# Brief Announcement: Distributed SplayNets*

**Bruna S. Peres[1], Olga Goussevskaia[2], Stefan Schmid[3], and Chen Avin[4]**

1 **Computer Science Department, Universidade Federal de Minas Gerais, Brazil**
bperes@dcc.ufmg.br
2 **Computer Science Department, Universidade Federal de Minas Gerais, Brazil**
olga@dcc.ufmg.br
3 **Department of Computer Science, Aalborg University, Denmark**
schmiste@cs.aau.dk
4 **Comm. Sys. Eng. Department, Ben Gurion University of the Negev, Israel**
avin@cse.bgu.ac.il

## Abstract

SplayNets are reconfigurable networks which adjust to the communication pattern over time. We present *DiSplayNets*, a distributed (concurrent and decentralized) implementation of SplayNets.

**1998 ACM Subject Classification** C.2.1 Network Architecture and Design, Distributed networks

**Keywords and phrases** Decentralization, Concurrency, Reconfigurable Networks

**Digital Object Identifier** 10.4230/LIPIcs.DISC.2017.58

## 1 Introduction

SplayNets [3] are locally-routable tree networks whose topology self-adjusts to the workload: nodes communicating more frequently become topologically closer to each other over time. SplayNets are hence reminiscent of classic splay tree *data structures*: however, in contrast to splay trees where requests always originate from the root, in a SplayNet, requests occur between arbitrary *node pairs*. SplayNets are motivated, among other, by the advent of reconfigurable datacenter interconnects like ProjecToR [2], and are very different from many traditional network designs which are either entirely oblivious to the communication demand or are optimized towards the demand but cannot be reconfigured over time [1].

In this work, we present DiSplayNets, the first distributed, i.e., decentralized and concurrent implementation of SplayNets. Moving from centralized-sequential to decentralized-concurrent algorithms is challenging, as simultaneous local network reconfigurations can interfere, potentially leading to starvation or even deadlocks, and hence ruining the potential benefits of concurrent operations. Moreover, it needs to be ensured that traffic forwarding and (locally/greedy) routing is unaffected by the topological changes.

We present a distributed algorithm that overcome these challenges, and demonstrate that decentralized SplayNets are feasible.

▶ **Theorem 1.** *DiSplayNets self-adjust to the communication pattern in a fully-decentralized manner, eventually serving all communication requests (in a starvation- and deadlock-free manner).*

* This work is part of the PhD thesis of B.S. Peres and was supported by CNPq, CAPES, Fapemig, and partially by the German-Israeli Foundation for Scientific Research (GIF) Grant I-1245-407.6/2014


## 2 DiSplayNets

**Background and Model.** We want to design a tree $\mathcal{T}$ (i.e., a SplayNet) which adjusts according to a sequence $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{m-1})$ of communication requests occurring over time, where $\sigma_i = (s, d)$ denotes that source $src(\sigma_i) = s$ communicates to destination $dst(\sigma_i) = d$. Each request $\sigma_i$ is generated in some time-slot $t_b(\sigma_i)$, and we will denote its completion time (which depends on the algorithm) by $t_e(\sigma_i)$.

SplayNets are Binary Search Trees (BST) and hence naturally support greedy routing. SplayNets aggressively move communicating nodes together, using the classic splay operations zig, zig-zig and zig-zag [4] of splay trees. However, rather than splaying nodes to the root of the BST, in contrast to splay tree data structures, locality is preserved in that the source and the destination nodes are only rotated to their *least common ancestor* ($LCA(s, d)$).

This motivates us to use the following notion of *splay request* $\mathcal{S}_i(s, d)$: A splay request between the source and destination node of a communication request $\sigma_i(s, d) \in \sigma$ is comprised of two sequences of local network transformations, requested by $s$ and $d$, with the objective to bring these two nodes topologically closer, without violating the BST properties. We say that the splay request $\mathcal{S}_i(s, d)$ has been completed in time-slot $t'$ if the distance $d_{t'}(s, d) = 1$, i.e., $s$ becomes the parent of $d$ or vice versa, whichever happens first.

We define a *distributed* SplayNet as follows: **DiSplayNet** $\mathcal{T}_t = (V, E_t)$ is comprised of $n$ nodes, with distinct identifiers, interacting *concurrently* according to the communication pattern of $\sigma$. In each time-slot $t$, the set of edges $E_t$ connects the nodes in $V$, s.t. $\mathcal{T}_t$ is a BST. We assume that the execution starts with an arbitrary BST topology $\mathcal{T}_0$. Each node $u$ stores the identifiers of its direct neighbors in the tree, i.e., its parent ($u.p$), its left child ($u.l$) and its right child ($u.r$), and the smallest ($u'$) and the largest ($u''$) identifiers currently present in the sub-tree rooted at $u$. This information is used for local routing and for splaying.

**DiSplayNet Design and Distributed Algorithm.** In DiSplayNet, a changing communication pattern leads to local adjustments (possibly concurrently to prior requests) of the communication links in $\mathcal{T}_t$ over time. Consider a DiSplayNet instance $\mathcal{T}_t$, and a communication request $\sigma_i(s, d) \in \sigma, t_b(\sigma_i) \leq t$. Differently from sequential SplayNets, $s$ and $d$ rotate *in parallel* towards the $LCA_t(s, d)$. Due to concurrency, the LCA might change over time. Therefore, instead of approaching a specific LCA node, $s$ and $d$ keep splaying towards the root of $\mathcal{T}_t$, until becoming each other's ancestor. Upon generating a request $\sigma_i(s, d)$, node $s$ must advertise node $d$ so that both start splaying.

DiSplayNet can be described in terms of a state machine, executed by each node in parallel: (1) *Passive:* a node is in passive state in time-slot $t$ if it is not the source or destination of any request in $\sigma_i \in \sigma, t_b(\sigma_i) \leq t$; (2) *Climbing:* a node $s$ (or $d$) is climbing in time-slot $t$ if it has an *active request*: $\exists \sigma_i(s, d) \in \sigma, t_b(\sigma_i) \leq t$ and $d_t(s, d) > 1$, and additionally $s$ (or $d$) $\neq LCA_t(s, d)$; (3) *Waiting:* a node $s$ (or $d$) is waiting in time-slot $t$ if has an active request and $s = LCA_t(s, d)$. (4) *Communicating:* a node $s$ or $d$ is communicating in time-slot $t$ if $\exists \sigma_i(s, d) \in \sigma, t_b(\sigma_i) \leq t$ and $d_t(s, d) = 1$. Figure 1 shows the states transitions.

In order to ensure deadlock and starvation freedom, concurrent rotations are performed according to a *priority*. An older request in the network has a higher priority than a more recent request (Figure 2). Note that, a node $s$ in the *waiting* state might be removed from the LCA position by a rotation with higher priority. If that happens, $s$ returns to the *climbing* state and resumes requesting rotations. Finally, when $s$ and $d$ meet, they communicate.

To synchronize the process between the nodes, we execute the algorithm in rounds. Each round is composed of five phases (1. Rotation Requests; 2. Top-down Acks; 3. Bottom-up

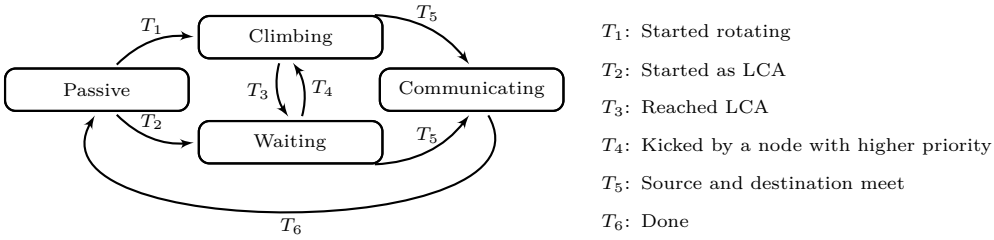


**Figure 1** State Transition Diagram.

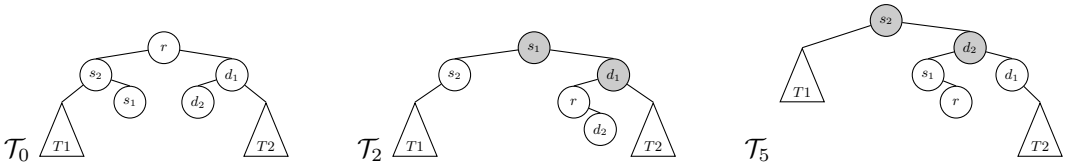


**Figure 2** DiSplayNets: $\sigma_1(s_1, d_1)$ and $\sigma_2(s_2, d_2) \in \sigma$, $t_e(\sigma_1) < t_e(\sigma_2)$.

**Algorithm 1** The distributed DiSplayNets algorithm (one round).

**1: Rotation Requests** (3 time-slots)
**if** Climbing for some $\sigma_i(s, d)$ **then**
send own rotation request $\beta(u)$ upward;
insert $\beta(u)$ into buffer;
upon receiving rotation request $\beta(v)$:
insert $\beta(v)$ into buffer;
forward $\beta(v)$ upward;

**2: Top-down Acks** (3 time-slots)
get highest priority request $\beta(x)$ in buffer
**if** Master($\beta(x)$) **then** ▷ farthest node from $x$
send Ack($\beta(x)$) downward

**3: Bottom-up Acks** (3 time-slots)
upon receiving top-down ack($\beta(v)$)
**if** $\beta(v) = \beta(x)$ **then** ▷ highest priority ack
send ack($\beta(v)$) up toward master

**4: Link Updates** (1 time-slot)
**if** received bottom-up ack($\beta(x)$) **then**
update links according to $\beta(x)$;

**5: State Updates** (1 time-slot)
update state; ▷ Figure 1
clear buffer;

Acks; 4. Link Updates; 5. State Updates), summarized in Algorithm 1. Each node $u$ maintains a local buffer, containing a queue of rotation requests, generated by itself, its right or left child, one of its four grandchildren or eight great-grandchildren. In each round, each rotation request $\beta(u)$ is sent upwards until reaching its *master* (2 hops ancestor in case of a zig and 3 hops ancestor in case of a a zig-zig or zig-zag). Once all requests have been received, the highest priority request is acknowledged top down, from master to requesting node. Upon receiving a top-down ack, the requesting node sends an acknowledgment upwards to the master. We say that neighboring nodes agree to perform rotation $\beta(u)$ if all of them received one top-down and one bottom-up acknowledgment for $\beta(u)$.

**Future Work.** It remains to rigorously analyze the efficiency, i.e., amortized work and time. Our simulations show first promising results.

## References

1 Avin et al. Demand-aware network designs of bounded degree. In *DISC*, 2017.
2 M. Ghobadi et al. Projector. In *ACM SIGCOMM*, 2016.
3 S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker. Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Tran. on Networking*, (99):1–13, 2015.
4 D. Sleator and R. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.