Quorum Placement in Networks: Minimizing Network Congestion

Daniel Golovin* Anupam Gupta* Bruce M. Maggs*†‡ Florian Oprea† Michael K. Reiter*†

ABSTRACT

A quorum system over a universe of logical elements is a collection of subsets (*quorums*) of elements, any two of which intersect. In numerous distributed algorithms, the elements of the universe reside on the nodes of a physical network and the participating nodes access the system by contacting every element in some quorum, potentially causing the added network congestion induced by these quorum accesses to play a limiting factor in the performance of the algorithm.

In this paper we initiate the study of algorithms to place universe elements on the nodes of a physical network so as to minimize the network congestion that results from quorum accesses, while also ensuring that no physical node is overloaded by access requests from clients. We consider two models, one in which communication routes can be chosen arbitrarily and one in which they are fixed in advance. We show that in either model, the optimal congestion (with respect to the load constraints) cannot be approximated to *any* factor (unless P=NP). However, we show that at most doubling the load on nodes allows us to achieve a congestion that is close to this optimal value. We also shed some light on the extent to which element migration can reduce congestion in this context.

Categories and Subject Descriptors: C.2.4 [Computer- Communication Networks]: Distributed Systems - *distributed applications*

General Terms: Algorithms, Performance, Theory.

Keywords: Quorum Systems, Congestion Problems, Approximation Algorithms, LP Rounding.

1. INTRODUCTION

Given a universe (set) U of *elements*, a *quorum system* $\mathcal{Q} \subseteq 2^U$ (where 2^U denotes the power set of U) is a collection of subsets of U such that any two subsets in $\mathcal{Q}$ have a non-empty intersection. Each subset in $\mathcal{Q}$ is called a *quorum*. Quorums are the basic unit of access in many distributed algorithms; e.g., to ensure data consistency while allowing distributed access, one could use a quorum system: the copies of the object are its elements, and each client is required to perform each read or write on a quorum of objects. Since each pair of quorums intersect, this ensures that each client sees at least one copy of the latest version of the object. There are many other examples of using quorum systems to implement distributed coordination of some type, e.g., [2, 5, 9, 13, 15, 16, 18, 23, 24, 28].

Quorum constructions have been studied for over 25 years, and quorum systems with many different properties and guarantees have been developed (e.g., [5, 24]). For example, note that quorum systems naturally provide load dispersion: since the quorum sizes may be smaller than the size of the universe U, quorum systems allow us to reduce the *load*, i.e., the probability with which the busiest server is accessed in any given quorum invocation. Through careful design of the quorums in $\mathcal{Q}$, and of the *access strategy*—i.e., the probability distribution p over quorums in $\mathcal{Q}$ such that $Q \in \mathcal{Q}$ is chosen in any client with probability $p(Q)$—a load of $O(1/\sqrt{|U|})$ can be achieved [22].

Despite this research, it is only very recently that *networking issues* have also been considered in the study of quorum systems: while we may understand the properties of a good quorum system over some abstract universe U, we do not yet understand how to map the elements of U to the physical nodes V in our network so as to give us *good network performance*. For instance, some very recent work of the authors and others has proposed algorithms for finding placements that minimize the *routing delays* that clients incur when accessing quorums [8, 10, 11, 14, 29]. (We discuss this work in more detail in Section 2.) In this paper, we initiate the study of an orthogonal issue: that of the *network congestion* caused by the deployment of quorum systems in networks. Roughly speaking, our goal is to develop algorithms to place quorums in a network so as to minimize the congestion resulting from client accesses to quorums, while additionally ensuring that each physical node is not overloaded by quorum requests from clients.

The Model. Formally, we model the network as an undirected graph $G = (V, E)$ of size $n = |V|$. Each physical node $v \in V$ is given a *node capacity* $\mathsf{node\_cap}(v) \in \mathbb{R}_{\geq 0}$, which is an upper bound on the amount of quorum load it wishes to handle. Furthermore, each edge $e \in E$ also has an *edge capacity* $\mathsf{edge\_cap}(e) \in \mathbb{R}_{\geq 0}$, which represents its bandwidth, i.e., the amount of traffic it can carry. For a given quorum system $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ on a universe U, an access strategy p, and a network $G = (V, E)$, a map $f : U \to V$ placing the elements on the physical nodes is called a *quorum placement*; we will use $f(Q) \subseteq V$ to denote the set of nodes $\cup_{u \in Q}\{f(u)\}$.

This work was partially supported by NSF award number CCF-0424422, and by KISA and MIC of Korea.

*Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

†Electrical & Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA

‡Akamai Technologies, Cambridge, MA, USA

PODC'06, July 22-26, 2006, Denver, Colorado, USA.

We assume that the set of clients accessing the quorums in $\mathcal{Q}$ is just the node set V. To make the explanations simpler, we will assume that each of the clients $v \in V$ generates requests at some *rate* r_v with $\sum_{v \in V} r_v = 1$. It will be convenient to think of the rate r_v as the probability that client v makes a request.

The Measures of Goodness. We are concerned with two measures in this paper: the *network congestion* caused by routing the requests from the clients to the nodes hosting the quorum elements, and the *load* generated on these nodes due to processing these requests. Let us formally define these.

Load. Given a quorum system $\mathcal{Q}$ over U and an access strategy p, we can define the load of an element $u \in U$ to be $\mathsf{load}(u) = \sum_{Q \in \mathcal{Q}: u \in Q} p(Q)$. (In other words, this is the probability that the element u is accessed when using the access strategy p.) Given a placement $f : U \to V$, we can extend this notion of load to any node v by defining $\mathsf{load}_f(v) = \sum_{u \in U: f(u)=v} \mathsf{load}(u)$. Ideally, we would like $\mathsf{load}_f(v) \leq \mathsf{node\_cap}(v)$ for every node $v \in V$.

Congestion. To access a quorum Q, a client v needs to access each member $f(u) \in f(Q)$ individually, and has to send a request to $f(u)$. This access from v to $f(u)$ naturally increases the traffic on the edges of some path from v to $f(u)$, and we would like to minimize this. We consider two models for selecting the path.

1. In the *arbitrary routing* model, the path used for routing in the network may be chosen arbitrarily, and hence it is convenient to model traffic between any two nodes $v, v' \in V$ as a *flow* $g_{v,v'} : E \to \mathbb{R}_{\geq 0}$. (Note that each access will use a single path, but we may vary the paths used between a pair of nodes so that the average traffic on any edge is the same as in the flow.)

2. In the *fixed routing paths* model, the paths $\{P_{v,v'}\}$ are specified as part of the input, and while we can define the flow $g_{v,v'}$ as before, all the flow must travel along the path $P_{v,v'}$. This is motivated by networks like the Internet where senders and receivers cannot control or select the paths along which their traffic travels.

Thus in either model we may define the expected traffic on any edge $e \in E$ due to requests from a fixed node v to be

$$\sum_{Q \in \mathcal{Q}} p(Q) \sum_{u \in Q} g_{v,f(u)}(e)$$

Finally, since the node v is responsible for an r_v fraction of the requests, we can define the *average traffic* on the edge e to be

$$\mathsf{traffic}_f(e) = \sum_{v \in V} r_v \sum_{Q \in \mathcal{Q}} p(Q) \sum_{u \in Q} g_{v,f(u)}(e).$$

(This can be read thus: we choose the client v with probability r_v, choose a quorum Q with probability $p(Q)$, and incur a traffic of $g_{v,f(u)}(e)$ for every $u \in Q$.)

Since we are always considering averages, we will usually just refer to this as the *traffic* on the edge e. Finally, given an edge capacity $\mathsf{edge\_cap}(e)$, the *congestion* due to the placement f is

$$\mathsf{cong}_f(e) = \mathsf{traffic}_f(e)/\mathsf{edge\_cap}(e) \tag{1.1}$$

Ideally, this quantity should be as low as possible. Indeed, the objective function we seek to minimize is the *congestion* of the placement f, which is defined to be the congestion of the most congested edge, namely $\mathsf{cong}_f = \max_{e \in E} \mathsf{cong}_f(e)$.

Before we proceed, note that we used a flow $g_{v,v'}$ in the above discussion to model the flow of messages between v and v'. Given a placement f in the arbitrary routing model, finding a set of flows $\{g_{v,v'}\}$ that minimize the congestion (1.1) subject to the placement f is just a flow problem, and can be optimized in polynomial time. (Of course, the flow in the fixed-paths model is just $P_{v,v'}$.) Hence, the rest of the paper will focus on finding the placements f: whenever we refer to a "placement f with congestion c" in the arbitrary routing model, it should be taken to read "placement f *for which there exist flows* $\{g_{v,v'}\}$ that give congestion c".

We are finally in a position to formally define the problem addressed in this paper:

Problem 1.1 (Quorum Placement Problem for Congestion (QPPC)) *Given a quorum system $\mathcal{Q}$ over the universe U, an access strategy p, and an undirected network $G = (V, E)$ with capacities* $\mathsf{edge\_cap} : E \to \mathbb{R}_{\geq 0}$ *and* $\mathsf{node\_cap} : V \to \mathbb{R}_{\geq 0}$ *on edges and nodes, respectively, and client access rates $\{r_v\}$, find a placement $f : U \to V$ that (a) minimizes the congestion* cong_f *subject to (b)* $\mathsf{load}_f(v) \leq \mathsf{node\_cap}(v)$ *for all nodes $v \in V$.*

Since we can scale the capacities on the edges, we will assume (for simplicity of exposition) that the edge congestion cong_{f^*} of the optimum placement f^* is precisely 1.

Before we present our results, let us note that all our analyses apply in the *unicast* model, where an individual request is sent to *each* element of the quorum being accessed. An alternate model (which we do not consider here) would permit *multicast* messages from the source to the quorum members. Using these multicasts clearly decreases the congestion incurred: for instance, if two quorum elements are mapped to the same physical node v, these co-located elements could be reached using a single message. (Moreover, the node v could intelligently process the information reaching these co-located elements just once, thereby incurring less load.) We leave the study of these models and optimizations for future work.

1.1 Our Contributions

As stated, the QPPC problem turns out to be highly intractable:

Theorem 1.2 *Even determining whether a feasible solution for the QPPC exists (in either model) is NP hard if we do not allow any node capacities to be violated.*

We go on to show a number of approximation results for the QPPC problem if we are allowed to violate the node capacities by at most a factor of two. We use the following notation: if f^* is the optimal solution to a QPPC instance (that satisfies the load constraints), then *an (α, β)-approximation* is a placement f such that $\mathsf{cong}_f \leq \alpha \cdot \mathsf{cong}_{f^*}$ and $\mathsf{load}_f(v) \leq \beta \cdot \mathsf{node\_cap}(v)$ for all nodes v.

Theorem 1.3 (Approximations for Arbitrary Routing) *For any instance of QPPC in the arbitrary routing model, we can find an $(O(\log^2 n \log\log n), 2)$ - approximation in polynomial time. If the graph G is a tree, we obtain a $(5, 2)$-approximation.*

Theorem 1.4 (Approximations for Fixed Paths) *Given an instance of QPPC in the fixed routing paths model, we can find an $(O(\frac{\eta \cdot \log n}{\log\log n}), 2)$-approximation in polynomial time, where η is the size of $\{\lfloor \log(\mathsf{load}(u)) \rfloor \mid u \in U\}$. For example, if there exists an N such that $\mathsf{load}(u) \in [1/N, 1]$ for all $u \in U$, then the algorithm above yields an $(O(\frac{\log N \log n}{\log\log n}), 2)$-approximation.*

Our Techniques. The basis of the algorithm for the arbitrary routing model QPPC lies in a reduction of the problem to the case in which the graph is a tree *and there is only one client in the system* (i.e., there is a node v with $r_v = 1$). This reduction uses some of the

properties of quorum systems, combined with the general graph decomposition result of Räcke [25]; however, it costs us a factor of $O(\log^2 n \log\log n)$ in the congestion. For the single-client tree case, we give an approximation algorithm by first writing an integer programming formulation, and then rounding its linear-programming relaxation: the rounding uses an algorithm for unsplittable flows from Dinitz et al. [6], and is possibly of independent interest.

Our results for the fixed paths model use a different set of tools: we first develop an algorithm for instances where all the elements of U have identical ("uniform") loads. For this we use a different linear programming relaxation, and then round it using a different rounding technique that does not allow node capacities to be violated [27]. We then use this algorithm as a subroutine to solve the non-uniform case by carefully placing down sets of elements in decreasing order of their loads.

We also show hardness results for QPPC in the fixed paths model, even when the loads are all uniform. Theorem 6.1 states that it is NP-hard to approximate the congestion to any constant factor (even if we completely ignore the load constraints); in fact, we can obtain stronger inapproximability results under stronger complexity-theoretic assumptions.

Finally, we provide preliminary results regarding the utility of migration of universe elements between physical nodes of the network as a technique to further reduce congestion. The details of this analysis are included in Appendix A.

2. RELATED WORK

As mentioned above, quorum systems are well-studied; see, e.g., [1, 2, 5, 9, 18, 20, 21, 28] and the references therein. There is much less work on quorum placement problems that seek to minimize objectives that capture network performance, which we summarize here.

To the best of our knowledge, previous work on quorum placement in networks has only considered minimizing various notions of *delay* that a client incurs by accessing a quorum. Specifically, let $d(v,v')$ denote the distance from node v to v' in a graph $G=(V,E)$, and let $\delta(v,Q) = \max_{v'\in Q} d(v,v')$ and $\gamma(v,Q) = \sum_{v'\in Q} d(v,v')$ be the delays incurred by a node v when accessing a quorum $Q \subseteq V$ in parallel and sequentially, respectively (hence the max and the sum in the two definitions). Previous work has included algorithms to design quorum systems to minimize objectives like **(a)** average delay $\mathsf{Avg}_{v\in V}\left[\min_{Q\in\mathcal{Q}} \delta(v,Q)\right]$ for particular classes of graphs [8] or for arbitrary graphs [14]; **(b)** or max-delay to the closest quorum $\max_{v\in V} \min_{Q\in\mathcal{Q}} \delta(v,Q)$ [29].

Furthermore, there has been work on finding placements of a given quorum system $\mathcal{Q}$ on an arbitrary graph $G=(V,E)$: there is work on **(i)** designing bijections $f: U \to V$ and $q: V \to \mathcal{Q}$ that minimize $\mathsf{Avg}_{v\in V}\gamma(v, f(q(v)))$ [10], or **(ii)** designing a placement $f: U \to V$ to (approximately) respect load constraints on nodes and minimize $\mathsf{Avg}_{v\in V}[\mathbf{E}[\gamma(v, f(Q))]]$ or $\mathsf{Avg}_{v\in V}[\mathbf{E}[\delta(v, f(Q))]]$, where expectations are taken with respect to the selection of Q according to p [11]. Among these, only the work in **(ii)** considers the load of the quorum system; however, it does not consider the congestion incurred by these placements, and indeed may give us fairly poor placements with respect to network congestion.

Minimizing network congestion for both specific and general networks is a problem that has received considerable attention in the past; given the impossibility of summarizing this work, we mention just some of the most important results here. Early work in this area included the seminal results of Valiant [30] and Valiant and Brebner [31] who gave randomized routing algorithms in hypercubes and meshes to get small congestion. Leighton, Makedon and Tollis then gave deterministic algorithms for meshes [17]. Linear programming relaxations and randomized rounding was first used by Raghavan and Thompson [26] to find unsplittable paths with low congestion. Single-source versions of unsplittable flow were studied by Dinitz, Garg and Goemans [6], who gave constant-factor approximation algorithms for various versions of the problem.

In a model similar to ours, Maggs et al. [19] consider a data management problem for special networks (trees, meshes, and clustered networks). In their work, clients issue read and write requests for objects, where a read request is serviced by any node holding a copy of the object, but a write request must update all copies of the object. Just as in this paper, the goal of their work is to place the objects optimally on the nodes of a network to minimize congestion. However, while their paper considered the questions behind replicating objects and the static and dynamic issues therein (i.e., how many copies of an object to maintain at any time? where to place them?), here we take a fixed quorum system and client request rates as input and try to find congestion-optimal placements that respect node capacities.

The results of Maggs et al. [19] are extended by Westermann [32] to a model in which objects are allowed to *migrate* between nodes of the network: while migrating an object increases congestion, moving the object closer to a source may eventually decrease traffic in the network. He gives a 3-competitive algorithm for congestion for trees, and extends these results to other classes of networks.

Räcke [25] further generalizes these results by giving a general method to solve a congestion problem in arbitrary graphs. His method is based on the construction of a *congestion-tree* T_G that "simulates" the original graph with a polylog $|V|$ factor loss in congestion; more details on this general method are given in Section 3.1.

3. PRELIMINARIES

In this section, we introduce some concepts and results that will be used in developing algorithms for the QPPC problem in the arbitrary routing model. The "congestion preserving" trees of Räcke mentioned in Section 2 are directly related to the problem at hand, so we discuss them in more detail in the next section. The results on unsplittable flows in Section 3.2 will be used in rounding a linear-programming relaxation of one of the problems we consider here.

3.1 Congestion Trees

Given an instance of a congestion-minimization problem on a general graph G, one may try to reduce the problem to one on a simpler graph—for instance, a tree T—where it is algorithmically easier to find a good solution. Of course, we would like that the tree T "approximates" the graph G well; the following definition formally states the notion of approximation we will use. Recall that a *multicommodity flow* on a graph $G=(V,E)$ is a set $g=\{g_i : E \to \mathbb{R}_{\geq 0}\}_i$ of flows where g_i carries d_i units from s_i to t_i ($s_i, t_i \in V$); the vector $\{d_i\}_i$ is the *value* of the flow.

Definition 3.1 *A tree $T=(V_T,E_T)$ with edge capacities given by $\mathsf{edge\_cap}_T : E_T \to \mathbb{R}_{\geq 0}$ is a β-approximate congestion tree for a graph $G=(V,E)$ with edge capacities $\mathsf{edge\_cap}_G : E \to \mathbb{R}_{\geq 0}$ if:*

1. *The vertices of G are the leaves of T.*
2. *For any multicommodity flow g on pairs $\{(s_i,t_i)\}_i$ that is feasible on G (i.e., $\sum_i g_i(e) \leq \mathsf{edge\_cap}_G(e)$ for each $e \in E$) there is a feasible multicommodity flow of the same value on leaves $\{(s_i,t_i)\}_i$ in T.*

3. *For any feasible multicommodity flow g_T on pairs of leaves $\{(s_i,t_i)\}_i$ in T, there exists a multicommodity flow g on $\{(s_i,t_i)\}_i$ in G such that g has the same value as g_T and $\sum_i g_i(e) \leq \beta \times \mathsf{edge\_cap}_G(e)$ for each $e \in E$.*

In a surprising recent result, Räcke [25] showed that one can find congestion trees for general networks with $\beta = \text{poly}\log n$. His initial result was existential, but subsequent results of [3, 12] made the construction algorithmic, and also improved the value of β to give us the following theorem.

Theorem 3.2 *Given any undirected graph $G = (V,E)$, there exists an $O(\log^2 n \log\log n)$ - approximate congestion tree T_G; furthermore, this congestion tree can be found in time polynomial in n and the maximum capacity of any edge (assuming edge capacities are bounded to within a fixed polynomial factor of each other).*

Working in the arbitrary routing model, we will use this result to reduce an instance of the Quorum Placement Problem for Congestion on general graphs to an instance on trees, and then we will give algorithms to solve the Quorum Placement Problem for Congestion on trees.

3.2 Single Source Unsplittable Flow

In general, a flow from s to t could be *fractional*, i.e., the commodity travels on multiple paths from s to t. In contrast, an *unsplittable flow* is one that is constrained to travel only on a single path. The **Single-Source Unsplittable Flow Problem** (SSUFP), then, is specifically the following: given a directed graph $G = (V,E)$ with edge capacities $\mathsf{edge\_cap} : E \to \mathbb{R}_{\geq 0}$, a source node $s \in V$ and k *terminals* $t_i \in V$, with each t_i in $1 \leq i \leq k$ having a *demand* d_i, find a multicommodity flow from the source to the terminals such that the flow $g_i : E \to \mathbb{R}$ from s to t_i (of d_i units) is unsplittable (i.e., travels on a *single path*), and the total flow on any edge e is $\sum_i g_i(e) \leq \mathsf{edge\_cap}(e)$. Note that a solution to this problem is given by a set of paths $\{P_i\}_{i=1}^k$, where P_i is a path from s to t_i.

This problem was studied by Dinitz, Garg and Goemans [6], who proved the following: given any feasible instance of the single-source unsplittable flow problem, there is a polynomial time algorithm to obtain a set of paths P_i (one for each terminal t_i), such that the total traffic $\sum_{i:e\in P_i} d_i$ on any edge e is at most $\mathsf{edge\_cap}(e) + \max_i\{d_i\}$. In fact, they prove a slightly stronger result, which we now state in a form most convenient to us:

Theorem 3.3 *Given a fractional multicommodity flow that satisfies terminal demands and the edge capacities (where the flow of d_i units from s to t_i is denoted by g_i), the algorithm of Dinitz et al. [6] converts it into an unsplittable flow P_i where the total traffic over an edge e is*

$$\sum_{i:e\in P_i} d_i \leq \mathsf{edge\_cap}(e) + \max\{\, d_i \mid g_i(e) > 0\}.$$

Note that the maximum on the right hand side is only over the commodities using the edge e in the input fractional flow.

In Section 4.2, we will use this theorem to round a fractional solution of a linear programming relaxation for the QPPC problem in the arbitrary routing model.

4. THE ARBITRARY ROUTING MODEL: THE SINGLE CLIENT CASE

In this section, we present our first results for the Quorum Placement Problem for Congestion (QPPC) in the arbitrary routing model: we consider the special case when there is *only one client* in the system generating the requests. For this case, we show that it is NP-hard to approximate the congestion within *any* factor if we enforce the node capacities $\mathsf{node\_cap}(v)$. We then show that if we are allowed to violate the node capacities by a "small" amount, we can achieve a "small" congestion as well.

4.1 A Hardness Result

Let us begin by proving the following simple theorem that shows that this problem is NP-hard to approximate within *any* factor. This hardness result motivates a line of inquiry we will pursue, where we allow the node capacities to be violated by a small amount, and then try to minimize the edge congestion incurred.

Theorem 4.1 *Finding* any *feasible solution to the Single Client case of QPPC (in either model) is NP-hard if no node capacities $\mathsf{node\_cap}(v)$ are violated.*

Proof. The reduction is from the PARTITION problem, an instance of which contains a set of numbers $\{a_1, a_2, \ldots, a_l\}$ with $\sum_i a_i = 2M$, and the goal is to find a subset of the a_i's that sum to exactly M.

We now construct a quorum system $\mathcal{Q}$ on $l+1$ nodes $U = \{u_0, u_1, \ldots, u_l\}$ with l quorums $Q_i = \{u_0, u_i\}$, and the access strategy $p(Q_i) = a_i/2M$. Note that $\mathsf{load}(u_0) = 1$ and $\mathsf{load}(u_i) = a_i/2M$ otherwise. Finally, let the graph $G = (V,E)$ consist of the complete graph with 3 nodes $\{v_0, v_1, v_2\}$, with node capacities $\mathsf{node\_cap}(v_0) = 1$, and $\mathsf{node\_cap}(v_1) = \mathsf{node\_cap}(v_2) = 0.5$. (The edge capacities are not relevant in this reduction.) Finally, let all the requests originate from a single client located at v_0.

Note that any feasible placement f that respects the node capacities must place the element u_0 at the root v_0, and hence the set of elements placed at node v_1 must have $\sum_{a_i} = M$. Thus it is NP-hard to find *any feasible placement* for this instance, let alone a placement that approximates the edge congestion. ■

4.2 The Algorithm for the Single Client Case

Our result for the special case of a single client works for the more general case of directed graphs. In fact, we also permit the presence of the following additional constraints:

- for each edge e, we can give a set of *forbidden elements* denoted by $F_e \subseteq U$ such that traffic to any element $u \in F_e$ is not allowed to traverse edge e; and
- for each node v, a set of forbidden elements $F_v \subseteq U$ that cannot be placed at the node v. (I.e., forbidden placements f are those with $f(u) = v$ for some $u \in F_v$.)

Let us denote by $\mathsf{loadmax}_v$ the maximum load of any element that can be placed on v, i.e., $\mathsf{loadmax}_v = \max_{u\notin F_v} \mathsf{load}(u)$. Similarly, let $\mathsf{loadmax}_e = \max_{u\notin F_e} \mathsf{load}(u)$. We will use these quantities to parameterize the performance of the following theorem.

Theorem 4.2 *Given a directed instance of the Quorum Placement Problem for Congestion in the arbitrary routing model, with a single client v_0 generating requests, let f^* be the optimal placement that respects node capacities $\mathsf{node\_cap}$ and achieves a congestion of cong^* on the edges. We can find, in polynomial time, a placement f for which:*

- *the load $\mathsf{load}_f(v)$ on any node v is at most $\mathsf{node\_cap}(v) + \mathsf{loadmax}_v$, and*
- *the traffic on any edge e is at most $(\mathsf{cong}^* \times \mathsf{edge\_cap}(e)) + \mathsf{loadmax}_e$.*

Proof. To prove this theorem, we formulate the Quorum Placement Problem for Congestion as an integer linear program (ILP), consider its linear programming (LP) relaxation, and round a (possibly fractional) solution to this LP relaxation to an integer solution to (ILP) while losing at most $O(\mathsf{loadmax}(e))$ during this rounding.

Consider the following integer linear programming formulation (ILP):

$$\lambda^* = \text{minimize } \lambda \tag{4.2}$$

$$\sum_i x_{iu} = 1, \qquad \forall\, u \in U \tag{4.3}$$

$$\sum_u \mathsf{load}(u)\, x_{iu} \le \mathsf{node\_cap}(v_i),\ \forall\, v_i \in V \tag{4.4}$$

$$x_{iu} = 0, \qquad \forall\, u \in F_{v_i} \tag{4.5}$$

$$\sum_{P \in \mathscr{P}_i} g_u(P) = \mathsf{load}(u) x_{iu}, \forall\, u \in U, \forall\, v_i \in V \tag{4.6}$$

$$\sum_{\substack{P \in \mathscr{P}_i \\ e \in P}} g_u(P) = 0, \qquad \forall\, u \in F_e, \forall\, e \in E \tag{4.7}$$

$$\sum_{u \in U} \sum_{v_i \in V} \sum_{\substack{P \in \mathscr{P}_i \\ e \in P}} g_u(P) \le \lambda \times \mathsf{edge\_cap}(e),\ \forall\, e \in E \tag{4.8}$$

$$x_{iu} \in \{0,1\}, \forall\, v_i \in V, \forall\, u \in U. \tag{4.9}$$

Here x_{iu} is the indicator variable for the element u being placed on node v_i, $\mathscr{P}_i$ is the set of paths from the client v_0 to the node v_i,[1] $g_u(P)$ is the amount of traffic destined for element u that uses some path P, and λ is the overall congestion of the resulting solution. Since each of x_{iu} is either 0 or 1, the $g_u(P)$'s tell us how to send the traffic from the client v_0 to the node v_i with $x_{iu} = 1$. (Since we do not require that the $g_u(P)$'s be integral, technically the above program is a mixed-integer program.)

Note that given a solution f to the single-client QPPC problem with congestion cong_f, we may set $x_{iu} = 1 \iff f(u) = v_i$ and use the flows prescribed by the given solution to obtain $\lambda = \mathsf{cong}_f$, and hence this is indeed a formulation of the original problem.

Since we cannot solve this ILP optimally in polynomial time, we relax the integrality constraints: instead of (4.9), we throw in the constraint $0 \le x_{iu} \le 1$ and solve the resulting linear program; now we have to round the resulting fractional solution (λ, x, g) to one where $x_{iu} \in \{0,1\}$ for all i and u. For simplicity of exposition, we scale the edge capacities by a factor of λ, so that with the new edge capacities $\lambda^* = 1$.

Preprocessing. We will use the rounding scheme used for the Single-Source Unsplittable Flow Problem to round our fractional solution, and hence we first construct an instance of SSUFP. Consider the graph $G = (V,E)$, and let us add a new "sink" vertex t to it, with directed arcs (v_i,t) from each $v_i \in V$ to this new vertex t, with each arc (v_i,t) having a capacity of $\mathsf{edge\_cap}((v_i,t)) = \mathsf{node\_cap}(v_i)$. Now we create $|U|$ new "terminals" $\{t_u \mid u \in U\}$, all of which are located at the "sink" node t. Define the client v_0 to be the "source".

Finally, note that total amount of flow ending at v_i is equal to $\sum_{u \in U} \sum_{P \in P_i} g_u(P_i) = \sum_u \mathsf{load}(u) \times x_{iu}$ using equality (4.6), which by (4.4) is at most $\mathsf{node\_cap}(v_i)$. Thus we can take all the flow that previously ended at the node v_i, and send it on the arc (v_i,t) to the sink t without violating capacities. Doing this for all vertices v_i, we get a flow that for each $u \in U$, sends $\mathsf{load}(u)$ units of flow from the source v_0 to the terminal t_u.

Using SSUFP to Round the LP Solution. Finally, we apply Theorem 3.3 to the flow created in the above construction: the answer it returns is a set of paths $\{P_u\}_{u \in U}$, one for each $u \in U$, such that the flow on e is

$$\sum_{u : e \in P_u} \mathsf{load}(u) \le \mathsf{edge\_cap}(e) + \max_{u : g_u(e) > 0} \{\mathsf{load}(u)\}. \tag{4.10}$$

Finally, if the path P_u uses the edge (v_i,t) to reach $t_u = t$, define $f(u)$ to be v_i.

Proving the Claimed Guarantees. Let us first consider the load $\mathsf{load}_f(v_i)$, which is equal to the traffic on the arc (v_i,t). Recall that $\mathsf{edge\_cap}((v_i,t)) = \mathsf{node\_cap}(v_i)$. Also, if $g_u((v_i,t)) = \sum_{P \in \mathscr{P}_i} g_u(P)$ is non-zero, then $u \notin F_{v_i}$ by the constraint (4.7), and thus $\mathsf{loadmax}_{v_i} \ge \mathsf{load}_u$. Plugging these facts into (4.10) implies that $\mathsf{load}_f(v_i) \le \mathsf{node\_cap}(v_i) + \mathsf{loadmax}_{v_i}$, as claimed.

Now for the traffic on an edge $e \in E$: this was originally at most $\mathsf{edge\_cap}(e)$, and now can increase by at most $\mathsf{loadmax}_e$ (due to the constraint (4.5)), thus proving the theorem. ∎

5. THE GENERAL CASE OF QPPC IN THE ARBITRARY ROUTING MODEL

To obtain the result for an arbitrary number of clients claimed in Section 1, we use the following strategy:

(A) Reduce the problem to trees. We first translate the QPPC problem on a general graph G to the β-approximate congestion tree T_G with $\beta = O(\log^2 n \log\log n)$, as guaranteed by Theorem 3.2.

It follows from the definition of a congestion tree, and the fact that the leaves of T_G correspond to nodes of the network G, that any placement $f : U \to \mathsf{leaves}(T_G)$ which is an α-approximation for the optimal congestion in T_G corresponds to a placement $f : U \to V(G)$ which approximates the optimal congestion in G to within $\alpha \times \beta$. (The details of this translation are given in Section 5.1.)

(B) Reduce the problem to the single-source case. In Section 5.2, we show that there is a placement f_0 that maps all elements in U to a single node v_0 in the tree T_G and minimizes the congestion of the tree edges. However, this placement has very high load, and since our goal is to achieve low loads in addition to a low network congestion, this solution is clearly not acceptable. However, this will be a convenient structural result for the rest of the argument.

(C) Solve the single-source problem. Finally, in Section 5.3, we imagine the above single-node solution v_0 as a *single client generating all the requests*, and use the algorithm of Section 4 to find a good placement $f : U \to \mathsf{leaves}(T_G)$ for this single-client case. We show that f is also a "good" placement for the original set of clients in T_G, and achieves a congestion of $\alpha \le 5$ times the optimum.

5.1 Translating the QPPC Instance to a Congestion Tree

Consider a graph $G = (V,E)$ and a β-approximate congestion tree $T_G = (V_T, E_T)$. Recall that V is equal to the set of leaves of T_G, i.e. $V = \mathsf{leaves}(T_G)$. Let $f^*_G : U \to V$ be the placement in the graph G with the least edge congestion cong^*_G. Let $f^*_{T_G} : U \to \mathsf{leaves}(T_G)$ be the placement that has the least congestion over the edges of the tree T_G, and let $\mathsf{cong}^*_{T_G}$ be the value of this congestion. By the definition of congestion trees, it follows that $\mathsf{cong}^*_{T_G} \le \mathsf{cong}^*_G$. Since we assumed that the optimal congestion on G is exactly 1, we get the following fact.

Lemma 5.1 *The optimal congestion on T_G is at most* 1.

Moreover, if $f : U \to \mathsf{leaves}(T_G)$ is a placement with congestion at most $\alpha \times \mathsf{cong}^*_{T_G}$ over the edges of T_G, then f has congestion of at most $(\alpha \times \beta) \times \mathsf{cong}^*_{T_G} \le (\alpha \times \beta) \times \mathsf{cong}^*_G$ over the edges of G. This implies the following result:

[1] Note that $|\mathscr{P}_i|$ could be exponential in n; one can write an equivalent formulation of this ILP with a number of variables and constraints polynomial in n. However, the formulation we present here will be easier to argue about.

Theorem 5.2 *Any placement $f : U \to \mathsf{leaves}(T_G)$ with edge congestion $\alpha \times \mathsf{cong}^*_{T_G}$ over the edges of T_G has a congestion of $\alpha\beta \times \mathsf{cong}^*_G$ over the edges of G. In other words, a placement on the leaves of T_G that is an α-approximation for congestion on T_G is an $\alpha\beta$-approximation for congestion on G.*

Note that the above theorem only works for placements that map elements to the leaves of T_G, and as such cannot be used directly with the results of the next section.

5.2 Single Node Solutions are Good on Trees

For any node $v \in V_T$, let $f_v : U \to V_T$ be the trivial placement with $f_v(u) = v$ for all $u \in U$; i.e., all the elements of U are placed on the single node v. We will show that on a tree, an optimal placement of $(\mathcal{Q}, p)$, provided we ignore node capacity constraints, is on a single node of the tree.

Lemma 5.3 *Given a tree $T = (V_T, E_T)$ and a placement $f : U \to V_T$, one can find (in polynomial time) a node $v_0 \in V$ such that the placement f_{v_0} has congestion no greater than that of f.*

Proof. Let $f^{-1}(v)$ denote $\{u \mid f(u) = v\}$. For a node $v \in T$, recall that r_v was the fraction of all the requests in the system that are generated by the client v, and also that

$$\begin{aligned}\mathsf{load}_f(v) &= \sum_{u \in U : f(u) = v} \mathsf{load}(u) \\ &= \sum_{u \in U : f(u) = v} \; \sum_{Q \in \mathcal{Q} : u \in Q} p(Q) \\ &= \sum_{Q \in \mathcal{Q}} p(Q) \times |f^{-1}(v) \cap Q|\end{aligned}$$

is the expected number of messages that reach the node v (where the expectation is taken over the choice of Q under the access strategy p). It is a simple exercise to prove that there exists a node v_0 in T such that each subtree T' of $T - \{v_0\}$ has at most half the demands; i.e., $\sum_{v \in T'} r_v \le \frac{1}{2} \le \sum_{v \notin T'} r_v$.

Consider an edge e, and let T_L and T_R be the subtrees formed by deleting e. Let $r(T_L) = \sum_{v \in T_L} r_v$ be the total fraction of demands generated by clients in T_L. The expected number of messages seen by nodes in T_L is $\mathsf{load}_f(T_L) = \sum_{v \in T_L} \mathsf{load}_f(v)$. Let $r(T_R)$ and $\mathsf{load}_f(T_R)$ be defined similarly for the subtree T_R. Then the total congestion of the edge e under the placement f is

$$\frac{r(T_L) \times \mathsf{load}_f(T_R) + r(T_R) \times \mathsf{load}_f(T_L)}{\mathsf{edge\_cap}(e)} \tag{5.11}$$

Without loss of generality, let $r(T_L) \le r(T_R)$, and hence the node v_0 must lie in T_R. Thus all the messages traversing the edge e under the placement f_{v_0} go from T_L to T_R, with e having a congestion of $r(T_L) \times [\mathsf{load}_f(T_R) + \mathsf{load}_f(T_L)]/\mathsf{edge\_cap}(e)$; the $r(T_L)\mathsf{load}_f(T_R)$ term corresponds to messages generated by nodes in T_L which are sent across e under both placements, while the $r(T_L)\mathsf{load}_f(T_L)$ term corresponds to messages generated by nodes in T_L that are sent to nodes in T_L under placement f but are sent across e under placement f_{v_0}. Since $r(T_L) \le r(T_R)$, this quantity is at most (5.11), the congestion under the placement f. Finally, we note that the node v_0 can be found in linear time simply by trying all the nodes of T, which completes the proof of the lemma. ∎

While this lemma tells us how to find the best quorum placement on trees, it is unsatisfying for at least two reasons. First, the node v_0 in the above theorem suffers all the load in the system under the placement f_{v_0}. Second, this node v_0 may be an internal node of T_G, and hence we cannot directly obtain a solution for the graph G by applying Lemma 5.3 on the congestion tree T_G, and then using Theorem 5.2 to translate the solution back to G. In the next section we provide a solution to these problems.

5.3 The Algorithm for General QPPC

Consider a congestion tree T_G, let f^* be the best placement of U on the leaves of T_G that respects the node capacities (i.e., $\mathsf{load}_{f^*}(v) \le \mathsf{node\_cap}(v)$ for all v); let cong_{f^*} be the congestion in T_G under f^*. Let the best (single-node) placement given by Lemma 5.3 for the tree T_G be f_{v_0}, which places the entire quorum on v_0. Let the congestion incurred under this placement be $\mathsf{cong}_{f_{v_0}}$; Lemma 5.3 shows that $\mathsf{cong}_{f_{v_0}} \le \mathsf{cong}_{f^*}$.

Let us show that if v_0 were generating all the requests (instead of the node v generating requests with probability r_v), the placement f^* would still be a fairly good placement.

Lemma 5.4 *The congestion incurred by the placement f^* if all the requests in the system originate at v_0 (instead of at the individual clients) is $\mathsf{cong}_{f^*,v_0} \le 2\mathsf{cong}_{f^*}$.*

Proof. Indeed, the congestion is no worse than if we use the following routing strategy for messages: let v_0 choose Q according to the access strategy p, and a leaf v with probability r_v, and send the messages to the various nodes in $f(Q)$ by first sending them to v, which forwards them on to $f(u)$. The first part of this indirect route incurs the same congestion as the case when v were generating all the $|Q|$ messages and using the placement f_{v_0}, which is just $\mathsf{cong}_{f_{v_0}} \le \mathsf{cong}_{f^*}$ (by Lemma 5.3). The second part of the route incurs a further congestion of cong_{f^*}, which proves the result. ∎

Recall that $\mathsf{cong}_{f^*} \le 1$ due to Lemma 5.1. We now prove the main result for the QPPC problem on trees:

Theorem 5.5 *There is a placement f on the leaves of the tree T_G that incurs a congestion of at most $3\mathsf{cong}_{f^*} + 2 \le 5$, and which places a load of at most $2\mathsf{node\_cap}(v)$ on each leaf v.*

Proof. Let us imagine the node v_0 of Lemma 5.4 to be the sole client, and use the algorithm of Section 4 to find a placement f on the leaves of T_G with "low" load and congestion. Each leaf node v of T_G corresponds to a node of G and hence has a node capacity already defined; for each internal node $v \in T_G$, define the $\mathsf{node\_cap}(v) = 0$, thus ensuring that no elements are mapped to internal nodes.

Recall that one could specify *forbidden sets* for nodes and edges in the algorithm of Section 4.2: let the forbidden set F_v for node v be the set of elements u with $\mathsf{load}(u) > \mathsf{node\_cap}(v)$. Also, the forbidden set F_e for a tree edge e is defined to be the set of all elements u such that $\mathsf{load}(u) > 2\mathsf{edge\_cap}(e)$. Note that these settings ensure that $\mathsf{loadmax}_e \le 2\mathsf{edge\_cap}(e)$ and $\mathsf{loadmax}_v \le \mathsf{node\_cap}(v)$.

Note that the placement f^* on the leaves of T is a possible solution to this instance of the single-client QPPC, having a congestion of at most 2 (due to Lemma 5.4 and Lemma 5.1) and load of at most $\mathsf{node\_cap}(v)$, for each $v \in V$. Hence, Theorem 4.2 guarantees us that (a) each node has a load of at most $\mathsf{node\_cap}(v) + \mathsf{loadmax}_v = 2\mathsf{node\_cap}(v)$, and that (b) each edge sees a traffic of at most $(\mathsf{cong}_{f^*,v_0} \times \mathsf{edge\_cap}(e)) + 2\mathsf{edge\_cap}(e)$, and hence the congestion is at most $\mathsf{cong}_{f^*,v_0} + 2 \le 2\mathsf{cong}_{f^*} + 2$.

Now, since the requests are generated by the various nodes of the network and not by the single node v_0, one has to add in the extra congestion incurred by sending all the requests to v_0. By Lemma 5.3, this extra congestion is at most $\mathsf{cong}_{f_{v_0}} \le \mathsf{cong}_{f^*}$.

Finally, putting the pieces together, the idea of conceptually "delegating" all the requests to v_0 and using the placement f that (approximately) optimizes the congestion for the "source" v_0 gives us the claimed congestion of $3\mathsf{cong}_{f^*} + 2 \leq 5$ (since $\mathsf{cong}_{f^*} \leq 1$ by Lemma 5.1). ∎

Now combined with the results of Section 5.1, we get the result for general graphs.

Theorem 5.6 *Given an instance of QPPC on general graphs, we can find a placement f that incurs on any node v a load of at most $2\mathsf{node\_cap}(v)$, and an edge congestion of at most 5β times the optimum (where β is the performance of the best known congestion tree).*

Since $\beta = O(\log^2 n \log\log n)$, this proves Theorem 1.3.

6. THE FIXED ROUTING PATHS MODEL

In this section we consider a variant of QPPC in which we are given routing paths $P_{v,v'}$ between each pair of vertices. A node v generating an access to element u thus incurs a unit of flow on the edges of $P_{f(u),v}$, where u has been placed at node $f(u)$. In general, we do not require $P_{v,v'}$ and $P_{v',v}$ to be equal. As before, our goal is to find a placement f of quorum elements onto the nodes to minimize the congestion, while respecting the node capacities. First note that Theorem 1.2 applies to this variant; if we are not allowed to violate the node capacities then even finding a feasible solution is NP hard. As before, we retreat to the task of finding solutions that approximate the congestion well, but may violate the node capacities by a small multiplicative factor. However, even if we allow ourselves to ignore the node capacity constraints entirely (i.e., violate them by arbitrary factors), minimizing the congestion is still fairly inapproximable, as the following result states.

Theorem 6.1 *In the fixed routing paths model, it is NP hard to c-approximate the minimum congestion of a QPPC problem, for all $c \in \mathbb{N}$, even on instances where $\mathsf{node\_cap}(v) = \infty$ for all v, and $\mathsf{load}(u) = \mathsf{load}(u')$ for all $u,u' \in U$. Furthermore, unless* **NP** $\subseteq$ **ZPTIME** $\left(n^{O((\log\log(n))^2)}\right)$*, it is NP hard to $o(\sqrt{\log\log(n)})$ - approximate the minimum congestion QPPC solution, even on instances where $\mathsf{node\_cap}(v) = \infty$ for all v, and $\mathsf{load}(u) = \mathsf{load}(u')$ for all $u,u' \in U$.*

Proof. Recall that for a vector x, $\|x\|_p := \left(\sum_i x_i^p\right)^{1/p}$, and $\|x\|_\infty = \max_i\{x_i\}$. The proof proceeds along similar lines as the proof of hardness of the Vector Scheduling problem given by Chekuri and Khanna [4]. We reduce Independent Set to QPPC instances with $\mathsf{node\_cap}(v) = \infty$ for all v, and $\mathsf{load}(u) = \mathsf{load}(u')$ for all $u,u' \in U$. For a graph G, let $\alpha(G)$ be the size of the largest independent set in G, and let $\omega(G)$ be the size of the largest clique in G. Lemma 6.2 states that $\alpha(G) \geq \frac{1}{2e} n^{1/\omega(G)}$, where G has n vertices. Now consider the following multi-dimensional packing problem (MDP): given $A \in \{0,1\}^{d\times n}$ and $k \leq n$, minimize $\|Ax\|_\infty$ such that $x \in \{0,1\}^n$ and $\|x\|_1 = k$. We can reduce MDP to QPPC instances with $\mathsf{load}(u) = \mathsf{load}(u')$ for all $u,u' \in U$ in an approximation preserving fashion as follows. We construct a quorum system on k elements with uniform load. We add d vertex disjoint edges of unit capacity, one for each row of the matrix A, as well as two sources of quorum accesses, s_1 and s_2. Partition of columns of A into sets $S_1, S_2, \ldots, S_r$ using the natural equivalence relation on the column vectors, and add a vertex v_i for each S_i with $\mathsf{node\_cap}(v_i) = |S_i|$. Note that if $|S_i| = k$, we can set $\mathsf{node\_cap}(v_i) = \infty$. We also add a bottleneck edge of capacity $1/n^2$. We route the paths to ensure that placing an element at v_i is like selecting a column in S_i (add some infinite capacity edges to the graph as needed). Finally, we ensure that no elements are placed at nodes other than $\{v_1, \ldots, v_r\}$ by routing paths to these other nodes through the bottleneck edge.

Note that since we want to restrict ourselves to MDP instances that reduce to QPPC instances with uniform load and $\mathsf{node\_cap}(v) = \infty$ for all v, we require the matrix A to satisfy the following property: if $\vec{a}$ is column vector of A, then A must have at least $k-1$ other column vectors that equal $\vec{a}$.

We now proceed by reducing Independent Set to such MDP instances. Let G be an Independent Set instance on n nodes. Fix parameters k and B. We construct a matrix A' with n columns, corresponding to each node of G. For each clique C in G of size $B+1$ or smaller, add a row C to A' such that $a'_{C,v} = 1$ if $v \in C$, and zero otherwise. Now construct a matrix A with kn columns, consisting of k copies of each column of A'. Call $x \in \{0,1\}^{kn}$ *valid* if $\|x\|_1 = k$. Note that if $\|Ax\|_\infty > 1$ for all valid x, then $\alpha(G) < k$. Furthermore, if there exists a valid x such that $\|Ax\|_\infty \leq B$, then $\alpha(G) \geq \frac{1}{2e} k^{1/B}$. To prove this, consider a graph G' that is constructed from G by replacing each node v of G with a clique C_v of size k, and adding all edges in $C_v \times C_{v'}$ to G' whenever (v,v') is an edge of G'. Clearly, $\alpha(G) = \alpha(G')$. Note that since $\|Ax\|_\infty \leq B$, the subgraph G'_x of G' induced on $\{v | x_v = 1\}$ has $\omega(G'_x) \leq B$, so

$$\alpha(G'_x) \geq \frac{1}{2e}|V[G'_x]|^{1/\omega(G'_x)} \geq \frac{1}{2e}|V[G'_x]|^{1/B} = \frac{1}{2e}k^{1/B} \qquad (6.12)$$

and clearly, $\alpha(G) = \alpha(G') \geq \alpha(G'_x)$.

Given a ρ-approximation for MDP on these instances (obtained from a ρ approximation for QPPC on uniform load, infinite node capacity instances), we approximate Independent Set on G as follows. Set $k := n^{\rho/(\rho+1)}$, $B := \rho$, and construct matrix A accordingly. Let x be the output of the MDP algorithm. If $\|Ax\|_\infty > B$, output one, otherwise output $\frac{1}{2e}k^{1/B}$. The output is always at most $\alpha(G)$ by equation 6.12. Furthermore, in the first case $\|Ax\|_\infty > 1$ for all valid x, since we used a ρ-approximation for MDP, and thus $\alpha(G) < k$. In the latter case, $\alpha(G) \leq n$ trivially, so we have a $\max\{k, en/k^{1/B}\} = 2e \cdot (n^{1-\frac{1}{B}})$-approximation. (Note that the reduction takes poly(n^ρ) time.) Combining this reduction with known hardness results for Independent Set (see [7] and references therein), completes the proof. ∎

Lemma 6.2 *In any undirected graph G on n nodes, $2e \cdot \alpha(G) \geq n^{1/\omega(G)}$ where $\alpha(G)$ is the size of the largest independent set in G, and $\omega(G)$ is the size of the largest clique in G.* [2]

Proof. Suppose for a contradiction that $n > (2e \cdot \alpha(G))^{\omega(G)}$. Using the well known Erdös-Szekeres bound on the Ramsey number $R(s,t)$, namely $R(s,t) \leq \binom{s+t-2}{s-1}$, we conclude that

$$n > (2e \cdot \alpha(G))^{\omega(G)} \geq \binom{\alpha(G)+\omega(G)}{\omega(G)} \geq R(\alpha(G)+1, \omega(G)+1)$$

Thus, by the definition of $R(\cdot,\cdot)$, G has an independent set of size $\alpha(G)+1$ or a clique of size $\omega(G)+1$, which yields the desired contradiction. ∎

We now develop an approximation algorithm for QPPC in the fixed paths model, starting with instances with uniform element loads.

[2] We note that stronger versions of this lemma exist, and a similar lemma is stated without proof in [4], however this version is sufficient for our purposes.

6.1 Uniform Element Loads

Theorem 6.3 *There is a polynomial time randomized algorithm that, given an instance of the QPPC problem in the fixed routing paths model in which* $\mathsf{load}(u) = \mathsf{load}(u')$ *for all* $u, u' \in U$*, yields a* $(O(\log n / \log\log n), 1)$*-approximation.*

We reformulate the QPPC problem in the fixed paths model with uniform elements loads as follows. Assume WLOG that for each $u \in U$, $\mathsf{load}(u) = l$. Consider placing a logical element u at a node v. Since the loads are uniform, placing any logical element at v results in the same increase in congestion to the edges of the network. We represent this as a vector $c_v \in \mathbb{R}^{|E|}$, where the coordinates are indexed by edges. Thus coordinate e of c_v is the expected congestion incurred by placing an element at v. For each v, suppose we can place at most $h(v) := \left\lfloor \frac{\mathsf{node\_cap}(v)}{l} \right\rfloor$ logical elements at v while respecting the node capacities. Consider a matrix A that has exactly $b := \sum_v h(v)$ columns, consisting of $h(v)$ copies of c_v for each v. We say these $h(v)$ columns are *associated* with v. Our variant of the QPPC problem thus becomes

$$\textbf{minimize} \quad \|Ax\|_\infty \quad \textbf{s.t.} \quad x \in \{0,1\}^b \text{ and } \|x\|_1 = |U|$$

We say that x *selects* columns i for which $x_i = 1$, and for each column associated with v that x selects, we place a logical element at v. We call the resulting assignment f_x. It is easy to encode this formulation as an ILP and take the LP relaxation.

$$\begin{array}{lcll} \lambda^* = & \text{minimize } \lambda & & \\ \lambda & \geq & \sum_j a_{ij} x_j & \forall i \\ \sum_j x_j & = & |U| & \\ x_j & \in & [0,1] & \forall j \end{array}$$

To solve this LP we can start by guessing the optimal congestion [3] cong^*, and remove all columns containing any entry $a_{ij} > \mathsf{cong}^*$ from the matrix A. We then solve the resulting LP, and apply the rounding scheme of Srinivasan [27] to the resulting optimal fractional solution x to get an integral vector y.

Using this rounding procedure, Srinivasan guarantees that $\|y\|_1 = |U|$, and for all vectors a such that $a_j \in [0,1]$ for all j, and for all $\delta \geq 0$ and $\mu \geq \mathbf{E}\left[\sum_j a_j y_j\right]$

$$\mathbf{Pr}\left[\sum_j a_j y_j \geq \mu(1+\delta)\right] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu \qquad (6.13)$$

As before, we can scale the values a_{ij} by $1/\mathsf{cong}^*$, so that the optimal congestion becomes one, and each $a_{ij} \leq 1$. We can apply then equation 6.13 to bound the congestion on a fixed edge i. Note that $\mathbf{E}\left[\sum_j a_{ij} y_j\right] = \sum_j a_{ij} x_j \leq 1$, since the optimal congestion is one, so we set $\mu = 1$. For any constant c, we can apply equation 6.13 with some $\delta = \Theta(\log n / \log\log n)$ to prove that the congestion on edge i exceeds the optimal congestion by more than an additive factor of δ with probability at most $1/n^c$. Taking a union bound over the edges, we infer that the congestion is $O(\log n / \log\log n)$ with high probability. Thus the placement f_y is a $(O(\log n / \log\log n), 1)$-approximation.

The algorithm is summarized as follows:

[3] If guessing cong^* requires too much nondeterminism, it is sufficient to guess $t = \left\lceil \log_{(1+\varepsilon)}(\mathsf{cong}^*) \right\rceil$, for any $\varepsilon > 0$, and use $(1+\varepsilon)^t$ as an estimate for cong^*. This increases the bound on congestion by a factor of $1+\varepsilon$.

Algorithm for uniform load instances:
Generate matrix A and guess cong^*.
Remove columns j of A with $\max_i\{a_{ij}\} > \mathsf{cong}^*$.
Optimally solve the resulting LP to get solution x.
Round x to y using the rounding in [27].
Output f_y.

6.2 The General Case

Here, let $\mathscr{A}$ be any algorithm for uniform load instances. If $\mathscr{A}$ is the algorithm given above, we suppose it is given its guess for cong^* as part of its input.

Algorithm for general instances:
Guess $\kappa = \mathsf{cong}^*$.
For each $u \in U$, round $\mathsf{load}(u)$ down to the nearest power of two. Call the result $\mathsf{load}'(u)$.
Let $L := \{\mathsf{load}'(u) \mid u \in U\}$.
For each $l \in L$, in decreasing order of size
  Run $\mathscr{A}$ on $U_l := \{u \in U \mid \mathsf{load}'(u) = l\}$,
  using κ as the input guess if needed.
  Place U_l as $\mathscr{A}$ suggests, and decrease $\mathsf{node\_cap}(\cdot)$ accordingly. That is, if t elements of U_l are placed on v, decrease $\mathsf{node\_cap}(v)$ by tl.

Lemma 6.4 *If* $\mathscr{A}$ *is a* (α, β)*-approximation for QPPC instances with uniform load in the fixed routing paths model, then the above algorithm is a* $(\alpha|L|, 2\beta)$*-approximation for general QPPC instances in the fixed routing paths model.*

Proof. Suppose f is the placement output by the algorithm. We first prove $\mathsf{load}_f(v) \leq 2\beta\,\mathsf{node\_cap}(v)$ for each v. Note that it suffices to show that $\mathsf{load}'_f(v) \leq \beta\,\mathsf{node\_cap}(v)$, since $\mathsf{load}(u) \leq 2\mathsf{load}'(u)$ for all v. From now on all references to load refer to load'.

Fix any v. Suppose $\mathscr{A}$ is run on elements u with $\mathsf{load}'(u) = l$ and places t of them on v. There are two cases: either at this stage, $\mathsf{node\_cap}(v) \geq tl$, in which case we can charge the load these elements cause to the corresponding decrease in $\mathsf{node\_cap}(v)$, or else $\mathsf{node\_cap}(v) < tl$. In the latter case, we know that $\mathsf{node\_cap}(v) \geq tl/\beta$ since $\mathscr{A}$ is an (α, β) - approximation, so we can charge tl/β to $\mathsf{node\_cap}(v)$. Furthermore, since $\mathsf{node\_cap}(v)$ is reduced to zero, v is not assigned any additional elements later on. Thus we can charge $1/\beta$ of the load to $\mathsf{node\_cap}(v)$, and conclude that $\mathsf{load}'_f(v) \leq \beta\,\mathsf{node\_cap}(v)$.

We now bound the congestion caused by each execution of $\mathscr{A}$ by $\alpha \cdot \mathsf{cong}^*$. To do this, it suffices to prove that the optimal congestion is at most cong^* in each instance on which $\mathscr{A}$ is run. Fix an instance, say on elements with $\mathsf{load}'(u) = l$, denoted U_l. All elements with larger loads have already been placed, thus reducing the node capacities at some nodes. For a placement f, node v, and $W \subseteq U$, let

$$\mathsf{cap}(f, v, W) \;=\; \mathsf{node\_cap}(v) \;-\; \sum_{u \in W : f(u) = v} \mathsf{load}'(u)$$

denote the remaining load at v after placing down W using f. Here, $\mathsf{node\_cap}(v)$ are the original input node capacities. Let $U' := \{u \mid \mathsf{load}'(u) \geq 2l\}$, and let f be the partial placement of U' created by the algorithm so far. Fix any optimal solution f^*. We can place down elements of U_l at nodes v such that $\mathsf{cap}(f, v, U') - \mathsf{cap}(f^*, v, U' \cup U_l) > 0$. Specifically, we place down $\lfloor(\mathsf{cap}(f, v, U') - \mathsf{cap}(f^*, v, U' \cup U_l))/l\rfloor$ such elements at v. It remains to show that we can place all of U_l down this way. To see this, first note that, by a simple volumetric argument,

$$\sum_v \left(\mathsf{cap}(f, v, U') - \mathsf{cap}(f^*, v, U' \cup U_l)\right) = l \cdot |U_l|$$

Next, observe that $\mathsf{cap}(f, v, U') - \mathsf{cap}(f^*, v, U' \cup U_l)$ is always a multiple of l, since all elements of U' and U_l have loads that are multiples of l. (Note how we have used the fact that the loads $load'(u)$ are multiples of two.) Combining these two facts, we see that we can pack U_l in node capacity occupied by elements of $U' \cup U_l$ under placement f^*, while respecting node capacity constraints (with respect to load'), no matter how f placed down U'. Having done this, it is clear that the resulting congestion due to placing U_l is no more than cong^*.

Since the congestion due to $\mathscr{A}$ on each instance is at most $\alpha \cdot \mathsf{cong}^*$, we conclude that all executions of $\mathscr{A}$ together contribute congestion at most $|L| \cdot \alpha \cdot \mathsf{cong}^*$. ∎

Note that $|L| = |\{\lfloor \log_2(\mathsf{load}(u)) \rfloor \mid u \in U\}| = \eta$, so using the algorithm given above for $\mathscr{A}$, with $\alpha = O(\log n / \log\log n)$ and $\beta = 1$, we complete the proof of Theorem 1.4.

7. CONCLUSIONS

In this paper we studied the problem of placing the elements of a universe U underlying a quorum system $\mathscr{Q}$ on a network G in a way that minimizes congestion due to quorum accesses, while respecting the computing capacity of each network node. We considered this problem in two models, differing on the basis of whether communication routes are fixed or can be chosen. We showed that in either case, this problem cannot be approximated to within *any* factor (unless P=NP). However, by allowing doubling of the capacity of each node, we present efficient approximation algorithms for this problem in both models.

8. REFERENCES

[1] Y. Amir and A. Wool. Optimal availability quorum systems: theory and practice. *Inf. Proc. Lett.*, 65(5):223–228, 1998.

[2] D. Barbara and H. Garcia-Molina. The reliability of voting mechanisms. *IEEE Trans. Comput.*, 36(10):1197–1208, 1987.

[3] M. Bienkowski, M. Korzeniowski, and H. Räcke. A practical algorithm for cosntructing oblivious routing schemes. In *Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures*, pages 24–33, 2003.

[4] Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. In *SODA '99: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 185–194, 1999.

[5] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. *Knowledge and Data Engineering*, 4(6):582–592, 1992.

[6] Y. Dinitz, N. Garg, and M. X. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19(1):17–41, 1999.

[7] Lars Engebretsen and Jonas Holmerin. Towards optimal lower bounds for clique and chromatic number. *Theor. Comput. Sci.*, 299(1-3):537–584, 2003.

[8] A. Waichee Fu. Delay-optimal quorum consensus for distributed systems. *IEEE Trans. Parallel and Dist. Sys.*, 8(1):59–69, 1997.

[9] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP)*, pages 150–162, 1979.

[10] S. Gilbert and G. Malewicz. The quorum deployment problem. In *8th International Conference on Principles of Distributed Systems (OPODIS'04)*, pages 218–228, 2004.

[11] A. Gupta, B. Maggs, F. Oprea, and M.K. Reiter. Quorum placement in networks to minimize delays. In *PODC '05: Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing*, pages 87–96, 2005.

[12] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minimize congestion. In *Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures*, pages 34–43, 2003.

[13] Y. Hassin and D. Peleg. Average probe complexity in quorum systems. In *PODC '01: Proceedings of the twentieth annual ACM Symposium on Principles of Distributed Computing*, pages 180–189, 2001.

[14] N. Kobayashi, T. Tsuchiya, and T. Kikuno. Minimizing the mean delay of quorum-based mutual exclusion schemes. *Journal of Systems and Software*, 58(1):1–9, 2001.

[15] A. Kumar, M. Rabinovich, and R. K. Sinha. A performance study of general grid structures for replicated data. In *Proceedings 13th International Conference on Distributed Computing Systems*, pages 178–185, 1993.

[16] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[17] T. Leighton, F. Makedon, and I. G. Tollis. A 2n-2 step algorithm for routing in an nxn array with constant size queues. In *SPAA '89: Proceedings of the first annual ACM Symposium on Parallel Algorithms and Architectures*, pages 328–335, 1989.

[18] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. on Computer Systems*, 3:145–159, 1985.

[19] B. M. Maggs, F. Meyer auf der Heide, B. Vocking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *IEEE Symposium on Foundations of Computer Science*, pages 284–293, 1997.

[20] D. Malkhi, M. K. Reiter, and A. Wool. The load and availability of Byzantine quorum systems. *SIAM J. Comput.*, 29(6):1889–1906 (electronic), 2000.

[21] D. Malkhi, M. K. Reiter, A. Wool, and R. N. Wright. Probabilistic quorum systems. *Inf. Com.*, 170(2):184–206, 2001.

[22] M. Naor and A. Wool. The load, capacity, and availability of quorum systems. *SIAM J. Comput.*, 27(2):423–447, 1998.

[23] D. Peleg and A. Wool. The availability of quorum systems. *Inf. Comp.*, 123(2):210–223, 1995.

[24] D. Peleg and A. Wool. Crumbling walls: A class of practical and efficient quorum systems. *Dist. Comp.*, 10(2):87–97, 1997.

[25] H. Räcke. Minimizing congestion in general networks. In *IEEE Symposium on Foundations of Computer Science*, pages 43–52, 2002.

[26] P. Raghavan and C. D. Thompson. Provably good routing in graphs: regular arrays. In *STOC '85: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 79–87, 1985.

[27] Aravind Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *FOCS '01: Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, page 588, 2001.

[28] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180–209, 1979.

[29] T. Tsuchiya, M. Yamaguchi, and T. Kikuno. Minimizing the

maximum delay for reaching consensus in quorum-based mutual exclusion schemes. *IEEE Transactions on Parallel and Distributed Systems*, 10(4):337–345, 1999.

[30] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, May 1982.

[31] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *STOC '81: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, pages 263–277, 1981.

[32] M. Westermann. *Caching in Networks: Non-Uniform Algorithms and Memory Capacity Constraints*. Dissertation, Universität Paderborn, Heinz Nixdorf Institut, Theoretische Informatik, 2001.

APPENDIX

A. THE MIGRATION MODEL

In this section we study the congestion of a quorum system placement in a variant of the arbitrary routing model. We assume that the logical elements of U can *migrate* from one physical node to another. For simplicity we ascribe zero cost to the migration of logical elements, leaving as future work the study of the problem in a model with non-zero costs for migration.

Our objective function is the congestion of the most congested edge $e \in E$ amortized over Δ time units, where each element $u \in U$ is stationary during each time unit (and can migrate in between). A solution to this problem is a *placement with migration*, i.e., a function $h : U \times \{1,\ldots,\Delta\} \to V$, where $h(u,t)$ specifies the node v that hosts u during time unit t. No bounds are placed on the capacity of any physical node, in other words, load is not an issue here. As with migration cost, we leave the problem of addressing load in a migration model as future work.

We now give an example which shows that, in arbitrary graphs, migration can indeed help reduce congestion. Consider the complete graph K_n on n vertices, with each edge having unit capacity, and assume that the universe of logical elements consists of a single node, $U = \{u\}$. A static strategy would specify a placement $f : U \to V$ of u on $v = f(u)$, one of the nodes of K_n. Assuming that each client sends a request each time unit, the amortized congestion of the placement is 1.

Consider now what happens when we allow migration. Suppose that after each client request we move the logical element from one physical node to the next in a circular manner such that all nodes are used. In this case all edges have congestion $1/\frac{n}{2}$ which is less than the one obtained for a fixed placement. In fact, a simple averaging argument shows that $O(n)$ is the largest gap that can be obtained between the congestions of the two models (with and without migration).

This example indicates that studying the model in which migration is allowed can have possible benefits in terms of congestion. Unfortunately, this is not true for all graphs, in particular, it is not true for trees, as we will now prove.

Lemma 1 *For a tree T there exists a node v_0 such that no placement with migration h of a quorum system over the universe with a single element $U = \{h\}$ can have a congestion better than that of h_{v_0}, where $h_{v_0}(u,t) = v_0$ for each $t \in \{1,\ldots,\Delta\}$.*

Proof. The proof is similar to that of Lemma 5.3. Let $h : U \times \{1,\ldots,\Delta\} \to V$ be an arbitrary placement with migration, and let $h_t = h(\cdot,t) : U \to V$ be the placement specified by h at time t. For an edge $e \in E$, let $r(T_L)$ and $r(T_R)$ be the request rates of clients coming from the two subtrees T_L and T_R (obtained by removing e from T). Let also $\mathsf{load}_{h_t}(T_L) = \sum_{v \in T_L} \mathsf{load}_{h_t}(v)$ and $\mathsf{load}_{h_t}(T_R)$ defined similarly, be the expected number of messages seen by nodes in T_L and T_R respectively, at time t. Then the congestion of the edge e over the time period Δ is

$$\sum_{t=1}^{\Delta} \frac{r(T_L) \times \mathsf{load}_{h_t}(T_R) + r(T_R) \times \mathsf{load}_{h_t}(T_L)}{\mathsf{edge\_cap}(e)} \qquad \text{(A.14)}$$

Let v_0 be the node found in Lemma 5.3 and assume that $r(T_L) \leq r(T_R)$. Node v_0 has to lie in T_R and thus the congestion of e for the placement h_{v_0} with migration is $\frac{\Delta \times r(T_L) \times \left(\mathsf{load}_{h_{v_0}}(T_R) + \mathsf{load}_{h_{v_0}}(T_L)\right)}{\mathsf{edge\_cap}(e)}$. Since $r(T_L) \leq r(T_R)$ and the total load of the system does not change, this is at most the quantity given by A.14, which completes our proof. ∎

A.1 A solution for arbitrary graphs.

To obtain a solution for arbitrary graphs we will use Räcke's results on congestion trees. Consider an arbitrary graph G and construct its associated congestion tree T_G. Then find the node v_0 from Lemma 5.3 that minimizes congestion, assuming the request rates of clients are known. If the node v_0 is a leaf we are done, we can simply use the placement f_{v_0} in the original graph G with only a polylog n loss in congestion. If v_0 is an internal node in T_G we need to specify a way in which v_0 gets mapped to one of the nodes of G in the cluster corresponding to v_0 in G. In Räcke's work this was done by choosing the leaf onto which v_0 is mapped, independently at random from a special distribution depending on the cluster corresponding to v_0. More precisely, each leaf was chosen with a probability proportional to its *weight* in that cluster (which was equal to the sum of the capacities of the edges incident to that node that were leaving the cluster). This is done independently at random for each message that is routed through the node v_0.

To obtain the same approximation ratio for congestion, we can do something similar here (this is based on ideas from [32]). After a fixed amount of time, the node in the cluster of v_0 onto which v_0 is mapped, makes a decision as to whether it should keep all the logical elements of U mapped onto itself or it should migrate them to another node of the cluster corresponding to v_0. The next node in the migration chain is picked independently at random from the special distribution mentioned before from the nodes of the cluster. This ensures that over a longer period of time, we will match the conditions that enable Räcke's construction to provide the polylog approximation factor for congestion. By an argument similar to the one from Section 5.1, we can see that our solution will also suffer only a polylog loss in congestion compared to the optimal one in the migration model, regardless of whether that solution uses migration or not.

Here is an example illustrating how our algorithm works for a particular graph. Consider the congestion tree T_{K_n} for the complete graph K_n and assume that all edges of K_n have unit capacity. Assume further, that clients issue requests uniformly from all the nodes of K_n. The tree T_{K_n} will consist of a root and n leaves, each leaf being connected to the root by an edge of capacity $n-1$. The algorithm will find the root as the node minimizing congestion and will place all the elements of U on it. The root is mapped to one of the leaves with probability $\frac{1}{n}$ and then migrated after some fixed amount of time to a new leaf chosen independently at random (and uniformly in this case) from all the leaves of T_{K_n}. This, in fact, corresponds to the optimal solution for the complete graph K_n.