

Network's Liquid Throughput: Topology Aware Optimization of Collective Communication

Emin Gabrielyan, Roger D Hersch
École Polytechnique Fédérale de Lausanne
{Emin.Gabrielyan,RD.Hersch}@epfl.ch

Abstract

The upper limit of the network's capacity for a collective communication is its liquid throughput related to the flow capacity of a liquid in a network of pipes. However, in a communication network the aggregate throughput of a traffic carried out by straight-forward topology-unaware techniques may be several times lower than the maximal potential throughput of the network. In most of the cut-through, wormhole and wavelength division optical networks, the loss of performance is caused by congestions between simultaneous transfers sharing a common link resource. We propose to carry out the transfers of the traffic according to special liquid schedules, which, relying on the knowledge of the underlying network topology and by ensuring successive utilization of bottleneck links, obtain the network's liquid throughput. To build a liquid schedule we need to partition and distribute the traffic into time-frames consisting of mutually non-congesting transfers and also keeping the bottleneck links occupied all the time. Time frames filled up until saturation by non-congesting transfers lead to an exponential explosion of many unsuccessful configurations, resulting in a very large search space. In this paper we present an efficient algorithm which non-redundantly traverses all possible subsets of simultaneous transfers for time-frames and, further on, a liquid schedule construction technique, which efficiently reduces the search space without affecting the solution space. There is a practical minority of collective communications yielding no liquid throughput for any resequencing of its transfers. The optimal or sub-optimal schedules for such communications we propose to obtain by applying one of the heuristic graph coloring algorithms.

Keywords: optimal network utilization, collective communication, liquid schedules, network topology, topology-aware scheduling.

1. Introduction

Collective multicast communications are of increasingly high importance in multimedia scientific applications. The aggregate throughput of a collective communication pattern depends on the application's underlying network topology. The amount of data that has to pass across the most loaded links of the network (bottleneck links) gives the utilization time of the bottleneck links. The total size of the traffic divided by the utilization time of the bottleneck links gives an estimation of the *liquid throughput* corresponding to the flow capacity of a non-compressible fluid in a network of pipes [Melamed]. In the wormhole electronic networks as well as in the WDM optical networks not any combination of transfer requests may be carried out simultaneously. For a physical network modelled as an undirected graph $G = (V(G), E(G))$ there is an objective to minimize the number T of timeslots and/or wavelengths required to carry out the given set of transfer requests. The proposed formulation is as follows:

Minimize: T

Subject to:

$$\sum_{s,d} A_t^{s,d} \cdot R_e^{s,d} \leq 1 \quad \forall e \in E(G), \forall t \in \{1 \dots T\}$$

$$F_e^{s,d} = 0, 1$$

$$A_t^{s,d} = 0, 1$$

Here R_e^{sd} denotes the routing, i.e. indicates if the transfer (flit stream flow for wormhole switching or lightpaths for optical networks) from source s to destination d traverses the link e . A_t^{sd} indicates if the transfer from source s to destination d is assigned to the timeslot t .

It remains to express the partitioning constraint of the schedule, i.e. each transfer of traffic must be assigned to one and only one timeslot:

$$\sum_{t=1}^T A_t^{s,d} = 1 \quad \forall s, d$$

The present problem is hard to solve. We will show that for the sizes of practical interest the application of the exact method MILP [CPLEX], [AMPL] (Mixed Integer Linear Programming) to the problem leads to very long solution times. Application of heuristic method of graph colouring algorithms efficiently providing a suboptimal solution is possible. However the suboptimal results of heuristic graph coloring method, translated back to the original problem in term of performance results to a loss of up to 18%. The exact method proposed in this paper, is fast enough to allow real time scheduling of an evolving over time broadcast and multicast communication in congestion sensible transmission networks. There are numerous applications requiring highly efficient network resources: parallel acquisition and distribution of multiple video streams [Chan], [Sitaram]; switching of simultaneous voice communication sessions [H323], [Fritz]; and high energy physics data acquisition and transmission from a large number of detectors to a cluster of processing nodes for data filtering and event assembling [CERN].

An applicaion of a particular interest is an optical network capable of the lightpath assignment on demand. Liquid scheduling is successfully tested on the cluster of parallel supercomputer interconnected by a high performance wormhole [Liu] switch fabric. A communication intensive application well integrating with the liquid schedules is a Striped File I/O¹ for parallel computation of out of core applications [Puente].

2. Applicable networks

This section introduces different types of electronic, optical, wireless networks having in common the problem of congestion avoidance in the traffic. Liquid scheduling of collective broadcast and multicast communication is applicable for each of these networks.

2.1. Cut-through switching

In many high performance multicomputer communication networks links lying on the path of a message are exclusively kept occupied during the transmission of that message. Unlike packet switching (or store-and-forward switching) with each network packet being solely present at intermediate router [Ayad],

wormhole switching (or cut-through switching) transmits a message as a worm propagating itself across intermediate switches, i.e. a continuous stream of bits which make their way through the fabric spanning multiple switches. In a cut-through switching [Duato], [Shin], [Rexford], [Colajanni] a message entering into a network is being broken up into small parts of equal size (e.g. one 32-bit word) called flits (standing from flow-control digit), which are streamed across the network. All the flits of a packet follow the same path. As soon as a switch on the path of a message receives the head flit and processes the routing header, it triggers the flow of flits to the corresponding outgoing link. If the message encounters a busy outgoing link, the wormhole switch stalls the message in the network along the already established path until the link becomes available². Occupied channels are not released. A channel is released only when the last tail flit of the message is transmitted through it. Thus each link laying on the path of the message is kept occupied during the whole transmission time of a message.

Compared with store and forward switches, cut-through switching considerably decreases the latency of message transmission across multiple routers. Cut-through switching makes the latency insensitive to the message distance. Therefore, most contemporary research and commercial multicomputer routers use some form of cut-through switching (Myrinet, Quadrics, Tnet) [Boden].

Wormhole switching only pipelines message and thus requires no more than a very small buffer. This makes easier realization of a high performance switch on a single chip considerably reducing the cost of a large scale network [Yocum]. However, wormhole switching alone, quickly saturates as load increases due to blocked message paths. Congestions occurring due to simultaneous transmission of messages sharing common network links result in an aggregate data throughput lower than the liquid throughput offered by the network.

For the same set of collective communications the rate of network congestions may significantly vary depending in which order individual communications are carried out. Channel contentions can be avoided if the transfers are scheduled so that no congesting messages are transmitted at a time.

1.2. Lightpaths on demand

Lightpaths are end to end optical connections from a source node to a destination node over a wavelength within each intermediate link. Different lightpaths in a wavelength

1. Striped File I/O for parallel programs was our first application which indicated the importance of the knowledge of the underlying network topology by the parallel application.

2. If the message encounters a busy outgoing link virtual cut-through (VCT) switching buffers the entire packet in the router.

routing network can use the same wavelength as long as they do not share any common links. Fig. 1 shows an example of optical network.

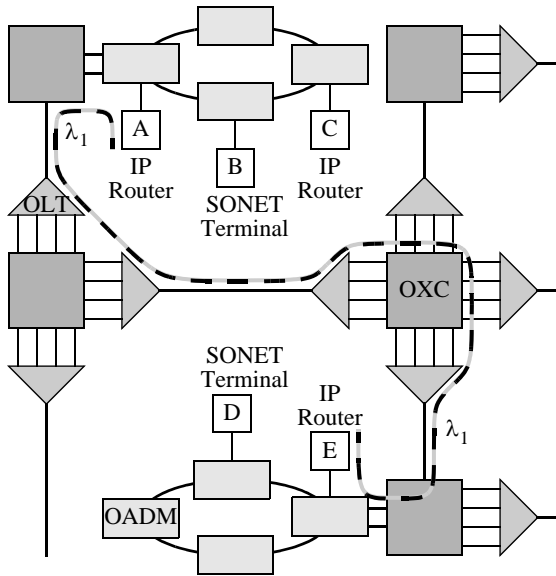


Fig. 1. Optical layer, wavelength-routing network.

OLT, Optical Line Terminal, multiplexes multiple wavelengths into a single fiber and demultiplexes a set of wavelengths on a single fiber into separate fibers; OADM is an Optical Add/Drop Multiplexer; and OXC, Optical Cross-connect, switches wavelengths from one port to another.

End nodes are IP routers and SONET terminals. The network is designed to provide permanent lightpaths between terminal nodes, e.g. between the node A and E. Both OADMs and OXCs in the network may incorporate wavelength conversion capabilities. OXCs that provides no wavelength conversion features are called wavelength-selective cross-connects (WSXC). In networks uniquely based on WSXC optical switches, it is assumed that the basic optical transmission channel remains on a fixed wavelength from end to end (a condition called wavelength continuity constraint) and therefore any lightpath must be assigned the same wavelength on all the links it traverses (two lightpaths traversing a common link must be assigned different wavelengths [Ramaswami]). Assuming that the OXCs of the example in the Fig. 1 do not carry out wavelength conversion, the lightpath between the nodes A and E uses one single wavelength λ_1 all along its route and therefore λ_1 cannot be reused by another lightpath at any link of the route¹.

In most cases we are already given the routing, in which case, we are concerned only with the Wavelength Assignment (WA) problem. WA is a problem of optimal design in the optical layer. There has been work on

theoretical considerations about the required number of wavelengths and the complexity of finding a solution according to the network topology [Bermond], [Caragiannis].

Under wavelength continuity condition, when the network is based on WSXC switches, the problem of wavelength assignment to the set of lightpaths and the problem of the timeslot assignment to the set of transmissions are very similar.

Additional interest for liquid scheduling is found in optical transmission systems deploying the capability of dynamic switching of lightpath circuits. The functionality providing the ability to set up and take down lightpaths across the network in a dynamic fashion is currently evolving in the optical layer very fast [Stern]. An issue in such a network providing lightpaths on demand is a fast and efficient scheduling of multicast communications within minimal number of timeframes.

1.3. Time division networks and spread spectrum wireless networks (CDMA).

Satellite-switch time division multiplexing networks [Jain] and CDMA spread spectrum wireless networks [Battiti] also can be viewed as interconnection topologies. The links of the interconnection topology reflecting CDMA networks represent orthogonal frequency spectra. Congestion occurs if there is a common frequency for two simultaneous transmissions. The liquid throughput may be obtained for a traffic over a CDMA wireless network by applying our algorithms to the same radio-communication traffic, but carried out over a virtual interconnection topology reflecting the configuration of the wireless network. The obtained distribution of radio transmissions over successive time frames will yield the liquid throughput of the wireless network.

We propose a dynamic scheduling of multicast communications, where the application is capable of determining its underlying topology and apply liquid schedules to its data intensive collective communication requests. Unlike flow control based congestion avoidance mechanisms [Chiu], [Ozbay], [Loh] we establish schedules for the data transfers without trying to regulate the sending nodes' data rate. Liquid schedules do not always exists, in that cases we turn to optimal or sub-optimal solutions

1. In some cases, lightpaths may be converted from one wavelength to another wavelength along their route. OXCs providing the wavelength conversion are called wavelength-interchanging cross-connects (WIXC). WIXCs do both space switching and wavelength conversion.

offered by one of efficient heuristic algorithms of the graph colouring problem.

We integrated implementations of the presented algorithm as a library with an MPI communication intensive application running on a cut-through high performance network. The obtained measurement results are very close to the expected theoretical values.

2. Examples and demonstrations

The example of network topology shown in Fig. 2 consists of ten end nodes $t1...t5$, $r1...r5$ (henceforth called processors), two wormhole cut-through switches s_a , s_b and twelve unidirectional links $l_{t1}...l_{t5}$, $l_{r1}...l_{r5}$, l_{ab} , l_{ba} having identical throughputs. In this example the processors $t1...t5$ only transmit data and $r1...r5$ only receive. It's easy to guess the routing, e.g. a message from $t4$ to $r3$ traverse links l_{t4} , l_{ba} and l_{r3} , a message from $t2$ to $r3$ uses only links l_{t2} and l_{r3} , etc.

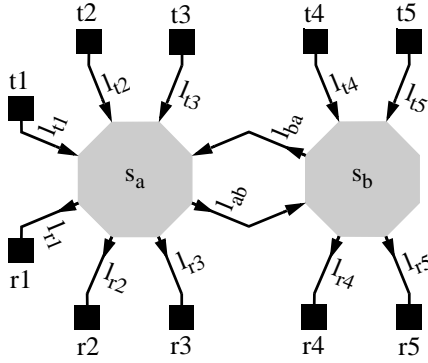













Fig. 2. Example of a network topology.

We denote transfers symbolically to mark out the network links occupied by the transfer. For example the transfer from $t4$ to $r3$ is symbolically represented as , the transfer from $t1$ to $r2$ as , etc. Extending the graphical representation a set of transfers carried out simultaneously we also denote symbolically, e.g.  corresponds to a traffic simultaneously transferring messages from $t4$ to $r3$ and from $t1$ to $r2$.

Let each sending processor have a messages destined to each receiving processor and let all messages have identical sizes [Naghshineh]. Thus we have 25 transfers to carry out. Each of the ten links $l_{t1}...l_{t5}$, $l_{r1}...l_{r5}$ carries 5 transfers and the two links l_{ab} , l_{ba} must carry 6 transfers each. Therefore the links l_{ab} , l_{ba} are the network bottlenecks and have the longest active time. If the duration of whole collective communication is as long as the active time of the bottleneck links, we say that the collective

communication reaches its liquid throughput. In that case the bottleneck links are obviously kept busy all the time along the duration of the communication traffic. Assuming in this example a single link throughput 1 Gb/s, the liquid throughput offered by the network is $(25/6) \times 1 \text{ Gb/s} = 4.17 \text{ Gb/s}$. Under the identical transfer size and link throughput constraints (kept all along this paper for the sake of simplicity) the *liquid throughput* of a traffic X is the ratio $\#(X)/\Lambda(X)$ multiplied by single link throughput, where $\#(X)$ is the total number of transfers and $\Lambda(X)$ is the number of transfers carried by one bottleneck link.

Now let us see if the order in which the transfers are carried out in this wormhole network can have any impact to the collective communication performance. A straight forward schedule to carry out these 25 transfers is the round-robin schedule, according to which at first each transmitting processor sends the message to the receiving processor staying in front, then to the receiving processor staying in the next position, etc. (the order turns from the last fifth processor to the first one). This round robin schedule consists of 5 phases. The transfers of the first  second  and fifth  phases of the round-robin schedule may be carried out simultaneously, but the third  and fourth  phases contain congesting transfers, e.g. at the third phase the transfer  stays blocked until the transmission  is accomplished (or vice a versa). None of these two phases can be carried out in less than two time-frames and therefore the whole schedule lasts 7 time-frames, instead of simply 5. Consecutively the performance of our collective communication carried out according to the round-robin schedule corresponds to the throughput of $25/7 = 3.57$ messages per time-frame or $(25/7) \times 1 \text{ Gb/s} = 3.57 \text{ Gb/s}$, which is less than liquid throughput.

Nevertheless, the 25 transfers may be carried out within 6 time-frames. We call a *liquid schedule* the schedule yielding the liquid throughput of the collective communication. The following sequence of time-frames  is an example of the liquid schedule for the 25-transfer collective communication request.

3. Definitions

The method we propose allows efficient construction of liquid schedules for complex network topologies resulting in considerable increase of collective data exchange throughputs, compared with traditional topology-unaware techniques such as round-robin or random schedules. This section introduces the definitions that will be further used in this paper for presentation of the underlying algorithms of the construction method.

A single “point-to-point” transfer is represented by the set of communication links forming the network path between a transmitting and a receiving processor according to a given routing. A *transfer* is a set of links (i.e. the path between a sending processor and a receiving processor). A *traffic* is a set of transfers (i.e. the collective data exchange). Fig. 3 shows the traffic for a data exchange carried out on a network topology shown in the Fig. 2. In the figure the bottleneck links of the network are marked in bold.

$$\left\{ \begin{array}{l} \{l_{11}, l_{r1}\}, \{l_{11}, l_{r2}\}, \{l_{11}, l_{r3}\}, \{l_{11}, \mathbf{l_{ab}}, l_{r4}\}, \{l_{11}, \mathbf{l_{ab}}, l_{r5}\}, \\ \{l_{12}, l_{r1}\}, \{l_{12}, l_{r2}\}, \{l_{12}, l_{r3}\}, \{l_{12}, \mathbf{l_{ab}}, l_{r4}\}, \{l_{12}, \mathbf{l_{ab}}, l_{r5}\}, \\ \{l_{13}, l_{r1}\}, \{l_{13}, l_{r2}\}, \{l_{13}, l_{r3}\}, \{l_{13}, \mathbf{l_{ab}}, l_{r4}\}, \{l_{13}, \mathbf{l_{ab}}, l_{r5}\}, \\ \{l_{14}, \mathbf{l_{ba}}, l_{r1}\}, \{l_{14}, \mathbf{l_{ba}}, l_{r2}\}, \{l_{14}, \mathbf{l_{ba}}, l_{r3}\}, \{l_{14}, l_{r4}\}, \{l_{14}, l_{r5}\}, \\ \{l_{15}, \mathbf{l_{ba}}, l_{r1}\}, \{l_{15}, \mathbf{l_{ba}}, l_{r2}\}, \{l_{15}, \mathbf{l_{ba}}, l_{r3}\}, \{l_{15}, l_{r4}\}, \{l_{15}, l_{r5}\} \end{array} \right\}$$

Fig. 3. Example of traffic composed of 25 transfers carried out over a network shown on Fig. 2.

This half-all-to-all data exchange is a particular case of a traffic, any collective exchange comprising of transfers between possibly overlapping sets of sending and receiving processors is a traffic. A link l is *utilized* by a transfer x if $l \in x$. A link l is utilized by a traffic X if l is utilized by a transfer of X . Two transfers are in *congestion* if they share a common link. If they don't use a common link they are *simultaneous*. Note that we will be limiting ourselves to data exchanges consisting of identical packet sizes.

A *simultaneity* of a traffic X is a subset of X consisting of mutually non-congesting transfers. A transfer is in congestion with a simultaneity if the transfer is in congestion with at least one member of the simultaneity. A simultaneity of a traffic is *full* if all transfers in the complement of the simultaneity in the traffic are in congestion with that simultaneity. A simultaneity of a traffic is obviously can be carried out in one timeframe required by a single transfer. $\lambda(l, X)$, the *load* of link l in the traffic X , is the number of transfers in X using link l . Now comes the formal definition of $\Lambda(X)$ already introduced before. The *duration* $\Lambda(X)$ of a traffic X is the maximal value of the load among all links involved in the traffic.

$$\Lambda(X) = \max_{\left\{ l \in \bigcup_{x \in X} \left(\bigcup_{l \in x} l \right) \right\}} \lambda(l, X)$$

The links having maximal load values, i.e. $\lambda(l, X) = \Lambda(X)$, are called *bottlenecks*. The *liquid throughput* of a traffic X is the ratio $\#(X)/\Lambda(X)$ multiplied by the single link throughput, where $\#(X)$ is the number of transfers in the traffic X .

Let us define a simultaneity of X as a *team* of X if it uses all bottlenecks of X . A team of X is *full* if it is a full simultaneity of X . Let $\mathfrak{R}(X)$ and $\mathfrak{T}(X)$ be respectively the sets of all full simultaneities and all full teams of X .

In order to form liquid schedules, we try to schedule transfers in such a way that all bottleneck links are always kept busy. Therefore we search for a liquid schedule by trying to assemble non-overlapping teams carrying out all transfers of the given traffic (i.e. partitioning of the traffic into teams [Halmos]). To cover the whole solution space we need means of generating all possible teams of a given traffic. This is an exponentially complex problem. It is therefore important that the team traversing technique be non-redundant and efficient, i.e. each configuration is evaluated once and only once, without repetitions.

4. Obtaining full simultaneities

The construction of liquid schedules requires the ability of traversing the set of all full teams of an arbitrary traffic. To limit redundant search steps, each full team should be constructed once and only once. We first optimize the retrieval of all simultaneities and then use that algorithm to retrieve all full teams.

Recall that in a traffic X , any mutually non-congesting combination of transfers is a simultaneity. A full simultaneity is a combination of non-congesting transfers taken from X , such that its complement in X contains only transfers congesting with that simultaneity.

We can categorize full simultaneities according to the presence or absence of a given transfer x . A full simultaneity is x -positive if it contains transfer x . If it does not contain transfer x , it is x -negative. Thus the set of full simultaneities $\mathfrak{R}(X)$ is partitioned into two non-overlapping subsets: an x -positive and x -negative subset of $\mathfrak{R}(X)$. For example, if y is another transfer, the set of x -positive full simultaneities may be further partitioned into y -positive and y -negative subsets. Repetition of this concept allows us to design a recursive technique

traversing whole set of all full simultaneities $\mathfrak{R}(X)$ one by one without repetitions.

Let us define a *category* of full simultaneities of X as an ordered triplet $(excluder, depot, includer)$, where the includer is a simultaneity of X (not necessarily full) and the transfers of X non-congesting with the includer are either in the depot or in the excluder.

We say that a full simultaneity is *covered* by a category R , if the full simultaneity contains all the transfers of the category's includer and does not contain any transfer of the category's excluder. Consequently, any full simultaneity covered by a category is the category's includer together with some transfers taken from the category's depot. The collection of all full simultaneities of X covered by a category R is defined as the *coverage* of R . We denote the coverage of R as $\phi(R)$.

The category $(\emptyset, X, \emptyset)$ is a *prim-category* since it covers all full simultaneities of X , i.e. $\phi(\emptyset, X, \emptyset) = \mathfrak{R}(X)$.

By taking an arbitrary transfer x from the depot of a category R , we partition the coverage of R into x -positive and x -negative subsets. The x -positive and x -negative subsets of a coverage of R respectively are coverages of two categories derived from R : a positive sub-category and a negative sub-category of R .

The positive sub-category R_{+x} is formed from the category R by adding transfer x to its includer, and removing from its depot and excluder¹ all transfers congesting with x . The negative sub-category R_{-x} is formed from the category R by moving transfer x from its depot to its excluder (see Fig. 4).

-
1. Since transfers congesting with x can not be in a full simultaneity covered by R_{+x} , we may safely remove them from the excluder.

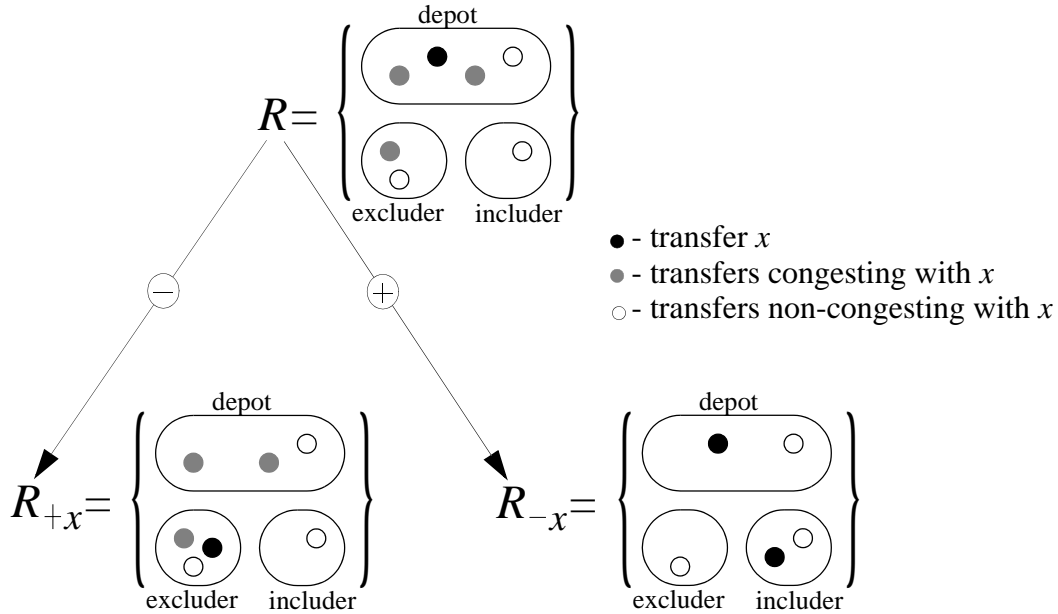


Fig. 4. Fission of a category into two sub-categories

The coverage of R is partitioned by the coverages of its sub-categories R_{+x} and R_{-x} , i.e. the coverage of a category is the union of coverages of its sub-categories:

$\phi(R_{+x}) \cup \phi(R_{-x}) = \phi(R)$, where the coverages of the sub-categories have no common transfers, $\phi(R_{+x}) \cap \phi(R_{-x}) = \emptyset$. The replacement of a category R by its two sub-categories R_{+x} and R_{-x} is defined as a *binary fission* of a category.

A *singular* category is a category that covers only one full simultaneity. That full simultaneity is equal to the includer of the singular category. The depot and excluder of a singular category are empty.

We apply the binary fission to the prim-category and split it into two categories. Then, we apply the fission to each of these categories. Repeated fission increases the number of categories and narrows the coverage of each category. Finally, the fission will lead to singular categories only, i.e. categories whose coverage consists of a single full simultaneity. Since at each stage we have been partitioning the set of full simultaneities, at the final stage we know that each full simultaneity is covered by one and only one singular category.

The algorithm carries out recursively the fission of categories and yields all full simultaneities without repetitions.

There is a further optimization to be considered. Full simultaneities covered by a category have no transfer from the category's excluder. Therefore each full simultaneity covered by a category must contain a congesting transfer for each member of the excluder. Since we keep in the excluder transfers which do not congest with the includer, congesting transfers must be taken from the depot. A category whose depot doesn't have a congesting transfer for at least one of the excluder's transfers is *blank*. The coverage of a blank category is empty and there is therefore no need to pursue its fission.

Let a category within X be *idle* if its includer and its depot together don't use all bottlenecks of X . The coverage of an idle category does therefore not contain a team.

An algorithm that is carrying out successive fissions, starting from the prim-ancestor and contiguously removing all the blank and idle categories ultimately leads to all full teams.

5. Speeding up full team formation

This section presents a further method for speeding up the search for all full teams $\mathfrak{T}(X)$ of an arbitrary traffic X .

Let us consider from the original traffic X only those transfers that use bottlenecks of X and call this set of transfers *skeleton* of X . We denote the skeleton of X as $\zeta(X)$. Obviously, $\zeta(X) \subset X$.

Considering the skeleton of a traffic X as another traffic, the bottlenecks of the skeleton of a traffic are the same as the bottlenecks of the traffic. Consequently, a team of a skeleton is also a team of the original traffic.

Let us obtain all full teams of the traffic's skeleton by applying the fission algorithm eliminating the idle categories.

Then, a full team of the original traffic may be obtained by adding a combination of non-congesting transfers to a team of the traffic's skeleton.

We therefore obtain the set of a traffic's full teams

$\mathfrak{T}(X)$, by carrying out the following steps:

1. Obtain the set of the skeleton's full teams $\mathfrak{T}(\zeta(X))$ by applying the fission algorithm.
2. Create for each skeleton's full team a category by
 - 2.1. initializing the includer with the transfers of the skeleton's full team,
 - 2.3. initializing the excluder as empty,
 - 2.2. and putting into the depot all transfers of X non-congesting with the includer.
3. Apply the fission to each category, discarding the check for idle categories, since the includer is already a team, i.e. it uses all bottlenecks.

By first applying the fission to the skeleton and then expanding the skeleton's full teams to the traffic's full teams, we strongly reduce the required processing time and at the same time we obtain all full teams of the original traffic without repetitions.

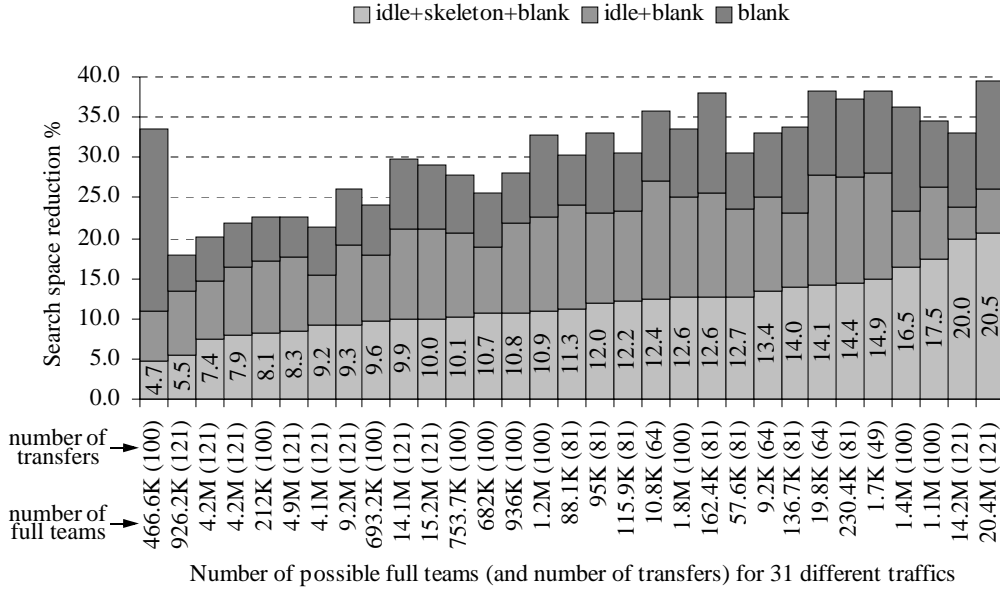


Fig. 5. Search space reduction

We measured the reduction in search space according to the different search space reduction methods we propose. We consider 31 different traffics within the T1 32 node cluster computer (Fig. 8). The search space is given by the number of nodes that are being explored within the recursion tree. Fig. 5 shows the obtained search space reductions compared with a naive algorithm that would build full teams without any of the proposed optimizations. The skeleton algorithm reduces on average the search space to 12.48%, i.e. full teams are computed 8 times faster than without search space reduction techniques. Note that all presented algorithms, including the naive algorithm, are smart enough to avoid repetitions of full simultaneities.

6. Liquid schedules

Having the capability of building full teams, let us now show how to compute the liquid schedule of a data exchange. A *schedule* α of a traffic X is a collection of simultaneities of X partitioning the traffic X . A *step* of a schedule α is an element of the schedule α . $\#(\alpha)$, the *length* of a schedule α , is the number of steps in α . A schedule of a traffic is *optimal* if the traffic does not have any shorter schedule. If the length of a schedule is equal to the duration of the traffic then the schedule is *liquid*. A liquid schedule is optimal, but the inverse is not always true, meaning that a traffic may not have a liquid schedule. Fig. 6. demonstrates a traffic that does not have a team and

therefore no liquid schedule. Fig. 7 shows a liquid schedule for the collective traffic shown in Fig 1.

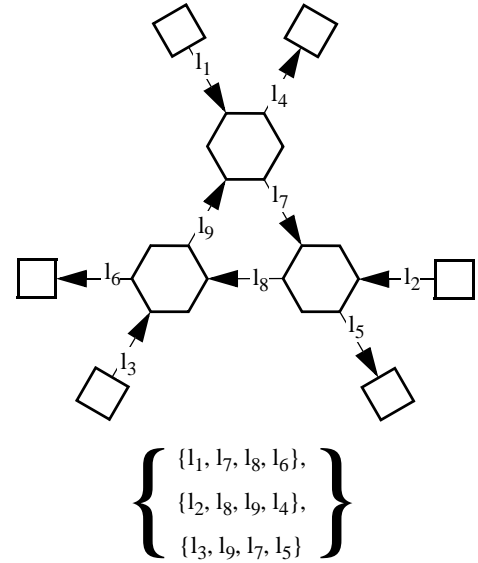


Fig. 6. This traffic has no team and no liquid schedule.

The duration of a traffic X is the load of its bottlenecks. If a schedule is liquid, then each of its timeframes must use all bottlenecks. Inversely, if all steps of a schedule use all bottlenecks, the schedule is liquid.

$$\left(\begin{array}{c} \left\{ \begin{array}{c} \{1_1, \mathbf{l}_{12}, l_9\}, \\ \{2, l_7\}, \\ \{3, l_8\}, \\ \{4, \mathbf{l}_{11}, l_6\}, \\ \{5, l_{10}\} \end{array} \right\} \\ \left\{ \begin{array}{c} \{1_1, \mathbf{l}_{12}, l_{10}\}, \\ \{2, l_6\}, \\ \{4, \mathbf{l}_{11}, l_7\}, \\ \{5, l_9\} \end{array} \right\} \\ \left\{ \begin{array}{c} \{1_1, l_8\}, \\ \{2, \mathbf{l}_{12}, l_9\}, \\ \{3, l_6\}, \\ \{4, l_{10}\}, \\ \{5, \mathbf{l}_{11}, l_7\} \end{array} \right\} \end{array} \right) \\ \left(\begin{array}{c} \left\{ \begin{array}{c} \{1_1, l_7\}, \\ \{2, \mathbf{l}_8\}, \\ \{3, \mathbf{l}_{12}, l_9\}, \\ \{5, \mathbf{l}_{11}, l_6\} \end{array} \right\} \\ \left\{ \begin{array}{c} \{1_1, l_6\}, \\ \{2, \mathbf{l}_{12}, l_{10}\}, \\ \{3, l_7\}, \\ \{4, \mathbf{l}_{11}, l_8\} \end{array} \right\} \\ \left\{ \begin{array}{c} \{3, \mathbf{l}_{12}, l_{10}\}, \\ \{4, l_9\}, \\ \{5, \mathbf{l}_{11}, l_8\} \end{array} \right\} \end{array} \right)$$

Fig. 7. A liquid schedule of the collective traffic shown in Fig. 2.

The necessary and sufficient condition for the liquidity of a schedule is that all bottlenecks be used by each step of the schedule. Since a simultaneity of X is defined as a *team* of X , if it uses all bottlenecks of X , an equivalent condition for the liquidity of a schedule α on X is that each step of α be a team of X .

6.1. Liquid schedule naive search algorithm

Let us propose a naive liquid schedule construction technique. Consider all possible teams of the original traffic X . Take one of them (for example A_1) and consider it as the first step of the liquid schedule. Remove the team A_1 from the traffic and look at the reduced traffic. The choice for the second step is limited by only those teams of the original traffic X , which are included in the reduced traffic $X - A_1$. Take a candidate for the second step from the current choice (for example $A_{1,1}$). Remove from the traffic the team $A_{1,1}$ as well. Similarly the choice for the third step is limited by only those teams of the original traffic X , which are included in the reduced traffic $X - A_1 - A_{1,1}$, etc.

The algorithm recursively search a liquid schedule in the depth-wise order. A node in the search tree represents a dead end if there are no choice for the successive step, i.e. no team of the original traffic may be formed from the reduced traffic (i.e. not yet carried out transfers). At the dead end nodes the algorithm backtracks one or more steps back and analyses other possibilities. The algorithm stops at the node whose reduced traffic is a team of the original traffic. The collection of all teams on the path from the root to that terminal node is a liquid schedule.

6.2. Speeding up the search of a liquid schedule

Let's analyse the liquid schedule shown in Fig. 7. Remove from the traffic a few steps of the schedule and look at the reduced traffic.

Load of bottlenecks decreases in the reduced traffic. However the reduced traffic may contain additional bottlenecks. More steps of a liquid schedule are carried out more additional bottlenecks appears in the reduced traffic (not yet carried out transfers).

The construction strategy of a liquid schedule presented in the subsection 4.1. form a choice of candidates to the successive step from all teams of the original traffic X included in the reduced traffic.

However note, that the steps of the liquid schedule, are not only teams for the original traffic, but they are also teams of the corresponding reduced traffics. We are going to prove that this property of steps is a necessary and sufficient condition of the liquidity of a schedule. Note that this property is valid independently from the order of steps.

Therefore a step which is a team of the original traffic (uses the bottlenecks of the original traffic) but does not use the additional bottlenecks of the reduced traffic (isn't a team of the reduced traffic) may not lead to a liquid schedule. Such a choice ultimately brings to a dead end and algorithm shall backtrack for evaluating other choices.

Since the set of bottlenecks in the reduced traffic is larger than the set of bottlenecks of the original traffic, the number of teams of the original traffic is much fewer than the number of teams of the original traffic (included in the reduced traffic). Therefore by limiting our choice at each step by the set of teams of the reduced traffic we considerably reduce the search space without affecting the solution space.

DISCUSSION. Suppose A is a timeframe of a liquid schedule α on a traffic X . Therefore A is a team of α . Remove the team A from X so as to form a new traffic $X - A$. The duration of the new traffic $X - A$ is the load of the bottlenecks in $X - A$. The bottlenecks of X are also the bottlenecks of $X - A$. The load of a bottleneck of X decreases by one in the new traffic $X - A$ (note that the new traffic $X - A$ may have additional bottlenecks). The schedule α shortened by one element A is a schedule for $X - A$. The new schedule $\alpha - \{A\}$ has as many

timeframes as the duration of the corresponding new traffic $X - A$. Therefore, if α is a liquid schedule on X then for any of its step A the schedule $\alpha - \{A\}$ is a liquid schedule on $X - A$.

Consider traffic X as a problem whose solution is a liquid schedule α . The technique presented in section 3, is capable of generating the set of all teams of X . If X has a solution α then a timeframe A of the schedule α is a member of the set of all teams of X and $\alpha - \{A\}$ is a schedule on $X - A$. Therefore the problem X can be reduced into smaller problems. Examine each possible team A of X and search inductively (e.g. recursively) a solution for $X - A$. If a solution exists for X , then this method will find it. If the method does not find a solution for X , then, since we explored the full solution space, we conclude that X does not have a liquid schedule.

We limit at each iteration our choice to the collection of only those teams of the original traffic which are also teams of the current reduced sub-traffic (having an expanded number of bottlenecks). By doing so, we considerably reduce the search space without affecting the solution space.

By limiting the choice of the next step only by full teams of the reduced traffic we again reduce the search space of the construction algorithm. Let us show that here the solution space is not affected as well. Let us modify a liquid schedule so as to convert one of its teams into a full team. Let X (a traffic) have a solution α (a liquid schedule). Let A be a timeframe of α . If A is not a full team of X , then, by moving the necessary transfers from other timeframes of α , we can convert timeframe A to a full team. Evidently, the properties of liquidity (partitioning, simultaneousness and length) of α will not be affected. Therefore if X has a solution then it has also a solution when the team of one of its steps is full, hence the choice of the teams in the construction may be narrowed from the set of all teams to the set of full teams only.

By a choice of a full team A of a traffic X we are faced with the new smaller problem of searching a liquid schedule for a traffic $X - A$. The traffic $X - A$ may not have a solution, or it may not have even a team. In these cases we have to backtrack to evaluate other choices.

Evaluation of all choices ultimately leads to a solution if it exists.

Fig. 7 shows a liquid schedule built as explained above. Each its successive step being a team of the reduced traffic incorporates all bottlenecks of that reduced traffic (shown in bold).

Thanks to the presented chain of optimizations, for more than 90 percent of our testbed topologies (see Section 5) the search of liquid schedules took less than one tenth of a second on a single 500MHz processor. For 8 topologies out of 363 solution was not found within 24 hours.

7. Testbed and measurements

In this section we present a testbed consisting of test traffics for different topologies. Measurements of collective data exchange throughputs will help us to validate the efficiency of our scheduling strategy.

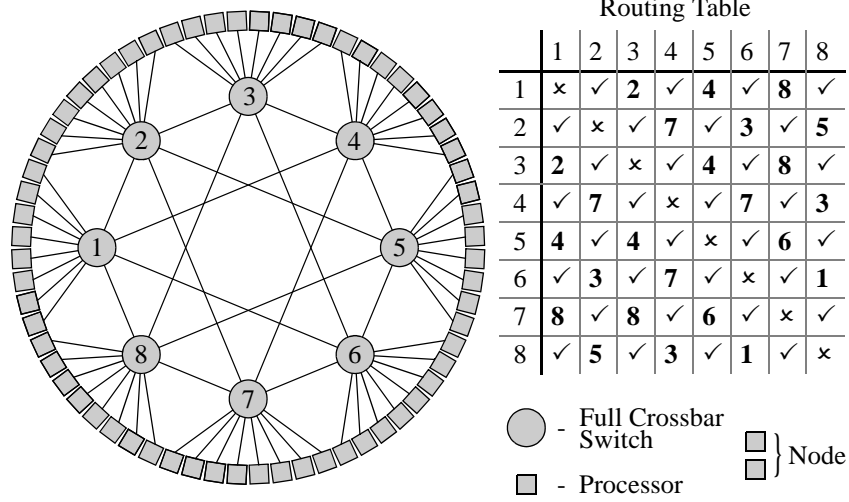


Fig. 8. Architecture of the T1 cluster computer.

As basic network topology for our testbed, we use the Swiss-T1 cluster (called henceforth T1, see Fig. 8). The network of the T1 forms a K-ring [Kuonen], [Sayoud] and has a static routing scheme. The throughputs of all links are identical and equal to 86MB/s . The cluster consists of 32 nodes, each one comprising 2 processors [SwissT1], [Gruber].

The test traffics are selected from different configurations of all-to-all collective data exchanges between a set of emitting and receiving processors, where each emitting processor sends one packet to each receiving processor. Within each node, one processor is an emitting and the other processor is a receiving unit. Therefore any given allocation of nodes gives us an equal number of emitting and receiving processors.

Since the T1 cluster incorporates 32 nodes, there exist

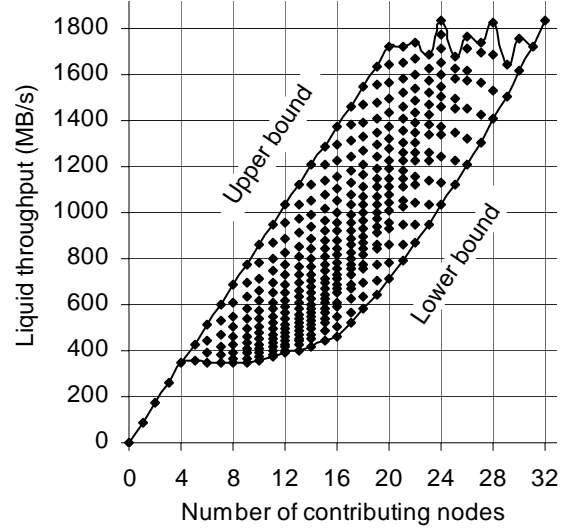


Fig 9. Liquid throughput in relation to the number of nodes with variations according to sub-topologies.

$2^{32} = 4294967296$ possible allocations of nodes to an application. Considering only the number of nodes in front of each switch, there are only $5^8 = 390625$ different node allocations, since there are 8 switches having each n used nodes ($0 \leq n \leq 4$). Because of symmetries within the network, many of these topologies are identical. To limit our choice to really different topologies, we've computed the liquid throughputs for each of 390625 topologies, taking in account the network's real routing tables. This resulted in only 363 different liquid throughput values. Accordingly,

we have formed a test-bed consisting of 363 really different topologies. Each topology is characterized by its liquid throughput and the number of allocated nodes (see Fig. 9)¹.

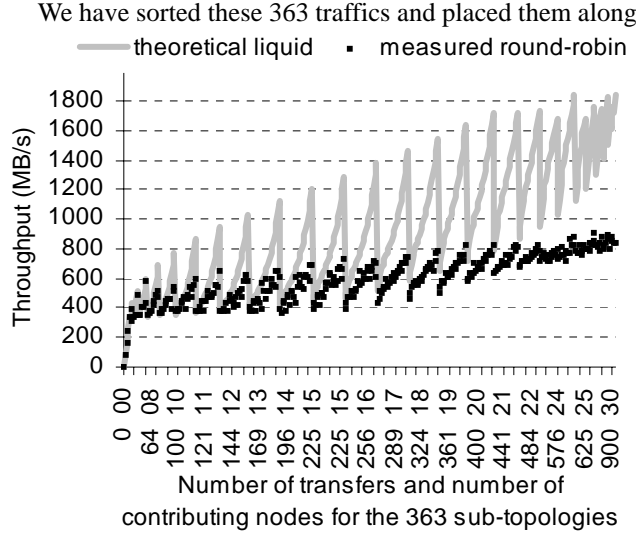


Fig. 10. Theoretical liquid throughput and measured round-robin schedule throughput for 363 network sub-topologies.

an axis. They are sorted first by the number of nodes and then according to the value of the liquid throughput. Fig. 10 shows the liquid throughput values together with the measured throughput of a round-robin schedule.

For each measurement, the amount of data transferred from a transmitting processor to a receiving processor is equal to *2MB*. For each topology, 20 measurements were made. The black dots represent the median of the collected results. Measured throughputs of the round-robin schedule are far below the network’s potential liquid throughput. Throughputs of collective exchanges carried out according to a random schedule do not perform better.

¹The figure demonstrates that depending on the sub-topology, the liquid throughput for a given number of nodes may considerably vary.

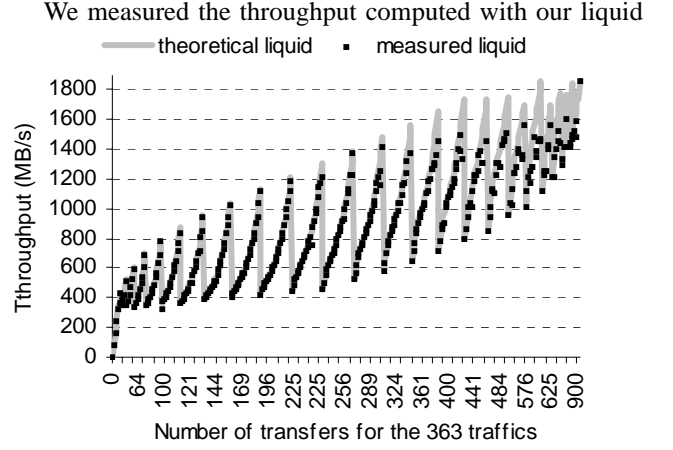


Fig. 11. Measured throughputs carried out according to computed liquid schedules.

scheduling technique, for the 363 data exchanges on the T1 cluster. Fig. 11 shows the measured throughput compared with the theoretical liquid throughputs.

Each black dot represents the median of 7 measurements. Processor to processor transfers have a size of *5MB*. The measured throughputs are close to the theoretically computed liquid throughput. For many sub-topologies, the proposed scheduling technique allows to increase the aggregate throughput by a factor of two compared with a simple, topology un-aware scheduling technique.

Thanks to the proposed liquid schedule search space reduction techniques, computing a liquid schedule takes for more than 97% of the considered sub-topologies of the T1 cluster takes a fraction of milliseconds on a simple computer.

8. Conclusion

We propose a method for scheduling collective data exchanges in order to obtain an aggregate throughput equal to the network’s liquid throughput. This is achieved by building at each step of the schedule mutually non-congesting sets of transfers using all bottleneck links. Exploration of the full solution space yields a liquid schedule if it exists. The proposed search space reduction techniques make the approach practical for networks having in the order of ten interconnected crossbar switches.

On the Swiss T1 cluster computer, the proposed scheduling technique allows for many sub-topologies to increase the collective data exchange throughput by a factor of two.

In the future, we intend to explore how to extend the presented scheduling technique in order to dynamically reschedule collective data exchanges when the set of planned exchanges evolves over time.

References

- [Melamed] Benjamin Melamed, Khosrow Sohraby, Yorai Wardi, "Measurement-Based Hybrid Fluid-Flow Models for Fast Multi-Scale Simulation", DARPA/NMS BAA 00-18 AGREEMENT No. F30602-00-2-0556, <http://www.darpa.mil/ito/research/nms/meetings/nms2001apr/Rutgers-SD.pdf>
- [CPLEX] ILOG-CPLEX 7.1 User's Manual
- [AMPL] AMPL: A Modeling Language for Mathematical Programming
- [Chan] S.-H.G. Chan, "Operation and cost optimization of a distributed server architecture for on-demand video services", IEEE Communications Letters, Vol. 5, No. 9, Sept. 2001, 384-386.
- [Sitaram] Dinkar Sitaram, Asit Dan, *Multimedia Servers*, Morgan Kaufmann Publishers, San Francisco California, ISBN 1-55860-430-8, 2000, 69-73.
- [H323] H.323 Standards, <http://www.openh323.org/standards.html>
- [Fritz] D.A. Fritz, D.W. Moy, R.A. Nichols, "Modeling and simulation of Advanced EHF efficiency enhancements", Proc. of Military Communications Conference, IEEE MILCOM 1999, Vol. 1, 354-358.
- [CERN] Large Hadron Collider, Computer Grid project, CERN, 20.09.2001, <http://press.web.cern.ch/Press/Releases01/PR10.01EGoaheadGrid>.
- [Liu] Pangfeng Liu, Jan-Jan Wu, Yi-Fang Lin, Shih-Hsien Yeh, "A simple incremental network topology for wormhole switch-based networks", Proc. 15th International Parallel and Distributed Processing Symposium, 2001, 6-12.
- [Puente] V. Puente, C. Izu, J. A. Gregorio, R. Beivide, J. M. Prellero, F. Vallejo, "Improving parallel system performance by changing the arrangement of the network links", Proc. of the International Conference on Supercomputing, May 2000, 44-53.
- [Ayad] N.M.A. Ayad, F.A. Mohamed, "Performance analysis of a cut-through vs. packet-switching techniques", Proc. Second IEEE Symposium on Computers and Communications, 1997, 230-234.
- [Duato] J. Duato, A. Robles, F. Silla, R. Beivide, "A comparison of router architectures for virtual cut-through and wormhole switching in a NOW environment", IEEE Symposium on Parallel and Distributed Processing SPDP, 1999, 240-247
- [Shin] K.G. Shin, S.W. Daniel "Analysis and implementation of hybrid switching", IEEE Transactions on Computers, Vol. 45 Issue 6, June 1996, 684-692
- [Rexford] Jenifer Rexford, Kang G. Shin, "Analytical Modeling of Routing Algorithms in Virtual Cut-Through Networks"
- [Colajanni] M. Colajanni, B. Ciciani, F. Quaglia, "Performance Analysis of Wormhole Switching with Adaptive Routing in a Two-Dimensional Torus"
- [Boden] N.J. Boden, et al., "Myrinet - A gigabit per second local area network," *IEEE Micro*, pp. 29-36, February 1995.
- [Yocum] K.G. Yocum, J.S. Chase, A.J. Gallatin, A.R. Lebeck, "Cut-through delivery in Trapeze: An Exercise in Low-Latency Messaging", 6th IEEE International Symposium on High Performance Distributed Computing, 1997, 243-252.
- [Ramaswami] R. Ramaswami, G. Sasaki, "Multiwavelength optical networks with limited wavelength conversion", Proc. of IEEE Infocom, 1997.
- [Bermond] J.-C. Bermond, L. Gargano, S. Perennes, A. A. Rescigno, and U. Vaccaro, "Efficient collective communication in optical networks", Proc. of ICALP'96. Lecture Notes in Computer Science, 574-585, 1996.
- [Caragiannis] I. Caragiannis and Ch. Kaklamanis and P. Persiano, "Wavelength Routing in All-Optical Tree Networks: A Survey", *Bulletin of the European Association for Theoretical Computer Science*, 2002, Vol. 76, 104-112.
- [Stern] Thomas E. Stern, Krishna Bala, *Multiwavelength Optical Networks: A Layered Approach*, Addison-Wesley, ISBN: 020130967X, May 1999
- [Jain] R. Jain, G. Sasaki, "Scheduling packet transfers in a class of TDM hierarchical switching systems", IEEE International Conference on Communications ICC '91, Vol. 3, 1991, 1559-1563.
- [Battiti] Roberto Battiti, Alan A. Bertossi, Maurizio A. Bonuccelli, "Assigning Codes in Wireless Networks: Bounds and Scaling Properties.", ACM/Baltzer Wireless Networks, Vol. 5, 1999, 195-209.
- [Chiu] Dah-Ming Chiu, Raj Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks", Computer Networks and ISDN Systems, 1989, Vol. 17, 1-14.
- [Ozbay] H. Ozbay, S. Kalyanaraman, A. Iftar, "On rate-based congestion control in high-speed networks: Design of an H-infinity based flow controller for single bottleneck", Proc. of the American Control Conference, June 1998, 2376-2380.
- [Loh] P.K.K. Loh, Wen Jing Hsu, Cai Wentong, N. Sriskanthan, "How network topology affects dynamic loading balancing", IEEE Parallel & Distributed Technology: Systems & Applications, Vol. 4, No. 3, Fall 1996, 25-35.
- [Naghshineh] M. Naghshineh, R. Guerin, "Fixed versus variable packet sizes in fast packet-switched networks", Proc. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM '93., Networking: Foundation for the Future, IEEE Press, Vol. 1, 1993, 217-226.
- [Halmos] Paul R. Halmos, *Naive Set Theory*, Springer-Verlag New York Inc, ISBN 0-387-90092-6, 1974, 26-29.
- [Kuonen] P. Kuonen, "The K-Ring: a versatile model for the design of MIMD computer topology", Proc. of the

High-Performance Computing Conference (HPC'99), San Diego, USA, April 1999, 381-385.

- [Sayoud] H. Sayoud, K. Takahashi, B. Vaillant, "Designing communication network topologies using steady-state genetic algorithms", IEEE Communications Letters, Vol. 5, No. 3, March 2001, 113-115.
- [SwissT1] Pierre Kuonen, Ralf Gruber, "Parallel computer architectures for commodity computing and the Swiss-T1 machine", EPFL Supercomputing Review, Nov 99, pp. 3-11, <http://sawwww.epfl.ch/SIC/SA/publications/SCR99/scr11-page3.html>
- [Gruber] Ralf Gruber, "Commodity computing results from the Swiss-Tx project Swiss-Tx Team", http://www.grid-computing.net/documents/Commodity_computing.pdf
- [Campers] G. Campers and O. Henkes and J. P. Leclercq "Graph Coloring Heuristics: A Survey, Some New Propositions and Computational Experiences on Random and 'L}eighton's' Graphs" Proc. Operational Research, 917-932, 1988.
- [Hertz] A. Hertz and D. de Werra "Using Tabu Search Techniques for Graph Coloring" in Computing(39) 345-351, 1987.
- [Beauquier] B. Beauquier, J.C. Bermond, L. Gargano, P. Hell, S. Pérennes, U. Vaccaro, "Graph Problems Arising from Wavelength-Routing in All-Optical Networks", 2nd IEEE Workshop on Optics and Computer Science (WOCS, part of IPPS '97), IEEE Press, April 1997.
- [Brelaz] Daniel Brelaz, "New Methods to Color the Vertices of a Graph", CACM(22), 1979, 251-256.

Annex A. Performance loss induced by a graph coloring heuristic algorithm

The search for a liquid schedule requires to partition the traffic [Halmos] into a set of non-overlapping mutually non-congesting transfers. The problem can also be formulated as an intersection graph colouring problem [Campers], [Hertz]. Application of conflict graph colouring is also studied in [Beauquier]. Vertices of the conflict intersection graph represent the transfers of the traffic. Edges between vertices represent congestions between transfers. Two vertices of the conflict graph are joined by an edge if the two corresponding transfers are congesting.

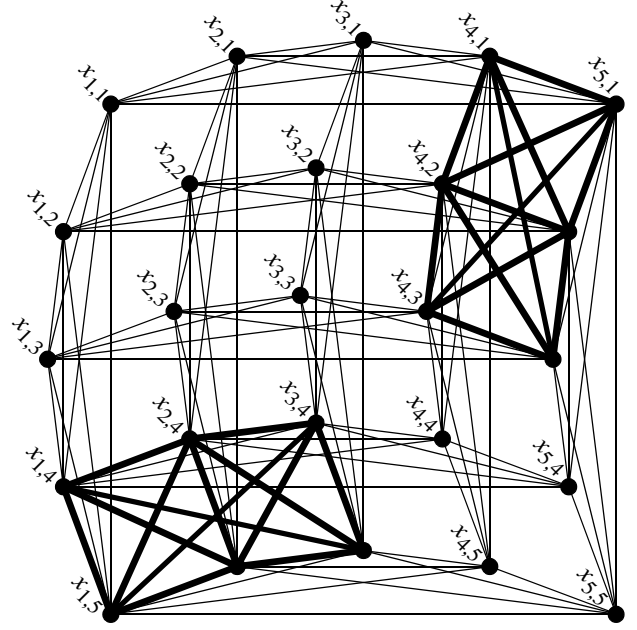


Fig. 12. Conflict graph corresponding to the 25-transfer the traffic of Fig. 2.

Fig. 12 shows the graph corresponding to the data exchange shown in the Fig. 2. The 25 vertices of the graph represent the transfers. The edges represent congestion relation between transfers. Each edge represents one or more communication links shared by two transfers. Bold edges of the figure represent all congestions due to bottleneck links l_{ab} and l_{ba} .

The objective is to colour the vertices of the graph with as few colours as possible, such that no two adjoint vertices have the same colour. Vertex $x_{i,j}$ corresponds to a transfer from the sending processor t_i to the receiving processor r_j . For example vertex $x_{4,1}$ represents the transfer $t4 \rightarrow r1 = \{l_{t4}, l_{ba}, l_{r1}\}$.

Whenever a liquid schedule exists, an optimal solution of the graph colouring problem is a liquid schedule. The chromatic number of the graph's optimal colouring is the length of the liquid schedule. Vertices having the same colour represent a time frame of the liquid schedule.

Density of edges for the sample topologies of Swiss-TI

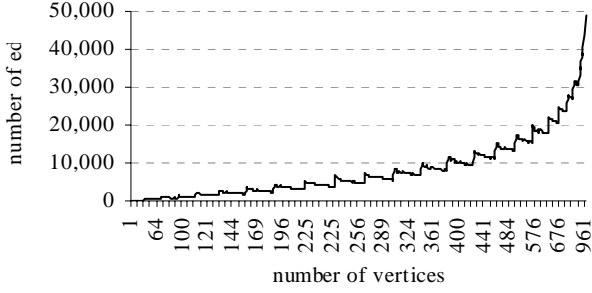


Fig. 13. Characteristics of the graphs corresponding to 362 sample traffic exchanges on the testbed shown in the Fig. 8.

The graph to be coloured is characterized by relatively low density of its edges. Fig. 13 shows the ratio of the number of vertices of a graph to the density of its edges. We can label each edge of the graph by the link(s) causing the congestion. A half-all-to-all data exchange on the Swiss T1 cluster with 32 transmitting and 32 receiving processors forms a graph with $32 \times 32 = 1024$ vertices and 48704 edges.

We compared our method of finding a liquid schedule with the results obtained by applying a greedy high-speed graph coloring algorithm *Dsatur* [Brelaz]. We compared the network performance corresponding to the suboptimal solution of the greedy algorithm with the performance delivered by liquid schedule algorithm. Fig. 14 shows the loss in overall throughput on the Swiss-T1 sample sub-topologies due to the redundant unnecessary colours induced by the greedy graph colouring algorithm.

Loss in performance due to suboptimal scheduling

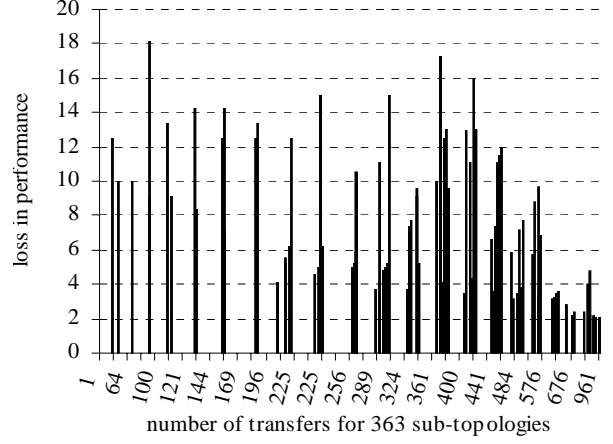


Fig. 14. Loss in performance induced by schedules computed with the *Dsatur* heuristic algorithm.

For 74% of the topologies there is no loss of performance. For 18% of the topologies, the performance loss is below 10% and for 8% of the topologies, the loss of performance is between 10% and 20%. However the computation time of the greedy algorithm is polynomial and compares therefore favourably with the liquid schedule construction algorithm.

Annex B. Comparison of efficiencies of liquid scheduling algorithm with MILP

For 362 test bed topologies introduced in the section 7 we applied MILP method. The efficiency of the MILP for this problem was far bellow compared with our liquid scheduling algorithm. The median of the gain factor of our algorithm is about 4000. Fig. 15 shows the computation times required for the optimal scheduling by both MILP and liquid scheduling methods.

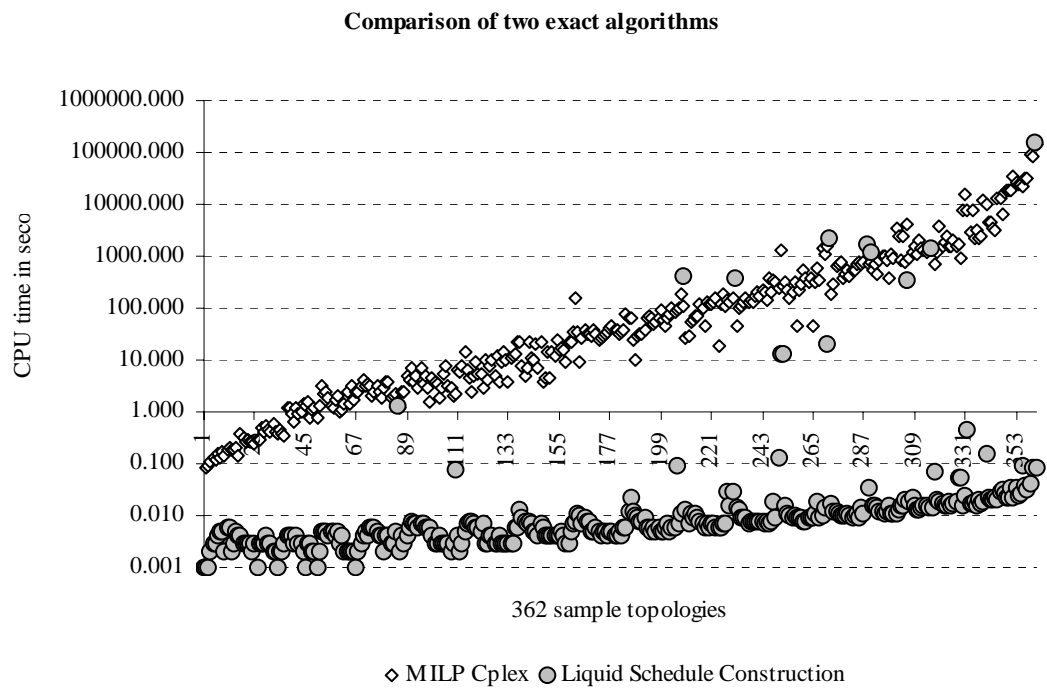


Fig. 15. Comparison of efficiencies of two algorithms, MILP method and liquid scheduling method.