

Liquid Schedules of Network Traffics

Emin Gabrielyan

*Computer Science Department
École Polytechnique Fédérale de Lausanne, 1015 Switzerland
Phone: +41 21 6935261, Fax: +41 21 6936680
Emin.Gabrielyan@epfl.ch*

Abstract

We introduce the theory of liquid schedules, a method for the optimal scheduling of collective data exchanges relying on the knowledge of the underlying network topology and routing scheme. Liquid schedules ensure the maximal utilization of network's bottlenecks and offers an aggregate throughput as high as the flow capacity of a liquid in a network of pipes. The limiting factors of liquid schedules' current theory are equality of packet sizes, ignoring of network delays and predictability of the traffic. In spite of limitations of the current theory the liquid schedules may be used in many contiguous data flow processing applications such as parallel acquisition of multiple video streams, high energy physics detector-data acquisition and event assembling, voice-over-data traffic switching, etc. The collective data flow processing throughput assured by liquid schedules in highly loaded complex networks may be multiple times higher in comparison with the throughput of traditional topology-unaware techniques such as round-robin, random or fully asynchronous transfer schemes. The measurements of the theoretically computed liquid schedules applied to the real low-latency network have given results very close to the theoretical predictions. On a 32 node (64 processor) low latency K-ring cluster we've doubled the aggregate throughput compared with the traditional exchange technologies. This paper presents the theoretical basis of the liquid schedules and an efficient technique for the construction of liquid schedules.

Keywords: Liquid schedules, optimal network utilization, traffic scheduling, all-to-all communications, collective operations, network topology, topology-aware scheduling.

1. Introduction

The interconnection topology is one of the key - and often limiting - factors of parallel applications [1], [2], [3], [4]. Depending on the transfer block size, there are two opposite factors (among others) influencing the aggregate throughput. Due to the message overhead, communication cost increases with the decrease of the message size. However, smaller messages allow a more progressive utilization of network links. Intuitively, the data flow becomes liquid when the packet size tends to zero [5], [6] (see also [7], [8]). The aggregate throughput of a collective data exchange depends on the application's underlying network topology. The total amount of data together with the longest transfer time across the most loaded links or *bottlenecks*, gives an estimation of the aggregate throughput. This estimation will be defined here as the *liquid* throughput of the network. It corresponds to the flow capacity of a non-compressible fluid in a network of pipes [6]. Due to the packeted behaviour of data transfers, congestions may occur in the network and thus the aggregate throughput of a collective data exchange may be lower than the liquid throughput. The rate of congestions for a given data exchange may vary depending on how

the sequence of transfers forming the data exchange is scheduled by the application. Similar problems have been shaped in one-to-all and all-to-all communications over satellite-switch/TDM networks [9] and wavelength division multiplexing optical networks [10]. However, besides a few relatively similar problems, we haven't found research on this topic.

For example consider an all-to-all collective data exchange represented by Fig. 1. Suppose the throughput of links is 100 MB/s. There are 5 transmitting processors (T1,... T5), each of them sending a packet to each of the receiving processors (R1... R5). One may easily compute that the liquid throughput of this data exchange is 416.67 MB/s (for details see the end of this section). A *round-robin* schedule consists of five logical steps: (1) {T1→R1,T2→R2...T5→R5}, (2) {T1→R2,T2→R3...T5→R1}, etc. Intuitively the round-robin schedule shall provide the best performance, however one may compute that its throughput (357.14 MB/s) is lower than the liquid throughput, due to the non-optimal utilization of the bottlenecks l_{11} and l_{12} . Nevertheless Fig. 9 shows that there exists a schedule achieving the liquid throughput of the data exchange. Our theory applied to much more complex topologies computes optimal traffic schedules considerably increasing collective data exchange throughputs relatively to the traditional topology-unaware techniques such as round-robin, random or fully asynchronous transfer modes. On the Swiss-Tx supercomputer [11], [12], a 32 node K-ring [13] cluster, we've doubled the aggregate throughput by applying the presented scheduling technique. Thanks to the presented theory, for most of the underlying topologies (allocations of computing nodes), the computational time required to find an optimal schedule had taken less than 1/10 of a second (the presentation of performance measurements is given in another paper).

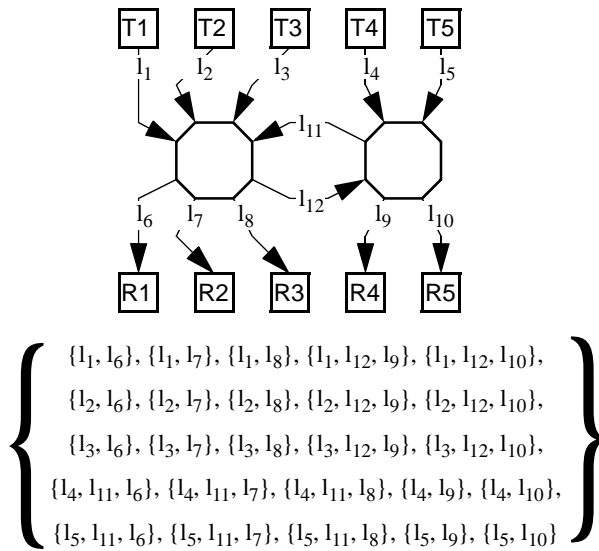


Fig. 2. All-to-all traffic. The links are unidirectional. Nevertheless each of the pairs of links (l_1, l_6)...(l_{11}, l_{12}) and each of the pairs of processors (T1,R1)...(T5,R5) may be considered respectively as single bidirectional link and single physical processor.

This section introduces the traffic-set model which underlies the proposed theory of optimal scheduling. In the traffic-set model 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 the static routing scheme. Let's give a few introducing definitions.

A *transfer* is a set of links (i.e. the path from a sending processor to a receiving processor). A *traffic* is a set of transfers. Fig. 1 shows the traffic for the all-to-all exchange. Note that the all-to-all exchange in a network for our model is just a particular case of 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 utilize a common link otherwise they are *simultaneous*. We see, therefore, that this model is limited by the representation of the data exchanges consisting of

identical size packets. The optimal scheduling of a traffic of variable size packets is a subject of another research.

One would think that the traffic-set model may not represent a collective exchange where a sending processor may transfer a packet to a given receiving processor more than once, however such a collective exchange may be easily converted into an equivalent problem so as to be represented by the traffic-set model. For example, suppose that a collective exchange of Fig. 1 in addition to all 25 transfers, performs once more the transfer $\{l_1, l_6\}$ (i.e. $T1 \rightarrow R2$). Clearly this 26-transfer-traffic may not be directly represented as a set of transfers. However, Fig. 2 shows that we may easily add to the topology of Fig. 1 two additional virtual links l_{13} and l_{14} and distinguish two identical transfers and therefore represent the 26-transfer-traffic through the set-traffic model.

Many contiguous data flow processing applications such as parallel acquisition of multiple video streams, high energy physics detector-data acquisition and event assembling, voice-over-data traffic switching, etc. may be covered by this model. Note that the limitation on the equality of packet sizes obviously doesn't limit applications by equal bandwidth cross-streams.

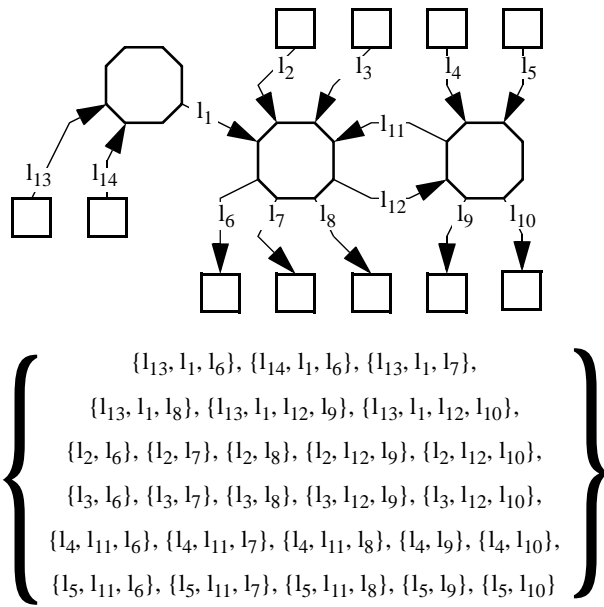


Fig. 2. Multiple transfers through same paths (modification of the topology of Fig.1)

Simultaneity is a subset of X formed from a collection of mutually simultaneous (non-congesting) transfers. A transfer is in congestion with a simultaneity if the transfer is in congestion with an element 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 the simultaneity (see section 2). A simultaneity of a traffic is processed in the timeframe of a single transfer. $\lambda(l, X)$, the *load* of link l in the traffic X is the number of transfers in X using l ,

$$\text{i.e. } \lambda(l, X) = \sum_{x \in X} \begin{cases} 1 & \text{if } l \in x \\ 0 & \text{if } l \notin x \end{cases} \quad (\text{see Fig. 5 and Fig. 6}).$$

The *duration* $\Lambda(X)$ of a traffic X is the maximal value of the load among all links involved in the traffic. The links having maximal load values are called *bottlenecks*. The *liquid throughput* of a traffic X is the ratio

$\#(X)/\Lambda(X)$ multiplied by a single link throughput, where $\#(X)$ is the number of transfers in the traffic X . For example, the traffic X shown in Fig. 1 has a number of transfers $\#(X) = 25$ and the duration of the traffic is $\Lambda(X) = 6$. Therefore the aggregate liquid throughput is the ratio $25/6$ of a single link throughput, i.e. $(25/6) \times 100\text{MB/s}$, supposing a single link throughput of 100MB/s .

DEFINITIONS. Let us define a simultaneity of X as a *team* of X if it uses all bottlenecks of X (note that a traffic may not have a team, see Fig. 8). A team of X is *full* if it is a full simultaneity of X

(see section 3). Let $\mathfrak{R}(X)$ and $\mathfrak{S}(X)$ be respectively the sets of all full simultaneities and all full teams of X . Let $\sigma(\bigcup X)$ be the set of bottlenecks of X , i.e. $\left\{ l \in \bigcup X \mid \lambda(l, X) = \Lambda(X) \right\}$.

In sections 2 and 3 we present techniques for the construction of full simultaneities and full teams of a traffic, respectively, and prove the coverage of the whole solution space. Based on the achievements of the previous sections we'll conclude this paper in section 4 by presenting a liquid schedule searching technique that will be proven to be successful whenever a solution exists.

2. Full simultaneities

The construction of the liquid schedules discussed in the section 4 is based on the ability to construct the set of all full teams of an arbitrary traffic. It is a critical requirement that each full team be efficiently built once and only once. This section presents the technique for the enumerated construction of the whole set of full simultaneities of a traffic counting each full simultaneity one by one by the use of a recursive tree. Meanwhile the following section, based on the simultaneities construction technique, presents an efficient algorithm for building the set of all full teams of the traffic. The building of the set of all full simultaneities $\mathfrak{R}(X)$ is based on its successive partitioning into subcollections. Subcollections of $\mathfrak{R}(X)$ are represented by so called *ancestors* (defined later). Take a - not necessarily full - simultaneity A of a traffic X . A direct mapping between the simultaneity A and a subcollection of $\mathfrak{R}(X)$ may be defined in natural way such that the subcollection of $\mathfrak{R}(X)$, corresponding to A , consists of all and only those full simultaneities of X which includes A . We may call the so defined subcollection of $\mathfrak{R}(X)$ as a *posterity* of A . Of course any member of the posterity of A besides of the elements of A may consists of only *A-simultaneous* transfers (non-congesting with any element of A). Also any *A-simultaneous* transfer of X may be found in one or more members of the posterity of A . Thereby a subcollection (but not any subcollection) of $\mathfrak{R}(X)$ may be represented as a posterity of a simultaneity of A . Relatively to a simultaneity A , in this section, an option will be required allowing to select only those members of posterity of A which doesn't contain some of *A-simultaneous* transfers, specified separately. The simultaneity A together with the "denied" subset of *A-simultaneous* transfers will be defined as an ancestor.

By definition the ordered pair (*outer*, *inner*) is an *ancestor* of full simultaneities of X if the inner is a simultaneity of X and the outer consists of some of simultaneous with the inner transfers of X . An ancestor may be also represented as an ordered triplet (*outer*, *depot*, *inner*) where the depot contains all remaining in X simultaneous with the inner transfers not contained in the outer. The triplet representation of ancestors formally does not involve additional information, although it will be used in the further context. An ancestor of full simultaneities of X may be referred to in short as an ancestor *within* X . The outer, depot and inner of an ancestor R may be denoted as $R_{[-1]}$, $R_{[0]}$ and $R_{[+1]}$ respectively.

Let's demonstrate an ancestors on an example. Consider an eight-transfer-traffic on a network consisting of two switches, 4 sending and 2 receiving processors as it's shown in Fig. 3. Let us introduce the following graphical notation for the 8-transfer-traffic $\{ \text{S}_1\text{S}_2\text{R}_1, \text{S}_1\text{S}_3\text{R}_1, \text{S}_1\text{S}_4\text{R}_1, \text{S}_2\text{S}_3\text{R}_2, \text{S}_2\text{S}_4\text{R}_2, \text{S}_3\text{S}_4\text{R}_2, \text{S}_1\text{S}_2\text{S}_3\text{R}_1, \text{S}_1\text{S}_2\text{S}_4\text{R}_1 \}$,

$\{\text{---}\}, \{\text{---}\}, \{\text{---}\}\}$. Accordingly the triplet $(\{\text{---}\}, \{\text{---}\}, \{\text{---}\})$ is an example of an ancestor within this traffic.

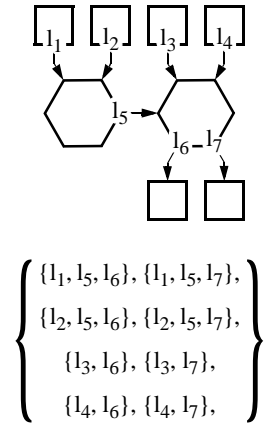



Fig. 3. A traffic on a simple network

An ancestor R is *completed* if its outer and its depot are empty i.e. $R_{[-1]} = \emptyset$ and $R_{[0]} = \emptyset$. The inner of a completed ancestor within X is a full simultaneity of X . For example the triplet $(\emptyset, \emptyset, \{\text{S}, \text{X}\})$ is a completed ancestor within the eight-transfer-traffic and accordingly the set $\{\text{S}, \text{X}\}$ is a full simultaneity of this traffic. Further, we may represent simultaneities compactly through a single shaped graphical symbol so as  is a synonym for $\{\text{S}, \text{X}\}$.

A *heir* of an ancestor within X is any full simultaneity of X which includes the inner of the ancestor and does not contain any element of the outer. Considering the traffic of the Fig. 3 the full simultaneity $\boxed{\text{grey } X}$ is the heirs of the ancestor ($\{\text{grey } \text{grey}\}$, $\{\text{grey } \text{grey}, \text{grey } \text{grey}\}$, $\{\text{grey } \text{grey}, \text{grey } \text{grey}\}$, $\{\text{grey } \text{grey}\}$), but $\boxed{\text{grey } \text{grey}}$ isn't. Completed ancestor have one heir, its inner, so that the sole heir of the


following completed ancestor $(\emptyset, \emptyset, \{\text{diagram 1}, \text{diagram 2}\})$ is $\boxed{\text{diagram 3}}$. Any full simultaneity of X is a heir of the *prim-ancestor* $(\emptyset, X, \emptyset)$. The prim-ancestor of the eight-transfer-traffic is $(\emptyset, \{\text{diagram 4}, \text{diagram 5}, \text{diagram 6}, \text{diagram 7}, \text{diagram 8}, \text{diagram 9}, \text{diagram 10}\}, \emptyset)$.

The collection of all heirs of an ancestor is the *progeny* of the ancestor. For example, the progeny of the ancestor $(\{\text{⚡⚡}\}, \{\text{⚡⚡}, \text{⚡⚡}\}, \{\text{⚡⚡}\})$ is $\{\text{⚡⚡}, \text{⚡⚡}\}$. By definition the operator ϕ applied to an ancestor R forms its progeny $\phi(R)$. The progeny of the prim-ancestor within X is the set of all full simultaneities of X , $\phi(\emptyset, X, \emptyset) = \mathfrak{R}(X)$.

Consequently to the definition, the progeny of an ancestor R within X is the collection of all those full simultaneities of X that may be built up from the inner by the elements of the depot. The depot of an ancestor shall be in congestion with each element of outer, otherwise the ancestor is *barren* and may not have an heir (the proof is straight forward). This is an example of a barren ancestor ($\{\text{diagram 1}, \text{diagram 2}, \text{diagram 3}, \text{diagram 4}, \text{diagram 5}, \text{diagram 6}, \text{diagram 7}\}, \{\text{diagram 8}\}, \emptyset$) within the eight-transfer-traffic. Precisely an ancestor R is barren if $\exists x \in R_{[-1]}|x \cap \bigcup R_{[0]} = \emptyset$.

Let R be an ancestor within X . Let a be an element of the depot. An R -heir may either contain a or not. All R -heirs containing a together form the progeny of an ancestor (1) having the inner of R enlarged by a and having the depot and the outer of R diminished by each congesting with a transfer. Further, all R -heirs non-containing a together form the progeny of an ancestor (2) obtained from R by moving the element a from the depot to the outer. Let us define ψ_{+a} and ψ_{-a} as the operators forming from R the two sub-ancestors, such that the progeny of $\psi_{+a}(R)$ is the set of all R -heirs containing a and the progeny of $\psi_{-a}(R)$ is the set of all R -heirs non-containing a , i.e. $\phi(\psi_{+a}(R)) = \{A \in \phi(R) | a \in A\}$ and $\phi(\psi_{-a}(R)) = \{A \in \phi(R) | a \notin A\}$.

DEFINITION. Formally the operators ψ_{+a} and ψ_{-a} from an ancestor to an ancestor are defined as follows: $\psi_{+a}(R)_{[+1]} = R_{[+1]} \cup \{a\}$, $\psi_{+a}(R)_{[-1]} = \{x \in R_{[-1]} | x \cap a = \emptyset\}$ and $\psi_{+a}(R)_{[0]} = \{x \in R_{[0]} | x \cap a = \emptyset\}$; $\psi_{-a}(R)_{[+1]} = R_{[+1]}$, $\psi_{-a}(R)_{[-1]} = R_{[-1]} \cup \{a\}$ and $\psi_{-a}(R)_{[0]} = R_{[0]} - \{a\}$; where $a \in R_{[0]}$.

For an example let the ancestor R in the above definition be the prim-ancestor of the eight-transfer-traffic and let the transfer a , an element of the prim-ancestor's depot, be . Then $\psi_{+a}(R) = (\{\emptyset\}, \{\text{transfer}, \text{transfer}\}, \{\text{transfer}\})$ and $\psi_{-a}(R) = (\{\text{transfer}\}, \{\text{transfer}, \text{transfer}, \text{transfer}, \text{transfer}, \text{transfer}, \text{transfer}, \text{transfer}\}, \{\emptyset\})$. The following partitioning properties about the operators ψ_{+a} and ψ_{-a} are true: $\phi \circ \psi_{+a}(R) \cup \phi \circ \psi_{-a}(R) = \phi(R)$ and $\phi \circ \psi_{+a}(R) \cap \phi \circ \psi_{-a}(R) = \emptyset$.

DEFINITIONS. Let ω be a set of ancestors within X . The operator ρ applied to ω removes from ω all barren elements. The *binary fission* operation Ψ applied to ω splits each ancestor of ω having a non empty depot into two sub-ancestors using the operators ψ_{+a} and ψ_{-a} with an arbitrarily chosen element a from the ancestor's depot. Note that the fission operation Ψ has an uncertainty property, since the assertion $\omega = \varpi$ does not imply that $\Psi(\omega) = \Psi(\varpi)$. An assertion about the fission operation Ψ may be true only if it is true for all possible outcomes.

Let us demonstrate the operator ρ and the fission operation Ψ on an example. The operator ρ applied to the following set of ancestors $\{(\emptyset, \emptyset, \{\text{transfer}, \text{transfer}\}), (\{\text{transfer}, \text{transfer}\}, \emptyset, \{\text{transfer}\}), (\{\text{transfer}\}, \{\text{transfer}\}, \{\text{transfer}\})\}$ forms the following set $\{(\emptyset, \emptyset, \{\text{transfer}, \text{transfer}\}), (\{\text{transfer}\}, \{\text{transfer}\}, \{\text{transfer}\})\}$. The fission operation Ψ applied on the last set of ancestors forms $\{(\emptyset, \emptyset, \{\text{transfer}, \text{transfer}\}), (\emptyset, \emptyset, \{\text{transfer}, \text{transfer}\}), (\{\text{transfer}\}, \text{transfer}, \emptyset), (\text{transfer}, \{\text{transfer}\}, \emptyset)\}$. We've omitted in these examples the separating comas in the sets of transfers.

The *progeny* of a collection of ancestors is the union of the progenies of its members. A collection of ancestors within a traffic is *dividing* if the corresponding collection of the progenies of its members partitions the set of all full simultaneities of the traffic. Consequently but particularly, the progeny of a dividing collection is the set of all full simultaneities of a traffic. Clearly, the singleton of the prim-ancestor is a dividing collection. The operation $(\rho \circ \Psi)$ applied to a dividing collection ω does not affect the dividing property of ω . Further, the fission operation Ψ applied to a collection ω reduces the depot of each non-completed ancestor in ω at least by one element, which from follows that at some point a finite composition $\rho \circ \Psi \circ \rho \circ \Psi \circ \dots \rho \circ \Psi \{(\emptyset, X, \emptyset)\}$ forms a collection of completed ancestors each containing as an inner a full simultaneity of X . The equation $\Re(X) = \bigcup_{R \in \rho \circ \Psi \circ \dots \rho \circ \Psi \{(\emptyset, X, \emptyset)\}} \{R_{[+1]}\}$ is the key point to the building of all full

disjoint subsets of X one by one without repetition. The implementations of the operator ρ and the fission operation Ψ do not require any additional techniques and have a low cost functionality.

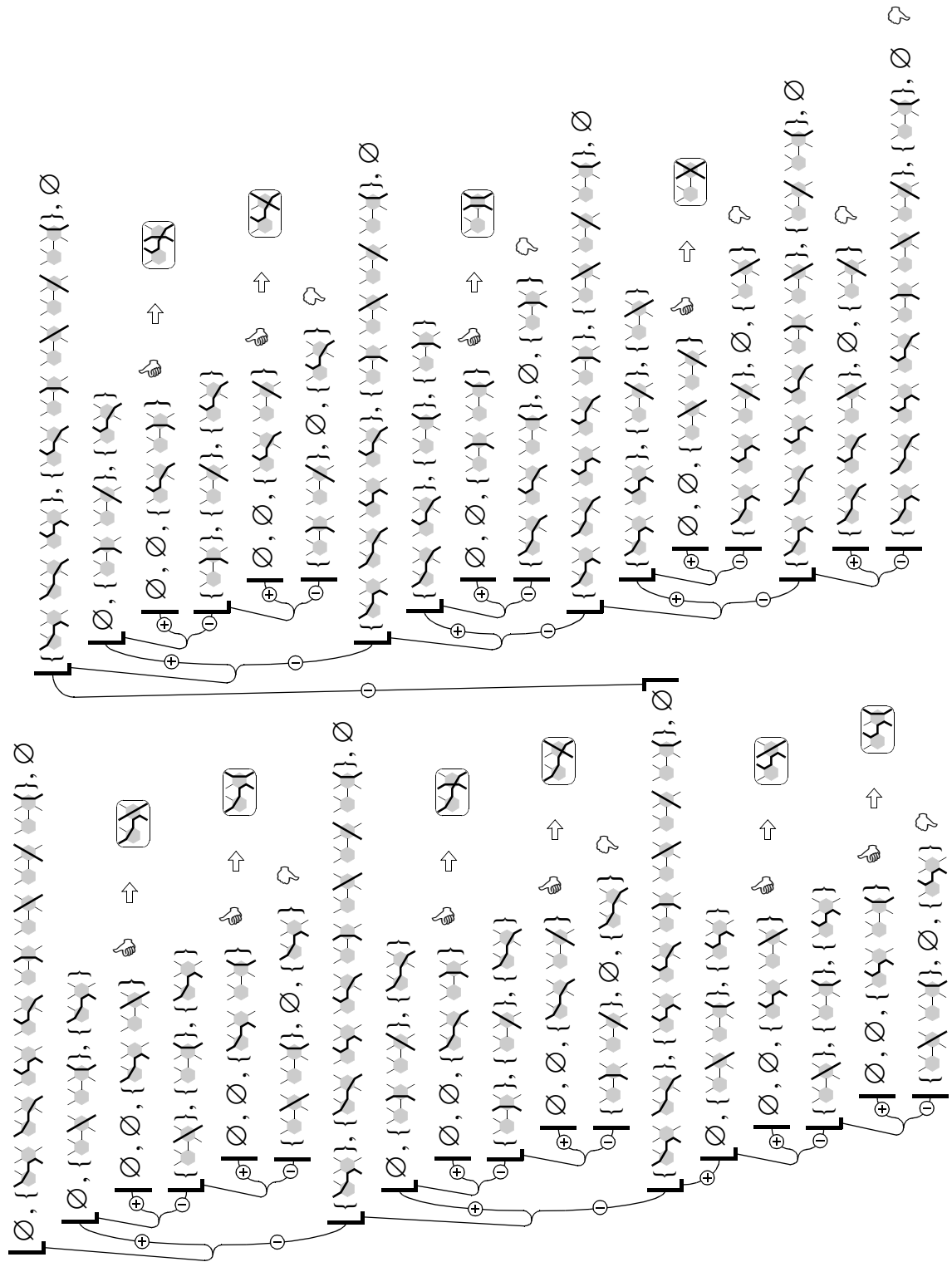












Fig. 4. Reproduction tree of the prim-ancestor. The signs '👍' and '❓' signify completed and barren ancestors respectively.

Fig. 4 shows the concluded reproduction tree of the eight-transfer-traffic's prim-ancestor's singleton leading to the formation of ten complete ancestors representing all ten full

simultaneities: , , , , , , , , , . (The parentheses of triplets and the comas are omitted in the figure).

3. Full Teams

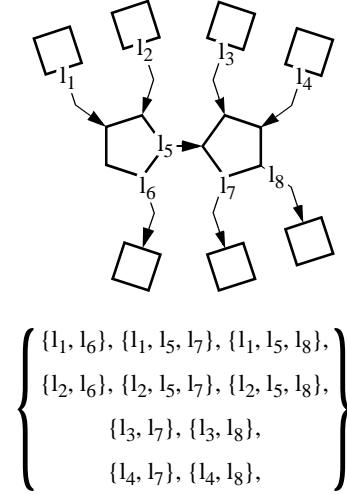


Fig. 5. A traffic on a simple network

Let an ancestor within X be *viscid* if its inner and its depot together don't use all bottlenecks of X . A viscid ancestor may not have a team in its progeny (the proof is straight forward). Precisely an ancestor R within a traffic X is viscid if

$$\sigma(\bigcup X) \not\subset \bigcup (R_{[0]} \cup R_{[+1]}).$$

Let τ be an operator that removes from a collection of ancestors all viscid members. Reproduction of the initial singleton $\{(\emptyset, X, \emptyset)\}$ through the operation $\tau \circ \rho \circ \Psi$ forms developing collections of ancestors σ . The progeny of the collection σ consists of full simultaneities of X and narrows from generation to generation, although it always envelopes all full teams of X and ultimately leads to $\chi(X)$ (the proof is straight forward). A finite composition $\tau \circ \rho \circ \Psi \circ \tau \circ \rho \circ \Psi \circ \dots \circ \tau \circ \rho \circ \Psi \{(\emptyset, X, \emptyset)\}$ contains one and only one

corresponding ancestor for each full team of X . Formally this technique, represented by the equation $\chi(X) = \bigcup_{R \in \tau \circ \rho \circ \Psi \circ \dots \circ \tau \circ \rho \circ \Psi \{(\emptyset, X, \emptyset)\}} \{R_{[+1]}\}$, is sufficient for the construction of all full

teams. Referring this equation as on the first approximation bellow we present an efficient construction.

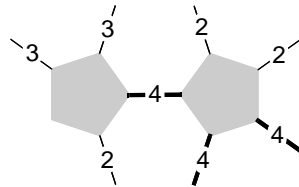










Fig. 6. The load of links of the traffic of Fig. 5.

Let the *skeleton* of a traffic X be the smallest subset of X using all bottlenecks of X . Let the *skeleton* of a team of X be the smallest subset of the team using all the bottlenecks of X . Let the operator $\zeta(X)$ be the skeleton of the traffic X , and the two operand operator $\zeta(A, X)$ be the skeleton of A , a team of X . For an example consider a ten-transfer-traffic shown on the Fig. 5. Similarly to what we've done in the previous section, let us represent this ten-transfer-traffic as a set of graphically denoted transfers $\{\text{diagram 1}, \text{diagram 2}, \text{diagram 3}, \text{diagram 4}, \text{diagram 5}, \text{diagram 6}, \text{diagram 7}, \text{diagram 8}, \text{diagram 9}, \text{diagram 10}\}$.

Fig. 6. shows in bold the three bottlenecks of the network in the example. The transfers  and  of the ten-transfer-traffic are not using the bottlenecks and therefore the skeleton of this ten-transfer-traffic is the following eight-transfer-traffic $\{\text{diagram 2}, \text{diagram 3}, \text{diagram 5}, \text{diagram 6}, \text{diagram 7}, \text{diagram 8}, \text{diagram 9}, \text{diagram 10}\}$. For example ,  and  are teams of the ten-transfer-traffic. Their corresponding skeletons respectively are ,  and .

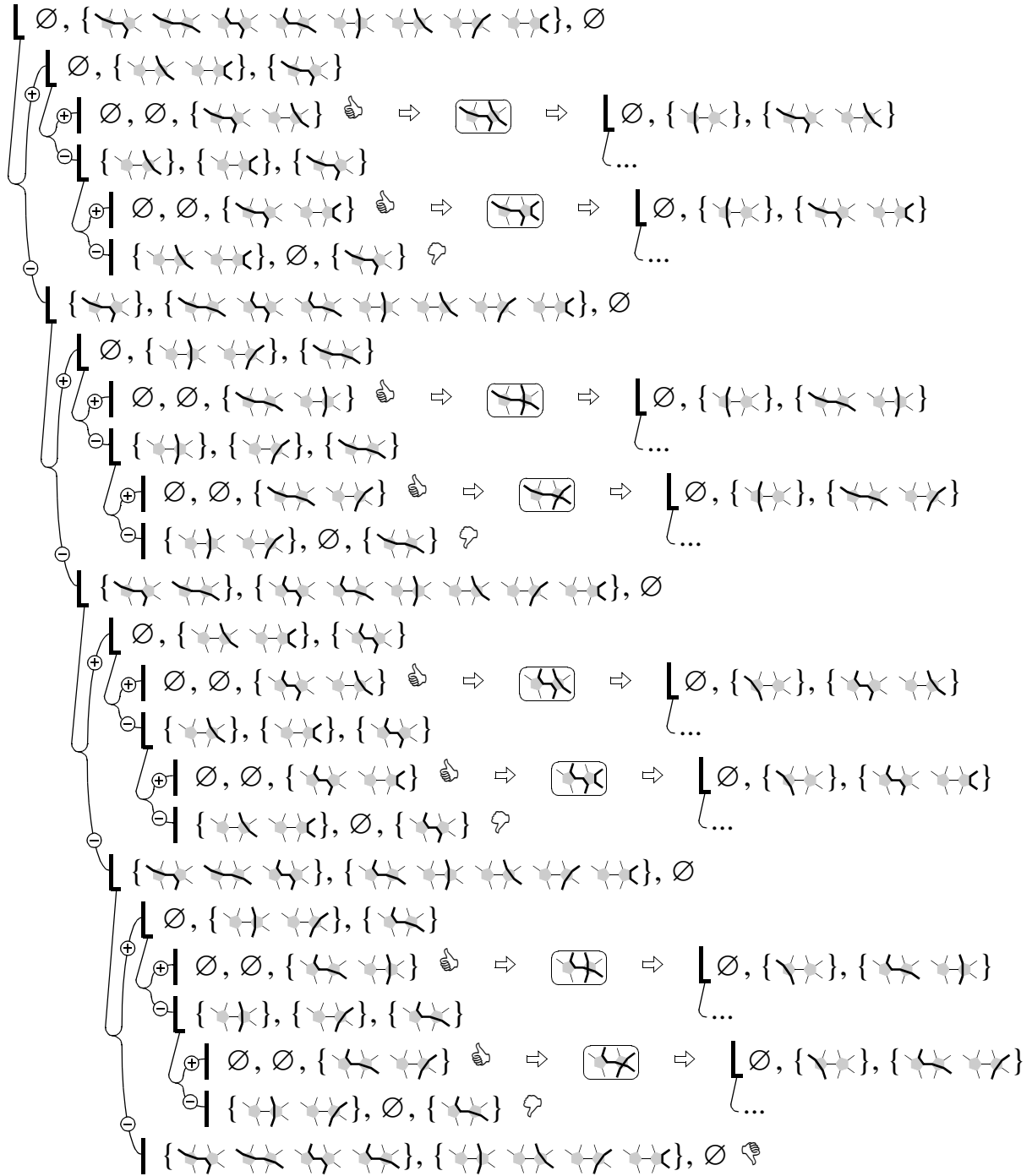


Fig. 7. Two phase reproduction leading to 8 full teams. The signs '👍', '👎' and '👉' signify completed barren and viscid ancestors respectively.

Consider a traffic X . A full team of the traffic's skeleton is a skeleton of the traffic's (full) team and a skeleton of traffic's (full) team is a full team of traffic's skeleton (the proof is out of scope of this paper). In other words, $\forall(A \in \mathfrak{S} \circ \zeta(X)), \exists(B \in \mathfrak{S}(X))|A = \zeta(B, X)$ and $\forall(B \in \mathfrak{S}(X)), \exists(A \in \mathfrak{S} \circ \zeta(X))|A = \zeta(B, X)$. Since each full team of X is a simultaneity built up

on a full team of $\zeta(X)$ an efficient building of full teams may be implemented in two phases. Initially we build all full teams of the skeleton of X using the above presented approximation

$$\mathfrak{S} \circ \zeta(X) = \bigcup_{R \in \tau \circ \rho \circ \Psi \circ \dots \circ \tau \circ \rho \circ \Psi \{(\emptyset, \zeta(X), \emptyset)\}} \{R_{[+1]}\}. \text{ Further, the idea is to build up all variations of}$$

bodies on each skeleton. For each full team A of $\zeta(X)$ we build an ancestor within X whose progeny consists of all those full teams of X whose skeleton is A . By doing so we form a collection of ancestors whose reproduction with the operator $\rho \circ \Psi$ ultimately leads to the set of all full teams of X , i.e.

$$\mathfrak{S}(X) = \rho \circ \Psi \rho \circ \Psi \dots \rho \circ \Psi \left(\bigcup_{A \in \mathfrak{S} \circ \zeta(X)} \left\{ \left(\emptyset, \left\{ x \in X \mid x \cap \bigcup A = \emptyset \right\}, A \right) \right\} \right)$$

Fig. 7 demonstrates the evolution of the two phase reproduction. First phase propagates the prim-ancestor of the skeleton of the original traffic by means of binary fission. The first phase is concluded by a set of completed ancestors each representing a full team of the traffic's skeleton and therefore a skeleton of some teams of the traffic. Second phase evolves each skeleton building up collection of traffic's full teams.

4. Liquid schedules

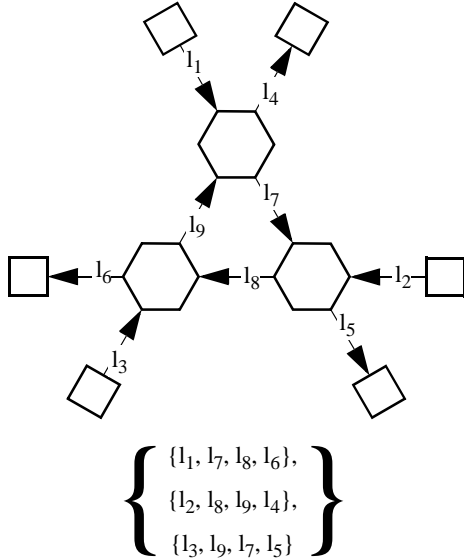


Fig. 8. No team and no liquid schedule

Recall that a *partition* of X is a disjoint collection of non-empty subsets of X whose union is X [14]. A *schedule* α of a traffic X is a collection of simultaneities of X partitioning the traffic X . A *timeframe* of a schedule α is an element of α . $\#(\alpha)$, the *length* of a schedule α , is the number of timeframes 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. The Fig. 8. demonstrates a traffic, which does not have a team and therefore may not have a liquid schedule. Fig. 9 shows a liquid schedule of the collective traffic shown in Fig 1.

The duration of a traffic X is the load of its bottlenecks. Consider l as one of the bottlenecks of X . The load of l is the number of transfers in X using l . Now let α be a schedule on X . By definition α is a collection of simultaneities of X , partitioning X . Since α partitions X , a transfer of X (and particularly a transfer using l) shall be found in one and only one of the timeframes of α . Since a timeframe of α is simultaneous it may contain only one or no

transfer using l . Therefore if the length of α is equal to the number of transfers in X using the bottleneck l , then each timeframe of α shall contain a transfer using l . Inversely, if each timeframe of α has a transfer using l , then the length of α shall be equal to the number of transfers using l . Hence if a schedule is liquid then each of its timeframes uses all bottlenecks, and if all timeframes of a schedule use all bottlenecks then the schedule is liquid.

$$\left(\begin{array}{c} \left\{ \begin{array}{c} \{l_1, l_{12}, l_9\}, \\ \{l_2, l_7\}, \\ \{l_3, l_8\}, \\ \{l_4, l_{11}, l_6\}, \\ \{l_5, l_{10}\} \end{array} \right\} \\ \left\{ \begin{array}{c} \{l_1, l_{12}, l_{10}\}, \\ \{l_2, l_6\}, \\ \{l_4, l_{11}, l_7\}, \\ \{l_5, l_9\} \end{array} \right\} \\ \left\{ \begin{array}{c} \{l_1, l_8\}, \\ \{l_2, l_{12}, l_9\}, \\ \{l_3, l_6\}, \\ \{l_4, l_{10}\}, \\ \{l_5, l_{11}, l_7\} \end{array} \right\} \end{array} \right) \left(\begin{array}{c} \left\{ \begin{array}{c} \{l_1, l_7\}, \\ \{l_2, l_8\}, \\ \{l_3, l_{12}, l_9\}, \\ \{l_5, l_{11}, l_6\} \end{array} \right\} \\ \left\{ \begin{array}{c} \{l_1, l_6\}, \\ \{l_2, l_{12}, l_{10}\}, \\ \{l_3, l_7\}, \\ \{l_4, l_{11}, l_8\} \end{array} \right\} \\ \left\{ \begin{array}{c} \{l_3, l_{12}, l_{10}\}, \\ \{l_4, l_9\}, \\ \{l_5, l_{11}, l_8\} \end{array} \right\} \end{array} \right)$$

In other words, we derived an equivalent condition for the liquidity of a schedule. The necessary and sufficient condition for the liquidity of a schedule is that all bottlenecks be used by each timeframe of the schedule. Recall that we've defined a simultaneity of X as a *team* of X if it uses all bottlenecks of X . Consequently, an equivalent condition for the liquidity of a schedule α on X is that each timeframe of

Fig. 9. A liquid schedule of the collective traffic shown in Fig. 1. The comas separating elements of the schedule are omitted.

α be a team of X .

Our goal is to design an algorithm that may partition a traffic so as to form a liquid schedule (whenever possible).

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 $X - A$. A chain of interesting properties are successively derived (whose formal proof is out of scope of this paper): If α is a liquid schedule on X then for any timeframe A of α the schedule $\alpha - \{A\}$ is a liquid schedule on $X - A$. Further, any non-empty subset β of a liquid schedule is liquid. Consequently, the necessary and sufficient condition of liquidity of a schedule α is that for any non-empty subset β of α each timeframe of β use all bottlenecks of $\bigcup \beta$ (note that (1) $\sigma(\bigcup \alpha) \subset \sigma(\bigcup \beta)$ i.e. the bottlenecks of $\bigcup \beta$ form larger set than bottlenecks of $\bigcup \alpha$).

If the traffic has a liquid schedule, then, according to the above discussion, a schedule reduced by one team is a liquid schedule on the shortened traffic. This is the key point in searching for a liquid schedule. Consider traffic X as a problem whose solution is a liquid schedule α . Assume a technique 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, see the equation (1) above). By doing so, we considerably reduce the search space without affecting the solution space.

We intend to limit the search space when building a liquid schedule. 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 one of its timeframes 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. 9 shows a liquid schedule built as explained above. Let us denote the timeframes in Fig. 9 as follows: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ (according to the order given in Fig. 9.) Traffic X is the union of

the timeframes $X = \bigcup_{i=1}^6 A_i$. The schedule is constructed such that at any step i , the timeframe A_i

is a full team of the sub-traffic $X - \bigcup_{k=1}^{i-1} A_k$. The timeframe A_i being a team of the sub-traffic

$X - \bigcup_{k=1}^{i-1} A_k$ incorporates therefore all bottlenecks of this sub-traffic (shown in bold).

References

- [1] 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.
- [2] Pangfeng Liu, Jan-Jan Wu, Yi-Fang Lin, Shih-Hsien Yeh, "A simple incremental network topology for worm-hole switch-based networks", Proc. 15th International Parallel and Distributed Processing Symposium, 2001, 6-12.
- [3] 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, 25-35.

- [4] V. Puente, C. Izu, J. A. Gregorio, R. Beivide, J. M. Prellezo, 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.
- [5] 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.
- [6] 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>
- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] 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>
- [12] Ralf Gruber, "Commodity computing results from the Swiss-Tx project Swiss-Tx Team", http://www.grid-computing.net/documents/Commodity_computing.pdf
- [13] 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.
- [14] Paul R. Halmos, *Naive Set Theory*, Springer-Verlag New York Inc, ISBN 0-387-90092-6, 1974, 26-29.