Haste: Practical Online Network Coding in a Multicast Switch

Shuang Yang, Xin Wang
School of Computer Science
Fudan University, China
{*06300720227, xinw*}*@fudan.edu.cn*

Baochun Li
Department of Electrical and Computer Engineering
University of Toronto, Canada
bli@eecg.toronto.edu

***Abstract*—The use of network coding has been shown to improve throughput in input-queued multicast switches, but not without costs of computational complexity and delays. In this paper, we investigate the design of efficient online network coding algorithms in a switch with multicast traffic. We present *Haste*, an online opportunistic coding algorithm designed to streamline the computation when network coding is involved in a network switch with multicast traffic. *Haste* enjoys the advantage of incurring no decoding delays, which reduces packet delays compared with existing network coding algorithms on switches. We have conducted extensive simulations to show the efficiency of *Haste*, and implemented an emulation framework to emulate input-queued switches using asynchronous network sockets. Our emulation framework is able to process actual UDP traffic using *Haste* with online network coding, and to show convincing evidence that *Haste* is suitable for practical use, and is beneficial in multicast switches.**

I. Introduction

Network coding [1] has recently been proposed in input-queued (IQ) switches to improve the throughput of multicast [2], [3]. Traditionally, it was shown that [4] 100% throughput cannot be achieved in IQ switches without a *speedup* (the ability to receive more packets within the same time for output ports), which needs to be implemented in hardware by adding extra switching fabric. It has been shown that network coding is able to effectively substitute the speedup in the switch [2], [3].

Despite its proven theoretical superiority, network coding is certainly not a panacea. Would it be practical to use linear network coding in a typical switch in lieu of extra switching fabric? It would certainly be good news if that is the case, but Fig. 1 has clearly illustrated some challenges when using network coding in a switch. Although all three output ports receive two packets, none of these packets can be sent out of the switch, since they have not been decoded yet. They have to be stored in the output buffer waiting for another coded packet in order to successfully decode. Consequently, a considerable amount of delay is incurred. In contrast, in a switch without network coding, packets arriving at the output ports can be immediately sent out without the buffering delay needed for decoding. In addition, since packets are used for encoding and decoding, they cannot be dropped from either the input or the output buffers. This requires a larger buffering capacity operating at line speed.

To make matters worse, network coding requires a substantial amount of computation, which is unacceptable for core Internet

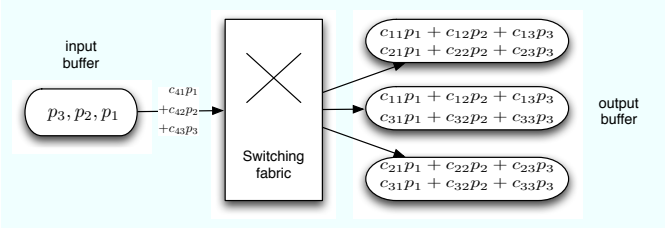


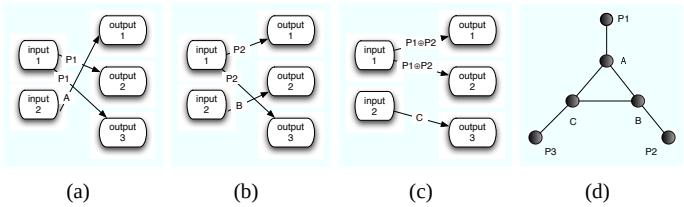
Fig. 1. Network coding in a multicast switch (c represents coding coefficients, and p represents packet payloads).

switches dealing with a very high volume of traffic at high sustainable rates. In the Cisco Catalyst 6500 series, for example, the switching throughput can be as large as 720 Gbps [5]. Consider the linear way of encoding, $E = \sum_i^n c_i \cdot p_i$, where n is the number of packets arriving in a unit of time. Every byte needs at least n multiplications on a finite field. Even if there is only 0.1% traffic needed to be encoded, and n is chosen to be a very small number such as 10 (even though a larger n is desirable in terms of throughput), the switch requires at least billions of multiplications per second just for encoding. In general, the additional buffering, delay, and computation that are necessary for network coding to be deployed in switches may not be easily justifiable or practical.

In this paper, we propose *Haste*, a practical design of opportunistic coding for a multicast switch. *Haste* is designed to mitigate or overcome the challenges when conventional network coding is used. It only uses XOR operations for encoding and decoding, incurring an extremely low computational complexity. Further, packets sent are always chosen to be immediately decodable, resulting in zero decoding delay.

To show the efficiency of *Haste*, we have not only conducted extensive simulations, but also implemented an emulation framework using cross-platform asynchronous sockets beyond the OS kernel, in order to emulate IQ switches with network coding. Our experiments will show that *Haste* performs much better than the existing algorithms, achieving higher throughput and lower packet delays.

The remainder of the paper is organized as follows. Sec. II introduces the preliminaries of online network coding in a multicast switch. In Sec. III, we present the design of *Haste* in the switch. Sec. IV analyzes the performance of Haste, compared with conventional network coding. Sec. V presents our simulation and emulation experiments. Finally, Sec. VI concludes this paper.



(a) (b) (c) (d)

Fig. 2. Network coding improves the rate of multicast flows in a switch: (a)-(c): the use of network coding; (d) the enhanced conflict graph.

II. Preliminaries and Related Work

An $M \times N$ IQ switch consists of M input ports and N output ports. One input port is allowed to send the same multicast packet to several output ports at one time, but different packets cannot be sent simultaneously. One output port can only receive one packet at the same time.

Fanout splitting is allowed in the multicast switch, such that one multicast packet can be sent to part of its destination ports. With $(2^N - 1)$ fanout splitting sets in total, $(2^N - 1)$ virtual output queues (VOQs) are used to support multicast traffic from each input port [6]. The Cisco Catalyst 6000 [5] is one of the real-world switches using VOQs. With network coding, VOQs are not restricted to be FIFO.

Considering fixed-size packets only, network coding is proved [2] to improve the multicast throughput of an IQ switch. Fig. 2(a)–(c) have shown an example given in [2] how network coding works. Conventionally, packet $P1$ and $P2$ cannot transmit simultaneously in one time slot. If we have to reserve an output port for traffic from input 2, at least 4 slots are needed to serve these packets. When network coding is used, it can be finished in 3 time slots.

The *Enhanced conflict graph* [2] is derived for the example, shown in Fig. 2(d). In this graph, a node denotes either a *subflow*, a part of a multicast flow that goes to a particular output port, or a unicast flow. Subflows or unicast flows that cannot be served simultaneously are in conflict with each other with an edge drawn in the enhanced conflict graph connecting the corresponding nodes. For example, $P1$ and A are in conflict, as well as A and B. With network coding, any stable set of the enhanced conflict graph can be served at the same time, so the throughput is improved.

Extending this example to the general case, Sundararajan *et al.* [2] have proposed the Maximum Weighted Stable Set (MWSS) algorithm. In every time-slot, the MWSS algorithm first computes a maximum weighted stable set of the enhanced conflict graph using virtual backlogs of the flow in terms of the degree of freedom as the weight. For every flow in the chosen set, the algorithm then computes a linear combination of all packets in the input buffer of the flow up to the current time, such that the linear combination is innovative to all chosen output ports of the flow. It finally transmits the computed linear combinations as the encoded packets to output ports of all subflows in the stable set. Since it is not guaranteed that output ports can decode often enough, the Finite Horizon MWSS algorithm is proposed in [2] to address this deficiency using a coding window of a smaller size, such that packets are processed in batches. When a batch of packets finishes decoding by all output ports, the entire batch is purged from the buffer and a new batch starts to be processed. A loss of throughput is incurred, but relatively small when the batch size is large.

The encoding algorithm in Haste, *opportunistic coding*, has been studied in the case of wireless networks [7]. However, no similar algorithms have been designed specifically for network switches, which are very different from wireless networks. In wireless networks, there is interference, so senders can only keep the status of the receivers with feedback. In contrast, since all input ports and output ports are within the same switch, input ports know which output ports would receive packets. Consequently, their conclusions are significantly different.

III. Haste: An Opportunistic Coding Approach

Haste employs opportunistic coding with binary operations (XOR) only. The basic idea of opportunistic coding is to always transmit a packet e such that e can be immediately decoded by all output ports received. In this section, we describe the entire design of Haste, showing how to utilize online information and XOR operations to improve a switch.

The design of Haste consists of four parts: *scheduling*, *encoding*, *decoding*, and *buffer management*. We use the concept of time-slot to describe how Haste works during four stages of one time-slot.

Scheduling: At the beginning of each time-slot t, compute a maximum weighted stable set in the enhanced conflict graph, the same as the MWSS algorithm in [2]. The weight for each subflow is the number of undecoded packets in the corresponding VOQ. We use $S(t)$ to denote the subflow set chosen at time t, and the corresponding flow set is $F(t)$.

Because the maximum weighted stable set problem is NP-hard [8], we adopt an approximation algorithm similar to what has been proposed in [9]. With respect to scheduling, Haste randomly computes a stable set at time t, compares it to the stable set $S(t-1)$, and chooses the one with a larger weight as $S(t)$. $F(t)$ can be easily obtained by $S(t)$.

Encoding: The encoding scheme runs on every flow $f \in F(t)$ at time t after $S(t)$ and $F(t)$ are computed by scheduling. Assume that $E = \{e_1, e_2, \ldots, e_n\}$ are packets stored in the VOQ for the flow f. If there exists a packet $e \in E$ such that e has not yet been decoded by any subflows $\in f \bigcap S(t)$, transfer the original packet e at t for flow f. Otherwise, randomly pick one packet e_i in the VOQ. Let $e = e_i$. Then check all packets $e_j \in E$ in a random order until e is innovative to all subflows $\in f \bigcap S(t)$. If $e \oplus e_j$ is decodable for output ports of $f \bigcap S(t)$, let $e = e \oplus e_j$. A packet e is decodable for an output port, if at most one of packets involved in encoding e has not been decoded by the output yet. Transfer the coded packet e at t for flow f.

Decoding: Whenever an output port O_i receives a packet, it decodes immediately. Assume that the packet received is $e = e_1 \oplus e_2 \oplus \ldots \oplus e_m$. According to the encoding scheme, e is decodable for O_i, namely at most one of the packets in $\{e_1, e_2, \ldots, e_m\}$ has not yet been decoded by O_i. If all of the packets have already been decoded, e is not innovative to O_i, in which case it should simply be dropped. Otherwise,

$e_i \in \{e_1, e_2, \ldots, e_m\}$ has not yet been decoded by O_i, and all other packets $\in \{e_1, e_2, \ldots, e_m\}$ have been decoded. Then e_i can be decoded, $e_i = e \oplus e_1 \oplus \ldots \oplus e_{i-1} \oplus e_{i+1} \oplus \ldots \oplus e_m$. Whenever a packet has been decoded, it is sent out of the switch immediately. However, it might still be stored in the output buffer for decoding other packets.

Buffer Management: Buffer management in Haste involves the management of both the input and the output buffers.

We manage VOQs for input ports. A packet arriving at an input port is stored at the end of corresponding VOQ according to its destinations. When a packet at the front of a VOQ is decoded by all its destinations, it is removed from the VOQ.

A key issue for input buffer management is to prevent starvation, since packets are not sent in order in Haste. Haste thus employs a FIFO batch strategy, such that only packets in the first L positions in the VOQs can be processed in the encoding algorithm, where L is the batch size. In addition, packets are only removed from the front of VOQs, so the i^{th} packet is always transferred before the $(i+L)^{th}$ packet. This strategy may lead to a loss of throughput, but largely reduces the random accessible buffering demand, resulting in a lower cost, as well as speeding up coding and scheduling. Packets in the rest of VOQs conform to a FIFO policy.

We manage virtual input queues (VIQs) at the output for decoding. Decoded packets are stored in VIQs until the original packet has been removed from the VOQs. We assume that the VIQs have the knowledge of packet removals in VOQs.

The buffer management strategy ensures that the size of a VIQ L_O can be no larger than L, the maximum number of packets that can be processed in a VOQ. We do not use alternative packet removal schemes in Haste to reduce the size of the VOQs, such as drop-when-seen [10], since these schemes are not able to maintain the size of the VIQs, which might grow in an unbounded fashion with the same batch size.

One final note about Haste is that we do not need to apply network coding on packets belonging to unicast flows. When a unicast flow is chosen in the stable set by the Haste scheduling algorithm, simply transfer the packet at the front of its VOQ.

IV. Analysis: A Comparison

Haste is superior to the conventional algorithm in two aspects: packet delays and computational complexity. We will first analyze the maximum gain in the switch achieved by using XOR operations only. We will then show the considerable gap between Haste and the conventional algorithm with respect to the computational complexity.

A. Packet delays

Compared with conventional network coding, which has to wait for receiving enough packets to decode, Haste is superior in that it eliminates the decoding delay. Therefore, the only additional packet delay Haste might experience is the transmission of redundant packets. In this section, we measure the delay by the number of packets received that cannot be decoded immediately. In the online situation, only using XOR operations can achieve zero delay in an $N \times 3$ switch.

Theorem 1: For an $M \times 3$ switch, zero delay is achievable by an online algorithm using XOR operations only.

Proof: We label the packets according to their arrival time. We consider one multicast flow of three subflows only, because other cases can be easily extended. There are three cases when sending packets at time *t*: one, two or three subflows are chosen to transmit. When only one subflow is chosen, transmit the earliest undecoded packet of that flow. When two subflows are chosen, assume p_i, p_j are the earliest arrival but undecoded packets of the two subflows, respectively. One of the three packets $p_i, p_j, p_i \oplus p_j$ can deliver innovative information to both output ports. When three subflows are chosen, assume p_i, p_j, p_k are the earliest arrival but undecoded packets of the three subflows respectively, and $i \leq j \leq k$. Because of the schedule of the other two cases, p_k has not been decoded by all three output ports. Transmit p_k in this case. In general, it is always possible to transmit innovative packets in all three cases, thus incurring zero delay. ■

We now extend the analysis with the knowledge of a full schedule S, which can clear all backlogs by conventional network coding. In this case, zero delay can be achieved in a switch of no more than 4 output ports by XOR operations only.

Theorem 2: For an $M \times 4$ switch, zero delay is achievable by an algorithm using XOR operations only, with the knowledge of a full schedule S.

Proof: A new schedule can be constructed based on S using XOR operations only, which incurs no delay. We can divide the time space into several parts, $(0, t_1), (t_1, t_2), \ldots, (t_{n-1}, t_n)$ such that no packets are partially decoded in the given schedule at time t_i. In each time interval, rearrange the schedule such that 3 or 4 subflows of one flow are always chosen before 1 or 2 subflows chosen. Then, no delay would occur using XOR operations only. ■

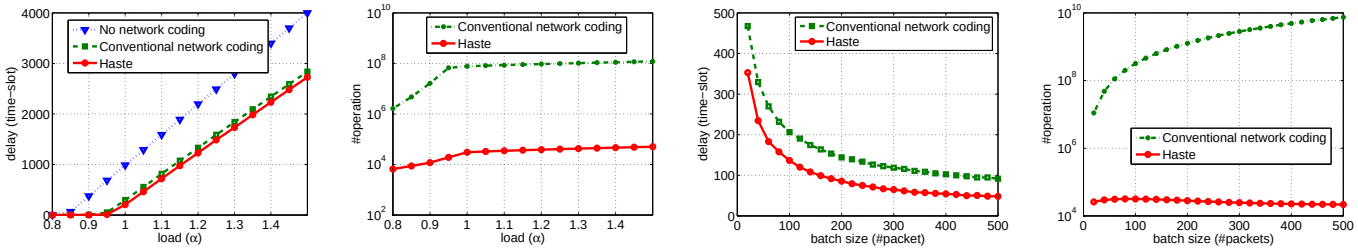
For general cases with more than 4 output ports, binary operations are not sufficient to construct a zero-delay algorithm. As a result, a loss of throughput is incurred. Therefore, there exists a throughput gap between Haste and the ideal network coding algorithm.

However, the baseline MWSS algorithm mentioned in Sec. II might result in an infinitely large decoding delay. The employment of batches also incurs a loss of throughput, as well as considerable decoding delays. In addition, both throughput and decoding delay increase with the batch size. Further comparisons of delay will be shown in Sec. V.

B. Computational complexity

Haste employs XOR operations only, so we can measure the computational complexity by the number of operations needed. The following theorem analyzes the computational complexity of decoding a residue flow by N output ports.

Theorem 3: Let $U(i)$ be the number of packets not decoded by output port O_i, E be the total number of packets in the corresponding VOQ. The upper bound of the number of XOR operations for decoding all packets is $\sum_{i=1}^{N} \frac{U(i)(2E-U(i)-1)}{2}$.



(a) packet delays with a batch size of 50 packets.
(b) computational complexity with a batch size of 50 packets.
(c) packet delays with α = 1.
(d) computational complexity with $\alpha = 1$.

Fig. 3. Delays and computational complexity in the 8×8 switch.

Proof: $U(i)$ implies the number of packets that have not been decoded by output O_i, so $E - U(i)$ is the number of packets already decoded by O_i but still required for decoding. Thus, decoding the k^{th} packet requires at most $E-U(i)+k-1$ XOR operations. There are $U(i)$ packets in total. Therefore, the operations needed by O_i is $\frac{U(i)(E-U(i)+E-1)}{2}$. The sum of XOR operations of all output ports are $\sum_{i=1}^{N} \frac{U(i)(2E-U(i)-1)}{2}$. ■

These upper bounds are actually not achievable by Haste, because Haste can prevent too many redundant packets from being involved in encoding. We have searched all possibilities of scheduling and encoding by simulation, and found the following empirical formula of the largest number of XOR operations required by the optimal encoding scheme in the worst scheduling case:

$$T_d = \begin{cases} 0 & : \quad N \leq 2 \text{ or } M \leq 2 \\ N-1 & : \quad N \geq 3 \text{ and } M = 2 \\ N(M-1) & : \quad N > M > 2 \end{cases} \tag{1}$$

T_d is the number of XOR operations used for decoding. When $N \leq M$, we can decompose $M = m_1 + m_2 + \ldots + m_k$, while $m_i < \max\{N, 2\}$ for any $i = 1, 2, \ldots, k$. We can calculate T_{d_i} by using N and m_i. $T_d = \max \sum_i T_{d_i}$.

According to the analysis, even in the worst case, decoding one packet requires fewer than one XOR operation on average. Compared with decoding, the encoding complexity is smaller, since encoding once can serve multiple output ports.

The conventional algorithm is well known for its large coding complexity. Assume that L is the number of packets in a batch, the complexity is $O(L)$ for encoding and $O(L^2)$ for decoding per packet. In addition, conventional network coding uses linear operations including additions and multiplications on a large finite field, which cost much more than XOR operations.

Compared with conventional network coding, Haste reduces the coding complexity from $O(L^2)$ to $O(1)$, largely speeding up the coding process with simple binary operations.

V. Performance Evaluation

A. Simulation

In this subsection, we conduct extensive experiments to compare Haste with no network coding and conventional network coding in both aspects of packet delays and computational complexity for coding.

The packet delay is the time from a packet arriving at the input port till when it is decoded at the output port, measured in time-slots. The computational complexity is measured by the total XOR operations for Haste and linear operations for conventional network coding. We count L linear operations when L packets are involved in encoding, and L^3 linear operations when a batch of L packets is decoded.

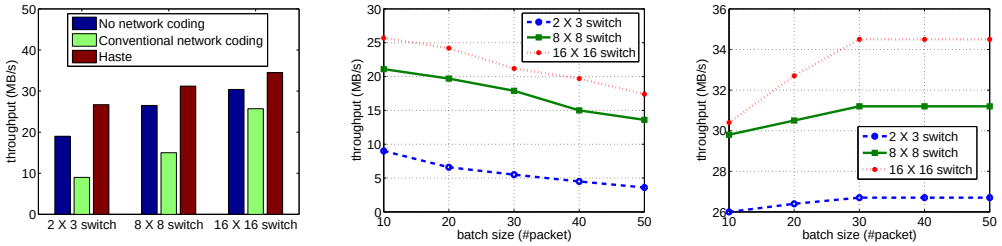
We simulate an 8×8 switch with two kinds of flows. The first one is a multicast flow from one input port to all output ports. The second one includes 7 unicast flows from the other 7 input ports respectively, to 7 different output ports. The load of all flows is 0.5α. Packets arrive at input ports according to an i.i.d. Bernoulli process for each flow independently in 10,000 time-slots. After 10,000 time-slots, the simulator finishes when all backlogs are sent out. We run each algorithm 1000 times and obtain the average results. In the algorithm with no network coding, we employ a randomized modification of [4]. In the conventional algorithm, a batch consists of a fixed number of packets, except packets in the VOQ are fewer than the number.

Fig. 3 illustrates the simulation results. Although this traffic pattern does not benefit from offline algorithms, it benefits from the online algorithm, due to the lack of the randomized online schedule. There are always performance gaps between the three algorithms. Since the delay is measured in time-slots, the delay gain is small. However, note that due to the high computational complexity, a time-slot in the conventional algorithm might be much larger than in Haste. We will show this considerable impact made by time complexity using emulation in the next subsection. In addition, the delay drops with an increase of the batch size, which implies that transmission delays make a larger impact on the overall delay.

B. Emulation

It is hard to measure throughput in the simulation because it is time-slot based, while computational complexity cannot be ignored. In this subsection, we build an emulation framework to measure the achievable throughput.

We emulate a 16×16 IQ switch in our framework, using asynchronous sockets beyond the OS kernel to process actual UDP traffic. Therefore, our implementation is highly efficient, without redundant thread dispatching or scheduling. The switch receives packets of 1 KB from 16 different ports, and then stores the packets in the input buffer. Every input port has a buffer,



(a) achievable throughput. (b) the conventional algorithm. (c) Haste.

Fig. 4. The throughput of the switch in the emulation.

TABLE I
EMULATION SCENARIOS

Flow ID	multicast	input	output	load
A 2×3 switch				
1	yes	1	1, 2, 3	α
2/3/4	no	2	1/2/3	0.5α
An 8×8 switch				
1	yes	1	$1, 2, \ldots, 8$	α
$2/3/\ldots/8$	no	$2/3/\ldots/8$	$1/2/\ldots/7$	α
$9/10/\ldots/15$	no	1	$1/2/\ldots/7$	$\frac{1}{8}\alpha$
A 16×16 switch				
1	yes	1	$1, 2, \ldots, 16$	α
$2/3/\ldots/16$	no	$2/3/\ldots/16$	$1/2/\ldots/15$	α
$17/18/\ldots/31$	no	1	$1/2/\ldots/15$	$\frac{1}{8}\alpha$

which is able to store 1024 packets. New packets are dropped immediately if the buffer is full.

We build a scheduler within the switch to emulate the switching fabric that no different packets can be sent simultaneously, nor can one output port receive multiple packets at once. The scheduler first randomly computes a schedule, compares it with the schedule last time, and chooses the one with the larger weight. The scheduler then computes a coded packet sent to corresponding output ports, which is checked immediately if decodable. We use 3 scenarios in the emulation experiments, described in Table I. We control the load factor α of the sending rate at the senders, to obtain the maximum achievable throughput. The results are shown in Fig. 4.

A considerable gap is clear in Fig. 4(a). The performance of the conventional algorithm is even worse than no network coding, due to its extremely high time complexity, which cannot be ignored in real environments. However, Haste performs well, since its computation load is lighter.

Additional emulation results are shown in Fig. 4(b) and Fig. 4(c), illustrating how the batch size makes an impact on both network coding algorithms. With an increase in the batch size, heavier computational complexity further exacerbates the performance of conventional network coding. However, similar to simulation, a larger batch size improves the throughput in Haste. It can be seen that the throughput gain can be ignored when the batch size is larger than 30. In other words, when Haste is implemented, a small batch size, say, 30, is sufficient, instead of an exceedingly large random accessible buffer size with high hardware costs.

VI. CONCLUSIONS

In this paper, we have studied the performance of a switch with network coding. Though network coding is able to improve the throughput, the conventional network coding algorithm failed in two practical aspects: large packet delays and high computational complexity. We have proposed Haste, which only uses XOR operations in an opportunistic coding fashion to streamline encoding and decoding. We have not only analyzed the maximum gain that XOR operations can achieve in a switch, but also conducted extensive experiments with both simulations and packet-based emulations. In all scenarios, Haste performs consistently well, compared with existing algorithms. We are convinced that the design of Haste constitutes one major step closer towards practical implementations of network coding in network switches.

ACKNOWLEDGMENTS

This work was supported in part by NSFC under Grant No. 60702054, Shanghai Municipal R&D Foundation under Grant No. 09511501200, and the Shanghai Rising-Star Program under Grant No. 08QA14009, NSERC Discovery Grant RGPIN 238994-06 and NSERC Strategic Grant STPGP 364910-08. Xin Wang is the corresponding author.

REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[2] J. Sundararajan, M. Médard, M. Kim, A. Eryilmaz, D. Shah, and R. Koetter, "Network coding in a multicast switch," *in Proc. of INFOCOM*, pp. 1145–1153, May 2007.

[3] M. Kim, J. K. Sundararajan, and M. Médard, "Network coding for speedup in switches," *in Proc. of IEEE International Symposium on Information Theory (ISIT)*, pp. 1086–1090, June 2007.

[4] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260–1267, August 1999.

[5] [Online]. Available: www.cisco.com

[6] M. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Multicast traffic in input-queued switches: optimal scheduling and maximum throughput," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 465–477, June 2003.

[7] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: Practical wireless network coding," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 497–510, June 2008.

[8] R. M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, pp. 85–103, 1972.

[9] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," *in Proc. of INFOCOM*, vol. 2, pp. 533–539, April 1998.

[10] J. Kumar Sundararajan, D. Shah, and M. Médard, "ARQ for network coding," *in Proc. of IEEE International Symposium on Information Theory (ISIT)*, pp. 1651–1655, July 2008.