A Generic Scheme for Building Overlay Networks in Adversarial Scenarios

Ittai Abraham
School of Engineering and Computer Science
Hebrew University
ittaia@cs.huji.ac.il

Baruch Awerbuch
Department of Computer Science
Johns Hopkins University
baruch@acm.org

Yossi Azar
Department of Computer Science
Tel Aviv University
azar@post.tau.ac.il

Yair Bartal*
School of Engineering and Computer Science
Hebrew University
yair@cs.huji.ac.il

Dahlia Malkhi
School of Engineering and Computer Science
Hebrew University
dalia@cs.huji.ac.il

Elan Pavlov
School of Engineering and Computer Science
Hebrew University
elan@cs.huji.ac.il

Abstract

This paper presents a generic scheme for a central, yet untackled issue in overlay dynamic networks: maintaining stability over long life and against malicious adversaries. The generic scheme maintains desirable properties of the underlying structure including low diameter, and efficient routing mechanism, as well as balanced node dispersal. These desired properties are maintained in a decentralized manner without resorting to global updates or periodic stabilization protocols even against an adaptive adversary that controls the arrival and departure of nodes.

1 Introduction

Overlay networks are employed in many settings to provide logical communication infrastructure over an existing communication network. For example, Amir et al. use in [1] an overlay network for wide area group communication; many ad hoc systems use overlay routing for regulating communication (see [17] for a good exposition); Kleinberg explores in [8] routing phenomena in natural worlds using random long-range overlay edges; and recently, much excitement revolves around peer-to-peer schemes that utilize an overlay routing network to discover and search resources in highly dynamic environments, e.g., [5, 9, 11, 13, 14, 16, 18]. The exploration of overlay networks deviates from standard research in routing networks in its attention to scale, dynamism of the network structure and the lack of centralized control.

One of the main motivations stimulating this work is recent interest in using overlay networks for lookup in peer-to-peer (P2P) settings. The routing network is used for storing and searching a distributed hash table. A distributed hash service is a fundamental tool for supporting large peer-to-peer applications, that may support efficient storage and retrieval of shared information for cooperating distributed applications. Examples of contemporary stellar services that may benefit from it are file sharing systems such as Freenet [4], and music sharing systems, e.g., Gnutella [6].

Work todate on P2P overlay networks (e.g., [5, 9, 11, 13, 12, 14, 16, 18]) employs randomization to achieve uniform dispersal of hash values among peers and for building the routing topology. There are several problems that result from the reliance on randomization: First, a random distribution of hash values creates with high probability load imbalance among peers of up to a logarithmic factor (see e.g., [7, 16, 9]). Second, over a long period of time, the departure and addition of peers may impair the randomization of initial selections, and result in poor balance in such systems. In particular, node departures might be correlated due to failures or due to the banning of a P2P service from a particular organization. Lastly, uniformity by randomization is sensitive to adversarial intervention through peer removal and/or joins.

Our aim in this work is to enhance the technology for overlay networks in several important ways. First, our

*Research supported in part by the Israel Science Foundation (195/02).

overlay network maintains its desired properties for efficient routing even against *adversarial* removal and additions of nodes. Second, it maintains load balance among peers. Both of these desirable goals are achieved with no global coordination, using localized operations with reasonable costs. Third, our techniques are of interest in themselves. They provide an insight that links the issues of load balance and resilience in overlay networks with tree balancing. Lastly, our techniques are generic, and are applicable to most known network topologies, including the hypercube, the De Brujin bitonic network, the Butterfly network, and others. We provide a characterization of the families of graphs that can make use of our approach.

There are thus two concrete angles in which our work compares favourably with previous works: The load balance and fault tolerance. With respect to load balancing, we first note that any method that preserves initial peer distribution choices cannot be resilient to adversarial removal and addition of nodes. The only previous work that we are aware of that allows peer re-positioning is CAN [13]. In CAN, a background stabilization process is employed in order to recover balance, introducing a constant overhead. In contrast, our method does maintain load balance against adversarial settings, but incurs only local cost per join/leave operation, and maintains the desired balance properties immediately.

The second facet in which our work enhances the technology is in its resilience to adversarial scenarios. Our overlay networks can withstand node removals and additions even when done by a malicious adaptive adversary. Most previous works, with the exception of [15, 5], do not attempt to address a malicious adversary. Consequently, their performance may be signficantly degraded, e.g., as a result of removal of servers concentrated in one part of the network. Additionally, random failures and departures are handled, e.g., in [13, 16], via global overhaul background mechanisms whereas our method has no global operations. The censorship resistant network of [15, 5] is designed to cope with malicious removal of up to half of the network nodes. In contrast to our scheme, it is designed with a rough apriori knowledge of the number N of participants, and with the assumption that the actual number of peers is within a known linear envelope of N. Additionally, randomization is relied upon in node joining.

Our approach to generic overlay emulation is as follows. We consider a graph topology to be a family of graphs $\mathcal{G} = \{G_1, G_2, ...\}$ for a monotonically increasing system sizes. We observe that most families of graphs may be emulated by viewing the dynamic overlay construction process as a virtual tree process, in which new nodes join at the leaves. Each member G_i of the graphs family naturally maps to layer i of the tree. We provide a scheme for keeping a dynamic graph in which nodes on different levels of the tree co-exist simultaneously.

More specifically, we make use of a view suggested originally in [13] to represent the overlay construction process as a *dynamic tree*. The process adds and removes nodes to a tree, such that inner vertices represent nodes that no longer exist (they were split), and the leaves represent current nodes. In order to maintain the dynamic tree, when a node joins the network, it chooses some location to join and "splits" it into leaves. To the contrary, when a node leaves the network, it finds a full set of siblings and "merges" them into a single parent. The branching factor of the tree is set so that each tree layer corresponds to one member G_i. If the tree is balanced, we can easily overlay the leaves of the tree with G_i and be done. Generally, the tree will not be balanced. In fact, at the very least if the number of nodes does not match any tree layer, then the highest tree level is not full. Hence, we need to build an overlay network that, though inspired by the simple-level overlay approach, connects leaves on different levels.

In order to maintain an overlay graph over an unbalanced tree, we first need to require that G_i's exhibit certain recursive structure: G_{i+1} are mapped onto G_i via a *parent* function, such that the neighboring relation in G_{i+1} induced neighborhoods on the parents in G_i. We call such families of graphs *child-neighbor commutative* (precise definition is given below). We further connect the edges of every leaf at level i to either the parents or the children of its would-be end-points at level i, whichever exists.

Using this construction, we prove that the routing properties of the overlay network are related directly to the *gap* in levels in the resulting dynamic tree. It is worth noting that one could keep the tree balanced (inevitably, up to a highest, unfull level) as follows: All entries would occur at the 'step' position at the highest level. However, this approach requires serializing all entries and creates an unacceptable contention point for very large systems.

This leads us to construct several strategies for balancing the dynamic tree. The first is a localized, deterministic balancing scheme, that guarantees even against a malicious scheduler that the level-gap of the tree remains bounded by the diameter of the smallest graph that could fit the existing nodes. We further show that the diameter is bounded by that of the highest tree level, and hence, by the gap bound, it is also bounded. The second is a a randomized balancing strategy. The randomized balancing strategy makes use of balanced allocation techniques of Azar et al. [2] and extension [10] to guarantee probabilistically that the gap in levels is constant. The diameter is consequently appropriately bounded. In order to make use of balanced allocation, we need to extend known results to analyze the *emptiest*, rather than the *fullest*, bin in a balanced allocation process.

Overlay networks are used for reliable and efficient message dissemination as well as for routing and searching. For the latter, we finally show a generic routing strategy that makes use of the underlying graph-family routing strategy, and finds routes that are within our proven diameter bound.

The rest of this paper is organized as follows. Preliminaries and notation are exhibited in Section 2. The dynamic graph process is defined in Section 3, and is exemplified with several families of graphs, including the hypercube and the de Bruijn networks. Balancing methods are presented in Section 4. The properties of balanced dynamic graphs are proven in Section 5. Finally, routing is discussed in Section 6.

2 Preliminaries and Notation

Consider a family of directed graphs $\mathcal{G} = \{G_1, G_2, G_3, \ldots\}$, where $G_i =< V_i, E_i >$. Our interest is in families that have a recursive structure, and hence, we first require that all the nodes of the graph G_i can be mapped to nodes of the graph G_{i-1} using a *parent* function $p_i : V_i \rightarrow V_{i-1}$. Denote $\mathcal{P} = \{p_2, p_3, \ldots\}$ the set of parent function for $\mathcal{G}$. Since all p_i's have disjoint input domains (likewise, output domains), there should be no confusion when omitting the index of a parent function, and hence we simply use $p()$.

Second, we require that for every i, every node $u \in G_{i-1}$ has at least two nodes $v, w \in G_i$ such that $p(v) = p(w) = u$. Denote the inverse of the parent function, the *child* function as $c_i : V_i \rightarrow 2^{V_{i+1}}$, where $u \in c_i(v) \Leftrightarrow p_{i+1}(u) = v$. We have that $\forall u \in V_i : |c_i(u)| \geq 2$. Here again, we omit the index of a child function and simply use $c()$.

For a set of nodes $X \subseteq V_i$ define $p(X) = \bigcup_{x \in X} p(x)$, and $c(X) = \bigcup_{x \in X} c(x)$. Define the *siblings* of $u \in V_i$ as $s(u) = c(p_i(u))$. For a graph $G =< V, E >$ and a set $X \subseteq V$ define $\Gamma_G(X) = \{y | \exists x \in X \wedge (x, y) \in E\}$, when G is obvious from the context, we omit it.

We will focus on a particular group of graphs and parent functions having the following recursive nature:

DEFINITION 2.1 (The child-neighbor commutative property.) *A family of graphs and child/parent functions $(\mathcal{G}, \mathcal{P})$ is said to have the child-neighbor commutative property if for all i and for all $u \in V_i$: $\Gamma_{G_{i+1}}(c(u)) = c(\Gamma_{G_i}(\{u\}))$.*

Let us consider some example families of graphs to clarify the definitions. Note that in the following examples the edges are *directed*.

Example 1 (The Hypercube.) *The hypercube $HC_i =< V_i, E_i >$ is a graph $V_i = \{0,1\}^i$ with 2^i nodes, namely all the binary strings of length i. Node $\langle a_1, \ldots, a_i \rangle$ has an edge to node $\langle b_1, \ldots, b_i \rangle$ if and only if there exists $1 \leq j \leq i$ such that $a_j \neq b_j$ and for all $k \neq j$: $a_k = b_k$. Consider the parent function $p_i(\langle a_1, \ldots, a_{i-1}, a_i \rangle) = \langle a_1, \ldots, a_{i-1} \rangle$.*

Lemma 2.1 *$\{HC_i\}$ and $\{p_i\}$ have the child-neighbor commutative property.*

The next example we consider is the de Bruijn network [3].

Example 2 (The de Bruijn graph.) *The de Bruijn $DB_i =< V_i, E_i >$ is a graph with 2^i nodes, $V_i = \{0,1\}^i$. Node $\langle a_1, \ldots, a_i \rangle$ has an edge to node $\langle b_1, \ldots, b_i \rangle$ if and only if for all $1 \leq j \leq i-1$: $a_{j+1} = b_j$. Thus every node $\langle a_1, \ldots, a_i \rangle$ has two outgoing edges to nodes: $\langle a_2, \ldots, a_i, 0 \rangle$ and $\langle a_2, \ldots, a_i, 1 \rangle$ (shuffle, then choose the last bit). Again, consider the parent function $p_i(\langle a_1, \ldots, a_{i-1}, a_i \rangle) = \langle a_1, \ldots, a_{i-1} \rangle$.*

Lemma 2.2 *$\{DB_i\}$ and $\{p_i\}$ have the child-neighbor commutative property.*

For every node u of any $\mathcal{G}$ we define its *level* as the index i of the graph G_i it belongs to, formally $\ell(u) = i \Leftrightarrow u \in V_i$. We say that u in an *ancestor* of v if $\ell(u) < \ell(v)$ and $p^{\ell(v)-\ell(u)}(v) = u$ (where $p^k(u) = p(p^{k-1}(u))$). We also say that u is a *descendant* of v if v is an ancestor of u. The diameter of a graph G is denoted $diam(G)$.

3 The Dynamic Graph

In this section, we introduce an algorithm for maintaining a dynamic overlay network that derives its characteristics from a family of static graphs $\mathcal{G}$. Our goal is to make use of a family of graphs as above in order to maintain a dynamic graph that nodes can join and leave. Intuitively, this works by having each node join some location at G_i by *splitting* it into a set of children at G_{i+1}, and vice versa for leaving. However, this means that at any moment in time, different nodes may be in different G_i's. We therefore specify how to connect nodes from different G_i's in our dynamic overlay network. Unless mentioned otherwise, the nodes and edges refer to the dynamic graph.

Given a family of graphs $\{G_i\}$ and parent functions $\{p_i\}$ with the child-neighbor commutative property as defined above, we now define the dynamic as follows:

DEFINITION 3.1 (Dynamic graph.) *Graph $D =< V, E >$ is a* dynamic graph *for a child-neighbor commutative pair $(\mathcal{G}, \mathcal{P})$ if it has the following properties:*

1. *$V \subset \cup_{i=1}^{\infty} V_i$.*
2. *If $v \in V$ then no ancestor of v exists in V.*
3. *For all $u \in V$ and for all v such that $(u, v) \in E_{\ell(u)}$ then either:*

   (a) $v \in V$ and u has an edge to v.

(b) $v \notin V$ *but some* ancestor $\hat{v}$ *of* v *exists in* V*. In this case* u *has an edge to* $\hat{v}$*.*

(c) $v \notin V$ *but* v *has some descendants in* V*. In this case* u *has an edge to all of* v*'s descendants that are nodes in* V*.*

The nodes of the dynamic overlay graph can be thought of as the leaves of a tree. The inner vertexes represent nodes that no longer exist (they were split), and the leaves represent current nodes. In order to maintain the tree, when a node joins the network, it chooses some location to join and "splits" it into leaves. On the other hand, when a node leaves the network, it finds a full set of siblings and "merges", switches location with one sibling, and merges the remaining subset into a single parent. In the next section we will present algorithms that use the basic split and merge operations while keeping the dynamic graph balanced.

More precisely, We now define the dynamic graph as a process of split and merge operations as follows: The dynamic graph starts as G_1. The graph can change from $D =< V, E >$ into $\hat{D} =< \hat{V}, \hat{E} >$ by one of the two basic operations:

1. Split: For any $u \in V$, the node u is split into $c(u)$, i.e., $\hat{V} = V \setminus \{u\} \cup c(u)$.
2. Merge: For any $u \in V$ if $s(u) \subset V$ then all nodes $s(u)$ merge and form the node $p(u)$. Formally, $\hat{V} = V \setminus s(u) \cup \{p(u)\}$

The change from E to $\hat{E}$ is as follows:

1. Split of node u into nodes $c(u)$:
   (a) For every $v \in c(u)$, and every $w \in \Gamma_{G_{\ell(u)}}(v)$, connect v to w, or to w's ancestor, or to all of w's descendants (whichever exists in D).
   (b) For every node x that had an edge to u, then if $\ell(u) \geq \ell(x)$ then connect x to each node of $c(u)$. Otherwise, if $\ell(u) < \ell(x)$ then due to the child-neighbor commutative property there exists some $\bar{u} \in G_{\ell(x)}$ that is a descendant of u such that $(x, u) \in E_{\ell(x)}$. Find the node $\bar(x) \in c(x)$ that is either $\bar{u}$ or an ancestor of $\bar{u}$ and connect x to $\bar{u}$
2. Merge of nodes $c(u)$ into node u:
   (a) For each $w \in \Gamma_{G_{\ell(u)}}(u)$, connect u to w, or w's ancestor, or all of w's descendants (whichever exists in D).
   (b) For each node x that had an edge to a node $\bar{u} \in c(u)$, connect x to u.

For example, Figure 1 shows a merge and a split operation on a dynamic hypercube.

It is easy to see that the split and merge operations keep the dynamic graph properties above.

4 Balancing Strategies

In this section, we introduce strategies for choosing joining and leaving positions in the dynamic graph so as to keep it balanced. Our goal is to keep the dynamic graph's tree balanced at all times, i.e., to minimize the level gap among nodes that belong to different G_i's. Intuitively, the reasons for this are two-fold. First, each G_i has certain desirable characteristics of diameter and routing complexity. By keeping the level-gap minimized, we can keep these properties to some degree in the dynamic graph despite the level gap. Second, the gap in levels also represents gap in load incurred on each node, e.g., by routing. Naturally, low level gap results in better load balance.

We first introduce some notation. The *local gap* of a node $v \in V$ is the maximum difference between its level to the levels of its neighbors, $gap(v) = \max_{i \in \Gamma_D(v)} |\ell(i) - \ell(v)|$. The *local gap* of a dynamic graph D is the maximum local gap over all nodes $v \in V$. Formally $localGap(D) = \max_{v \in V} gap(v)$. Similarly, the global gap of the dynamic graph D is the maximum difference between the levels of any two nodes and is defined as $globalGap(D) = \max_{i,j \in V} |\ell(i) - \ell(j)|$. We present two algorithms, a deterministic algorithm against adversarial additions and removals of peers that maintains a local gap of 1 and a randomized algorithm against a random series of additions of peers that maintains a global gap of $O(\log \log n)$ w.h.p.

4.1 Deterministic Balancing

Consider the following model: The algorithm and adversary take turns. At the adversary's turn, he may choose to add one node and provide an access node, or choose one node to be removed. At the algorithm's turn, he may use some computation and message passing and eventually rebalance the graph by executing a merge or a split operation.

For simplicity, we present balancing algorithms for binary dynamic graph trees, i.e. $\forall u : |c(u)| = 2$. The full paper will include the generalized algorithm for any order of $c()$.

1. Re-balancing a node addition, given a new node p and an access node u. Begin at node u, as long as there is an edge toward a lower level node follow that node, until a node v is reached with gap at most 1 and no lower level neighbors. Add the new node p by splitting node v.

2. Re-balancing a node removal, given the removed node u. Begin at node u, as long as there is an edge toward a higher level node, or there is a sibling node on a higher level, follow that node. Eventually, two siblings s_1, s_2 at the same level with no higher level edges will be

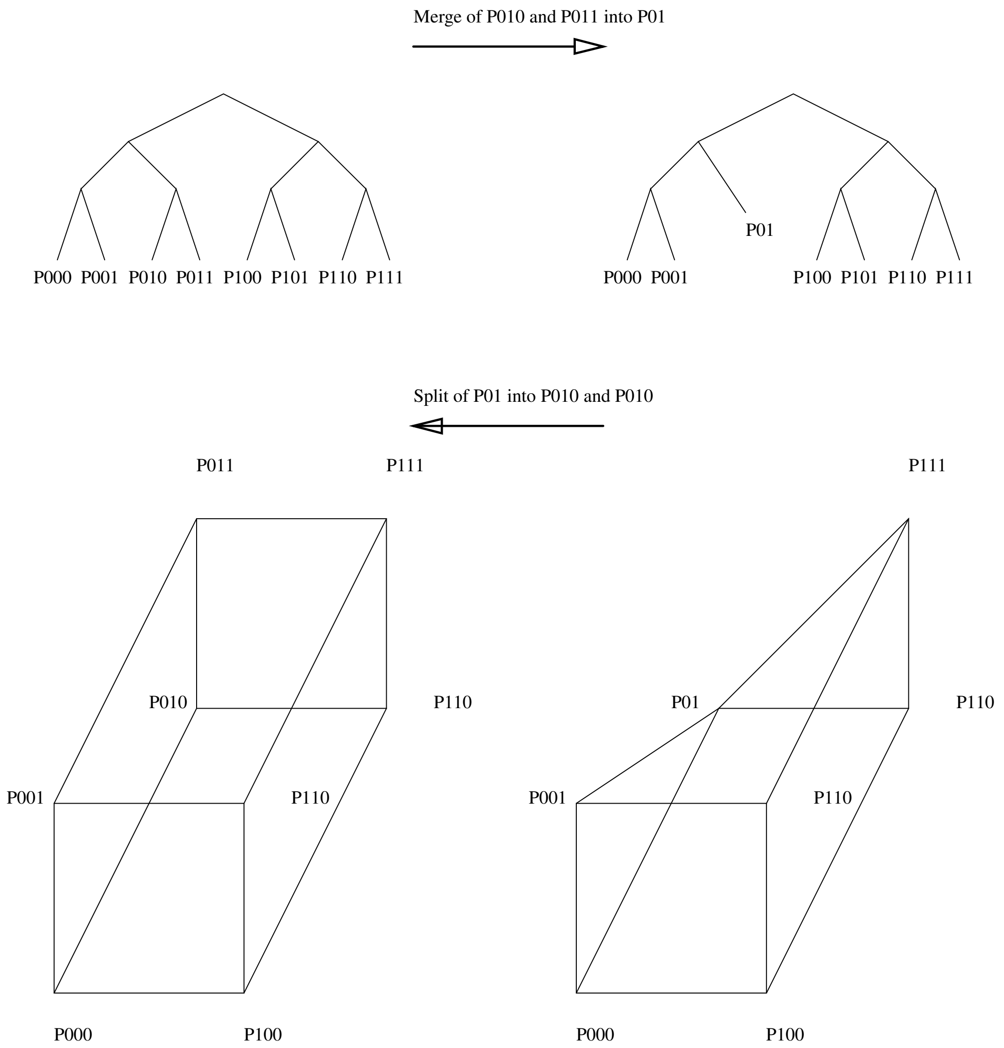

Figure 1. Example of a merge and split on a dynamic hypercube: view of the dynamic graph as a tree (above) and the graph itself (bottom).

found (possibly at the highest level). Change the location of s_1 to that of u, and change the location of s_2 to $p(s_2)$ (i.e., merge s_1, s_2).

Since nodes that get split (respectively merged) are in a locally minimal (respectively maximal) level the local gap of the dynamic graph remains 1 at all times.

In section 5, we show that a dynamic graph with n nodes and a local gap of 1 has a global gap that is bounded by the diameter of $G_{\log n}$. So for dynamic networks that are built from a family $\{G_i\}$ with a logarithmic diameter this balancing scheme maintains a logarithmic global gap.

Lemma 4.1 *The number of nodes examined during re-balancing is at most the global gap.*

Proof: In re-balancing of node addition (respectively node removal) each message searches for a node in lower (respectively higher) level on a dynamic graph with a local gap of 1 . □

Once the balancing algorithm determines which node to split or merge, the new nodes may efficiently locate the nodes to whom to maintain their connections in a decentralized manner using the routing scheme of the existing overlay network (the routing scheme is described later in section 6).

4.2 Randomized Balancing

A different approach to randomizing the dynamic graph is to use balanced allocation techniques during joining in order to keep the tree balanced. The randomized balancing strategy is parameterized by an additional parameter d. Given a parameter d, a node that wants to enter the network chooses d infinite strings, looks at the nodes defined by the strings (their respective longest prefixes) and chooses to split the one that is closest to the root.

As in the deterministic algorithm, the underlying decentralized routing network is used for locating the nodes that correspond to the infinite strings and for building the new edge connections of the dynamic graph (routing is discussed in section 6).

This model is interesting primarily against an oblivious adversary. In order to show bounds on the quality of this balancing we reduce it to the well known balls in bins model. We have the following two lemmas.

Lemma 4.2 *The dynamic graph G constructed by the randomized balancing process above maintains w.h.p. minimal level of at least* $\log n - \log(\log n/d) - \Theta(1)$ *of any leaf.*

Proof: We start with the following lemma on the balls into bins model.

Lemma 4.3 *Suppose that* $\Theta(n(1+\frac{\ln(n/d)}{d}))$ *balls are sequentially placed into* n *bins. For each ball we choose* d *bins uniformly at random and assign the ball to an empty bin if found. Then at the end of the process there are no empty bins with high probability.*

Proof: First we compute the expected time from moving from i non-empty bins to $i+1$ non-empty bins. Clearly once we have i non-empty bins the probability to move to $i+1$ non-empty bins is $1-(i/n)^d$ for each step. Hence, the expected time is $\frac{1}{1-(i/n)^d}$. Thus, the total expected time from the state that all bins are empty until the state that no bins are empty is

$$\begin{aligned}
\sum_{i=0}^{n-1}\frac{1}{1-(i/n)^d} &= \Theta(n)+\sum_{i=n-n/d}^{n-1}\frac{1}{1-(i/n)^d}\\
&= \Theta(n)+\sum_{j=1}^{n/d}\frac{1}{1-(1-j/n)^d}\\
&\le \Theta(n)+\sum_{j=1}^{n/d}\frac{1}{1-(1-jd/(2n))}\\
&= \Theta(n)+\sum_{j=1}^{n/d}\frac{2n}{jd}\\
&= \Theta(n+\frac{n}{d}\log(n/d))\\
&= \Theta(n(1+\frac{\ln(n/d)}{d})).
\end{aligned}$$

Now, we claim that by standard Chernoff bounds it is easy to see that with high probability one would need only $\Theta(n(1+\frac{\ln(n/d)}{d}))$ balls to fill all bins. $\square$

We are now ready to proof the lemma. Assume that all the leaves of the tree are at level i or more. We would like to compute the number of items that are needed to be inserted until all the leaves of the tree reach a level of at least $i+1$ with high probability. Clearly, the process can be modeled by balls assigned to 2^i bins and hence in time $\Theta(2^i(1+\frac{\ln(2^i/d)}{d}))$ all the leaves are of level at least $i+1$ with high probability. We conclude that in time

$$\sum_{i=0}^{r}\Theta(2^i(1+\frac{\ln(2^i/d)}{d}))=\Theta(2^r(1+\frac{\ln(2^r/d)}{d}))$$

all the leaves are of level of at least r. By choosing $r=\log n-\log((\log n)/d)-\Theta(1)$ we conclude that happens with high probability in at most n steps as needed. $\square$

Lemma 4.4 *The dynamic graph G constructed by the randomized balancing process above maintains w.h.p. maximal height* $\log n+\ln\ln n/\ln d+O(1)$ *of any leaf.*

Proof: We will use the following theorem from [2].

Theorem 1 *Suppose that* n *balls are sequentially placed into* n *bins. Each ball is placed in the least full bin, at the time of the placement, among d bins,* $d\ge 2$*, chosen independently and uniformly at random. Then after all the balls are placed with high probability, the number of balls in the fullest bin is* $\ln\ln n/\ln d+O(1)$.

We can simulate our process of splitting the leaves by the process of placing the balls in the bins such that the number of balls in the highest bin is an upper bound for the number of levels that a leaf can reach above the $\log n$ level in the tree. Specifically, we fix a virtual binary tree of depth $\log n$. Each leaf of the virtual tree corresponds to a bin. For each ball, we choose d random infinite strings, we consider first only the prefix string of size $\log n$. Each such prefix corresponds to a leaf in the virtual tree. If one of these nodes is still not a node in the real tree then certainly the node that is split in the real tree will be of depth at most $\log n$. We view this as if the bin that corresponds to the chosen string was empty and remained empty. In case all the d string chosen corresponds to real nodes then the new node will be a descendant of one of them. If we add a new ball to the least full bin (this is not necessarily were the node was split) still by induction the number of balls in each bin is an upper bound on the depth (minus $\log n$) of the deepest leaf which is a descendant of the node that corresponds to the bin. By the above theorem no bin will have more then $\ln\ln n/\ln d+\Theta(1)$ balls hence the level of the leaves will be bounded by $\log n+\ln\ln n/\ln d+O(1)$. $\square$

Putting d to be logarithmic in n, we obtain that the randomized balancing algorithm obtains constant global level-gap w.h.p.

4.3 A combined balancing approach

We can also define a combined strategy where we first randomly choose d strings, use the deterministic balancing algorithm on each string, and finally choose to split the node with the lowest level found.

From a practical point of view combining the two approaches is advantageous. Theoretically it strives to minimize the global gap using both algorithms. This strategy works both against a random sequence and an adaptive adversary. When peer dynamism is random the global gap remains constant w.h.p., and even if a malicious adversary adaptively tries to enlarge the global gap, the local gap remains at most 1. As we shall show in the next section, a constant local gap bounds the global gap as a function of the size of the network and the diameter of the underlying family $\{G_i\}$ (see Corollary 5.4).

5 Dynamic Graph Properties

5.1 Paths in the dynamic graph

DEFINITION 5.1 *A path $P = u_1, u_2, \ldots u_\ell$ will be called* a descendant path *of a path $Q = v_1, v_2, \ldots v_m$ (and Q* an ancestor path *of path P) if P can be partitioned into m consecutive nonempty subsequences $S_1, \ldots S_m$, s.t. for each i, all nodes of S_i are descendants of v_i. The* extension *of a descendant path P of Q is defined as $|P| - |Q| = l - m$.*

The child-neighbor commutativity naturally extends to paths. If $(u, v) \in E_i$ then for any $j > i$ let $U, V \subseteq V_j$ be the sets of all the descendants of u and v in G_j, respectively. Then $U \subset \Gamma_{G_j}(V)$ and so from any $\bar{u} \in U$ there exists an edge to some $\bar{v} \in V$. Thus if $\bar{u}$ is a node of the dynamic graph and a descendant of u then there exists some node $\bar{v}$ of the dynamic graph that is a descendant of v such that $\bar{u}$ has an edge to $\bar{v}$ in the dynamic graph. The following is a direct result:

Lemma 5.1 *Let ℓ be the lowest level of the dynamic graph, fix any two nodes u, v in the dynamic graph and let $\hat{u}, \hat{v}$ be their ancestors in G_ℓ then:*

1. *Every path between u and v in the dynamic graph has an ancestor path between $\hat{u}$ and $\hat{v}$ in G_ℓ*
2. *Every path Q between $\hat{u}$ and $\hat{v}$ in G_ℓ has a descendant path P in the dynamic graph between u and some descendant $\tilde{v}$ of v in the dynamic graph with extension 0 ($|P| = |Q|$).*

5.2 Diameter

Lemma 5.2 *Fix any node s on the lowest level ℓ then the distance from s to any node in the dynamic graph is at most $diam(G_{\log n})$.*

Proof: Consider a source node s on the lowest level ℓ and any target node t on the highest level h in the dynamic graph, let $\hat{t}$ be the ancestor of t in G_ℓ. Consider the shortest path Q in G_ℓ from $\hat{t} \in V_\ell$ to s. From lemma 5.1 there exists a descendant path Q with extension 0 from s to t in the dynamic graph. Since $|c(u)| \geq 2$ we have $\ell \leq \log n$. □

Corollary 5.3 *For a dynamic graph with n nodes and local gap 1, the global gap g, is at most $diam(G_{\log n})$ and the highest level is at most $diam(G_{\log n}) + \log n$.*

Theorem 2 *For a dynamic graph with n nodes and global gap g, the diameter is*

$$\min\{2diam(G_{\log n}), diam(G_{\log n+g})\} .$$

Proof: The $2diam(G_{\log n})$ bound follows directly from lemma 5.2

For the $diam(G_{\log n+g})$ bound, denote the highest level $h \leq \log n + g$. For any $s, t \in V$, fix any descendant $\bar{s}$ of s in V_h. Due to the commutative property, any path from s to t in the dynamic graph is an ancestor of some path from $\bar{s}$ to $\bar{t}$ in G_h where $\bar{t}$ is some descendant of t in V_h.Thus the shortest path from s to t in the dynamic graph is bounded by the diameter of G_h. □

Corollary 5.4 *If for all i the diameter of G_i is at most i, then a dynamic graph on n nodes, with local gap 1, has a diameter of at most than $2\log n$.*

For the examples above, these results imply the following: The diameter of the dynamic hypercube or de Bruijn graphs is at most $2\log n$, and their global gap is at most $\log n$.

Thus the onus of creating a good network lies on the choice of a good family $\{G_i\}$ since $diam(G_i)$ is crucial to the diameter of the dynamic network.

6 Routing on dynamic graphs

The dynamic graph binary tree naturally induces a binary labeling, i.e., each left branch adds a postfix of '0' and each right branch adds a postfix of '1'. A routing target is given as an infinite series $t_1, t_2, \ldots$, and the goal is to find a network node that matches a prefix of the target.

In order to find a certain target, each node must be able to route the lookup request to a neighboring node until the target is reached. A locally computable routing function needs to compute the 'next' node to traverse to. We will say that a routing function $R : V \times \{0,1\}^* \to V$ is k bounded on G if the following properties hold on $G = (V, E)$:

1. Routing function gives an existing edge $(u, R(u, t)) \in E$.

2. If $R(v,t) = v$ then v is a prefix of t.

3. Define $R^j(u,t) = R(R^{j-1}(u,t),t)$ then $R^k(u,t) = R^{k+1}(u,t)$.

We will now show that if routing functions with some recursive properties exist for each G_i then the dynamic graph has a routing function.

DEFINITION 6.1 *Given a graph $G = (V,E)$ and a routing function R, for any node $u \in V$ and target t we define the path $P_R(u,t)$ as the sequence of nodes which are traversed using the routing function R when routing from u to the node who is a prefix of t.*

DEFINITION 6.2 *A family of routing functions $\mathcal{R} = \{R_1, R_2, \ldots\}$ is k fully recursive for a commutative family $(\mathcal{G},\mathcal{P})$ if for any $u \in V_i$ and for any child $v \in c(u)$ we have that $P_{R_{i+1}}(v,t)$ is a descendant path with extension k of the path $P_{R_i}(u,t)$.*

Given fully recursive routing functions $\mathcal{R} = \{R_1, R_2, \ldots\}$ for $(\mathcal{G},\mathcal{P})$, we define a local routing R on the dynamic graph, given a node $u \in V$ and a target binary string t as follows:

1. Let $h = \max_{v \in \Gamma_D(u)} \ell(v)$ be the highest level of all of u's neighbors.

2. Choose any descendant $\bar{u}$ of u in V_h and compute $R_h(\bar{u},t) = v$.

3. Return either v or some ancestor of v that is a neighbor of u.

Theorem 3 *Given a fully recursive routing $(\mathcal{G},\mathcal{P},\mathcal{R})$ where R_i is $f(|V_i|)$ bounded. On a dynamic graph on n nodes with a global gap of g the routing R is a $f(|V_{\log n+g}|)$ bounded routing function.*

Proof: Consider the path Q taken by the above routing R originating at node u and ending at node v that matches the prefix of t. Denote the highest level $h \le \log n + g$. Now examine the path P taken on graph G_h from any descendant of u in V_h to the descendant $\hat{v} \in V_h$ of v that matches the prefix of t using the routing function R_h. Due to the child-neighbor commutative property, and the fully recursive nature of R_i, the real path Q taken on the dynamic graph is an ancestor path of path P and thus $|Q| \le |P|$. □

Corollary 6.1 *Given a $\log(n)$ bounded recursive routing function for each G_i a dynamic graph on n nodes, with local gap 1 has a $2\log n$ bounded routing function.*

6.1 Examples of routing on dynamic networks

Routing on the dynamic hypercube. Consider the routing function R_i that 'fixes' the left most bit that does not equal the target, clearly $\{R_i\}$ is recursive. Remember that the lowest level of such a graph with n nodes is $2\log n$. Now consider the routing function R on a dynamic hypercube with local gap 1. Each move fixes one bit, so after at most $2\log n$ steps the correct node will be found.

Routing on the dynamic butterfly. We consider the butterfly network as a further example. In the butterfly family $\mathcal{B} = \{B_1, B_2, B_3, \ldots\}$ every graph B_i has $i2^{i-1}$ nodes, so some nodes need to split into more than 2 children in order to maintain the child-neighbor commutative property. Thus the encoding of nodes is nontrivial.

DEFINITION 6.3 *Each B_i is a triplet (V_i, E_i, L_i), s.t. (V_i, E_i) is a graph and $L_i \subset V_i$. L_i will be called the lower nodes of the graph B_i. We now define B_i recursively. B_1 is a single node graph $V_1 = \{e\}$ and $L_1 = V_1$. B_k is defined as follows: $L_k = L_{k-1} \times \{01, 00\}$, $V_k = V_{k-1} \times \{10, 11\} \cup L_k$. Any $u = \langle u_1, \ldots, u_{2k}\rangle \in V_k \setminus L_k$ is connected to $\Gamma_{B_{k-1}}(u_1, \ldots, u_{2k-2}) \times \{u_{2k-1}u_{2k}\}$, and any $u = \langle u_1, \ldots, u_{2k}\rangle \in L_k$ is connected to $\{u_1, \ldots, u_{2k-2}\} \times \{10, 11\}$.*

The parent function is defined as follows: for any $u = \langle u_1, \ldots, u_{2k}\rangle \in V_k$, $p(u) = \langle u_1, \ldots, u_{2k-2}\rangle$. From the recursive nature of the definition it is clear that the child-neighbor commutative property holds. Note that any node in L_i splits into 4 children nodes, and any node not in L_i splits into two nodes. For this encoding of nodes we provide a fully recursive routing family based on a standard $3\log n$ routing (details in the full paper) and thus it is possible to route to any target on a dynamic butterfly on n nodes with a local gap of 1 in $O(\log n)$ steps.

Routing on the dynamic de Bruijn network.

DEFINITION 6.4 *A family of routing functions $\mathcal{R}$ is partially recursive for a commutative family $(\mathcal{G},\mathcal{P})$ if for any $u \in V_i$ there exists a child $v \in c(u)$ such that $P_{R_{i+1}}(v,t)$ is a descendant path of the path $P_{R_i}(u,t)$.*

In general, we do not have a routing strategy for the dynamic graph of a family with partially recursive routing only. For such routing functions a node must know which child to choose to be used in the routing algorithm.

However, in the case of the de Bruijn network introduced above, we have a partially recursive routing that can be used for the dynamic graph, as follows: The function R_i for the de Bruijn network G_i computes the 'next' node in the following simple manner: given a node v with a binary identifier $\langle v_1, \ldots, v_k\rangle$ and a target $t = \langle t_1, t_2, \ldots\rangle$, find the minimal j such that $v = \langle v_1, \ldots, v_j, t_1, \ldots, t_{k-j} \rangle$. The 'next' node is the neighbor $\langle v_2, \ldots, v_j, t_1, \ldots, t_{k-j+1} \rangle$. This routing is partially recursive:

DEFINITION 6.5 *Routing on a dynamic de Bruijn network with local gap 1: Consider a node $v = \langle v_1, \ldots, v_k \rangle \in V$ and a target $t = \langle t_1, t_2, \ldots \rangle$, find the minimal j such that $v = \langle v_1, \ldots, v_j, t_1, \ldots, t_{k-j} \rangle$. Compute $R_{k+1}(vt_{k-j+1}, t) = u$ and route to u, or to $p(u)$, or to $p(p(u))$ whichever exists in the dynamic graph.*

The lowest level of a dynamic de Bruijn network on n nodes is at most $2 \log n$. Routing on the lowest level and thus on the dynamic graph is bounded by $2 \log n$.

References

[1] Y. Amir, C. Danilov, and J. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *International Conference on Dependable Systems and Networks* (FTCS-30, DCCA-8), New York, 2000.

[2] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. SIAM Journal of Computing 29(1):180-200, 1999.

[3] N. G. de Bruijn. A combinatorial problem, Konink. Nederl. Akad. Wetersh. Verh. Afd. Natuurk. Eerste Reelss, A49 (1946), pp. 758-764.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. "Freenet: A distributed anonymous information storage and retrieval system". In *Proceedings the ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, 2000.

[5] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[6] `http://gnutella.wego.com`.

[7] D. Karger, E. Lehman, F. T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web". *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 654–663, May 1997.

[8] J. Kleinberg. The small world phenomenon: An algorithmic perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, May 2000, pp. 163–170. (A shorter version available as "Navigation in a Small World", *Nature* 406, August 2000, pp. 845.)

[9] D. Malkhi, M. Naor and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing* (PODC '02), August 2002.

[10] M. Mitzenmacher, A. Richa and R. Sitaraman. The power of two random choices: a survey of techniques and results. IEEE Transactions on Parallel and Distributed Systems 12(10):1094–1104, 2001.

[11] G. Pandurangan, P. Raghavan and E. Upfal. Building low-diameter p2p networks. In *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, 2001.

[12] C. Plaxton, R. Rajaram, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures* (SPAA 97), pp. 311–320, June 1997.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*. August 2001.

[14] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *18 Conference on Distributed Systems Platforms*, Heidelberg (D), 2001, LNCS 2218.

[15] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically Fault-Tolerant Content Addressable Networks, In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, March 2002, Cambridge, MA USA

[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the SIGCOMM 2001*, August 2001.

[17] C. K. Toh. Ad Hoc mobile wireless networks: Protocols and systems. Prentice Hall, 2001.

[18] B. Y. Zhao, J. D. Kubiatowicz and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. U. C. Berkeley Technical Report UCB/CSD-01-1141, April, 2001.