基于硬件逻辑的网络编码编解码算法

李挥[1]，张明龙[1]，尘福兴[1]，侯韩旭[1]，潘凯[1]，王蔚[1]，袁虎声[2]，孙涛[2]

（1. 北京大学 深圳研究生院 深圳市云计算关键技术和应用重点实验室，广东 深圳 518055；

2. 深圳大学城网络信息中心，广东 深圳 518055）

摘 要：提出了一种基于硬件逻辑实现的通用网络编码编解码算法。编码算法运用随机线性网络编码对数据分组进行编码，解码算法则运用克莱默法则进行解码。对编码器和解码器的算法和结构进行了详细的设计，并最终运用硬件描述语言在 NetFPGA 开发板上实现了该设计。测试结果表明，与传统的路由节点相比，使用线速的网络编码编解码器的网络能够达到最大流最小割定理所确定的流量极限，并且端到端的传输延迟稳定在一个很小的常数上。

关键词：网络编码；编码器；解码器；NetFPGA

中图分类号：TN762 **文献标识码**：A **文章编号**：1000-436X(2012)07-0001-08

Co-decode algorithm of network coding with hardware logic

LI Hui[1], ZHANG Ming-long[1], CHEN Fu-xing[1], HOU Han-xu[1],
PAN Kai[1], WANG Wei[1], YUAN Hu-sheng[2], SUN Tao[2]

(1. Shenzhen Key Lab of Cloud Computing Technology and Application, Shenzhen Graduate School, Peking University, Shenzhen 518055, China;

2. Network & Information Center of Shenzhen University Town, Shenzhen 518055, China)

Abstract: Practical general coder and decoder of network coding (NC) with HDL (hardware description language) logic for wire-speed nodes was presented. The NC coders applied random linear network coding (RLNC) and the decoders recovered the original packets by Cramer's rule. The structures and algorithms of NC coder and decoder were designed in detail and implemented in HDL with NetFPGA boards. Comparing with traditional stored-and-forward mechanism, network emulations showed that networks with wire-speed NC coder and decoder nodes could achieve the capacity bound of max-flow min-cut theorem, and the end-to-end delay was guaranteed on a small constant.

Key words: network coding; coder; decoder; NetFPGA

1 引言

众所周知，网络编码可以显著提高网络的吞吐量[1]。目前，随着研究的不断深入，网络编码的应用已经吸引了大批研究人员。文献[2]提出了一种分布式的方案，它运用了随机线性网络编码并且能够消除节点对网络拓扑集中知识的需要。同时，Bhattad 等人则提出了一种分散化的算法，使得在网络达到最大流量的情况下尽量减小需要编码的数据分组的数量[3]。文献[4]则提出了一个基于遗传算

收稿日期：2011-05-06；**修回日期**：2011-10-25

基金项目：国家重点基础研究发展计划（“973”计划）基金资助项目（2012CB315904）；国家自然科学基金资助项目（61179028）；深圳基础基金资助项目（201005260234A, 201104210120A）；深圳产业化基金资助项目（201006110044A）；广东省自然科学基金资助项目（2011010000923）

Foundation Items: The National Basic Research Program of China (973 Program) (2012CB315904); The National Natural Science Foundation of China (61179028); Basic Research of Shenzhen (201005260234A, 201104210120A); Shenzhen Industry (201006110044A); The Natural Science Foundation of Guangdong Province (2011010000923)

法的方法，它能够降低在多播中使用网络编码对资源的消耗。这些研究推动了网络编码在实际中的应用进程。但是迄今为止，这些研究都是基于软件执行的。更重要的是，编码节点和解码节点都在某些特定的算法之下处理着大量的数据。与传统的存储转发模式相比较，这些算法在一般情况下更加复杂，需要更大的运算量。如果这些工作都由软件来完成，网络就有可能遭受巨大的延迟和不稳定状态，因为在传统的转发节点中，大量数据是由硬件完成的。在某些情况下，需要一个通用的硬件平台来评估网络编码在实际应用中的复杂度或作更为深入的研究。在基于 NetFPGA 的平台上，构造了这样一个通用的具有编解码功能的硬件平台[5,6]。根据文献[7]，在多播交换中同一个流内使用线性网络编码可以使系统有更大的速率范围。与之相比，本文的系统不但可以实现同一个流之间的编码，而且具有不同流之间编码的功能，展现出了更为一般的特性。除此之外，这也是首次在硬件逻辑上研究网络编码及其应用。

在文献[8]的研究中运用随机线性网络编码消除网络中存在的瓶颈并且使之达到了最大流最小割定理所确定的容量极限。在文献[8]所提出的拓扑中有 3 个信源节点和 4 个信宿节点，每个信宿节点同时又是解码节点。编码节点将不同流之间的 2 个数据分组编码成一个数据分组然后发送到转发节点，转发节点则独立地将接收到的数据分组直接转发至信宿节点。每个信宿节点只有在这样的条件下才能完全解码出原始数据分组：在输入的多条数据流之间，至少有一条数据流是未经过编码的。这个特点大大简化了解码的操作，并且只要编码系数不为零它们总是线性无关的。但是，其编码器和解码器却不具有一般性，其应用受到限制并只能用于特定的拓扑中。

本文在第 2 部分中主要分别介绍网络编解码器系统的架构和编解码算法，分析成功解码的概率。第 3 部分提供了编解码器使用硬件逻辑的实现以及在典型拓扑中的测试结果与性能。最后概括了所做的研究并指出未来的工作。

2 架构和方案

2.1 网编解码系统的定义和拓扑图

从以上分析得知，文献[8]所提出的编码器和解码器的方案并不是通用的。由于在端到端的数据传输过程中，数据分组编码次数不能多于一次，并且解码操作依赖于未编码的数据分组。因此一个更为实用的线速编解码器是值得研究的。

设在一个有 m 个信源的网络中，在时隙 t 内信源 $S_1, S_2, \cdots, S_m$ 分别发出数据分组 $X_1, X_2, \cdots, X_m$。在经过编码节点或其他中间的节点时被随机线性地编码到一起，最终数据分组到达接收节点 $R_i(i=1,2,\cdots,n)$，用 $P_1, P_2, \cdots, P_m$ 表示接收到的数据分组。每个编码节点就是一个编码器同时每个接收节点也是一个解码器。如果每个解码器能够成功地恢复出原始数据分组 $X_1, X_2, \cdots, X_m$，那么这就是一个特定网络形式的通用的编解码系统。可以用以下方程组描述上述过程：

$$\begin{cases} P_1 = a_1X_1 + a_2X_2 + \cdots a_mX_m \\ P_2 = b_1X_1 + b_2X_2 + \cdots b_mX_m \\ \vdots \\ P_m = h_1X_1 + h_2X_2 + \cdots h_mX_m \end{cases} \tag{1}$$

为了实现并测试这个通用的编解码方案，拟采用如图 1~图 3 所示的几种不同的拓扑图[9~11]。

在图 1 和图 2 中，有 8 个信源 $S_k(k=1,2,\cdots,8)$和 n 个信宿节点 $R_i(i=1,2,\cdots,n)$。显然，如果中间节点

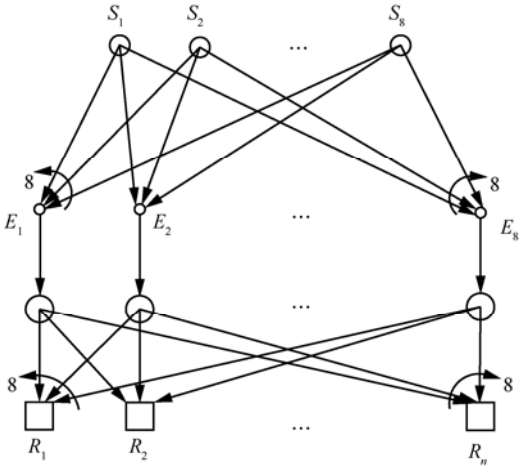


图 1 一个具有 8 个信源和 n 个信宿的网络

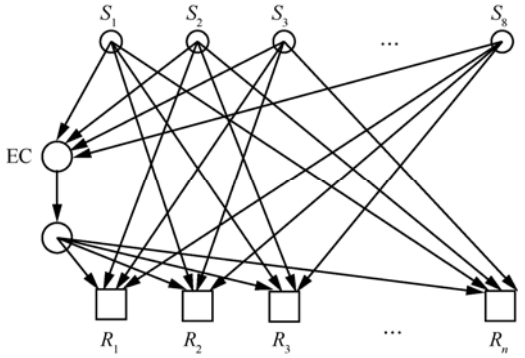


图 2 另外一个具有 8 个信源和 n 个信宿的网络

$E_j(j=1,2,\cdots,8)$和信宿节点没有网络编码和解码的功能，那么，对于每个信宿节点，将不能同时收到 8 个信源发送出来的数据分组。

为了区别同一信源发送出的数据分组，用 $S_{(n,x-1)}$ 表示信源 S_n 发出的第 X 个数据分组(X>0)。

对于图 1 所示的任一编码节点，假设输入向量 $in(E_j)=\begin{pmatrix} S_{(1,x)} \\ S_{(2,y)} \end{pmatrix}$，$j=1,2,\cdots,8$，来自于 2 个不同的输入信道，从伽罗瓦域 GF(256)中随机选择一个本地编码向量$(\alpha\ \beta)$，编码后得到编码数据分组的有效负载为

$$E_j=(\alpha \quad \beta)\begin{pmatrix} S_{(1,x)} \\ S_{(2,y)} \end{pmatrix}=\alpha S_{(1,x)}+\beta S_{(2,y)} \tag{2}$$

在大规模和复杂的网络环境下，数据分组在传输过程中可能不止一次被编码。在方程组(1)中，如果系数矩阵是满秩的，那么就可以解出方程，即可以恢复出原始数据分组。图 3 描述了这种数据分组多次编码的情形：若信源 S_0 和 S_1 要将数据同时发送给信宿节点 R_0 和 R_1，且假设每条链路的容量是相等的，那么在传输过程中，在节点 c、e、f、i 处就会形成瓶颈，显然，要想消除瓶颈的影响，就必须在这 4 个节点处对数据进行编码。可见，数据在到达信宿节点时可能已经发被编码 3 次。

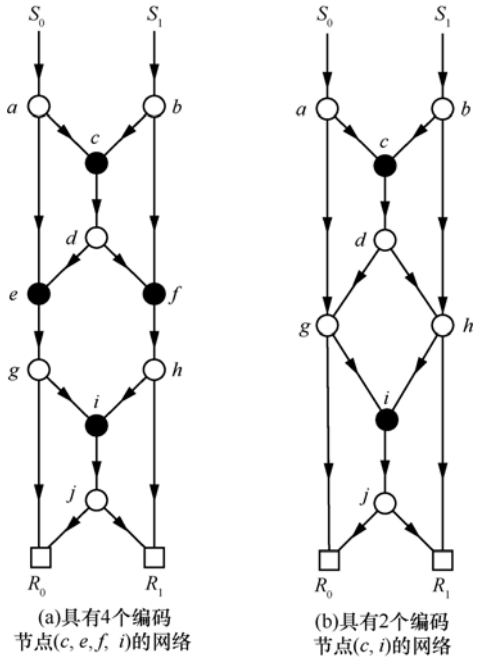


图 3　分组多次编码的网络

2.2　编码方案和结构

对大多数网络来说，数据分组是其最小的传输单元。在此提出了一种对不同信源之间的数据分组进行编码的方案。给定一个网络 $G=(V,E)$，设 $S_i\in V$ $(i=1,2,\cdots,m)$信源，$R_j\in V$ $(j=1,2,\cdots,n)$是信宿。考虑到多次编码的情况，编码的对象是整个 IP 数据分组，包括 IP 分组头和有效负载。编码完成后添加一个新的 IP 数据分组头。为了能够解码，需要在新的 IP 分组头和编码负载之间添加一个新的协议分组头，称之为 MNCP 分组头。新的 IP 分组头是由编码节点产生并且是面向多播的。即其源 IP 地址是本地编码节点的地址，目的 IP 地址是目的组地址。在新 IP 分组头的协议字段，用 254 标记 MNCP 协议。MNCP 分组头记录了一些关键的编码信息，详细内容会在后面讨论。在解码的时候，当数据分组解码完成后就直接丢弃 MNCP 分组头和新的 IP 分组头，直接得到原始数据分组。一个有 MNCP 分组头的编码数据分组称为 MNCP 数据分组，其封装格式如图 4 所示。

MAC 分组头	IP 分组头	MNCP 分组头	编码后负载

图 4　MNCP 数据分组封装格式

由于在网络中传输的数据分组大多数是 IP 数据分组，因此只对 IP 数据分组进行编码，非 IP 数据分组则直接转发。

在某些节点处，此前已经编码过的数据分组可能需要再次编码。在这种情况之下，需要递归地计算出全局编码向量[2]，这是为了解码的需要。因此，编码程序需要更新储存在 MNCP 分组头中的编码向量。以图 3(a)为例，逐步分析数据分组从信源到信宿的编码过程。

假定在节点 e 处，其 2 条输入通道上的数据分组分别是 $S(0,x)$和$(\alpha_c S(0,x)+\beta_c S(1,y))$，而$\alpha_e$ 和β_e 是本地编码向量。则编码后的数据分组可表示为

$$\begin{aligned} O(e)&=(\alpha_e \quad \beta_e)\begin{pmatrix} 1 & 0 \\ \alpha_c & \beta_c \end{pmatrix}\begin{pmatrix} S_{(0,x)} \\ S_{(1,y)} \end{pmatrix} \\ &=(\alpha_e+\beta_e\alpha_c \quad \beta_c\beta_e)\begin{pmatrix} S_{(0,x)} \\ S_{(1,y)} \end{pmatrix} \\ &=(\alpha_e+\beta_e\alpha_c)S_{(0,x)}+\beta_c\beta_e S_{(1,y)} \end{aligned} \tag{3}$$

此处$(\alpha_e+\beta_e\alpha_c)$和 $\beta_c\beta_e$ 是更新后的全局编码系数，需要储存在 MNCP 分组头里面。

同理，节点 f 发送出的数据分组可表示为

$$O(f)=\begin{pmatrix}\alpha_f & \beta_f\end{pmatrix}\begin{pmatrix}0 & 1\\ \alpha_c & \beta_c\end{pmatrix}\begin{pmatrix}S_{(0,x)}\\ S_{(1,y)}\end{pmatrix}$$

$$=\begin{pmatrix}\beta_f\alpha_c & \alpha_f+\beta_c\beta_f\end{pmatrix}\begin{pmatrix}S_{(0,x)}\\ S_{(1,y)}\end{pmatrix}$$

$$=\beta_f\alpha_c S_{(0,x)}+\left(\alpha_f+\beta_c\beta_f\right)S_{(1,y)} \tag{4}$$

节点 i 发送的数据分组为

$$O(i)=\begin{pmatrix}\alpha_e & \beta_e & 0 & 0\\ 0 & 0 & \alpha_f & \beta_f\end{pmatrix}\begin{pmatrix}1 & 0\\ \alpha_c & \beta_c\\ 0 & 1\\ \alpha_c & \beta_c\end{pmatrix}\begin{pmatrix}S_{(0,x)}\\ S_{(1,y)}\end{pmatrix}$$

$$=\begin{pmatrix}\alpha_e\alpha_i+\beta_e\alpha_c\alpha_i+\beta_i\beta_f\alpha_c & \alpha_f\beta_i+\beta_f\beta_i\beta_c+\beta_c\beta_e\alpha_i\end{pmatrix}\cdot\begin{pmatrix}S_{(0,x)}\\ S_{(1,y)}\end{pmatrix}$$

$$=\left(\alpha_e\alpha_i+\beta_e\alpha_c\alpha_i+\beta_i\beta_f\alpha_c\right)S_{(0,x)}+\left(\alpha_f\beta_i+\beta_f\beta_i\beta_c+\beta_c\beta_e\alpha_i\right)S_{(1,y)} \tag{5}$$

对于接收节点 R_0，假定 2 个输入端口的数据分组为 P 和 K，那么可以用以下方程组来表示：

$$\begin{cases}P=\beta_f\alpha_c S_{(0,x)}+\left(\alpha_f+\beta_c\beta_f\right)S_{(1,y)}\\ K=\left(\alpha_e\alpha_i+\beta_e\alpha_c\alpha_i+\beta_i\beta_f\alpha_c\right)S_{(0,x)}+\left(\alpha_f\beta_i+\beta_f\beta_i\beta_c+\beta_c\beta_e\alpha_i\right)S_{(1,y)}\end{cases} \tag{6}$$

可见，对于不同的拓扑结构，不管是如图 1 和图 2 所示的数据只在编码节点被编码了一次，还是如图 3 所示被多次编码，在编码节点计算出全局编码向量并更新 MNCP 分组头后，总能转化为方程（1）的形式。也就是说，编码系统是不依赖于拓扑结构，是通用的。

对于所有的编码节点，如果有来自于不同信源的 IP 数据分组需要在相同的时隙内转发到相同的输出链路上时，就进行编码，否则只加上 MNCP 分组头后直接转发。编码的算法见算法 1。

算法 1 编码 IP 数据分组

1) for 每个输入的数据分组 do
2) if (是非 IP 数据分组) then
3) 直接转发；
4) else if (只有一个 IP 数据分组或其他端口是非 IP 数据分组) then
5) 在 IP 数据分组中添加 MNCP 分组头后直接转发；
6) else
7) 如方程(2)所示编码数据分组；
8) 按图 5 的形式产生 MNCP 分组头；
9) 按图 4 所示封装数据分组；
10) 将封装好的数据分组排队输出；
11) end if

为了能够解码，需要一个存储编码信息的 MNCP 分组头，其格式如图 5 所示。下面对其字段的含义逐一解释。

版本号：MNCP 协议的版本号，依据此前的文献[8]，此处应为$(0010)_2$。

分组头长度：MNCP 分组头的长度，单位是 4byte。

总长度：整个 MNCP 数据分组的长度，包括 MNCP 分组头，单位是 byte。

标志：指明 MNCP 数据分组的类型，“01”代表一个只加了 MNCP 分组头的数据分组，而负载没有编码，“10”代表负载编码过后的数据分组。

信源号：指明被编码的数据分组来自于哪个信源。

原始长度：未编码之前的 IP 数据分组的原始长度。

序号：同一个信源的数据分组的顺序。

编码系数：随机地产生于 GF(256)中用于对数据分组的负载编码。当一个已经编码过的 MNCP 数据分组再次被编码时，需要更新该值。例如，在节点 i 处，根据方程(5)，若本地编码系数分别是α_i和β_i，则系数重新计算更新后，应将 $\alpha_e\alpha_i+\beta_e\alpha_c\alpha_i+\beta_i\beta_f\alpha_c$ 和 $\alpha_f\beta_i+\beta_f\beta_i\beta_c+\beta_c\beta_e\alpha_i$ 保存到 MNCP 分组头中。

图 6 所示为编码器的结构。数据接收模块连接着物理层的数据收发器。如果数据分组需要进行编码，那么首先进入缓存，否则输入仲裁直接将其发送输出端口，然后根据数据分组的目的地址转发出去。编码模块对数据分组的负载运用随机线性的方法编码，然后再封装成 MNCP 数据分组。整个处理过程在编码控制模块的控制下完成。最后，输出队列模块对需要发送的数据分组

版本号 (4bit)	分组头长度 (4bit)	总长度 (16 bit)	标志 (2bit)	保留 (6bit)	第 1 个信源号 (4bit)	第 2 个信源号 (4 bit)	…	第 8 个信源号 (4bit)	原始长度 (12bit)	序号 (12bit)	编码系数(8bit)	原始长度 (12bit)	序号 (12bit)	编码系数(8bit)	…	编码有效负载

图 5 MNCP 协议的分组头格式

进行排队，然后发送给 MAC 层的数据发送器数据输出模块。

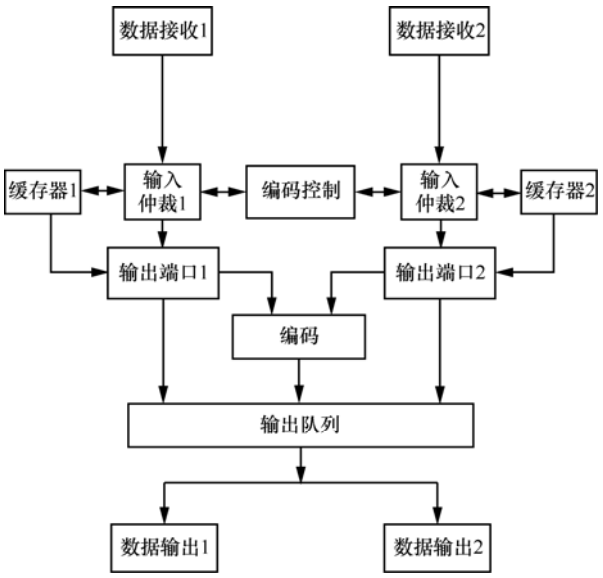


图 6　编码器的硬件结构

2.3　解码方案与结构

信宿节点能够成功解码概率的大小是非常值得关注的。在图 1 中，每个编码节点都在独立地执行随机线性网络编码的功能。根据文献[2,12]对于任一信宿节点，编码系数线性无关的概率至少是 $1-2^{-8}=0.996$。而在图 2 中，这个概率是 1。在图 3(a)中，对于信宿 R_0，若方程(6)可解，则其编码系数应线性无关，即

$$\begin{vmatrix} \beta_f\alpha_c & \alpha_f+\beta_c\beta_f \\ \alpha_e\alpha_i+\beta_e\alpha_c\alpha_i+\beta_i\beta_f\alpha_c & \alpha_f\beta_i+\beta_f\beta_i\beta_c+\beta_c\beta_e\alpha_i \end{vmatrix} \neq 0$$

化简，得

$$\alpha_e\beta_i\alpha_f+\alpha_e\beta_i\beta_f\beta_c+\alpha_c\alpha_f\beta_e\beta_i\neq 0 \tag{7}$$

设 $\alpha_e\beta_i\alpha_f+\alpha_e\beta_i\beta_f\beta_c+\alpha_c\alpha_f\beta_e\beta_i=T$，那么，$T$ 不为零的概率 $P(T\neq0)=1-P(T=0)$。其中，每个系数都均匀地分布在域 F_{2^8} 内(除 0 之外)。因此可先计算出 $P(T=0)$，过程如下

$$\alpha_e\beta_i\alpha_f+\alpha_e\beta_i\beta_f\beta_c+\alpha_c\alpha_f\beta_e\beta_i=0 \tag{8}$$

根据伽罗瓦域内的运算规则，可表示为

$$\alpha_e\alpha_f+\alpha_e\beta_f\beta_c=\alpha_c\alpha_f\beta_e$$

因此

$$\alpha_f=\frac{\alpha_e\beta_f\beta_c}{\alpha_e+\beta_e\alpha_c} \tag{9}$$

其中，α_e、α_c、β_f、β_c 和 β_e 都是非零元素。根据有限域的性质，任何元素都可以表示为指数形式，假设 $\alpha_e=g^h$，$\alpha_c=g^k$，$\beta_f=g^w$，$\beta_c=g^t$，$\beta_e=g^v$，故

$$\alpha_f=\frac{g^{h+w+t}}{g^h+g^{v+k}} \tag{10}$$

而 α_f 存在并且是唯一的。因此 $P(T=0)=1/255$。容易得到 $P(T\neq0)=0.996$。

经以上分析，数据分组虽然被编码了多次，但是其线性相关性还是可以满足实际应用的要求。

在实际的网络中，数据分组并非是同步地流过整个网络。事实上，数据分组在网络中会有不同程度的延迟，比如传输和排队引起的延迟。由于以上原因，用来临时存储数据分组的输入缓存是必要的。

由于受 NetFPGA 开发板端口数量的限制，因此每个解码节点只有 2 个输入端口，故需要 2 个输入缓存。为了能在解码时快速找到缓存中的数据分组，本文用内容可寻址存储器(CAM)存储数据分组的信源号和序号。通过查找 CAM 中的信源号和序号，就可得到该数据分组在缓存中的地址。通过读取该地址中的内容，可以得到想要的数据分组。

为了能尽可能多地解码出数据分组，根据数据分组存储地址的控制逻辑来决定当前需要解码的数据分组。该逻辑在输入缓存间遵循轮回循环策略，在数据分组的地址上采取逐步增大的策略。

对于一个两信源的网络，无论其拓扑如何，也无论数据分组被编码几次，信宿节点接收到的数据分组总是可以用方程(1)来描述。解方程(1)有很多方法，如高斯消元法、逆矩阵相乘法等。为了在硬件逻辑上便于实现，采取克莱默法则，即对于方程(1)，假定 $\begin{vmatrix} c & d \\ a & b \end{vmatrix}\neq 0$，那么

$$X_t=\frac{\begin{vmatrix} P & d \\ K & b \end{vmatrix}}{\begin{vmatrix} c & d \\ a & b \end{vmatrix}},\quad Y_t=\frac{\begin{vmatrix} c & P \\ a & K \end{vmatrix}}{\begin{vmatrix} c & d \\ a & b \end{vmatrix}}$$

对于方程(6)，如果不等式(7)成立，那么

$$S_{(0,x)}=\frac{\begin{vmatrix}\alpha_f+\beta_c\beta_f & P\\ \alpha_f\beta_i+\beta_f\beta_i\beta_c+\beta_c\beta_e\alpha_i & K\end{vmatrix}}{\begin{vmatrix}\beta_f\alpha_c & \alpha_f+\beta_c\beta_f\\ \alpha_e\alpha_i+\beta_e\alpha_c\alpha_i+\beta_i\beta_f\alpha_c & \alpha_f\beta_i+\beta_f\beta_i\beta_c+\beta_c\beta_e\alpha_i\end{vmatrix}}$$

$$S_{(1,y)}=\frac{\begin{vmatrix}P & \beta_f\alpha_c\\ K & \alpha_e\alpha_i+\beta_e\alpha_c\alpha_i+\beta_i\beta_f\alpha_c\end{vmatrix}}{\begin{vmatrix}\beta_f\alpha_c & \alpha_f+\beta_c\beta_f\\ \alpha_e\alpha_i+\beta_e\alpha_c\alpha_i+\beta_i\beta_f\alpha_c & \alpha_f\beta_i+\beta_f\beta_i\beta_c+\beta_c\beta_e\alpha_i\end{vmatrix}}$$

由以上表达式，可以运用几个加法器和乘法器实现上述的数学运算。

可见，解码系统对于 2 个信源的任意拓扑来说，都可以恢复原始数据分组，是 2 个信源拓扑结构的通用解码系统。

为了清楚地表达整个解码过程，算法 2 给出了具体的解码算法。

算法 2 解码数据分组

N_{DP} 为下一个需要解码的数据分组；

C_{DP} 为当前正在解码的数据分组；

P_N 为未解码数据分组的数量。

1) while (P_N≠0) do
2) if (C_{DP} 已经解码) then
3) $C_{DP} = N_{DP}$;
4) $N_{DP} = N_{DP} + 1$;
5) Return to 2);
6) else if (系数线性相关) then
7) 丢弃该数据分组;
8) $P_N = P_N - 1$;
9) $C_{DP} = N_{DP}$;
10) $N_{DP} = N_{DP} + 1$;
11) Return to 1);
12) else
13) 执行解码运算;
14) 输出解码后的数据分组;
15) $P_N = P_N - 1$;
16) $C_{DP} = N_{DP}$;
17) $N_{DP} = N_{DP} + 1$;
18) Return to 1);
19) end if
20) end while

解码器的内部结构如图 7 所示。输入仲裁模块从数据接收端口接收数据。如果接收到的是 MNCP 数据分组，则发送到模块输入控制缓存，从而将数据分组缓存到输入缓存中，否则将数据分组发送至输出端口模块，然后根据其目的地址将数据分组发送出去。同时，对于一个 MNCP 数据分组，控制解码在存储器中保存了它的信源号和序号。除此之外，该模块还完成对执行解码操作的解码模块的控制。

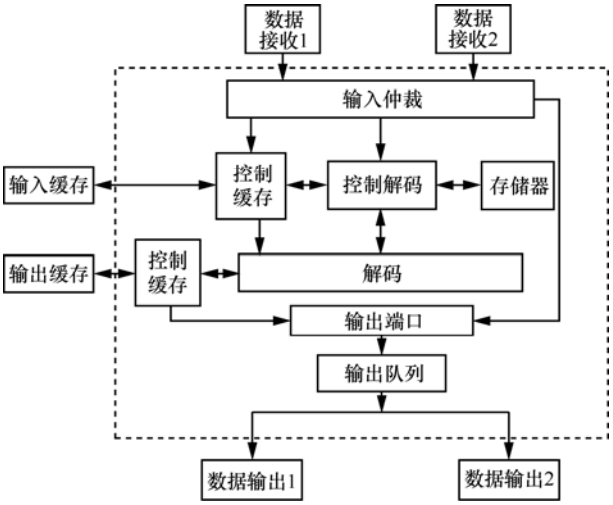


图 7 解码器内部结构

为了能够加速运算速度，本文设计了一种由查找表设计的运算单元，它能够快速完成有限域内的运算。当数据分组完成解码之后，依照其序号的大小顺序在模块输出控制缓存的控制下存入到输出缓存内。当数据输出时，从输出缓存中按地址大小的顺序读出数据分组，这样就克服了数据分组在解码之后乱序的问题。

3 测试结果

本文用 Verilog 语言实现了上述通用的网络编解码器，同时设计基于由斯坦福大学提供的开源开发板 NetFPGA。此外，本文几乎重新设计了其 IP 层的逻辑并且充分运用了现有 MAC 层的设计。在时钟频率为 125MHz 的情况下，成功地在目标器件上综合出设计方案。但是受 FPGA 内部资源的限制布局布线无法完成。因此，本文不得不精简了部分逻辑并且把 CAM 的深度降为 64。这在一定程度上影响了分组丢失率和吞度量。可见，完全运用硬件逻辑实现网络编码，会消耗较大的硬件逻辑资源。

运用 IXIA 公司的网络测试仪 Optixia XM12 模拟信源并且捕获解码后的数据分组。同时，在相同的网络拓扑之下，也测试了基于 IPv4 原理的参考路由器网络的性能。它是基于存储转发的传统模式工作的。

在不同的发送速率下，主要关注数据分组的丢失率和延迟时间。限于篇幅，在此只展示了图 3(a)的测试结果，图 1 和图 2 的测试结果是类似的。在试验中，每次发送 5 000 000 个数据分组，测试结果如图 8 和图 9 所示。

在图 8 中，由通用网络编解码器组成的网络分组丢失率约为 0.5%，并且在速率增大的情况下波动较小。而对于 IPv4 参考路由器，在发送速率超过 500Mbit/s 时分组丢失率会剧烈上升，这是由于某些节点已变成瓶颈。

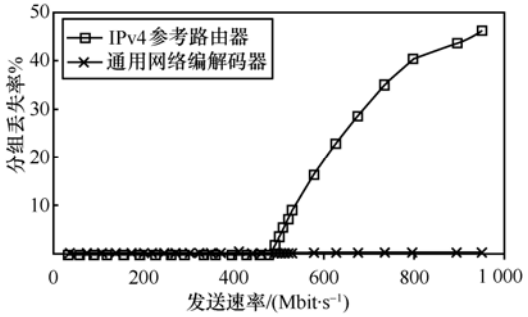


图 8　网络编码和 IPv4 系统的分组丢失率对比

图 9 展示了数据分组的平均时延，由图可以看出，网络编码系统的时延随着发送速率的增大而减小，最后稳定到了 0.4ms。在速率较低时，延迟较大的原因是：为了克服网络的不同步性和提高解码率，解码器输入缓存中需要维持一定数量的数据分组。这样就造成了解码的延迟时间较大。然而，IPv4 参考路由器的延迟时间在发送速率超过 500Mbit/s 时突然增大到 10ms，这是瓶颈的产生所导致的。

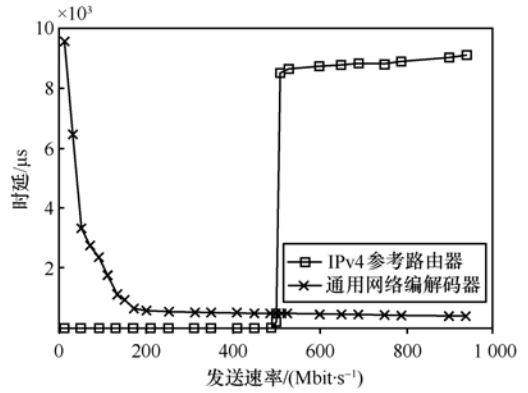


图 9　网络编码和 IPv4 系统的时延对比

4　结束语

本文设计了多信源多信宿的通用的网络编码硬件平台，在提出其硬件结构和编解码算法的同时用 12 000 行 HDL 代码实现本设计。最后在不同拓扑的网络中测试其性能并与传统的 IPv4 路由器做比较。由于硬件系统的线速和较高稳定性的特点，系统在较小延迟的情况下达到了最大流最小割定理所确定的容量极限。

从结果可以看出，尽管本系统在多信源多播网络中展现出了前所未有的优势，但其硬件复杂度与传统的路由器相比较还是比较高的。因此，在不降低性能的同时如何减小硬件复杂度和提高资源利用率值得未来进一步研究。

参考文献：

[1] AHLSWEDE R, CAI N, LI S Y, *et al*. Network information flow[J]. IEEE Trans on Information Theory, 2000, 46(4): 1204-1216.

[2] CHOU P A, WU Y, JAIN K. Practical network coding[A]. Allerton Conference on Communication, Control, and Computing, Monticello[C]. 2003.

[3] BHATTAD K, RATNAKAR N, KOETTER R, *et al*. Minimal network coding for multicast[A]. Proceedings of International Symposium on Information Theory[C]. 2005. 1730-1734.

[4] KIM M, MEDARD M, AGGARWAL V, O'REILLY U, *et al*. Evolutionary approaches to minimizing network coding resources[A]. IEEE INFOCOM[C]. 2007.1991-1999.

[5] http://www.netfpga.org/[EB/OL].

[6] GIBB G, LOCKWOOD J, NAOUS J, *et al*. NetFPGA an open platform for teaching how to build gigabit-rate network switches and routers[J]. IEEE Transactions on Education, 2008,51(3):364-369.

[7] SUNDARARAJAN J K, MEDARD M, KIM M J, *et al*. Network coding in a multicast switch[A]. IEEE INFOCOM[C]. 2007.1145-1153.

[8] ZHANG M L, LI H, LI Y N, LI S Y R. Hardware prototyping of network coding in HDL[A]. The 6th International Conference on Wireless Communications, Networking and Mobile Computing[C]. 2010.1-4.

[9] CHEKURI C, FRAGOULI C, SOLJANIN E. On average throughput and alphabet size in network coding[J]. IEEE Transactions on Information Theory, 2006, 52(6):2410-2424.

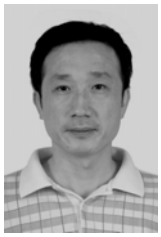
[10] FRAGOULI C, SOLJANIN E. Network coding applications[J]. Foundations and Trends in Networking, 2007, 2(2):135-269.

[11] FRAGOULI C, SOLJANIN E. Network coding fundamentals[J]. Foundations and Trends in Networking, 2007, 2(1): 1-133.

[12] FRAGOULI C, WIDMER J, BOUDEC J Y L. Network coding: an instant primer[J]. ACMSIGCOMM Computer Communication Review

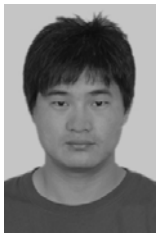
archive, 2008, 36(1):63-68

作者简介：



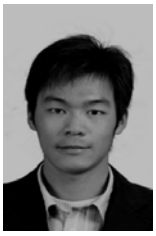
李挥（1964-），男，广东汕头人，博士，北京大学研究生院教授、博士生导师，主要研究方向为三网合一流媒体云计算网络视频关键技术研发、下一代网络体系结构研究、网络路由和宽带交换结构、网络编码理论及其应用、嵌入式系统开发。

张明龙（1982-），男，甘肃武威人，北京大学硕士生，主要研究方向为网络编码及其在多信源网络中的应用。



尘福兴（1986-），男，山东单县人，北京大学博士生，主要研究方向为网路编码、代数交换、路由交换结构及虚拟化。

侯韩旭（1987-），男，安徽砀山人，北京大学博士生，主要研究方向为多信源网络编码理论及应用、三网合一流媒体云计算网络视频关键技术研发。



潘凯（1986-），男，江苏常州人，北京大学博士生，主要研究方向为网络编码、网络协议。

王蔚（1982-），女，湖北当阳人，北京大学博士后，主要研究方向为感知无线电、网络编码和资源优化算法。

袁虎声（1981-），男，广东龙川人，深圳大学城网络信息中心高级工程师、副主任，主要研究方向为计算机网络与系统。

孙涛（1980-），男，山西大同人，深圳大学城网络信息中心工程师，主要研究方向为网络技术与信息系统。