

Distributed Online Call Control on General Networks

Harald Räcke*

Adi Rosén†

Abstract

We study the problem of online call admission and routing (“call control”) on general networks. We give new algorithms that with high probability achieve a polylogarithmic fraction (in the size of the network) of the optimal solution. The decisions of our algorithms do not depend on the current load of all network links, as in previous algorithms for general network topologies [AAP93]. Instead, their admission decisions depend only on link loads along a single path between the communicating parties, and they can thus be performed in a distributed hop-by-hop manner through the network. Furthermore, our algorithms can handle concurrent requests in the network.

1 Introduction

One of the most important problems in the area of network algorithms is the problem of the online admission and routing of virtual circuits. This problem is many times referred to as the “call control problem”. A sequence of request for calls is received in an online manner. Each request consists of a pair of nodes that wish to communicate. For each such request the algorithm has to immediately either accept the call or reject it. To accept the call the algorithm has to immediately select a virtual circuit (path) between the communicating parties, obeying the network constraints such as link capacities. The aim of the algorithm is that of maximizing the number of accepted calls.

Networks with unstructured topology are becoming increasingly important due to the dramatic growth of the Internet and the widening popularity of ad-hoc networks. A call control algorithm for general network topologies was given in [AAP93]. This algorithm is many times referred to as the “AAP” algorithm. The AAP algorithm suffers however from the drawback that its admission and routing decisions are based on knowledge

of the current load on *all* network links. Therefore, a central node in the network has to be in charge of taking all admission and routing decisions (or link loads have to be continuously distributed in the network). Such procedure will create a “hot spot” in the network with high volumes of control traffic, and the operation of the whole network will depend on the availability of this central node. Furthermore, once a call is admitted, the identities of the links along its chosen path have to be communicated to the nodes along that path.

In this work we resolve this issue. We give new online call control algorithms for general network topologies that work distributively. Furthermore, our algorithms can handle concurrent requests in the network. Our algorithms have a path-establishment and call admission procedure that can operate in a hop-by-hop manner through the network. That is, for a given request for a call, a sequence of request-messages is sent along some path between the communication parties, where the specific path can be determined in a hop-by-hop manner. Based on the loads on the edges along this path the call admission decision is taken. If the call is admitted, it is routed along the same path. We give two algorithms. Both guarantee that, for any sequence of requests for calls, with high probability, the number of accepted calls is a polylogarithmic fraction (more precisely an $O(\log^3 n \log \log n)$ fraction, where n is the number of nodes in the network) of the optimum number of calls that can be accepted. Our algorithms require that there is a lower bound on the capacity of any network link. We note that results in [BFL96] imply that for any online algorithm on general networks to guarantee (even on expectation) an $o(n^\epsilon)$ fraction of the optimum solution, this is necessary. The requirement of our first algorithm is that link capacities are $\Omega(\log^2 n (\log \log n + \Delta(G)))$, where $\Delta(G)$ is a function of the network. The parameter $\Delta(G)$ is at most n in general, but for specific topologies such as the mesh it is a constant. Since for some networks this minimum capacity requirement may be large, we also give another algorithm, where the edge capacity requirement is $\Omega(\log^3 n \log \log n)$. We are able to achieve that at the cost that the routes of the calls are not always guaranteed to be simple paths.

Both our algorithms have the following property (that we find elegant). The admission decision for a

*Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213, USA. email: harry@cs.cmu.edu.

†Dept. of Computer Science, Technion, Haifa 32000, Israel. email: adiro@cs.technion.ac.il. Research supported in part by a grant from the G.I.F., the German-Israeli Foundation for Scientific Research and Development.

given call is based only on the link loads along a single path between the communicating parties, where (the randomized) choice of the specific path does not depend on previous calls in the network. Furthermore, the admission decision is simple, in that the call is rejected only if for some link along that path, the number of calls (of a certain type) already on the link, exceeds some threshold. The call itself, if admitted, is then routed on the same path.

Related work. The online call control problem has been extensively studied. We briefly review below the results that are most closely related to the present work. A deterministic online algorithm for call control on general networks with an $O(\log n)$ competitive ratio is given in [AAP93]. This algorithm requires that each link capacity is $\Omega(\log n)$. This has later been extended in [KT95] to be $O(\log D)$ competitive with edge capacity requirement of $\Omega(\log D)$, where D is the length of the longest simple path in the network. The results in [BFL96] imply that in order to achieve on general networks an $o(n^\epsilon)$ competitive ratio by an online (even randomized) algorithm, some minimum edge capacity is required. In [AA94] a competitive distributed algorithm for a class of problems, including call control, is given. The obtained competitive ratios are polylogarithmic when the number of possible paths for a given call is polynomial (which is not the case in general network topologies).

Our work makes use of the oblivious routing schemes recently given in [Răc02] and subsequently improved in [HHR03]. The latter result gives a virtual circuit routing algorithm that chooses routing paths independently of the traffic in the network and obtains a competitive ratio of $O(\log^2 n \log \log n)$ with respect to edge congestion (i.e., maximum load of a network link, normalized by link capacity). In Section 2 we give a brief overview of this routing scheme.

Organization. The rest of the paper is organized as follows. In Section 2 we give some preliminary definitions and notations, and review results from [HHR03] and [BKR03] that we use for our algorithms. In Sections 3 and 4 we give our algorithms and analyze their performance. In Section 5 we show how to make our algorithms distributed, and how they can be made to handle concurrent requests in the network.

2 Preliminaries

We model the network as an undirected graph $G(V, E)$, $|V| = n$, $|E| = m$. Each edge $e \in E$ has a capacity $\text{cap}(e)$. That is, the number of calls that can be routed on edge e is at most $\text{cap}(e)$. A request for a call consist of a pair of nodes (s, t) , $s \in V$, $t \in V$, and the requests arrive in an online manner. That is, a request (s, t)

is generated at the node s , and the call has either to be rejected or accepted. If the call is accepted a path between s and t has to be assigned to it, obeying the capacities of the edges. For a call control algorithm A and a request sequence σ we denote by $A(\sigma)$ the number of calls accepted by A . OPT will usually denote the optimal algorithm, that accepts the optimum (i.e., maximum cardinality) set of calls.

Oblivious routing. Most of this section is devoted to a short description of the oblivious routing schemes as given in [HHR03] and [BKR03], and to point out some key properties of these schemes that are useful for our purposes (the only difference between the two schemes that is relevant for our application is that [HHR03] guarantees a much better bound on the congestion). The various properties that we state below are proved in [HHR03].

The scheme uses a decomposition tree T_G for the network G , which is a rooted tree whose leaf nodes correspond to vertices of G , i.e., there is a 1-1 relation between nodes of G and leaf nodes of T_G . Each inner node $v_t \in T_G$ corresponds to a subset of nodes of G , namely the set of graph nodes that correspond to a leaf node in the subtree rooted at v_t . We call this set the *cluster* corresponding to v_t and denote it by S_{v_t} . The *subclusters* or *child-clusters* of a cluster S_{v_t} are the clusters that correspond to a child of v_t in the tree.

For every cluster S there are two weight functions on the set of nodes of S . The first function $w_S : S \rightarrow \mathbb{R}^+$ counts for each node $v \in S$ the capacity of edges that are adjacent to v and leave a subcluster of S (i.e., edges (u, v) for which u is not in the same subcluster as v). The second weight function $w_S^b : S \rightarrow \mathbb{R}^+$ counts for a node v the capacity of adjacent edges that leave S , i.e., it counts the capacities of all edges of the form (u, v) , where $u \notin S$. For convenience we introduce the notation $w_S(X) := \sum_{x \in X} w_S(x)$ (resp. $w_S^b(X) := \sum_{x \in X} w_S^b(x)$) for subsets $X \subset S$.

The height of T_G is denoted $h(T_G)$. We denote by $\Delta(T_G)$ the maximum degree of a tree node plus 1 (i.e., the maximum number of edges adjacent to a tree node, including the edge leading to its parent). Every edge in the tree is also assigned a capacity in the following way. The capacity of a tree edge $e_t = (v_t, p_t)$ connecting a child v_t to its parent p_t is defined as $w_{S_{v_t}}^b(S_{v_t})$, i.e., the total capacity of edges leaving S_{v_t} .

Since the leaves of the decomposition tree represent the nodes of G there is for every call request (s, t) , $s, t \in V$ a natural, unique path in T_G between the leaves of s and t that passes through a sequence of clusters. In the sequel we will consider the imaginary admission of the call requests on the decomposition tree as well.

Given a sequence of requests of calls, we consider

the optimum (i.e., maximum cardinality) set of calls that can be routed on G without violating edge capacities, and also the optimal set of calls that can be routed (imaginarily) on T_G (as defined above) without violating the capacities of the edges of T_G . A key property of the construction of the decomposition, and of the definition of T_G and the capacities of its edges, is that any set of calls that can be routed in G without violating any link capacity, can also be routed on T_G without violating the edge capacities in T_G . Therefore, we can state the following lemma.

LEMMA 2.1. *Let $\text{Opt}(G)$ and $\text{Opt}(T_G)$ denote the optimum number of calls accepted in the graph G and the decomposition tree T_G , respectively. Then*

$$\text{Opt}(T_G) \geq \text{Opt}(G) .$$

In our proofs we will compare the performance of our algorithms to the performance of an adversary on T_G rather than on G . The same results will then apply against the adversary that routes on G itself.

The second main feature of the oblivious routing scheme is a simulation of the routing of any set of calls on T_G , on the graph G itself, while preserving “low” edge congestion on G . To this end, for every leaf-to-leaf path on T_G the oblivious routing scheme defines a randomized process to select a path on G between the two corresponding nodes in G . Observe that this in fact defines a randomized process to select a path between any two nodes in the graph G . We now define this process. Let the two leaf nodes be u_t and v_t (corresponding to clusters $\{u\}$ and $\{v\}$, respectively). Let $u_t = w_t^1, w_t^2, \dots, w_t^k = v_t$ denote the sequence of tree nodes on the path from u_t to v_t in T_G . For every such tree node w_t^i the routing process chooses a random graph node w^i , from within the set of nodes of the cluster, according to the weight function $w_{S_{v_t}}$, i.e., the probability that a particular node x is chosen is $w_{S_{v_t}}(x)/w_{S_{v_t}}(S_{v_t})$. The chosen path on G will go through the sequence of these intermediate nodes. (Note that this means that the endpoints of the path in G will be the nodes that correspond to the leaves in T_G , i.e., u and v).

For choosing the routing paths between these randomly chosen intermediate nodes, each cluster S in the decomposition tree has the solution to a certain concurrent multicommodity flow problem (CMCF-problem) defined on the subgraph induced by S . The definition of the CMCF-problem for cluster S is as follows. There is one commodity g_{uv} for every node-pair $(u, v) \in S \times S$. The demand for commodity g_{uv} is $w_S(u) \cdot w_S(v)/w_S(S)$.

The solution of the CMCF problem is used as follows. Suppose there are two consecutive tree nodes w_t^i and

w_t^{i+1} , where w.l.o.g. w_t^i is the child and w_t^{i+1} is its parent. The routing algorithm chooses a random “border node” b of $S_{w_t^i}$, i.e., a node chosen according to “border weight” of $S_{w_t^i}$ (i.e., the probability that a node v is chosen is $w_{S_{w_t^i}}^b(v)/w_{S_{w_t^i}}^b(S_{w_t^i})$). Then it chooses the path from w^i to w^{i+1} as a concatenation of a path from w^i to b and a path from b to w^{i+1} . The first path (from w^i to b) is chosen according to the flow for commodity $g_{w^i b}$ in the CMCF-solution for $S_{w_t^i}$. The second path is chosen according to the flow of commodity $g_{bw^{i+1}}$ in the CMCF-problem for $S_{w_t^{i+1}}$. (A CMCF-solution that contains commodity g_{xy} for some $x, y \in V$ provides a flow between x and y for this commodity. Choosing a path according to this flow means to decompose the flow into a convex combination over x - y paths, and to choose one of these paths at random). In the following we call the first type of paths a *child path* for tree edge (w_t^i, w_t^{i+1}) and the second type a *parent path* for this tree edge.

Let $e_t = (v_t, p_t)$ denote a tree edge that connects v_t to its parent p_t . We give the following notations for the probabilities implied by the above process.

DEFINITION 2.1. *Let $P_{e_t}^c(e)$ denote the probability that a child-path for a tree edge e_t contains the edge e . Let $P_{e_t}^p(e)$ denote the probability that a parent-path for a tree edge e_t contains the edge e .*

In the following we relate the probabilities $P_{e_t}^c(e)$ and $P_{e_t}^p(e)$ to the flow that traverses e in the CMCF-solutions for cluster S_{v_t} and S_{p_t} . We can write $P_{e_t}^c(e)$ as

$$P_{e_t}^c(e) = \sum_{v \in S_{v_t}} \sum_{b \in S_{v_t}} \frac{w_{S_{v_t}}(v)}{w_{S_{v_t}}(S_{v_t})} \cdot \frac{w_{S_{v_t}}^b(b)}{w_{S_{v_t}}^b(S_{v_t})} \cdot \text{Pr}[g_{vb}\text{-path cont. } e].$$

This holds because $w_{S_{v_t}}(v)/w_{S_{v_t}}(S_{v_t})$ is the probability that the path starts at v and $w_{S_{v_t}}^b(b)/w_{S_{v_t}}^b(S_{v_t})$ is the probability that it ends at b . Let $f_{vb}^{S_{v_t}}(e)$ denote the amount of flow that traverses e for commodity g_{vb} in the CMCF-solution for S_{v_t} . The probability that e is chosen as a path for commodity g_{vb} is $f_{vb}^{S_{v_t}}(e)/\text{dem}_{vb}(S_{v_t})$, where $\text{dem}_{vb}(S_{v_t}) = w_{S_{v_t}}(v)w_{S_{v_t}}(b)/w_{S_{v_t}}(S_{v_t})$ is the total flow between v and b in the CMCF-solution for S_{v_t} .

Plugging in the values gives

$$(2.1) \quad \begin{aligned} P_{e_t}^c(e) &= \sum_{v \in S_{v_t}} \sum_{b \in S_{v_t}} \frac{w_{S_{v_t}}(v)}{w_{S_{v_t}}(S_{v_t})} \cdot \frac{w_{S_{v_t}}^b(b)}{w_{S_{v_t}}^b(S_{v_t})} \cdot \frac{f_{vb}^{S_{v_t}}(e)}{\text{dem}_{vb}(S_{v_t})} \\ &\leq \frac{1}{\text{cap}(e_t)} \sum_{v \in S_{v_t}} \sum_{b \in S_{v_t}} f_{vb}^{S_{v_t}}(e) , \end{aligned}$$

where the inequality follows from $w_{S_{v_t}}^b(b) \leq w_{S_{v_t}}(b)$, and $\text{cap}(e_t) = w_{S_{v_t}}^b(S_{v_t})$.

An analogous calculation for $P_{e_t}^p(e)$ gives

$$(2.2) \quad P_{e_t}^p(e) = \frac{1}{\text{cap}(e_t)} \sum_{b \in S_{v_t}} \sum_{p \in S_{p_t}} f_{bp}^{S_{p_t}}(e) .$$

The results in [HHR03] give a decomposition tree T_G , such that (1) $h(T_G) = O(\log n)$ and (2) the CMCF-problem for each cluster can be solved with small congestion internally in that cluster. More precisely, the load on a graph edge due to the solution of *all* CMCF-problems together is at most $O(\log^2 n \cdot \log \log n) \cdot \text{cap}(e)$, i.e., the congestion of e is small. In the following we denote this factor, i.e., the maximum ratio between the CMCF-induced load of an edge and the capacity of the edge, with L_{\max} .

3 Algorithm Admit on Edge-Channels

Given a graph G , we use the decomposition tree T_G and its properties as given in [HHR03] and [BKR03] and briefly described in the previous section. In order to define our call control algorithm we split the capacity of any graph edge e into a number of distinct channels. We say that an edge e is contained in a cluster S_{v_t} of the decomposition tree if both its endpoints are in S_{v_t} . For every cluster S_{v_t} in which e is contained, and for any edge e_t adjacent to v_t in T_G , there will be 2 channels. One channel $C_{e_t}^c(e)$ will be used for child-paths of e_t and one channel $C_{e_t}^p(e)$ for parent paths of e_t . Note that an edge e is used in a (child or parent) path for a tree edge $e_t = (u_t, v_t)$ only if either S_{u_t} or S_{v_t} contain e . We assign a capacity to a channel in the following way. For child-path channels we define

$$\text{cap}(C_{e_t}^c(e)) = 2\alpha \cdot P_{e_t}^c(e) \cdot \text{cap}(e_t) + k \cdot \ln n .$$

and for parent path channels

$$\text{cap}(C_{e_t}^p(e)) = 2\alpha \cdot P_{e_t}^p(e) \cdot \text{cap}(e_t) + k \cdot \ln n .$$

The factor α is chosen such that the total capacity of the channels on any given edge does not exceed the capacity of that edge. Lemma 3.1 shows that it is possible to choose $\alpha = \Omega(1/\log^2 n \log \log n)$. The constant k is an appropriately chosen constant to satisfy the requirements of Lemma 3.2 below. Before we give the proof of Lemma 3.1 we provide a bound on the number of channels on a graph edge e .

CLAIM 1. *The number of channels on a given graph edge e is at most $2h(T_G)\Delta(T_G)$.*

Proof. Edge e has 2 channels for every tree edge e_t which is adjacent to a tree node v_t , such that e is in the cluster

S_{v_t} . Therefore it remains to give an upper bound on the number of such edges. Consider the decomposition tree. Edge e is contained in a set of clusters whose corresponding tree nodes appear along a path from the root of T_G towards its leaves. The length of this path is at most $h(T_G)$, and each node along this path has at most $\Delta(T_G)$ edges adjacent to it. The claim follows. \square

LEMMA 3.1. *Suppose that for all edges $e \in E$, $\text{cap}(e) \geq 4k \log n \cdot \Delta(T_G)h(T_G)$. Then there exists a scaling factor $\alpha = \Omega(1/\log^2 n \log \log n)$ such that for any edge e*

$$\sum_{e_t} (C_{e_t}^c(e) + C_{e_t}^p(e)) \leq \text{cap}(e) ,$$

i.e., the total capacity of edge channels does not exceed the capacity of the edge.

Proof. Due to inequalities 2.1 and 2.2 we have

$$\sum_{e_t} P_{e_t}^c(e) \cdot \text{cap}(e_t) \leq \sum_{e_t=(v_t, p_t)} \sum_{v \in S_{v_t}} \sum_{b \in S_{v_t}} f_{vb}^{S_{v_t}}(e)$$

and

$$\sum_{e_t} P_{e_t}^p(e) \cdot \text{cap}(e_t) = \sum_{e_t=(v_t, p_t)} \sum_{b \in S_{v_t}} \sum_{p \in S_{p_t}} f_{bp}^{S_{p_t}}(e) .$$

Observe that no commodity appears twice in the right-hand side of one of the above equations. Therefore each right-hand side is bounded by the total flow that traverses e due to the solutions of all CMCF-problems. This is at most $L_{\max} \cdot \text{cap}(e)$ due to [HHR03].

Therefore, we can choose $\alpha = \Theta(1/L_{\max}) = \Omega(1/\log^2 n \log \log n)$ such that

$$2\alpha \cdot \sum_{e_t} (P_{e_t}^c(e) + P_{e_t}^p(e)) \cdot \text{cap}(e_t) \leq \frac{\text{cap}(e)}{2} .$$

This gives

$$\begin{aligned} & \sum_{e_t} (C_{e_t}^c(e) + C_{e_t}^p(e)) \\ &= 2\alpha \sum_{e_t} (P_{e_t}^c(e) + P_{e_t}^p(e)) \cdot \text{cap}(e_t) + \sum_{e_t} 2k \log n \\ &\leq \text{cap}(e)/2 + \Delta(T_G) \cdot h(T_G) \cdot 2k \log n \\ &\leq \text{cap}(e) , \end{aligned}$$

as desired. \square

The algorithm. We are now ready to formally define our first algorithm, that we denote AEC, for Admit on Edge-Channels. Given a request for a call (s, t) the algorithm chooses a random path in G , and for each edge along that path it specifies the channel of that edge to be used. That is, the algorithm in fact specifies a path of channels that connect between s and t . The

random path is chosen by the same process described in Section 2. The channel to be used on each edge is selected as follows. When edge $e \in E$ is used in the child path for tree edge e_t , then channel $C_{e_t}^c(e)$ is used, and when $e \in E$ is used in the parent path for tree edge e_t , then channel $C_{e_t}^p(e)$ is used. The call is accepted on the selected path if and only if the remaining bandwidth on all the *selected channels* along its path is sufficient to add the call.

For a given (s, t) pair, the above randomized process defines a distribution over paths of channels between s and t . These paths of channels are however not necessarily simple paths in the graph. For any given path of channels we can eliminate cycles (if any) in that path, so as to transform it into a path of channels between s and t which is also a simple path in the graph. In the following we analyze the algorithm that uses the original (possibly non-simple) paths. Our proof will go through with the same results also for the algorithm that uses the simple paths.

In Section 5 we show how to make the algorithm distributed. We show how the above path selection and call admission procedure can be performed in a distributed hop-by-hop manner through the network, and how this procedure can be further modified so as to guarantee that the selected routes are simple paths. Furthermore, we show how to modify the algorithm so that it can handle concurrent requests in the network.

3.1 Analysis. We now give the analysis of our algorithm. We assume for the analysis that the capacity of each edge e is $\Omega(\log^2 n \log \log n + \log n \cdot h(T_G)\Delta(T_G))$. In the analysis of our algorithm we make use of the decomposition tree, while shrinking the capacities of all edges by the factor α as defined above. We denote this tree by T_G^α . For the sake of the analysis, when a call is accepted by AEC we will also consider its acceptance and natural routing on T_G^α (as explained in Section 2) and the bandwidth it consumes on the edges of that tree. A key point in the analysis is the relation between the actual load on a given tree edge e_t in T_G^α and the load on any of the channels (in the graph G) that correspond to e_t . We give the following definition.

DEFINITION 3.1. A channel $C_{e_t}^p(e)$ (resp. $C_{e_t}^c(e)$), for some $e_t \in T_G^\alpha$ and $e \in G$ is said to be overused if, when the algorithm terminates, the available bandwidth on $C_{e_t}^p(e)$ (resp. $C_{e_t}^c(e)$) is less than 1 (i.e., the channel is saturated), and the number of accepted calls that use e_t on T_G^α is less than $\lfloor \alpha \cdot \text{cap}(e_t) \rfloor$ (i.e., e_t is not saturated on T_G^α).

LEMMA 3.2. For a given channel $C_{e_t}^p(e)$ (resp. $C_{e_t}^c(e)$), the probability that it is overused is at most $n^{-k/3}$

(where k is the constant used in the definition of channel capacities).

Proof. We prove the lemma for $C_{e_t}^p(e)$. The proof for $C_{e_t}^c(e)$ is analogous. The channel is overused only if the corresponding tree edge e_t on T_G^α is not saturated. That is, the number of accepted calls that use it is less than $\lfloor \alpha \cdot \text{cap}(e_t) \rfloor$. Since only calls that use e_t on T_G^α may use channel $C_{e_t}^p(e)$, the expected number of calls that use $C_{e_t}^p(e)$ is bounded from above by $\mu := P_{e_t}^p(e) \cdot \lfloor \alpha \cdot \text{cap}(e_t) \rfloor$. On the other hand, the channel is overused only if the number of calls that use it is more than $2\alpha \cdot P_{e_t}^p(e) \cdot \text{cap}(e_t) + k \ln n \geq 2\mu + k \ln n$. Using Chernoff bounds (See Corollary A.1 in the appendix), the probability of this event is at most $n^{-k/3}$. \square

COROLLARY 3.1. The probability that at least one channel is overused is at most $O(\frac{1}{n})$.

Proof. The total number of channels on any given edge is at most $4n$ since there are at most $2n$ tree edges and each tree edge may induce at most 2 channels on an edge. It follows that the total number of channels in the graph is $O(n^3)$. The corollary then follows by using Lemma 3.2 and a simple union bound (for an appropriately chosen constant k in the definition of the channel capacities). \square

We now consider the performance of the algorithm when no channel is overused. The above implies that this happens with probability at least $1 - O(\frac{1}{n})$. We now show that in this case the number of calls AEC accepts is at least a “good” fraction of the number of calls accepted by the optimal algorithm.

LEMMA 3.3. Consider a request sequence σ and a run of AEC on σ such that no channel is overused. If $\forall e \in E$, $\text{cap}(e) \geq \frac{1}{\alpha}$ then $\text{OPT}(\sigma) \leq (1 + \frac{4 \cdot h(T_G)}{\alpha}) \text{AEC}(\sigma)$. (Where $\text{AEC}(\sigma)$ and $\text{OPT}(\sigma)$ are the number of calls accepted by AEC, and by OPT, respectively, out of request sequence σ).

Proof. Let C_a be the set of calls accepted by AEC, and C_{opt} the set of calls accepted by OPT. Consider a call $c \in C_{\text{opt}} \setminus C_a$. Since AEC did not accept c , the path that was randomly chosen by AEC for c contained a channel to which c could not be added. In other words, when c was presented to AEC the available bandwidth on that channel was less than 1, and therefore when AEC terminates the available bandwidth on this channel is less than 1. Let this channel be a channel that corresponds to tree edge e_t . Since we assume that no channel is overused, it follows that when AEC terminates e_t on T_G^α is saturated (by calls accepted by AEC, and imaginarily routed also on T_G^α).

Let E_{sat} denote the set of edges of T_G^α that are saturated when AEC terminates. We have shown above that any call $c \in C_{\text{opt}} \setminus C_a$ must contain an edge in E_{sat} on T_G^α . On the other hand every call in C_{opt} uses bandwidth 1 on each edge that it contains on T_G^α . Therefore, $|C_{\text{opt}} \setminus C_a| \leq \sum_{e_t \in E_{\text{sat}}} \text{cap}(e_t)$. At the same time, $|C_a| \geq \frac{1}{2 \cdot h(T_G)} \cdot \sum_{e_t \in E_{\text{sat}}} [\alpha \cdot \text{cap}(e_t)] \geq \frac{\alpha}{4 \cdot h(T_G)} \cdot \sum_{e \in E_{\text{sat}}} \text{cap}(e_t)$, because each accepted call only uses capacity on at most $2 \cdot h(T_G)$ edges and the total used capacity by accepted calls is at least $\sum_{e_t \in E_{\text{sat}}} [\alpha \cdot \text{cap}(e_t)]$ (using $\alpha \cdot \text{cap}(e_t) \geq 1$ for any e_t , which follows from the fact that the capacity of e_t is lower bounded by the minimum capacity of an edge $e \in E$). This gives

$$|C_{\text{opt}}| \leq |C_{\text{opt}} \setminus C_a| + |C_a| \leq \left(1 + \frac{4 \cdot h(T_G)}{\alpha}\right) \cdot |C_a|$$

□

Using the upper bound on $h(T_G)$ from [HHR03], the chosen value of α , and the requirement we stated above for the minimum capacity of an edge we can conclude with the following theorem.

THEOREM 3.1. *For any graph G with edge capacities at least $\max(\frac{1}{\alpha}, 4k \log n \cdot \Delta(T_G)h(T_G)) = O(\log^2 n(\log \log n + \Delta(T_G)))$ and for any sequence of requests σ , with probability at least $1 - O(\frac{1}{n})$, $\text{OPT}(\sigma) \leq (1 + \frac{4 \cdot h(T_G)}{\alpha}) \text{AEC}(\sigma) = O(\log^3 n \log \log n) \cdot \text{AEC}(\sigma)$.*

The parameter $\Delta(T_G)$ in the above theorem is the maximum degree of the decomposition tree. For meshes, e.g., $\Delta(T_G)$ is constant. However, we do not have a general non-trivial upper bound on $\Delta(T_G)$ in terms of network parameters, other than n . Therefore, for some network topologies AEC may require large edge capacities to perform well. In the following section we introduce another algorithm that addresses this issue.

4 Improved edge-capacity requirement

The algorithm from the previous section uses channels at the graph edges in order to decide whether to accept or reject calls in the network. One drawback of this approach is that it may require large edge-capacities (i.e., $\Omega(\log n \cdot \Delta(T_G) \cdot h(T_G))$) because the capacity of all edge-channels must not exceed the capacity of the edge, and each channel should have a capacity of at least $\Omega(\log n)$ (in order to allow for a “high probability argument”).

In this section we present a second algorithm that has a substantially lower edge-capacity requirement. This is obtained however at the cost that the algorithm may use non-simple routing paths.

The idea of the algorithm is to set up (virtual) *node channels* instead of edge channels. A graph node v is

assigned one channel $C_S(v)$ for every cluster S such that S contains v , and $w_S^b(v) > 0$. We define the capacity of the channel $C_S(v)$ as

$$\text{cap}(C_S(v)) = 2\alpha \cdot w_S^b(v) + k \cdot \ln n .$$

Here we choose α such that $\forall v, S$, $\text{cap}(C_S(v)) \leq \frac{1}{4L_{\text{max}}} w_S^b(v)$. (Recall that L_{max} is the maximum congestion of a graph edge due to the solution of all CMCf-problems.) The constant k is an appropriately chosen constant to satisfy the requirements of Lemma 4.2

LEMMA 4.1. *Suppose that for all edges $e \in E$, $\text{cap}(e) \geq 8k \ln n \cdot L_{\text{max}}$. Then there exists a scaling factor $\alpha = \Omega(1/\log^2 n \log \log n)$ such that for any channel $C_S(v)$*

$$\text{cap}(C_S(v)) \leq \frac{1}{4L_{\text{max}}} \cdot w_S^b(v) .$$

Proof. Note that $w_S^b(v)$ is either 0 or at least the smallest edge capacity, because the function w_S^b counts the edge capacities of edges which are adjacent to v and cross the boundaries of cluster S . Therefore, $k \cdot \ln n \leq 1/(8L_{\text{max}}) \cdot w_S^b(v)$. Now, choosing $\alpha = 1/(8L_{\text{max}})$ gives the result. □

The algorithm. Given a request for a call (s, t) the algorithm chooses a random path according to the scheme described in Section 2. This path consists of subpaths between randomly chosen nodes $s = w^1, w^2, \dots, w^\ell = t$. For routing between two consecutive nodes w^i and w^{i+1} (which correspond to tree edge (w_t^i, w_t^{i+1})) a random intermediate node b at the border of $S_{w_t^i}$ is chosen (cf. Section 2). Then a child path from w^i to b and a parent path from b to w^{i+1} are selected. The call admission decision is done at the border nodes b as follows. If a call is routed such that node b is the border node for some tree edge $e_t = (v_t, p_t)$, we say that this call uses channel $C_{S_{v_t}}(b)$ at node b . The node b rejects the call if the channel to be used by the call is already saturated. A call is accepted by the algorithm if and only if it is accepted by all intermediate border nodes along its path *and* all graph edges along its path have enough available capacity to add the call (i.e., all edges along the path are not saturated). We denote this algorithm with ANC (for Admit on Node-Channels) in the following.

In Section 5 we show how to make the algorithm distributed. We show how the above path selection and call admission procedure can be performed in a distributed hop-by-hop manner through the network, and how to further adapt this procedure to handle concurrent requests in the network.

4.1 Analysis As in the previous section we relate the saturation of channels to the load on the corresponding tree edges.

DEFINITION 4.1. Let $e_t = (v_t, p_t)$ denote a tree edge with child v_t and parent p_t . The channel $C_{S_{v_t}}(v)$ of a node v is said to be overused if, when the algorithm terminates, $\lfloor \text{cap}(C_{S_{v_t}}(v)) \rfloor$ calls are routed on this channel (i.e., the channel is saturated), and the number of accepted calls that use e_t on T_G^α is less than $\lfloor \alpha \cdot \text{cap}(e_t) \rfloor$ (i.e., e_t is not saturated on T_G^α).

LEMMA 4.2. The probability that a given channel $C_S(v)$ is overused is at most $n^{-k/3}$, where k is the constant used in the definition of the channel capacity.

Proof. Fix a tree edge $e_t = (v_t, p_t)$ and a graph node $v \in S_{v_t}$. We consider the channel $C_{S_{v_t}}(v)$. The channel is only overused if the corresponding tree edge e_t on T_G^α is not saturated. This means that the number of calls on e_t is less than $\lfloor \alpha \cdot \text{cap}(e_t) \rfloor$. Therefore the expected number of calls that use channel $C_{S_{v_t}}(v)$ is less than $\mu := w^b S_{v_t}(v) / w_{S_{v_t}}^b(S_{v_t}) \cdot \alpha \cdot \text{cap}(e_t) = \alpha \cdot w_{S_{v_t}}^b(v)$ (Recall that $w^b S_{v_t}(v) / w_{S_{v_t}}^b(S_{v_t})$ is the probability that v is chosen as an intermediate border node for the edge e_t). On the other hand, the channel is only overused if the number of calls that use it is at most $2\alpha \cdot w_{S_{v_t}}^b(S_{v_t}) + k \ln n$. Using a Chernoff bound gives the result. \square

COROLLARY 4.1. The probability that at least one channels is overused is at most $O(\frac{1}{n})$.

Proof. Follows from the above lemma, an appropriately chosen constant k in the definition of the channel capacities, and the fact that there are at most $n \cdot h(T_G) = O(n \log n)$ channels. \square

The key to the analysis in the previous section was that only saturated channels may reject a call. Then, assuming that a channel is only saturated if the corresponding tree edge is saturated (i.e., no channel is overused, an event that we show to occur with high probability) one can use Lemma 3.3 to show that a large fraction of calls is accepted. In the present algorithm calls may also be rejected because the capacity of a graph edge is exceeded. The following lemma shows that with high probability this does not happen.

LEMMA 4.3. The probability that during the run of ANC an edge rejects a call that is accepted by the admission process on the node channels, is at most $O(\frac{1}{n})$, provided that the capacity of an edge is at least $18 \cdot \ln n$.

Proof. Let $\text{load}_S^p(e)$ denote a random variable that describes the load of an edge e due to parent-paths that are chosen according to some commodity in the CMCF-solution for cluster S . The probability that a parent path starting at border node b uses edge e is

$$\sum_{v \in S} \frac{w_S(v)}{w_S(S)} \cdot f_{bv}^S(e) / \frac{w_S(v)w_S(b)}{w_S(S)} = \sum_{v \in S} \frac{f_{bv}^S(e)}{w_S(b)},$$

since $w_S(v)/w_S(S)$ is the probability that the parent-path ends at v and $f_{bv}^S(e) / \frac{w_S(v)w_S(b)}{w_S(S)}$ is the probability that in this case it contains edge e . Multiplying this with the maximum number of child paths that can start at b (at most $\text{cap}(C_{S'}^b(v)) \leq w_{S'}^b(b) / (4L_{\max}) = w_S(b) / (4L_{\max})$, where S' denotes the child cluster of S that contains b) gives that the expected load on e for parent-paths starting at b is smaller than $\frac{1}{4L_{\max}} \sum_{v \in S} f_{bv}^S(e)$. Summing this over all b we get

$$(4.3) \quad \mathbb{E}[\text{load}_S^p(e)] \leq \frac{1}{4L_{\max}} \sum_{b \in S} \sum_{v \in S} f_{bv}^S(e).$$

For child-paths we get that the expected load $\text{load}_S^c(e)$ of an edge e is at most

$$(4.4) \quad \mathbb{E}[\text{load}_S^c(e)] = \sum_{b \in S} \sum_{v \in S} \frac{w_S(v)}{w_S(S)} \cdot \frac{f_{vb}^S(e)}{\text{dem}_{vb}(S)} \cdot \frac{w_S^b(b)}{4L_{\max}} \\ \leq \frac{1}{4L_{\max}} \sum_{b \in S} \sum_{v \in S} f_{vb}^S(e),$$

where $\text{dem}_{vb}(S) = w_S(v)w_S(b)/w_S(S)$ denotes the demand between v and b in the CMCF problem for cluster S . Note that the summations in the above inequalities contain each commodity at most once. Therefore the total expected load due to child and parent paths summed over all clusters is at most $\frac{1}{4L_{\max}} \text{CMCF}(e)$, where $\text{CMCF}(e)$ denotes the load of e due to the solution of all CMCF-problems. This is at most $\mu := \text{cap}(e)/4$ due to the definition of L_{\max} . An edge has to reject a call if the load is larger than $\text{cap}(e) \geq 2\mu + 9 \ln n$. Using a Chernoff bound this happens with probability smaller than n^{-3} . Now, applying a union bound for the at most n^2 edges gives the lemma. \square

Assume now that no (node) channel is overused, and no call is rejected on an edge. Using arguments similar to those in the proof of Lemma 3.3 we have the following.

LEMMA 4.4. Consider a request sequence σ and a run of ANC on σ such that no node channel is overused, and no call is rejected on an edge. If $\forall e \in E, \text{cap}(e) \geq \frac{1}{\alpha}$ then $\text{OPT}(\sigma) \leq (1 + \frac{4 \cdot h(T_G)}{\alpha}) \text{ANC}(\sigma)$. (Where $\text{ANC}(\sigma)$ and $\text{OPT}(\sigma)$ are the number of calls accepted by ANC, and by OPT, respectively, out of request sequence σ).

We conclude with the following theorem which follows from Corollary 4.1 and Lemmas 4.3 and 4.4.

THEOREM 4.1. For any graph G with edge capacities at least $\max(\frac{1}{\alpha}, 8k \ln n \cdot L_{\max}) = O(\log^3 n \log \log n)$,

and for any sequence of requests σ , with probability at least $1 - O(\frac{1}{n})$, $\text{OPT}(\sigma) \leq (1 + \frac{4h(T_G)}{\alpha})\text{ANC}(\sigma) = O(\log^3 n \log \log n) \cdot \text{ANC}(\sigma)$.

5 Making the algorithms distributed

The description of our algorithms in Sections 3 and 4 did not specify how to perform the algorithms in a distributed hop-by-hop manner through the network. In this section we describe how to transform our algorithms into distributed ones. We further describe how to handle concurrent requests in the network.

5.1 Hop-by-hop procedures In this section we describe how the path-establishment and call admission procedures of our algorithms can be performed in a hop-by-hop manner through the network. Recall that in both algorithms, given an (s, t) request, a path between s and t is selected according to the process described in Section 2. For algorithm AEC a channel on each edge is also selected, and for algorithm ANC a (virtual) channel on some nodes is also selected.

The path between s and t is selected as a concatenation of paths between intermediate nodes, where each intermediate node corresponds to a tree node in T_G along the natural path in T_G between s and t . The first and last (“intermediate”) nodes along this path are s and t (cf. Section 2). Therefore, the selection of these intermediate nodes can be done in a hop-by-hop manner, where each intermediate node selects the next intermediate node according to the appropriate distribution. The path between two consecutive intermediate nodes, say v_1 and v_2 , is associated with some tree edge e_t in T_G . To select the path between v_1 and v_2 a “cluster border node” b is selected (cf. Section 2). This node can be selected by the first intermediate node (v_1), according to the appropriate distribution. The path between v_1 and v_2 is selected as a concatenation of a path between v_1 and b and a path between b and v_2 . Each of these paths is selected according to a flow between v_1 and b (resp. b and v_2) (cf. Section 2). Therefore, the actual path can also be selected in a hop-by-hop manner where each node along the path selects the next edge according to the v_1 -to- b (resp. b -to- v_2) flow that passes through it. The channel to be used on each edge (in algorithm AEC) is defined by the tree edge e_t that corresponds to the respective portion of the path between s and t . The (virtual) channel on the border nodes b (in algorithm ANC) is also defined by the tree edge e_t .

The path-selection (and channel selection) procedure can therefore be performed in a hop-by-hop manner by carrying along the two communicating parties (s and t), the tree edge currently being “simulated” (e_t), the next intermediate nodes (v_2) and the current border node (b).

The call will be admitted on the selected path after the hop-by-hop procedure reaches the other communicating party, and the admittance conditions have been met all along the path.

5.2 Simple paths For algorithm AEC we can modify the above procedure so as to guarantee that the selected routes for the calls are simple paths. Observe that for an (s, t) pair, the distribution over paths of channels is in fact a flow between s and t that uses the channels. At the same time this is also an s - t -flow on the edges of the graph, where the flow on an edge is the sum of flows on the channels of that edge. Thus, we can remove all the cycles from this flow. This does not increase the total flow on any edge. It remains however to define how to select a channel to be used on each edge, such that the new path (and channel) selection does not increase the load on any channel. For a given graph edge we distribute the (new) flow on that edge between the various channels, in the same proportions that the original flow on that edge was distributed between the channels. It follows that the flow on any channel is not larger than the flow on that channel in the original flow. We now have in each node, and for each (s, t) pair, a distribution over outgoing channels, which is defined by the flow on the edges leaving this node. The path-selection procedure can now be performed in a hop-by-hop manner by selecting the next channel in each node. The resulting path will be a simple path in the network. Since for a given (s, t) pair, the probability that a given channel is used does not increase (compared to the original algorithm), our analysis for the algorithm will go through with the same results.

5.3 Concurrent requests So far we have assumed that the routing requests do not appear concurrently but are serialized. For the case of concurrent requests we have to specify what to do if two hop-by-hop processes, of two call requests, “meet” in the network. We modify the hop-by-hop procedure such that when a request that wants to use some channel (an edge channel $C_{e_t}^c/C_{e_t}^p$ or a node channel $C_S(v)$, respectively) reaches this channel, it first makes a preliminary bandwidth reservation at this channel. After the call is accepted this bandwidth is allocated for the call. If the call is rejected the bandwidth reservation is discarded, and the bandwidth can therefore be used by other calls. If a request arrives at a channel for which the allocated bandwidth plus the reserved bandwidth saturates the channel, the request waits in a queue until either all bandwidth reservations have been actually allocated (in which case the call is rejected), or a call with an active bandwidth reservation is rejected (in this case the first call in the waiting queue resumes its

hop-by-hop procedure). Now observe that if every call is eventually accepted or rejected, then the algorithms will accept a large fraction of the optimum solution (as guaranteed by [Theorems 3.1](#) and [4.1](#)). To see that, observe that if we consider the requests for the calls in the order of their acceptance/rejection times in the above process, and apply the serialized algorithm (where for a call the same random choices are used), then the same calls will be accepted. It remains therefore to guarantee that indeed every call is eventually accepted or rejected, i.e., that no deadlock occurs. For this AEC and ANC have to be adapted in the following way.

Deadlock avoidance for ANC. In algorithm ANC deadlock avoidance is done as follows. We duplicate all node channels $C_S(v)$ into $C_S^1(v)$ and $C_S^2(v)$. All calls that enter a cluster from the parent cluster will use channel $C_S^1(v)$, and all calls that leave the cluster will use channel $C_S^2(v)$ at the respective node v . Then, because of the tree structure, no deadlocks can occur while waiting for node channels. To see that, observe that a request r , that is waiting at a node channel $C_S^2(v)$ of some node v , cannot block the request it is waiting for, because that request has left the cluster to some higher level of the decomposition tree. An analogous argument holds for requests waiting at a channel $C_S^1(v)$.

In order to avoid that edges create deadlocks we do the acceptance/rejection decisions of the calls in a two-phase process. In the first phase a call selects (in a distributed hop-by-hop manner) a route according to the random process described in [Section 2](#), and checks whether it is accepted at all the node channels on this route (possibly waiting for preliminary bandwidth reservations to be released). If not, the call is rejected. If the call is accepted at all the node channels on its route, then in a second phase the algorithm checks whether all the edges on the selected route have enough capacity left to accommodate the call. If all edges have enough capacity, the call is accepted, otherwise it is rejected. During this second phase no preliminary bandwidth reservation is used. Every call that tries to use an edge e and is not rejected by a node is counted for the load of e (even if the call is rejected by an edge).

[Lemma 4.3](#) guarantees that the probability that there is a call that is accepted on node channels but cannot be supported by the edges on its route is at most $O(1/n)$. Hence, the algorithm achieves the performance guarantee of [Theorem 4.1](#).

Deadlock avoidance for AEC. For the algorithm AEC deadlock avoidance is more involved. We give a short sketch. First we duplicate the channels in the same way as done for the algorithm ANC, i.e., we create channels $C_{e_t}^{c1}, C_{e_t}^{c2}, C_{e_t}^{p1}, C_{e_t}^{p2}$, for using e_t in the downward direction and the upward direction,

respectively. Channels are considered to be of the same “type”, if they are on different graph edges, but correspond to the same other parameters (i.e., tree edge e_t , parent/child path, and upward/downward direction).

Using the above defined channels, it is impossible that a request that waits for some channel corresponding to a tree edge e_t (in e.g. the upward direction) blocks in some way (i.e., via a circular wait condition) the call it is waiting for, *provided* that this call has already switched the type of channel it is currently waiting for. Therefore a circular wait condition (which may lead to a deadlock) may only occur on call requests concurrently waiting for the same channel type (i.e., tree edge e_t , parent/child path, and upward/downward direction) but on different graph edges.

In order to avoid such a circular wait situation we introduce a new CMCF-problem for a cluster S which is defined as follows. We add a super-source src_i and super-sink sink_i for each sub-cluster S_i . The source src_i is connected by directed edges to each node of S_i where the capacity of edge (src_i, v) is $w_S(v)/L_{\max}$. Every node of S is connect to sink_i with a directed edge of capacity $\frac{w_S(v)}{w_S(S)} \cdot w_S(S_i)/L_{\max}$. We consider the flow problem that has one commodity with demand $w_S(S_i)$ for each sub-cluster S_i between src_i and sink_i . This new flow problem can be solved as least as good as the original CMCF-problem w.r.t. congestion. To see this observe that first sending a demand of $w_S(v)$ from each src_i to $v \in S_i$; then distributing this demand according to the original CMCF-solution; and finally sending for all i a demand of $\frac{w_S(v)}{w_S(S)} \cdot w_S(S_i)$ from v to sink_i , solves the problem with congestion L_{\max} .

The parent paths for the routing scheme are chosen according to the new flow in the following way. Observe that each node $v \in S_i$ is traversed by a flow of $w_S(v)$ for commodity i . This corresponds to the probability that a parent path starts at v . From the starting node each node on the path chooses an outgoing edge with probability according to the flow along this edge. If at some node $v' \in S$ an edge to the super-sink sink_i is chosen, then v' becomes the “intermediate node” for cluster S , and the parent path ends (this means that intermediate nodes are chosen “on-the-fly”).

For the child paths we proceed similarly. We introduce a CMCF-problem with one commodity in which a super-source src is connected to all nodes of a cluster via directed edges. The capacity of edge (src, v) for $v \in S$ is defined as $\frac{w_S(v)}{w_S(S)} \cdot w_S^b(S)/L_{\max}$. Furthermore, a node is connected to a super-sink via an edge with capacity $w_S^b(v)/L_{\max}$. Again, this flow problem can be solved with congestion at most L_{\max} . The child paths can be chosen according to this flow such that the actual

path chosen determines the boundary node b .

The idea behind choosing the child and parent paths according to the new CMCF-flow problem is as follows. Both flow problems can be solved such that the flow for any commodity does not contain any cycles. By choosing e.g. the parent paths according to such a flow it is guaranteed that the union of all parent-paths originating from the same sub-cluster does not contain cycles. Therefore there will not be a circular wait condition for calls waiting on parent paths of the same sub-cluster. An analogous argument holds for child paths. Thus, the new mechanism for selecting child and parent paths avoids deadlocks.

The analysis of the algorithm has to be somewhat changed in the following way. Intermediate nodes are not independent any more. For routing along a tree edge e_t the algorithm uses the respective commodity with the respective CMCF-solution, and starts at the current node in the graph until it reaches the intermediate node or the border node, respectively (which are determined on-the-fly). This means that the next intermediate node (border node) that ends the current path may depend on the border node (intermediate node) at which the previous path ended. However, the overall probability that an (s, t) -call will use a specific channel $C_{e_t}^p(e)$ (resp. $C_{e_t}^c(e)$) only depends on the solution of the (new) CMCF-problems (more precisely it is equal to the fraction of the flow for the corresponding commodity that goes through e). This is all what is needed for proving [Lemma 3.2](#).

We can further transform this concurrent version of AEC into a version that routes along simple paths by using ideas similar to [Section 5.2](#).

Acknowledgments We thank Stefano Leonardi for useful discussions.

References

- [AA94] Baruch Awerbuch and Yossi Azar. Local optimization of global objectives: Competitive distributed deadlock resolution and resource allocation. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 240–249, 1994.
- [AAP93] Baruch Awerbuch, Yossi Azar, and Serge A. Plotkin. Throughput-competitive on-line routing. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 32–40, 1993.
- [BFL96] Yair Bartal, Amos Fiat, and Stefano Leonardi. Lower bounds for on-line graph problems with application to on-line circuit and optical routing. In *Proceedings of the 28th ACM Symposium on Theory of Computing (STOC)*, pages 531–540, 1996.
- [BKR03] Marcin Bienkowski, Mirosław Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 24–33, 2003.
- [HHR03] Chris Harrelson, Kirsten Hildrum, and Satish B. Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 34–43, 2003.
- [Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
- [KT95] Jon M. Kleinberg and Éva Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 52–61, 1995.
- [Räc02] Harald Räcke. Minimizing congestion in general networks. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 43–52, 2002. Co-Winner of Best Paper Award.

Appendix

A Chernoff bounds

We use the following form of the Chernoff-Hoeffding bounds, due to Hoeffding [[Hoe63](#)].

LEMMA A.1. ([[Hoe63](#)]) *Let X_1, \dots, X_n be independent random variables that take values in the range $[0, W]$ for some $W > 0$. Let $X = \sum_{i=1}^n X_i$, $\mu \geq E[X]$. Then for $\delta \geq 1$,*

$$\Pr[X \geq (1 + \delta)\mu] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^{\mu/W} \leq e^{-\frac{\delta\mu}{3W}}.$$

COROLLARY A.1. $\Pr[X \geq 2\mu + kW] \leq e^{-k/3}$.

Proof. We choose $\delta = 1 + \frac{W}{\mu}k$ in the above lemma. This gives $\Pr[X \geq 2\mu + kW] \leq e^{-(1+Wk/\mu)\mu/(3W)} \leq e^{-k/3}$, as desired. \square