

Compression of Multicast Labels in Large IP Routers

Marco G. Ajmone Marsan, *Fellow, IEEE*, Fabio M. Chiussi, *Member, IEEE*, Andrea Francini, *Member, IEEE*, Giulio Galante, *Student Member, IEEE*, and Emilio Leonardi, *Member, IEEE*

Abstract—In small cell-based Internet protocol routers, multicast traffic is generally handled by appending to each cell a local multicast label (LML) containing a bitmap with as many bits as switch ports, so as to identify the ports to which a copy of the cell has to be transferred. This approach is not feasible for switches having 128 ports or more, because the LML length would rise above 16 bytes, thus representing an intolerable overhead, given the small size of cells (typically 64 bytes). In this paper, we discuss both static and adaptive lossy compression algorithms to reduce the size of LMLs to be attached to multicast cells, at the price of the delivery of cells to a larger set of outputs than necessary, and we compare the compression algorithms performance in terms of switch bandwidth waste, using both analytical and simulation models.

Index Terms—Data compression, multicast traffic, packet switching, router.

I. INTRODUCTION

TODAY, we are witnessing a clear trend toward the integration of traditional communication services (telephony, radio, and TV broadcasting, etc.) in the Internet, which is becoming the candidate of choice for the implementation of a flexible packet-switched infrastructure, capable of satisfying all communication needs of the end users. This trend is producing a number of effects, the most evident and frequently quoted of which are the explosive increase of Internet Protocol (IP) traffic, the need for techniques to guarantee the quality of the services offered to end users, and the need for terabit channels and routers. Also, the integration of radio and TV broadcasting in IP networks contributes to the increasing demand for the efficient support of multicast applications and for IP routers with large number of ports.

While multicasting already received considerable attention at the algorithmic (or network) level (e.g., with the development of multicast protocols for IP networks [2]–[4]), techniques to support multicast traffic *within* IP routers have not been thoroughly investigated yet. The support of multicast traffic in IP routers is indeed a crucial problem, especially for large routers, with 128 or more input/output ports.

Manuscript received August 6, 2002; revised January 15, 2003. The work of M. G. Ajmone Marsan, G. Galante, and E. Leonardi was supported in part by a Contract between Lucent Technologies and Politecnico di Torino. This paper was presented in part at the IEEE GLOBECOM Conference, Taipei, Taiwan, November 2002 [1].

M. G. Ajmone Marsan, G. Galante, and E. Leonardi are with the Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy (e-mail: ajmone@polito.it; galante@polito.it; leonardi@polito.it).

F. M. Chiussi and A. Francini are with the Data Networking and Wireless Systems Department, Bell Laboratories, Lucent Technologies, Holmdel, NJ 07733 USA (e-mail: fabio@bell-labs.com; francini@bell-labs.com).

Digital Object Identifier 10.1109/JSAC.2003.810560

In Internet Protocol Version 4 (IPv4) [5], multicast addresses are specified on 28 bits, and, except for a few permanently-assigned values, they are dynamically bound to transient groups which exist only as long as they have connected members. In Internet Protocol Version 6 (IPv6) [6], multicast addresses are 117 bits long, and comprise a 112-bit *group identifier*, four *scope bits*, and one *flag bit*. The scope bits limit the scope of the multicast group to either the local node, the local link, the local site, the local organization, or the global Internet. The flag bit specifies if an address belongs to the set of addresses permanently assigned by the global Internet numbering authority, or is instead a dynamically-bound transient address. IP hosts use the Internet group management protocol (IGMP) [7] to notify any change of multicast-group membership status to their closest multicast routers.

The forwarding portion of an IP router typically consists of a high-speed packet switch, which most often handles fixed-size data units (or *cells*) that include a header and a payload. The routing of cells from the switch inputs to the switch outputs relies on local address labels contained in the cell headers. The switch maintains an *address translation table* (ATT) to map IP unicast addresses onto *local unicast labels* (LULs) and IP multicast addresses onto *local multicast labels* (LMLs). At the switch inputs, incoming variable-sized IP packets are segmented in chunks that fit into the cell payload, whereas the cell headers are filled with the local address labels retrieved from the ATT and other control data local to the switch.

For unicast traffic, the ATT entries are fixed and obtained from the IP routing table. The LUL simply denotes the destination port of the unicast cell. The treatment of multicast flows is quite different. The ATT entries must be added and removed dynamically, because the multicast group composition changes over time. Whenever a host joins a previously empty multicast group through IGMP, a new LML is generated for the corresponding multicast address and added to the ATT. Subsequently, the LML is updated as hosts join and leave the group using IGMP messages, and completely removed from the ATT only when the last host leaves the group.

In small switches, the LML typically contains a bitmap with as many bits as switch ports. The bitmap specifies the output ports for which the multicast cell is destined. Unfortunately, the bitmap approach is not feasible in switches with more than 128 output ports, where the LML would comprise more than 128 bits (i.e., 16 bytes), an intolerable overhead with 64-byte cells. Furthermore, since the ATT must be accessed on a packet-by-packet basis to convert IP addresses into LULs and LMLs, it must mandatorily be implemented with a fast, possibly context-associative, memory. Technological constraints limit the size of fast memories (throughout this paper, we assume a maximum of

8 Mb to be available in a single chip). In a 128-port switch, no more than 64 K bitmaps can be stored in an 8-Mb memory. The amount of storable bitmaps becomes smaller and smaller as the switch size increases, going down to 8 K in a 1024-port switch. Note that these numbers are much smaller than the $2^{28} \approx 2.7 \times 10^8$ and $2^{117} \approx 1.7 \times 10^{35}$ multicast addresses supported by IPv4 and IPv6.

In this paper, we focus on $N \times N$ switches with $N = 128, 256, 512, 1024$, and evaluate the possibility of compressing the N -bit LMLs into shorter M -bit *compressed local multicast labels* (CLMLs), using either static or adaptive *lossy* compression techniques that reduce both the cell header overhead and the per-entry occupation in the ATT.

The rest of this paper is organized as follows. In Section II, we start with an overview of available approaches to the compression of multicast bitmaps. Then, we illustrate our reference switch and traffic models, and develop a mathematical formulation of the LML compression problem. In Section III, we focus on static compression algorithms, which have some *a priori* knowledge of the LML probability distribution. In Section IV, we address adaptive compression algorithms, which acquire information on the input data distribution at run time. In Section V, we study hybrid techniques, obtained from the combination of static and adaptive compression techniques. In Section VI, we present a complex adaptive coding scheme. The proposed scheme is, perhaps, too complex to be implemented today in large high-bandwidth switches; however, it provides significant insight in the performance improvements achievable by increasing the complexity of the coding scheme. Section VII presents results for realistic input traffic distributions, which comprise both unicast and multicast flows. Finally, in Section VIII, we draw our conclusions and provide directions for future work.

II. PROBLEM STATEMENT

A. Compression Algorithms

A *bit string* of length l is an element of the set $\mathcal{B}(l) = \{0, 1\}^l$, and a *compression algorithm* is a mapping $C : \mathcal{A} \rightarrow \mathcal{A}_c$ of any arbitrary-length bit string $\mathbf{s} \in \mathcal{A}$ onto a possibly shorter bit string $\mathbf{s}_c \in \mathcal{A}_c$. Each compression algorithm is associated with a *decompression algorithm* $D : \mathcal{A}_c \rightarrow \mathcal{A}$ which undoes its action by mapping each bit string $\mathbf{s}_c \in \mathcal{A}_c$ into a generally longer bit string $\mathbf{s} \in \mathcal{A}$. The composite function $R = D \cdot C : \mathcal{A} \rightarrow \mathcal{A}$ resulting from the consecutive application of both the compression and the decompression algorithms on \mathbf{s} yields the *recovered* bit string $\mathbf{s}_r = D(C(\mathbf{s}))$.

Compression algorithms are classified depending on the characteristics of R . If C and D are such that R is the identity function, i.e., $R(\mathbf{s}) = \mathbf{s}$, $\forall \mathbf{s} \in \mathcal{A}$, it is always possible to recover an exact copy of the original bit string \mathbf{s} from its compressed version \mathbf{s}_c , and the compression algorithm is said to be *lossless*. If this is not always true, the compression algorithm is called *lossy*.

If $f_{\mathcal{A}} : \mathcal{A} \rightarrow \mathbf{R}^+$ is the probability density function of the appearance at the input of the compression algorithm of the bit strings in \mathcal{A} , and $l(\cdot)$ denotes the length of the bit string to which

the compression algorithm is applied, the average *compression ratio* of an algorithm can be defined as

$$r = \sum_{\mathbf{s} \in \mathcal{A}} f_{\mathcal{A}}(\mathbf{s}) \frac{l(\mathbf{s})}{l(C(\mathbf{s}))}. \quad (1)$$

If $d : \mathcal{A} \times \mathcal{A} \rightarrow \mathbf{R}^+$ is a distance on \mathcal{A} , the *distortion* ϵ introduced in the compressed data by a lossy algorithm (causing $\mathbf{s}_c = C(\mathbf{s})$ to be mapped onto $D(\mathbf{s}_c) \neq \mathbf{s}$ upon decompression) can be measured as either an average value

$$\bar{\epsilon} = \sum_{\mathbf{s} \in \mathcal{A}} f_{\mathcal{A}}(\mathbf{s}) d(R(\mathbf{s}), \mathbf{s}) \quad (2)$$

or as a maximum value

$$\epsilon_{\max} = \max_{\mathbf{s} \in \mathcal{A}} d(R(\mathbf{s}), \mathbf{s}). \quad (3)$$

As a rule of thumb, a clear dependence exists between the compression ratio r and the distortion ϵ experienced by the compressed data: the higher the compression ratio (i.e., the smaller the compressed LML), the higher the distortion. Also, we remark that both the compression ratio and the data distortion for a given algorithm depend strongly on \mathcal{A} , so that the same compression algorithm may achieve strikingly different performance when applied to different data sets.

Compression algorithms can be further divided into *static* and *adaptive* algorithms. Static algorithms require to know in advance the probability density function $f_{\mathcal{A}}$, whereas adaptive algorithms acquire the same knowledge at run time, from the incoming data.

An example of optimal static *lossless* compression algorithm was defined by Shannon in [8]. In Shannon's algorithm, the average compressed bit string length is equal to the entropy of \mathcal{A}

$$H(\mathcal{A}) = \sum_{\mathbf{s} \in \mathcal{A}} f_{\mathcal{A}}(\mathbf{s}) \log_2 \frac{1}{f_{\mathcal{A}}(\mathbf{s})}. \quad (4)$$

No algorithm can achieve an average compressed bit length that is lower than $H(\mathcal{A})$ in (4) [8]. However, the length of the compressed bit strings $\mathbf{s}_c = C(\mathbf{s})$ generated by Shannon's algorithm may be widely variable, since they are only constrained by the following inequality:

$$\log_2 \left(\frac{1}{f_{\mathcal{A}}(\mathbf{s})} \right) \leq l(\mathbf{s}_c) < \log_2 \left(\frac{1}{f_{\mathcal{A}}(\mathbf{s})} \right) + 1. \quad (5)$$

The variable size of the compressed bit strings constitutes an obstacle to the application of any compression algorithm to high-speed routers, where, in order to simplify circuitry, cell headers must have fixed size. In particular, it cannot be accepted that some (highly unlikely) bit strings have a compressed version longer than the cell header, or even longer than the original bit string.

Static *lossy* compression algorithms, instead, are usually fine-tuned to the data set to which they are applied by solving optimization problems that aim at the minimization of either (2) or (3).

Finally, adaptive compression algorithms (either lossy or lossless) are generally considered to be better suited to scenarios in which the data probability distribution changes broadly over time, or is not known in advance.

A good survey of compression algorithms used in the field of vector quantization and data compression can be found in [9] and [10].

B. Switch Model

We focus our attention on $N \times N$ switches where $N \in \{128, 256, 512, 1024\}$. We assume the switching fabric to operate internally on 64-byte cells that contain an 8-byte header and a 56-byte payload.

We do not address the details of the actual transfer of data packets across the switch, which include the segmentation of packets into cells at the input line cards and the scheduling algorithm that regulates access to the switch fabric.

Incoming packets are instantaneously fragmented when they arrive to the switch inputs and subjected to the ATT-lookup procedure. Indeed, switching fabrics with a large number of ports are usually implemented with multistage switching architectures, where enough information must be added to the local cell headers to route the packets to their intended destination ports. An approach based on attaching LULs and LMLs in a three-stage switching fabric is presented in [11] and further extensions are in [12] and [13].

We assume all traffic to be best effort, which implies that no call admission control (CAC) is in place to restrict the allocation of bandwidth resources.

C. Compression Problem

In this section, we address the problem of compressing N -bit LMLs into M -bit CLMLs, with $M < N$. In such a case, $\mathcal{A} = \mathcal{B}(N)$, $\mathcal{A}_c = \mathcal{B}(M)$, and $r = N/M$, regardless of the probability distribution function $f_{\mathcal{A}}$. Moreover, we assume that the value of M is always an integer power of two and can never exceed 64 bits, so that the compression ratio r is always an integer number.

We emphasize that it is impossible to compress without loss all the bit strings in $\mathcal{B}(N)$ into bit strings of length $M < N$, because the number of all bit strings of lengths $1, 2, \dots, M$ is not large enough to enumerate all the 2^N possible bit strings of length N . Therefore, if some strings are shortened, other strings must be lengthened if we want to avoid losses. Yet, this is not a big issue, because in our context 2^N is an enormous number; indeed, even in 128-port switches, if a new bit string were to be generated at every microsecond, enumerating all the 2^{128} possible bit strings would require more than 10^{15} times the age of the universe (10^{10} years). Therefore, in most practical cases, adaptive algorithms that do not consider all bit strings in $\mathcal{B}(N)$, but only the bit strings that correspond to LMLs currently in use, should lead to lower distortion than static algorithms.

A viable approach is given by static lossy compression algorithms, which associate groups of N -bit LMLs with one M -bit CLML. Since only $m = 2^M$ CLMLs are available, a lossy compression algorithm C induces a partition of \mathcal{A} into m subsets $\mathcal{A}_i = \{\mathbf{s} \in \mathcal{A} : C(\mathbf{s}) = \mathbf{s}_i, \mathbf{s}_i \in \mathcal{A}_c\}$ such that all the LMLs in \mathcal{A}_i are mapped onto the same CLML $\mathbf{s}_i, \forall i \in \mathcal{I} = \{1, 2, \dots, m\}$. Furthermore, if the decompression algorithm D implements a bijective function, the m CLMLs $\mathbf{s}_i \in \mathcal{A}_c, i \in \mathcal{I}$ are mapped onto m LMLs $\mathbf{c}_i = D(\mathbf{s}_i)$ called *codewords* (the

set $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m\} \subset \mathcal{A}$ is called the *codebook* of the composite function R).

The lossy compression algorithm C , while trying to compress LML $\mathbf{s} \in \mathcal{A}$, selects the CLML $\mathbf{s}_c \in \mathcal{A}_c$ such that the distance $d(\mathbf{s}_r, \mathbf{s})$ between the resulting decompressed LML $\mathbf{s}_r = D(\mathbf{s}_c)$ and the original LML is minimum; in case of ties, i.e., if more than one CLML satisfies the minimum-distance condition, the winning CLML is chosen randomly.

For static compression algorithms, the codebook must adapt to the LML distribution $f_{\mathcal{A}}$, so as to minimize either $\bar{\epsilon}$ (2) or ϵ_{\max} (3). The optimal codebook design is a very difficult optimization problem, because even the evaluation of the average distortion $\bar{\epsilon}$, which measures the quality of a given solution, requires the computation of 2^N sums, so that its complexity grows exponentially with the length of the LML to compress. The complexity of computing ϵ_{\max} also grows exponentially with the number of bits in the sequence to compress, because it requires to compare the codewords with every word in \mathcal{A} .

The codebook design problem has been thoroughly studied and solved in the field of information and coding theory, with the objective of designing codes for error correction and detection in the case where the distance d of (2) and (3) is the Hamming distance (the Hamming distance $d_H(\mathbf{v}, \mathbf{w})$ between two LMLs $\mathbf{v}, \mathbf{w} \in \mathcal{B}(N)$ is the number of bits where \mathbf{v} and \mathbf{w} differ, i.e., the number of bits for which $v_i \neq w_i, i \in \mathcal{P} = \{1, 2, \dots, N\}$). Unfortunately, the compression problem that we are addressing requires the adoption of a metric d_a that is not a distance, because it does not satisfy the symmetry property. Thus, we cannot rely on the results from information and coding theory.

Consider now $\mathbf{v}, \mathbf{w} \in \mathcal{A}$; \mathbf{v} is said to *cover* \mathbf{w} if $v_i = 1$ at least when $w_i = 1, i \in \mathcal{P}$. Note that with this definition it could be $\mathbf{v} \neq \mathbf{w}$, i.e., for some $i \in \mathcal{P}$ it could be $v_i = 1$ also when $w_i = 0$. Thus, if \mathbf{v} covers \mathbf{w} , but $\mathbf{v} \neq \mathbf{w}$, then \mathbf{w} does not cover \mathbf{v} . In the codes that can be considered as solutions to our problem, each codeword \mathbf{c}_i must cover all the configurations $\mathbf{w} \in \mathcal{A}_i, i \in \mathcal{I}$, to guarantee that a cell is always delivered at least to all the output ports to which it is headed in the uncompressed LML \mathbf{w} .

We can now define the asymmetrical metric $d_a : d_a(\mathbf{w}, \mathbf{v}) = d_H(\mathbf{v}, \mathbf{w})$ if \mathbf{v} covers \mathbf{w} , otherwise, $d_a(\mathbf{w}, \mathbf{v}) = \infty$. The quantity $d_a(\mathbf{v}, R(\mathbf{v}))$ expresses the number of output ports that receive a copy of a packet without actually needing it. Thus, the lossy compression algorithm that we are describing reduces the CLML length at the price of some bandwidth waste in the switch. It should also be noticed that, in order for the lossy scheme to work properly and avoid unnecessary packet flooding in the network, the output line cards must mandatorily be designed to discard all the unwanted cells that they receive.

We compare the compression algorithms in terms of the average bandwidth waste (ABW), which is a function of the load ρ offered to the switch

$$\text{ABW}(\rho) = \frac{\mathbf{E}\{d_a(\mathbf{w}, R(\mathbf{w})) | \text{load} = \rho\}}{N}. \quad (6)$$

Since the switch has only limited memory for storing the codebook, we consider techniques where the codebook is fully stored in the memory, and techniques where the codebook is generated from a smaller set of bit strings \mathcal{G} of size $|\mathcal{G}|$, which

we call *generators*. In the latter case, each codeword in the codebook is obtained either by summing (by bit-wise logical OR) up to M generators of length N , producing a codebook of size $2^{|\mathcal{G}|}$, or by tiling (by Cartesian product) k shorter generators having length N/k , producing a codebook with cardinality $|\mathcal{G}|^k$.

D. Traffic Model

In order to simplify the formulation of analytical models, we assume that the probability distribution of the N -bit LMLs is Bernoulli with parameter $p \in [0, 1]$: each bit s_i of LML \mathbf{s} , $i \in \mathcal{P} = \{1, 2, \dots, N\}$, is set at one, with probability p , to indicate that the packet is headed to output port i , or at zero, with probability $1 - p$, to indicate that the packet is not directed to output port i . This way, the average number of bits that are set at one in an LML \mathbf{s} , i.e., the average size of a multicast group is Np .

Note that the entropy of Bernoulli traffic is higher than 64 bits for a quite extended range of values of the probability parameter p . More particularly, when $N = 128, 256, 512, 1024$, the Bernoulli entropy exceeds 64 bits if, respectively: $0.11 \leq p \leq 0.89$, $0.04 \leq p \leq 0.96$, $0.02 \leq p \leq 0.98$, and $0.01 \leq p \leq 0.99$.

When presenting numerical results, since the assumption of purely multicast traffic is not realistic for today's Internet, we shall also consider traffic patterns that comprise a mixture of unicast and multicast connections. We model the switch at the session level, therefore, we only account for connections setup and tear down, and neglect the actual flow of data packets in the switch.

The call arrival process is assumed to be Poisson with rate λ arrivals/s, while the average call duration is taken to be $1/\mu$ s. Thus, the system load is $\rho = \lambda/\mu$.

III. STATIC COMPRESSION

A. A Lower Bound on ABW

The *weight* of an LML \mathbf{w} is the number of bits at one that it contains. We assume that LMLs having weight l appear with probability $f_{\mathbf{w}}(l)$ at the input of the switch, and that $n_{\mathcal{C}}(l)$ gives the *weight distribution* of codewords, i.e., the number of codewords in the codebook having weight l . The set of the LMLs covered at any distance by the codeword \mathbf{c} is called *neighbor set* of \mathbf{c} .

A codeword having weight $l + d$ covers $\binom{l+d}{l}$ LMLs having weight l , at distance d . Note that in an ideal codebook \mathcal{C} the intersections among codeword neighbor sets, i.e., the number of LMLs covered by more than one codeword should be minimized to increase the number of multicast configurations covered.

Thus, a lower bound on the average distance $\mathbf{E}\{d_a\}$ among LMLs and codewords can be obtained by using a *perfect code*, that is, a code in which different codewords cover disjoint sets of LMLs. By so doing, the average distance satisfies the following inequality:

$$\mathbf{E}\{d_a\} \geq \sum_{l=1}^N \mathbf{E}\{d_a | l\} f_{\mathbf{w}}(l) \quad (7)$$

where $\mathbf{E}\{d_a | l\}$, the average distance provided that the covered word has weight l , can be expressed as

$$\mathbf{E}\{d_a | l\} = \sum_{d=0}^{N-l} d \frac{M(l+d, l)}{\binom{N}{l}}.$$

The function $M(l+d, l)$ gives the number of LMLs having weight l covered at distance d by code words with weight $l+d$ contained in the codebook. In general, its value should be given by

$$M(l+d, l) = \binom{l+d}{l} n_{\mathcal{C}}(l+d).$$

However, since we neglect intersections among neighbor sets, it may happen that $\sum_{d=0}^{N-l} M(l+d, l)$ exceeds the total number $\binom{N}{l}$ of codewords of weight l ; thus, we force $\sum_{d=0}^{N-l} M(l+d, l)$ to be not greater than $\binom{N}{l}$. Therefore, some of the $M(\cdot, l)$ may have to be set at zero. Hence, $M(\cdot, l)$ is computed recursively, starting with $M(l, l)$ as

$$M(x, l) = \min \left\{ \binom{N}{l} - \sum_{k=l}^{x-1} M(k, l), \binom{x}{l} n_{\mathcal{C}}(x) \right\}.$$

Note that the above expression of the average distance depends on the weight distribution $n_{\mathcal{C}}$ of the codewords. As a consequence, in order to obtain a lower bound on the average distance achievable by any stateless code, the optimal distribution of codeword weights must be selected. The optimization of codeword weights is, however, difficult, thus, some simplifications are required.

Indeed, we compute the performance of a perfect code for which the function $n_{\mathcal{C}}(l)$ is optimized individually for each LML weight l , by using N different codebooks (one for each l), containing up to $|\mathcal{C}|$ codewords each. LMLs of weight l can be represented with no loss (i.e., $\mathbf{E}\{d_a | l\} = 0$) as long as their number $\binom{N}{l}$ is smaller than the codebook size $|\mathcal{C}| = 2^M$. Let l^* be the greatest l for which this holds; we conjecture that, when $l > l^*$, all the LMLs with weight l are covered at minimum average distance $\mathbf{E}\{d_a | l\} = d^*$ by using codewords with the same fixed weight $l + d^*$, where d^* is the smallest integer d such that $\binom{l+d}{l} |\mathcal{C}| \geq \binom{N}{l}$.

From the lower bound on $\mathbf{E}\{d_a\}$, using (6) we obtain the lower bound on $\text{ABW}(\rho)$.

Notice that when the LMLs are Bernoulli distributed with parameter p , $f_{\mathbf{w}}(l) = \binom{N}{l} p^l (1-p)^{N-l}$.

B. Structured Codes

Since we are mainly interested in low-complexity compression algorithms, we require that the codewords in the codebook have a particular structure; thus, we define on $\mathcal{B}(N)$ three binary operators, $*$, $+$ and $-$. Let $\mathbf{v}, \mathbf{w}, \mathbf{z} \in \mathcal{B}(N)$ be three LMLs, then

$$\mathbf{v} \begin{Bmatrix} * \\ + \\ - \end{Bmatrix} \mathbf{w} = \mathbf{z} \quad \text{such that} \quad z_i = v_i \begin{Bmatrix} \text{AND} \\ \text{OR} \\ \text{XOR} \end{Bmatrix} w_i$$

$\forall i \in \mathcal{P}$. The two LMLs $\mathbf{v}, \mathbf{w} \in \mathcal{B}(N)$ are said to be *orthogonal* if $\mathbf{v} * \mathbf{w} = \mathbf{0}$, where $\mathbf{0}$ is the null, all-zeroes, LML. A set $\mathcal{A} \subseteq \mathcal{B}(N)$ such that for each $\mathbf{v}, \mathbf{w} \in \mathcal{A}$, $\mathbf{v} + \mathbf{w}$, $\mathbf{v} - \mathbf{w}$, and $\mathbf{v} * \mathbf{w}$, are still elements of \mathcal{A} , is said to be *closed with respect to the* $+$,

–, * operators. Any set $\mathcal{A} \subseteq \mathcal{B}(N)$ can always be extended to a set $\bar{\mathcal{A}}$ called the *closure of \mathcal{A}* which is the smallest superset of \mathcal{A} closed with respect to +, –, *. A codebook that is closed with respect to the +, –, * operators is called a *structured codebook*.

Given any nonempty set of codewords $\mathcal{G} \subseteq \mathcal{B}(N)$, it is always possible to build a structured code containing \mathcal{G} by choosing $\mathcal{C} = \bar{\mathcal{G}}$ as its codebook. It is also possible to prove the following.

Proposition 1: Given any structured codebook $\mathcal{C} \subseteq \mathcal{B}(N)$, it is always possible to find a set $\mathcal{G} \subset \mathcal{C}$ of orthogonal codewords, called *generator codewords*, such that $\mathcal{C} = \bar{\mathcal{G}}$.

Proof: Let $\mathcal{P} = \{1, 2, \dots, N\}$ be the set of all the possible positions that a bit can occupy in a N -bit codeword. Let $\mathcal{C}_i \subseteq \mathcal{P}$ be the set of the positions of all the bits in codeword \mathbf{c}_i whose value is one. Let \mathcal{F} be the family of sets \mathcal{C}_i associated with the codewords in codebook \mathcal{C} . In this way, there is an isomorphism between codebooks and finite families of subsets of \mathcal{P} .

It is easy to show that the family \mathcal{F} associated with any structured codebook is closed with respect to the set union, the set intersection, and the set difference operations. Then, by standard set theory, it follows that there is a subfamily of disjoint sets in \mathcal{F} whose closure is \mathcal{F} (see [14]), and such subfamily is isomorphic to the orthogonal generator codeword set \mathcal{G} . ■

The properties of structured codes lead to a great simplification of the compression and decompression algorithms.

Compression is performed by evaluating the product (*) between the incoming LML $\mathbf{w} \in \mathcal{B}(N)$ and each of the M generator codewords, and setting the corresponding bit in the CLML either at zero, if they are orthogonal, or at one, if they are not. Note that in this way we are sure to get the nearest covering codeword as all generator codewords are orthogonal and each bit in an LML is covered by exactly one generator codeword.

Decompression recovers the LML corresponding to a given CLML by simply adding (+) the generator codewords corresponding to the bits set at one in the M -bit packet tag.

In this way, the generated codebook \mathcal{C} contains $2^{|\mathcal{G}|} = 2^M$ codewords, but only $M \times N$ switch memory bits are needed for storing \mathcal{C} instead of the $2^M \times N$ that would be required for containing \mathcal{C} . Moreover, the compression function can be implemented with M products (*), and the decoder function with at most M sums (+).

C. A Simple Structured Code

In this section, we design a simple structured code, assess its performance for different values of the compression ratio r and compare it with the lower bound of (7) when the incoming LML probability distribution is Bernoulli with parameter p .

We consider a structured code generated by M generator codewords, each containing $r = N/M$ bits at one. Since under Bernoulli traffic each switch output port has the same probability p of being selected, and there is no correlation among output ports, in the computation of code performance, the position occupied by bits at one in generator codewords is completely irrelevant. One of the simplest possible choices consists in having a set of M orthogonal generator codewords, where the i th generator codeword $i \in \{1, 2, \dots, M\}$ contains bits at one only in the i th group of r contiguous positions $\{(i-1)r + 1, \dots, ir\}$.

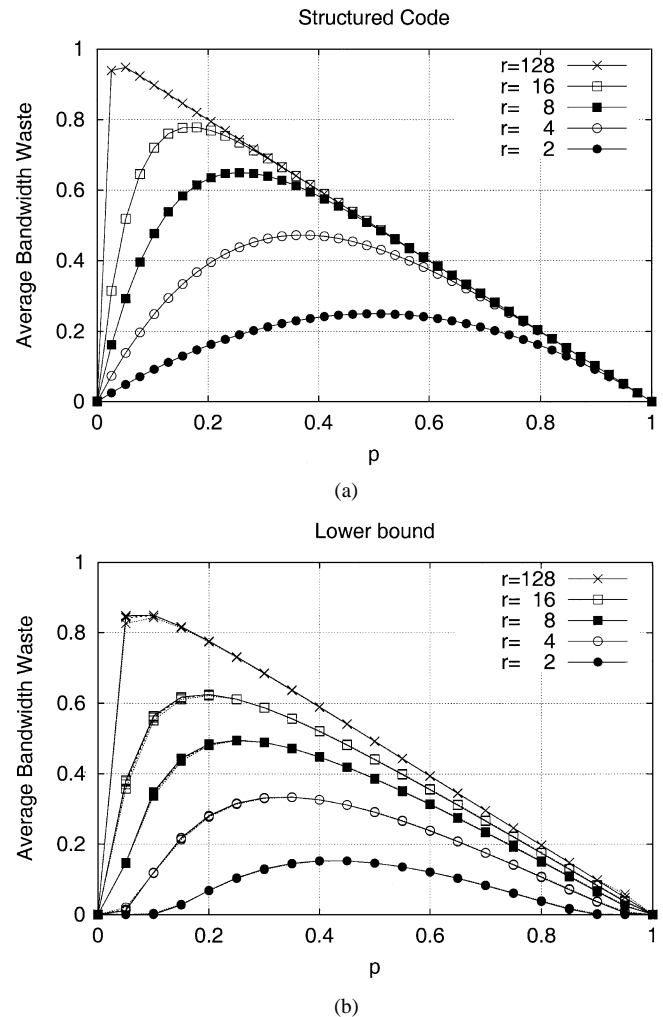


Fig. 1. Average bandwidth waste for static codes with generator codewords of length r versus the Bernoulli probability p . (a) Structured code. (b) Lower bound.

D. Performance Evaluation

The average bandwidth waste for this structured code can be easily computed. Consider the first r bits of an LML; the probability that exactly i of them are set at one is $\binom{r}{i} p^i (1-p)^{r-i}$. The all-zero (i.e., $i = 0$) configuration is covered at distance zero by setting at zero the first CLML bit, while the configurations containing $i = 1, 2, \dots, r$ bits at one are covered at distance $r - i$ by the first generator codeword, setting at one the first CLML bit. Thus, the average asymmetrical distance when considering only the first r LML bits is $r[(1-p) - (1-p)^r]$. Owing to the Bernoulli traffic properties, the same holds also for each of the M groups of r adjacent LML bits on which the generator codewords are nonnull. Therefore, the average asymmetrical distance for the whole LML is M times the average distance restricted to the first r LML bits and the average bandwidth waste does not depend on the switch offered load

$$ABW(\rho) = ABW = (1-p) - (1-p)^r. \quad (8)$$

In Fig. 1, we report the average bandwidth waste versus the Bernoulli probability p for both the structured code [Fig. 1(a)] and the lower bound [Fig. 1(b)]: several curves are plotted, for

TABLE I
QUASI-STRUCTURED CODES: MAXIMUM VALUES OF K ,
FOR DIFFERENT VALUES OF M_0 AND M

M_0	M	K
64	32	8
128	32	6
256	32	5
512	32	5
1024	32	4
128	64	16
256	64	12
512	64	10
1024	64	8

different values of the compression ratio $r = N/M$. While the structured code ABW in (8) depends only on r , not on the switch size, it is difficult to see from the mathematical formulation whether the lower bound exhibits some form of dependence on the switch size N . Hence, on the lower plot of Fig. 1, we plot four different lower bound curves (for $N = 128, 256, 512, 1024$) for each value of r , in order to assess how well they match; the four lower bound curves are practically indistinguishable.

It can be observed that the distance between the actual ABW and its lower bound is not very large.

E. Quasi-Structured Codes

Since it can be expected that lightweight LMLs are the most likely and the most critical from the point of view of performance, in this section, we present a methodology aimed at the design of codes that better cover the subspace of lightweight LMLs. For this purpose, we can prune a structured code of all the heavyweight codewords, except for the broadcast codeword. In this way, we increase the number of generators, and decrease their weights, thus augmenting the code resolution for lightweight LMLs, and obtaining a codebook comprising all the codewords of a finer structured code whose weights do not exceed a given threshold, plus the broadcast codeword.

More precisely, we consider a structured code generated by M_0 (with $M_0 > M$) generator codewords, each containing $r_0 = N/M_0$ bits at one, and we eliminate all the codewords whose weight exceeds the value Kr_0 , (where K is an integer constant), except for the broadcast codeword of weight N . Since the size of the codebook cannot exceed 2^M , K must satisfy the following relation:

$$\sum_{k=0}^K \binom{M_0}{k} + 1 < 2^M.$$

In Table I, we report the maximum values of K for different M and M_0 .

F. Performance Evaluation

Figs. 2 and 3 report the ABW for the code we just described, versus p , for $M = 64$, respectively, for switch size $N = 1024$ and $N = 256$. In both figures, different curves refer to different values of $\alpha = M_0/M$. Note that, as expected, the code performance for very small values of p improves as α increases; for larger values of p , instead, when the probability of heavyweight LMLs increases, the structured code (associated with $\alpha = 1$) largely outperforms all quasi-structured codes.

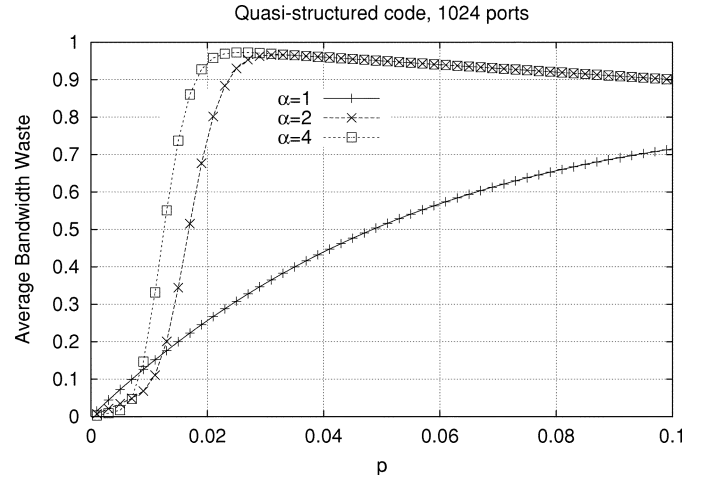


Fig. 2. Average bandwidth waste for quasi-structured codes with $\alpha = 1, 2, 4$, versus the Bernoulli probability p for 1024-port switches.

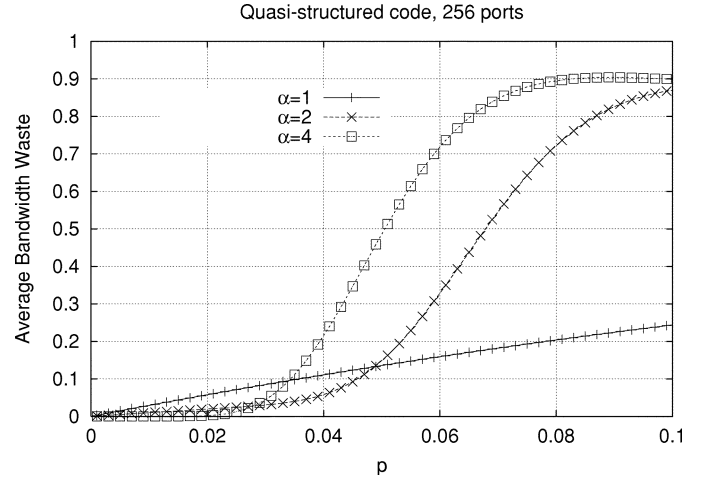


Fig. 3. Average bandwidth waste for quasi-structured codes with $\alpha = 1, 2, 4$, versus the Bernoulli probability p for 256-port switches.

The numerical results in Fig. 2 and 3 were obtained with the simple analytical model described below.

Let $P_b = (1-p)^{r_0}$ denote the probability that an incoming LML contains r_0 zeros in r_0 assigned positions. The probability that the incoming LML is covered by a codeword with weight kr_0 is

$$\binom{M_0}{k} P_b^{M_0-k} (1-P_b)^k.$$

Thus, the average weight of the codeword covering the incoming LML is

$$\begin{aligned} \mathbf{E}\{w\} &= \sum_{k=0}^K kr_0 \binom{M_0}{k} P_b^{M_0-k} (1-P_b)^k \\ &\quad + N \left(1 - \sum_{i=0}^K \binom{M_0}{i} P_b^{M_0-i} (1-P_b)^i \right) \end{aligned}$$

and the resulting ABW is

$$\text{ABW} = \frac{\mathbf{E}\{w\} - Np}{N}$$

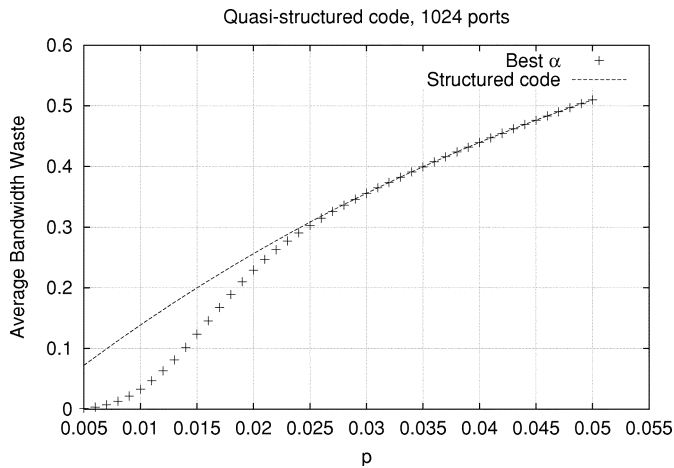


Fig. 4. Average bandwidth waste for structured code, and for five quasi-structured codes used in conjunction, versus Bernoulli probability p for $N = 1024$, $M = 67$.

The performance of quasi-structured codes can be further improved (specially for small values of p) by applying several different quasi-structured codes in conjunction. In Fig. 4, for a 1024-port switch, we report a performance comparison between a structured code with $M = 64$, and a scheme in which five different quasi-structured codes with $M = 64$ and $\alpha = 1, 2, 4, 8, 16$ are used in conjunction. For every incoming LML, the resulting bandwidth waste is computed by applying each of the five quasi-structured codes, and the code that minimizes the waste is chosen. Of course, in this case the length of the CLML must be increased from $M = 64$ to $M = 67$, since three bits are necessary to identify the code that has been used to represent each LML. The results in Fig. 4 are obtained by simulation,¹ and show that a significant improvement in performance can be obtained by applying several quasi-structured codes in conjunction for values of $p \leq 0.02$, while for values of $p > 0.025$ the performance becomes almost identical to that of the structured code.

Finally, we can observe that the complexity in implementing quasi-structured codes is mainly related to the difficulty in obtaining an easily implementable bijective function that maps the LML set into the space of the 2^M M -bit CLMLs.

IV. ADAPTIVE COMPRESSION

We now consider adaptive codes, i.e., codes that are dynamically generated by storing the LMLs of active multicast flows (or their covering codewords) in a buffer of finite size until space is available. By so doing, the codebook contained in the buffer is dynamically adapted to the set of active multicast connections.

A. Codebook Management Algorithm

We assume that technological constraints limit to 8 Mb the maximum size of the buffer, so that only a limited number $b = 2^{23}/N$ of codewords, ranging from 65 536 to 8192 as N increases from 128 to 1024, can be stored in the switch memory.

¹All the simulation results reported in this paper were obtained with a 1% accuracy in a 95% confidence level.

The buffer is filled on-the-fly, when multicast connections are activated. Thus, the adaptive code can be fine tuned to both stationary and nonstationary input data distributions, provided they can be adequately represented by b codewords.

The codebook \mathcal{C} stored in the buffer always contains in c_1 the broadcast (i.e., the all-one) codeword, to guarantee that any multicast configuration can be covered.

In this framework, a CLML is simply the binary representation of the position of the covering codeword inside the codebook, and its length ranges from 13 to 16 bits as the codebook size varies from 8192 to 65 536 entries.

Unfortunately, in the most favorable case, we use only one quarter of the 64 bits available in the packet header, thus multiplying by four the compression ratio, and consequently increasing the ABW. Therefore, in order to achieve a better performance, we adopt a Cartesian product coding scheme where incoming LMLs are split in k parts, each containing N/k bits,² so that the buffer can contain up to kb LML segments. Covering LMLs are then obtained by tiling groups of k LML segments which are, therefore, used as generator codewords, and CLMLs are obtained by concatenating the binary representations of the indexes of the k covering segments. Note that, also in this case, the buffer must always contain the all-one generator codeword to guarantee that any LML is covered.

The advantages of this strategy are twofold: first, the size of the generated codebook becomes $(kb)^k$, second the utilization of the 64-bit packet header increases from $\lceil \log_2(b) \rceil$ to $k \lceil \log_2(kb) \rceil$. However, since the latter quantity cannot be greater than 64 bits, there is a tradeoff between k and the number of generator codewords that can be stored in the buffer. If k becomes too big, we are obliged to store in the buffer a number $n < kb$ of LML segments, so that $k \lceil \log_2(n) \rceil$ does not exceed 64 bits. Therefore, in the sequel, in order to achieve a full buffer utilization, we will use $k = 2$ for both 128- and 256-port switches, and $k = 4$ for both 512- and 1024-port switches.

The switch, during its operations, extracts from each incoming multicast connection setup request the N -bit LML $\mathbf{w} \in \mathcal{B}(N)$ and runs the codebook allocation algorithm to process setup requests; when the multicast connection is terminated, the switch codebook deallocation algorithm is run, to manage tear-down requests.

The codebook allocation algorithm is very simple: for each incoming packet, the LML $\mathbf{w} \in \mathcal{B}(N)$ is split into k segments $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k$. Then, for each $j \in \{1, \dots, k\}$, if the buffer is not full, segment \mathbf{w}_j is inserted into the first buffer empty position i_j , else the index i_j of the nearest covering segment in the buffer is selected and a counter collecting the number of LMLs parts covered by that segment is incremented. The k resulting indexes i_j are packed into a $k \lceil \log_2(kb) \rceil$ bits CLML, and attached to the packet, which is queued at the input of the switch.

Connection tear downs are managed by splitting the LML into the k segments, decreasing the counters associated with the

²Clearly, since we want N/k to be an integer quantity, k can only be an integer power of two.

corresponding covering codewords, and freeing the entries for which the counter goes to zero.

B. Analytical Model

A new multicast connection setup request that finds k free positions in the codebook is always represented with no loss by allocating k empty codewords, and since one entry is occupied by the broadcast configuration, up to $\lfloor (kb-1)/k \rfloor = b-1$ LMLs can be represented exactly. Instead, a new multicast connection set-up request that finds the codebook full has to be mapped on the k nearest allocated covering codewords.

If we denote with “cf” and “cnf,” respectively, the events *codebook full* and *codebook not full*, the average asymmetrical distance per LML segment, provided that the load offered to the system is ρ , and the Bernoulli parameter is p , can be expressed as

$$\mathbf{E}\{d_a|\rho, p\} = \mathbf{P}\{\text{cnf}|\rho, p\}\mathbf{E}\{d_a|\text{cnf}, \rho, p\} + \mathbf{P}\{\text{cf}|\rho, p\}\mathbf{E}\{d_a|\text{cf}, \rho, p\}. \quad (9)$$

For simplicity, let $\tilde{b} = b - 1$ and $\tilde{N} = N/k$. For what was said above, $\mathbf{E}\{d_a|\text{cnf}, \rho, p\} = 0$. The expected waste on one segment when the codebook is full can be computed as

$$\mathbf{E}\{d_a|\text{cnf}, \rho, p\} = \sum_{l=0}^{\tilde{N}} \mathbf{P}\{w = l|\rho, p\}\mathbf{E}\{d_a|w = l, \rho, p\}.$$

The probability that an LML extracted from a Bernoulli distribution with parameter p has weight l does not depend on the offered load ρ and is

$$\mathbf{P}\{w = l|\rho, p\} = \binom{\tilde{N}}{l} p^l (1-p)^{\tilde{N}-l}.$$

The waste in representing an LML with weight l is independent from the offered load ρ , but depends on whether it is possible to find in the codebook a covering codeword different from the broadcast configuration \mathbf{c}_1 ; the two events are denoted by “f” (a codeword different from \mathbf{c}_1 is found), and “nf” (a codeword different from \mathbf{c}_1 is not found).

If the codewords in the codebook are assumed to be extracted from a Bernoulli distribution with parameter p , the average waste is given by

$$\mathbf{E}\{d_a|w = l, \rho, p\} = \mathbf{P}\{f|w = l, p\}\mathbf{E}\{d_a|f, l, p\} + \mathbf{P}\{\text{nf}|w = l, p\}\mathbf{E}\{d_a|\text{nf}, l, p\}.$$

The waste when it is impossible to find a covering codeword besides \mathbf{c}_1 is always the difference between the weight of the broadcast configuration and the weight of the LML

$$\mathbf{E}\{d_a|\text{nf}, l, p\} = \tilde{N} - l.$$

The average waste provided that the incoming LML having weight l is covered by a codeword in the codebook is

$$\mathbf{E}\{d_a|f, l, p\} = \sum_{k=0}^{\tilde{N}-l} k \mathbf{P}\{d_a = k|f, l, p\}.$$

Note that, since in the previous expression we assume the probability of covering an incoming LML with a nonbroadcast word

to be small, we neglect the probability that multiple nonbroadcast codewords in the table cover an incoming LML. We will present later in a section an extension of the model that takes into account the case in which multiple nonbroadcast codewords in the table may cover an incoming LML.

Since the LML values are Bernoulli distributed with parameter p

$$\mathbf{P}\{d_a = k|f, l, p\} = \binom{\tilde{N}-l}{k} p^k (1-p)^{\tilde{N}-l-k}$$

then

$$\mathbf{E}\{d_a|f, l, p\} = (\tilde{N} - l)p.$$

The probability of finding a codeword covering an incoming LML of weight l is p^l , while the probability of not finding it is $1 - p^l$. Therefore, the probability of not finding a covering codeword in the whole codebook is

$$\mathbf{P}\{\text{nf}|w = l, p\} = (1 - p^l)^{kb-1}$$

and the probability of finding at least one covering codeword in the codebook is

$$\mathbf{P}\{f|w = l, p\} = 1 - (1 - p^l)^{kb-1}.$$

Since the buffer can contain up to $b - 1$ LMLs, it can be approximately described with an M/M/ \tilde{b} /0 queue. Thus, the Erlang B function approximates the probability $\mathbf{P}\{\text{cf}|\rho, p\}$ of a setup request finding all the \tilde{b} LML entries full, when the load offered to the system is ρ

$$\mathbf{P}\{\text{cf}|\rho, p\} = \frac{\frac{\rho^{\tilde{b}}}{\tilde{b}!}}{\sum_{i=0}^{\tilde{b}} \frac{\rho^i}{i!}} \approx \left(1 - \frac{b}{\rho}\right) u(\rho - b)$$

where the function $u(x)$ is zero when $x < 0$ and one otherwise. It should be noted that this result is exactly true only when no more than one connection is associated with a codeword belonging to the set $\{\mathbf{c}_2, \mathbf{c}_3, \dots, \mathbf{c}_{kb}\}$, or when all of the overflowing requests are mapped on the broadcast codeword \mathbf{c}_1 .

When the probability that an incoming LML segment finds the codebook full and is covered by a nonbroadcast codeword becomes nonnegligible, more LML segments could be mapped at once on the same buffer entry, slowing down the codewords' release frequency. In this situation, the probability of finding the codebook full increases and the codebook behavior can no longer be exactly described by an M/M/ \tilde{b} /0 queue. Nevertheless, the Erlang B formula can still provide a good approximation just substituting the switch offered load ρ with $\rho' = \rho(1 + \mathbf{P}\{f|p\})$.

This trick allows us to account for the larger codeword renewal time by artificially augmenting the load offered to the system according to the average probability of finding a covering codeword

$$\mathbf{P}\{f|p\} = \sum_{l=0}^{\tilde{N}} \mathbf{P}\{f|w = l, p\}\mathbf{P}\{w = l|p\}$$

The average asymmetrical distance on a whole LML can be easily obtained by multiplying by k its value restricted to a generic LML segment given in (9), while the average bandwidth waste ABW can be computed applying (6).

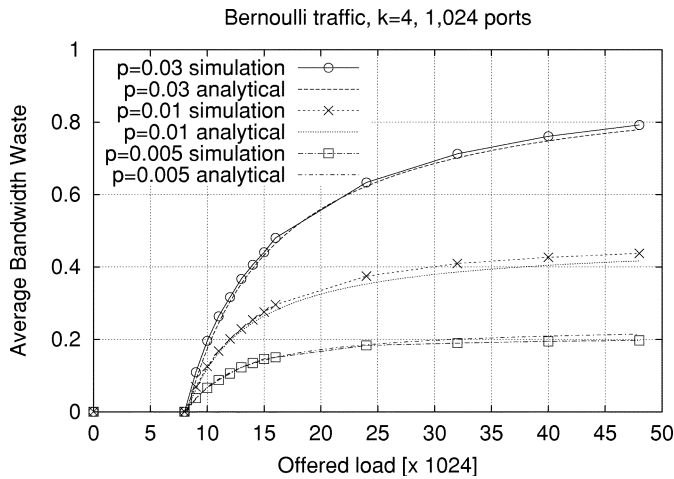


Fig. 5. Average bandwidth waste for the adaptive compression scheme, with $N = 1024$ and $M = 64$, computed both by simulation and by the analytical model for $p = 0.005, 0.01, 0.03$ versus the switch load.

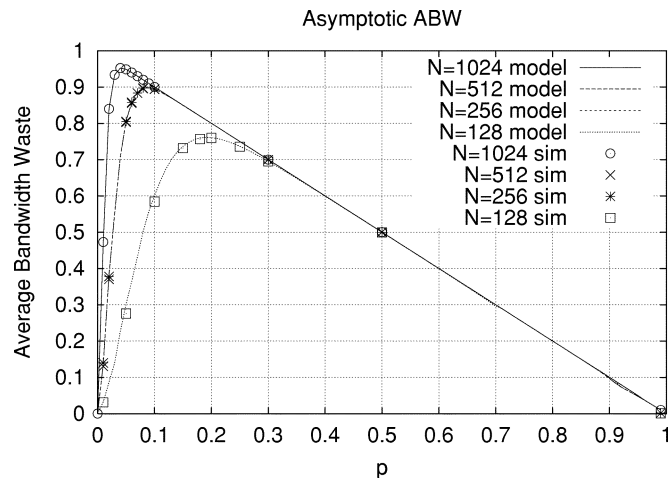


Fig. 6. Asymptotic ABW for the adaptive compression scheme, computed both by simulation and by the model for $N = 128, 256, 512, 1024$, and $M = 64$ versus the Bernoulli probability p .

C. Performance Evaluation

The curves of the ABW versus the offered load resulting from the simulations are well matched by the curves obtained with the analytical model, as shown in Fig. 5 for a 1024×1024 switch when $p = 0.005, 0.01, 0.03$. The main advantage of this approach is that the first \tilde{b} LMLs entering into the switch are served with no waste.

The average waste $\mathbf{E}\{d_a|cf, p\}$ representing the asymptotic ABW when the offered load ρ tends to ∞ is calculated both by simulation and by the model, and plotted versus the Bernoulli probability p in Fig. 6 for $N = 128, 256, 512, 1024$ when $M = 64$.

The curves for $N = 512$ and $N = 256$ are superimposed because both the LML segment size \tilde{N} (128 bits) and the codebook size $kb - 1$ (65 536 generators) are the same. The 128-port switch, instead, achieves better results since it uses a larger set of generators (131 072) and a smaller LML segment size (64 bits). Finally, the 1024-port switch experiences the worst performance because it has the smallest codebook (only 32 768 generators) and the highest LML segment size (256 bits).

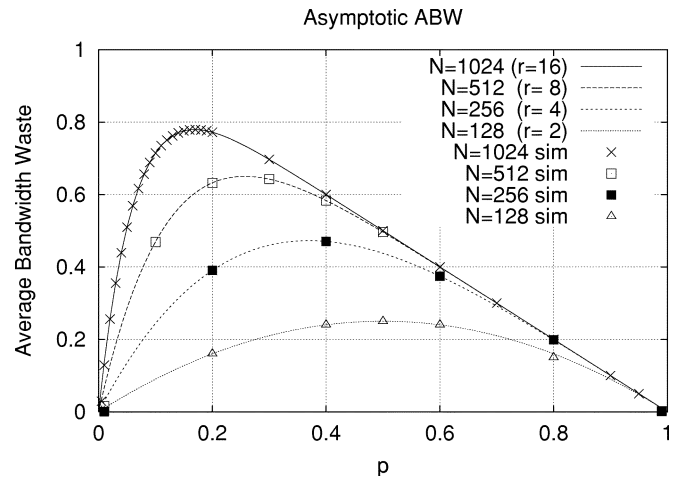


Fig. 7. Asymptotic ABW for HA, computed both by simulation and by the model, for $N = 128, 256, 512, 1024$, and $M = 65$ versus the Bernoulli probability p .

It is interesting to note that the static code presented in Section III can be interpreted as a Cartesian product code with an LML segment size of r and a codebook containing only the empty (i.e., all-zero) and the broadcast (i.e., all-one) generators, so that only 1 bit is needed to indicate the one that is chosen. The static codes corresponding to a switch size N equal to 128, 256, 512, 1024 ports and a CLML size M equal to 64 bits have, respectively, a compression ratio r equal to 2, 4, 8, 16. From a quick comparison between the curves in Fig. 1 and those in Fig. 6 it is clear that the better performance of adaptive codes when the offered load is low, is traded for a worse ABW asymptotic value.

V. HYBRID APPROACHES

In order to improve the compression algorithm performance, hybrid static and adaptive approaches can be applied. In this way, the best characteristics of both static and adaptive schemes can be exploited.

Conceptually, the first scheme that we present in this section, called the hybrid algorithm (HA), is quite simple: when a new LML \mathbf{w} arrives at the switch, both the static and the adaptive compression algorithms are applied, and the one achieving the lowest ABW is selected for representing \mathbf{w} . Of course, one extra bit must be added to the CLML to identify which compression scheme is used.

Fig. 7 shows that the asymptotic ABW achieved by the HA when $N = 128, 256, 512, 1024$ and $M = 65$ is almost equal to the ABW for the static code (compare Figs. 1 and 7). This means that when the buffer is full, HA almost invariably uses the static code (as we observed by simulation); thus, we can simplify HA as follows: if the buffer is empty put the incoming LML in the buffer (i.e., perform adaptive compression), else, if the buffer is full, compress the LML immediately with the static code, instead of performing a costly and almost surely unsuccessful search in the codebook for the nearest covering codeword.

A further performance improvement may be obtained observing that LMLs with light weight can be exactly represented

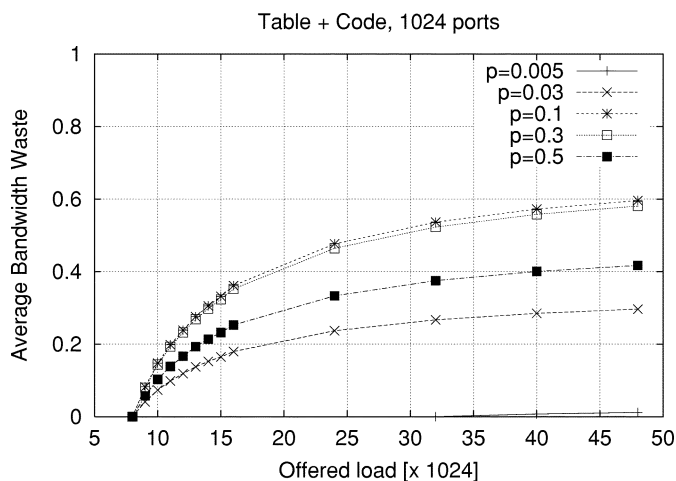


Fig. 8. Average bandwidth waste for IHA, with $N = 1024$ and $M = 66$ for $p = 0.005, 0.03, 0.1, 0.3, 0.5$ versus the switch load.

with a positional coding algorithm, i.e., by concatenating the addresses of all the output ports to which the multicast data are directed.

For example, in a 1024-port switch, if the CLMLs are 64 bits long, LMLs with weight not greater than six can be exactly represented, since 10 bits are sufficient to represent any output port address. Furthermore, when the switch has 128, 256, or 512 ports, output port addresses can be represented, respectively, on 7, 8, or 9 bits; thus, LMLs with weight up to either 9, 8, or 7 can be compressed without loss. Note that this coding scheme falls in the class of quasi-structured codes with $M_0 = N$, where a simple positional coding bijective function has been used to map codewords into CLML. In this case, however, since the image of the positional coding mapping function does not cover all the space of 2^M CLMLs, a reduction in the size of the usable codebook is experienced with respect to the value $K = 8$ reported in Table I.

We, therefore, devise a new *improved hybrid algorithm* (IHA) that works as follows: if the incoming LML has weight such that it can be compressed with no loss, apply positional coding, else revert to the above simplified HA. Note that, in this case two bits must be added to the CLML to specify whether the positional, the static, or the adaptive compression algorithm is used.

The ABW for the IHA is reported in Fig. 8 for a 1024×1024 switch. As expected, this scheme outperforms both static and adaptive compression algorithms. The ABW for large load resembles that obtained with static structured codes (see Fig. 7), since the effect of the buffer becomes negligible; however, for very small values of p (i.e., for $p = 0.005$), a performance improvement with respect to structured codes is observed, due to the effect of the exact positional coding. For medium loads, the effect of the buffer becomes more significant, leading to a decrease of the ABW value. For light loads (i.e., for offered load lower than 8192), the effect of the adaptive code buffer becomes dominant, and the ABW tends to vanish.

VI. A MORE COMPLEX SCHEME

In this section, we present a rather complex adaptive coding scheme, called ultra adaptive scheme (UAS), and we assess its

performance by simulation. The considered scheme is probably too complex to be implemented today in large high-bandwidth switches; however, it provides significant insight in the performance improvements achievable by increasing the complexity of the coding scheme.

Unlike for previously described coding schemes, when a new LML arrives, we allow the modification of some existing buffer entries, increasing their weights so as to more efficiently cover the newly arrived LML. More precisely, when a new LML arrives at the switch, if there is enough room in the buffer, the LML is partitioned in k segments, and each segment is stored in the buffer, like in the case of previously described adaptive codes. If the buffer is full, for each segment associated to the new LML, a search through all the entries stored in the buffer is performed; for each entry, the waste that would result by modifying it in order to cover the new LML is computed. The entry that minimizes the waste is chosen to represent the segment of the new LML, and is updated, if necessary. Note that when an existing entry in the buffer is modified, by adding ones in appropriate positions so as to cover the segment associated to a new LML, an increase in the bandwidth waste for all the LMLs that are using the considered entry is experienced. Thus, we need to take into account two contributions when we evaluate the average bandwidth waste that would result by using this entry for the new LML:

- the bandwidth waste associated with the new LML segment; this contribution is given by the asymmetric distance between the buffer entry and the segment;
- the increase in the bandwidth waste for all the LMLs that are currently represented by the considered entry, due to the increase of the entry weight; this contribution can be computed as the product between the increase of the entry weight and the number of currently active connections using the considered entry.

Regarding the algorithm complexity, we further notice that a counter must be associated with each bit of all entries in the buffer to efficiently manage the system dynamics. Indeed, for each entry in the buffer, it is necessary to keep trace of the number of active connections that require each bit to be set to one. When a connection ends, all the counters associated to bits at one that were requested by the expired connection, are decreased, and bits whose corresponding counter goes to zero are turned to zero. The buffer entry is instead freed when the number of associated connections goes to zero.

Fig. 9 reports the ABW for the UAS, versus the switch load ρ , for several values of probability p in a 1024×1024 switch, when $M = 64$ and $k = 4$. Comparing these results for the UAS against those reported in Fig. 5 for conventional adaptive schemes, or against those reported in Fig. 8 for the IHA, we can observe that UAS significantly outperforms the other two schemes for moderate values of ρ , specially for small values of p . For large values of ρ , the advantages of applying UAS tend instead to vanish. Results not presented here for the sake of brevity show that, asymptotically, when the load becomes very high, IHA outperforms UAS due to the presence of the structured code. In addition, results show that, asymptotically, the lifetime of each bit set to one in the buffer tends to infinity when UAS is applied; thus, for very high loads, the adaptivity of

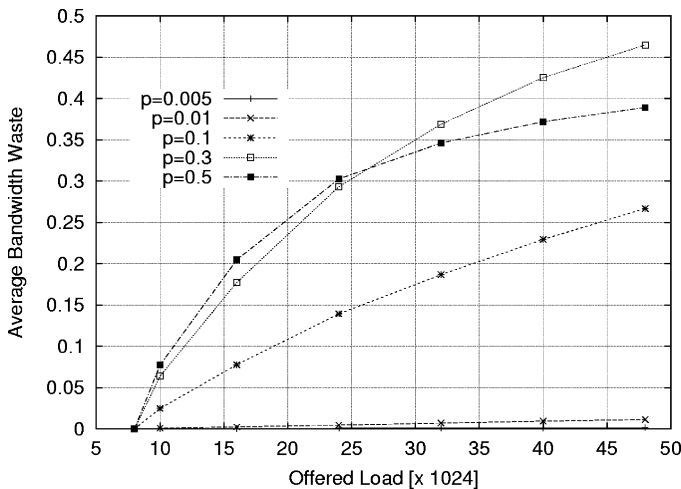


Fig. 9. Average bandwidth waste for the UAS with $N = 1024$ and $M = 64$ for $p = 0.005, 0.01, 0.1, 0.3, 0.5$ versus the switch load.

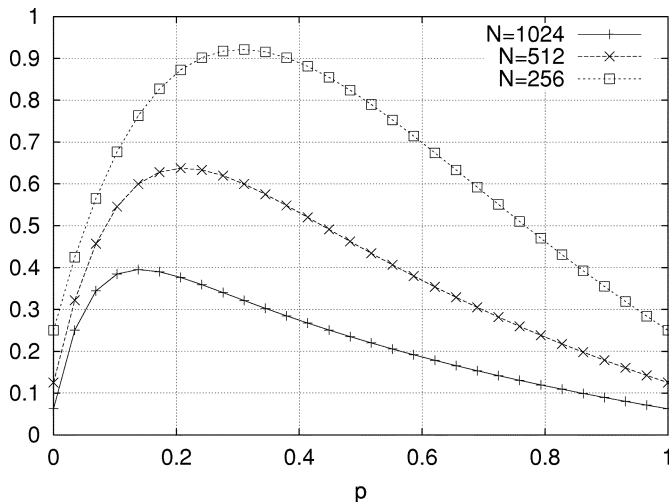


Fig. 10. Performance comparison between different schemes.

the scheme tends to vanish. However, for a large range of values of ρ , the adaptivity of UAS is significantly rewarded in terms of performance.

VII. REALISTIC TRAFFIC PATTERNS

In this section, we present results for traffic patterns that comprise only a fraction of multicast flows. In particular, we assume that the traffic mix is such that 90% of all traffic is unicast, 5% is multicast with Bernoulli distribution with parameter p , and 5% is broadcast. The performance of the IHA, in our opinion provides the best tradeoff between performance and complexity, is compared with that of the trivial scheme in which the LML is not compressed at all, and the raw N -bit LML bitmap is attached to each cell.

Curves in Fig. 10 show the ratio between the ABW produced by the two schemes at high load, for different switch sizes, versus probability p .

Note that the ABW for the hybrid scheme is due to two different contributions: 1) a CLML of 8 bytes plus 2 bits must be attached to each cell and 2) cell copies are transmitted toward

extra unwanted outputs, due to the lossy compression. In the trivial scheme, instead, the bandwidth waste is caused by the N -bit label attached to each cell.

Fig. 10 shows that our approach leads to significant advantages with respect to the trivial scheme, especially for large switches. The advantages are quite significant for either large or small values of p , but smaller for medium values of p . It can be however claimed that most multicast applications refer to a set of destinations that is either rather small, or close to broadcast; thus, both small and large values of p have great significance. Note, finally, that while the performance of the hybrid scheme depends on the load, the performance of the trivial approach does not; thus, for moderate load values the gain achieved by our algorithm increases.

VIII. CONCLUSION

We have discussed techniques to support the transfer of multicast, as well as unicast traffic, within cell-based packet switches with a large number of input/output ports. In particular, we have devised approaches for the compression of the local multicast labels that in small switches are normally prepended to each cell to indicate the ports to which a copy of the cell must be transferred. The proposed lossy compression approaches can either be static or dynamic, in the sense that they may either consider or ignore the multicast flows that at a specific instant are traversing the switch. In addition, it is also possible to devise hybrid schemes trying to combine the advantages of static and dynamic techniques.

Analytical and simulation models were used to investigate the performance of the different compression approaches, showing that hybrid schemes can provide very good performance in some cases, and also that even the performance of static compression algorithms can often be satisfactory.

Future work will address the problem of adapting our compression algorithms to a DiffServ scenario where several classes of traffic with different quality-of-service guarantees exists, and the capacity of both the switching fabric and the input/output ports becomes relevant.

REFERENCES

- [1] M. Ajmone Marsan, F. M. Chiussi, A. Francini, G. Galante, and E. Leonardi, "Efficient multicast support in large IP routers," in *Proc. IEEE GLOBECOM 2002*, Taipei, Taiwan, R.O.C., Nov. 17–21, 2002.
- [2] D. Waitzman, S. Deering, and C. Partridge, Distance Vector Multicast Routing Protocol, RFC 1075, Nov. 1988.
- [3] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. G. Liu, and L. Wei, "The PIM architecture for wide area multicasting," *IEEE/ACM Trans. Networking*, vol. 4, pp. 153–162, Apr. 1996.
- [4] A. Ballardie, Core Based Trees (CBT) Multicast Routing Architecture, RFC 2201, Sept. 1997.
- [5] S. Deering, Host Extensions for IP Multicasting, RFC 1112, Aug. 1989.
- [6] R. Hinden and S. Deering, IP Version 6 Addressing Architecture, RFC 1884, Dec. 1995.
- [7] W. Fenner, Internet Group Management Protocol, Version 2, RFC 2236, Nov. 1997.
- [8] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 623–656, July/Oct. 1948.
- [9] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer, 1992.
- [10] M. Nelson and J.-L. Gailly, *The Data Compression Book*, 2nd ed. New York: M&T Books, 1995.

- [11] F. M. Chiussi, J. G. Kneuer, and V. P. Kumar, "Low-cost scalable switching solutions for broadband networking: the ATLANTA architecture and chipset," *IEEE Commun. Mag.*, vol. 35, no. 12, pp. 44–53, Dec. 1997.
- [12] F. M. Chiussi and A. Francini, "A distributed scheduling architecture for scalable packet switches," *IEEE J. Select. Areas Commun.*, vol. 18, pp. 2065–2083, Dec. 2000.
- [13] A. Francini and F. M. Chiussi, "Providing QoS guarantees to unicast and multicast flows in multistage packet switches," *IEEE J. Select. Areas Commun.*, vol. 20, pp. 1589–1601, Oct. 2002.
- [14] A. N. Kolmogorov and S. V. Fomin, *Elements of Theory of Functions and Functional Analysis*. New York: Dover, 1999.



Marco G. Ajmone Marsan (S'76–M'76–SM'86–F'99) received electronic engineering degrees from Politecnico di Torino, Torino, Italy, and the University of California (UCLA), Los Angeles.

He is a Full Professor in the Electronics Department of Politecnico di Torino. Since September 2002, he has been the Director of the Institute for Electronics, Information and Telecommunications Engineering of the National Research Council. From November 1975 to October 1987, he was with the Electronics Department, Politecnico di Torino, first

as a Researcher, then as an Associate Professor. From November 1987 to October 1990, he was a Full Professor at the Computer Science Department, University of Milan, Milan, Italy. During the summers of 1980 and 1981, he was with the Research in Distributed Processing Group, Computer Science Department, UCLA. During the summer of 1998, he was an Erskine Fellow at the Computer Science Department, University of Canterbury, New Zealand. He has coauthored over 300 journal and conference papers in the areas of Communications and Computer Science, as well as *Performance Models of Multiprocessor Systems* (Cambridge, MA: MIT Press, 1986) and *Modeling with Generalized Stochastic Petri Nets* (New York: Wiley, 1995). His current interests are in the fields of performance evaluation of communication networks and their protocols.

Dr. Ajmone Marsan received the Best Paper Award at the Third International Conference on Distributed Computing Systems, Miami, FL, in 1982. In 2002, he was awarded a "Honoris Causa" degree in telecommunication networks from the Budapest University of Technology and Economics, Budapest, Hungary. He participates in a number of Editorial Boards of international journals, including the IEEE/ACM TRANSACTIONS ON NETWORKING.



Fabio M. Chiussi (M'93) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1993.

Since 1993, he has been with Bell Laboratories, Lucent Technologies, Holmdel, NJ, where he is currently Director, Data Networking and Wireless Systems. During these years, he has led the architectural design of three generations of the Lucent ATLANTA chipset, an industry-leading silicon solution for ATM and IP switching and port processing (now a product of Agere Systems); within the ATLANTA project, he

has also held various development responsibilities, including leading the development of the switch fabric devices for the latest generation of the chipset. His department is currently working on the development of several MPLS-centric switching systems, which are targeted for applications ranging from Metro Gigabit Ethernet to wireless access, and offer various flavors of Virtual Private Networks and other advanced IP services. He has been conducting fundamental research in the area of scalable switch architectures, traffic management and scheduling, protocols and architectures for wireless land networks, congestion control, and VLSI design. He has authored more than 80 technical papers and holds ten patents, with 20 more pending.

Dr. Chiussi was named the 1997 Eta Kappa Nu Outstanding Young Electrical Engineer and a Bell Laboratories Fellow.



Andrea Francini (S'97–M'98) received the Dr. Eng. degree in electrical engineering (*summa cum laude*) and the Ph.D. degree in electrical engineering and communications, in 1993 and 1998, respectively, both from the Politecnico di Torino, Torino, Italy.

Since 1996, he has been with Bell Laboratories, Lucent Technologies, Holmdel, NJ, in the Data Networking and Wireless Systems Department. He has contributed to the architectural design of three generations of the Lucent ATLANTA chipset (now a product of Agere Systems), leading the effort

for the deployment of advanced QoS support in the modules of the chipset. He is currently working on the architectural specification of MPLS-based switching systems in support of wireless access and transparent LAN services for the Metro environment. His research interests include traffic management, scheduling, scalable packet-switching architectures, and QoS frameworks for IP and MPLS networks.



Giulio Galante (S'00) received the Dr.Eng. degree in electronics engineering from Politecnico di Torino, Torino, Italy, in 1998, and is currently working toward the Ph.D. degree at the Electronics Department of Politecnico di Torino.

His Ph.D. research is mainly focused on the design of all-optical networks and on the support of multicast traffic in input-queued switches. During the summer 2000, he was an Intern at Lucent Technologies, Bell Labs, Holmdel, NJ. During 2001–2002, he visited the research group of Prof. N.

McKeown at Stanford University, Stanford, CA.



Emilio Leonardi (M'99) received the Dr.Eng. degree in electronics engineering and the Ph.D. degree in telecommunications engineering, in 1991 and 1995, respectively, both from Politecnico di Torino, Torino, Italy.

He is an Assistant Professor in the Electronics Department, Politecnico di Torino. In 1995, he was Visiting Scholar in the Computer Science Department, University of California, Los Angeles (UCLA). He was Visiting Researcher in the High-Speed Networks Research Department, Bell

Laboratories, Lucent Technologies, Holmdel, NJ (summer 1999) and in the Electrical Department, Stanford University, Stanford, CA (summer 2001), hosted by Prof. B. Prabakhar. He participated in several National and European projects, IST-SONATA and IST DAVID. He is also involved in several consulting and research projects with private industries, including Lucent Technologies, British Telecom, and TILAB. He has coauthored over 100 papers published in international journals and presented in leading international conferences. His areas of interest are in all-optical networks, queueing theory, and scheduling policies for high-speed switches.

Dr. Leonardi received the "IEEE TCGN Best Paper Award" for a paper presented at IEEE GLOBECOM 2002, High-Speed Networks Symposium. He has participated in the technical program committees of several conferences, including IEEE INFOCOM and IEEE GLOBECOM.