Distributed Long-Lived List Colouring: How to Dynamically Allocate Frequencies in Cellular Networks

Naveen Garg
Dept. of Comp. Science and Engineering
Indian Institute of Technology, Delhi
Hauz Khas, N. Delhi, 110016, India
Email: naveen@cse.iitd.ernet.in

Marina Papatriantafilou* Philippas Tsigas
Department of Computer Science
Chalmers Un. of Technology & Göteborg Un.
S-412 96 Göteborg, Sweden
Email: {ptrianta,tsigas}@cs.chalmers.se

Abstract

To avoid signal interference in mobile communication it is necessary that the frequencies used for communication within each cells are allocated so that no signal interference occurs with neighbouring cells.

We model this channel allocation problem as a generalised list colouring problem and we show how to analytically measure and provide worst-case guarantees regarding *request satisfiability*. To the best of our knowledge, this has not been done before and gives a new perspective to the problem, as well as a clear direction for further investigation.

We propose distributed approaches for solving the problem, which are able to adapt fast to temporal variations in channel demands in different cells, as well as to cope with crash failures, by limiting the *failure-locality* — the size of the network that can be affected by a faulty station, in terms of the distance from that station.

Our first approach is inspired by a relatively recent theorem relating graph colourings and orientations; it achieves the equivalent of the best known sequentially achievable upper bound for request satisfiability, implied by the theorem. It also employs a powerful synchronisation mechanism to achieve worst-case response time that depends only on Δ —the degree of the signal interference graph— and failure locality 4.

Our second proposal is a first approach towards exploring what bound in request satisfiability is achievable without the use of extra synchronisation; by employing randomisation in frequency choices, in only one round of communication, a base station can expect to pick $f/(4\Delta)$ frequencies where f is the size of the list at the node; the failure locality of this solution is only 1.

Keywords: Cellular Networks, Distributed Frequency Allocation, Dynamic Frequency Allocation, Request Satisfiability, Failure Locality.

* *Corresponding Author:* Tel.: (+46 31) 7725413, Fax (+46 31) 165655.

1 Introduction

The integration of mobile communication and computing in fixed networks introduces new issues and problems in distributed computing. The ability of a host to be reachable while moving requires some wireless form of communication. A wireless medium intrinsically supports broadcast communication within a specific region, called a *cell*. To provide communication between different cells, a specific host, called *base station*, is associated with each cell. The set of base stations are interconnected via a fixed network, which usually does not physically support broadcast communication, but rather point-to-point message transmission.

Here we address the problem of *frequency* (or *channel*) *allocation* to base stations. As mentioned before, a mobile host can establish communication with other entities of the network only through the base station associated with the cell in which it is present. In order to satisfy a communication request of a mobile host, the respective base station should allocate, depending on the bandwidth required, a certain number of wireless channels to it. Wireless channels are divisions of the frequency spectrum provided for broadcast communication; the signal carried on a wireless channel is carried on the respective frequency. Henceforth, the terms "channel" and "frequency" are used interchangeably. The wireless channels should be chosen by the base stations in such a way that no interference between signals can occur. A solution to the frequency allocation problem should guarantee that if a channel is used for a communication session of a mobile host in a specific cell, it should not be used concurrently for other communication sessions, except in distant cells, outside the scope of the signal. A solution to the channel allocation problem is evaluated using the following criteria:

- The number of *satisfiable requests* should be maximised (or, equivalently, the number of dropped requests is minimised). The dual way to view this criterion for making efficient use of the bandwidth (which is a "scarce" resource in the mobile setting [13, 14]), is to require that the number of free frequencies needed at a base station so that its requests can be satisfied, is as small as possible. This implies also that a solution should adapt as fast as possible to *temporal variations in channel demand* in different cells.

- The *connection setup time*, i.e. the time when a communication request is issued to the time the session is established, should be minimised.

- The *communication complexity*, i.e. the amount of information (number and size of messages) that needs to be exchanged per request, should be minimised.

- The number of *hand-offs* should be kept to a minimal. Hand-off is a situation when some frequency used by a mobile host for a communication session has to change during the session, in which case the communication might be interrupted until a new frequency is assigned. For ordinary data communication buffering can help, but the quality of real-time traffic can degrade if the number of hand-offs and the time it takes to serve them are high. For this reason, apart from the inter-cell hand-offs — which occur when a mobile host moves from one cell to another — a solution must avoid introducing intra-cell hand-offs.

- The solution should be able to cope with *failures*, since they are a fact of life in any distributed setting. A fault-tolerant solution should guarantee that what happens in a specific neighbourhood has an effect as local as possible. One way to measure this property is the *failure locality* [8] which measures the size of the network affected by

a faulty station in terms of the distance from that station, i.e. smaller failure locality implies higher fault-tolerance.

Clearly, there are also trade-offs, i.e. not all these optimisation criteria can be met concurrently. We are discussing trade-offs in the relevant parts of the paper, where their demonstration can be supported by more detail.

1.1 Contributions of the Paper and Related Previous Work

1.1.1 Problem modelling

We model the possible interference of signals using an *interference graph*, in which the nodes are the base stations (or, equivalently the cells) and the edges between them represent possible signal interferences in case of concurrent use of the same channel. The *free frequencies* at a base station are the frequencies of the spectrum that are not in use by the base station or any of its neighbours in the interference graph. Thus this is the set of frequencies from which channels can be allocated to satisfy the requests at this base station. Note that the set of free frequencies for a base station changes over time. Furthermore, the same frequency could potentially belong to the set of free frequencies of two adjacent base stations and, hence, even when a base station is picking channels from its set of free frequencies it needs to ensure that there is no interference.

The problem of allocating frequencies to mobile hosts by the base stations is now viewed and analysed as a generalised version of the *list colouring* problem. In the list colouring problem, every vertex of the graph has associated with it a list of colours, which, in our case are the set of free frequencies. The requirement is to find a proper vertex-colouring of the graph such that each vertex is coloured with one of the colours in its list. The optimisation objective is to minimise the size of the list that is necessary for properly colouring the graph.

The list colouring problem was introduced in [10]; sequential protocols for solving it can be found in [1, 15, 19]. In sections 2 and 3 we define the problem more formally, we explain its difference with the ordinary vertex colouring problem (see also fig. 2) and its suitability for the channel allocation problem; we also briefly present the known sequential solutions to the problem, which can be used to provide centralised approaches for solving channel allocation.

A relation between the ordinary list colouring and the channel allocation problems has also been noticed in [16]. In the generalised version that we define in the paper, each vertex needs to be coloured with a certain number of colours from its list —the number of colours required for a vertex being the number of requests at the base station. Thus any protocol for the generalised list colouring problem that is *long-lived* (i.e. it can be invoked multiple times, with lists and requests that vary in time), can be used for solving the channel allocation problem in a dynamic manner. Moreover, the optimisation criterion for minimising the list sizes in the generalised list colouring, is in order with the optimisation criterion for maximising *request satisfiability* in channel allocation.

1.1.2 Previous solutions

The first solutions to the channel allocation problem were static, in the sense that each base station was assigned a fixed set of frequencies that it could use; this strategy clearly does not behave well under temporal variations in channel demand.

Existing networks for cellular telephony employ dynamic, but centralised solutions: there exists a *channel manager* (called Mobile Telephone Switching Office or MTSO in the cellular telephony terminology), which is a central fixed base station that collects the requests from the base stations and decides how to satisfy them (a brief description of the system can be found in [13, 14]). Hybrid settings, in which some base stations are allocated frequencies in a static manner, while others are allocated frequencies dynamically (but centrally), have also been considered and studied [16, 18].

Centralised solutions have the drawback that they imply dependency on a central node. This, in turn, implies poor fault-tolerance; whatever happens to the central station is reflected in the whole network. Moreover, for the network to be able to handle rapidly changing load patterns it is important that all the base stations inform the channel manager sufficiently often (aiming at maximising request satisfiability), which would lead to significant traffic. On the other hand, any grouping of frequency requests and releases by the base stations could lead to large waiting times (or denial of service) for the mobile hosts. This trade-off between the amount of network traffic generated and the response time that the solution offers to the mobile hosts is very significant in centralised solutions, where each node has to communicate with a central node some distance away in the network.

A distributed solution can offer better fault tolerance. Additionally, since the nodes communicate only with their neighbours, with the goal of maximising request satisfiability, the amount of network traffic generated can be much less and the trade-off mentioned above is no more a serious consideration. Furthermore, a distributed solution is better poised to exploiting and preserving the locality inherent in the problem.

There have been attempts to solve the problem in a distributed manner. In [9] the proposed protocol involves participation of the mobile hosts themselves in the procedure of frequency assignment; this is not desirable, because of energy constraints. In [17] the problem is analysed as a multiple mutual exclusion problem. In order to satisfy requests, a base station competes for only one channel at a time. The connection set-up time is measured as the number of rounds of message-exchanges with the neighbouring base stations and can be as high as the size of the spectrum. The size of messages too can be as high as the size of the frequency spectrum. The protocol for allocating and releasing frequencies might lead to a situation in which a busy cell which has collected a large set of frequencies prevents its neighbours from satisfying their own requests, thus starving these base stations. Furthermore the solution in [17] uses time-stamps as a priority scheme to achieve exclusion; this can result in arbitrary long process waiting chains, which furthermore implies a failure locality equal to the diameter of the network. Thus, to satisfy a request in the worst case, one may take time proportional to the network size and the failure of some station could potentially cause the whole network to fail.

1.1.3 New protocols

We model the problem as generalised list colouring (cf. Fig. 1) and we first show a way to solve it by deriving the long-lived (i.e. invokable multiple times, with lists and requests that vary in time) generalised distributed equivalent of a known sequential solution to the problem [1], which implies the best known sequentially achievable bound for list sizes, hence, also for request satisfiability. With respect to request satisfiability, the one-shot sequential solution, in order to satisfy the requests in every cell, would require, in the worst case, that for every cell, the sum of its requests and of those of its outgoing-edge-neighbours in an acyclic orientation of

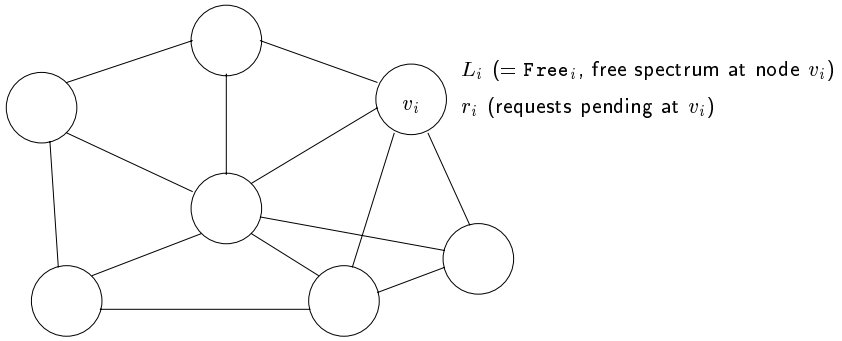

Figure 1: Signal interference graph and modelling frequency allocation as generalised list colouring

the graph does not exceed the number of free frequencies in that cell. Our approach achieves the distributed equivalent of this, depending also on timing and synchronisation aspects of the network and responding to temporal variations of channel demand in different cells. By employing the double doorway and privilege release mechanisms introduced in [8], the worst-case number of messages exchanged and the response time for a set of requests are $O(\Delta^2)$ for general interference graphs and $O(\Delta)$ if the interference graph is planar — a natural setting for mobile communication networks — where Δ is the degree of the graph. Only Δ of the messages exchanged have size that depends on the number of frequencies requested; the rest of the messages need to carry only 2 bits of information. The failure locality of the solution is 4; in asynchronous networks any deterministic protocol has failure locality at least 2 [8]. The solution does not introduce any unnecessary (intra-cell) hand-offs.

Towards exploring what bound in request-satisfiability is achievable without extra synchronisation, we make a first step, by proposing a method which employs randomisation; it achieves constant response time and requires $O(\Delta)$ messages, in order to satisfy $f/(4\Delta)$ requests, in an expected sense, where f is the number of free frequencies. The failure locality of the randomised solution is only 1, which is also the lower bound for randomised protocols. This solution does not introduce any unnecessary (intra-cell) hand-offs, either.

In the next two sections we give a more formal definition of the problem and the model, we explain the difference between the ordinary vertex colouring and the list colouring and the suitability of the latter for the channel allocation problem; we also briefly present the known sequential solutions to the problem, which can be used to provide centralised approaches for solving channel allocation. In the three sections following these we present the new proposed distributed solutions for the generalised long-lived list-colouring, and their completion for frequency allocation. Concluding, we discuss the results of this work and some of the directions for future research.

2 Computation Model and Problem Statement

The network is described with an interference graph $G = (V, E)$. Each node $v_i \in V(G)$ $(1 \leq i \leq n)$, also called *process* or *station*, models a base station of the network. There is an edge between two nodes if there is a signal interference when the same frequency is used

concurrently in the cells of the respective stations. $N(v_i)$ denotes the set of neighbours of node v_i.

A general remark (not a requirement by any of the solutions proposed here) is that the maximum degree (number of neighbours of any vertex) Δ of G is expected to be independent of the number of nodes n in the network. Also, for well designed wide area networks the interference graph G is planar. However our solutions do not assume planarity and are therefore appropriate for all types of networks, including nano-cellular ones (e.g. for multi-story office buildings).

Each process v_i can execute some local computation and communicate with its neighbours in G by sending and receiving messages. If all the processes in the system have the same speed and all message delays are the same, the network is *synchronous*. If no assumption is made on the relative speeds of the stations or on the communication links connecting them, the network is considered to be completely *asynchronous*. Here we focus on asynchronous networks (although, for the sake of a step-by-step presentation of one of our solutions, we consider the behaviour of a preliminary approach in synchronous networks, as well as in asynchronous ones).

Each node v_i has an associated set of colours L_i, which models the set of free frequencies from which the frequencies to be allocated to v_i must be chosen. Finally, each node v_i has an associated number r_i of colour (channel) requests (cf. Fig. 1); r_i may change in time. The requirement is to *properly* colour each node v_i of G, i.e. that each v_i gets r_i colours from its list L_i, in a way that none of the colours used for v_i is used for any of its neighbours in G. This is a generalisation of the *list colouring* problem, which requires that each v_i is properly coloured with *one* colour from its list L_i.

For a centralised solution a protocol should collect the L_i and r_i from every node v_i and compute a proper colouring for all nodes. For a distributed solution, the protocol executed at station v_i should take as input L_i and r_i (which may be different in different invocations) and after a certain number of communication steps it should choose r_i colours from L_i to be assigned to v_i, so that it gets properly coloured.

The *list chromatic number* $\chi_l(G)$ of G is the smallest number k for which, for *any* assignment of a list L_i of size at least k to every vertex $v_i \in V(G)$ it is possible to colour G so that every vertex gets one colour from its list. If $\chi_l(G) \leq k$ then G is said to be *k-choosable*. Note that the list chromatic number $\chi_l(G)$ of a graph G is not necessarily the same as its chromatic number $\chi(G)$ (that is the smallest number of colours which are necessary in order to properly vertex-colour the graph). Clearly, $\chi(G) \leq \chi_l(G)$. An example which shows that the two numbers may differ is shown in figure 2; the bipartite graph $K_{3,3}$ can properly and optimally be coloured using two colours but there are assignments of lists of size two, as the figure shows, which do not allow to colour the graph so that every vertex gets one colour from its list.

Clearly an algorithm which minimises the list sizes which are necessary to properly colour each node v_i with r_i colours from its list, also maximises the *request satisfiability* in the system (or, similarly, minimises the *blocking factor*). Hence, this provides a good way to measure the efficiency of a solution with respect to request satisfiability as the size of the list which is sufficient to ensure that the node gets properly coloured.

The time it takes for a request by a base station to be resolved is measured in units of the maximum message delay in the network; it should be pointed out that this bound is not assumed for the sake of the correctness of any of the protocols proposed here, but only for the sake of measuring the response times.

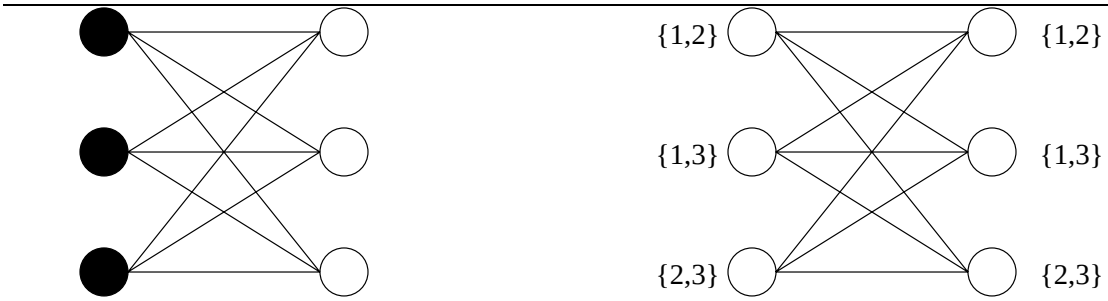

Figure 2: $K_{3,3}$ is 2 colorable but not 2-choosable

It is required that any solution be *starvation free*, that is, as long as a process does not fail, the response time for its requests are bounded.

Failure locality is used in order to measure the effect of process stopping failures. An algorithm has failure locality m if a process is free from starvation even if some process outside its m-neighbourhood has failed by stopping, the *m-neighbourhood* of a process being the set of processes within distance m from it in the interference graph.

3 Sequential List Colouring – Centralised Channel Allocation

As mentioned also in the previous section, the problem of list colouring a graph generalises the vertex colouring problem and is hence **NP**-hard [6]. First we present in brief what is known in the sequential case for list colouring.

General graphs, lists of size $\Delta+1$: Given a graph of maximum degree Δ, one straightforward way of colouring the graph with $\Delta + 1$ colours is to consider vertices one at a time and assign them a colour different from their neighbours. Clearly, we need at most $\Delta + 1$ colours to ensure that for each vertex we can find one colour different from its neighbours. This simple strategy can be adapted to list-colouring – for each vertex we pick a colour from its list that is different from the colours of its neighbours. Therefore, if each list is of size at least $\Delta + 1$ we can always list-colour the graph. Note that if the graph was complete we would actually require that the lists be of size $\Delta + 1$.

Directed acyclic graphs, lists of size equal to (number-of-outgoing-neighbours + 1): If we have an initial acyclic orientation of the edges of the graph then we could first colour all vertices that are sinks (i.e. vertices with no outgoing edges; note that no two sinks are adjacent) by picking an arbitrary colour from their lists. We then remove these sinks and colour the new sinks created taking care to assign them a colour different from their neighbours. It is now easy to see that a list colouring is always possible if each vertex has a list of size more than its out-degree in the initial orientation[1]. This is actually the best known bound for list colouring [1].

Planar graphs, lists of size 5: An elegant argument due to Thomassen [19] shows that every planar graph is 5-choosable. In particular, a planar graph is list colorable if every vertex

[1]Alon and Tarsi [1] actually use linear algebraic techniques to argue that lists of these sizes are sufficient even when the initial orientation of the edges is not acyclic provided the orientation satisfies some properties with respect to the number of Eulerian subgraphs.

in the graph not on the infinite face has a list of size 5 while vertices on the infinite face need only have lists of size 3. This is optimal since there are planar graphs known which are not 4-choosable [20]. We refer the interested reader to [19] for the exact description of the solution.

These sequential solutions can naturally be used to provide centralised solutions to the dynamic channel allocation problem as follows. The channel manager maintains the set of frequencies that are in use at each base station: Every time the channel manager receives requests for frequencies from base stations it runs one of the sequential algorithms to allocate the frequencies. We focus on the first two presented bounds, which are for general graphs. The following lemmas describe the implied bounds for generalised list colouring:

Lemma 3.1 *Lists of size* $(\Delta + 1)r$ *are sufficient to properly list colour each node of a graph whose maximum degree is* Δ*.*

Lemma 3.2 *In a directed acyclic graph* $G = (V, E)$ *in order to properly list-colour each node* v_i *with* r_i *colours, it suffices that*

$$|L_i| = \sum_{(v_i, v_j) \in E} r_j + r_i$$

where $(v_i, v_j) \in E$ *implies that there is an edge connecting the two vertices and the direction is towards* v_j*.*

Next we present a distributed algorithm that achieves the equivalent of the bound in lemma 3.2. To achieve it, the solution needs some extra synchronisation. By relaxing the bound achievable with respect to request satisfiability, a distributed algorithm that employs randomisation and asymptotically achieves the equivalent of the bound in lemma 3.1 with minimal synchronisation, is also presented subsequently.

4 Deterministic Distributed Approach

We show how to derive the generalised distributed equivalent of the solution of the previous section that list colours a graph given an initial acyclic orientation. We take a step-by-step approach to present the final solution.

A preliminary solution: Protocol PREL-DET-DLC

An acyclic orientation can be obtained by first executing a distributed vertex colouring protocol and then orienting each edge from the higher to the lower colour. Arbitrary graphs can be coloured in a distributed manner with $\Delta + 1$ colours [2], while for planar ones 6 is sufficient [11]. To maintain the orientation, a privilege token pr_{ij} is associated with each edge (v_i, v_j). The node for which the edge is incoming holds the token (i.e. its privilege variable is true).

- Starting from such an acyclic orientation, let the sinks concurrently and independently list-colour themselves first (cf. Fig. 3). There is no conflict between them when they choose colours since no two sinks are adjacent.

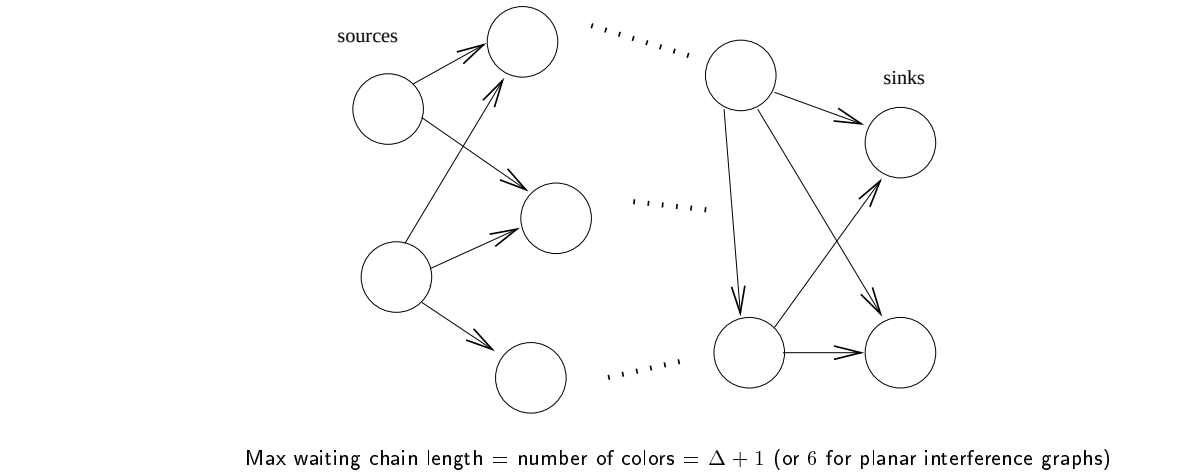

Figure 3: (Solution using) an (initial) acyclic orientation

- When a sink has chosen colours, it communicates to its neighbours the colours it chose and reverses the orientation of its incident edges (by sending to its neighbours the respective token for each edge).
- The resulting orientation is again acyclic, so the new sinks can list-colour themselves next.

By repeating this procedure, in time proportional to the length of the longest *waiting chain* (directed path) in the initial orientation, all the nodes of the graph will be list-coloured once, provided that for each node, the sum of the number of its requests and of those of its outgoing-edge neighbours in the initial orientation does not exceed its list size.

Note that in the initial orientations chosen above, the longest chain length depends only on local measures; namely it is at most $\Delta+1$ and 6 for general and planar graphs, respectively. Hence, this will work well as a *one-shot protocol* (i.e. if each node is interested in choosing colors once), since it also gives the best known guarantee for request satisfiability.

The scheme is also a long-lived solution but it has a drawback: The chains of the initial orientation get modified by becoming longer at their "tails" and shorter at their "heads" due to the edge-reversal mechanism. In synchronous systems this would work fine as it would not cause the longest chain size to grow [4, 7], but if the system is asynchronous, due to differences in the speed of the processes and the communication lines, these modifications may, in the worst case result in waiting chains of length up to n, the size of the network.

The pseudocode for the described preliminary solution is given in figure 4. The sets of frequencies $\texttt{Free}_i$ correspond to the lists of colours L_i of the list-colouring problem. Moerover, each process v_i maintains a set $\texttt{Busy}_i$ of its own occupied frequencies (colours) and for each one v_j of its neighbours a set $\texttt{Occupied}_{ij}$ of the frequencies (colours) occupied by v_j. In order that the neighbors of a process v_i have a correct view of their free frequency sets when they are choosing, v_i informs them about the frequencies it picked using `conf` messages and waits for acknowledgments before it passes the tokens to its neighbours (edge-reversal). The acknoledgements are needed to ensure that the stations have a correct view of the free spectrum (i.e. their `Occupied` sets are updated) when they are choosing. When a frequency

param p_i: **in** $\{1, \ldots, C\}$ {priority (colour) number computed for the orientation}

var $\texttt{Busy}_i, \texttt{Free}_i$: **set of int** ; {busy, free frequencies ($\texttt{Free}_i = L_i$)}
$\forall j : v_j \in N(v_i), \texttt{Occupied}_{ij}$: **set of int** ; {$v_i$'s knowledge of $\texttt{Busy}_j$}
$\forall j : v_j \in N(v_i), pr_{ij}$: **boolean** ; {privilege shared between v_i, v_j; init true if $p_i < p_j$ }

procedure acquire(r_i);
1. wait until $\forall j : v_j \in N(i)$, $pr_{ij} = true$; {wait to become sink }
2. choose $S \subseteq \texttt{Free}_i$ s.t. $|S| = min(|\texttt{Free}_i|, r_i)$;
3. SEND(conf(S), all $v_j \in N(v_i)$) ; **wait** for all ack's ;
4. $\texttt{Busy}_i = \texttt{Busy}_i \cup S$; $\texttt{Free}_i = \texttt{Free}_i - S$;
5. SEND(token, all $v_j \in N(v_i)$) ; $\forall j : v_j \in N(i)$, $pr_{ij} := false$; {pass the privilege tokens}
6. return(S) ;
end

procedure release(f);
1. $\texttt{Busy}_i = \texttt{Busy}_i - f$; $\texttt{Free}_i = \texttt{Free}_i \cup f$;
2. SEND($rel(f)$, all $v_j \in N(v_i)$) ;
end

on RECEIVE(token from v_j) **do**
$pr_{ij} = true$;

on RECEIVE(conf(S) from v_j) **do**
for all $f \in S$ **do**
$\texttt{Free}_i = \texttt{Free}_i - f$; $\texttt{Occupied}_{ij} = \texttt{Occupied}_{ij} \cup f$; SEND(ack, v_j) ;

on RECEIVE(rel(f) from v_j) **do**
$\texttt{Occupied}_{ij} = \texttt{Occupied}_{ij} - f$;
if $\forall j : v_j \in N(v_i), f \notin \texttt{Occupied}_{ij}$ **then** $\texttt{Free}_i = \texttt{Free}_i \cup f$;

Figure 4: Protocol PREL-DET-DLC for node v_i

is not used any more, v_i informs its neighbors using rel messages to update their sets.

The following theorem summarizes the properties of PREL-DET-DLC:

Theorem 4.1 *Protocol PREL-DET-DLC is a correct distributed solution for the generalised list-colouring and the channel allocation problems. In the worst case, for a station v_i to get r_i frequencies (colours) from its $\texttt{Free}_i$ list at time t, it must be the case that*

$$|\texttt{Free}_i| \geq r_i + \sum_{v_j \in N(v_i) : pr_{ij} = false} r_j$$

at that time t; for this, under continuous demand for frequencies, at most 3Δ messages are needed. The failure locality of the solution is $\Delta + 1$ (6 if G is planar) for synchronous systems, or n for asynchronous systems. The worst-case response time is $O(\Delta)$ for synchronous systems and $O(n)$ for asynchronous systems. The size of the messages is 2 bits, except from Δ messages which are $O(r)$ bits long.

Proof. The correctness of the protocol follows from the following argument: Assuming FIFO channels, node v_i's `rel` messages arrive before its next `conf` messages. This guarantees consistent views of the free frequencies by neighbouring nodes. Combining this with the fact that no two neighbouring processes can ever be sinks concurrently, hence they cannot choose frequencies at the same time, it follows that no frequency is used concurrently by neighbouring processes.

Regarding the request satisfiability, a competing node v_i at time t, will wait for all its outgoing neighbours to select frequencies before it is able to choose for itself. This will result in subtracting exactly those frequencies from its `Free` set. Hence, the claim follows.

The failure locality and the waiting time of the protocol are determined by the length of the process waiting chains that are formed. These, in turn are implied by the longest directed paths in the dynamic orientation. In a synchronous network these will be bounded by the initial $\Delta + 1$ (6 if G is planar) [4]. If the speeds of the processes and the message delays vary very much, though, these can be up to n long.

In order to become sink a process must receive the privilege tokens from its outgoing neighbours (at most Δ). After choosing frequencies it exchanges at most 2Δ `conf` and `ack` messages and then it passes all the privilege tokens to its neighbours. Thus every time a node chooses frequencies it would exchange at most 3Δ messages with its neighbours.

∇

Protocol DET-DLC

In a synchronous system the solution just described would work just fine. However, if we are looking for an approach whose worst case behaviour depends only on local measures and which is suitable for any system, we need to further develop it for the long lived protocol. The double doorway synchronisation mechanism of Choy and Singh [8] is shown to be appropriate for our setting too. For ease of reference we briefly rephrase its principles, in parallel with the description of our final solution (see also fig. 5).

An initial vertex coloring of the graph is assumed, in order to serve as a priority scheme. Assume C colors are used. Between neighboring nodes that compete for frequencies (i.e. for list-colouring) at the same time, the one with smaller color (p_i) chooses first. In order to avoid starvation of the nodes with low priority due to preemption by their higher priority neighbours, a double doorway synchronization is used.

The asynchronous doorway is equivalent to the process asking for permission to enter and waiting until it receives one (i.e. a synchronization message $\neq \texttt{sync}_1$) from each of its neighbors *once* (fig 5, line 1 of procedure *acquire*). Although this alone could guarantee starvation freedom — provided that a process that has already crossed the doorway defers giving permission to its neighbors until after it has chosen frequencies — it could result in exponential response time ($O(\Delta^C)$ [8]). This is because after each process v_i has crossed the doorawy, its neighbours with smaller color (which had received v_i's permission before it crossed the doorway) may also cross and they may do so one by one, in such a "staggered" timing that each one of them prevents v_i from entering the choosing phase exactly at the time that v_i is ready to enter it. As this can, in the worst case, happen with all the processes in a C-long waiting chain, the claim follows.

The synchronous doorway is equivalent to the process waiting until it finds an instant when

var $\texttt{Busy}_i, \texttt{Free}_i$: **set of int** ; {busy, free frequencies ($\texttt{Free}_i = L_i$) }
$\forall j : v_j \in N(v_i), \texttt{Occupied}_{ij}$: **set of int** ; {$v_i$'s knowledge of $\texttt{Busy}_j$}
$\forall j : v_j \in N(v_i), l_{ij}$: **msgtype** ; {last *synchronization* message received from v_j}
$\forall j : v_j \in N(v_i), pr_{ij}$: **boolean** ; {privilege shared between v_i, v_j; one is true at a time}

param p_i: **in** $\{1, \ldots, C\}$ {priority (colour) number computed for the orientation}

procedure acquire(r_i);
1. $\forall j : v_j \in N(i)$, **wait until** $l_{ij} \neq \texttt{sync}_1$;
{1st doorway: wait until true once for *each* v_j }
SEND($\texttt{sync}_1$, all $v_j \in N(v_i)$) ;
2. **wait until** $\forall j : v_j \in N(i)$, $l_{ij} \neq \texttt{sync}_2$;
{2nd doorway: wait until concurrently true for *all* v_j's }
SEND($\texttt{sync}_2$, all $v_j \in N(v_i)$) ;
3. < request and wait for all privileges pr_{ij} >; {see fig. 6}
4. choose $S \subseteq \texttt{Free}_i$ s.t. $|S| = min(|\texttt{Free}_i|, r_i)$;
5. SEND($\texttt{conf}(S)$, all $v_j \in N(v_i)$) ; **wait** for all ack's ;
6. $\texttt{Busy}_i = \texttt{Busy}_i \cup S$; $\texttt{Free}_i = \texttt{Free}_i - S$;
7. SEND($\texttt{sync}_3$, all $v_j \in N(v_i)$) ; {signal for doorways }
8. **return**(S) ;
end
procedure release(f);
1. $\texttt{Busy}_i = \texttt{Busy}_i - f$; $\texttt{Free}_i = \texttt{Free}_i \cup f$;
2. SEND($rel(f)$, all $v_j \in N(v_i)$) ;
end

on RECEIVE($\texttt{sync}_x$ from v_j) **do**
$l_{ij} = \texttt{sync}_x$;

on RECEIVE($\texttt{conf}(S)$ from v_j) **do**
for all $f \in S$ **do**
$\texttt{Free}_i = \texttt{Free}_i - f$; $\texttt{Occupied}_{ij} = \texttt{Occupied}_{ij} \cup f$; SEND($\texttt{ack}$, v_j) ;

on RECEIVE($\texttt{rel}(f)$ from v_j) **do**
$\texttt{Occupied}_{ij} = \texttt{Occupied}_{ij} - f$;
if $\forall j : v_j \in N(v_i), f \notin \texttt{Occupied}_{ij}$ **then** $\texttt{Free}_i = \texttt{Free}_i \cup f$;

Figure 5: Protocol DET-DLC for node v_i

none of its neighbors is past this second doorway, i.e. *at the same time* the last synchronization message received by each one of its neighbours must be $\neq \texttt{sync}_2$ (fig 5, line 2 of procedure *acquire*). Although this doorway alone would prevent the above described exponential scenario, by a simple observation it can easily be seen that it can unfortunately result in starvation.

However, the two doorways combined (first the asynchronous, then the synchronous) cancel each-other's drawbacks and, furthermore, guarantee that each process that tries to enter the choosing phase can only possibly be "taken over" at most once by each of its (faster) neighbours.

After having crossed both doorways (sending $sync_2$ messages to all neighbors) a process v_i has to wait for its neighbors that are also past both doorways, to pass to it the respective privileges (fig 5, line 3 of procedure *acquire*). Initially this is meant to implement the edge-reversal procedure described previously in this section; additionally, the use of a smart *privilege-release mechanism*, guarantees good failure locality and works as shown in figure 6.

- *A competing node v_i, which has crossed both doorways, first requests the privileges from its higher priority neighbours and then (i.e. after having obtained those) from the lower priority ones.*
- *A node which is non-competing or has not crossed both doorways, always gives a requested privilege.*
- *A competing node v_i, which has crossed both doorways, gives the privilege to a:*
  - *lower priority neighbour if it has not collected the privileges from all its higher priority neighbours; otherwise it defers answering until after it has finished choosing frequencies.*
  - *higher priority neighbour if it has not collected all the other privileges.*

Figure 6: Privilege release mechanism to reduce the waiting-chain length

The way that privileges are requested and passed from one node to the other ensure that the maximum length of any process waiting chain is bounded by four (two processes waiting for the two doorways, followed by two for the privilege acquisition) at any time instant in any execution [8].

Like in PREL-DET-DLC, here, too, each process v_i maintains $\texttt{Busy}_i$ and `Occupied` sets of frequencies and follows the same steps to inform its neighbours of the frequencies it picks/releases. Finally, it signals them with $\texttt{sync}_3$ messages that its own choosing phase is over.

The waiting time of a process after it has crossed both doorways is $O(C)$; this is due to the privilege transfer mechanism from higher color nodes to the lower colored ones. In order to cross the second doorway, which is synchronous, the process might have to wait — once and no more — for each one of its neighbors to cross it, finish picking frequencies and send the $\texttt{sync}_3$ messages. Therefore, the waiting time at this doorway is $O(\Delta C)$. In order to cross the first doorway, which is asynchronous, a process, in the worst case, has to wait for the slowest of its neighbors that had already crossed it before, to finish and send the $\texttt{sync}_3$ messages. Therefore, the total response time is $O(\Delta C)$.

The following theorem summarises the properties of DET-DLC:

Theorem 4.2 *Protocol DET-DLC is a correct distributed solution for the generalised list-colouring and channel allocation problems. In the worst case, after a process v_i has executed step 1 of procedure acquire, it is guaranteed to get r_i colours (frequencies) from its $\texttt{Free}_i$ list if*

$$|\texttt{Free}_i| \geq r_i + \sum_{v_j \in N(v_i)} r_j$$

at that point in the execution. For this, the worst case response time and the number of messages needed are $O(\Delta^2)$ ($O(\Delta)$ if G is planar). The failure locality of the solution is 4. The size of the messages is 2 bits, except from Δ messages which are $O(r_i)$ bits long.

Proof. The consistency of views of the free frequencies by neighbouring nodes is guaranteed in the same way as in PREL-DET-DLC. This, together with the fact that no two

neighbouring processes choose frequencies at the same time (which follows from [8]) implies that no frequency is used concurrently by neighbouring processes.

As there are no new dependencies introduced, the failure locality the solution is the same as the failure locality of the synchronisation mechanism used, which is 4 [8].

Regarding the request satisfiability, a node, after having crossed the first doorway, will, in the worst case, have to wait for all its neighbours to select frequencies. Since this does not happen more than once [8], the claim follows.

The waiting time and the number of messages exchanged by a competing process is dominated by the respective measures of the synchronisation used (plus a constant of one round of `conf` and `ack` messages), which is $O(\Delta C)$ (which gives $O(\Delta^2)$ for general graphs and $O(\Delta)$ and for planar ones). The size of the messages exchanged for the synchronisation phase is two bits long, while the `conf` messages are $O(r)$ bits long. ∇

5 Randomised Distributed Approach

Our previous solution for the generalised distributed list colouring problem relied on mutual exclusion. The natural question, then, would be to determine what is achievable without any advanced synchronisation. Here we take a first step towards answering this question and we present a randomised approach (see figure 7) that assigns colours to a node such that they do not conflict with the neighbours; the number of colours assigned depends (with high probability) on the ratio of the size of the list and the degree of the node.

Let node v_i have degree Δ and a list L_i of cardinality f. The idea is, for each competing node, to pick randomly, an ϵ fraction of the colours in L_i; thus each colour in the list is picked with a probability ϵ. The node then checks with its neighbours to ensure that some colour it picked is not picked by a neighbour; the colour is dropped if such is the case. Thus a colour picked by two neighbours could potentially be dropped by both of them.

Node v_i, as in protocol DET-DLC, maintains two sets $\texttt{Free}_i$, the set of frequencies that are available to be picked and $\texttt{Busy}_i$, the set of frequencies that are in use; it also maintains for each neighbour v_j a set $\texttt{Occupied}_{ij}$, its view of the set $\texttt{Busy}_j$.

Protocol RAND-DLC ensures that at any point a frequency that is in the $\texttt{Free}_i$ set of v_i does not belong to its $\texttt{Busy}_i$ set or to the $\texttt{Busy}_j$ set of any of its neighbours, $v_j \in N(v_i)$. After node v_i picks $\epsilon|\texttt{Free}_i|$ frequencies from $\texttt{Free}_i$ it updates the sets $\texttt{Busy}_i$ and $\texttt{Free}_i$, accordingly. These updates are tentative since not all the frequencies picked will be eventually acquired. We also require that these updates are made without interruption; the node suspends all message handling while making these changes to the sets. Node v_i then sends the list of frequencies picked to its neighbours to check if they are in conflict. After it has received replies from all its neighbours it knows whether a particular frequency can be acquired, in which case this frequency is left in the $\texttt{Busy}_i$ set; otherwise it is deleted from the $\texttt{Busy}_i$ set and added back to the $\texttt{Free}_i$ set if it has not (in the meanwhile) become occupied by some of v_i's neighbours.

Node $v_j \in N(v_i)$, on receiving a list of tentatively busy frequencies from its neighbour v_i, checks, for each frequency in this list, whether it belongs to $\texttt{Busy}_j$. Only if the frequency does not belong to $\texttt{Busy}_j$ does it reply with a 'Yes', saying 'No' otherwise.

As in DET-DLC, when v_i acquires or no longer uses a frequency it informs its neighbours using `conf` and `rel` messages, respectively, in order to update their sets. Note that here v_i does not need acknowledgement messages by its neighbours after its `conf` messages, since

var $\texttt{Busy}_i, \texttt{Free}_i$: **set of int** ; {busy, free frequencies ($\texttt{Free}_i = L_i$)}
$\forall j : v_j \in N(v_i), \texttt{Occupied}_{ij}$: **set of int** ; {$v_i$'s knowledge of $\texttt{Busy}_j$}

procedure acquire();
1. Pick, *randomly*, $S \subseteq \texttt{Free}_i$ s.t. $|S| = \epsilon|\texttt{Free}_i|$
$\texttt{Busy}_i = \texttt{Busy}_i \cup S$; $\texttt{Free}_i = \texttt{Free}_i - S$;
2. SEND($\texttt{pick}(S)$, all $v_j \in N(v_i)$) ; **wait** for all replies ;
3. **for** all $f \in S$ **do**
if some neighbour has replied 'No' for f **then**
$\texttt{Busy}_i = \texttt{Busy}_i - f$; $S = S - f$;
if $\forall j, v_j \in N(v_i), f \notin \texttt{Occupied}_{ij}$ **then** $\texttt{Free}_i = \texttt{Free}_i \cup f$;
4. SEND($\texttt{conf}(S)$, all $v_j \in N(v_i)$) ; {S is now the set of acquired frequencies}
5. **return**(S) ;
end
procedure release(f);
1. $\texttt{Busy}_i = \texttt{Busy}_i - f$; $\texttt{Free}_i = \texttt{Free}_i \cup f$; $S = S - f$;
2. SEND($\texttt{rel}(f)$, all $v_j \in N(v_i)$) ;
end

on RECEIVE($\texttt{pick}(S)$) **do**
for all $f \in S$ **do**
if $f \in \texttt{Busy}_i$ **then** REPLY('No', f) **else** REPLY('Yes', f) ;

on RECEIVE($\texttt{conf}(S)$ from v_j) **do**
for all $f \in S$ **do**
$\texttt{Free}_i = \texttt{Free}_i - f$; $\texttt{Occupied}_{ij} = \texttt{Occupied}_{ij} \cup f$;

on RECEIVE($\texttt{rel}(f)$ from v_j) **do**
$\texttt{Occupied}_{ij} = \texttt{Occupied}_{ij} - f$;
if $\forall j, v_j \in N(v_i), f \notin \texttt{Occupied}_{ij}$ **then** $\texttt{Free}_i = \texttt{Free}_i \cup f$;

Figure 7: Protocol RAND-DLC for node v_i

it knows that they know about its choice and they update their sets accordingly. When v_j receives the list of acquired frequencies from v_i it makes the necessary updates to its $\texttt{Free}_j$ and $\texttt{Occupied}_{ji}$ sets.

The following theorem summarises the properties of RAND-DLC.

Theorem 5.1 *Protocol RAND-DLC is a correct distributed solution for the generalised list-colouring and channel allocation problems. With the exchange of 3Δ messages of size $O(r)$ bits and in 3 rounds of communication a node v_i is expected to get at least*

$$\mu = \frac{|\texttt{Free}_i|}{4\Delta}$$

colours (frequencies) from its $\texttt{Free}_i$ list, while with probability at least $1 - e^{-1}$ it gets at least $r = \mu - \sqrt{2\mu}$ colours (frequencies). The solution has failure locality 1.

Proof. For correctness, we need to argue that no two adjacent nodes can acquire the same frequency at the same time. Note that a frequency that is acquired by v_i is in $\texttt{Busy}_i$

from the point that it is picked by v_i to the point it is released. Node v_i acquires a frequency only after it receives replies from each of its neighbours, v_j, saying that the frequency is not in their $\texttt{Busy}_j$. Hence v_i cannot acquire the frequency if one of its neighbours has already picked it.

To compute the probability that a frequency that was picked by a node is retained, observe that the probability that a neighbour v_j also picked this frequency is ϵ. Since the frequency is retained when none of the neighbours picks it and since the number of neighbours who could be competing is at most Δ, the probability that a frequency is retained is at least $(1-\epsilon)^{\Delta}$.

Let X_j be a random variable that is 1 if the j^{th} frequency in the `Free` list (of some v_i) is acquired (and is 0 otherwise). Then the probability that $X_j = 1$ is at least $\epsilon(1-\epsilon)^{\Delta}$. Let further $X = X_1 + X_2 + \ldots + X_f$ be the sum of these random variables. The expected value of the random variable X is at least $f \cdot \epsilon(1-\epsilon)^{\Delta}$ which is maximised when $\epsilon = 1/(\Delta+1)$. For this choice of ϵ, the expected number of frequencies acquired is at least

$$\mu = \frac{f}{\Delta}\left(1 - \frac{1}{\Delta+1}\right)^{\Delta+1}$$

Note that $\Delta \geq 1$ and hence the expected number of frequencies acquired is at least $f/(4\Delta)$. Thus with this simple randomised scheme each node can expect to acquire a fraction $1/(4\Delta)$ of the free frequencies available to it. This compares well with the best centralised algorithm for general (unstructured) graphs, where it is essential that a node have at least $\Delta+1$ free frequencies for it to be able to acquire a frequency (cf. 3.1).

Since X is the sum of independent Bernoulli trials we can use Chernoff's bounds [12] to bound the probability that the number of frequencies acquired is at least $(1-\delta)\mu$ as

$$\Pr(X > (1-\delta)\mu) > 1 - e^{-\mu\delta^2/2}$$

Thus for $\delta = \sqrt{\frac{2}{\mu}}$ the probability that we acquire $\mu - \sqrt{2\mu}$ frequencies is at least $1 - e^{-1}$.

The failure locality of the solution follows from the fact that a node may be kept waiting only due to its immediate neighbours (no answers are ever deferred), hence, the maximum waiting chain is of length only one. ∇

6 Completing the Solutions for Frequency Allocation

In this section we show a general scheme of how a base station would invoke any of the proposed protocols for frequency allocation.

This scheme also addresses the trade-off between connection set-up time and request satisfiability at high loads (or highly changing load patterns). In particular, provision by each base station to require/retain extra frequencies for being able to accommodate calls locally results in ensuring faster response times and in saving communication overhead. On the other hand, to maximise request satisfiability it is best to allocate frequencies to base stations only when the demand arises.

In protocol DRIVE-DLC shown in fig 8, each base station can keep a local "pool" of frequencies (`Busy` set), some of which may be actually used (`InUse` set), while the others are available for satisfying locally new connection requests. The size of this local set (`LocalNeed` variable) can reflect the temporal variations in channel demands in different cells and allow the base stations to adapt accordingly.

var $\texttt{LocalNeed}_i$: **int** ; {estimate of number of frequencies needed to satisfy requests locally}
$\texttt{InUse}_i$: **set of int** ; {frequencies actually used among those taken by v_i ($\texttt{InUse}_i \subseteq \texttt{Busy}_i$)}
{$\texttt{Busy}_i, \texttt{Free}_i$ are icluded from the other -DLC protocols}

when < connection request > **do**
<estimate/update $\texttt{LocalNeed}_i$> ; {e.g. using Little's law (i.e. $\lambda_i T_i$)}
if $|\texttt{Busy}_i| \geq \texttt{LocalNeed}_i$ **and** $|\texttt{Busy}_i| - |\texttt{InUse}_i| \geq nr$ **then**
<satisfy the request locally>
else acquire($max(nr, \texttt{LocalNeed}_i - |\texttt{Busy}_i|)$) ;
$\texttt{InUse}_i := \texttt{InUse}_i \cup <$ nr of the allocated frequencies > ;

when < connection release > **do**
nr :=<released connection's number of frequencies> ;
$\texttt{InUse}_i := \texttt{InUse}_i -$ < released frequencies > ;
<estimate/update $\texttt{LocalNeed}_i$> ;
if $|\texttt{Busy}_i| \geq \texttt{LocalNeed}_i$ **then**
<release $min(nr, |\texttt{Busy}_i| - \texttt{LocalNeed}_i)$ > {else retain, to satisfy expected new requests}

Figure 8: General scheme of protocol DRIVE-DLC for node v_i (to be used in combination with PREL-DET-DLC or DET-DLC or RAND-DLC)

The idea is that a base station v_i decides on the number of frequencies r_i to ask for, each time that it invokes *acquire*, so as to be able to accommodate the traffic in its cell. One way for this to be done is by monitoring the average rate of frequency requests (λ_i) and the average job duration in the system (T_i) and applying Little's formula [5], according to which, the expected load at a station (i.e. the expected demand for frequencies in the respective cell) is $\lambda_i * T_i$. Hence, r_i can be determined using this estimate and the number of frequencies that are already in use at that point.

In this way, the current expected load at the station is satisfied locally. Any deviations from this can be satisfied with worst case response times determined from the use of protocol invoked (cf. theorems 4.1, 4.2, 5.1), while the base station updates the expected load so that this does not happen too often.

Finally, as concerns hand-offs, in the approaches proposed in this paper, hand-offs may arise *only* when a mobile host changes cells. Then the communication must be interrupted and frequencies available in the new cell must be assigned to that session. For plain data communication the gap can be closed using buffering of the data packets until the new frequencies are allocated. For real-time data communication (e.g. audio/video), though, this cannot apply; hence, it is very important that the procedure be fast, since otherwise the quality of communication might degrade. A natural enhancement is to have the base stations (i) distinguish the type of requests, between regular and those that result from hand-offs, and service the latter with higher priority, and/or (ii) keep a set of "back-up" frequencies that will be used to serve requests due to hand-offs, to ensure availability. The latter may restrict the bandwidth available for regular requests, but will preserve the quality of communication in presence of hand-offs.

7 Discussion and Further Research

Summarising, we have shown distributed solutions for the generalised list colouring problem that can be used to solve the dynamic channel (frequency) allocation problem for networks of base stations that support mobile communication. By being distributed, the protocols that use these approaches can take advantage of the locality and node independence in the network; thus they have good scalability properties, as opposed to currently existing centralised solutions, whose performance depends on the distances between the base stations and the respective channel managers. Another big advantage of the approaches presented is that they can tolerate node failures by limiting their effects to only a small neighbourhood around the stations where they happen, and not letting them propagate and affect the whole network. Moreover, they guarantee fast response to frequency requests (local or, in the worst case depending on local measures and small constants) and are simple and easy to apply in practice, provided the existence of distributed infrastructure in networks that are in use.

Regarding the request satisfiability, our deterministic approach achieves in a distributed manner the generalised equivalent of the "outgoing-edges"- bound for list colouring. It seems that the only way to obtain a solution without relaxing this condition is by an exclusion approach. Our second approach employs randomisation and allows a node to satisfy a number of requests that depends on the ratio of the free spectrum size and the degree of the node, without any extra synchronisation. It is an important open problem to either prove that this amount of synchronisation is necessary or design a distributed protocol which will preserve (or closely follow) the first condition without employing exclusion.

Furthermore, as discussed in several points in the paper, there are important trade-offs between the optimization criteria for the solution (e.g. between availability and bandwidth utilisation); to analyse them is an important set of open problems.

Acknowledgements

This work was done while the authors were researchers at the Max-Planck Institute for Computer Science, in Saarbrücken, Germany. The first author was also partially supported by the EU ESPRIT LTR Project No. 20244 (ALCOM-IT). We greatly acknowledge the scientifically stimulating and friendly working environment of the institute.

We also wish to thank the anonymous referees for their constructive comments, which have helped significantly to improve the presentation of this work.

References

[1] N. Alon and M. Tarsi. Colourings and Orientations of Graphs. *Combinatorica* 12 (2) (1992), pp. 125–134.

[2] B. Awerbuch, A. Goldberg, M. Luby and S. Plotkin. Network Decomposition and Locality in Distributed Computation. *Proceedings of the IEEE 30th Symp. on Foundations of Comp. Science*, pp. 364–369, 1989.

[3] B.R. Badrinath, A. Acharya and T. Imielinski. Impact of Mobility on Distributed Computations. Operating Systems Review, Vol. 27, No. 2, Apr. 1993.

[4] V. Barbosa and E. Gafni. Concurrency in Heavily Loaded Neighbourhood-Constrained Systems. *ACM Transactions on Programming Languages and Systems*, Vol. 11, No. 4, pp. 562-584, Oct. 1989.

[5] D. Bertsekas, R. Gallager. *Data Networks*, Prentice Hall International Editions, 1987.

[6] M. Biró, M. Hujter and Z. Tuza. Precoloring Extensions. I. Interval Graphs. *Discrete Math.* **100**, pp. 267–279, 1992.

[7] K.M. Chandy and J. Misra. The Drinking Philosophers Problem. *ACM Transactions on Programming Languages and Systems*, Vol. 6, No. 4, pp. 632-646, Oct. 1984.

[8] M. Choy and A. Singh. Efficient Fault Tolerant Algorithms for Resource Allocation in Distributed Systems. *ACM Trans. Prog. Lang. Syst.*, Vol. 17, No. 3, pp. 535-559, May 1995. (Also in *Proceedings of the 24th ACM Symposium on Theory of Computing*, pp. 593-602, 1992.)

[9] J. C.-I. Chuang. Performance Issues and Algorithms for Dynamic Channel Assignment. *IEEE Journal on Selected Areas in Communications*, 11(6), pp. 955–963, Aug 1993.

[10] P. Erdös, A.L. Rubin and H. Taylor. Choosability in Graphs. *Proceedings of the West Coast Conference on Combinatorics, Graph Theory and Computing*, (Congr. Num 26), pp. 125–157, 1979.

[11] S. Ghosh and M.H. Karaata. A self-stabilising algorithm for colouring planar graphs. *Distributed Computing* **7**, pp. 55-59, 1993.

[12] T. Hagerup and C. Rüb. A Guided Tour of Chernoff Bounds. *Information Processing Letters* **33** (1989/90), pp. 305-308.

[13] S.G. Hild. A Brief History of Mobile Telephony *Technical Report No. 372, University of Cambridge, Computer Laboratory*, Jan. 1995.

[14] G. Ioannidis. Mobile Computing. *PhD Thesis, Columbia University*, 1992.

[15] T.R. Jensen and B. Toft. *Graph Colouring Problems.* Wiley-Interscience Series in Discrete Mathematics and Optimisation, 1995.

[16] E. Malesinsca. An Optimisation Method for the Channel Assignment in Mixed Environments. *Proceedings of ACM MobiCom '95.* Also available as TR. No. 458/1995, Fachbereich Mathematik, Technische Universität Berlin.

[17] R. Prakash, N. G. Shivarati and M. Singhal. Distributed Dynamic Channel Allocation for Mobile Computing. *Proceedings of the 14th ACM Symp. on Principles of Distr. Computing*, pp. 47–56, 1995.

[18] H.-U. Simon. The Analysis of Dynamic and Hybrid Channel Assignment. *Technical Report, Universität des Saarlandes*, SFB144–B1, 10/1988.

[19] C. Thomassen. Every Planar Graph Is 5-Choosable. *Journal of Combinatorial Theory*, Series B **62**, pp. 180-181, 1994.

[20] M. Voigt. List Colourings of Planar Graphs. *Discrete Mathematics* **120**, pp. 215-219, 1993.