**CODED IP: on the Feasibility of IP-layer Network Coding**

Zheng Huang, Xunrui Yin, Xin Wang†, Jin Zhao, Xiangyang Xue  
Department of Computer Science and Engineering  
Fudan University, Shanghai 200433, P. R. China  
†Corresponding Author E-mail: xinw@fudan.edu.cn

*Abstract*—Nowadays, the real practice of network coding in wireline networks is focused on the P2P overlay networks. Although it can help to utilize network resources more efficiently, P2P network coding does not exhibit benefits in terms of the maximum throughput.

Our work aims to implement network coding at the IP layer, which is an idea not fundamentally new, but with little real practice because of the enormous difficulties involved. In this paper we propose CODED IP, a protocol framework that plugs network coding into the current IP stack. Experiments on a 22-node testbed show that CODED IP provides multicast traffic with not only a significantly higher throughput than overlay network coding and naive IP multicast, but also a more balanced load distribution as compared with overlay network coding.

*Index Terms*—IP-layer Network Coding, Switch

I. INTRODUCTION

The research interest in network coding was initially ignited by the work of Alshwede et al.[1] which revealed that by coding at routers one can achieve the optimal throughput of multicast networks. It has later turned out that linear codes [2] are sufficient to achieve this capacity and can be constructed in polynomial time. Ho et al. proposed [3] the so-called “randomized network coding” which allows to implement network coding in a distributive manner.

Since these benefits were revealed in theoretical studies, an intense effort has emerged to implement network coding in the real world. In wireline networks, network coding has almost been implemented at the application-layer overlay networks, for services such as content distribution[4], Video-on-Demand[9] and live media streaming[10].

However, application-layer coding does not exhibit its potential in terms of the maximum throughput, which is the original purpose of network coding. This issue has been well addressed in the literature[5][6]. Wang and Li [5] implemented a real testbed to evaluate the performance of randomized network coding with refer to the downloading time. Their observation was rather pessimistic: “Total downloading time fails to beat the worst possible noncoding protocol.” Based on this, they proposed the open question: “How practical is network coding?”

In our estimate, it is unjustifiable to answer this question in the overlay networks because of its innate deficiency to achieve the multicast capacity as indicated above. A more natural paradigm, however, is at the IP layer.

We aim to estimate the feasibility of the real implementation of network coding at the IP layer. The idea of coding at the IP-layer is not fundamentally new. Noguchi et al.[7] studied the benefits of network coding at network switches in terms of throughput and load balancing effect. Nevertheless, their evaluation was based on computer simulation, not real implementation. More recently, Kumar et al. investigated in [8] the benefits of intra-flow network coding in a multicast switch. They demonstrated that linear network coding can increase the achievable rate region of a multicast switch. Their work was focused on a single multicast switch and in a theoretical fashion. To our best knowledge, the performance of IP-layer coding in a real multicast network has yet not been fully explored.

In this paper, we propose CODED IP, a protocol framework for implementing IP-layer network coding in a single-source multicast network. With CODED IP, network coding is handled at the IP stack automatically and transparent to the application-layer programs at end hosts. The switches in the network take part in coding, which is the key characteristic for the multicast networks to achieve the maximum throughput.

We evaluate CODED IP by implementing a 22-node testbed as a prototype of multicast network to support a file transfer application. We estimate the performance of CODED IP in comparison with P2P network coding and naive IP Multicast.

The main contribution of our work is twofold.

- **CODED IP** presents the first real implementation of IP-layer network coding. Although the idea of IP-layer network coding is not new, there is no prior work on its implementation because of the overwhelming complexities involved in the topic. Our work is the first attempt in this direction.

- We provide quantitative experimental results that echo with the observations in prior (mostly simulation-based) work. Moreover, CODED IP gives insights to the potential challenges in the real application of IP-layer network coding.

II. CODED IP IN A NUTSHELL

CODED IP is a protocol framework for implementing IP-layer network coding in single-source and multiple-sink content distribution networks. Nodes in these networks are required to be equipped with coding functions at the IP layer. There are two kinds of switch nodes in the networks. One
is core switch which encodes and forwards packets to other switches and the other is edge switch which connects directly to end users. CODED IP sits between the IP layer and the transport layer.  

A. Source

CODED IP at the source segments the upper-layer data into generations, each generation containing $K$ packets. The $K$ uncoded packets within one generation are called original packets or original blocks. For each output link, the source produces a linear combination of the $K$ original blocks in the current generation and forwards the resulting packet. The output packet on link $e$ is of the form $p_e = \sum_{i=1}^K c_{ei}p_i$, where $p_i$’s are the original packets in the generation and $c_{ei}$’s are coefficients chosen from the finite field $GF(2^n)$. The vector $\vec{c}_e = [c_{e1}, \ldots, c_{eK}]$ is called the coding vector of the packet $p_e$, which describes how the original packets are mapped to the coded packets.

The source inserts a CODED IP header into each coded packet. The header keeps the information about packet’s generation ID, coding vector and generation size (the number $K$).

B. Forwarder

Forwarders refer to the core switches in the network. For the arrival of each new packet, the forwarder nodes check whether the packet is innovative, i.e., whether it is linearly independent from the previous received packets within the same generation. The checking process is performed according to the global coding vectors in the packets by Gaussian Elimination. Forwarders buffer only the innovative packets in the memory.

At the sending side, forwarder nodes create a linear combination of the coded packets with the same generation ID and send out at output links. A linear combination of the coded packets is of the form $\sum_{i=1}^K c_{ei}p_i$, where $c_{ei}$’s are coefficients chosen by the forwarder nodes, $p_i$’s are the packets received from the incoming links. The resulting $p_e$ can be reformulated in the form $p_e = \sum_{i=1}^K c_{ei}p_i$.

C. Destination

Destination nodes are referring to the edge switches. For the arrival of new packets, destinations do innovativeness checking and buffering in the same manner as forwarder nodes.

Once $K$ innovative packets are received ($K$ can be learnt from the CODED IP header), the destination node begins decoding of the whole generation using matrix inversion:

$$
\begin{pmatrix}
    p_1 \\
    \vdots \\
    p_K
\end{pmatrix} = 
\begin{pmatrix}
    c_{11} & \cdots & c_{1K} \\
    \vdots & \ddots & \vdots \\
    c_{K1} & \cdots & c_{KK}
\end{pmatrix}^{-1}
\begin{pmatrix}
    p_1 \\
    \vdots \\
    p_K
\end{pmatrix},
$$

where $p_i$ is the original packet and $p_i'$ is the received packet with global coding vector $\vec{c}_i = [c_{i1}, \ldots, c_{iK}]$.

After decoding, destination nodes remove the CODED IP headers and forward the original contents to the end hosts within “normal” IP packets.

III. IMPLEMENTATION DETAILS

We explain the implementation details in this section.

A. Packet format

CODED IP inserts a packet header after the IP header, as indicated in Fig. 1.

![Fig. 1. CODED IP Header. The optional CODED IP header is inserted after the IP header and the upper-layer data is encoded.](image)

- **PROTOCOL(8-bit):** The upper layer protocol. It stores the original PROTOCOL field in the IP header.
- **SESSION_ID(8-bit):** The content distribution session the current packet belongs to.
- **SESSION_LEN(16-bit):** The number of total generations in the content distribution session.
- **GEN_ID(16-bit):** Generation ID to which the packet belongs.
- **GEN_SIZE(8-bit):** Generation size. This field indicates the number of original blocks in the generation and the length of coding vector.
- **BLOCK_LEN(8-bit):** The length of coded data.
- **CODING VECTOR(variable length):** A variable-length field. It stores the global encoding vector of the coded data.

Except for the CODING VECTOR, all the fields are initialized by the source node and are copied to the corresponding fields of the packets produced by the forwarder nodes. The coding vectors are computed locally, with the coefficients chosen at each forwarder node.

B. Complexity of network coding

The overhead of coding is a key characteristic for a network coding system. Coding $K$ packets of length $N$ requires $\mathcal{O}(K^2N)$ multiplications and additions.

Two tricks are adopted in CODED IP to ensure low coding overhead.

---

1We would like to point out that similar schemes have been proposed in the wireless case, such as [11].
1) **Check for innovativeness**: Once a new packet is received at an incoming link, the switch nodes (forwarders and destinations) check for its innovativeness. The coding vectors of the received packets are buffered in row echelon form, which can expedite the checking process.

2) **The choice of $K$ and $N$**: The choice of $K$ and $N$ has a great impact on the complexity of coding operation. At the IP-layer, the size of IP packets is limited by the MTU of underlying physical links. In CODED IP we try to avoid IP fragmentation, thus choose the value of $KN$ as below 1500 (1200 and 1400 in our experiment).

**C. Memory management**

For efficient memory management, each CODED IP node maintains state for each content distribution session. For the first packet from a new session, the node determines the maximal number of generations that can be buffered according to its free memory space. Once receiving the first packet of a new generation ID from the session, the node 1) increments a generation counter which indicates how many generations have been buffered for the current session; 2) initializes a pointer array of length $K$, each of whose element will point to the first address of the memory that to be allocated to store packets; 3) allocates a memory space of length $N$ and set the corresponding element of the pointer array; 4) maintains a time counter which indicates the latest time at which a new innovative packet of the generation is received. If the memory space is full, the generation with the earliest time counter will be flushed; 5) For each session, we set a “timeout” threshold. If the time interval since the latest update time among all the generations of the session exceeds the threshold, all the packets buffered for the session will be flushed because it indicates the end of the session.

The per-session states can be summarized as following.

- The $ses_{\text{max}}_{\text{gen}}$ indicates the maximal number of generations that can be buffered for the session.
- The $ses_{\text{gen}}_{\text{counter}}$ is incremented by the reception of the first packet of a new generation.
- Each generation has a $gen_{\text{buf}}_{\text{pointer}}$ which is an pointer array of length $K$ whose element points to the first address of memory space where an innovative packet is to be buffered.
- Each generation has a $gen_{\text{last}}_{\text{updated}}_{\text{time}}$ that tracks the latest time at which an innovative packet is received for the generation.
- The $ses_{\text{time}}_{\text{out}}$ is a threshold indicating the end of the session. The $ses_{\text{time}}_{\text{interval}}$ is the time interval since the $gen_{\text{last}}_{\text{updated}}_{\text{time}}$ among all the generations of the session.

**D. Logical flow**

It is shown in Fig. 2 the logical architecture of CODED IP.

On the receiving side, the nodes (forwarders and destinations) check the **PROTOCOL**, **SESSION_ID** and **GEN_ID** field of the received packets and perform the corresponding operations, which are illustrated in the figure. There is a daemon process at each forwarder node which iteratively increments the $ses_{\text{time}}_{\text{interval}}$ of each session and checks whether it exceeds the $ses_{\text{time}}_{\text{out}}$. The $ses_{\text{time}}_{\text{interval}}$ will be reset to zero once an innovative packet is received for the current session.

On the sending side, the coding operation is triggered by the reception of an innovative packet. Once triggered, the node encodes a packet from the generation to which the innovative packet belongs. Then it calculates the global encoding vector, inserts the CODED IP header and forwards the packet. This operation is iterated on each output link. At the source node, CODED IP iteratively partitions the upper layer data into generations of $K$ packets, as is described in Sec. II.A. For each generation, it encodes and forwards a packet on each output link. The source node stops once there is no data from the upper layer.

Further operation depends on whether the nodes are destinations. For the destination node, it buffers, decodes the original content, removes the CODED IP header and delivers the content in “normal” IP packets to the end hosts.

**IV. Experiment**

We implement a 22-node testbed to evaluate CODED IP, in comparison with both P2P application-layer network coding and single-session IP multicast. We employ a file-transfer application and estimate the performance in terms of throughput (at end hosts) and bandwidth consumption (at switches).
A. Testbed

1) Characteristics: The network topology is shown in Fig. 3. Switches A-F are acting as core switches, while G-I are edge switches which connect 5, 3, 4 end hosts respectively.

![Fig. 3. The 22-node wired testbed. Switch A-F are core switches (forwarders in CODED IP) and switch G, H, I are edge switches (destinations in CODED IP).](image)

2) Hardware: All the nodes are emulated by PCs with Intel Pentium(R) 4 1.80GHz CPU and 512KiB L2 cache. Nodes are connected by the Ethernet links.

3) Software: The source and switch nodes run Linux, with FedoraCore of version 2.6.13. Our implementation of CODED IP runs as kernel modules at the IP layer of Linux network stack. The end hosts run Windows XP SP2.

B. Compared Scheme

- **CODED IP** as explained in Sec. II and Sec. III.
- **P2P** with and without network coding, which is to be described in Sec. C.
- **Single-session IP multicast (SSIM)**, the distribution path of which will be discussed in Sec. D.

C. P2P scheme

We implement two P2P versions, one with and one without network coding.

D. Multicast distribution tree of SSIM

For SSIM, the multicast distribution tree is shown in Fig. 6. The red arrows construct a Steiner tree for the multicast session in our testbed. SSIM’s multicast traffic is sent along the branches of this tree.

E. Setup

In the experiments we run CODED IP, P2P and SSIM in sequence. We take average over 3 runs for each experiment. Each experiment transfers a 16Mbyte file between the same single-source multiple-destination pairs.2

The generation size ($K$) for CODED IP is 3. We vary the block size for CODED IP by $N = 400$ and 476B. As for SSIM, the packet size is 1200 and 1400B. We measure the performance of P2P schemes under three block sizes, which 4

2For CODED IP and SSIM, there is no such a “dynamic” scenario because the switch nodes are stationary and dynamic variations of end users will not affect the performance.

3The distribution scheme for each scheme as explained in Sec C and D are the optimal ones in each case, at least in our view.

4Although the edge switches act as destinations in CODED IP, the ultimate destinations are the end hosts as well.
are 128, 192 and 256KB respectively\(^5\). In P2P coding version, the maximal number of blocks allowed to be coded in one coding operation is 15. In P2P dynamic scenario, each peer is determined every 0.5 second whether it should leave by the given probability 0.05. The time interval for which it sleeps is randomly chosen from 0.5 second to 25 seconds.

All experiments are performed with a fixed sending rate of 45KB/s on each output link at the source node. As TCP is unicast oriented, we choose UDP as the transport layer in CODED IP and SSIM.

For CODED IP, the \(\text{ses\_max\_gen} \) is set to be 150. The \(\text{ses\_time\_out} \) is 3 minutes.

**F. Throughput**

![Fig. 7. Average throughput in KB per second at the end hosts in various configurations. a)CODED IP, SSIM and P2P static scenario; b)CODED IP, SSIM and P2P dynamic scenario. We choose the block size with the minimum downloading time (in Fig. 8) for each scheme.](image)

Fig. 7 shows the average throughput at the end hosts. In contrast, the average downloading time is plotted in Fig. 8. The block size chosen for each scheme in Fig. 7 is the size with minimal downloading time in Fig. 8. The results show that CODED IP significantly outperforms P2P and SSIM. In general case, CODED IP achieves throughput around 130KB/s, which approximates the network capacity. It achieves 2\(x\) higher throughput than SSIM, while its throughput gain over P2P is around 80\% in the steady state.

In our experiments, CODED IP's throughput is around 3 times as high as that of SSIM. This reflects the well-known fact that IP multicast's throughput is limited by the bottleneck link in the multicast distribution tree.

One negative effect of CODED IP worth mentioning is that there is a “jitter” effect from the curve of CODED IP's throughput in Fig. 7. The reason for this effect largely concerns with the mechanism at the sending side in CODED IP. As is described in Sec. II, forwarders in CODED IP need buffer and wait for innovative packets. However, innovative packets may not arrive at a constant speed, mainly because of the different number of hops along different paths between the source and the forwarder node.

**G. Burden on the switches**

We evaluate two performance metrics, namely the “total bandwidth consumption” and the “variance of bandwidth consumption”.

Let \(X_s\) denote the bandwidth consumption at switch \(s\). It is defined as:

\[
X_s = \sum_{e \in \text{In}(s)} X_e,
\]

where \(\text{In}(s)\) is the set of incoming links of switch \(s\), and \(X_e\) is the bandwidth consumed on link \(e\). The total bandwidth consumption is defined as \(X_{\text{tot}} = \sum_{s \in S} X_s\) where \(S\) is the set of all switch nodes in the testbed. Fig. 9 plots the total bandwidth consumption under three schemes.

From the figure, the total bandwidth consumptions of CODED IP and SSIM are much less than those of P2P schemes. The extra bandwidth consumptions with P2P non-coding schemes come from the transmission of buffer maps between the peers and the control packets between the source and peers.

\(^5\)It was suggested in [5] that the block size around 2 – 32KB is optimal. We tested 32KB and 64KB in our testbed, it performed worse than the block sizes that we present here.
In CODED IP and SSIM, the total bandwidth consumptions are roughly the same (116MB Vs. 118MB in Fig. 9). In both schemes there is little communication overhead compared with P2P schemes.

The second performance metric is the variance of bandwidth consumed at each switch, which evaluates how the traffic load is distributed among the switch nodes. Mean bandwidth consumption at switch nodes $\bar{X}$, and the variance $\sigma^2$, are calculated as:

$$\bar{X} = \frac{1}{|S|} \sum_{s \in S} X_s, \sigma^2 = \frac{1}{|S|} \sum_{s \in S} X_s^2 - \bar{X}^2.$$ 

We show the value of $\sigma$ (the standard deviation) under the three schemes in Fig. 10. The packet size that we choose for each scheme is the one with the least bandwidth consumption in Fig. 9.

We thank the anonymous reviewers for helpful advices.

REFERENCES