a codec, the number of parity packets node 6 requires from node 3 is 8.

In the following description of the distributed algorithm, we use the node number as index to reference the above parameters of a particular node. For example,  $avg\_recv_i$  represents the average received packets per FEC block of node  $i$ .

Initially, there is no codec in the multicast session. Each node records its average received packets per block in  $avg\_recv$ . At this point,  $avg\_recv\_imm$  and  $avg\_recv$  are equal. The number of parity packets each node requires

from its immediate codec upstream is  $(k - avg\_recv\_imm)$ . The parity packets sent by a codec, however, are suffered from losses. Assuming the loss rate from the nearest codec to node  $i$  is  $r_i$ , then the number of parity packets node  $i$  requires may adjust to:

$$avg\_req_i = (k - avg\_recv\_imm_i) * (1 + r_i)$$

Each parity packet sent by a codec have a sequential number and a group tag, this can help a node to estimate  $r_i$ . Also there is variation for the received packets per FEC block; this is especially true when loss occurs in bursts. Assuming the variation that node  $i$  receives  $avg\_recv\_imm_i$  packets is  $avg\_dev_i$ , then the number of parity packets node  $i$  requires is adjusted to

$$avg\_req_i = (k - avg\_recv\_imm_i) * (1 + r_i) + avg\_dev_i$$

The estimations of the above referred parameters are listed below:

$$\begin{aligned} avg\_recv(K+1) &= (1-g) \times avg\_recv(K) + g \times recv(K+1) \\ avg\_recv\_imm(K+1) &= (1-g) \times avg\_recv\_imm(K) \\ &\quad + g \times recv\_imm(K+1) \\ avg\_err(K+1) &= recv\_imm(K+1) - avg\_recv\_imm(K) \\ avg\_dev(K+1) &= (1-h) \times avg\_dev(K) + h \times avg\_err(K+1) \end{aligned}$$

Here,  $avg\_recv(K)$  represents the average received packets per FEC block up to the time when a node receives the  $K$ th FEC block;  $recv(K+1)$  represents the packets received by a node for block  $K+1$ ,  $recv\_imm(K+1)$  represents the packets received by a node for block  $K+1$  without counting the parity packets sent by the immediate codec upstream of the node. In our estimation,  $g$  is set to 0.1 and  $h$  is set to 0.2.

Let  $max\_req_i$  be the largest number of parity packets required by the node itself and its immediate children;  $sum\_children_i$  be the total number of nodes in the subtree rooted at this node (including itself) that, on average, can not decode FEC blocks if they do not receive parity packets produced by its immediate codec upstream. If node  $i$  is a leaf node, then  $max\_req_i = avg\_req_i$ ; if  $avg\_recv\_imm_i < k$ , then  $sum\_children_i = 1$ , otherwise,  $sum\_children_i = 0$ . If node  $i$  is an intermediate node,  $max\_req_i$  is initialized to  $avg\_req_i$ ; if  $avg\_recv\_imm_i < k$ , then  $sum\_children_i$  is initialized to 1, otherwise, it is initialized to 0. For the intermediate nodes,  $max\_req$  and  $sum\_children$  will be updated when it receives feedback from its children.

Each node sends feedback to its parent periodically. In our algorithm, a node will send feedback each time a new FEC block is received. As an example, if we use  $RS(255,239)$  code, then a node will send a feedback message for every 255 packets it receives.

In the feedback message, node  $i$  reports to its parent  $max\_req_i$  and  $sum\_children_i$ . As this information is passed from the leaf nodes toward the root of the

multicast tree, eventually, each node will know the largest number of parity packets required among the nodes that belong to the subtree rooted at this node; each node will also know the total number of nodes in this subtree that, on average, can not decode FEC blocks if they do not receive extra parity packets from the immediate codec upstream this node.

Figure 2 describes the action taken by a node when it sends a feedback message. Here, a codec will also send feedback; however it will only send to its parent the number of parity packets it requires itself, not the largest number required by its children.

```

Procedure Send_feedback
Node i send feedback its parent node j
IF node i is a leaf node THEN
     $max\_req_i = avg\_req_i$ 
    IF  $avg\_recv\_imm_i < k$  THEN
         $sum\_children_i = 1$ 
    ELSE IF  $avg\_recv\_imm_i \geq k$  THEN
         $sum\_children_i = 0$ 
    END IF
END IF
IF node i is a intermediate node THEN
     $max\_req_i = \max_{m \in C_i} (child\_info_m \rightarrow req\_parity, avg\_req_i)$ 
    IF  $avg\_recv\_imm_i < k$  THEN
         $sum\_children_i = 1 + \sum_{m \in C_i} (child\_info_m \rightarrow req\_children)$ 
    ELSE IF  $avg\_recv\_imm_i \geq k$  THEN
         $sum\_children_i = \sum_{m \in C_i} (child\_info_m \rightarrow req\_children)$ 
    END IF
END IF
IF node i is codec THEN
    Send  $max\_req_i = avg\_req_i$ 
END IF
Send  $max\_req_i, sum\_children_i$  to node j
END Send_feedback

```

**Figure 2** *send\_feedback* procedure

Assume node *j* is node *i*'s parent. When node *j* receives a feedback from node *i*, node *j* will first update node *i*'s information in *child\_info<sub>i</sub>*, then it will make a decision to see if it can act as a codec. If  $avg\_recv_j \geq k$  and the sum of nodes among its children that can not decode FEC blocks is bigger than a certain threshold, say *thresh\_children* (we use the threshold to set a limit on the number of codecs), then the node can make a decision to become a codec. Figure 3 and Figure 4 summarize the action taken by node *j* when it receives feedback from node *i*

```

Procedure Receive_Feedback
Node j receive feedback from node i
 $child\_info_i \rightarrow req\_children = sum\_children_i$ 
 $child\_info_i \rightarrow req\_parity = max\_req_i$ 
Call Procedure Decision_making
END Receive_Feedback

```

**Figure 3** *receive\_feedback* procedure

```

Procedure Decision_making
Node j make a decision whether it should be a codec
 $sum\_children_j = \sum_{m \in C_j} (child\_info_m \rightarrow req\_children)$ 
 $req\_max_j = \max_{m \in C_j} (child\_info_m \rightarrow req\_parity)$ 
IF node j not a codec THEN
    IF  $avg\_recv_j \geq k$ 
        and  $sum\_children_j \geq thresh\_children$ 
        and  $req\_max_j > 0$  THEN
            Set node j a codec
        END IF
ELSE IF node j is a codec THEN
    IF  $avg\_recv_j < k$ 
        or  $sum\_children_j < thresh\_children$ 
        or  $req\_max_j \leq 0$  THEN
            Set node j not a codec
        END IF
    END IF
END Decision_making

```

**Figure 4** *decision\_making* procedure

Once a node becomes a codec, if it can decode an FEC block, it can reconstruct the original FEC block; at the same time, it will produce the largest number of parity packets required by its children (the maximum among *child\_info* → *req\_parity\_imm*) and forward to each child the number of parity packets they have required (*child\_info* → *req\_parity\_imm*). Each node along the forwarding path of the multicast tree also knows the largest number of parity packets required by each of its children (*child\_info* → *req\_parity\_imm*), so the node does not necessarily forward all the parity packets it receives to all of its children, but only forward the number of parity packets each child has requested; this way, the transmission overhead is decreased.

The algorithm runs on top of an overlay multicast routing protocol. The random join and leave of nodes to a multicast session will not affect the performance of the algorithm. For example, when a node first joins the multicast session, its parent will allocate a *child\_info* data structure for this node. The node will estimate its *avg\_recv* and *avg\_recv\_imm* according to whether or not there are codecs upstream. When the node sends

feedback to its parents, the parent will update the information stored in *child\_info* for this node.

If a leaf node leaves a session, the parent of this node will detect this leave (by the routing protocol or other mechanism) and will delete the *child\_info* data structure for this child. If this child has been the node that requested the biggest number of parity packets among all the children, the parent will then select the biggest number of parity packets required among all its other children and send a feedback to its parent.

If an intermediate node leaves a multicast session, the behavior of the parent of this node will be the same as that when a leaf node leaves a session. The children of this node will reset their estimation of *avg\_recv* and *avg\_recv\_imm*. The overlay multicast routing protocol will find new parents for the children. The parents will allocate *child\_info* data structures to these children as if they are newly joined nodes, and the children will restart their estimation of *avg\_recv* and *avg\_recv\_imm* once they receive packets from their new parents.

### III. PERFORMANCE OF DISTRIBUTED ALGORITHM

We evaluate the performance of the distributed NEF codec placement algorithm on an overlay network that has a de Bruijn topology [6]. In [7], the authors evaluated the performance of structured overlay networks in terms of routing distance and fault tolerance. It has been shown that de Bruijn graph outperforms Chord [1] and CAN [3].

Several procedures have been proposed to construct de Bruijn graphs [7][10][11]. Here we choose the *Optimal Diameter Routing Infrastructure* (ODRI) described in [7]. ODRI build de Bruijn graphs incrementally, which is useful when we study the dynamics of the network, where nodes join and leave a multicast session randomly. We build a de Bruijn graph with a degree of 3 and a diameter of 4, with a total of 81 nodes. To construct a multicast tree, we first select a random node as source, each node then routes along the shortest path to the source, until it reaches a node already in the multicast tree.

We implemented the distributed codec placement in network simulator ns2 [12]. The channel code we use is  $RS(255,239)$ ; the packet loss rate on each branch is uniformly distributed between 1% and 3%. The packet loss correlation  $\rho$  is set to 0.5 [9]. The *thresh\_children* in the distributed algorithm is set to 3.

When a node joins a multicast session, it is appended to the multicast tree as a leaf node. When a leaf node leaves the multicast session, its parent just delete its states and update the corresponding parameters. These cases do not have an immediate impact on the performance of NEF. When an intermediate node leaves a multicast session,

however, all its children need to find their new parents. In our simulation, when an intermediate node leaves a multicast session, it will send a message to its parent, so the parent will no longer forward packets to this node, and delete the states kept for this node. This node will also send a leave message to all of its children. All its children then will run the join procedure to reconnect to the multicast tree.

Figure 5 shows the average loss ratio of all the clients in the multicast session of NEF and traditional (end-to-end) FEC for different FEC blocks (or FEC *groups* as indicated by the *x-axis* label). It can be seen that average loss ratio for NEF is always smaller (mostly significantly smaller) than that of FEC. Between block 5 and 6, an intermediate node leaves the multicast session, all its children are affected, and we can observe that there is a spike on the average loss ratio. Overall, traditional FEC losses vary between 4% and 6%, whereas the codec distributed algorithm mostly maintains lower than 2% in average packet losses.

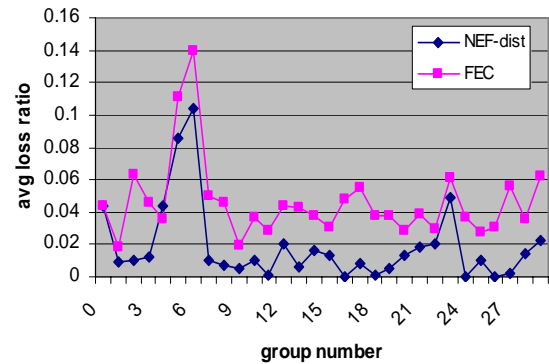


Figure 5 average loss ratio of different groups

Figure 6 plots the received packets of a leaf node for different groups. Again, we can observe the drop in the received packets when an intermediate node leaves the multicast session. We can see that when running NEF, the receiver can receive more than  $k$  packets for most of the blocks, while for end-to-end FEC, the receiver receives less than  $k$  packets for many blocks.

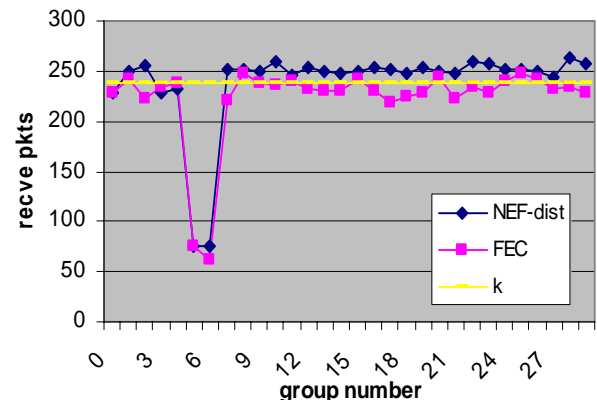


Figure 6 received packets of different groups for a leaf node

Despite the clear superior performance for NEF under the proposed distributed algorithm, NEF codecs send extra packets into the network, and hence it is important to characterize the bandwidth overhead of NEF and end-to-end FEC. Here, we measure the bandwidth overhead in terms of the cost for each message packet received[13]. The cost of a message packet is the product of the average number of packets transmitted per message packet and the average number of links (hops) these packets traversed. The bandwidth overheads of NEF and FEC are the differences between their bandwidth costs and the ideal bandwidth cost, which is measured when there is no loss in the network.

We compare the performance of the distributed codec placement algorithm with the centralized algorithm and end-to-end FEC. In this case, we do not simulate the join and leave scenario. We run the three schemes on the same multicast tree. The FEC codes we used is Reed Solomon codes with  $k = 239$  and  $n$  is increased from 245 to 275 in steps of 5. For the centralized codec placement algorithm, for each FEC code we use, the number of codecs is changed from 1 to 5. In the distributed codec placement algorithm, the condition parameter *thresh\_children* is set to 1, 3 and 7 respectively. In the centralized and distributed codec placement algorithms, if there are multiple implementations that achieve the same average loss ratio, we select the implementation that has the minimum overhead.

Figure 7 shows the average loss ratio versus bandwidth overhead of these three schemes. It can be observed that the two NEF schemes can achieve less average loss ratio with less bandwidth overhead than that of end-to-end FEC. For certain range of overhead (less than 0.07), the distributed algorithm outperforms the centralized algorithm, this is because in the centralized algorithm, codecs always use the same code rate as the one used by the source node, while in the distributed algorithm, codecs use adaptive code rate.

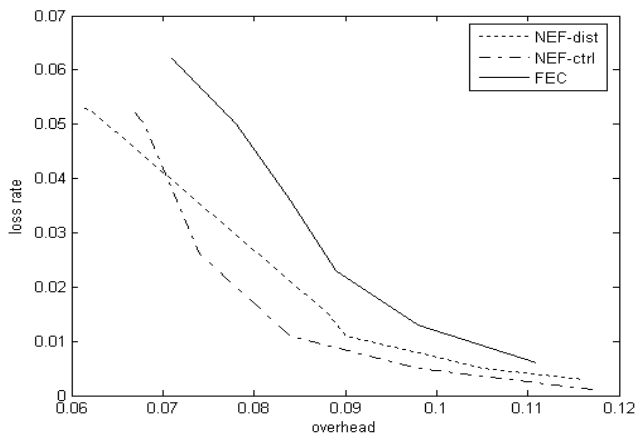


Figure 7 bandwidth overhead versus loss ratio

#### IV. SUMMARY

In this paper, we designed and implemented a distributed codec placement algorithm for NEF. Our simulation shows that the distributed algorithm can cope with the network dynamics very well. We also compare the bandwidth overhead of the centralized and the distributed NEF with that of the end-to-end FEC, and have shown that to achieve the same loss rate, the NEF approach occurs less bandwidth overhead than end-to-end FEC.

#### REFERENCES

- [1] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in Proc of ACM SIGCOMM, August 2001.
- [2] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in Proc of Middleware, Nov. 2001.
- [3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," in Proc. Of ACM SIGCOMM, August 2001.
- [4] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," Technical Report UCB/CSD-01-1141, Computer Science Division, U.C. Berkeley, April 2001.
- [5] L. Cabrera, M. Jones, and M. Theimer, "Herald: Achieving a global event notification service," in Proc. of the Eighth Workshop on Hot Topics in Operating Systems. May 2001.
- [6] N. de Bruijn, "A combinatorial problem," in Proceedings of the section of science, appl. Mathematical Science, Koninklijke Nederlandse Academie van Wetenschappen, pp. 758-764, 1946.
- [7] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh, "Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience," ACM SIGCOMM, August 2003.
- [8] Hayder Radha and Mingquan Wu, "Overlay and Peer-to-Peer Multimedia Multicast with Network-Embedded FEC," IEEE International Conference on Image Processing (ICIP), September 2004.
- [9] Mingquan Wu and Hayder Radha, "Network Embedded FEC for Overlay and P2P Multicast over Channels with Memory," Conference on Information Sciences and Systems (CISS), March 2005.
- [10] M. Naor, U. Wieder, "Novel Architectures for P2P Applications: The Continuous-Discrete Approach," SPAA '03, June 2003.
- [11] D. Malkhi, M. Naor, D. Ratajczak, "Viceroy: A Scalable and Dynamic Emulation of the Butterfly," ACM PODC, 2002.
- [12] <http://www.isi.edu/nsnam/ns/>.
- [13] Sassan Pejhan, Mischa Schwartz and Dimitris Anastassiou, "Error control using retransmission schemes in multicast transport protocols for real-time media," IEEE/ACM Transactions on Networking, vol. 4(3), pp.413-427, June 1996.