

Minimizing congestion in General Networks using oblivious on-line routing algorithms

Amin Ali and Gurashish Brar

1 Introduction

A principle task in parallel and distributed systems, such as networks of workstation or the Internet, is to reduce the communication load in the interconnection network, for the bandwidth of the connection network is usually the major bottleneck for the performance of distributed applications. This is due to the fact that it is often more expensive or more difficult to increase the bandwidth of the interconnection network than to increase the processor speed and memory capacity at individual nodes. Therefore, a principle task for these systems is to design distributed applications in such a way that they cause as little communication overhead as possible. However, it may not be sufficient to simply reduce the total communication load i.e. the total traffic in the network, as this can result in bottlenecks. In addition, the load has to be distributed among all network resources. This corresponds to minimizing the congestion i.e. the maximum amount of data transmitted by the link taken over all network links divided by the respective bandwidth.

A recent result by Racke shows that there is an oblivious algorithm for arbitrary networks that achieves near optimal congestion i.e. within logarithmic factors from optimal. In his paper, Racke proved that in an arbitrary network, one can route any multicommodity flow with a single, randomized oblivious routing strategy which incurs maximum congestion on any edge at most an $O(\log^3 n)$ factor more than the best possible. His construction is based on producing a tree decomposition of the network, and then showing that routing on the tree is almost as good as routing in the original network.

Racke's results marked a breakthrough in this field for it was the first oblivious path selection algorithm that achieves a poly-logarithmic competitive ratio in general networks. The previously best known factor for an oblivious algorithm was polynomial in n . Unfortunately, his result was essentially existential, since the algorithm used to construct the routing strategy took exponential time. In 2003, Bienkowski et al. showed how to make Racke's original tree decomposition-based routing scheme constructive, and proved a competitive ratio bound of $O(\log^4 n)$. At the same time, Harrelson et al. proved a similar result, albeit independently, which improved the bound to $O(\log^2 n \log \log n)$.

Our main objective for this paper is to illustrate Racke's results. We start off by showing that randomization is essential for an optimal oblivious routing strategy by looking at lower bounds established for deterministic and randomized routing problem. Then we move on to the $\Omega(\text{Log}N)$ lower bound on the competitive ratio of the network congestion problem presented by Aspnes et al. He also provides an $O(\text{Log}N)$ competitive on-line routing algorithm, but the drawback of the algorithm is that it is centralized and that the current network load must be known in order to make a routing decision, hence not practical. We finish off our paper with Racke's recent result and in which he remarkably proves that any undirected multicommodity flow problem can be routed in an oblivious fashion with congestion that is within a factor of $O(\text{Log}^3 N)$ of the best off-line solution to the problem.

2 Preliminaries

2.1 Parallel Computer Architecture

Parallel computers consist of multiple processing units, called nodes, interconnected by a specific interconnection topology. Each node is uniquely identified by a number between 1 and N . All interconnection between the nodes occur in synchronous steps. Each link can carry a finite number of unit messages, called the packets, in one step. The maximum number of packets a link can carry at any given time is called the bandwidth of the link. During a step, a node can send at a finite number of packets to each of its neighbors. This interconnection topology is modeled by undirected graphs.

2.2 Permutation Routing

Assume a network of N nodes, $\{1, \dots, N\}$. Each link can carry at most one packet in one step, and during each step, each node can send at most one packet to each of its neighbours. Each node i contains one packet v_i that should be routed to the destination node. Each destination node $d(i)$ for each node i , for $1 \leq i \leq N$ should form a permutation of $\{1, \dots, N\}$, i.e. every node is the destination of exactly one packet.

A route for a packet is a sequence of edges it can follow from its source to its destination. An algorithm for permutation routing problem must specify a route for each packet. In following a route, a packet may occasionally have to wait at an intermediate node because the next edge on its route is busy transmitting another packet. We assume that each node contains one queue for each edge leaving the node; the queue holds the packets waiting to leave via that edge. A routing problem must also specify a queuing discipline for resolving conflicts between packets that simultaneously wish to follow the same edge out of a node.

2.3 Oblivious Routing

A permutation routing algorithm is *oblivious* if the path of every packet is specified independently of the paths of other packets. We use the following convention:

Definition 1. *A permutation routing problem is oblivious if the route followed by v_i depends on $d(i)$ alone, and not on $d(j)$ for any $j \neq i$. An oblivious routing specifies for each pair $(i, d(i))$, a route between node i and node $d(i)$.*

Oblivious routing algorithms are attractive for their simplicity of implementation: the communication hardware at each node in the network can determine the next link of its route simply by looking at the source and destination information carried by a packet. Often, the topology of the network makes this operation very simple. The communication hardware at a node does not have to compare the source and destination of different packets in its queues. Oblivious Routing algorithms are useful in Distributed algorithms and Online Routing problems, where packets continuously arrive in the network.

2.4 Variants of On-line Routing Problem

The Routing problem has two different variants. The difference between the two variants is the metrics they use for optimization. In the first variant, load represents execution time. The goal in

this case is to minimize the deal time of the packet sequence. In the second variant, load represents bandwidth used. The goal here is to minimize the maximum load.

The known techniques used to construct efficient on-line algorithms for the first variant cannot be directly applied to the second variant. Intuitively, the reason is that the techniques for the first variant rely on a delaying assignment (queuing) of a packet until previously transmitted packets reach their destination. In the second variant, there is no notion of delay. However, an algorithm for the second variant can be used to solve a modification of the first variant where the packets are delivered immediately upon their arrival i.e. no packet delays are allowed. This indicates that the second variant is a harder problem.

3 Deterministic Oblivious On-Line Routing

In this section, we illustrate the lower bound on the number of steps it takes to route a packet from its source to its destinations using deterministic on-line routing algorithms, as proved by Kalamanis et al.

3.1 Lower Bound

Theorem 1. *Any deterministic oblivious permutation routing algorithm in a N -node degree- d network requires $\Omega(\frac{\sqrt{N}}{d})$ steps.*

Proof: reference [1].

3.2 Hypercube

A scalable topology that has been used in several parallel processors is the Hypercube. A line connecting two nodes defines a 1-dimensional 'cube'. A square with four nodes is a 2-dimensional cube, and a 3D cube has eight nodes. This pattern reveals a rule for constructing an n -dimensional cube: begin with an $(n - 1)$ -dimensional cube, make an identical copy, and add links from each node in the original to the corresponding node in the copy. Doubling the number of nodes in a hypercube increases the degree by only 1 link per node, and likewise increases the diameter by only 1 path. An n -dimensional hypercube has 2^n nodes and n degree .

Communication in a hypercube is based on the binary representation of node addresses. The nodes are numbered so that two nodes are adjacent if and only if the binary representations of their addresses differ by one bit. For example, nodes 0110 and 0100 are immediate neighbors but 0110 and 0101 are not. An easy way to label nodes is to assign node addresses as the cube is constructed. When you copy an $(n - 1)$ -dimensional cube, make sure the corresponding nodes in the two copies have the same addresses. Then extend all the addresses by one bit. Append a 0 to the addresses of nodes in the original cube, and append a 1 to the addresses of nodes in the copy.

3.3 Bit-Fixing Algorithm

Node addresses are the basis for a simple algorithm for routing information in a hypercube. An n -dimensional cube will have a n -bit node address. Sending a message from node A to node B can be done in n cycles, where on each cycle a node will either hold a message or forward it along one

of its links. On cycle i the node that currently holds the message will compare bit i of its own address with bit i of the destination address. If the bits match, the node holds the message. If they don't match, it forwards the message along dimension i , where dimension i is the dimension that was added in the i^{th} step of the construction of the cube (i.e. it is the same 'direction' at all nodes).

We now define the Bit-Fixing algorithm, which is an example of a deterministic oblivious routing algorithm:

Definition 2. *The Bit-Fixing Algorithm routes all partial permutations on an n -dimensional hypercube as follows:*

- *Given a destination address $d(i)$ and an intermediate node $a(i)$, compare the bits of $d(i)$ with $a(i)$ from left to right*
- *If $a(i)$ is the same as $d(i)$, then terminate.*
- *Else identify the first bit position at which these two addresses differ*
- *Route this packet to its neighbor $n(i)$ such that $a(i)$ and $n(i)$ differ only in this bit position.*
- *Set $a(i)$ to $n(i)$ and loop.*

Example:

- Source: (0,0,0,0,0,0), Destination: (1,0,1,0,1,1)
- (0,0,0,0,0,0) \rightarrow (1,0,0,0,0,0) \rightarrow (1,0,1,0,0,0) \rightarrow (1,0,1,0,1,0) \rightarrow (1,0,1,0,1,1)

3.4 Lower bound on the Bit-Fixing algorithm

Corollary 1. *On an n -dimensional hypercube, there is an instance of permutation requiring $\Omega(\frac{\sqrt{2^N}}{n})$ steps for the bit-fixing routing algorithm. (Note: It satisfies earlier theorem where $N = 2^n$ and $d = n$).*

Proof: Let $(i.j)$ be the address of a node, where i and j are two binary strings of each length $\frac{n}{2}$ and $.$ is the concatenation operation. Consider a packet which is stored on node $(i.j)$ and is routed to the destination node $(j.i)$. We look at the sources where $j = 0$.

- $(i.0) \rightarrow (0.i)$

Using the Bit-Fixing algorithm, we observe that for odd i (i.e. the rightmost bit is 1), the packet must pass through node $(1,0)$. The number of nodes such that $(i.0)$ for i odd is $\frac{2^{n/2}}{2}$. Now imagine a scenario where all these nodes send off a packet to their respective destination. Since only one packet can be routed on the same edge at a time, packets would start queuing up at node $(1,0)$. The last packet in the queue will be delivered after $\frac{2^{n/2}}{2}$ steps, which is worse than $\frac{\sqrt{N}}{d}$, and thus illustrates the $\Omega(\frac{\sqrt{N}}{d})$ lower bound on deterministic oblivious on-line routing algorithms.

4 Randomized Oblivious On-line Routing

In this section, we illustrate the lower bound on the number of steps it takes to route a packets from its source to its destination using randomized online routing algorithms.

4.1 Randomized Bit-Fixing Algorithm

Definition 3. *The Randomized Bit-Fixing Algorithm routes all partial permutations on an n -dimensional hypercube as follows:*

Given a node i and destination address $d(i)$, route a packet v_i by executing the following two steps independently of all the other packets:

- *Choose a random intermediate destination t_i from $1, \dots, N$, and route v_i from i to t_i using bit-fixing algorithm.*
- *Route v_i from t_i to its final destination $d(i)$ using bit-fixing algorithm*

4.2 Analysis of Randomized Bit-Fixing algorithm

Lemma 1. *If the bit-fixing algorithm is used to route a packet v_i from i to t_i and v_j from j to t_j , then their routes do not rejoin after they separate*

Proof: Assume k is the node at which the two paths separate and l is the node at which they rejoin. According to bit-fixing scheme, v_i and v_j from k to l depends only on the bit representations of k and l . Therefore, v_i and v_j must follow the same route, which contradicts to the assumption.

Let the route of packet v_i follow the sequence of edges $p_i = (e_1, e_2, \dots, e_k)$. Let S be the set of packets (other than v_i) whose routes pass through at least one of $\{e_1, e_2, \dots, e_k\}$

Lemma 2. *The delay incurred by v_i is at most $\|S\|$*

Proof: Define lag l for any packet w , $l = t - j$ (a packet is ready to follow edge e_j at time t). If the lag of v_i increase from l to $l + 1$, some packet should have lag l in front of v_i . Let t_j be the last time step at which any packet in S has lag l . A packet w must follow the edge e_j where $l = t_j - j$ and it must leave at $t_j + 1$. If the lag of v_i reaches $l + 1$, some packet in S leaves p_i with lag l .

By the previous lemma, the routes of different packets will not rejoin after they separate. Each member of S whose route intersects p_i is charged at most one delay for v_j . Hence, the delay incurred by v_i is at most $\|S\|$.

Definition 4. *Define a random variable H_{ij} as:*

$$H_{ij} = \begin{cases} 1 & p_i \text{ and } p_j \text{ share atleast one edge} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Let $delay_i$ be the total delay incurred by v_i . Then:

$$delay_i \leq \sum_{j=1}^N H_{ij} \quad (2)$$

From linearity of expectation:

$$E[delay_i] \leq E\left[\sum_{j=1}^N H_{ij}\right] = \sum_{j=1}^N E[H_{ij}] \quad (3)$$

For an edge e of the hypercube, let the random variable $T(e)$ be the number of routes that pass through e . If $p_i = (e_1, e_2, \dots, e_k)$, then:

$$\sum_{j=1}^N H_{ij} \leq \sum_{i=1}^k T(e_i) \quad (4)$$

We have:

$$E\left[\sum_{j=1}^N H_{ij}\right] \leq E\left[\sum_{i=1}^k T(e_i)\right] = \sum_{i=1}^k E[T(e_i)] \quad (5)$$

Since all edges in the hypercube are symmetric,

- $E[T(e_l)] = E[T(e_m)]$ for any two edges e_l and e_m
- Total number of edges: Nn
- The expected length of each route is $n/2$
- Expected length of total route is $Nn/2$
- $E[T(e)] = 1/2$ for all edges

We have:

$$E\left[\sum_{j=1}^N H_{ij}\right] \leq E\left[\sum_{i=1}^k T(e_i)\right] = \frac{k}{2} \leq \frac{n}{2} \quad (6)$$

Define the following:

$$\mu = E\left[\sum_{j=1}^N H_{ij}\right] \leq \frac{n}{2} \quad (7)$$

By using Chernoff Bound, we get:

$$Pr\left[\sum_{j=1}^N H_{ij} > (1 + \delta)\mu\right] \leq 2^{-(1+\delta)\mu} \quad (8)$$

Using $\delta = 11$ gives us

$$Pr\left[\sum_{j=1}^N H_{ij} > 6n\right] \leq 2^{-6n} \quad (9)$$

Theorem 2. *With probability at least $1 - 2^{-5n}$, the packet v_i reaches t_i in $7n$ or fewer steps*

Proof: Since the total number of packets is 2^n , the probability that any of them have a delay exceeding $6n$ is less than

$$2^n * 2^{-6n} = 2^{-5n} \tag{10}$$

The packet requires addition n steps to route from the source to the destination.

Theorem 3. *A packet reaches its destination in $14n$ or fewer steps with a probability larger than $(1 - 1/N)$*

Proof:

- Phase 2 of the Valiants scheme is identical to Phase 1
- Fail probability = $2 * 2^{-5n} < 2^{-n} = 1/N$

4.3 Conclusion:

Deterministic routing algorithm may give very poor result at some specific cases, but Randomized routing algorithm can give satisfactory result for all cases with high probability. In fact, any deterministic oblivious algorithm must have performance *exponentially* worse than that of our randomized algorithm.

5 Online Routing

Problem Statement: Given a graph $G(V, E)$ with $\|V\| = n$ and $\|E\| = m$ and a $s : E \rightarrow R^+$. A set of requests arrive as tuples (s_i, t_i, p_i) where s_i is source and t_i is sink and $p_i \in R^+$, is the bandwidth required for the request. A request has to be satisfied immediately on arrival.

Request i is satisfied by assigning it to the Route/Path P_i from s_i to t_i . The solution to the above problem will be a set $P = P_1, P_2 \dots P_k$, for the k requests.

5.1 Lower Bound for Online Routing problem

We first show a lower Bound of $O(\log n)$ for the competitive ratio of any online routing algorithm for the above problem. We consider an oblivious adversary who chooses the next request to be routed immediately.

Consider the following graph $G(V, E)$:

The V consists of one source, n intermediate vertices v_1, v_2, \dots, v_n , n is power of 2. There are n sinks, $\{S(1,1)\}, \{S(2,1), S(2,2)\} \dots \{S(i,1), S(i,2), \dots, S(i, 2^{i-1})\} \dots \{S(\log n, 1) \dots S(\log n, 2^{\log n - 1})\}$

The source is connected to all the intermediate vertices.

- Sink $S(1, 1)$ is also connected to all intermediate vertices.
- $S(2, 1)$ is connected to intermediate vertices $\{v_1, v_2 \dots V_{n/2}\}$
- $S(2, 2)$ is connected to intermediate vertices $\{V_{n/2+1} \dots V_n\}$
- $S(i, j)$ is connected to intermediate vertices $\{V_{(j-1) \frac{n}{2^{i-1}} + 1} \dots V_{j \frac{n}{2^{i-1}}}\}$

The size of graph is clearly $O(n)$. Next we show a sequence of requests for which the offline load on any edge is at most 1 and the online load must be at least $\frac{\log n}{2}$.

The sequence of requests appears in $\log n$ phases and each request has a bandwidth of 1. In phase i , $1 \leq i \leq \log n$, there are $n/2^i$ requests of paths from source to a sink $S(i, j)$, for some j , $1 \leq j \leq 2^{i-1}$. We construct the sequence inductively. Before the first phase begins the load on all edges is 0. Now during any phase i , there are $n/2^i$ paths between the source and the sink $S(i, j)$, the expected increase in load at the end of phase i , on the edges will be at least $\frac{1}{2}$. From the construction of the graph we notice that the set of intermediate vertices connected to sink $S(i, j)$ is union of two disjoint sets of intermediate vertices connected to sinks $S(i+1, 2j-1)$ and $S(i+1, 2j)$.

The adversary for phase $i+1$, always picks the Sink whose paths experienced an increase of $\frac{1}{2}$ during the previous phase. Thus the average expected load of the edges on paths between source and sink $S(i, j)$ is at least $i/2$ at the end of phase i . Hence after the last phase the average expected load of edges on two paths between the source and sink $S(\log n, j)$ is at least $\frac{\log n}{2}$.

The offline algorithm on the other hand can maintain a maximum load of 1 by not using the vertices used in previous phase to route the new requests. This follows because at each phase there are $n/2^i$ requests and there are two disjoint sets of intermediate vertices (or paths) each of size $n/2^i$, and one of which is not associated with the subsequent requests. Thus the offline algorithm achieves a maximum load of 1.

5.2 Optimal Online Route allocation Algorithm

Aspnes et al. [2] presented a centralized, $O(\log n)$ -competitive online routing algorithm. This algorithm is based on the use of an exponential cost function. Each edge e is assigned a length that is exponential in the current load of e . If a routing request occurs the algorithm chooses a shortest path between source and destination with respect to the length assigned to the edges. The competitive ratio of this algorithm is optimal due to the lower bound we provided earlier.

Why Oblivious approach? The drawback of the above algorithm is that it is centralized and that the current network load must be known in order to make a routing decision. A distributed online- algorithm was developed in [3]. But still the major drawback of the earlier online algorithm remained, i.e. requirement of network state information. It is still difficult for a distributed implementation to realize the up-to-date knowledge of the state of the network.

H-Racke [4] later showed that it is not necessary to have the state information of network in order to achieve poly-logarithmic competitive ratio w.r.t. the congestion. Next we present an oblivious online routing algorithm. This oblivious routing algorithm involves randomization to achieve a low competitive ratio, since there exist large lower bounds for deterministic routing protocols.

5.3 A non-constructive $O(\log^3 n)$ competitive Oblivious Routing strategy

This framework is based on a bi-simulation between a target network $G(V, E)$ and an associated decomposition tree T_G , which is constructed from the general network G . G and T_G are weighted with $b : E \rightarrow R^+$ and $b_t : E \rightarrow R^+$. Here b and b_t describe the bandwidth of edges in G and T_G respectively.

The reason we want to do a bi-simulation is because the oblivious routing problem can be solved efficiently on Trees (Since there is only one unique path). So we construct an appropriate tree from

the general network G , and solve the problem for this tree and map the efficient solution back to the general graph G .

We first show that T_G can simulate network G , i.e. any request sequence which produces a congestion C on G produces the same congestion C on T_G . Similarly we will also show that a request sequence that produces congestion C_t on T_G can be simulated by G with congestion $c.C_t+K$ with high probability, where K and c are constants.

5.3.1 Decomposition Tree

T_G corresponds to a hierarchical clustering of the node set in G (V, E). The hierarchical clustering is described by a system of subsets of V . Complete Laminar Set System: A system of sets that contains no intersecting subsets X and Y is called laminar. A complete laminar system contains the set V and all sets $v, v \in V$. Given a complete laminar set system \mathbf{S} we construct the decomposition tree $T_G(V_t, E_t)$ as follows: For each $S \in \mathbf{S}$ the set V_t contains a node v_t . Therefore we have a direct mapping between the clusters in the hierarchy with the tree nodes.

Notation: We denote a cluster in G corresponding to the tree node v_t , with S_{v_t} . Two nodes u_t and v_t in T_G are connected if $S_{u_t} \subset S_{v_t}$ or $S_{v_t} \subset S_{u_t}$ and there is no S such that $S_{u_t} \subset S \subset S_{v_t}$ or $S_{v_t} \subset S \subset S_{u_t}$.

T_G is assumed to be rooted at v_t corresponding to the cluster V .

5.3.2 Definitions

Level The root node is considered to be at level 0 and all nodes whose parents are at level l are considered to be at level $l+1$.

Capacity Function Given any two subsets $X, Y \subset V$, $cap(X, Y) = \sum_{x \in X, y \in Y} b((x, y))$. Where $b(x, y)$ denotes the bandwidth of edge $(x, y) \in E$ in $G(V, E)$.

Out(X) := $cap(X, X)$: Defines the total bandwidth of edges leaving X

Bandwidth of Tree edge (u_t, v_t) u_t being parent of v_t . $b_t(u_t, v_t) = Out(S_{v_t})$

Height $h(T_G)$ denotes the height of tree T

Set V_t^l denotes the set of vertices v_t at level l in T_G .

Weight Function The weight function defines the bandwidth of all edges adjacent to nodes in X that do not connect nodes of same cluster, where the clusters correspond to the set of vertices in V_{l_t} .

$$w_l(X) = cap(X, Y) - \sum_{v_t \in V_t^l} cap(X \cap S_{v_t}, S_{v_t}) \quad (11)$$

Properties of weight function

1. Additive
2. $w_l(S_{v_t}) = out(S_{v_t})$
3. $w_{l-1}(X) \leq w_l(X)$

Bandwidth Ratio $\lambda(v_t)$ The bandwidth ratio λ_{v_t} describes the ratio between the capacity of edges that leave a set U (i.e., $\text{out}(U)$) and the capacity of edges that connect U with the rest of the cluster (i.e., $\text{cap}(U, S_{v_t} \setminus U)$).

$$\lambda_{v_t} = \max_{\substack{U \subset S_{v_t} \\ \|U\| \leq \|S_{v_t}\|/2}} \left\langle \frac{\text{out}(U)}{\text{cap}(U, S_{v_t} \setminus U)} \right\rangle. \quad (12)$$

λ defines the maximum $\lambda_{v_t} \forall v_t$.

Weight Ratio δ_{v_t} The weight-ratio describes for a set U the ratio between $w_{l+1}(U)$ and the capacity of edges connecting U with the rest of the cluster.

$$\delta_{v_t} = \max_{\substack{U \subset S_{v_t} \\ \|U\| \leq \|S_{v_t}\|/2}} \left\langle \frac{w_{l+1}(U)}{\text{cap}(U, S_{v_t} \setminus U)} \right\rangle. \quad (13)$$

δ defines the maximum $\delta_{v_t} \forall v_t$.

5.3.3 Step 1 Simulating G on T_G

The node v in G has a corresponding cluster v in the laminar system. Therefore we have a corresponding leaf node v_t in T_G . Thus all nodes in G are simulated by the leaf nodes of the tree. A message sent between u and v in G is sent along the unique shortest path between u_t and v_t in the tree.

Theorem 4. *For any request sequence σ for an online problem on network G that can be processed with congestion C the straightforward simulation on T_G yields congestion $C_t \leq C$*

Proof: Let $e_t = (u_t, v_t)$ denote an edge of T_G that connects a level l node v_t to a level $l-1$ node u_t and that has relative load C_t , when processing on T_G . Then the absolute load of e_t is $C_t \cdot b_t(e_t)$. Now, consider how σ is processed on G . Any message that crosses edge e_t in T_G has either to leave or to enter the cluster S_{v_t} . The total bandwidth of all edges leaving cluster S_{v_t} is $\text{out}(S_{v_t}) = b_t(e_t)$. Thus, one of those edges must have relative load $\frac{C_t \cdot b_t(e_t)}{\text{out}(S_{v_t})} = C_t$. Hence, $C \geq C_t$.

5.3.4 Step 2 Simulating T_G on G

A level l node v_t in T_G is simulated by a random node of the corresponding cluster S_{v_t} with a probability depending on the weight function of the node v . i.e. v_t is mapped to node $v \in S_{v_t}$ with probability $\frac{w_{l+1}(v)}{w_{l+1}(S_{v_t})}$. Therefore a node with higher weight function will have a higher chance of simulating the corresponding tree node, and a leaf node in Tree is always simulated by the corresponding unique node in the G .

An edge (u_t, v_t) in Tree is simulated as follows: Consider a level l node u_t of T_G with d children $v_i, i \in \{1, \dots, d\}$. In an initialization phase we solve a concurrent multicommodity flow problem (CMCF-problem) on the cluster S_{u_t} and afterwards we select the routing paths for the tree edges (u_t, v_i) according to this solution.

Initialization Define a Concurrent Multi-commodity flow (CMCF) for the cluster S_{u_t} as:

$\|S_{u_t}\|^2$ commodities called $f_{u,v}$ for $(u,v) \in S_{u_t}$. The source and sink of $f_{u,v}$ is u and v respectively. The demand of $f_{u,v}$ is given by

$$d_{u,v} := \frac{w_{l+1}(v)}{w_{l+1}(S_{v_i})} \cdot \text{out}(S_{v_i}) \cdot \frac{w_l(u)}{w_l(S_{u_t})}, \quad (14)$$

We solve this CMCF problem on the cluster S_{u_t} while respecting the bandwidth of edges in S_{u_t} . Let q be the minimum fraction of demand met over all commodities. The routing path between u and v , which simulate the two tree vertices u_t, v_t , is chosen in such a way that the expected load on any edge equals the flow of commodity $f_{u,v}$ along that edge.

Theorem 5. *The expectation of the relative load $L(e)$ of an edge $e \in E$ due to the simulation of a tree strategy on G is bounded by $E(L(e)) \leq O((\log(n)h.\max\{\delta, \lambda\}).C_t)$ where C_t is the congestion on T_G .*

Proof: Let q denote the minimum throughput fraction that was achieved when solving the multicommodity flow problems during the initialization phase. We first relate the expected load $E(L(e))$ of an edge e to the value of q and then we will give a lower bound on q in terms of the parameters $\delta(T_G)$ and $\lambda(T_G)$

Lemma 3. *For any edge e of G , $E(L(e)) = O(h.\frac{C_t}{q})$.*

Proof: Let $L_l(e)$ denote the load of an edge e due to the simulation of edges connecting nodes on level l to nodes on level $l-1$ of T_G . We show that $E(L_l(e)) = O(\frac{C_t}{q})$, which yields the lemma.

Fix a level l . Let (u_t, v_t) denote an edge of T_G and assume that u_t is on level $l-1$ and v_t is on level l . The simulation of such a tree edge induces load on $e = (x,y)$ only if both endpoints x and y of e lie in the cluster S_{u_t} . This holds because the routing paths between the nodes simulating u_t and v_t do not leave the cluster S_{u_t} . Hence, let u_t denote a level $l-1$ node such that $x, y \in S_{u_t}$. (If no such node exists $E(L_l(e)) = 0$.) Further, let d denote the degree of u_t and let $v_i, i \in 1, \dots, d$ denote the children of u_t .

We have to analyze the number of messages that traverse e due to the simulation of the tree edges $(u_t, v_i), i \in 1, \dots, d$. Assume for a worst case scenario that every edge (u_t, v_i) has relative load C_t . Then the absolute load of this edge is $C_t \cdot b_t((u_t, v_i)) = C_t \cdot w_l(S_{v_i})$. We say that this tree edge is mapped to a pair $(u,v) \in VXV$ iff u_t is simulated by u and v_i is simulated by v . In this case $C_t \cdot w_l(S_{v_i})$ messages have to be routed between u and v for the simulation of this edge. For simulating a tree node v_i the node v must be contained in the corresponding cluster S_{v_i} .

Therefore, only one edge of $(u_t, v_i), i \in 1, \dots, d$ comes into question for being mapped to a fixed pair (u,v) . Consequently, the expected number of messages that have to be routed between u and v for simulation of level l edges is

$$\frac{w_{l+1}(v)}{w_{l+1}(S_{v_i})} \cdot \frac{w_l(u)}{w_l(S_{u_t})} \cdot C_t \cdot \text{out}(S_{v_i}), \quad (15)$$

because $\frac{w_{l+1}(v)}{w_{l+1}(S_{v_i})}$ is the probability that v simulates v_i and $\frac{w_l(u)}{w_l(S_{u_t})}$ is the probability that u simulates u_t . This number equals $C_t \cdot d_{u,v}$, where $d_{u,v}$ is the demand for commodity $f_{u,v}$ in the definition of the CMCF-problem (see Equation 1).

This means that if at most $q \cdot d_{u,v}$ messages would be routed between any pair (u,v) the expected load on any edge e would be smaller than the capacity $b(e)$, because the routing paths are chosen in such a way that the expected load of e equals the flow that passes e in the solution of the

CMCF-problem. Consequently, if we route $C_t.d_{u,v}$ messages on expectation between any pair (u,v) , then the expected (absolute) load is at most $b(e) \cdot \frac{C_t}{q}$ for these messages on any edge e . Thus, $E(L_l(e)) = O(\frac{C_t}{q})$, holds for any edge e . This yields the lemma.

Now we have to derive a bound on the throughput fraction of the multicommodity flow problems that are solved during the initialization phase.

The proofs of the following theorems are involved and out of scope of this lecture. So we simply state the theorems and obtain the required bounds on the throughput fraction. Interested readers are encouraged to refer to the original paper [4].

Before stating the theorems we define the term **Sparsest cut**

Definition 5. A cut in S_{u_t} is a partition of S_{u_t} into two disjoint sets U and $S_{u_t} \setminus U$. The sparsity of the cut is its capacity divided by the total demand that has to cross the cut, i.e., the sum of the demands of commodities for which source and destination lie in different portions of the cut. A cut with minimum sparsity is called sparsest cut.

Theorem 6. Any CMCF-problem can be satisfied up to $q = \Omega(\frac{\phi}{\log k})$, where ϕ is the value of the sparsest cut and k is the number of commodities

Lemma 4. The value of sparsest cut of the CMCF-problem on cluster S_{u_t} is atleast $\frac{1}{5\delta + 2\lambda}$.

From Theorem 6 and Lemma 4, we get $q = \Omega(\frac{1}{\log(n) \cdot \max\{\delta, \lambda\}})$. Combining this with Lemma 3 we get $E(L(e)) \leq O((\log(n) \cdot h \cdot \max\{\delta, \lambda\}) \cdot C_t)$.

To complete the proof we need to find appropriate bounds on the values of h, δ and λ . The following theorem completes the proof:

Theorem 7. For any graph $G = (V, E)$ with $n = \|V\|$ nodes there exists a decomposition tree T_G that has height $h(T_G) = O(\log n)$, maximum bandwidth-ratio $\lambda(T_G) = O(\log n)$ and maximum weight-ratio $\delta(T_G) = O(\log n)$.

Proof: Out of scope of lecture please refer to original paper [4].

Theorem 7 and Theorem 5 gives us the expected load on any edge as $E(L(e)) \leq O((\log^3(n) \cdot C_t))$.

Theorem 8. Given a graph G and an associated decomposition tree T_G there exists an oblivious online routing algorithm that is (strictly) $O(\log^3(n))$ -competitive with respect to the congestion.

Proof: A crucial part of the whole framework, was there exists an efficient scheme for online routing in trees. This is straightforward since in a tree there is only one path. So the Competitive ratio for an online algorithm in a tree will be 1.

The simulation technique can be adopted as follows. In the online routing problem the internal nodes of the decomposition tree do not store any information. They are only used during the simulation to ensure that appropriate routing paths are selected. Therefore, for each routing request a new random embedding of these nodes in the network G can be used. Let $L_i(e)$ denote a 0-1 random variable describing the load on edge $e \in E$ due to the routing of the i -th request σ_i . (The routing algorithm can easily ensure that each edge is traversed at most once for each request. Hence, we can assume that $L_i(e)$ is a 0-1 random variable.) The variables $L_i(e)$ are independent and hence, the load $L(e)$ of an edge e , is a sum of independent random variables.

By applying a Chernoff bound to this sum we get $L(e) = O(E(L(e)))$, w.h.p. As this holds for any edge e we get the desired result on the congestion. Note that the independence of different routing requests means that the routing algorithm is oblivious as stated in the theorem.

5.4 Current results:

The Oblivious routing scheme presented here gives a non-constructive proof that a decomposition tree with the corresponding parameters satisfying logarithmic constraints exists for all general networks. In a later paper [5] a constructive algorithm was presented with competitive ratio of $O(\log^4 n)$. This result was further improved to $O(\log^2 n \log \log n)$ [6].

6 Reference:

1. C. Kaklamani, D. Krizanc, T. Tsantilas, Tight bounds for Oblivious Routing in the Hypercube, 1990.
2. J. Aspens, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. Online load balancing with applications to machine scheduling and virtual circuit routing. In Proc. of the 25th ACM Symp. on Theory of Computing (STOC), pages 623631, 1993.
3. B. Awerbuch and Y. Azar. Local optimization of global objectives: competitive distributed deadlock resolution and resource allocation. In Proc. of the 35th IEEE Symp. on Foundations of Computer Science (FOCS), pages 240249, 1994.
4. H. Racke. Minimizing congestion in general networks. In FOCS 43, 2002.
5. M. Bienkowski, M. Korzeniowski, and H. Racke. A practical algorithm for constructing oblivious routing schemes. In Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures, June 2003.
6. 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, June 2003.