

Belief-Propagation Assisted Scheduling in Input-Queued Switches

Shadi Atalla, Davide Cuda, Paolo Giaccone, Marco Pretti

Abstract—We consider the problem of scheduling packets transmission across the switching fabric in an input-queued switch. In order to achieve maximum throughput, scheduling algorithms usually employ the queue length as a parameter for determining the priority to serve a given queue. Unfortunately, optimal schedulers are too complex to be implemented directly in hardware, which is the only viable solution in high-end switches, since scheduling decisions must be taken in very short time, of order of ns. Conversely, implementable algorithms, based on parallel and greedy decisions, are inefficient in terms of throughput.

In this work we do not propose any new scheduling algorithm or switching architecture, but a novel scheme to optimize the performance of a pre-existing scheduler. Indeed, we propose to assist the scheduling decision by considering “message” values instead of queue lengths. Such messages are obtained by running an iterative parallel algorithm (efficiently implementable in hardware and fully compatible with pre-existing schedulers), based on a rigorous Belief-Propagation approach. We show that BP-assisted scheduling is able to boost the performance of a given scheduler, reaching almost optimal throughput, even under critical traffic scenarios.

I. INTRODUCTION

Traffic demand in the Internet is keeping growing and the switching architectures need to be designed to process continuously increasing amount of data. One classical reference model for high-speed switching architectures is the input-queued (IQ) switch, in which packets are stored in queues, before being transmitted across a bufferless, high-bandwidth switching fabric. Such a scheme allows for maximum switching speed, since packets are transferred across the switching fabric at no more than the line rate. In an IQ switch, a scheduling algorithm has to choose, at any time, a set of non-conflicting packets to be transferred across the switching fabric. This task requires to solve a combinatorial optimization problem in a very short time, namely, a few ns, corresponding to the packet transmission time. Only hardware implementations are capable of meeting so short timing constraints.

Scheduling algorithm design has stimulated a large scientific literature, motivated by the following dichotomy. On the one hand, the Maximum Weight Matching (MWM) algorithm could ensure optimal performance, using the queue lengths as weights in the matching computation. Unfortunately, this algorithm is too complex to be implemented directly in hardware, because of its centralized and sequential nature. On the other hand, simpler (heuristic) scheduling algorithms, based

on parallel architectures, can be implemented satisfying timing constraints, but they turn out to suffer from heavy performance limitations.

Here, we propose a novel approach to boost the performance of an already implemented scheduler, whose decisions are based on the queue lengths, as it is the case in several schedulers mimicking the MWM decisions. The main underlying idea relies on the theory of probabilistic inference over graphical models, and specially on a related class of distributed *message-passing* algorithms, generally known as Belief Propagation (BP). BP has been first conceived as a dynamic-programming algorithm for evaluating marginal probabilities for so-called Markov random fields, defined on graphs without loops (trees) [1], [2]. The corresponding algorithms for computing maximum-a-posteriori probabilities, which can be used to solve combinatorial optimization problems like MWM [3], are known as *max-product* or *min-sum* (although they are sometimes called BP as well, as we shall do in this paper). It has been demonstrated that, in most cases, BP and related algorithms work surprisingly well even for graphs with loops (whence the term “loopy” BP), where they can be viewed as iterative algorithms. Heuristically, such a successful behavior seems to be related to the fact that BP fixed-points are equivalent to minima of an approximate free-energy function (Bethe free energy) for a corresponding thermodynamic system. In this framework, max-product and min-sum turn out to be equivalent to pure energy minimization, i.e., to a zero-temperature problem. By the way, the *Bethe approximation* is well-known by statistical physicists since long [4], but the connection with BP is a relatively recent result [5]. In short, BP is based on the exchange of messages among simple nodes, each one computing the values of the messages to be sent, as a function of the last received messages. Note that this approach can be naturally mapped into a computational network. After many iterations, the values of the messages are expected to converge and they can be used to find the solution of the original optimization problem. An interesting fact is that the decision can be taken in a parallel and distributed fashion as well, just based on local information at each node. In other words, in principle it is possible to compute the MWM in a completely distributed and parallel manner. Unfortunately, in practice messages may be slow to converge, preventing the final scheduling decision from occurring.

Our contribution does not consist in a new scheduler architecture, nor in a novel scheduling algorithm, nor in the introduction of BP in the field of switching architectures (proposed in [6] to compute the MWM). Rather, we introduce the concept of *assisted scheduling*, in which message values, computed through a suitable BP algorithm, are used (instead of

S. Atalla and P. Giaccone are with the Dipartimento di Elettronica, Politecnico di Torino, Italy. D. Cuda and M. Pretti are with Consiglio Nazionale delle Ricerche, Italy. This work was supported by Newcom++ Network of Excellence, funded by the European Commission through the 7th Framework Programme.

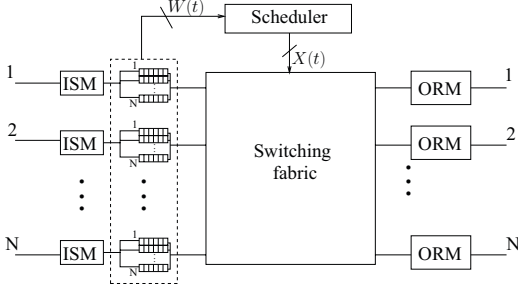


Fig. 1. Input queued switch architecture.

the usual queue lengths) as weights for conventional (heuristic) scheduling algorithms. Indeed, we do not require to perform any change on the original scheduling architecture and scheduler. Our solution is implementable natively in hardware, and it supports and improves the scheduler performance, at a low implementation cost. This choice, motivated by the nature of the BP algorithm itself, has the considerable advantage of not requiring to change the internal representation of the weights in the registers of the scheduler hardware. Furthermore, we improve the performance of BP by exploiting the time correlation induced by the queue behavior, namely, the fact that the queue lengths can vary at most by one for each packet departure or arrival. Accordingly, the optimal solution to the MWM problem at a given time is necessarily correlated with the one at subsequent times, and a similar correlation appears between the BP messages computed at adjacent times. In order to help BP convergence, we propose to keep memory of the previously computed messages and always to start iterating from such values.

The paper is organized as follows. In Sect. II we describe the scheduling architecture, and introduce the concept of assisted scheduling. In Sect. III we first describe the classical BP version of the algorithm and then our novel variation, aimed at improving performance and convergence speed. In Sect. IV we demonstrate by simulation-aid that our BP-assisted scheduling can actually boost throughput performance. In Sect. V we present a high-level hardware design of our BP-algorithm and, finally, in Sect. VI we draw some conclusions.

II. SYSTEM MODEL

We consider a $N \times N$ bufferless, non-blocking switching fabric, e.g. a crossbar, as shown in Fig. 1. We assume synchronous operations, i.e., time is divided into intervals of fixed duration (timeslots) and the switching fabric transfers data units of fixed size (cells). The timeslot duration is equal to the transmission time of a cell. In the case of variable-size packets, inputs are equipped with Input Segmentation Machines (ISM), to chop incoming packets into cells, whereas outputs are equipped with Output Reassembly Machines (ORM), to collect and reorder cells belonging to the same packet, before forwarding. No output queue is needed, except for reassembly and reordering purposes. Indeed, we also assume that no speedup is available, i.e., cells are transferred across the switching fabric at the line rate: This allows high scalability in terms of number of ports and bandwidth. When cells from different

inputs are addressed to the same output, just one cell can be actually transferred, while the other cells are stored in the input queues, organized according to the Virtual Output Queue (VOQ) architecture, to avoid the well known Head of the Line (HoL) blocking problem [7]. In more detail, each input is equipped with N separate FIFO queues (one for each output) of size Q_{\max} , each queue storing cells addressed to a specific output: a total of N^2 queues is present at the inputs, each one writing and reading cells at the line rate. Let $W(t) = [w_{ij}(t)]$ be a $N \times N$ matrix in which $w_{ij}(t)$ is the queue length of the VOQ from input i to output j at timeslot t .

A. The scheduler

At each timeslot, a centralized scheduler, shown in Fig. 1, selects a set of HoL cells to transfer, that satisfy the physical constraints of the switching fabric, i.e., at most one cell can be transferred from any input to any output during the same timeslot. A feasible switching configuration can be represented by a matching in a bipartite graph of $2N$ nodes, whose N left-side nodes correspond to the inputs and the N right-side nodes correspond to the outputs; the edge connecting input i to output j is associated to the corresponding VOQ. Let $X(t) = [x_{ij}(t)]$ be a $N \times N$ matrix in which $x_{ij}(t)$ is a boolean variable equal to 1 iff the scheduler selects to transfer a cell from input i to output j during timeslot t ; if $w_{ij}(t) = 0$, we set $x_{ij}(t) = 0$. To satisfy the transfer constraints, it should be:

$$\sum_{k=1}^N x_{ik}(t) \leq 1 \quad \sum_{k=1}^N x_{kj}(t) \leq 1 \quad \forall i, j, t \quad (1)$$

Each queue evolution is simply described by $w_{ij}(t+1) = w_{ij}(t) + a_{ij}(t) - x_{ij}(t)$, where $a_{ij}(t)$ is another boolean variable, equal to one iff a new cell has just arrived. For the sake of conciseness, in the following we will omit t from the notation when the context is not ambiguous.

The scheduling decision is based on the state of the queues, obtained by a control path connecting the queueing system to the scheduler: We assume that the control path communicates the actual occupancy state W of all VOQs to the scheduler, as shown in Fig. 1. If the weight of the edge from input i to output j is equal to w_{ij} , it is well known [8] that computing the MWM on the queue lengths guarantees 100% throughput under any admissible Bernoulli i.i.d. traffic; this result has been extended to more general arrival processes [9]. Unfortunately, MWM requires $O(N^3)$ operations and it cannot be efficiently implemented in hardware, due to its sequential nature, difficult to parallelize. Indeed, to the best of our knowledge, MWM has never been implemented on commercial chipsets but it has inspired a large literature investigating heuristic algorithms, simple to implement in hardware even if not throughput-optimal. Notably, we mention iSLIP [10], that is an iterative algorithm in which N arbiters concurrently select a subset of edges to compute the matching, in a parallel and distributed fashion. Due to its parallel nature, compatible also with a pipeline processing, as discussed in [11], it was implemented on a single chip and used in a commercial core router. iSLIP does not take into account the queue lengths and hence approximates a maximal size matching. To better approximate



Fig. 2. Assisted scheduling approach: BP-MP module feeds the scheduler with the BP-messages instead of the queue lengths.

the MWM, iLQF [12] has been proposed: It can be designed with the same architecture of iSLIP, but with a larger control information exchanged by the arbiters. Many other algorithms have been proposed to approximate the MWM or to achieve similar performance [13], [14], [15], by giving priority to longer queues. The crucial issue is that optimal or almost optimal algorithms are difficult to be implemented in hardware; conversely, implementable algorithms do not achieve 100% throughput under specific traffic scenarios.

B. Assisted scheduling

Motivated by such dichotomy, we propose a method to boost the performance of a generic scheduler that takes its decisions based on a queue length matrix $W(t)$. This method can be compatible with pre-existing scheduler implementations, since it adds just a functional module between the VOQ and the scheduler, denoted as *Belief-Propagation Message-Processing* (BP-MP) in Fig. 2. Basically, BP-MP computes some message values, as a function of the queue lengths, based on a rigorous BP algorithm. The scheduler works in the usual way, but, instead of being fed by the queue lengths $W(t)$, it is fed by the messages $F(t)$, with the same numerical format as $W(t)$, computed by BP-MP. Roughly speaking, it is like if BP-MP were “cheating” the scheduler by providing fake queue lengths to the scheduler, in order to boost the performance. As one can argue from Appendix A (which reports a detailed presentation of the original BP algorithm), the messages $F(t)$ can be viewed as corrections over the corresponding queue lengths, done by taking into account possible violations of the matching constraints (1), which might occur if every output j should trivially decide to serve the input i with the maximum queue length w_{ij} . The interesting thing is that BP-MP can be independent from the scheduler hardware and is amenable to parallel implementation, as discussed later in Sec. V.

III. THE SCHEDULING ALGORITHM

The standard and classical algorithm to compute the MWM is based on iterating the Bellman-Ford augmentation algorithm, and the final computational complexity is $O(N^3)$ with optimized data structures. It is a sequential algorithm, difficult to parallelize and implement in hardware. On the other hand, BP provides an alternative rigorous approach to compute the MWM (which we shall denote as “BP-MWM”), natively based on parallel computing. Sec. III-A describes the basic BP-MWM algorithm; Sec. III-B explains how to integrate it with a pre-existing scheduler and provides the main algorithmic contributions of our work.

A. Basic belief-propagation for MWM

The BP-based algorithm to compute the MWM, which we consider in this paper, is fully described in Fig. 3. It is basically

```

BP-MWM: Input:  $W = [w_{ij}]$ ; Output:  $X = [x_{ij}]$ ,  $I_{\text{conv}}$ 
// Initialization
for all  $i, j$ 
   $x_{ij} = 0$ ,  $f_{i \rightarrow j}^0 = w_{ij}$ ,  $b_{j \rightarrow i}^0 = w_{ij}$ 
// Update phase
 $n = 1$  // initialize the number of iterations
while (messages not yet converged)
  for all  $i, j$ 
     $f_{i \rightarrow j}^n = \max\{0, w_{ij} - \max_{k \neq j} b_{k \rightarrow i}^{n-1}\}$ 
     $b_{j \rightarrow i}^n = \max\{0, w_{ij} - \max_{k \neq i} f_{k \rightarrow j}^{n-1}\}$ 
   $n = n + 1$ 
 $I_{\text{conv}} = n - 1$  // store the number of iterations to converge
// Estimate phase
for all  $j$  // at each output
   $\hat{i} = \arg \max_i f_{i \rightarrow j}^{I_{\text{conv}}}$ 
  if  $f_{\hat{i} \rightarrow j}^{I_{\text{conv}}} > 0$  then  $x_{ij} = 1$ 

```

Fig. 3. Pseudocode of the basic BP-MWM algorithm.

equivalent to the min-sum formulation studied in [3] (though the latter is based on a different graphical model), and is also very similar to the one developed in [16] for the more general b -matching problem. In Appendix A we report a theoretical derivation, aimed at explaining the conceptual origin of the algorithm, without resorting to previous knowledge from the theory of graphical models. As previously mentioned, the algorithm takes as input a fixed weight matrix W and gives as output the matching, represented by the boolean matrix X . The quantities on which it operates are a matrix $F = [f_{i \rightarrow j}]$ of “forward messages”, from any input i to any output j , and an analogous matrix $B = [b_{j \rightarrow i}]$ of “backward messages” (from outputs to inputs). Both matrices are initialized as $F = B = W$, whereas all the entries of the X matrix are set at 0. Then, the messages are iteratively updated according to the rules reported in Fig. 3 (Update phase), until convergence. At the end of the iterative procedure, which we assume to be completed in I_{conv} iterations (a priori unknown), each output j determines the input \hat{i} corresponding to the maximum forward message (Estimate phase). If such a message is nonzero, then \hat{i} is connected to j in the matching. Note that it is possible to define an identical (mirrored) estimate phase at the inputs.

In principle, a BP algorithm like that of Fig. 3 is not guaranteed to converge on a generic loopy graph (as in our case), so that also the x_{ij} assignments might not represent a feasible matching. Actually, the recent work by Bayati, Shah, and Sharma [3] has proved both convergence and exactness of BP for precisely our problem (namely, MWM on a complete bipartite graph), under the assumption that the MWM is unique. Although this result is extremely interesting from a theoretical point of view, and the algorithm is amenable to parallel computing, a practical implementation of a pure BP algorithm still poses several problems. First of all, the number of iterations I_{conv} is still $O(N^3)$ but it is not a-priori known, and turns out to depend on the weights [3]. Note that, before reaching convergence, the messages cannot be directly used for the Estimate phase, since the distributed choice may lead to unfeasible solutions, not satisfying the matching

constraints (1). Moreover, [3] points out that the convergence time is bounded by $O(\epsilon^{-1})$, where ϵ is the difference between the weight of the “first-ranked” MWM and that of the “second-ranked” MWM. Indeed, the lower is such difference, the longer BP will take to converge; when the MWM is not unique, BP may not converge at all. We expect that the latter property would be rather limiting in a real device, since, in order to maximize throughput, the MWM scheduler tends to “equalize” queue lengths, likely giving rise to a large number of different matchings with similar or equal weights. Alternatively, one might satisfy the MWM uniqueness requirement, by adding some random “noise” to W before feeding it to the algorithm, but it is easy to realize that such a possibility would pose some extra challenges, concerning the noise samples generation and the numerical representation of the messages.

B. Belief-propagation assisted scheduling

Hereafter, we discuss the techniques to implement efficiently the BP-MP module and assist the scheduling decision. Basically, BP-MP runs just the update phase of BP-MWM. We start by observing that, when convergence is reached during the update phase, all the edges with the largest messages for each output (or input) will correspond to the exact MWM. Interestingly, most of the greedy algorithms to approximate the MWM will choose the same set of edges as if the weights were given by F instead of W . In particular, the estimate phase of BP-MWM is identical to the Grant phase of iLQF [12], where the output arbiters grant the requests corresponding to the largest weight. Hence, we can claim:

Property 1: If the MWM is unique, BP-assisted iLQF, running with the edge weights given by $F = [f_{i \rightarrow j}^{I_{\text{conv}}}]$, computes exactly the MWM.

In BP-MP we fix the number of iterations I usually equal to a very small constant value ($I = 2$ or 3); the messages may not converge ($I \ll I_{\text{conv}}$), but this is not a problem since the scheduler will compute a feasible (possibly, maximal) matching; in this case, we can just claim that BP-assisted iLQF computes an approximation of the MWM. Of course, we can expect that, as I grows, the scheduler performance will approach the MWM, and, for $I \geq I_{\text{conv}}$, iLQF will behave exactly as the MWM. To improve the message convergence, i.e., to reduce I_{conv} , and to simplify BP-MP hardware implementation, we propose the following three techniques: message memory, self-asynchronous update and integer message representation. Pseudocode in Fig. 4 shows the final scheme we propose.

1) *Message memory:* Each queue can vary its occupancy by at most one, due to arrivals and departures: $|w_{ij}(t+1) - w_{ij}(t)| \leq 1, \forall t$. This means that the state of the queues exhibits a strong time correlation, that is reflected in the message dynamics. Hence, BP-MP uses the last computed message from the previous timeslot to initialize the messages at the beginning of a new iteration:

$$f_{i \rightarrow j}^0(t) = f_{i \rightarrow j}^I(t-1) \quad b_{j \rightarrow i}^0(t) = b_{j \rightarrow i}^I(t-1)$$

2) *Self-asynchronous update:* Description of BP-MWM in Fig. 3 assumes synchronous and parallel message updates,

```

BP-MP: Input:  $W(t) = [w_{ij}(t)]$ ; Output:  $[f_{i \rightarrow j}^I(t)]$ 
// Memory
for all  $i, j$ 
   $f_{i \rightarrow j}^0(t) = f_{i \rightarrow j}^I(t-1), b_{j \rightarrow i}^0(t) = b_{j \rightarrow i}^I(t-1)$ 
// Update phase
for  $n = 1 \dots I$  // run for  $I$  iterations
  for all  $i, j$ 
    if  $w_{ij}(t+1) \neq w_{ij}(t)$  // self-asynchronous update
       $f_{i \rightarrow j}^n(t) = \max\{0, w_{ij}(t) - \max_{k \neq j} b_{k \rightarrow i}^{n-1}(t)\}$ 
       $b_{j \rightarrow i}^n(t) = \max\{0, w_{ij}(t) - \max_{k \neq i} f_{k \rightarrow j}^{n-1}(t)\}$ 
    else
       $f_{i \rightarrow j}^n(t) = f_{i \rightarrow j}^{n-1}(t)$ 
       $b_{j \rightarrow i}^n(t) = b_{j \rightarrow i}^{n-1}(t)$ 

```

Fig. 4. Pseudocode of Belief-Propagation Message-Processing (BP-MP).

rather easy to implement in hardware. Early studies on BP algorithms [17] have shown that asynchronous updates (i.e., messages are updated in a random sequential order) are beneficial for the convergence. Unfortunately, asynchronous implementation is very difficult to be obtained in hardware, since it requires $O(N^2)$ sequential iterations. BP-MP mimics the asynchronous behavior by updating, in the usual synchronous way, just a subset of messages, chosen on the basis of the arrival process: In some sense, this technique exploits the arrivals as a randomness source. The main idea consists of updating only messages corresponding to queues that changed their lengths: Hence, at most $2N$ messages are updated for each timeslot. This also emphasizes the memory effect, enhancing the time correlation between messages at different timeslots.

3) *Integer message representation:* We have already mentioned that, in BP-MWM, some “noise” must be added to the weights, in order to guarantee a unique MWM and convergence [3]. In general, this is not simple to obtain, because one has to generate random samples, which also need to be represented as real numbers. Fortunately, in our architecture, BP is not required to converge, and hence BP-MP does not need to add noise; intuitively, in the case of multiple optimal solutions, the arrivals in the future timeslots will change the MWM and will provide an effect similar to the random noise. The main consequence is that the messages are thus integer numbers. Furthermore, according to the update rules of Fig. 3, the following bounds hold: $0 \leq f_{i \rightarrow j} \leq w_{ij}$, $0 \leq b_{j \rightarrow i} \leq w_{ij}$, implying that the numerical range of the messages is the same as that of the queue lengths, that is, between 0 and Q_{max} . In the framework of the assisted-scheduling approach, this detail is of great importance, since it allows us to represent the messages with the same number of bits as the queue lengths. This ensures backward compatibility in the control path between the BP-MP module and the scheduler module.

IV. PERFORMANCE EVALUATION

We investigate the performance of BP-assisted scheduling through simulations, run on a proprietary discrete-time simulator written in C.

A. Simulation settings

We consider both uniform and unbalanced traffic scenarios, common reference testbeds to analyze the performance of schedulers in IQ switches. Let ρ be the load at each input port, and λ_{ij} the load from input i to output j ($\Lambda = [\lambda_{ij}]$ is called the traffic matrix). Under uniform traffic, $\lambda_{ij} = \rho/N$. Standard practice among researchers is to require a scheduling algorithm to obtain 100% throughput under such scenario, that is not considered critical. To better evaluate the performance under more critical traffic, we consider two different unbalanced scenarios: *bi-diagonal* and *log-diagonal*. Under bi-diagonal traffic, $\lambda_{ii} = 2\rho/3$ and $\lambda_{i|i+1|N} = \rho/3$, where $|x|_N = ((x-1) \bmod N) + 1$; i.e., input port i sends traffic just to output i and $|i+1|_N$. This scenario is a worst case, since just two matchings are “optimal”, in the sense that any other matching, chosen by the scheduler, leads to a throughput degradation. No greedy algorithm can achieve the maximum throughput under such critical traffic, as also shown later. Under log-diagonal traffic, $\lambda_{ij} \propto 2^{|j-i|_N} \rho$, i.e., the traffic grows as powers of 2 along the diagonals of the traffic matrix. This scenario can be considered a hybrid case between uniform and bi-diagonal.

As far as the maximum queue size is concerned, we set $Q_{\max} = 1000$ cells. We report here just the throughput vs. the normalized input load. Results are obtained with an accuracy better than 1% with a confidence interval of 95%.

B. The scheduling algorithms

We consider three different algorithms: iSLIP [10], Longest Queue First (iLQF) [12] and a basic greedy approximation of the MWM, denoted as GWM.

Let us start by describing iLQF. It is a three-phase algorithm (Request, Grant, and Accept), running in the following way. During the Request phase (from inputs to outputs), each unmatched input sends all its queue lengths as requests to corresponding outputs. During the Grant phase (from outputs to inputs), if an unmatched output receives requests, it sends a grant to the input corresponding to the longest queue (contentions are solved randomly). In the Accept phase (from inputs to outputs), if an unmatched input receives grants, it selects the output associated to the longest queue (contentions are solved randomly). iLQF may run several iterations, at each iteration considering only unmatched inputs and outputs. The final matching is guaranteed to be maximal if the number of iterations is N , but it has been shown that, under uniform traffic, $\log_2 N$ are actually sufficient [12]. In the following, we shall always consider $\log_2 N$ iterations for iLQF, unless differently specified.

iSLIP runs almost exactly as iLQF, but without taking into account the queue lengths: It is like setting $w_{ij} = 0, 1$ if the corresponding queue is empty or nonempty, respectively. Hence, iSLIP cannot be assisted by BP.

GWM is a centralized scheduling algorithm that computes maximal matchings in a greedy way. At each iteration, GWM selects the unmatched input-output pair associated to the longest queue; it matches such input-output pair and continues

Traffic	iSLIP	iLQF	BP-iLQF	GWM	BP-GWM
Uniform	0.99	0.99	0.99	0.99	0.99
Log-diagonal	0.83	0.97	0.97	0.97	0.97
Bi-diagonal	0.83	0.87	0.98	0.87	0.98

TABLE I
MAXIMUM THROUGHPUT ACHIEVED FOR $N = 32$ AND $\rho = 0.99$

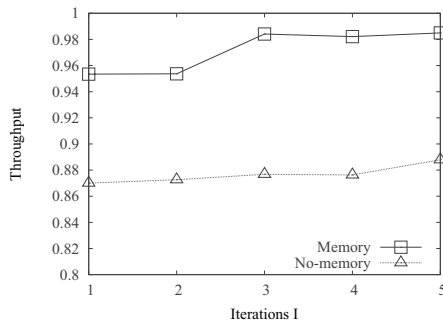


Fig. 5. Throughput achieved under bi-diagonal traffic for $N = 32$ and $\rho = 0.99$ for BP-iLQF, with self-asynchronous update, with or without memory.

with a new iteration. After N iterations, the final matching is maximal.

As far as notation is concerned, in the following we shall use iLQF and GWM to denote the original algorithms, described above, and BP-iLQF and BP-GWM to denote the corresponding BP-assisted versions.

C. Simulation results

Table I displays the maximum throughput under uniform, log-diagonal and bi-diagonal traffic scenarios. The observed average delays for BP-iLQF/GWM are at most 1.37 times the delays of iLQF/GWM. Except for uniform traffic, iSLIP behaves poorly, as well known in the literature. The performances of the BP-assisted schedulers are evaluated in the presence of message memory and self-asynchronous update; we set $I = 3$. The relevance of the contributions given by these techniques will be investigated in the following. Under uniform traffic, all the schedulers obtain maximum throughput. Under log-diagonal traffic, the performance gain due to BP is negligible, even if all the algorithms achieve throughput larger than 0.97. Under bi-diagonal traffic, for both iLQF and GWM, BP-assisted scheduling is able to boost the performance by more than 10%, which can be a relevant value in absolute terms for a terabit switch. Note that the performance does not depend significantly on the scheduler, meaning that BP-MP is beneficial for both schedulers.

Accordingly, from now on we shall consider only one scheduler, namely BP-iLQF, always under bi-diagonal traffic scenario, which appears to be the most meaningful one. Fig. 5 shows the effect of the memory. It turns out that, almost independently of the number of iterations, the memory strongly affects the system performance. This result confirms that the strong time correlation between messages can be efficiently exploited. Here, $I = 3$ is sufficient to obtain almost maximum throughput.

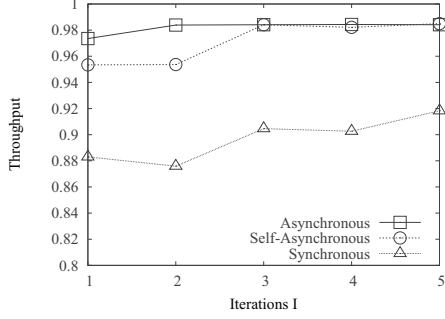


Fig. 6. Throughput achieved by BP-iLQF, with message memory, under bi-diagonal traffic for $N = 32$ and $\rho = 0.99$, when different update rules are adopted.

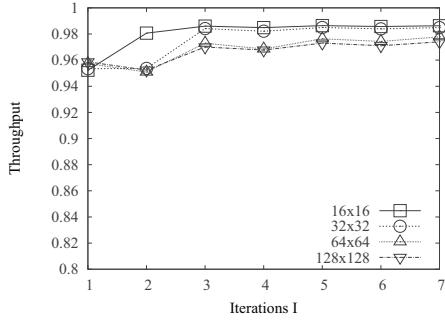


Fig. 7. Throughput achieved by BP-iLQF, with message memory and self-asynchronous update, under bi-diagonal traffic for different N and $\rho = 0.99$.

Fig. 6 shows the effect of the different BP updating rules. As expected, asynchronous update turns out to be the most efficient one, but, as previously mentioned, it is difficult to be implemented in hardware. On the other hand, synchronous update, even though amenable to parallel implementation, behaves poorly, almost independently of the number of iterations. Quite interestingly, the self-asynchronous update scheme turns out to be nearly as efficient as the purely asynchronous one: this means that arrival randomness somehow mimics an asynchronous update. Finally, Fig. 7 investigates the performance of BP-iLQF for different switch sizes; in all the cases, few iterations are needed to get almost optimal performances. $I = 1$ is enough to improve from 0.87 to 0.95, while slightly larger numbers of iterations ($I \geq 3$) reach almost optimal performance.

V. HARDWARE DESIGN ISSUES

In this section, we focus on a possible hardware design of the BP-MP module. We consider a centralized implementation within a single chip, installed between the VOQs and the scheduler, as shown in Fig. 2.

Fig. 8 displays the high-level architecture implementing BP-MP. This architecture is composed by $2N$ modules, $\{\text{IM}_i\}_{i=1}^N$ modules to compute forward messages $f_{i \rightarrow j}^n$ and $\{\text{OM}_i\}_{i=1}^N$ modules to compute backward messages $b_{j \rightarrow i}^n$, n being the current iteration. All the modules run in parallel and iteratively exchange messages between each other; after I iterations, the

modules send the final values of the forward messages to the scheduler. More precisely, module IM_i is connected with all the $\{\text{OM}_j\}_{j=1}^N$ modules and each OM_j module is connected with all the $\{\text{IM}_i\}_{i=1}^N$ modules. In Fig. 8, outgoing connections of a module are represented by OUT edges whereas input connections are represented by IN edges. At each iteration (but the first one), each module uses the messages received during the previous iteration by the IN-connected modules and the current queue lengths W to compute the new outgoing messages, which are then sent to the OUT-connected modules. Since we exploit message memory (see Sect. III-B), at the first iteration, messages stored since the last iteration of the previous time slot are used. Note that all the IM_i and OM_j modules perform the same operations in parallel at each iteration, but only the IM_i are responsible for updating the scheduler with the final forward messages $f_{i \rightarrow j}^I$.

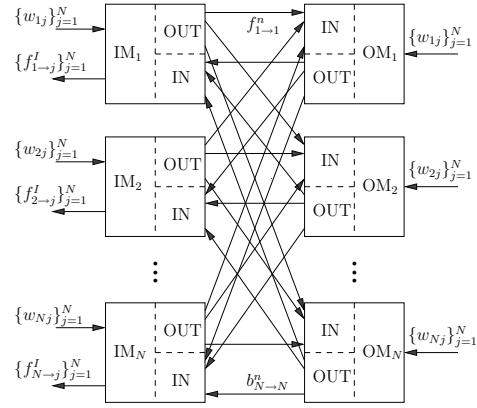


Fig. 8. Possible hardware architecture of the BP-MP module, implemented on a single chip.

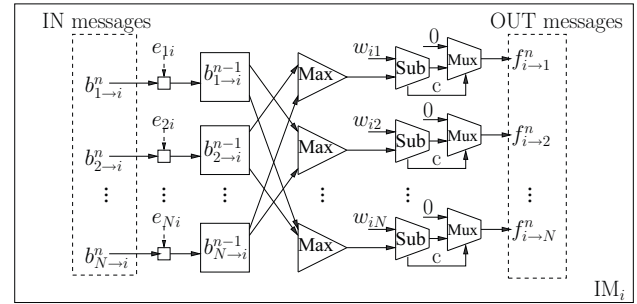


Fig. 9. Detail of the hardware design of each IM_i module elaborating forwarding messages. Identical design for OM_j .

Fig. 9 shows the detail of the operations performed by IM_i to compute the new outgoing messages. Each module is equipped with N registers of size $\log_2 Q_{\max}$ bits, N modules performing the Max operation and N modules executing the subtraction (Sub) operation. Both the Max and Sub operation modules require operands with size $\log_2 Q_{\max}$ since the message value can be bounded between 0 and $Q_{\max} - 1$. Let $\{e_{ij}(t)\}_{j=1}^N$ be the set of flags associated with VOQs at input i ; $e_{ij}(t)$ is equal to 1 if the length of the queue associated to

output j at input i changed from the previous time slot, i.e., if $w_{ij}(t) \neq w_{ij}(t-1)$. Flags e_{ij} are used to implement the self-asynchronous update mechanism (see Sect. III-B), indeed, the values stored in a register are updated only if $e_{ij} = 1$. Note that, at each time slot, an input port handles at most one cell arrival and one cell departure; i.e., at most two different queue lengths may change from one time slot to the following one. Hence, at each time slot, at most two registers are actually enabled for the updating operations in each module.

At each iteration n , each module computes N messages using the hardware structure depicted in Fig. 9 performing N Max and N Sub operations. In more detail, the Max operation is carried out through a tournament implementation, returning the maximum among the $N-1$ input messages $b_{i \rightarrow j}^n$ after $\lceil \log_2(N-1) \rceil$ stages performing only $N-2$ comparisons and selections. Outputs of Max operations are then subtracted from the queue lengths W and the multiplexer stage (Mux) selects either 0 or the outcome of the subtraction, depending on the underflow bit of the subtraction (labeled c in Fig. 9). The final $[f_{i \rightarrow j}^n]$ are sent to the OUT-connected modules. When n is equal to I , IM_i modules forward their $[f_{i \rightarrow j}^I]$ to the scheduler.

VI. CONCLUSIONS

We have proposed the belief-propagation assisted scheduling as a viable approach to boost the performance of a given scheduler, provided the scheduling decisions are based on the queue lengths. Our scheme is based on rigorous belief-propagation, enhanced by message memory and self-asynchronous message update; we show that the performances are very close to the optimal ones. We have discussed some hardware implementation issues to demonstrate actual implementability of the approach, leveraging a parallel architecture to compute the messages.

In conclusion, we believe that BP-based algorithms are very promising to design generic network control systems, also in contexts out of the scope of this work, because of their appealing tradeoff between implementation complexity and performance.

REFERENCES

- [1] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artificial Intelligence*, vol. 29, no. 3, pp. 241–288, Sep. 1986.
- [2] —, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco CA, 1986.
- [3] M. Bayati, D. Shah, and M. Sharma, "Max-product for maximum weight matching: Convergence, correctness, and LP duality," *Information Theory, IEEE Transactions on*, vol. 54, no. 3, pp. 1241–1251, Mar. 2008.
- [4] H. A. Bethe, "Statistical theory of superlattices," *Proc. R. Soc. A*, vol. 150, no. 871, pp. 552–575, Jul. 1935.
- [5] J. Yedidia, "An idiosyncratic journey beyond mean field theory," in *Advanced Mean Field Methods: Theory and Practice*, M. Opper and D. Saad, Eds. MIT Press, Cambridge MA, 2001, pp. 21–35.
- [6] M. Bayati, B. Prabhakar, D. Shah, and M. Sharma, "Iterative scheduling algorithms," in *INFOCOM 2007, IEEE, 6-12 2007*, pp. 445–453.
- [7] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queuing on a space-division packet switch," *Communications, IEEE Transactions on*, vol. 35, no. 12, pp. 1347–1356, Dec. 1987.
- [8] N. McKeown, A. Mekkitikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *Communications, IEEE Transactions on*, vol. 47, no. 8, pp. 1260–1267, Aug. 1999.
- [9] J. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *INFOCOM 2000, IEEE, vol. 2, 2000*, pp. 556–564.

- [10] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *Networking, IEEE/ACM Transactions on*, vol. 7, no. 2, pp. 188–201, Apr. 1999.
- [11] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar scheduler," *Micro, IEEE*, vol. 19, no. 1, pp. 20–28, Jan./Feb. 1999.
- [12] N. W. McKeown, "Scheduling algorithms for input-queued cell switches," Ph.D. dissertation, Berkeley, CA, USA, 1995.
- [13] N. Chrysos and G. Dimitrakopoulos, "Practical high-throughput crossbar scheduling," *Micro, IEEE*, vol. 29, no. 4, pp. 22–35, July-Aug. 2009.
- [14] S. Beldianu, R. Rojas-Cessa, E. Oki, and S. Ziaavras, "Scheduling for input-queued packet switches by a re-configurable parallel match evaluator," *Communications Letters, IEEE*, vol. 14, no. 4, pp. 357–359, Apr. 2010.
- [15] V. Tabatabaee and L. Tassiulas, "MNCM: A critical node matching approach to scheduling for input buffered switches with no speedup," *Networking, IEEE/ACM Transactions on*, vol. 17, no. 1, pp. 294–304, Feb. 2009.
- [16] M. Bayati, C. Borgs, J. Chayes, and R. Zecchina, "On the exactness of the cavity method for weighted b-matchings on arbitrary graphs and its relation to linear programs," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 06, Jun. 2008.
- [17] M. F. Tappen and W. T. Freeman, "Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters," in *ICCV, 2003*, pp. 900–907.

APPENDIX A

DERIVATION OF BP FOR MWM

In this appendix we derive a BP algorithm for solving the MWM problem on a generic (simple, undirected) weighted graph $G(V, E)$. This is a more general case than the bipartite graph considered for the scheduling problem. Vertices are denoted by labels $i, j, k, \dots \in V$, whereas $x_{ij} \in \{0, 1\}$ denote *configuration variables*, associated to each edge of the graph $ij \in E$ (as the graph is undirected, ij and ji denote the same edge). A matching is defined by a choice of the configuration variables such that, for each vertex $i \in V$, there is at most one variable $x_{ij} = 1$, i.e., (1) can be rephrased as:

$$\sum_{j \in \partial i} x_{ij} \leq 1 \quad (2)$$

Here, $\partial i \triangleq \{j \in V \mid ij \in E\}$ denotes the neighborhood of i , i.e. the set of vertices connected to i by an edge. Note that (2) allows for the possibility that the sum vanishes, meaning that the i vertex remains unmatched. Given the complete set of configuration variables $X \triangleq \{x_{ij}\}_{ij \in E}$, the weight of the corresponding matching is defined by the function

$$w(X) \triangleq \sum_{ij \in E} w_{ij} x_{ij} \quad (3)$$

where w_{ij} denotes the weight of the ij edge. Our optimization goal is therefore to find

$$\hat{X} \triangleq \arg \max_X w(X) \quad (4)$$

where it is understood that X varies over the sets of configuration variables satisfying the constraints (2).

In general, it is possible to derive a BP algorithm, for a given problem, by "faking" that the underlying graph is a tree. According to our fake assumption, let us consider, for a given edge ij , the subtree having i as a root site, obtained by removing the connections toward all its neighbors except j (see Fig. 10). We call this object the $j \rightarrow i$ subtree, and the corresponding set of edges is denoted by $E_{j \rightarrow i} \subseteq E$. Given

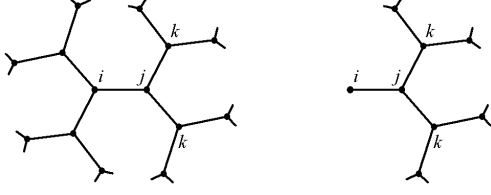


Fig. 10. Tree graph (left), and the $j \rightarrow i$ subtree (right).

the set of configuration variables in the subtree, denoted as $X_{j \rightarrow i} \triangleq \{x_{kl}\}_{kl \in E_{j \rightarrow i}}$, we can define a partial weight function

$$w_{j \rightarrow i}(X_{j \rightarrow i}) \triangleq \sum_{kl \in E_{j \rightarrow i}} w_{kl} x_{kl} \quad (5)$$

obtained by restricting the sum in (3) only to edges contained in the subtree itself. For each given vertex i , (5) allows us to rewrite the total weight function (3) as

$$w(X) = \sum_{j \in \partial i} w_{j \rightarrow i}(X_{j \rightarrow i})$$

Therefore, according to (4), the optimal values for $X_i \triangleq \{x_{ij}\}_{j \in \partial i}$ (i.e., the set of configuration variables in the neighborhood of i) can be determined as

$$\hat{X}_i = \arg \max_{X_i} \sum_{j \in \partial i} m_{j \rightarrow i}(x_{ij}) \quad (6)$$

where, as in (4), X_i varies over values allowed by (2), while we have defined the partial maxima

$$m_{j \rightarrow i}(x_{ij}) \triangleq \max_{X_{j \rightarrow i} \setminus x_{ij}} w_{j \rightarrow i}(X_{j \rightarrow i}) \quad (7)$$

In the latter equation, the maximization is performed over the set of subtree variables $X_{j \rightarrow i}$, except the ‘‘root variable’’ x_{ij} . These partial maxima are called *messages* in the BP jargon. Let us also define the *differential messages*

$$\mu_{j \rightarrow i} \triangleq m_{j \rightarrow i}(1) - m_{j \rightarrow i}(0)$$

which allow us to write

$$m_{j \rightarrow i}(x_{ij}) = \mu_{j \rightarrow i} x_{ij} + \text{constant}$$

where ‘‘constant’’ means, in this case, ‘‘independent of the configuration variables’’. The argmax operation of (6) is not affected by these additive constants, so that the optimal assignments of the configuration variables can be done by looking only at the differential messages, which simplifies the whole procedure. Now, if we assume that the MWM is unique, it turns out that (i) the differential messages cannot be zero, and (ii)

$$\hat{j} \triangleq \arg \max_{j \in \partial i} \mu_{j \rightarrow i}$$

is unique. As a consequence, taking into account (6), one can deduce the following optimal assignments

$$\hat{x}_{ij} = 0, \quad \forall j \in \partial i \setminus \hat{j}; \quad \hat{x}_{ij} = \begin{cases} 1 & \text{if } \mu_{j \rightarrow i} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

In order to compute the messages, let us observe that it is possible to decompose the partial weight function of the

$j \rightarrow i$ subtree as a sum of partial weight functions for its sub-subtrees $k \rightarrow j$ (with k varying over the neighbors of j except i , which we denote as $\partial j \setminus i$), plus the elementary weight of the ij edge (see Fig. 10):

$$w_{j \rightarrow i}(X_{j \rightarrow i}) = w_{ij} x_{ij} + \sum_{k \in \partial j \setminus i} w_{k \rightarrow j}(X_{k \rightarrow j}) \quad (9)$$

As a consequence, one obtains a set of relations between the messages from each given subtree $j \rightarrow i$ and those from its sub-subtrees $k \rightarrow j$. Due to the tree structure, the maximization in (7) can be decomposed into independent maximizations over the sub-subtrees variables (without root) $X_{k \rightarrow j} \setminus x_{jk}$, and a maximization over the only root variables $\{x_{jk}\}_{k \in \partial j \setminus i} = X_j \setminus x_{ij}$. From (7) and (9) we thus obtain

$$m_{j \rightarrow i}(x_{ij}) = \max_{X_j \setminus x_{ij}} \left\{ w_{ij} x_{ij} + \sum_{k \in \partial j \setminus i} m_{k \rightarrow j}(x_{jk}) \right\} \quad (10)$$

Although the x_{ij} variable is not subject to maximization, the corresponding weight term $w_{ij} x_{ij}$ cannot get out of the brackets, because the max operation must take into account the matching constraint, which depends on the value of x_{ij} . Indeed, (10) is very general; making explicit use of the matching constraint leads to a particular form. We have to consider the two cases $x_{ij} = 0, 1$ separately. If $x_{ij} = 1$, then the matching constraint imposes that, for all other $k \in \partial j \setminus i$, it must be $x_{jk} = 0$, so that (10) yields

$$m_{j \rightarrow i}(1) = w_{ij} + \sum_{k \in \partial j \setminus i} m_{k \rightarrow j}(0) \quad (11)$$

Conversely, if $x_{ij} = 0$, the matching constraint allows for different cases: Either $x_{jk} = 0$ for all other $k \in \partial j \setminus i$, as in the previous case, or $x_{jk} = 1$ for a particular k , and $x_{jl} = 0$ for all other $l \in \partial j \setminus i \setminus k$. As a consequence, (10) yields

$$m_{j \rightarrow i}(0) = \max \left\{ \sum_{k \in \partial j \setminus i} m_{k \rightarrow j}(0), \max_{k \in \partial j \setminus i} \left\{ m_{k \rightarrow j}(1) + \sum_{l \in \partial j \setminus i \setminus k} m_{l \rightarrow j}(0) \right\} \right\} \quad (12)$$

Eqs. (11) and (12) together yield the following, very simple *propagation equation* for the differential messages

$$\mu_{j \rightarrow i} = w_{ij} - \max\{0, \max_{k \in \partial j \setminus i} \mu_{k \rightarrow j}\} \quad (13)$$

Let us finally observe that, since in the assignment rule (8) negative messages play no role, one can obtain a totally analogous rule, making use of the ‘‘clipped’’ messages

$$\bar{\mu}_{j \rightarrow i} \triangleq \max\{0, \mu_{j \rightarrow i}\}$$

For these messages, the propagation equation (13) becomes

$$\bar{\mu}_{j \rightarrow i} = \max\{0, w_{ij} - \max_{k \in \partial j \setminus i} \bar{\mu}_{k \rightarrow j}\}$$

which, for a bipartite graph, coincides with the update rules described in the text.