Bubbles: Adaptive Routing Scheme for High-Speed Dynamic Networks*

(Preliminary Version)

Shlomi Dolev† Evangelos Kranakis‡ Danny Krizanc‡

Abstract

We present the first dynamic routing schemes for high-speed networks. The scheme is based on a hierarchical *bubbles* partition of the underlying communication graph. We rank dynamic routing schemes by their *adaptability*, i.e., the maximum number of sites to be updated upon a topology change.

We consider the case in which each node in the network may be directly connected with at most δ neighboring nodes. An advantage of our scheme is that it implies small number of updates upon a topology change. In particular, for the case of constant δ we prove that our scheme is optimal in its adaptability by presenting a matching tight lower bound.

Our bubble routing scheme is a combination of a distributed routing data-base, a routing strategy and a routing data-base update. We show how to perform the routing data-base update on a dynamic network in a distributed manner.

1980 Mathematics Subject Classification: 68Q99
CR Categories: C.2.1
Key Words and Phrases: Adaptability, Dynamic, Fiber Optic, Network, Routing
Carleton University, School of Computer Science: SCS-TR-253
Note: This paper will be submitted for publication elsewhere.

*Research supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) grant.

†School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada and Department of Computer Science, Texas A&M University, College Station, TX 77843, USA. Email: shlomi@cs.tamu.edu.

‡School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada. Email: {kranakis, krizanc}@scs.carleton.ca.

1 Introduction

The advent of fiber-optic technology dramatically changes the characteristics of distributed networks. It also improves the capabilities of distributed networks because it gives them the potential of supporting new services such as multimedia and real-time applications. Traditional algorithms designed for the point-to-point classical model of distributed networks may neither fit the new characteristics of the high-speed network nor support the new tasks it is capable of achieving. The relation between the bandwidth of a fiber-optics cable (on the order of a Gigabit per second) and the speed of a processor implies a bottleneck in the process time as opposed to the communication time. Therefore, high-speed networks use a fast switching subsystem in order to utilize the power of the fiber optic cables.

Algorithms for basic tasks that match the new network structure are of interest. In (high-speed) distributed networks messages are used for communication between different sites. A message sent from one site to another is transferred through the network according to a *routing scheme*. The routing scheme ensures that the message is forwarded towards its destination. The routing scheme serves the basic communication primitive in the network — message delivery. Being such a basic component, the performance of the distributed network as a whole may be dominated by the quality of the routing scheme. Thus, finding an efficient routing scheme is one of the most important tasks in distributed networks.

Imagine a network in which users may be connected and disconnected upon request. Users may migrate from one geographical region to another causing a change in the demand for services at different parts of the network. Assume further that the network spans the entire world and a single user (or a network junction) changes its location from one street of New York to another: is it reasonable to update the *entire* network with a new routing data-base? We would not like the *entire* distributed network to be updated upon each such dynamic change. In fact we would like to *minimize* the effect of a topology change as much as possible.

Beyond planned topology changes, such as users migration, some transient topology changes may take place due to a failure of communication links or processors. One would like the network to automatically change the routing data-base to reflect the new topology upon the change. The resources used by such distributed routing data-base update (messages and time) have an inherent relation with the number of sites that have to change their portion of the distributed routing data-base.

We distinguish between *static* and *dynamic* routing schemes. A *static* routing scheme is a combination of a distributed routing data-base and a fitting routing strategy. The routing data-base is tailored to the network topology. Whenever the network changes its topology, a new distributed routing data-base is assigned to the network possibly changing the routing data-base portion of each processor. A *dynamic* routing scheme has in addition a fitting routing data-base update. Upon a topology change (e.g. link addition or removal) this fitting routing data-base update would change the distributed data-base only in a *limited* number of sites. We rank the dynamic routing schemes by their *adaptability*, i.e., the maximum number of sites to

be updated upon a topology change. By this definition static routing schemes are associated with adaptability that is in the order of the number of nodes in the network.

The efficiency of a dynamic routing scheme is measured not only by its adaptability. It is also measured by the time and memory complexities associated with it. The time performance is measured by a *stretch factor* — the maximal ratio (over all possible origin-destination pairs) between the number of packets produced in order to deliver a message from an origin to a destination using the scheme and the minimal number of packets required for this delivery. The memory complexity is the total number of bits used for the routing data-base[1].

Previous work: Many clever routing schemes and lower bounds for the resources required for routing in point-to-point networks were presented in the past. The first set of works were mostly designed for special classes of networks like trees [SK85], complete networks [vLT86], and grids [vLT87]. Then routing schemes for general networks were presented in [PU89], [AB+90] and [AP92].

Unfortunately, most of the success in this field is for static networks. Few papers consider the dynamic property of the network. The following quotation is taken from [SK85]: "In actual network the topology may vary in time; in particular, nodes or links may be added or deleted". [SK85] present a partial solution for limited cases of topology changes that keep the network in a tree structure. In [AGR89] it is argued that network changes in static routing schemes (such as [PU89], [AB+89]) "require expensive pre-processing to reconstruct the routing scheme over the whole network. The newly constructed structure is used until the next change...". In contrast [AGR89] design a routing scheme for the restricted case of dynamic growing trees. The solution of [AGR89] can handle neither link nor processor failures nor can it be applied to the case of general graphs.

The previous works mentioned above are for the traditional point-to-point communication network. Those solutions are not applicable for the fiber optic high-speed networks of today. Recently, [GZ94] presented static routing schemes for high-speed networks. The scheme statically assigns links along a path to act as a virtual long link. Upon a single topology change the entire routing data-base might have to be updated.

Contributions of this paper: In this work we present the first dynamic routing scheme for high-speed networks. We present a family of hierarchical *bubbles* schemes. The intuition behind this structure is the behavior of natural bubbles. Assume a partition of a space into bubbles. Whereas bubbles may lose air (members are disconnected) or blow up (new members are inserted) we want to avoid total change of every bubble structure upon a change at a single bubble. We define an upper and lower threshold for the size of a bubble. When the upper threshold is reached the bubble is split into two bubbles. When the lower threshold is reached the bubble is combined with a single neighboring bubble which "swallows" the small bubble (and then is split if necessary).

[1] Note that there is an interesting relation between the adaptability and the memory requirement e.g., it is clear that the worst case in terms of adaptability and memory requirement is when the entire topology is stored in the memory of each processor. Roughly speaking, both the adaptability and the memory requirements gain when the processors have less information on the system.

We consider the case in which each node in the network may be directly connected with at most δ neighboring nodes. This is the case in reality where the number of communication ports of a single processor is limited. For the case of constant δ we prove that our scheme is optimal in its adaptability by presenting a matching tight lower bound.

Our bubble routing scheme is a combination of a distributed routing data-base, a routing strategy and a routing data-base update. We present a distributed routing data-base update for dynamic changing networks.

The rest of the paper is organized as follows. The definition of the problem appears in Section 2. Section 3 presents three routing schemes: The multiple spanning trees, the single leader, and the bubbles. The first two are brought as examples for our complexity measures and a base for comparison with the bubbles scheme. See figure 1 for the comparison summary. Section 4 presents a lower bound on the adaptability of any routing scheme. The distributed update of the bubbles routing data-base is sketched in Section 5. Concluding remarks are given in Section 6.

	Stretch Factor	Memory Required	Adaptability
Multiple Trees	1	$O(n^2 \log n)$	$O(n)$
Single Leader	2	$O(n^2 \log \delta)$	$O(n)$
Bubbles	k	$O((k-1)n^2 \log \delta)$	$O(k3^{k-1}\delta^2 n^{1/k})$

Figure 1: Comparison of Routing Schemes

2 Definition of the Problem

2.1 High-Speed Dynamic Network

We consider the high-speed model of a communication network as described in [CG88]. The network is described by an undirected graph $G = (V, E)$. The vertices, $V = \{1, \ldots, n\}$, represent the processors of the network (where n is essentially an upper bound on the number of processors in a connected component). The edges of the graph represent bidirectional communication channels between the processors. Each processor consists of two components, a *switching subsystem* and a *node control unit* that are connected by a virtual link. The *switching subsystem* is a fast and simple hardware device that switches arriving packets to the appropriate communication link or to its node control unit. Within the switching subsystem each communication link (including the virtual link to the switching subsystems' node control unit) has a unique label. When a packet p arrives at a switching subsystem and the header of p contains at least one label, then the switching subsystem removes the first label, l, and the shortened packet is sent on the link with label l. The *node control unit* of each processor contains the processing hardware and software necessary to extract the information content

of messages (delivered in the packets), do internal computation, and generate packets to be forwarded to other nodes via the processor's switching subsystem.

Due to the above network architecture, it is assumed that a message sent from any processor, P, to any destination, Q, in the network may arrive in one time unit provided the labels along the entire path from P to Q are known to P.

The network is *dynamic* in the strong sense: processors and links may crash and recover arbitrarily. However, the number of edges connected to a node P, which we call the *degree* of P, does not exceed δ.

2.2 Complexity Measures for Routing Schemes

The routing of packets in the network is done according to a *distributed routing data-base* maintained by the node control unit and a fitting *routing strategy* and *data-base update*.

Stretch Factor: The stretch factor of a routing scheme is the ratio between the number of packets *generated* by the scheme and the optimal number of packets *generated* in order to send a message from one node control unit to another. Obviously, a single packet is sufficient when every processor knows the entire topology (including the link labels). Thus, the ratio of a routing scheme is the maximum number of packets generated by node control units to deliver a message sent from one node control unit to another.

Memory: The memory complexity is the total number of bits maintained by the node control units for the routing data-structure.

Adaptability: A single topology change $\mathcal{C}$ occurs when a single processor or a single link joins (or recovers) or leaves (or crashes) the system. Note that for any two topologies $\mathcal{T}_1$ and $\mathcal{T}_2$ there exist a finite sequence of single topology changes $\mathcal{C}_1, \mathcal{C}_2, \ldots$ that transfer $\mathcal{T}_1$ into $\mathcal{T}_2$. For example the sequence can start with adding all the processors that do not appear in $\mathcal{T}_1$ but do appear in $\mathcal{T}_2$, one processor at a time. Then links are added in a similar fashion. Finally, links and then processors are removed to form $\mathcal{T}_2$.

Ideally a topology change causes only very limited number of processors to change their portion of the routing distributed data-base. Thus, we choose the adaptability measure to be the maximum number of processors that have to change their portion of the routing distributed data-base upon a single topology change.

We first ignore the issue of *how* the routing data-base is updated upon a topology change and only count the number of processors that have to change their routing data-base. Then we present a distributed update for the bubbles routing scheme which we call *bubbles update*.

Distributed adaptability: is the maximal number of node control units that have to participate in the distributed routing data-base update upon a single topology change.

The distributed adaptability of our bubbles update is in the same order of the (centralized) adaptability, and thus is optimal too.

3 Routing Schemes

3.1 Multiple Spanning Trees

The simplest routing scheme for our network is the *multiple spanning trees* routing scheme.

Routing Distributed Data-base: The routing distributed data-base is a description of a spanning tree of the entire topology at each processor.

Routing Strategy: The routing strategy is to use the entire path given by the routing distributed data-base.

Routing Update: The routing update has to change the distributed routing data-base upon every processor addition and removal as well as upon removal of a link used as part of a spanning tree by some processor. The update would change the spanning trees representation in an obvious way.

The stretch factor is defined with relation to this scheme — where a single packet is generated for the delivery of a message. The next two lemmas state the memory requirements and adaptability of the multiple spanning trees routing scheme.

Lemma 3.1 *The memory requirement of the multiple spanning trees scheme is* $n^2(\log n + \log \delta)$ *bits.*

Proof: $n(\log n + \log \delta)$ bits are required in order to describe a spanning tree with link labels for every processor. The description is by the use of parentheses form. ■

Lemma 3.2 *The adaptability of the multiple trees scheme is* n.

Proof: By the fact that a single crash or recovery of a processor requires the update of the tree of every processor. ■

The following theorem summarizes the properties of the multiple trees routing scheme.

Theorem 3.3 *The multiple trees routing scheme has the following properties:*
(1) Stretch factor = 1.
(2) Memory requirement $= n^2(\log n + \log \delta)$.
(3) Adaptability $= n$. ■

3.2 Single Leader

Routing Distributed Data-base: Only a single processor, L, has a spanning tree of the entire topology. Every other processor, P, has the path description to L, $path_L$. $path_L$ is a list of link labels that defines a path from P to L.

Routing Strategy: To deliver a message m to a processor Q a packet $(path_L, Q, m)$ is sent to L and then L sends a packet $(path_Q, m)$ to Q.

Routing Update: The routing update has to change the distributed routing data-base upon the leader failure, processors' addition, processors' removal, a failure of a link along a path used by some processor to reach the leader, an addition of a link that connects two connected components. For a given choice of a single leader the update is defined in a natural way.

The stretch factor is two since at most two packets are generated for each message.

The next two lemmas state the memory requirements as well as the adaptability of the single leader scheme.

Lemma 3.4 *The memory requirement for the single leader routing scheme is bounded by* $n(\log n + \log \delta) + (n-1)n \log \delta$.

Proof: The description of a spanning tree of the entire topology requires $n(\log n + \log \delta)$ bits as explained in the proof of Lemma 3.1. The description of a path from a processor P to L includes at most n link labels. Each link label requires $\log \delta$ bits. Thus, each processor but L uses at most $n \log \delta$ bits for the routing data-base. ■

Lemma 3.5 *The adaptability of the single leader scheme is* $n - 1$.

Proof: A topology change that may cause every processor to change its topology data-base is a crash of L. Upon such a crash every processor should change the path, $path_L$, to reach a new leader. ■

The following theorem summarizes the properties of the single leader routing scheme.

Theorem 3.6 *The single leader routing scheme has the following properties:*
(1) Stretch factor = 2.
(2) Memory requirement $= n(\log n + \log \delta) + n^2 \log \delta$.
(3) Adaptability $= n - 1$. ■

3.3 Bubbles

Routing Distributed Data-base: Given a list of k integers $x_1, x_2, \cdots x_k$, where $x_1 = n$ and $x_i > \delta x_{i+1}$, the communication graph G is partitioned into *connected* components called *bubbles*.

The partition is done in levels. The first level is a single bubble of the entire graph. We say that the first level is *illegal* since no partition can take place at this level. If the number of processors in G is greater than x_2 then each of the second level bubbles includes at least x_2 processors and at most δx_2 processors. Otherwise, no partition can take place at the second level and the bubble of the second level is *illegal* too.

Then, every bubble of the second level (whether it is legal or not) is further partitioned into third level bubbles that include at least x_3 processors and at most δx_3 processors. Similarly if G contains less than x_3 processors then the bubble of the third level is illegal. The i'th level of the bubble partition is a partition of the graph into connected components each containing at least x_i processors and at most δx_i processors. The definition of an illegal bubble is naturally applied to the i'th level. Typically we choose $x_i = \lceil n^{(k-i+1)/k} \rceil$. The next theorem uses a technique presented in [Va81].

Theorem 3.7 *Every graph has a bubble partition.*

Proof: The proof is by presenting a partition algorithm. Choose an arbitrary node of the graph, R, and construct a spanning tree, T_R, rooted at R with directed edges towards the leaves.

We use T_R in the presentation of the algorithm that partitions T_R into a forest of spanning trees of the second level bubbles. Then we show how each tree of the forest is further partitioned into spanning trees of all the other levels.

The algorithm has three steps:

(1) Mark every node, P, in T_R by, M_P, the total number of processor in the subtree rooted at P. If $M_R \leq \delta x_2$ then terminate.

(2) Find a processor P with $M_P \geq x_2$, but all of whose children Q satisfy $M_Q < x_2$ and disconnect the edge leading to P.

(3) Assign T_R to be the connected tree rooted at R following (2) and goto (1).

Obviously, each edge disconnection results in a bubble of size at least x_2 and no larger than δx_2. We now show that every execution of step two results in a new bubble. Step two is executed only when $M_R > \delta x_2$; thus R must have a child Q with $M_Q > x_2$; if Q has no child with more than x_2 nodes in its subtree then the subtree rooted at Q is disconnected to form a spanning tree of a new bubble. Otherwise, let S be the child of Q with $M_S \geq x_2$ and repeat

the arguments used for Q. Since the number of nodes in T_R is finite eventually a new bubble is disconnected from T_R. The fact that any execution of step two results in a new bubble and the fact that T_R is finite implies the termination of the algorithm.

Now we show that if the number of nodes in G is at least x_2 then when the algorithm terminates $x_2 \leq M_R \leq \delta x_2$. The termination condition of step one implies that when the algorithm terminates it holds that $M_R \leq \delta x_2$. If the number of processors in G is also smaller than δx_2 then we are done. Otherwise, there exist at least one bubble partition. Examine the last partition executed at step two before the termination of the algorithm. Right before this partition M_R is greater than δx_2 and during the partition no more than $(\delta - 1)(x_2 - 1) + 1$ nodes are disconnected from T_R leaving at least $\delta x_2 - (\delta - 1)x_2 + (\delta - 1) - 1 = x_2 + \delta - 2 \geq x_2$ nodes that are connected to R.

Essentially the same algorithm is used in order to partition any bubble of level $i - 1$ into bubbles at level i. We use the portion of T_R that spans the bubble at level $i - 1$ and partition it into a forest. In order to partition the connected component that is formed by every bubble of level $i - 1$ the algorithm uses x_i instead of x_2. ■

Next we describe the distributed routing data-base by the use of the bubble partition. We use $\mathcal{B}_i$ to denote a bubble at level i. For every bubble at level k choose one node of the bubble to be a *leader*. Define the *members* of a bubble $\mathcal{B}_{k-1}$ to be all the bubble leaders of level k that reside in the connected component of $\mathcal{B}_{k-1}$. In addition, we define the nodes that reside in a k'th level bubble, $\mathcal{B}_k$, to be members of $\mathcal{B}_k$. In general any leader of a bubble $\mathcal{B}_i$, $1 < i \leq k$, is a member of a bubble $\mathcal{B}_{i-1}$ and every bubble, but the single bubble of level one $\mathcal{B}_1$, has a single leader which is one of its members.

Figure 2 depicts a partition of a graph into bubbles. First a spanning tree is constructed (the upper part of the figure) then the spanning tree is partitioned into bubbles of level two and three (lower parts of the figure). In Figure 3 we show the way the members of the bubbles are chosen. The member of a bubble at level three are the nodes that reside in the bubble (omitted from the figure description). For each bubble at level three a single leader is chosen (e.g. A, D, E) this leader is a member of the bubble of level two in which it resides. For every bubble at level three a leader is chosen among its members (e.g. D, G).

For every bubble $\mathcal{B}_i$ let $\mathcal{TB}_i$ be a representation of the spanning tree portion of the bubble[2]. In the sequel we refer to both the description of the spanning tree and the spanning tree itself by $\mathcal{TB}_i$. The spanning tree $\mathcal{TB}_i$ is a combination of the spanning trees of the bubbles of level $i + 1$'st that resides in $\mathcal{B}_i$ (in the sequel we show that this property is important for the bubble partition upon a topology change).

$\mathcal{TB}_i$ is known to every member of $\mathcal{B}_i$. In addition every member of $\mathcal{B}_i$, $k \geq i > 1$, that is not the leader of $\mathcal{B}_i$ has a path description to its bubble leader. Note that every member of $\mathcal{B}_1$ maintains $\mathcal{TB}_1$ in its memory which is a spanning tree of the entire communication graph.

[2]The representation of the spanning tree can be extended to include the full topology of the bubble by increasing the memory requirement by a factor of δ.

Routing Strategy: P delivers a message m to a processor Q by the following procedure: Let $i \geq k$ be the lowest bubble level that P is a member in and say this bubble is $\mathcal{B}_i$. P searches for Q in the spanning tree description $\mathcal{TB}_i$. If Q is not found in $\mathcal{TB}_i$ then P sends the packet $(path_L, Q, m)$ to the leader of $\mathcal{B}_i$. The leader, L, of the bubble $\mathcal{B}_i$ is a member of a bubble at level $i-1$, say $\mathcal{B}_{i-1}$. When the packet reaches L, L checks $\mathcal{TB}_{i-1}$ and sends the packet $(path_Q, m)$ to Q upon finding Q. Otherwise, when Q is not found in $\mathcal{TB}_{i-1}$ then L sends the packet $(path_{L'}, Q, m)$ to its leader L'. This procedure must terminate at a member of $\mathcal{B}_1$ since every member of $\mathcal{B}_1$ has a spanning tree description of the entire communication graph namely, $\mathcal{TB}_1$.

Centralized Bubbles Update: We now present the strategy for each topology change. The strategy is centralized as if an outside operator changes the distributed routing data-base.

We use the following definitions:

Definition 3.1 *Two bubbles $\mathcal{B}'_l$ and $\mathcal{B}''_l$ both at level l are called* tree-neighboring *bubbles if the following two conditions are satisfied:*
(1) $\mathcal{B}'_l$ and $\mathcal{B}''_l$ reside in the same bubble, say $\mathcal{B}_{l-1}$.
(2) There exists a tree link of $\mathcal{TB}_{l-1}$ that connects $\mathcal{TB}'_l$ with $\mathcal{TB}''_l$.

Definition 3.2 *The* combination *of two tree-neighboring bubbles $\mathcal{B}'_l$ and $\mathcal{B}''_l$ that belong to the same bubble $\mathcal{B}_{l-1}$ results in a bubble $\mathcal{B}_l$ that is obtained by connecting $\mathcal{TB}'_l$ and $\mathcal{TB}''_l$ using a tree-link of $\mathcal{TB}_{l-1}$. (Typically the combined bubble includes less than $\delta x_i + x_i$ nodes.)*

Definition 3.3 *The* split *of a bubble $\mathcal{B}_i$ is done according to the algorithm presented in Theorem 3.7. The algorithm chooses one of the nodes of $\mathcal{B}_i$ to be R and directs $\mathcal{TB}_i$ from R towards the leaves. Then steps (1) to (3) of the algorithm are executed using x_i instead of x_2. (Note that only a single split is executed for a combined bubble of less than $\delta x_i + x_i$ nodes.)*

As detailed below there are four possibilities for topology changes:

Link Removal: If the removal of the link does not partition the spanning tree of any bubble then no change of the bubbles routing data-base is required[3].

If the spanning tree of a bubble is disconnected then the link removal is handled at each level starting with level one. Let $\mathcal{B}_l$ be the bubble whose tree $\mathcal{TB}_l$ has been disconnected. $\mathcal{TB}'_l$ and $\mathcal{TB}''_l$ denote the two portions of the broken $\mathcal{TB}_l$, while $\mathcal{TB}'_{l-1}$ and $\mathcal{TB}''_{l-1}$, respectively, denote the trees in which $\mathcal{TB}'_l$ and $\mathcal{TB}''_l$ reside. Each of these portions is combined with a tree neighboring bubble. Then the combined bubble is split if it include more than δx_l members. The members of $\mathcal{B}_l$ and the two tree neighboring bubbles have to

[3]If $\mathcal{TB}_i$ is extended to include the full topology of the bubble then a change takes place at every $\mathcal{TB}_i$ of any member of a bubble to which the removed link belonged. Since each link belongs to at most k bubbles the number of members that are updated is less than k times the maximal number of members e.g. for $x_i = n^{(k-i+1)/k}$ the total number of processors that need to update their routing data-base is $k\delta n^{1/k}$.

update the spanning tree description of their bubble. Note, that the number of members of $\mathcal{B}_l$ together with the members of the two tree neighboring bubbles is at most $3\delta x_l/x_{l+1}$.

Now we analyze the number of processors that have to be updated in level $l+1$. The removal of the link may disconnect a bubble at level $l+1$ as well. In addition at most two split operations take place at level l. Since the bubbles at level $l+1$ that resides in a bubble $\mathcal{B}_l$ are connected components of $\mathcal{TB}_l$ it holds that a split of $\mathcal{TB}_l$ causes at most one split of a bubble at level $l+1$. Thus, there are at most three tree links that disconnect bubbles at level $l+1$. For each of these links we use the same procedure we used for the single link removal at level l. Thus, the number of processors that have to update their routing data-base is no more than $9\delta x_{l+1}/x_{l+2}$.

Similarly, the number of links that are removed at level $l+i$ is 3^i and the number of processors that have to update their data-base is $3^i\delta x_{l+i}/x_{l+i+1}$ (where $l+i \leq k$ and $x_{k+1}=1$).

Link Addition: An addition of a link that does not connect two previously separated connected components does not change the routing data base. An addition of a link that does connect two previously disconnected components G and G' is handled as follows. Let $\mathcal{TB}'_1$ be the spanning tree of G before the link addition and let $\mathcal{TB}''_1$ be the spanning tree of G' before the link addition. Connect $\mathcal{TB}'_1$ with $\mathcal{TB}''_1$ using the new link to form a spanning tree $\mathcal{TB}_1$ of the new connected communication graph. Update every member of $\mathcal{B}_1$ with the new $\mathcal{TB}_1$. Continue with levels two up to k one at a time. For level $2 \leq i \leq k$ if the new link connects two legal bubbles then no update operations are triggered by level i. Otherwise, when the new link connects at least one illegal bubble then combine the illegal bubble with the new tree neighboring bubble and split the combined bubble if necessary. The split operation may result with a link removal at the next level. This is handled by the link removal procedure described above. Then the update for level $i+1$ may be started.

Node Addition or Removal: Both the addition and the removal of a node can be described in terms of addition and removal of links. The addition is handled by first adding a link that connects two separated connected components one of which is the single node. Then adding the rest of the links one by one. Node removal is done by removing one link at a time and then removing the node.

The next three lemmas state the stretch factor, the memory requirement and the adaptability of the bubble routing scheme. For these lemmas we use $x_i = n^{(k-i+1)/k}$.

Lemma 3.8 *The stretch factor of the bubble routing scheme is bounded by k.*

Proof: In the worst case a processor P that is only a member in level k sends a packet to its bubble leader, Q, that is a member at level $k-1$, and so on. With no more than $k-1$ packets a member of level 1 is reached; this member knows $\mathcal{TB}_1$ and sends a direct packet to the destination of the message. ■

Lemma 3.9 *The memory requirement for the bubble routing scheme is bounded from above by* $k\delta n^{1+1/k}(\log n + \log \delta) + (k-1)n^2 \log \delta$.

Proof: Every processor maintains a spanning tree of the lowest bubble it is a member of. The members of bubbles at level i maintain a spanning tree of at most $\delta n^{(k-i+1)/k}$ nodes. This requires $\delta n^{(k-i+1)/k}(\log n + \log \delta)$ bits. Since there are at most $n^{i/k}$ members at level i the memory requirement for the i'th level is $\delta n^{(k-i+1)/k}(\log n + \log \delta)n^{i/k} = \delta n^{1+1/k}(\log n + \log \delta)$ bits.

Each non-leader processor in G_i has a path of at most n hops to its leader. This requires $n \log \delta$ bits. Note that members at level one does not need a path to their leaders. Thus, the total memory used is bounded from above by $k\delta n^{1+1/k}(\log n + \log \delta) + (k-1)n^2 \log \delta$. ■

Lemma 3.10 *The adaptability of the bubble routing scheme is bounded from above by* $k3^{k-1}\delta^2 n^{1/k}$.

Proof: Obviously, the number of processors that change their routing data-base is greatest when a processor is removed. This number is bounded from above by the effect of removal of δ links one at a time. The removal of a link that partitions a spanning tree of a bubble at every level implies the largest number of updates. Such a link removal requires at most $n^{1/k}$ updates of the members of $\mathcal{B}_1$. We continue our analysis according to the description of the link removal procedure. The members of 3^{i-1} bubbles at level i have to change their routing data-base. Thus, the number of updates at each level is no more than $3^{k-1}\delta n^{1/k}$. Therefore, the total number of processors that are updated upon a node removal is no more than $k3^{k-1}\delta^2 n^{1/k}$. ■

The following theorem summarizes the properties of the bubbles routing scheme.

Theorem 3.11 *The bubble routing scheme has the following properties:*
(1) Stretch factor $= k$.
(2) Memory requirement $= k\delta n^{1+1/k}(\log n + \log \delta) + (k-1)n^2 \log \delta$.
(3) Adaptability $= k3^{k-1}\delta^2 n^{1/k}$. ■

4 Lower Bounds

In this section we prove a lower bound on the adaptability for graphs with bounded degree δ and stretch factor k.

Given any source-oblivious routing scheme with stretch factor k we build a spanning tree T_k as follows. The root of the spanning tree is a processor P. Assume that every processor is to send a message m to P and connect each processor Q with the destination of its first packet when Q sends m to P. The *depth* of a tree is the maximal number of edges from the root to a leaf.

Claim 4.1 *T_k is of depth no more than k.*

Proof: Otherwise the stretch factor is greater than k. ■

Claim 4.2 *There exists at least one node in T_k with at least $n^{1/k}$ subtrees.*

Proof: There are n processors in T_k and the depth of T_k is k. Thus, by the pigeon-hole principle there must be a node with at least $n^{1/k}$ subtrees. ■

Theorem 4.3 *Any source oblivious routing scheme with stretch factor k has adaptability $\Omega(n^{1/k})$ in a communication graph of constant degree.*

Proof: Let Q be a processor that has at least $n^{1/k}$ subtrees in T_k. Q is connected to the rest of the processor in G by at most δ links. Thus, by the pigeon-hole principle there must be a link, (Q, R), used by at least $n^{1/k}/\delta$ processors in sending their first packet to carry the message to P. We now show a sequence of at most three topology changes that implies $n^{1/k}/(3\delta)$ adaptability. If a disconnection of (Q, R) does not partition the communication graph then obviously $n^{1/k}/\delta$ processors must change their routing data base.

Now we analyze the case in which (Q, R) does partition the graph. We need to show that there exists a sequence of changes that preserve the bound on the degree δ and force an omission of the link (Q, R) from the routing tables. We claim on two different cases.

Case 1: Before the disconnection of (Q, R) G is a tree and the number of processors is greater than two. In this case following the disconnection of (Q, R) there must be at least one leaf $X \notin \{Q, R\}$, say w.l.o.g. in the connected component of Q. The connection of X to R is possible without exceeding δ.

Case 2: Before the disconnection of (Q, R) G contains at least one cycle. In this case following the disconnection one of the connected components must contain a cycle (otherwise no partition would take place). Let (X, Y) be a link in this cycle. W.l.o.g assume that (X, Y) is in Q's connected component and $X \neq Q$. Disconnect (X, Y) and connect X to R. Again the connection of X to R is possible without exceeding δ.

Thus at most three topology changes cause $n^{1/k}/\delta$ processors to change their routing data-base. ■

5 Distributed Bubbles Update

So far we have not been concerned with the issue of how to distribute the bubbles update algorithm. Still the above upper bound for the adaptability of the bubble routing scheme is useful

Figure 2: Bubble Partition $k = 3$

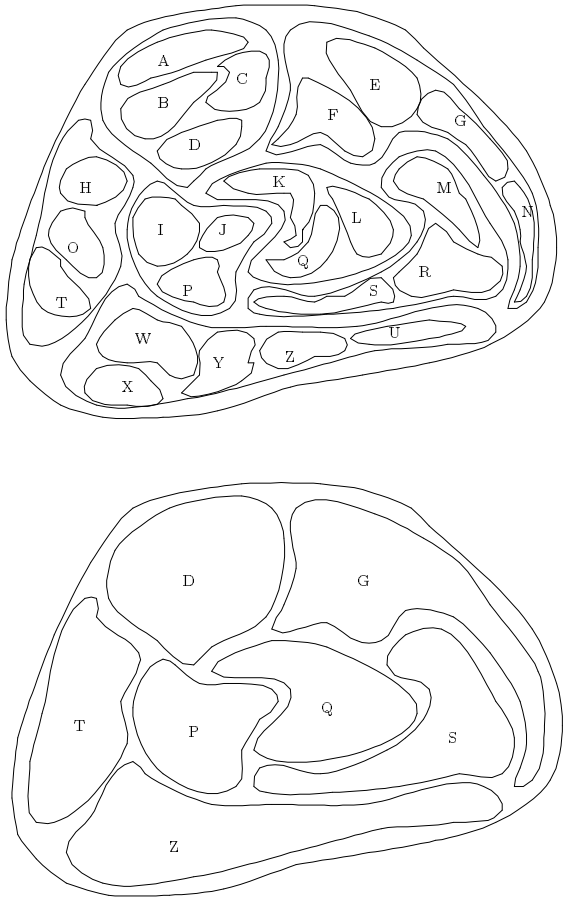

Figure 3: The members at level 2 and level 1

for the case of manually adding and removing links and nodes. This is the case for telephone networks where new users may be connected and existing users may be disconnected. However, our results are made stronger by handling automatically topology changes while keeping the low adaptability. For the sake of keeping a low adaptability we introduce a distributed bubbles update algorithm that can cope with transient failures and recoveries as well as permanent disconnections and connections.

The distributed bubble update uses additional features of the switching subsystem. Each link connected to the switching subsystem has, in addition to the unique link label, a set of labels that may be controlled by the node control unit. It is also assumed that each switching subsystem has a *sack variable* which may be modified by the node control unit. This sack variable can be collected by a special arriving message. We also assume that an arriving message may be concatenated with the label of the link through which it arrived. Note that no direct modification of the set of labels or the sack variable by an arriving message is allowed.

The main idea for the distributed bubble update is to have the processors neighboring a topology change to be the *monitor* of the change. The task of the monitor is (1) to collect information on the tree structure it belongs to and on the existing routing data-base, (2) to try to merge with other neighboring trees and (3) to modify the existing routing data-base to fit the current topology. The modification should be performed in a way that involves the least number of node control units.

Towards this end we assume that the sack variable of every processor contains its portion of the distributed routing data-base. The spanning tree link $\mathcal{TB}_1$ is distributively marked on edges of the system. The two end points of a tree link are marked by a label T. In the sequel, we use the term *marked tree* to be the graph obtained by the set of marked links and the node they connect. Upon failure of a tree links the marked tree of a connected component may essentially be a marked forest that has to be fused to a new marked tree. Each tree in the forest is an autonomous entity with a monitor. The monitors of trees in the forest negotiate in order to promote a non tree link into a tree link. Upon such a promotion the number of trees (and monitors) in the forest is reduced by one. More details follow.

A monitor is associated with the time of the topology change that created it. We assume the existence of a synchronized clock at every processor. Each monitor uses the marked tree it belongs to for *tree broadcast with feedback* (in short TBF). The TBF collects the information in the sacks of every processor that belongs to the tree and delivers the information to the monitor. The monitor initiates the TBF by sending a message with its identifier, the timestamp of its creation and a TB (tree-broadcast) label on every marked tree link. Every switch that receives a message m of type TB concatenates the unique label of the link through which m arrived to the end of the message and forwards a copy of the message to each link that is labeled T except the link through which m arrived. If no other tree link exists (the switch is a leaf in the marked tree) then the switch modifies the message as follows: TB is replaced by TF (tree-feedback) the labels at the end become the address of the message (that transfer it back to the TBF initiator), and the content of the sack is concatenated to the end of the message. When a switch receives a TF message it removes the first label of the message arrived and

concatenates the value of its sack to the tail of the message.

A processor P starts to act as a monitor when: Either one of the tree links attached to it fails or an attached link recovers. A monitor marks the label that leads from its switching subsystem to itself by M. This enables the switching subsystem to deliver a copy of every BF message to its node control unit when the node control unit is a monitor. Then the monitor waits for a period of time that ensures the completion of every ongoing corrections and collects information on its tree and the distributed routing data-base of this tree. This is done by the tree broadcast with feedback mechanism. Whenever a tree broadcast with feedback of another monitor arrives to P and the timestamp of the arriving TBF is greater (breaking ties by the monitors identifiers) then P stops being a monitor.

Upon receiving the information from the tree broadcast with feedback P checks whether there are neighboring processors that do not belong to its tree. If such neighboring processor exists P sends a *promote* message to the processor Q attached to the link with the highest lexicographic order[4] that leads to a neighboring tree. Q becomes the monitor and enquires the other side on the possibility to merge. Upon receiving an accept message the link changes its status to a tree link and the monitor with the smaller identifier sends the information collected in its TBF to its neighbor. At the same time it stops being a monitor. When there is no further possibilities to merge the corrections of the routing data base is sent to the appropriate node control units, and the processor that acted as a monitor resumes operation as a regular processor.

Lemma 5.1 *In every instance of time and for every connected component, if the marked tree does not span the entire connected component then for each tree in the forest there exists at least one monitor.*

Proof: This is true in the first instance of the system. Further topology changes keep this invariant. A monitor stops existing only when there exist another monitor in the same tree or when the marked tree spans the entire connected component. ■

Lemma 5.2 *If no monitor exists in a connected component then the distributed routing data-base of this connected component is correct.*

Proof: For every connected component of more than a single processor there has been an instance in which one processor of this connected component has been a monitor. The instance that follows a connection of any two neighboring processors by a link. The last processor that stopped being a monitor in this connected component must have finished the right corrections of the distributed routing data-base. ■

[4]The name of a link is defined by the identifiers of the two processors that are attached to it. The first identifier in the pair is always greater than the second. Then the names of the links may be lexicographically ordered in a natural way.

Lemma 5.3 *Some time following the last topology change no monitor exists.*

Proof: The tree broadcast with feedback terminates. Then the monitor may wait for a promotion of a link into a tree link. Assume towards contradiction that the monitor waits for some link promotion forever. This in turn implies that the monitor of the other tree portion is waiting on another edge that has greater identifier. Since links that are to be promoted into tree links are chosen according to lexicographical order this chain of waiting monitors is finite and must end with two monitors waiting on the same edge. ■

Theorem 5.4 *The number of node control units that participate in a routing update upon a single topology change is of the adaptability order.*

Proof: Only monitors and the node control units that have to change their distributed routing data-base participate in the distributed bubbles update. The number of monitors is no more than 2δ for a topology change: At most δ are directly influenced and another δ are because of monitors migration.

Note that during the correction process of the distributed routing data-base by one monitor a topology change might take place stopping the update before its completeness. Never the less, the partial correction process took place in a limited number of bubbles (as explained for the centralized case) leaving most of the bubbles unaffected. Thus, the number of node control units that have to change their distributed routing data-base following c topology changes is $O(ck3^{k-1}\delta^2 n^{1/k})$.

The theorem is proved since the additional $O(\delta)$ monitors does not change the $O(k3^{k-1}\delta^2 n^{1/k})$ adaptability. ■

6 Concluding Remarks

In this paper we defined new measures for the efficiency of routing schemes for high-speed dynamic networks. We demonstrated the applicability of these measures by presenting the bubbles routing scheme. Many variants of our bubble routing scheme are possible. We now mention some of them.

- As shown in the analysis of the topology changes (in Section 3.3) the number of bubbles to be updated upon a topology change grows rapidly with the level index. In order to improve on the complexity of the adaptability (as a function of k), one would choose to have bubbles of different number of members at each level — the number of members shrinks for higher levels. When $x_i = \lceil 1/3^{i-1} n^{(k-i+1)/k} \rceil$ the number of bubble members that are effected at each level is the same. Resulting in an $O(k\delta^2 n^{1/k})$ adaptability.

- For certain graphs a different bubbles construction may fit better. These graphs are characterized by their small (e.g. constant) *bubble degree* denoted by π. The *bubble degree* of a graph G is the maximal number of links, over all possible connected components of G, that connects a node in the connected componet with a node outside the connected componet. One obvious example for a graph with small π is a ring or a chain (dynamic networks with $\delta = 2$). In such a case the graph may be partitioned first into bubbles at level k. This partition defines a new graph in which each cluster is represented by a node and two nodes are connected by an edge if they belong to neighboring bubbles (bubbles that have members that are connected by a physical link). By the fact that the bubble degree is π the new graph may be partitioned into bubbles of size x_{l-1} to πx_{l-1}. This procedure may repeat itself for levels with smaller index. When such a partition is possible the number of bubbles that are influenced by a topology change is at most 3 at each level.

- Note that the routing strategy presented in section 3.3 essentially uses a single spanning tree in order to communicate. However, by extending the spanning trees representations into the topologies of the bubbles we can achieve better distribution of the communication (choose randomly a path to the leader or to the destination). Note that the complexity of the adaptability will not be changed.

- For some cases the bubbles partition might be restricted due to other constraints e.g., geographical constraints. For instance, one would not like to have two bubbles (say at level two) to cover the network in the USA, such that one of the bubbles includes Japan and the other includes England, instead a single bubble for the USA is preferred. Our bubble partition can take into account such considerations during the partition by ignoring some of the communication links during the bubble partition in some levels.

- In some cases, the bubble partition can be used in a graph with more than δ links per a processor. Roughly speaking the algorithm marks at most δ links per a processor and uses the marked links for the bubble partition. In some cases, a link might become marked upon a disconnection of the marked graph. This link should connect separated connected components of the marked graph and still should not violate the δ upper bound.

References

[AGR89] Y. Afek, E. Gafni, and M. Ricklin. "Upper and lower bounds for routing schemes in dynamic networks," *Proc. 30th Symp. on Foundations of Computer Science*, pp. 370-375, 1989.

[Aw85] B. Awerbuch. "Complexity of network synchronization," *J. of the ACM*, 32(4):804-823, October 1985.

[AB+89] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. "Compact Distributed Data Structure for Adaptive Routing," *Proc. 21th ACM Symp. on Theory of Computing,* pp. 479-489, 1989.

[AB+90] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. "Improved routing strategies with succinct tables," *J. of Algorithms*, 11:307-341, 1990.

[AP92] B. Awerbuch and D. Peleg. "Routing with polynimial communication-space tradeoff," *SIAM J. on Discrete Math,* 5(2) pp. 151-162, 1992.

[CG88] I. Cidon and I. Gopal, "PARIS: An approach to private integrated networks," *Journal of Analog and Digital Cabled Systems* 1(2), pp. 77-86, 1988.

[GZ94] O. Gerstel and S. Zaks, "The Virtual Path Layout Problem in Fast Networks," *Proc. 13th ACM Symp. on Principles of Distributed Computing*, to appear, 1994.

[PU89] D. Peleg and E. Upfal. "A tradeoff between size and efficiency for routing tables," *J. of the ACM,* 36:510-530, 1989.

[SK85] M. Santoro and R. Khatib. "Labeling and implicit routing in networks," *The Computer Journal,* 28:5-8, 1985.

[Va81] L. G. Valiant. "Universality consideration in VLSI circuits," *IEEE Transactions on Computers,* 30, 135-140, 1981.

[vLT86] J. van Leeuwen, and R.B. Tan. "Routing with compact routing tables," *The Book of L*, G. Rozenberg and A. Salomaa, eds. Springer-Verlag, New York, 1986, pp. 259-273.

[vLT87] J. van Leeuwen, and R.B. Tan. "Interval Routing," *The Computer J.*, 30 (1987), 298-307.